Reproducibility through shared computing environments: a short introduction to virtualization technologies

QLS 612 - 2021

Peer Herholz (he/him)
Postdoctoral researcher - NeuroDataScience-ORIGAMI lab at MNI, McGill
Member - BIDS, ReproNim, Brainhack, UNIQUE

@peerherholz

A few things before we start ...

- to the folks with hearing impairments:
    - You can turn closed captions via the CC button on the bottom right of the window displaying the video. If something doesn't work, please let me/us know via the communication channels or during the live session.
- to the folks with vision impairments:
    - All graphics (except those taken from publications) should be high in contrast and color-blind friendly. If there are problems, please let me/us know in the chat.
- to everyone:
    - I will try my best to speak loud and clearly. If you have troubles understanding me, please ask during the live session.
    - If you feel uncomfortable doing this/asking questions via the public chat, please send me a direct message.
- to everyone (continued):
    - I sincerely hope you feel comfortable indicating your pronouns during the live session. If so, please add them to your name if you want to.
    - During the discussion/questions, please say your name as I (and others) would like to avoid mispronouncing it.
    - I try my best to present an objective overview. However, there's inherent bias to such presentations based on personal opinions and experience. Thus, I'm looking forward to openly discuss every point/comment with you all.
    - Let's all give our best to create an open, supportive and welcoming atmosphere for everyone.

Objectives

Your role

Schedule

Logistics

Good to know

Objectives

Gain skills

- Learn about open and reproducible methods and how to apply them using conda and Docker (or Singularity)
- Know the differences between virtualization techniques
- Empower you with tools and technologies to do reproducible, scalable and efficient research

Get involved

- Familiarize yourself with the virtualization/container ecosystem for scientific work

Share

- Bring everything you’ve learning to your home institution and/or lab

Objectives

Your role

- ask questions during the live Q&A / exercise session
- think quick and towards reproducibility
- further familiarize yourself with the shell and non-GUI applications
- start thinking about how you could apply/integrate the techniques introduced here into your own research workflow
- have a great time
- give us feedback and help improve the materials

Objectives

Your role

Schedule

Our *optimistic* schedule:

- Introduction and problem statement
- Virtualization using venv & conda
- Virtualization using containers - A new hope
- Virtualization using containers - 101
- Virtualization using containers - The build strikes back
- The return of reproducibility

Objectives

Your role

Schedule

Logistics

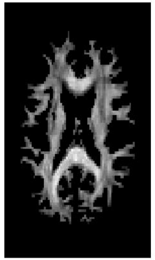
- Your machine:
  - able to get Docker up and running, pull Docker images
  - venv & conda
- You: shell
- The live session may take up 10-15 GB of space
  - can be deleted at end of lecture
  - talk to us if you don't have that much space or think you will run out of it
- Use the communication channels

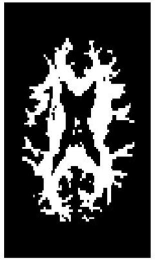
The problem statement

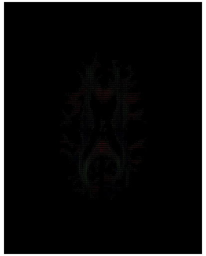
- within the course repository you downloaded, you’ll find a script called “`fancy_DTI_analyzes.py`” (it’s a modified version of great tutorial from the DIPY docs)

- using the shell, navigate to the respective folder and run the script via: “`python fancy_DTI_analyzes.py`”

- within the directory you should get several outputs (three `.png` & one `.trk`)







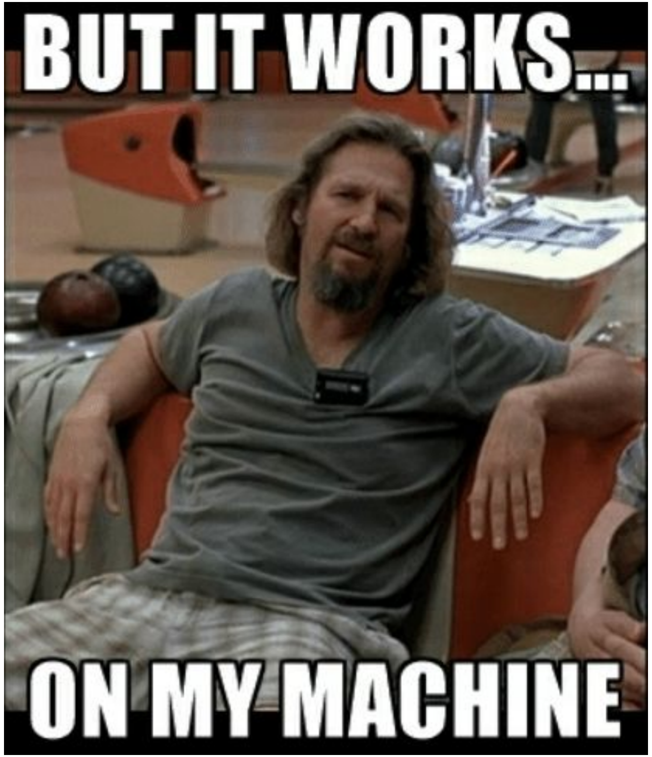
The problem statement

Waaaaaiiiit a hot Montreal minute!

The script doesn't run? The script leads to different results? What went wrong?

Let's gather some errors here ...
(i.e. write them down so we can discuss them during the live session)

The problem statement



https://towardsdatascience.com/how-to-build-slim-docker-images-fast-ecc246d7f4a7

The problem statement

“Modern scientists are doing too much trusting and not enough verifying - to the detriment of the whole of science, and of humanity.”

“The broad adoption of computers in experimental science has both enabled and obscured research” (Marwick, 2015)

- code and analyses is predominately just assumed to run reproducibly and stable, without being properly evaluated and tested
- especially problematic with increasing sample sizes and convoluted/complex models

https://www.economist.com/leaders/2013/10/21/how-science-goes-wrong

The problem statement

Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say "Usually Not"

Andrew C. Chang* and Phillip Li†

September 4, 2015

Abstract

We attempt to replicate 67 papers published in 13 well-regarded economics journals using author-provided replication files that include both data and code. Some journals in our sample require data and code replication files, and other journals do not require such files. Aside from 6 papers that use confidential data, we obtain data and code replication files for 29 of 35 papers (83%) that are required to provide such files as a condition of publication, compared to 11 of 26 papers (42%) that are not required to provide data and code replication files. We successfully replicate the key qualitative result of 22 of 67 papers (33%) without contacting the authors. Excluding the 6 papers that use confidential data and the 2 papers that use software we do not possess, we replicate 29 of 59 papers (49%) with assistance from the authors. Because we are able to replicate less than half of the papers in our sample even with help from the authors, we assert that economics research is usually not replicable. We conclude with recommendations on improving replication of economics research.

Many landmark findings in preclinical oncology research are not reproducible, in part because of inadequate cell lines and animal models.

Raise standards for preclinical cancer research

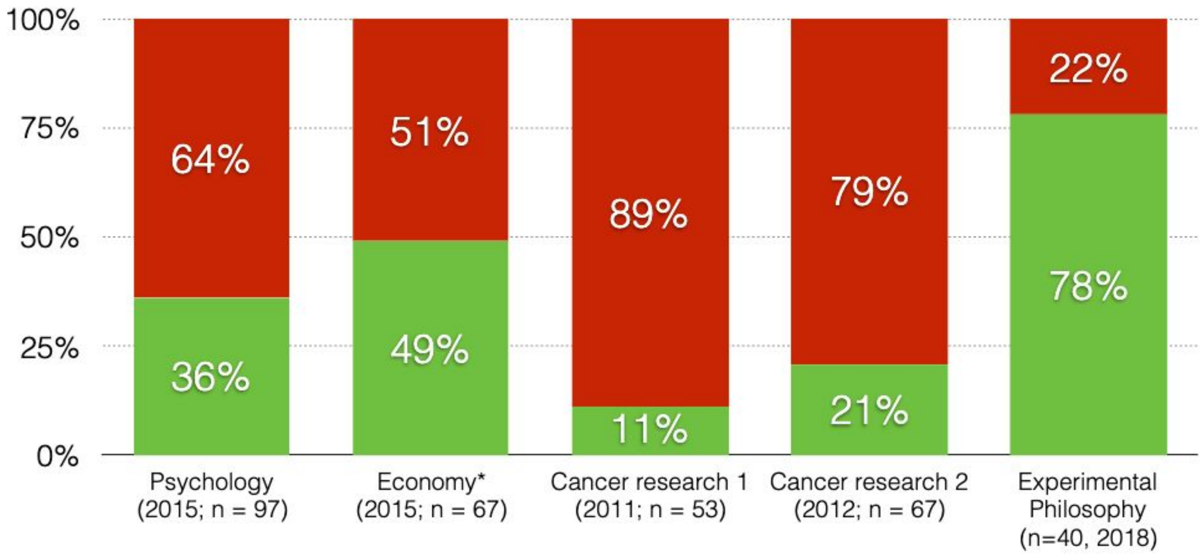
C. Glenn Begley and **Lee M. Ellis** propose how methods, publications and incentives must change if patients are to benefit.

Bayer Healthcare: 67 target-validation projects in oncology, women's health, and cardiovascular medicine.
Only 14 (21%) could be reproduced.

Amgen: **53 'landmark studies', only 6 (11%) could be reproduced.**
"Even knowing the limitations of preclinical research, this was a shocking result."

*adapted from Felix Schönbrodt

The problem statement



* The data on economics is about *reproducibility*; i.e. the attempt to get the same results if you apply the original data analysis on the original data set.

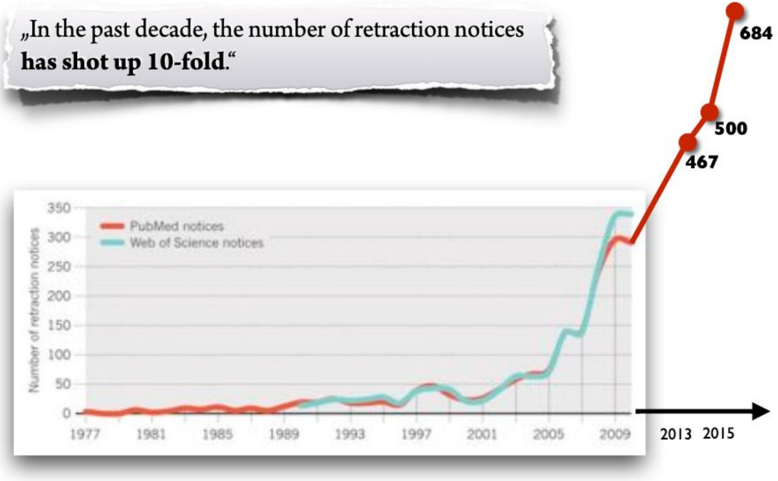
Open Science Collaboration (2015); Chang & Li (2015); Begley, C. G., & Ellis, L. M. (2012). Prinz, F., Schlange, T., & Asadullah, K. (2011); Cova et al. (2018)

*adapted from Felix Schönbrodt

The problem statement

What is happening?

