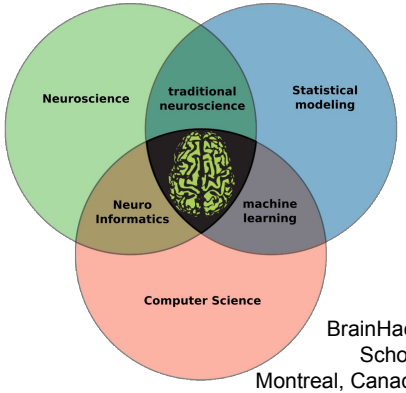


There and back again - A journey through virtualization

Instructor: Peer Herholz (@peerherholz)
TA: Sebastian Urchs, Elizabeth DuPre

BrainHack School
Montreal, Canada
May 11th - June 5th, 2020

CONDA

docker

BrainHack school MTL - week 1: NeuroDataScience Bootcamp - May 12th 2020

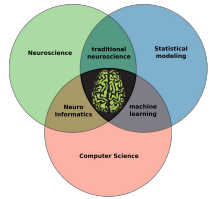
Objectives

Your role

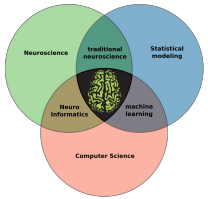
Schedule

Logistics

Good to know



Objectives



Gain skills

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

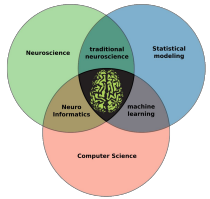
Get involved

- Familiarize you with the docker ecosystem for scientific work

Share

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

Objectives



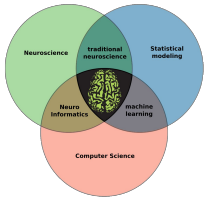
Your role

- ask questions
- 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:

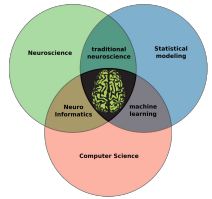
- 1 - 1:15 PM ~ Introduction and problem statement
- 1:15 - 2 PM ~ Virtualization using venv & conda
- 2 - 2:10 PM ~ Short break/discussion
- 2:10 - 3 PM ~ Docker 101 - A new hope
- 3 - 3:10 PM ~ Short break/discussion
- 3:10 - 4 PM ~ Docker 101 - The build strikes back
- 4 - 4:10 PM ~ Short break/discussion
- 4:10 - 4:30 PM ~ Docker 101 - The return of advanced concepts

Objectives

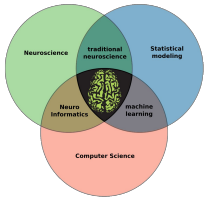
Your role

Schedule

Logistics



- Your machine:
  - able to get Docker up and running, pull Docker images
  - venv & conda
- You: shell
- This 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



Objectives

Your role

Schedule

Logistics

- get the materials via:

```
cd /path/to/wherever/materials/should/be/stored

git clone https://github.com/neurodatascience/course-materials-2020
```

Objectives

Your role

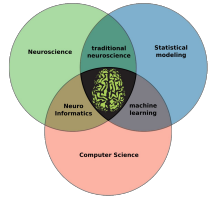
Schedule

Logistics

Good to know

- please netflix, instagram, twitch and what not kids do these days during the breaks only

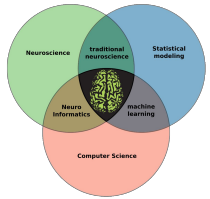
- That being said, if you want to tweet about what's happening (keeping folks who are not attending the course posted, you know: being open …) please use #brainhackschool



VIRTUALIZATION WARS

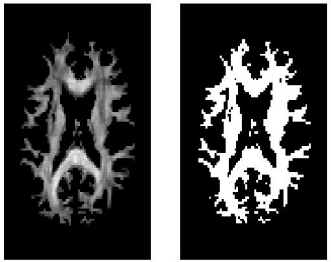
The problem statement

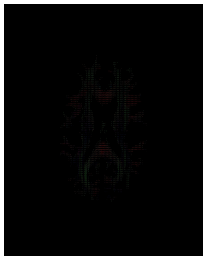
A colleague wrote a super fancy DTI analysis. Your PI is super happy and wants the whole lab and research galaxy to use it. Your colleague thus openly shares the script and everyone can use it ….



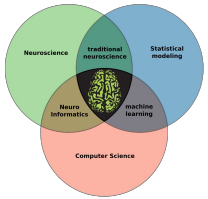
The problem statement

- within the NeuroDataScience 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`)





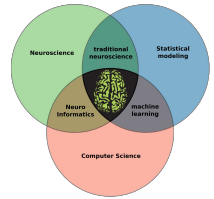

The problem statement



Waaaaaiiiit a hot Montreal minute!

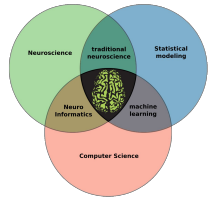
You don’t? What went wrong?

Let’s gather some errors here ...



The problem statement

https://towardsdatascience.com/how-to-build-a-linux-docker-image-fast-eeae2266f74a7



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

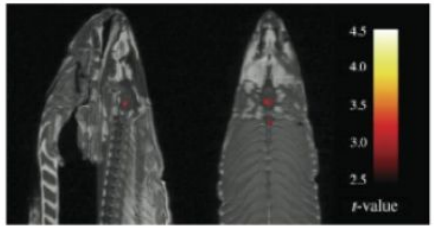
Aug 30, 2016 / David Biello https://ideas.ted.com/much-of-what-we-know-about-the-brain-may-be-wrong-the-problem-with-fmri/

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

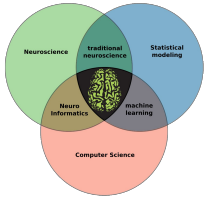
By Senzeni Mpofu
April 11, 2014 15:35

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

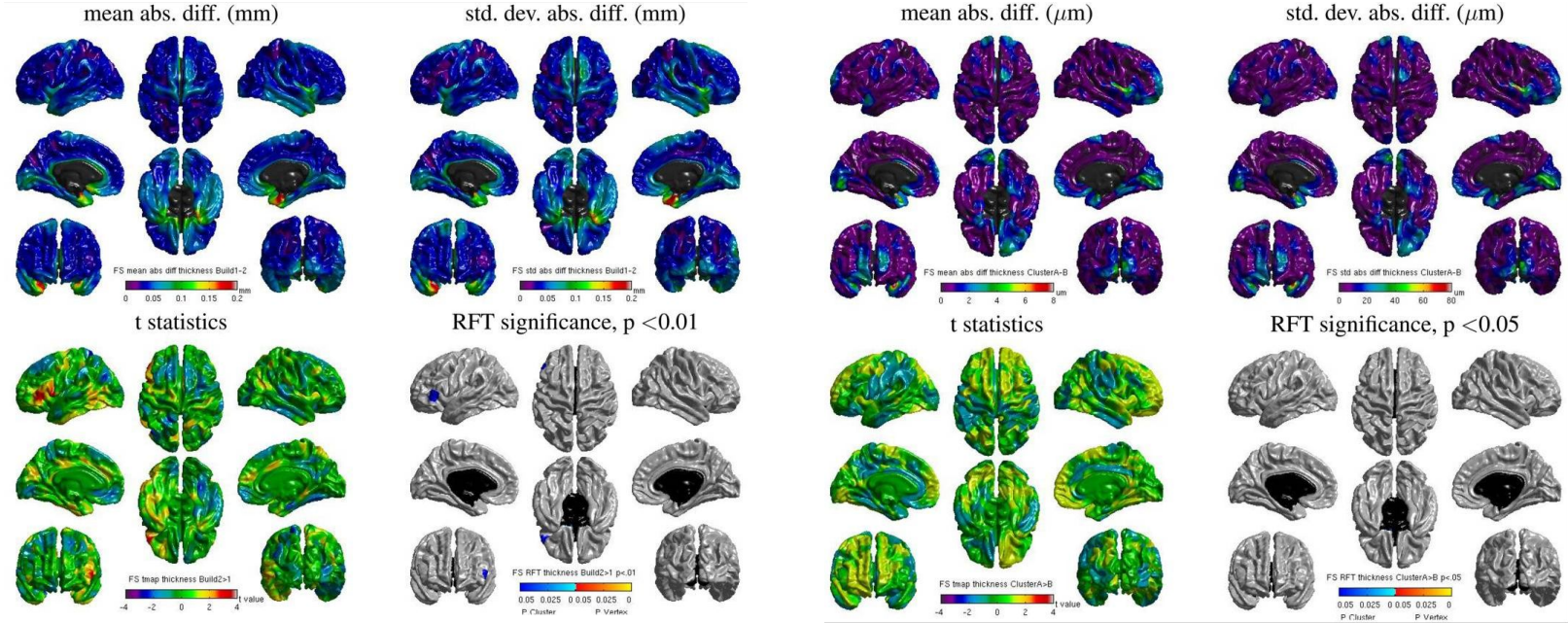
What Can fMRI Tell Us About Mental Illness?