- (un)intentional mistakes
- garden of forking paths
- questionable research practices/p-hacking
- fraud
- publication bias



*adapted from Felix Schönbrodt

The problem statement

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

The problem statement

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

- Sensitivity/ robustness analysis
- Multiverse analysis (Steegen et al., 2016)
- Specification curve (Simonsohn et al., 2015)
- Vibration of effects (Patel et al., 2015)
- Ensemble approach (e.g. climatology)
  → use a set of models with the same input data to produce a range of outcomes

*adapted from Felix Schönbrodt

The problem statement

New Study Calls the Reliability of Brain Scan Research Into Question

Three million analyses point to a problem with fMRI brain activity studies https://www.smithsonianmag.com/smart-news/new-study-calls-reliability-brain-scan-research-question-180959715/

SCIENCE

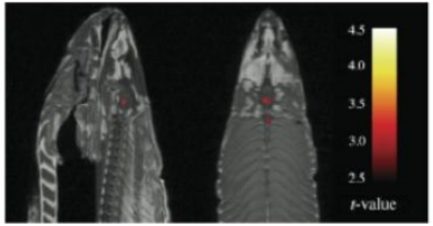
Much of what we know about the brain may be wrong: The problem with fMRI

Aug 30, 2016 / David Biello https://ideas.ted.com/much-of-what-we-know-about-the-brain-may-be-wrong-the-problem-with-fmri/

Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates http://www.pnas.org/content/113/28/7900

Anders Eklund, Thomas E. Nichols and Hans Knutsson

PNAS 2016 July, 113 (28) 7900-7905. https://doi.org/10.1073/pnas.1602413113



The fish that launched a thousand 'skeptics'
http://blogs.discovermagazine.com/neuroskeptic/2014/04/11/neuroimaging-dont-throw-baby-salmon/#.WqAMFhclGRs

A Bug in FMRI Software Could Invalidate 15 Years of Brain Research

This is huge.

BEC CREW 6 JUL 2016

https://www.sciencealert.com/a-bug-in-fmri-software-could-invalidate-decades-of-brain-research-scientists-discover

Debunking Science: fMRI: A Not So Reliable Mind-Reader

Senzeni Mpofu By Senzeni Mpofu
April 11, 2014 15:35

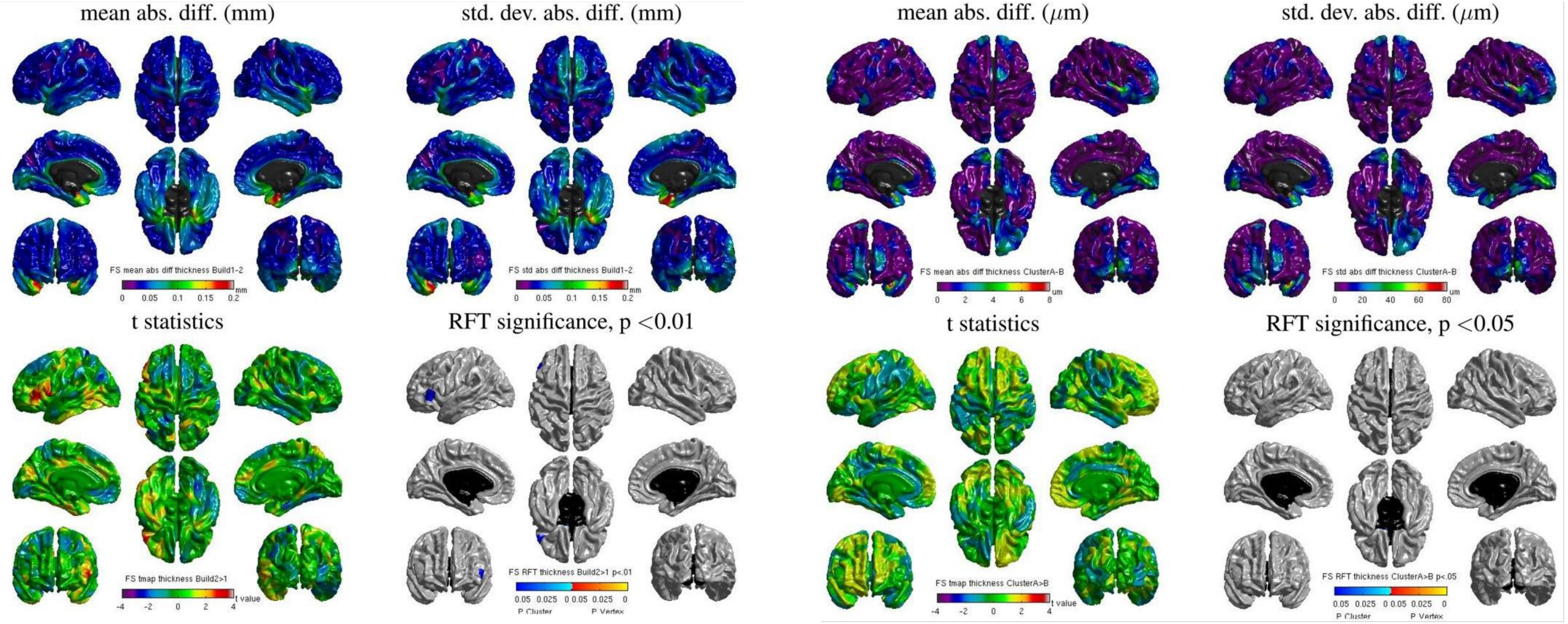
http://www.yalescientific.org/2014/04/debunking-science-fmri-a-not-so-reliable-mind-reader/

What Can fMRI Tell Us About Mental Illness?

By Neuroskeptic | January 14, 2017 10:53 am

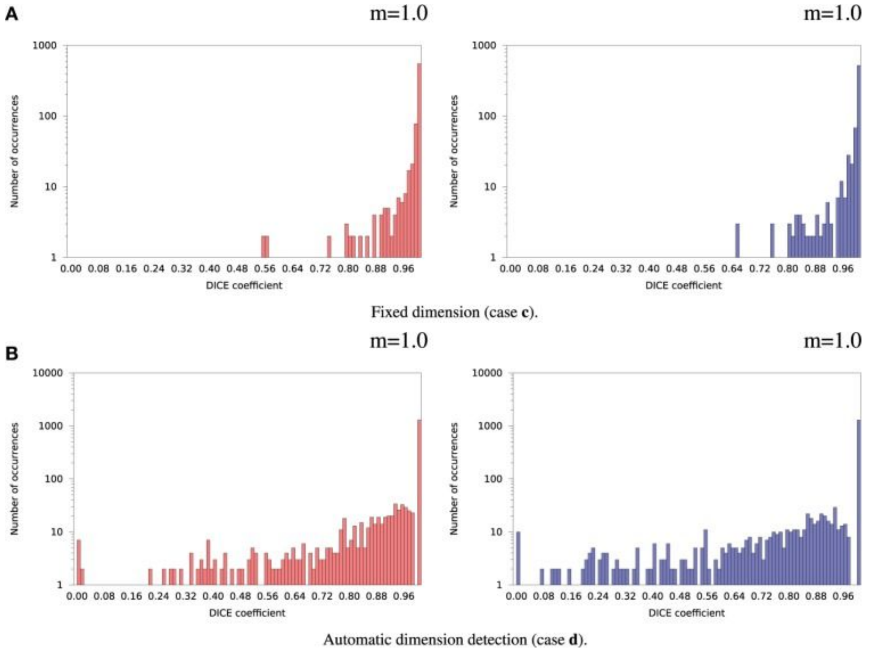
http://blogs.discovermagazine.com/neuroskeptic/2017/01/14/fmri-mental-illness/#.WqAOVhclGRt

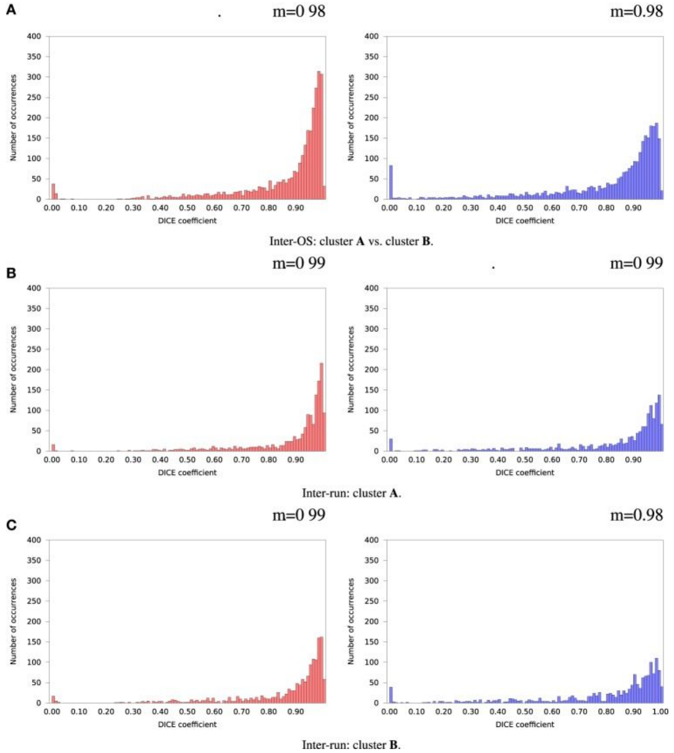
The problem statement



Glatard et al. (2015): Reproducibility of neuroimaging analyses across operating systems

The problem statement





Glatard et al. (2015): Reproducibility of neuroimaging analyses across operating systems

The problem statement

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

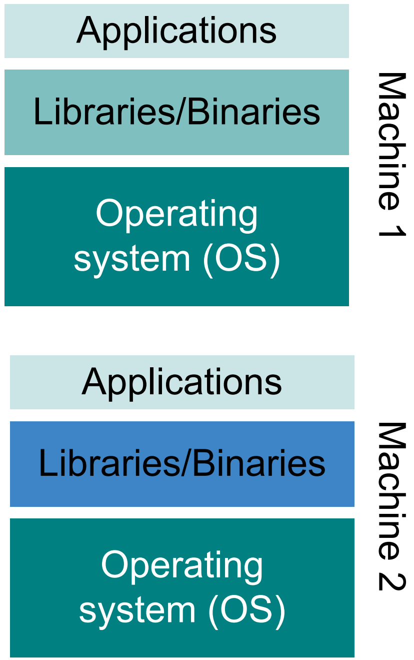
From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

The problem statement

Science reproducibility

- each and every single project in a lab depends on complex **software environments**:
  - operating system
  - Drivers
  - Software dependencies: Python, R, MATLAB + libraries

- we try to avoid:
  - the computer I used was shut down a year ago, I can't rerun the analyzes from my publication… (looking at everyone)
  - the analyzes were run by my student, I have no idea where and how...(looking at the PIs)



The problem statement

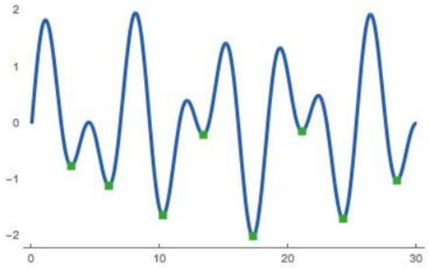
		Data: Same	Data: Different
Analysis	Same	Reproducible	Replicable
Analysis	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