By Neuroskeptic | January 14, 2017 10:53 am

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

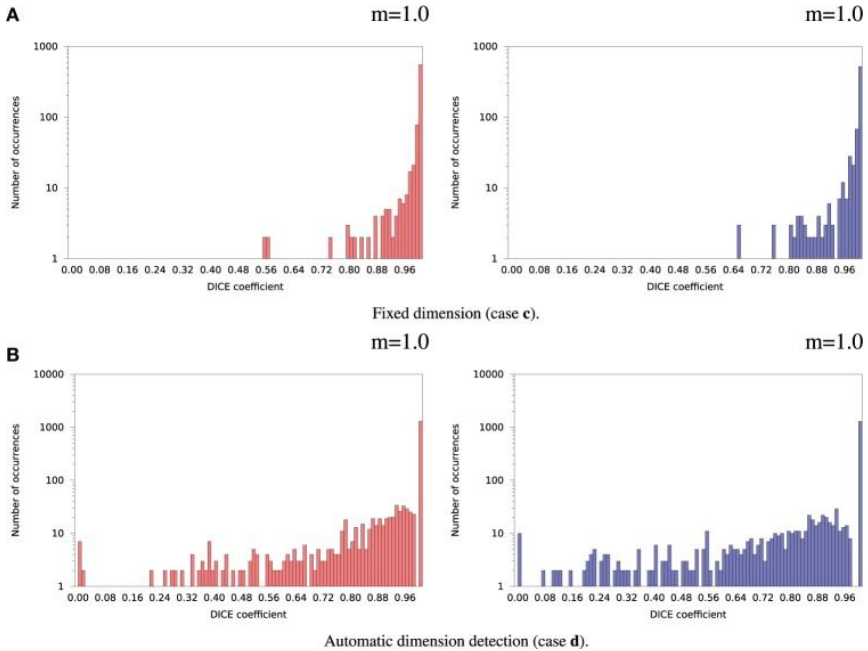


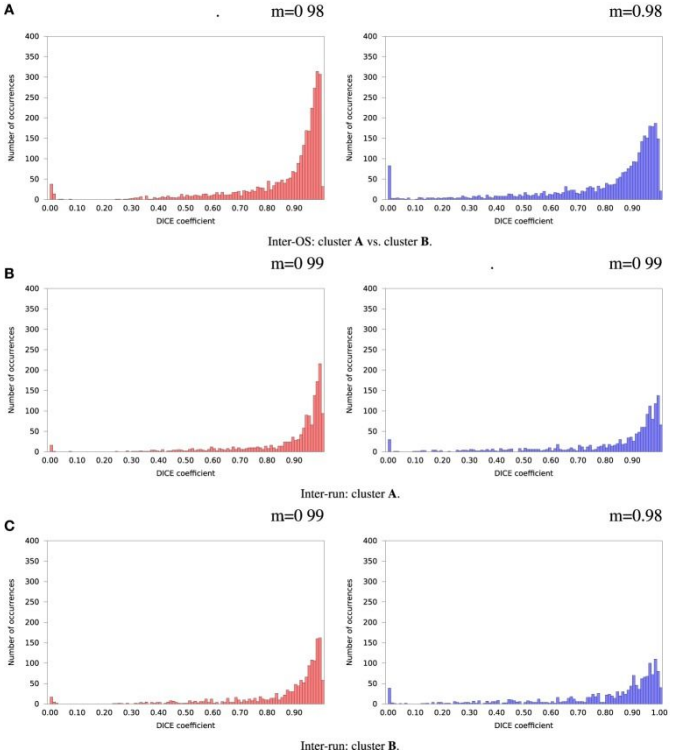
The problem statement



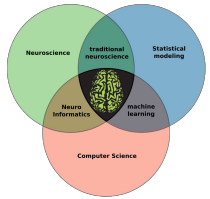
Glatard et al. (2015): Reproducibility of neuroimaging analyses across operating systems

The problem statement





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

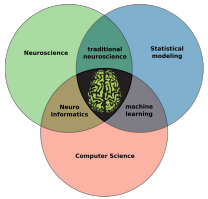


The problem statement

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)

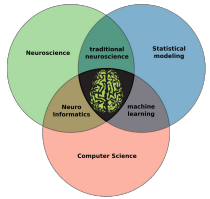


The problem statement

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.



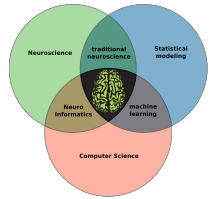
The problem statement

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)

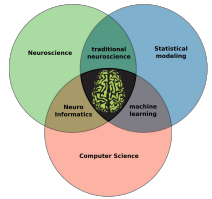
The problem statement



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...

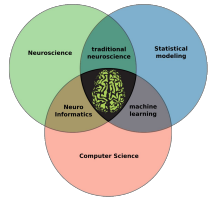
The problem statement - virtualization as solution?



Virtualization technologies aim to

- isolate the computing environment

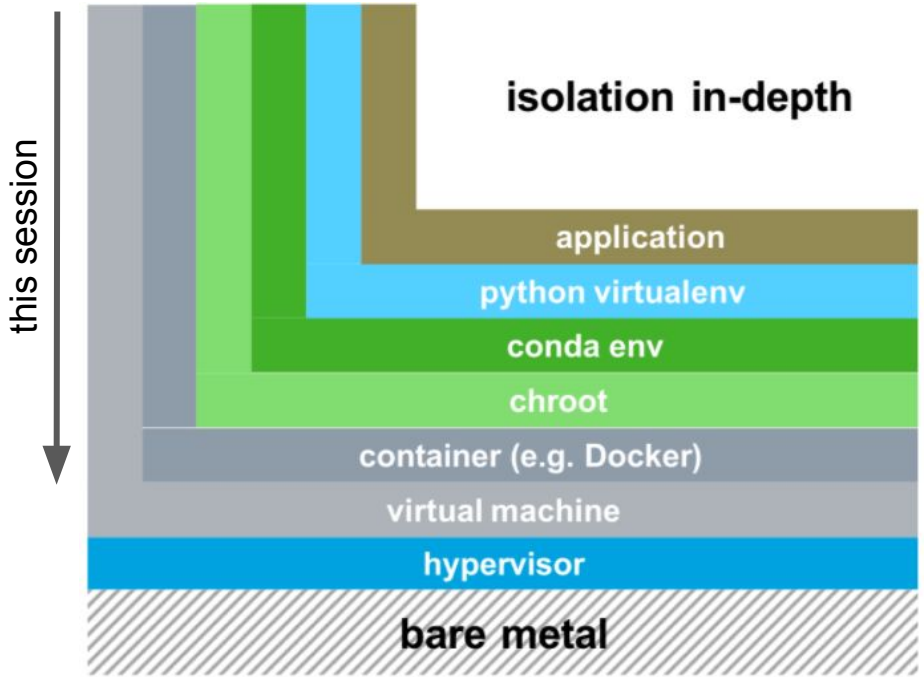
- provide a mechanism to encapsulate environments in a self-contained unit that can run anywhere
    - reconstructing computing environments
    - sharing computing environments

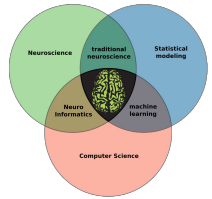


The problem statement - virtualization as solution?

Virtualization technologies have 3 main types:

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





Remember you must:

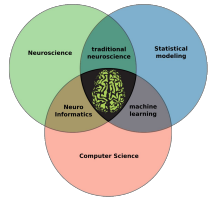
https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Every script/function and thus results dependent on complex computing environment are.

Isolating, reproducing and sharing of these environments the goal of virtualization is.

Multiple levels of virtualization exist: pure python, containers, virtual machines.

The problem statement - virtualization as solution?



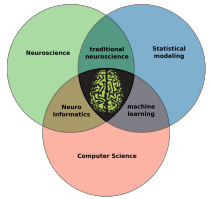
Going back to our fancy DTI analyzes: can we use these virtualization techniques to make it happen, that is running and producing the expected outputs?

Let’s find out and may the virtualization force be with us...

VIRTUALIZATION WARS

Virtualization using venv & conda

The research galaxy went on a dark path of non-existent-reproducibility. A small alliance of brave python based resources aim to bring back balance and ask you to join their movement ….



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

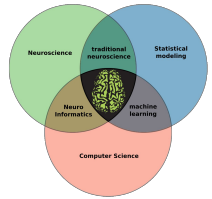
venv

- 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
- within a terminal type “which conda” and if it’s not installed do so now

Virtualization using venv & conda

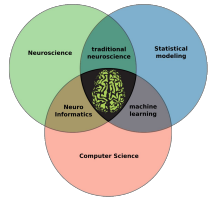


- before we start our fascinating adventure, we're going to create a directory within which we'll store the to be generated resources and materials
- Somewhere, beyond your machine … create a directory called "`galaxy`"

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

- within this new directory create a folder called "`tatooine`"

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



Remember you must:

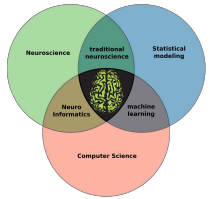
https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Copy paste if you must, but adapting the path and username to your machine necessary is.

Copy paste without thinking leads to the dark side.

Practice using the shell important is to become a master.

Virtualization using venv & conda

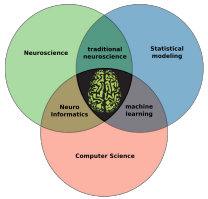


- the mysterious `venv` might hold the key to enable our fancy DTI analyzes
- we'll put this old tale to the test, working within a dedicated directory called "`moisture_farm`"

```
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

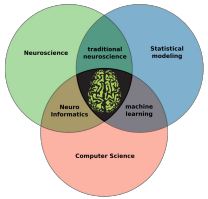
- you can use almost every character and name, but providing a meaningful name is never a bad idea, that being said, we’ll call our environment “`c3po`”:

```
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

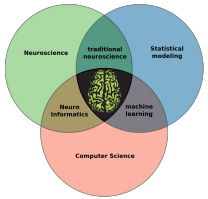


- we have three directories: `bin`, `include` and `lib`, here's what they entail:
  - `bin` : files that interact with the virtual environment
  - `include` : C headers that compile the Python packages
  - `lib` : a copy of the Python version along with a site-packages folder where each dependency is 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
- 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



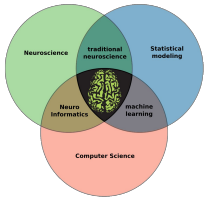
- using deactivate we (you guessed right) deactivate or "leave" our environment again

```
deactivate
```

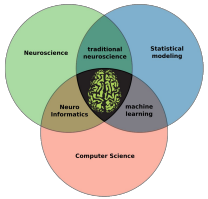
- now that we have that, let's try to run our fancy DTI analyzes again, after activating our environment (let’s move the function to our directory first for ease of use)

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

source c3po/bin/activate

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

Virtualization using venv & conda



- most likely, you'll receive the error message "`ModuleNotFoundError: No module named 'dipy'`"
- But why is that? The reason is fairly simple ...



Remember you must:

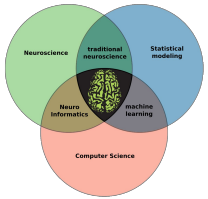
https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26vVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

When creating virtual environments, already installed packages not automatically included they are. Installing them again in the new environment you must.

Every python environment you have on your machine its own entity is in the sense that between environments
the binaries, libraries, etc. are not shared

Which python environment is used to execute a certain functionality set by the `$PATH` variable in your shell profile is or manually by you, activating it.

Virtualization using venv & conda



- you can check which python environment is currently set by running “ `which python` “ within your shell:

  ```
  which python
  ```

- you should see that the environment that is currently running is the one we just created. Now let's deactivate it and try again:

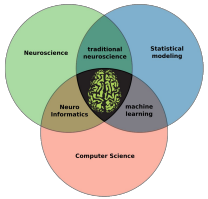
  ```
  deactivate
  ```

  ```
  which python
  ```

- you should now see the environment that is running as default on your system. Let's investigate `$PATH` to further grasp what's going on.

  ```
  echo $PATH
  ```

Virtualization using venv & conda



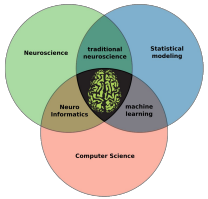
- and now from within our new environment:

```
source c3po/bin/activate

echo $PATH
```

- the first instance of `$PATH` is our newly created python environment, thus for everything that will be run from the command line the first path within which the search of an executable will take place is that environment and not your default one, meaning that everything will be executed through/within that environment based on its resources

- while this is at the core of virtualization it also requires you to pay close attention to what and how you're executing functions, scripts, etc.

Virtualization using venv & conda



- let's populate our so far rather empty environment with, foremost, `DIPY`:

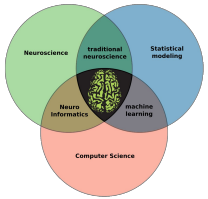
  ```
  pip install dipy
  ```

- this not only downloaded and installed `DIPY`, but also all its dependencies, which is based on most python libraries containing a `requirements.txt` where necessary dependencies are listed (the `DIPY` one looks like this)

- this handy functionality is possible through pip being a package manager, which also allows the easy investigation of our environment, for example which packages are installed:

  ```
  pip freeze
  ```

Virtualization using venv & conda



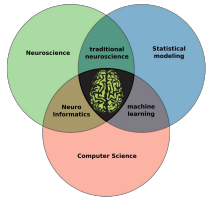
- we can also save the output of pip freeze to create our own requirements.txt which you then can share with whoever is interested in running your scripts and analyses using the same python libraries with the identical version:

```
pip freeze > requirements.txt

cat requirements.txt
```

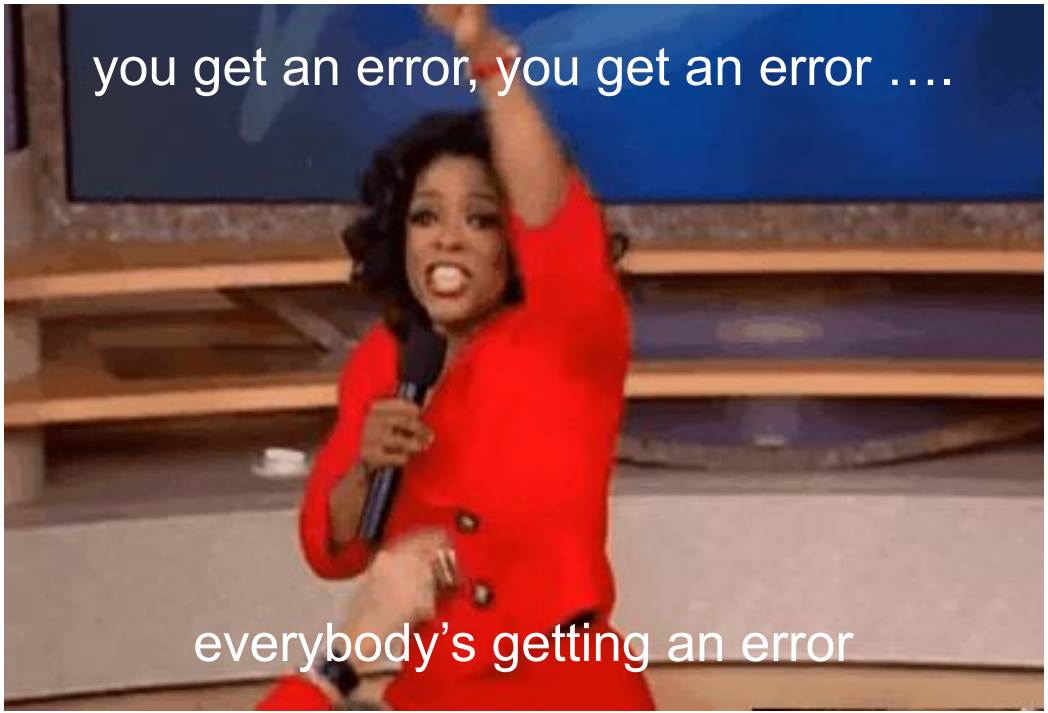
- this is one form of **virtualization of python environments**

- before we share our environment, we should ensure that everything is working as expected:

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

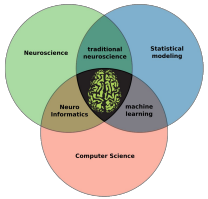


Virtualization using venv & conda



https://giphy.com/gifs/amyposhlersmartgirls-reaction-excited-oprah-xT0BKqB8KIOuqJemVW

Virtualization using venv & conda



- now we can work on resolving this error and thus create an environment that is rather specifically than sharing our own most likely large default python environment with lots of libraries that are not necessary

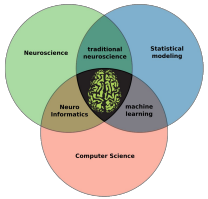
- *most* python packages have very useful error message like the one we're receiving, paying a bit more attention, we see that something is missing and after a short DuckDuckGo (or any other search engine) session, we know that the `fury` package is missing, which is an easy fix:

```
pip install fury
```

- as you can see, `vtk` was installed as a dependency of fury. So far so good, let's try the example again:

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

Virtualization using venv & conda



- the fun continues (remember this feeling of being stressed and annoyed, it'll be important later on) … we need to install matplotlib:

```
pip install matplotlib
```

- at this point "*it works for me*" (you remembered that one, didn't you?) as I get the expected outputs which can be checked via `ls`:

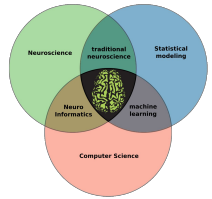
```
python ../../fancy_DTI_analyzes.py
ls
```

- I can even run it in `interactive` mode by changing the `interactive` variable within the function to `True`

- thus I can update my `requirements.txt` and share it

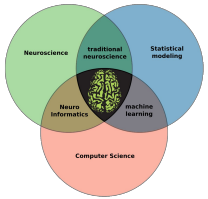
```
pip freeze > requirements.txt
cat requirements.txt
```

Virtualization using venv & conda



- however, we most likely have multiple problems and limitations:
  - for some of you, it still won't work
  - either at all or in `interactive` mode
  - sharing my `requirements.txt`, won't share my entire `python environment`, especially not `python` itself
- while everything so far was a good start, it usually won't be enough even though you already did more than the majority of scientists wrt reproducibility and sharing
- let's go one step further with `conda`

Virtualization using venv & conda



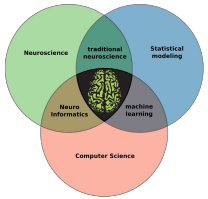
- as before, let's create a directory for our journey, this time it's called " `mos_eisley` ":

```
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 `dipy`, `fury` and `matplotlib`:

  ```
  conda create -y -n r2d2  python=3.7 dipy matplotlib
  ```

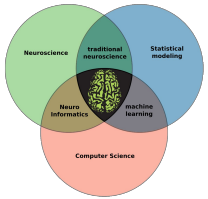
- `conda`, by default, already installs a fair amount of libraries as compared to `venv`

- 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



- 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
  ```

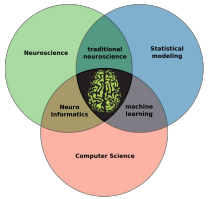
- instead of using "`source`" and pointing the wanted "`activate.sh`" script, we use `conda` specific commands as `conda` is it's own program, with its `environments` not being created at a specified directory as through `venv`, but within the `conda` installation at hand

- 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



- we can also check all the installed packages, along with important information such as the specific `version`, `build` and `channel`:

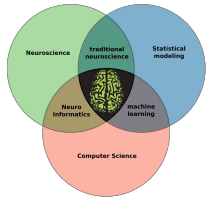
  ```
  conda list
  ```

- as within `venv`, we can `export` this `environment` and share it, the syntax however is somewhat different:

  ```
  conda env export > r2d2.yml
  ```

- we tell `conda` that we want to do something with the currently activate `environment`, that is `export`ing it, saving the required information to a `yaml file` called `r2d2.yml`

- besides all `packages` and their specific `version`, this file further contains the required `conda channels`, `python version` and `prefix` used at machine 0



Virtualization using venv & conda

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

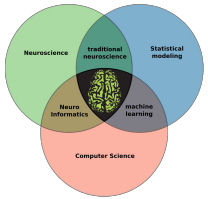
```
python ../../fancy_DTI_analyzes.py
```

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

I find your lack of controlling and evaluating installation processes disturbing.

Even though specified, `fury` was not installed and it never will be using `conda` as it's only available through `pip`.

Virtualization using venv & conda



- the dark side appears to be strong wrt unreproducible practices … luckily, we are already padawans and can tackle this without problems:

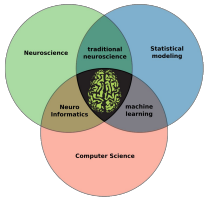
```
pip install fury
```

- and we try it again:

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

- hate to tell you once more: *it works for me*, but most likely not for everyone...

- the force of pure python virtualization is quite often not enough, as certain libraries and binaries, thus dependencies on the operating system level, are needed, our endeavour requires more ...



Remember you must:

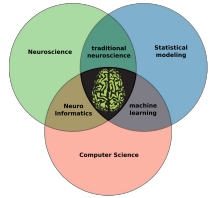
https://ftbmb.tqn.com/5Jqt2NdU5fitcwhv6FwNq26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Sharing of specific `python libraries` and their `version` possible through `venv` and `pip` is.

To also share the specific `python version`, `builds` and `channels` `conda` you need to use.

`Conda` is strong as `environment` (comparable to `venv`) and `package` (comparable to `pip`) `manager` it combines.

Pure `python virtualization` not for often required `libraries` and `binaries` on other levels does account.



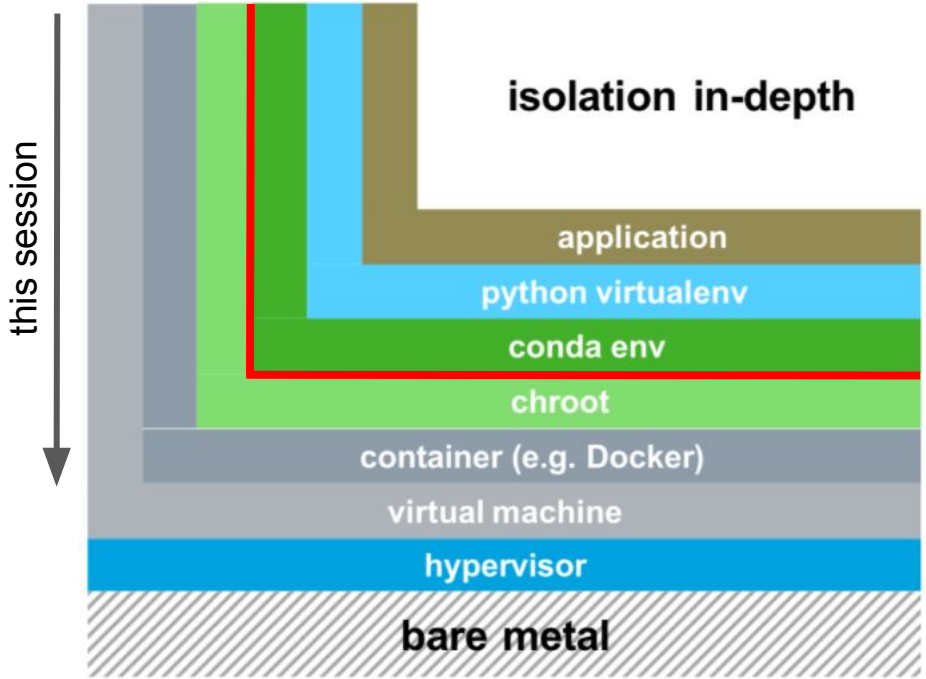
With a *New Hope* on the horizon, let's relax for 10 min

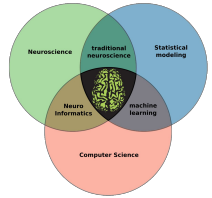
https://i.pinimg.com/600x315/e5/74/6a/e5746a117cbc6c0886090ac21536a7ab.jpg

The problem statement - virtualization as solution?