Intrinsic numerical errors & instabilities

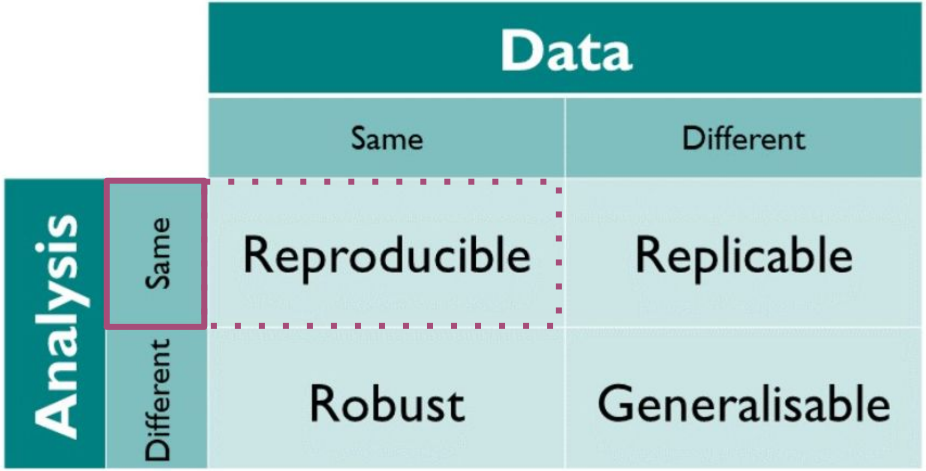
- errors may lead unstable functions towards distinct local minima

```
(~) gkiar $ python3 -c "print(sum([0.001 for _ in range(1000)]))"
1.0000000000000007
```



https://brilliant.org/wiki/extrema/ *adapted from Greg Kiar

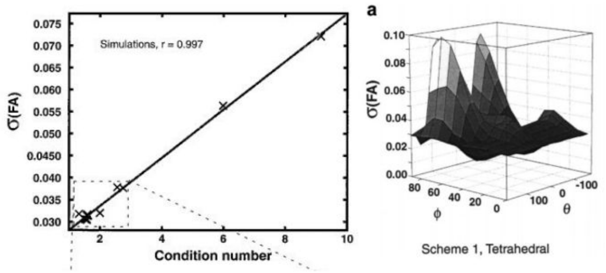
The problem statement



From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

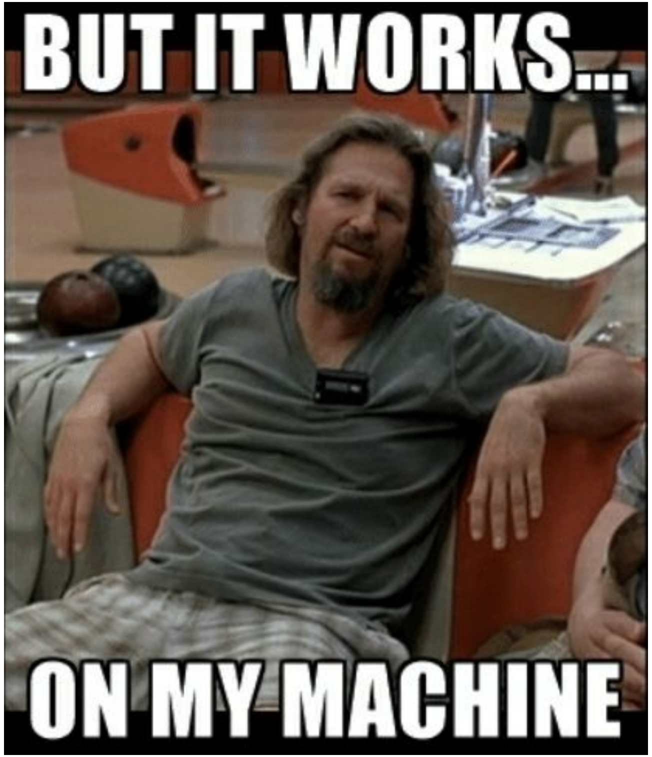
Intrinsic numerical errors & instabilities

- prominent differences across OS
- tensor stability relates to variability



(Skare, 2020) *adapted from Greg Kiar

The problem statement



https://towardsdatascience.com/how-to-build-slim-docker-images-fast-ecc246d7f4a7

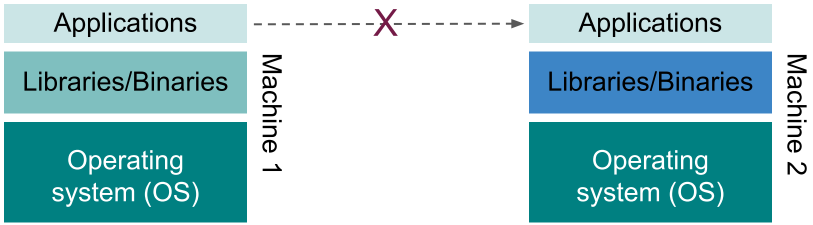
The problem statement

Collaboration with your colleagues and everyone else

- sharing your code or using a repository might not be enough (spoiler: it won't be enough) due to the aforementioned reasons

- we try to avoid:

    - Well, I forgot to mention that you have to use Clang and gcc never worked for me ...
    - I don't see any reason why it shouldn't work on Windows … (I actually have no idea about Windows, but won't say it … (I'm honest here: I have no idea about Windows))
    - etc.



The problem statement

Freedom to experiment

- universal install script from xkcd: *The failures usually don’t hurt anything … And usually all your old programs work ...*

- we try to avoid:
    - I just want to undo the last five hours of my (lab/work) life (virtualization won’t solve comparable problems in other life situations)

The problem statement

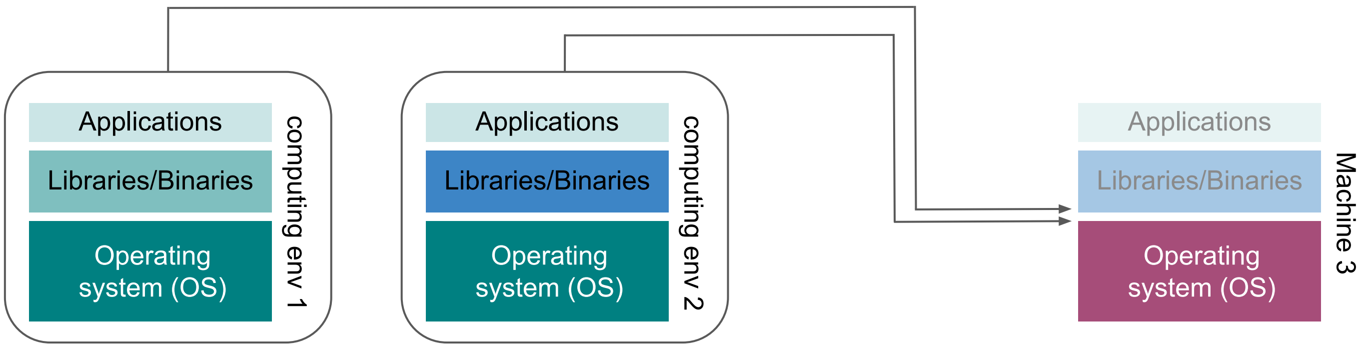
Are we all doomed to live in an unreproducible world, forced to painfully adapt and check every script we find?

Well… maybe, but you could also learn to utilize virtualization techniques...

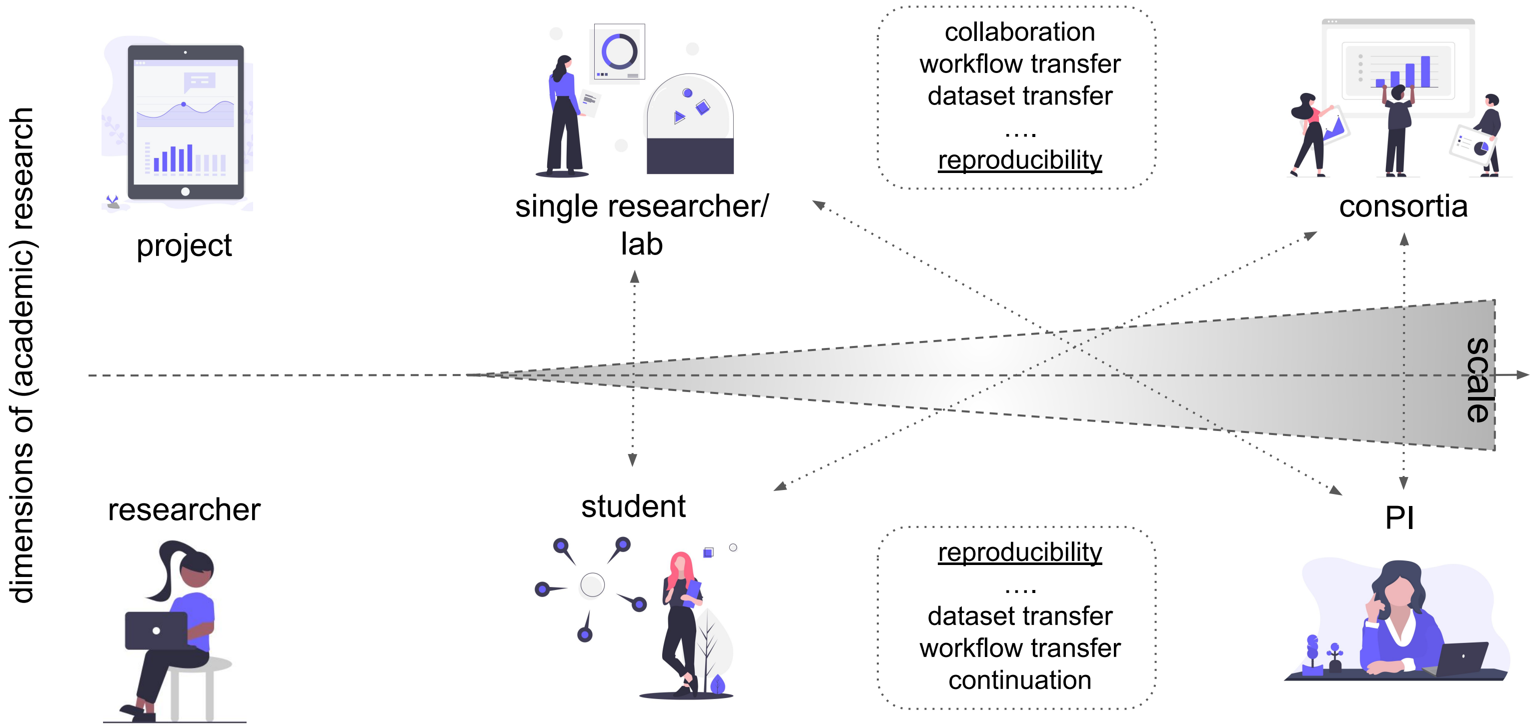
The problem statement - virtualization as solution?