Virtualization technologies have 3 main types:

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





The problem statement

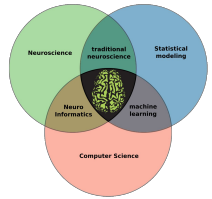
https://hotteahotbooks.files.wordpress.com/2012/09/darth_vader_i_am_your_father.jpeg

https://miro.medium.com/max/1400/1*A-lD0FHKmChS2vGelhUs2g.png

VIRTUALIZATION WARS

Docker 101 - A new hope

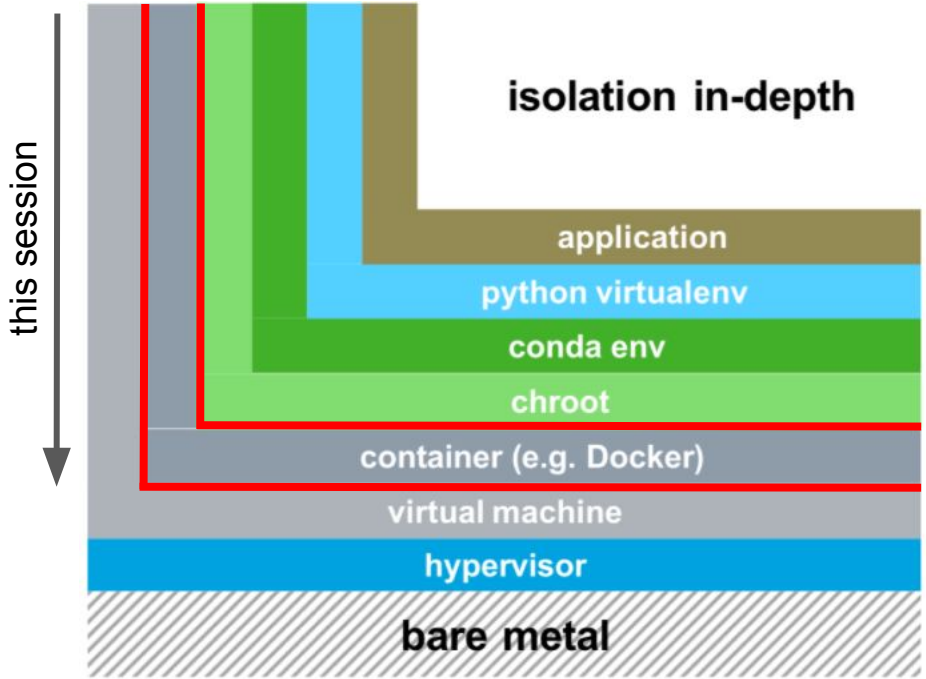
After a long time of training, you became a virtualization padawan which enables you the use a stronger virtualization force than pure python virtualization. A new, much needed, hope on the horizon ….



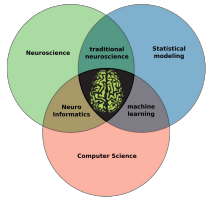
Docker 101 - A new hope

Virtualization technologies have 3 main types:

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

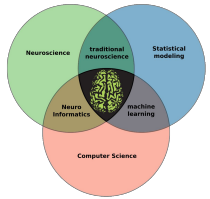


Docker 101 - A new hope

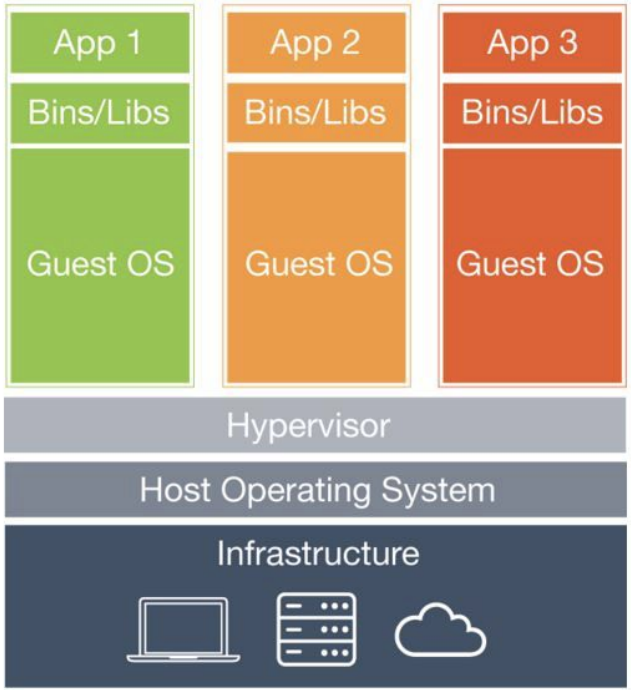


As we already saw in the Old Republic of `venv` and `conda`: the force of `pure python virtualization` is not enough. We need more force (virtually and literally) and thus need to explore new frontiers like `containers`.

However, before we can utilize this tremendous force, we need to learn about it. Let me welcome you in the `Docker` academy of the Container order...

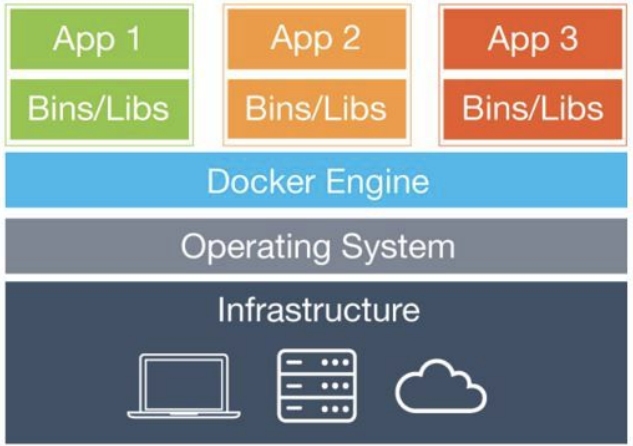


Docker 101 - A new hope



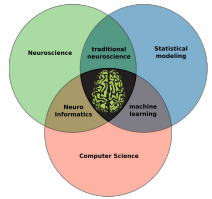
virtual machines

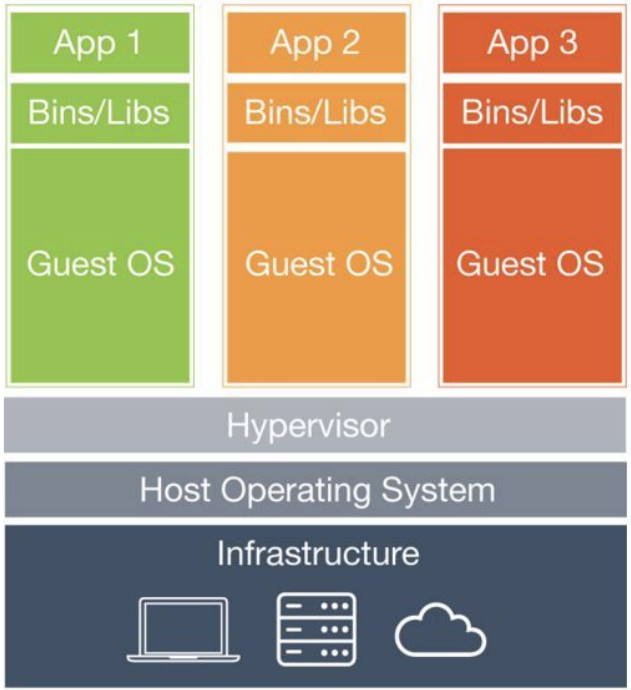
VS



containers

Docker 101 - A new hope

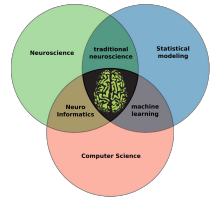




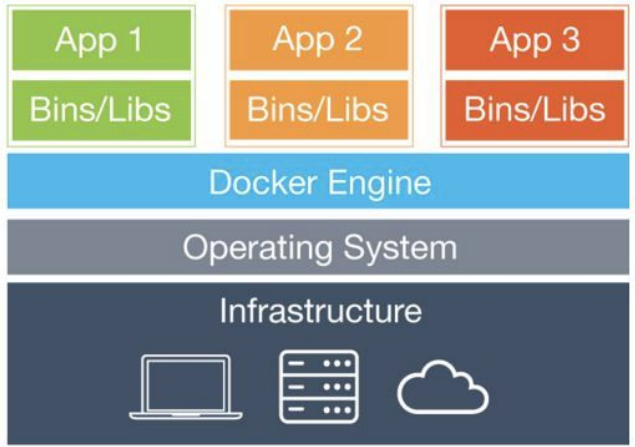
virtual machines

- 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

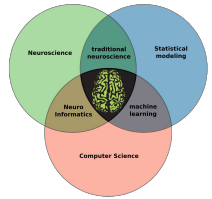
Docker 101 - A new hope



- share the host system's kernel with other containers
- 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

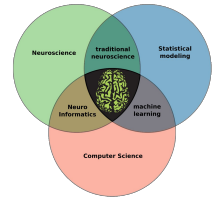


Docker 101 - A new hope

- 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

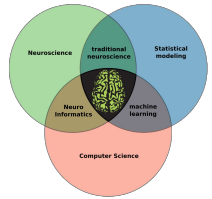
Docker 101 - A new hope



vs

- 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

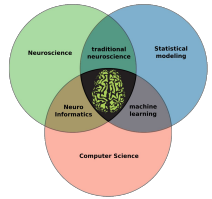
- a container solution created for scientific and application driven workloads
- supports existing and traditional HPC resources
- a user inside a Singularity container is the same user as outside the container
- can run existing Docker containers



Docker 101 - A new hope

https://i.redd.it/si1z0556v3m11.png

More on this within Felix' amazing lecture on Wednesday.



Remember you must:

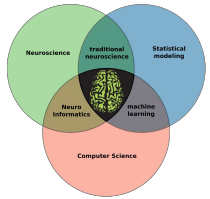
https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26vVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Shared the host's kernel between containers is.

Containers way more lightweight and faster than virtual machines are.

Singularity the way to the HPC side of the container force is.

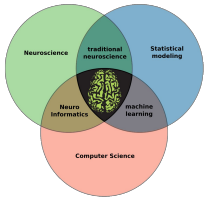
Docker 101 - A new hope



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

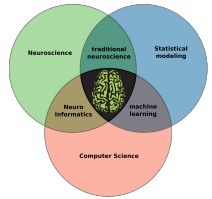
Docker 101 - A new hope



- 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:

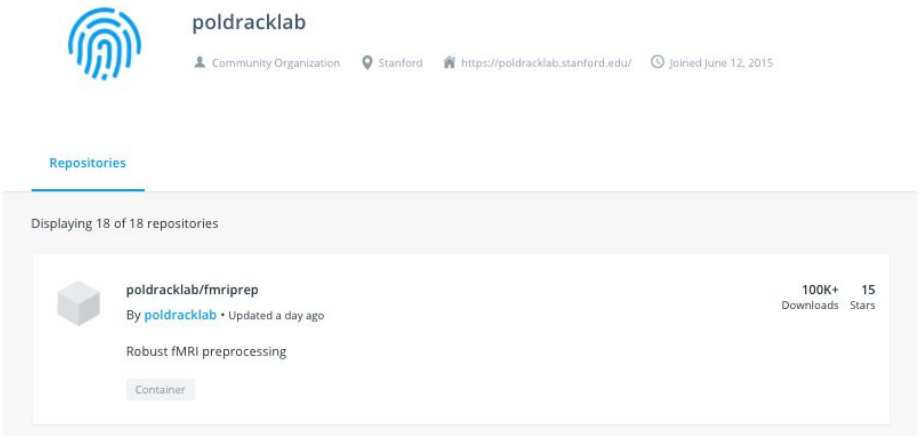
```
docker run hello-world
```

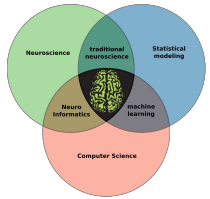
- What is happening? The above command ...
  - tells `docker` to `run` or `execute` the container `hello-world`
  - if the wanted `container` is not already on your machine, `docker` automatically searches for and downloads it
  - `docker` run then `runs` or `executes` the given `container` doing whatever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



Docker 101 - 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 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`





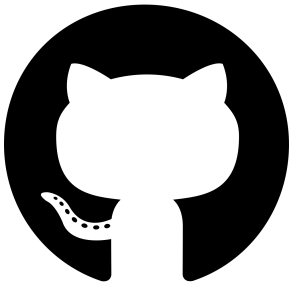
Docker 101 - A new hope

- Docker hub - a special place in the galaxy

    - once build and tagged, docker images can be pushed to docker hub via command line:

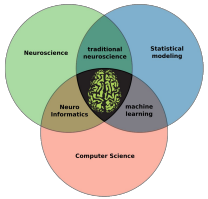
      `docker push death-star`

    - or automatically build from a `GitHub repository` after pushing `commits` to a respective repo



+

Docker 101 - A new hope

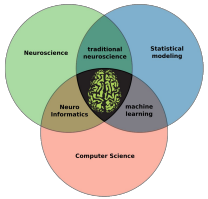


- now let's further explore `docker commands` within a typical workflow
- at first, we want to download a certain `docker container` to work with, for the sake of simplicity and time we're going to use the classic `ubuntu container`
- instead of automatically downloading the container via `docker run`, 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

Docker 101 - A new hope



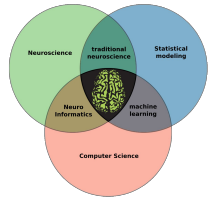
- we can check existing Docker images using:

  `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`



Remember you must:

https://ftbmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

No GUI Docker has.

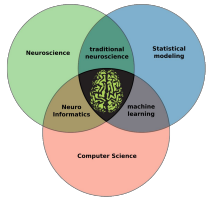
On Docker Hub Docker images hosted are.

Docker pull by default always for tag latest searches, to specific tags and thus versions necessary is.

Docker images composed of layers are, between images they shared.

Entire OS Docker images contain.

Docker 101 - A new hope



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

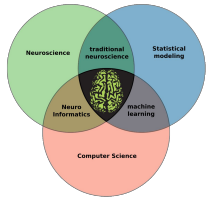
  ```
  docker run ubuntu
  ```

- `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

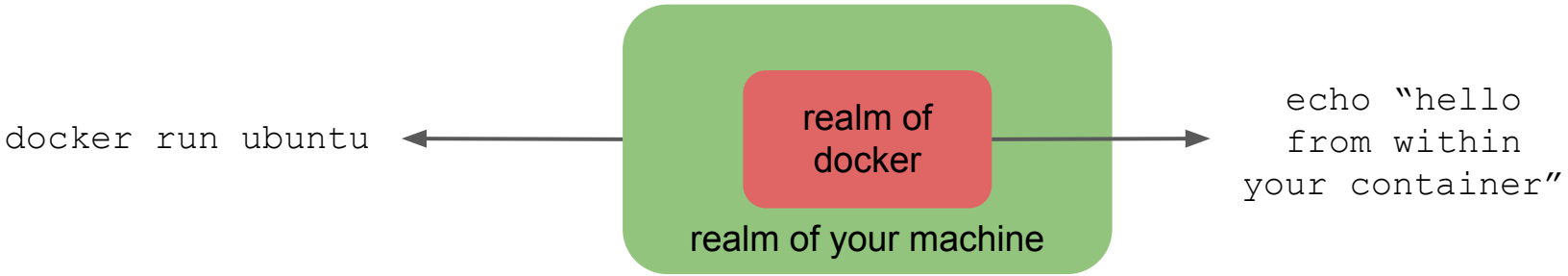


Docker 101 - A new hope

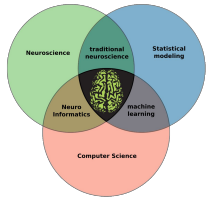
- *automated functionality* poses as one major approach to use `docker images` (e.g. within pipelines and workflows), but more in that later
- it's also possible to define tasks during inition:

  `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 ...



Docker 101 - A new hope



- 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:

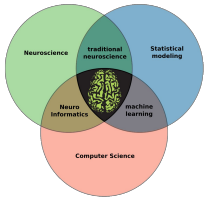
```
docker run -it ubuntu
```

- 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`

Docker 101 - A new hope



- 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
  ```

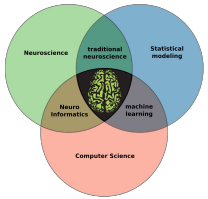
Docker 101 - A new hope

- in order to ensure that a given `docker container` is removed from `running instances` after exiting, the `--rm` flag can be included in the `docker run` command:

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

https://tipaltilabs.com/2018/06/26/nodejs-ms-sql-integration-testing-with-docker-mocha/

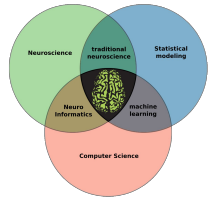
Docker 101 - A new hope



- now let's go back to our main goal: making the fancy DTI 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 ...

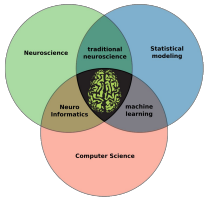


Docker 101 - A new hope

VIRTUALIZATION

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

Docker 101 - A new hope



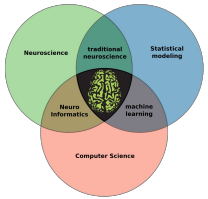
- when working within a `docker container`, **creating**, **modifying** and **deleting** files, **changes are neither permanent nor saved**, as this is against the encapsulation and reproducibility idea

- 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`

Docker 101 - A new hope



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

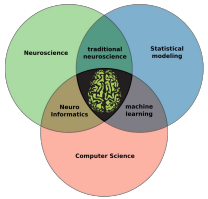
- for example, let's create a directory called “ `hoth` ” within our “ `galaxy` “ folder and make it available inside the docker container as “ `rebel_base` “ :