Virtualization technologies aim to

- isolate the computing environment
- provide a mechanism to encapsulate environments in a self-contained unit that can run anywhere
  - reconstructing computing environments
  - sharing computing environments



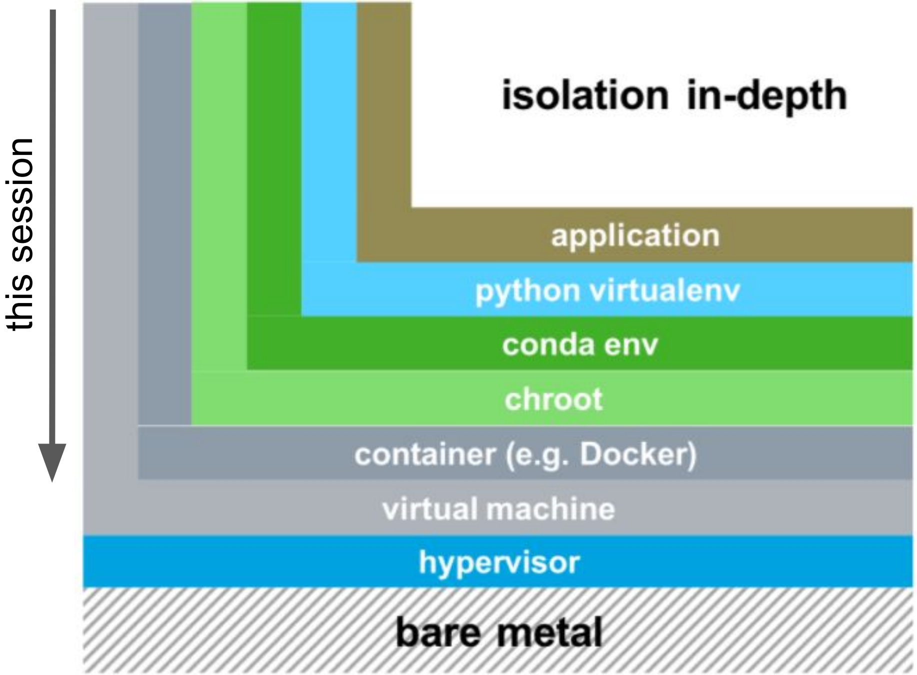
The problem statement - virtualization as solution?



The problem statement - virtualization as solution?

Virtualization technologies have 3 main types:

- python virtualization
  - venv
  - conda
- containers
  - Docker
  - Singularity
- virtual machines
  - Virtualbox
  - VMware



The problem statement - virtualization as solution?

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

		Interaction style	
		GUI	CLI
Shared	SW	Binder	conda
	OS	Binder/VMs	container

Adapted from The Turing Way, Fig.33

GUI - graphical user interface
CLI - command line interface
SW - software
OS - operating system
VMs - virtual machines

Virtualization using venv & conda

Virtual environments in python

- keep the dependencies required by different projects in separate places
- allows you to work with specific version of libraries or Python itself without affecting other Python projects

venv

- an environment manager for Python 3.4 and up, usually preinstalled
- within a terminal type “python -m venv”

conda

- an environment manager and package manager (for python and beyond)
- within a terminal type “which conda” and if it’s not installed do so now

Virtualization using venv & conda

Always create and utilize dedicated directories per environment.

- the mysterious `venv` might hold the key to enable the shared script

- we'll put this old tale to the test, following best practices and create a dedicated directory, here called "`moisture_farm`"

```
mkdir /Users/peerherholz/Desktop/galaxy/tatooine/moisture_farm

cd /Users/peerherholz/Desktop/galaxy/tatooine/moisture_farm
```

- within this directory we're going to create a new python environment using `venv` functionality

- `venv` is a command line program without a GUI and needs to be called within your `shell`

Virtualization using venv & conda

- The respective syntax is "`python -m venv *name*` " where name is the name of your new python environment

- you can use almost every character and name, but providing a meaningful name is never a bad idea, that being said, we'll call our environment "`c3po`":

```
python -m venv c3po
```

- by default no output will be given, so how can we check we happened? Just use `ls` to find out:

```
ls c3po
```

Virtualization using venv & conda

- we have three directories: `bin`, `include` and `lib`, here's what they entail:
    - `bin` : files that interact with the virtual environment
    - `include` : C headers that compile the Python packages
    - `lib` : a copy of the Python version along with a site-packages folder where each dependency is installed
- but how can we work with this newly created python environment? At first, we have to activate it, by utilizing the `activate.sh` script in `bin`:

```
source c3po/bin/activate
```

- we're now in our newly created python environment and can use its resources (what actually happens: we put our new environment at the beginning of our shell's `$PATH` variable)
- the change of environment is indicated through the display of its name left to the command prompt (only visible in the shell)

Virtualization using venv & conda

- using deactivate we (you guessed right) deactivate or "leave" our environment again

```
deactivate
```

- now that we have that, let's try to run our analyzes again, after activating our environment (let’s move the function to our directory first for ease of use)

```
mv path/to/fancy_DTI_analyzes.py /Users/peerherholz/Desktop/galaxy

source c3po/bin/activate

python ../../fancy_DTI_analyzes.py
```

Virtualization using venv & conda

venv is “restricted” to basic skeletons.

source activate
source deactivate

- most likely, you'll receive the error message "`ModuleNotFoundError: No module named 'numpy'`"
- But why is that? The reason is fairly simple ...
- we can also save the output of pip freeze to create our own requirements.txt which you then can share with whoever is interested in running your scripts and analyses using the same python libraries with the identical version:

```
pip freeze > requirements.txt
```

```
cat requirements.txt
```

- this is one form of **virtualization of python environments**

Virtualization using venv & conda

- as before, let's create a directory for our journey, this time it's called " `mos_eisley` ":

```
mkdir /Users/peerherholz/Desktop/galaxy/tatooine/mos_eisley
cd /Users/peerherholz/Desktop/galaxy/tatooine/mos_eisley
```

- while we used `venv` to create our virtual environment and `pip` as a package manager before, we can use `conda` for both as it combines the respective functionalities

- recreating our virtual environment from before is made very easy and straightforward through conda, with the general syntax being: `conda create -n *name* *python_version* *libraries*` , where `*name*` is the name of your virtual environment, `*python_version*` the python version you want to use and `*libraries*` the libraries you want to install

Virtualization using venv & conda

- adapted to our mission, this looks as follows (naming our environment "`r2d2`", installing `python 3.7` and the packages `scipy`, `numpy` and `matplotlib`:

  ```
  conda create -y -n r2d2  python=3.7 scipy numpy matplotlib
  ```

- `conda`, by default, already installs a fair amount of libraries as compared to `venv`

- when trying to check our environment as we did before using ls, we see … nothing:

  ```
  ls
  ```

- the last two points mark important differences between `venv` and `conda` which we will check further

Virtualization using venv & conda

conda environments are created within the conda installation path.

- let's activate our newly created `conda environment`, the steps and syntax are very similar to what we've done before, yet slightly different due to `conda`:

```
conda activate r2d2
```

- instead of using "`source`" and pointing to the wanted "`activate.sh`" script, we use `conda` specific commands as `conda` is its own program, with its `environments` not being created at a specified directory as through `venv`, but within the `conda` installation at hand

- we can check this via "`which python`"

- or even better using a `conda` command that additionally lists all available `conda environments`:

```
conda info --envs
```

Virtualization using venv & conda

conda is powerful but still requires caution.
conda activate
conda deactivate

- while this amazing and brings us further to goal with lightspeed, we actually didn't test if our fancy analyses works:

```
python ../../fancy_DTI_analyzes.py
```

https://media.vanityfair.com/photos/5852ca58972218dd20575570/master/w_2560%2Cc_limit/rogue-one-darth-vader-02.jpg

find your lack of controlling
and evaluating installation
processes disturbing.

Virtualization using venv & conda

Virtual environments in python - key points to remember

- `venv` and `pip enable` the creation and sharing of `python environments`, including specific `python libraries` and `versions` thereof.

- To also share the specific `python version`, `builds` and `channels`, you need to use `conda`. The same holds true for certain non-pure-`python` dependencies.

- `Conda` is already very powerful as it combines both `environment` (comparable to `venv`) and `package` (comparable to `pip`) `manager`.

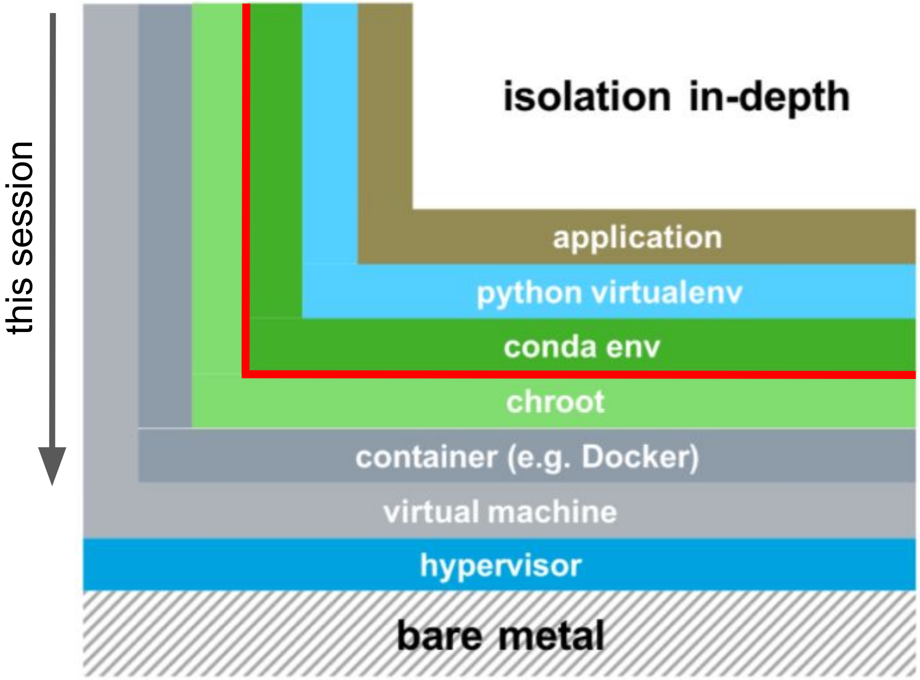
- Not all packages are installable via both `Conda` and `pip`. On the contrary: often packages are only available via one of them. Overall, more packages are available through `pip`.

- `python virtualization` does not account for often required `libraries` and `binaries` on other levels (beyond `python`).

The problem statement - virtualization as solution?

Virtualization technologies have 3 main types:

- python virtualization
  - venv
  - conda
- containers
  - Docker
  - Singularity
- virtual machines
  - Virtualbox
  - VMware

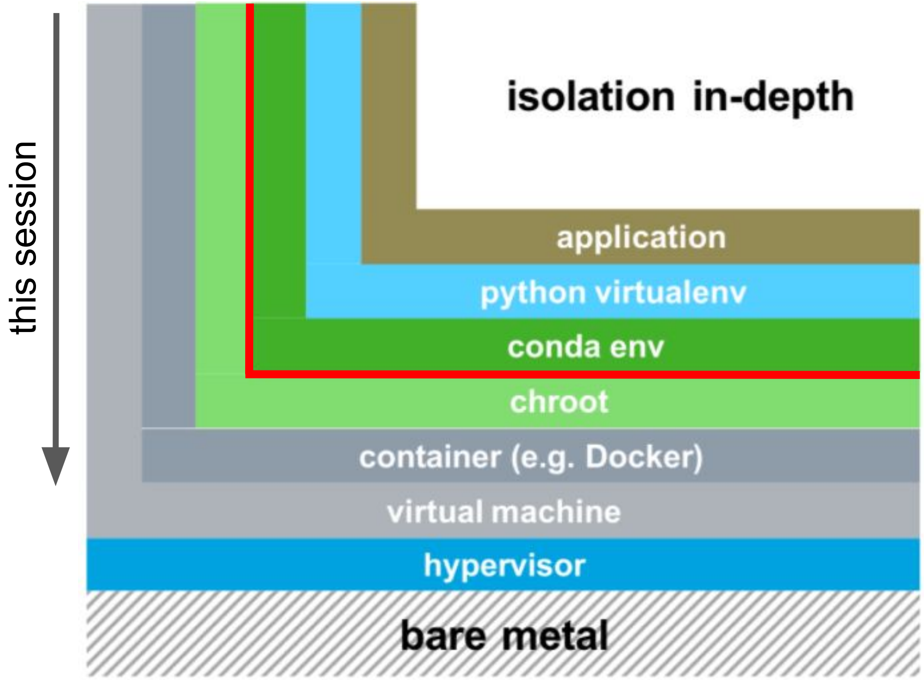


The problem statement - virtualization as solution?