```
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
```

Docker 101 - A new hope

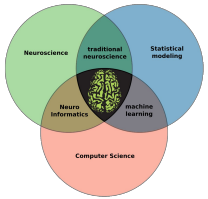


- you can also restrict the rights of mounted paths to e.g. read only in case any modification should be prevented (for example within automated functionality or if the empire wants to destroy your plan)):

```
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

Docker 101 - A new hope

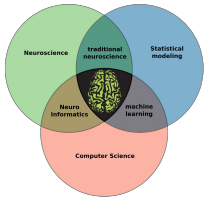


- in our example the directory `/rebel_base` didn't exist before mounting it, hence it was automatically created, however this is also depends on the `docker image` and it's setup/definition at hand as e.g. within automated functionality a certain directory can be expected

- 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`

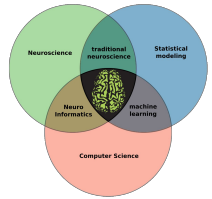
Docker 101 - A new hope



- 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 mission (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



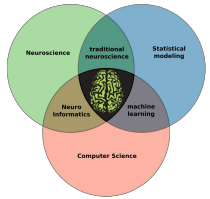
Remember you must:

https://ftbmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Usually Docker images specific purposes have, automated functionality and interactive usage examples for this are.

Running instances need to be checked and images never can be changed.

The Docker realm and your host machine realm two seperate places in the universe are. Connected through a mount they must be if you need to provide a flow of data. pull by default always for tag latest searches, to specific tags and thus versions necessary is.



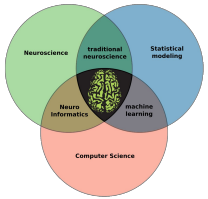
Remember you must:

https://ffhmb.tqn.com/5Jqt2NdU5fitcwhv6FwNq26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Time to test your Docker force it is:

- the `neurodebian docker image` in its `nd-non-free` version you must pull
- mount the `dagobah` directory on your machine to `/x-wing` within the `docker container`
- create a `.txt` file called `new formation plans.txt` within the `/x-wing` directory

Docker 101 - A new hope

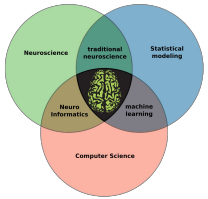


- let’s try if our fancy DTI analyzes works by making it available within the Docker realm:

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

python /rebel_base/fancy_DTI_analyzes.py
```

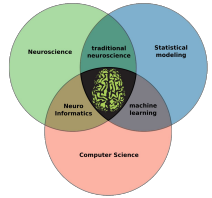
- python is not even installed… feeling the frustration? Good. Remember it, but don’t let it corrupt you (the dark side, etc.). Great things and new frontiers are waiting for us...

The problem statement



After spending quite some time as virtualization padawans, the reproducibility council entrusts us with more challenging missions.

We need to expand our Docker force, as we need to provide custom Docker images. We need to learn how to build Docker images ...



With something dark on the horizon, let's relax for 10 min

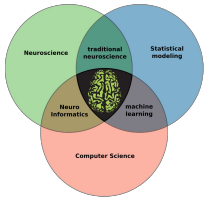
https://i.pinimg.com/600x315/e5/74/6a/e5746a117cbc6c0886090ac21536a7ab.jpg

VIRTUALIZATION WARS

Docker 101 - The build strikes back

The dark side is strong. You need to become a master of virtualization to tackle the empire of non-existent-reproducibility that agonizes the galaxy ….

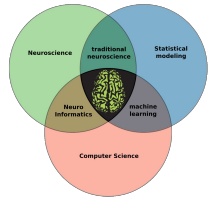
Docker 101 - The build strikes back



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

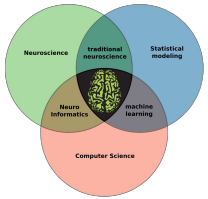
- but how are *docker images* actually created for specific projects, pipelines, etc.*a*?
- how can different programs be installed?
- how are locally created *docker images* shared?

Docker 101 - The build strikes back



- when it comes to creating `docker images`, two essential parts are relevant:
- a ***Dockerfile***
- the ***`docker build`*** command

Docker 101 - The build strikes back



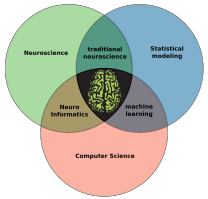
- at the beginning there was a ***Dockerfile***…
- a ***Dockerfile*** is a `text file` that contains a mixture of `docker build` and `bash` commands

- being the focus of our new mission, let's create one in our hoth directory

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

touch Dockerfile
```

- next, we're opening it within VS Code for an initial inspection

Docker 101 - The build strikes back



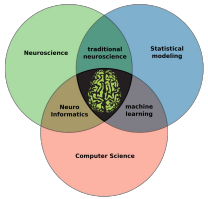
- it is …. empty … however, through the old tales of building `docker images`, we learn that we need to populate it, beginning with the `base`
- this corresponds to the underlying OS we want (or need to use)
- given that we already worked on it, we are going to use `ubuntu`:

```
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"
```

Docker 101 - The build strikes back



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

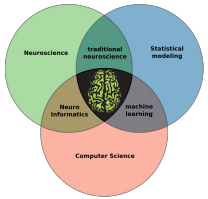
```
ENV LANG="en_US.UTF-8" \
LC_ALL="en_US.UTF-8" \
ND_ENTRYPOINT="/docker/startup.sh"
```

- 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 .
```

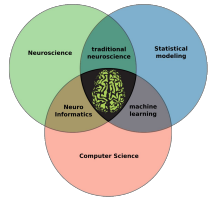
Docker 101 - The build strikes back



- through the force, everything went super fast and with the message “`Successfully tagged millennium_falcon:latest` ” your first own `docker image` was `build`, which you can check via:

  ```
  docker images
  ```

- why is the force so powerful here? Remember your training you must ...



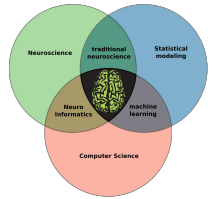
Remember you must:

https://ftbmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Never be changed can a given image.
To be build new ones in case functionality is missing or needs to be adapt thus is necessary.

Indicating the base command From ubuntu this is, as starting from this image we do to create a new one.

To share layers between images possible is and image already existed and nothing added was. The build process therefore very fast was.



Remember you must:

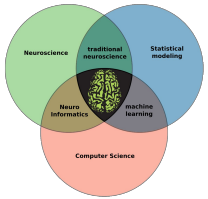
https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

Images composed of different layers are.

The build process this revealing to you is.

The population of a Dockerfile comparable principles follows, adding components and functionalities through layers.

Docker 101 - The build strikes back

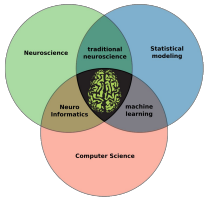


- as the force within our image is rather limited at this time, let's install some other useful packages and programs

- 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 \
```

Docker 101 - The build strikes back



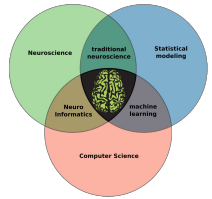
- from this point onwards, `Dockerfiles`, thus the creation of docker containers are almost identical to setting up any other `linux OS`, e.g. using `apt-get` or setting environment variables

- let's rebuild our *Docker image* again:

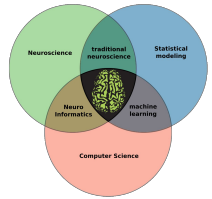
```
docker build -t millennium_falcon .

docker images
```

- the `layer` principle comes to our aid again, as we don't need to `pull layers` we already have, but only those we need to `build` the new `image`