Virtualization technologies have 3 main types:

- python virtualization
    - venv
    - conda

`python virtualization` does not account for often required `libraries` and `binaries` on other levels (beyond `python`). For this the next level of virtualization is needed.



The problem statement

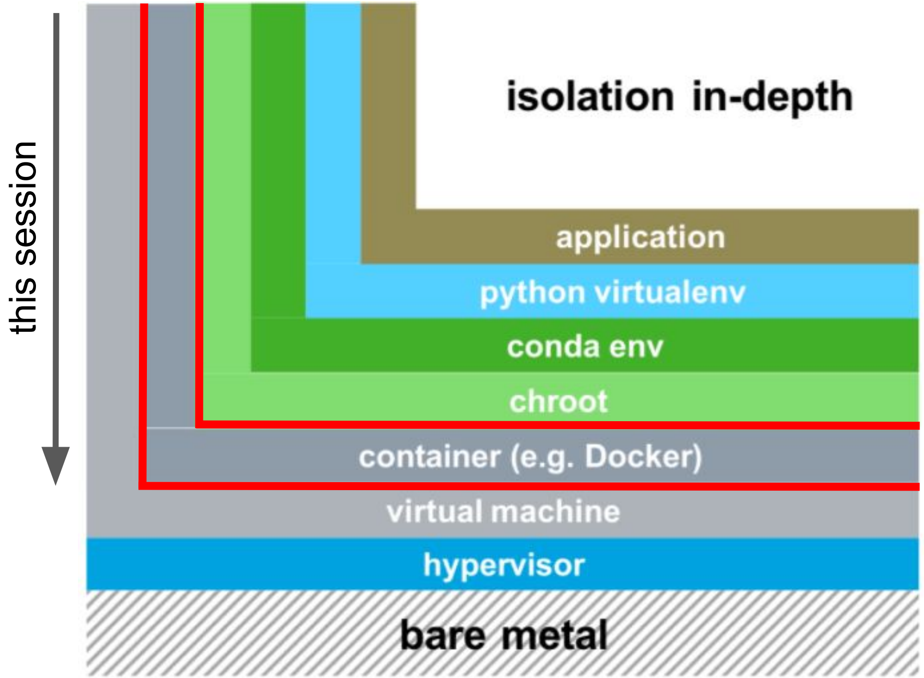
https://hotteahotbooks.files.wordpress.com/2012/09/darth_vader_i_am_your_father.jpeg

https://miro.medium.com/max/1400/1*A-ID0FHKmChS2vGelhUs2g.png

Virtualization using containers - A new hope

Virtualization technologies have 3 main types:

- python virtualization
  - venv
  - conda
- containers
  - Docker
  - Singularity
- virtual machines
  - Virtualbox
  - VMware



The problem statement - virtualization as solution?

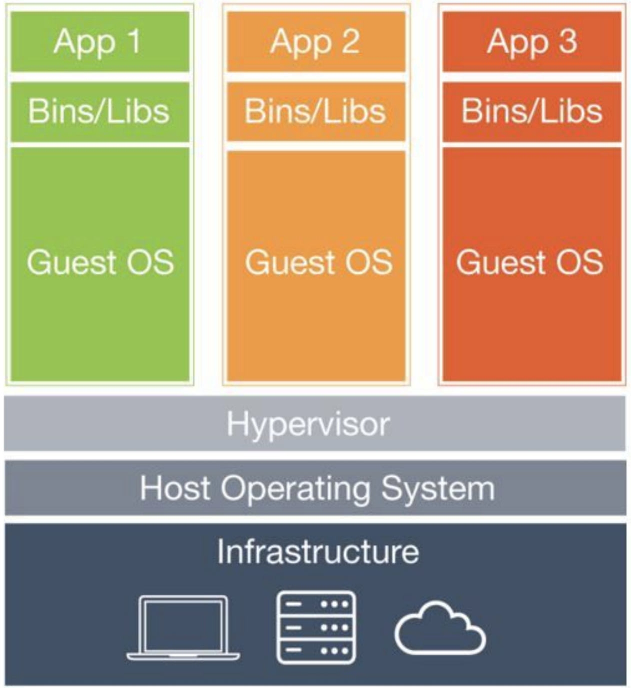
		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

		Interaction style	
		GUI	CLI
Shared	SW	Binder	conda
	OS	Binder/VMs	container

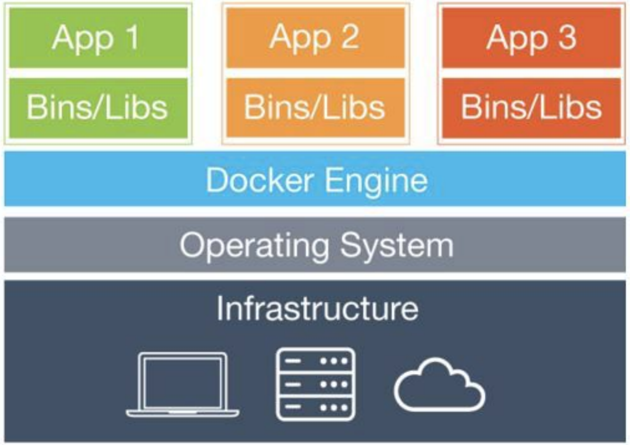
Adapted from The Turing Way, Fig.33

Virtualization using containers - A new hope



virtual machines

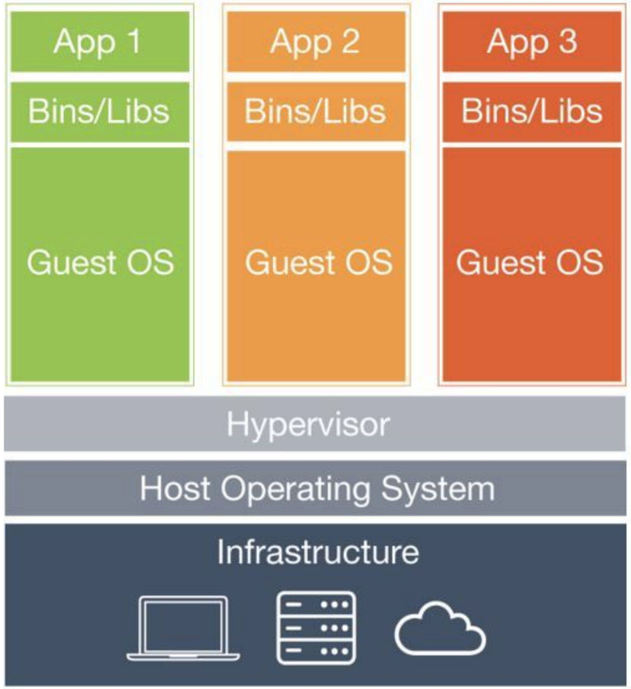
VS



containers

https://crmtrilogix.com/images/uploaded/virtualization-vs-containers.png

Virtualization using containers - A new hope



https://crmtrilogix.com/images/uploaded/virtualization-vs-containers.png

virtual machines

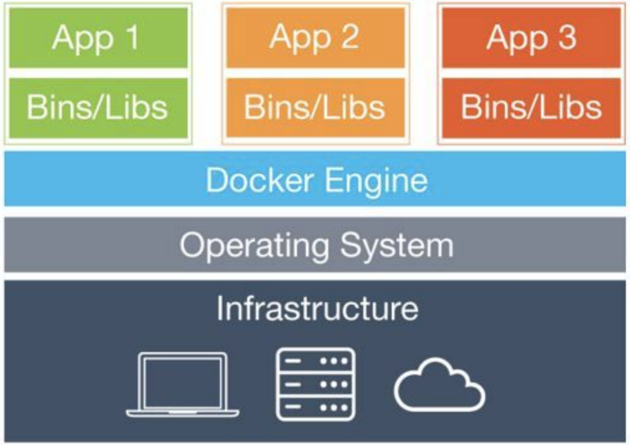
- emulate whole computer system (software+hardware)
- run on top of a physical machine using a *hypervisor*
- *hypervisor* shares and manages hardware of the host and executes the guest operating system
- guest machines are completely isolated and have dedicated resources

Virtualization using containers - A new hope

- share the host system's kernel with other containers

  → **kernel level virtualization**

- each container gets its own isolated user space
- only bins and libs are created from scratch
- containers are very lightweight and fast to start up



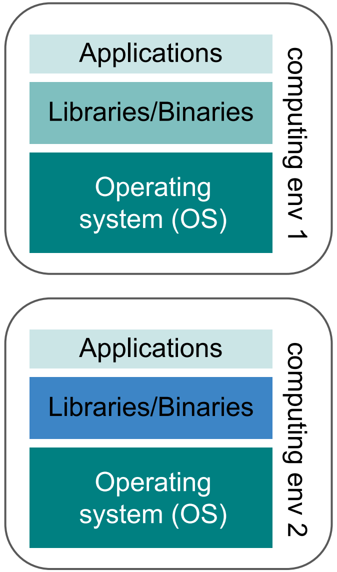
containers

https://crmtrilogix.com/images/uploaded/virtualization-vs-containers.png

Virtualization using containers - A new hope

- a container is a closed environment and will use the exact same software versions, drivers etc. independent from the system it runs on

- leading software container platform
- an open-source project
- it runs on Linux, Mac OS X and Windows



Virtualization using containers - A new hope

docker vs Singularity

- docker can escalate privileges, so you can be effectively treated as a root on the host system
- this is usually not supported by administrators from HPC centers

- a container solution created for scientific and application driven workloads
- supports existing and traditional HPC resources
- a user inside a Singularity container is the same user as outside the container
- can run existing Docker containers

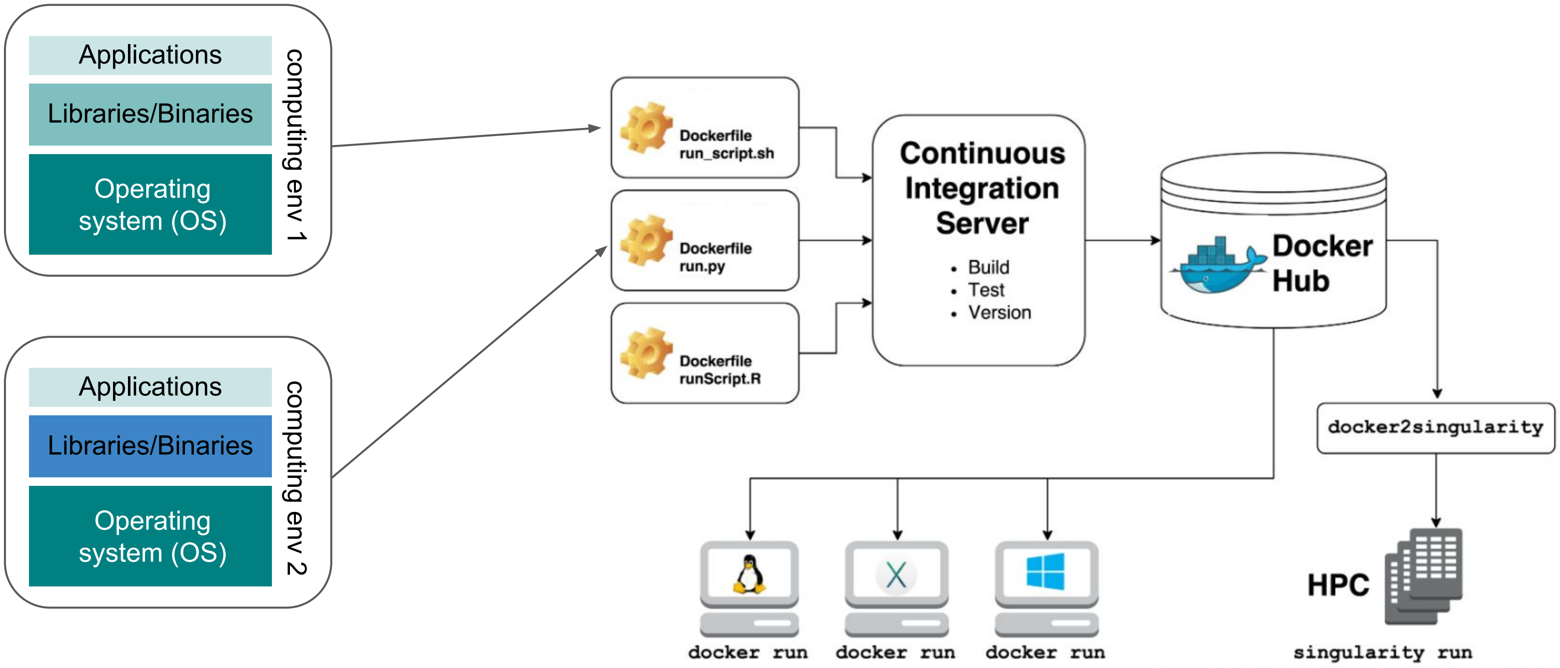
Virtualization using containers - A new hope

https://i.redd.it/si1z0556y3m11.png

docker

Singularity

Virtualization using containers - A new hope



From doi:10.1371/journal.pcbi.1005209.g001

Virtualization using containers - A new hope

Where's Docker?

- `docker` is command line based, thus does not have a GUI
- on unix based OS (e.g., Ubuntu, Mac OSX) open a terminal and type `docker`
- on windows open docker toolbox, engine or WSL (depending on your specific OS) and type `docker`
- what you see is the so called `docker man page` providing helpful information on how to use docker

Virtualization using containers - A new hope

- as you saw, there's a lot one can do with Docker

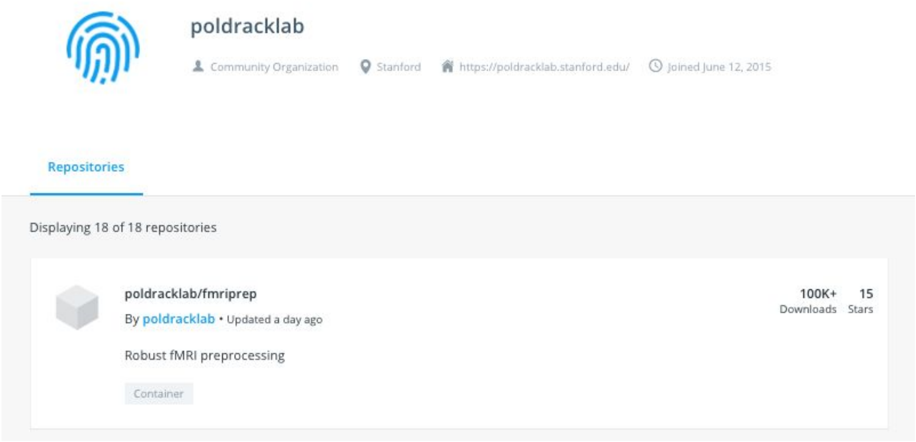
- before we go into the depths of the Docker galaxy, let's make sure it works:

```
docker run hello-world
```

- What is happening? The above command ...
  - tells `docker` to `run` or `execute` the container `hello-world`
  - if the wanted `container` is not already on your machine, `docker` automatically searches for and downloads it
  - `docker` run then `runs` or `executes` the given `container` doing whatever the `container` is supposed to do

Virtualization using containers - A new hope

- Docker hub - a special place in the galaxy
  - in most cases `docker images` are stored at a certain place in the galaxy from which they are also downloaded in lightspeed (depending on your network)
  - this mysterious place is called `docker hub` and contains an amazing and huge online repository where folks can upload and store as many `docker images` as they want for free, the only requirement: a `docker id`



Virtualization using containers - A new hope

You can and should combine virtualization and version control.

- Docker hub - a special place in the galaxy
  - once build and tagged, docker images can be pushed to docker hub via command line:

    ```
    docker push death-star
    ```

  - or automatically build from a `GitHub repository` after pushing `commits` to a respective repo (unfortunately only possible via paid plans)

+

Virtualization using containers - A new hope

- now let's further explore `docker commands` within a typical workflow

- at first, we want to download a certain `docker container` to work with, for the sake of simplicity and time we're going to use the classic `ubuntu container`

- instead of automatically downloading the container via `docker run`, we use the respective `docker command` "`docker pull *image_name*` ", hence:

```
docker pull ubuntu
```

- as you can see, we downloaded all `layers` that are needed to `build` the classic `ubuntu docker container`, with the message "`Status: Downloaded newer image for ubuntu:latest`" showing you that everything worked

Virtualization using containers - A new hope

- we can check existing Docker images using:

  `docker images`

- this provides us with the `repository`, `tag`, `image id`, `build` and `size` of all `docker images` available on our machine

- **super important**: by default, `docker pull` always searches and downloads the `image` that is `tagged` with `latest`, hence if you want to have a certain version (e.g., an `older release` or `developer`) it is necessary to indicate the respective tag:

  `docker pull ubuntu:version`

Virtualization using containers

Virtualization using Docker/Singularity - key points to remember

- The host machine's kernel is shared between containers.
  - Potentially problematic and can re-introduce external dependencies.
- Containers are way more lightweight and faster than virtual machines.
- Singularity is the way to the HPC side concerning containers.
- Docker and Singularity don't have a GUI.
- Docker/Singularity images hosted are Docker/Singularity Hub.
- By default `Docker/Singularity pull` always searches for the tag `latest`.
  - Both good and bad, specific tags and thus versions are necessary.
- `Docker/Singularity images` are composed of layers, which are shared between images and contain entire OS.

Virtualization using containers - 101

Images can have dedicated functionality or can be utilized more broadly.

- now we want to work with our newly downloaded `docker image`, so we decide to `run` or `execute` it via:

  `docker run ubuntu`

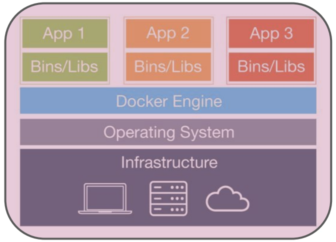
- `docker` used `run`, nothing happened (Pokémon pun): all the fuzz for that?

- each `docker image` is `build` for a specific reason and purpose, hence what happens when you `run` a given `docker container` depends (more or less) exclusively on its setup and definition

- the `docker image` we're currently using has no defined mission, that is functionality that automatically starts when running

- it is therefore super important to consult the `readme` or `docs` of a given `docker image` before using it

Virtualization using containers - 101

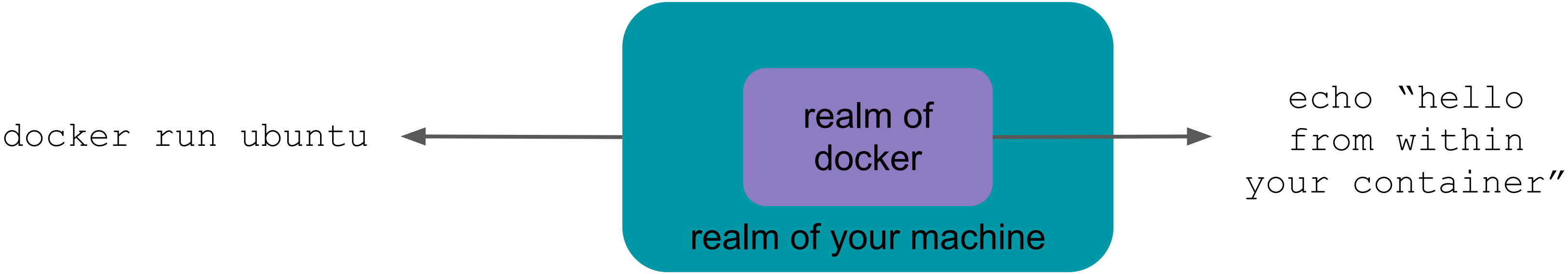


- *automated functionality* poses as one major approach to use `docker images` (e.g. within pipelines and workflows), but more in that later

- it's also possible to define tasks during inition:

```
docker run ubuntu echo "hello from within your container"
```

- you might not be aware of it, but this message doesn't come from your machine, but another realm within your machine, the docker realm ...



Virtualization using containers - 101

- however, if the force allows, one can also enter this realm, utilizing a given `docker container` within a interactive fashion

- which can be achieved through by including the `-it` flag within the docker run command:

  ```
  docker run -it ubuntu
  ```

- now inside the ubuntu `docker container` (realm), we can utilize the functionality from the present `ubuntu OS`, yes you're basically using a entirely different and new machine, just check the output of `ls`:

  ```
  ls
  ```

- we can leave this realm by typing `exit`

Virtualization using containers - 101

Make sure to check resource allocation and usage of images you work with.

- depending on a given `image`'s architecture and definition, it should remove itself from running instances when exiting, however to be sure it's worth to ensure that and check running instances when you notice e.g. a drop in performance

- this can easily be done via:

```
docker ps
```

- the `docker` force is powerful enough to even allow time travel by append `-a`:

```
docker ps -a
```

Virtualization using containers - 101

- now let’s go back to our main goal: making our analysis run everywhere in a reproducible manner

- we descend into the docker force realm and create a new mission base (fine, you can also refer to it as project folder…) named “ `defeat_unreproducibility_empire` “

```
docker run -it --rm ubuntu
```

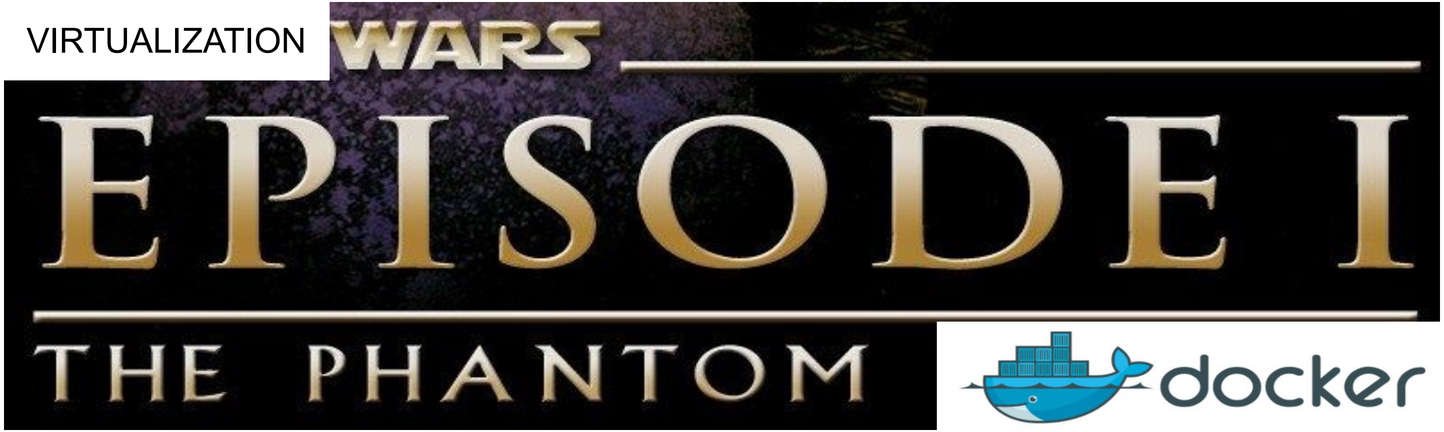
```
mkdir defeat_unreproducibility_empire
```

- let’s imagine 6 years and $76.5 million later, we did it, it runs within our container

- having achieved galactic reproducibility peace, we exit the realm and 16 years as well as a sell to Disney later, we decide to return ...

Virtualization using containers - 101

VIRTUALIZATION



docker

https://4.bp.blogspot.com/-of95ANrPcQk/VXAtYg5y6_I/AAAAAAAAATc/jZQYx90xwpo/s1600/star-wars-episode-i--the-phantom-menace.13951.jpg

- our project folder and everything in it disappeared from the docker realm ...

Virtualization using containers - 101

- when working within a `docker container`, **creating**, **modifying** and **deleting** files, **changes are neither permanent nor saved**, as this is against the encapsulation and reproducibility idea

- furthermore, we cannot interact with data stored on our `host machine` (realm) or somewhere outside the `docker container` (realm)

- in order to address both problems, we need to create a force bridge between realms or in other words: mount paths between `host machine` (realm) and the `docker container` (realm)

- mounting describes a mapping from paths *outside* the `docker container` to paths *inside* the `docker container`

Virtualization using containers - 101

- it is achieved through the `-v` flag within the `docker run` command and utilized as follows: `-v path/outside/container:/path/inside/container`

- for example, let's create a directory called “ `hoth` ” within our “ `galaxy` “ folder and make it available inside the docker container as “ `rebel_base` “ :

```
mkdir /Users/peerherholz/Desktop/galaxy/hoth

docker run -it --rm
          -v /Users/peerherholz/Desktop/galaxy/hoth:/rebel_base ubuntu
```

- if we now create a new file within the directory “ `rebel_base` “, it magically appears on our host machine within the “ `hoth` “ directory through the force bridge (path mount):

```
touch /rebel_base/death_star_plans.png
```

Virtualization using containers - 101

- you can also restrict the rights of mounted paths to e.g. read only in case any modification should be prevented (for example within automated functionality or if the empire wants to destroy your plan)):

```
docker run -it --rm
           -v /Users/peerherholz/Desktop/galaxy/hoth:/rebel_base:ro
           ubuntu

rm /rebel_base/death_star_plans.png
```

- most of the time, it's a good idea to indicate absolute paths on the host system

Virtualization using containers - 101

Make sure to check expected paths and mounts of images you work with.

- in our example the directory `/rebel_base` didn't exist before mounting it, hence it was automatically created, however this is also depends on the `docker image` and it's setup/definition at hand as e.g. within automated functionality a certain directory can be expected

- this can also lead to errors if e.g. an existing directory is overwritten through a directory specified during the `mount`, e.g. if a directory named `/rebel_base` would have already existed within our example, our specified `mount` would have automatically overwritten it, without telling us (the `Docker` force is mighty, but also mysterious)

- as usual: check the `readme` and/or `docs` of a given `docker image`

Virtualization using containers - 101

- you can mount as many directories and files as you want, indicating each with a `-v` flag
- for example you could map an `input` and an `output` directory wrt your analyses (let's say preprocessing/analyzing data):

```
docker run -it --rm
          -v /Users/peerherholz/Desktop/galaxy/hoth:/rebel_base:ro
          -v /Users/peerherholz/Desktop/galaxy/dagobah:/x-wing
          ubuntu
```

- again, check the `readme` and/or `docs` of the `docker container` at hand if some paths and/or files
  are expected and if the paths are generated automatically

Virtualization using containers

Virtualization using Docker/Singularity - key points to remember

- Docker/Singularity images usually have **specific purposes**, examples for this are **automated functionality** and **interactive usage**.

- `Running instances` need to be checked and `images` can **never be changed**.
  - Potential problem as changes/results from a running container can not be preserved without `mounting paths` or `recreating/updating` the image.
  - Good and bad: everything needs to be thought through before running certain things.

- The `Docker/Singularity realm` and your `host machine realm` are two seperate places. They must be connected through a `mount` if you need to provide a flow of data.

Virtualization using containers - the build strikes back

- so far, we (hopefully) learned at the Docker academy how `docker` works, how *images* can be downloaded, used and managed

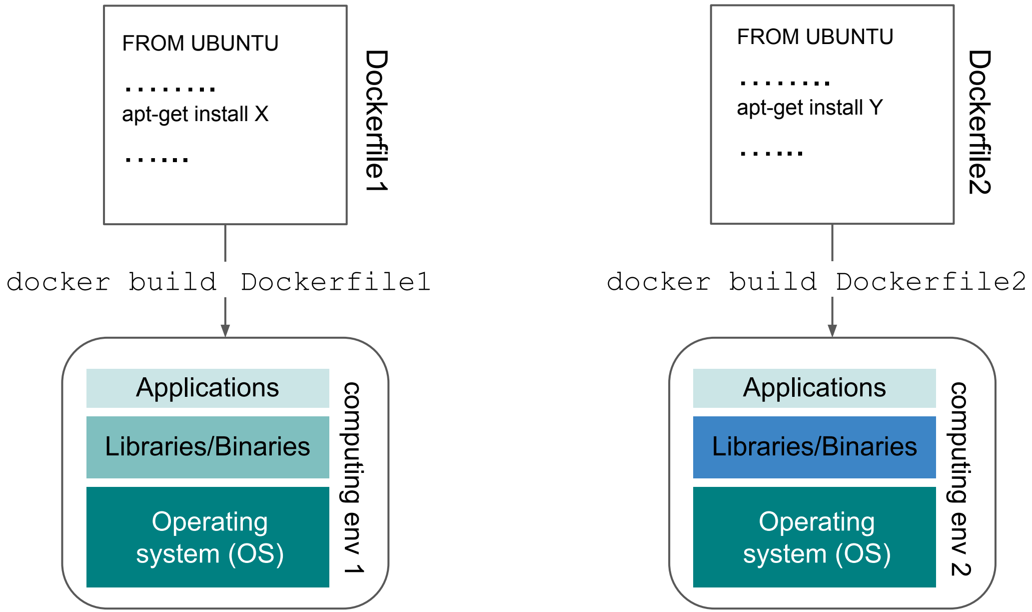
- but how are *docker images* actually created for specific projects, pipelines, etc.?
- how can different programs be installed?
- how are locally created *docker images* shared?

Virtualization using containers - the build strikes back

- when it comes to creating `docker images`, two essential parts are relevant:

- a ***Dockerfile***

- the ***`docker build`*** command



Virtualization using containers - the build strikes back

- at the beginning there was a ***Dockerfile***…
- a ***Dockerfile*** is a `text file` that contains a mixture of `docker build` and `bash` commands
- being the focus of this section, let's create one in our `hoth` directory

```
cd /Users/peerherholz/Desktop/galaxy/hoth

touch Dockerfile
```

- next, we're opening it within a text editor for an initial inspection

Dockerfile1

```
FROM UBUNTU
……..
apt-get install X
……
```

Dockerfile2

```
FROM UBUNTU
……..
apt-get install Y
……
```

Virtualization using containers - the build strikes back

- it is obviously empty, we need to populate it according to the environment we want to virtualize, the respective information is a mixture of `Docker/Singularity specific` and general `linux commands`

- we have to start with the `base`, which corresponds to the underlying OS we want (or need to use)

- given that we already worked on it, we are going to use `ubuntu`:

```
FROM ubuntu
```

- next, we specify that the installation of `packages`, `programs`, etc. is done `non-interactively`, i.e. we're not asked to approve each piece of software:

```
ARG DEBIAN_FRONTEND="noninteractive"
```

Dockerfile1

```
FROM UBUNTU
........
apt-get install X
......
```

Dockerfile2

```
FROM UBUNTU
........
apt-get install Y
......
```

Virtualization using containers - the build strikes back

- additionally, we set the `encoding` and `startup.sh` of our OS:

```
ENV LANG="en_US.UTF-8" \
LC_ALL="en_US.UTF-8" \
ND_ENTRYPOINT="/docker/startup.sh"
```

- now, we can already install `packages/libraries/etc.` in most "linux OS" this is done through "`apt-get`" and we go with some essentials:

```
RUN export ND_ENTRYPOINT="/docker/startup.sh" \
&& apt-get update -qq \
&& apt-get install -y -q --no-install-recommends \
                apt-utils bzip2 \
                ca-certificates curl \
                git \
                locales nano unzip \
```

Virtualization using containers - the build strikes back

- our complete Dockerfile looks like the following:

```
FROM ubuntu

ARG DEBIAN_FRONTEND="noninteractive"

ENV LANG="en_US.UTF-8" \
LC_ALL="en_US.UTF-8" \
ND_ENTRYPOINT="/docker/startup.sh"

RUN export ND_ENTRYPOINT="/docker/startup.sh" \
    && apt-get update -qq \
    && apt-get install -y -q
--no-install-recommends \
                    apt-utils bzip2 \
                    ca-certificates curl \
                    git \
                    locales nano unzip \
```

FROM UBUNTU
....
apt-get install X
......

Dockerfile1

FROM UBUNTU
....
apt-get install Y
......

Dockerfile2

Virtualization using containers - the build strikes back

- with that we already have enough information to build our first `Docker container`

- in order to do so, we need to utilize the docker build command as follows “ `docker build -t *image_name* *Dockerfile*` ”, where `*image_name*` provides a `name` for our to be `build image` via the `-t` flag and `*Dockerfile*` is the `path` to wherever our `Dockerfile` can be found in the galaxy:

```
docker build -t millennium_falcon .
```

- this should be comparably fast and with the message “`Successfully tagged millennium_falcon:latest` ” your first own `docker image` was `build`, which you can check via:

```
docker images
```

Virtualization using containers

Virtualization using Docker/Singularity - key points to remember

- A given image can never be changed.
- It is necessary to be build new ones in case functionality is missing or needs to be adapted.
- Two things are essential for building images: a build file and the build command.
- Images are composed of different layers and the build process mirrors this/follows respective principles via adding components and functionality.
- Build files mimic the setup of linux systems and contain a mixture of Docker/Singularity and linux commands.
- It’s possible to share layers between images which can speed up the build process and minimize storage.

Virtualization using containers - the build strikes back

- while this already seems like an overkill and very complex, we just created a very basic `image`

- the fitting compilation of an entire `Dockerfile` for a complex script, pipeline, analysis, appears very convoluted and prone to errors, as well as takes a lot of time and searching the galaxy (especially in the beginning)

- you might wonder: Isn't there a more sufficient, faster and easier way of creating virtualized environments?

Virtualization using containers - the build strikes back

- well, say no more and meet Neurodocker, a `docker image` that targets the creation of `docker images`. Yes, it's Dockerception!

```
FROM ubuntu

ARG DEBIAN_FRONTEND="noninteractive"

ENV LANG="en_US.UTF-8" \
LC_ALL="en_US.UTF-8" \
ND_ENTRYPOINT="/docker/startup.sh"

RUN export ND_ENTRYPOINT="/docker/startup.sh" \
     && apt-get update -qq \
     && apt-get install -y -q
--no-install-recommends \
                      apt-utils bzip2 \
                      ca-certificates curl \
                      git \
                      locales nano unzip \
```

```
docker run \

kaczmarj/neurodocker:0.4.3 \

generate docker --base=ubuntu \

--pkg-manager=apt \

--install git nano unzip \
```

Virtualization using containers - the build strikes back

- when building images, one can also combine virtualized environments with pure python virtualization, i.e. create a python environment within a virtualized computing environment, for example adding our conda environment from before can be achieved via:

```
docker run kaczmarj/neurodocker:0.4.3 \
                  generate docker --base=ubuntu \
                  --pkg-manager=apt --install apt-utils bzip2 \
                 ca-certificates curl git locales nano unzip \
                  --miniconda conda_install="python=3.7 scipy" \
                  create_env="r2d2" activate=true \
                  > Dockerfile
```

You can use neurodocker to make the generation of Dockerfiles easier and more reproducible.

Virtualization using containers - the build strikes back

- let's rebuild our *Docker image*:

```
docker build -t millennium_falcon .
```

- looks good, but we sense something dark is going on, hidden in the shadows …

```
docker images -a
```

Virtualization using containers - the build strikes back

- previous instances of the *images* are untagged and are not overwritten
- *note*: updating existing *docker image* does not overwrite or delete its previous instances, hence make sure to delete those as soon as you ensured the new *image* is working as intended to prevent unnecessary storage usage
- our *docker image* increased its size by an order of magnitude
- *note*: always keep track of the packages, programs, etc. you install (preferably via git) and keep track of tremendously large software in order to keep your *docker image* as small as possible

Virtualization using containers - the build strikes back

- after all this we're finally going back to our analyzes, as we actually did not test if our container works for its intended purpose

- we can combine the things we just learned, by running our image in an `interactive` manner and `map` our analysis script to make it available inside the container

```
docker run -it -v /Users/peerherholz/Desktop/:/rebel_base \

            millennium_falcon

python /rebel_base/fancy_DTI_analyzes.py
```

Virtualization using containers - the build strikes back

- while we could use the `docker save` and `load` commands to share our `image` locally via e.g. USB, we are going to use this thing called the internet to share it on `docker hub`

- to do so, we need essentially two lines of code, one to `tag` our `image` and one to actually `push` it:

  ```
  docker tag *image_name/id* *docker_id*/*image_name*:*tag*
  ```

  where `image_name/id` is either the `name` or the `id` of the `image` you want to `tag`, `docker_id` your `docker id`, `image_name` the name you want to provide for the `image` and `tag` the respective `tag`

- don't have a docker id? Head over to hub.docker.com to create one

Tags are important to ensure versioning of images, thus also the reproducibility of their respective function.

Virtualization using containers - the build strikes back

- docker id all set? Great! Let’s use the force:

```
docker tag millennium_falcon peerherholz/millennium_falcon:kessel-run
```

- after verifying that the image is and named/tag as intended:

```
docker images
```

- we can finally push it out into the galaxy:

```
docker push peerherholz/millennium_falcon:kessel-run
```

- and then we wait ...

Virtualization using containers - the build strikes back

- as mentioned before, we can combine the forces of `version control` and `virtualization` via integrating `GitHub` and `docker hub` to enable ***automated builds***

- the respective process is highly customizable and of course completely free
- all you need: a docker id and a *GitHub* account
- as usual, the setup is super easy and straightforward, as we just need to create a new *GitHub repository* in which we store our *Dockerfile*:
    - please create a new *Github repository* called `endor`
    - upload your `Dockerfile` to this *repository*

Virtualization using containers - the build strikes back

- now, on `docker hub`, go to your just uploaded `docker image` and within that, click on `builds`
- next, click on `Configure automated builds`
- we're now asked to indicate the `source repository` on `GitHub` and indicate some `build rules`, including `automated tests`, `branch` to `build` from, `tag` that should be applied, etc.
- after clicking on `save` and `build`, our `docker image` is `build automatically` from our corresponding `GitHub` repository
- depending on the size of your `docker image` and the current traffic on `docker hub`, it might take a while

Virtualization using containers - the build strikes back

- that’s it, that’s all we need
- the cool thing is, that every `push` or `commit` to this `GitHub` repository triggers a new `build`, hence your `docker image` remains nicely up to date
- you'll get email notifications wrt the currently running processes, also letting you know that the `automated build` finished

Virtualization using containers

Virtualization using Docker/Singularity - key points to remember

- The creation of `build files` and images is a **learning process with a steep curve** that can be convoluted and prone to errors, but the benefits can be tremendous: saving countless hours of installation, addressing reproducibility related problems, etc. .

- Certain tools, e.g. neurodocker, are very helpful wrt the **reproducible creation and testing** of complex images, as well as can combine python and container virtualization.

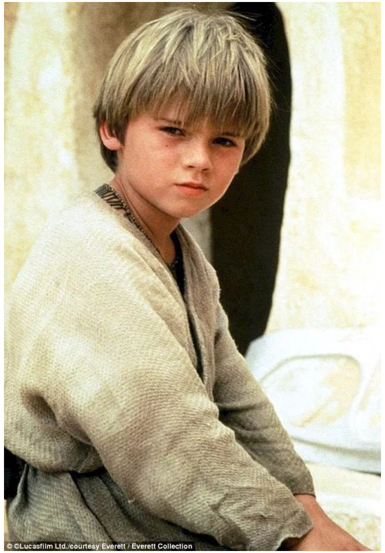
- Rebuilding images, e.g. during testing, **untags but not deletes** previous versions.
  - Potentially problematic wrt storage cluttering.

- If possible, your `image` should be **shared publicly** via `docker hub` through `naming`, `tagging` and `pushing` them.
  - Potentially problematic as old images will be deleted.

- `Github` and `Dockerhub` can be integrated to achieve **automated builds** via cloud instances.

The return of reproducibility

- no matter if you went on a selfish dark path of basically non-existent reproducibility and want to join the light side



https://img.srgcdn.com/e/w:1000/TG5lUThCQmVucVd2d3YzR2MyQjUuanBn.jpg

https://hotteahotbooks.files.wordpress.com/2012/09/darth_vader_i_am_your_father.jpeg

https://vignette.wikia.nocookie.net/starwars/images/e/ee/Spirits_copy1.jpg/revision/latest?cb=20060118180141

The return of reproducibility

- or if you were always on the light side of the force using it for the greater good

https://www.indiewire.com/wp-content/uploads/2019/11/960x0-2.jpg?resize=800,474

(NB: I know that Grogu is not baby yoda, I just wanted to have a nice continuation. Sorry.)

→

https://fthmb.tqn.com/5Jqt2NdU5fitcwhy6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

→

https://cdn.images.express.co.uk/img/dynamic/36/590x/secondary/yoda-force-ghost-1284826.jpg

The return of reproducibility

- we hope that this lecture/session provided you with an overview that will enable you to further dive into and explore the respective tools and hopefully utilize virtualization sufficiently and efficiently within your research workflow

- virtualization is great for:

  - sharing code/scripts/functions/pipelines with colleagues and everyone else without dependency issues (except virtualization software itself),
  - automize large parts of processing
  - rerunning analyses with identical or changed parameters

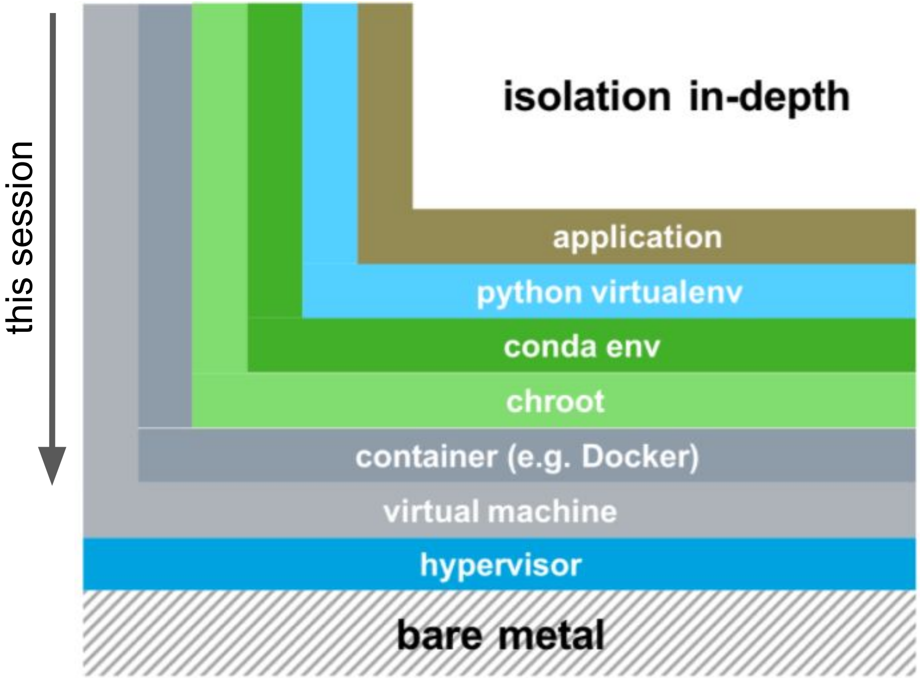
- virtualization is important for:

  - reproducibility of results
  - evaluation of soft-/hardware parameters

The return of reproducibility

Virtualization technologies have 3 main types:

- python virtualization
  - venv
  - conda
- containers
  - Docker
  - Singularity
- virtual machines
  - Virtualbox
  - VMware



The return of reproducibility

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

From The Turing Way, Ch. 2; doi:10.5281/zenodo.3233853

		Interaction style	
		GUI	CLI
Shared	SW	Binder	conda
	OS	Binder/VMs	container

Adapted from The Turing Way, Fig.33

The return of reproducibility

Virtualization technologies have 3 main types:

	python virtualization ○ venv ○ conda	containers ○ Docker ○ Singularity	virtual machines ○ Virtualbox ○ VMware
PROS	+ straight forward + (mostly) lightweight + cross-platform	+ share entire OS via dedicated infrastructure + complex environments & version control + comparably lightweight & fast + cross-platform & HPC	+ share entire OS + complex environments + cross-platform + GUI & thus “easier”
CONS	- restricted to python & adjacent SW (conda) - sharing cross-platform limited - no GUI	- steep learning curve - can reintroduce dependencies - no GUI	- very heavy & slow - (mostly) inefficient - no HPC & version control

The return of reproducibility

- further reading:
    - Understanding Conda and Pip
    - A beginner friendly intro to VMs and Docker
    - Intro to Docker from Neurohackweek
    - Understanding Images
    - Singularity examples
    - one day docker workshop
    - The Turing Way

The return of reproducibility

http://o.aolcdn.com/hss/storage/midas/c3fbbc0f098c3beb77509d60f084b8c2/203119611/HAN_FINAL_ILoveYou.gif

Virtualization using containers - notes on Singularity

- `Singularity` works comparably to `Docker`, except for the aforementioned points

- `Singularity` images are stored on singularity hub and can be pulled using:

  `singularity pull shub://`

- `Singularity images` can also be `pull`ed directly from `docker hub` (use with caution):

  `singularity pull docker://`

Virtualization using containers - notes on Singularity

- builds are working similar as well:

- `build` from a singularity file (called recipe):

  `singularity build *image_name*.img Singularity`

- again, `docker` support is no biggie:

  `singularity build *image_name*.img docker://`