Docker 101 - The build strikes back

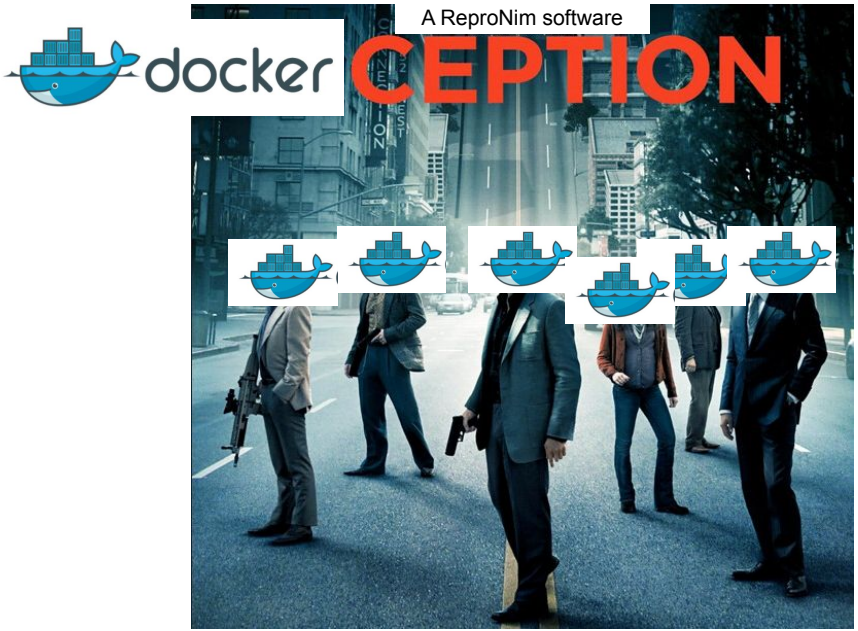


- while this already seems like an overkill and very complex, we just created a very basic `image`
- the fitting compilation of an entire `Dockerfile` for a complex script, pipeline, analysis, appears very convoluted and prone to errors, as well as takes a lot of time and searching the galaxy (especially in the beginning)
- through the force, we can see into the future and the prophecy was right: the Dockerfile we need to enable our fancy DTI analyzes is indeed tremendously complicated ...
- you might wonder: Isn't there a more sufficient, faster and easier way of using the Docker force?



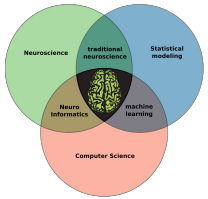
Docker 101 - The build strikes back

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



https://www.flickr.com/photos/methodshop/5245100040

Docker 101 - The build strikes back



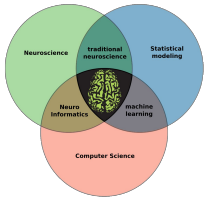
- let's check how we can recreate our `Dockerfile` using `Neurodocker`
- at first we need to get the `Neurodocker image`:

```
docker pull kaczmarj/neurodocker:0.4.3
```

- after checking the `readme` and `docs`, we know that we need to `run Neurodocker`, providing the necessary input arguments beginning with stating that we want to create a `Docker image` and that we want to use `ubuntu` as the base:

```
docker run kaczmarj/neurodocker:0.4.3 \

           generate docker --base=ubuntu \
```

Docker 101 - The build strikes back

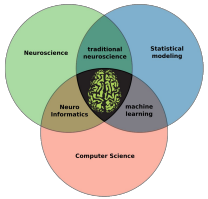


- next, we indicate that we want to use apt as a package manager, furthermore specifying the packages to install:

```
docker run kaczmarj/neurodocker:0.4.3 \

           generate docker --base=ubuntu \

           --pkg-manager=apt --install git nano unzip
```

- if we now `run` this command, you can see, the output is the content of our `Dockerfile`

Docker 101 - The build strikes back

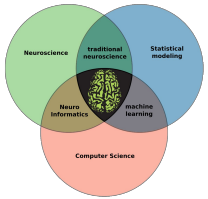


- as the latter is not automatically created, we need to pass the output of `Neurodocker` to the `Dockerfile` using >:

```
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 \
            > Dockerfile
```

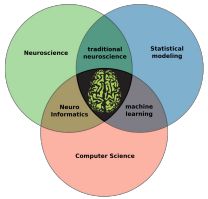
- that's all we need to recreate our image from before, except saving the stress and work of writing manually

Docker 101 - The build strikes back



- the force is indeed strong on `Neurodocker` and makes our reproducibility life super easy
- we know that we need to include a `python distribution` to run our fancy DTI analysis, preferably to one we used in our `conda virtualization` endeavor
- once more we remember our training and summon that we need to include at least `python 3.7`, `dipy` and `fury`
- creating a respective `conda environment` within our `image` using `Neurodocker` is as easy as space chess

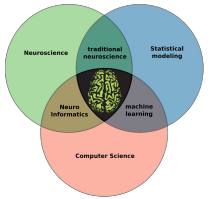
Docker 101 - The build strikes back



- all we need to do is adding that we want to use `miniconda` to install a `conda environment` with a specific `python version` and `packages`:

```
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 dipy" \
          pip_install="fury" create_env="r2d2" activate=true \
           > Dockerfile
```

Docker 101 - The build strikes back



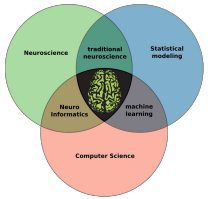
- with that force we have efficiently combined the two virtualization approaches within which we were trained, `python environments` and `containers`
- let's rebuild our *Docker image*:

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

- and try our fancy DTI analysis again:

```
docker run -it --rm \
           -v /Users/peerherholz/Desktop/galaxy/:/rebel_base:ro\
           millennium_falcon

python /rebel_base/fancy_DTI_analyzes.py
```

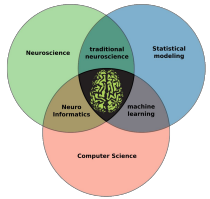
Docker 101 - The build strikes back



- a bit of searching around the galaxy, we find out that we to include `libgl1-mesa-glx` & `libsm6`:

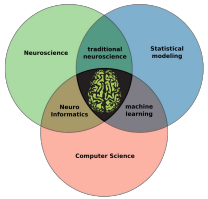
```
docker run kaczmarj/neurodocker:0.4.3 \
           generate docker --base=ubuntu \
           --pkg-manager=apt --install apt-utils bzip2 \
           ca-certificates curl git locales nano unzip libgl1-mesa-glx \
           libsm6 \
           --miniconda conda_install="python=3.7 dipy" \
           pip_install="fury" create_env="r2d2" activate=true \
            > Dockerfile
```

Docker 101 - The build strikes back



- and here we go again, first the build, then testing the fancy DTI analysis:

```
docker build -t millennium_falcon .

docker run -it --rm \
           -v /Users/peerherholz/Desktop/galaxy/:/rebel_base:ro\
           millennium_falcon


python /rebel_base/fancy_DTI_analyzes.py
```



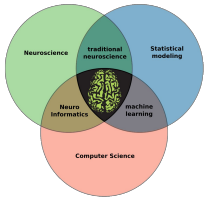
Docker 101 - The build strikes back

- a mysterious legendary error message appears:

```
ERROR: In
/work/standalone-x64-build/VTK-source/Rendering/OpenGL2/vtkXOpenGLRe
nderWindow.cxx, line 273

vtkXOpenGLRenderWindow (0x559825b22a80): bad X server connection.
DISPLAY=Aborted
```

- the problem is: `containers`, by default, have no `window`, no `screen` to `access` and thus `generate` and `save graphics`, corresponding code either throws an error or leads to graphics that are lost in the digital nimbus

Docker 101 - The build strikes back

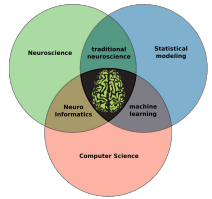


- after another quick search session, we find a potential solution: the `xvfb-wrapper` which should help you with problems related to the `VTK` and `mayavi` empire

- we need to include a virtual display setup within our fancy DTI analysis:

```
from xvfbwrapper import Xvfb

vdisplay = Xvfb(width=1920, height=1080)

vdisplay.start()
```

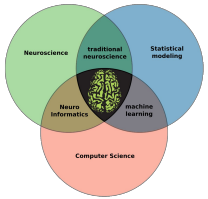
Docker 101 - The build strikes back



- and include `xvfb` as well as `xvfbwrapper` in our `Neurodocker` command and thus `build`:

```
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 libgl1-mesa-glx \

           libsm6 xvfb\

           --miniconda conda_install="python=3.7 dipy" \

           pip_install="fury xvfbwrapper" create_env="r2d2" activate=true \

            > Dockerfile
```

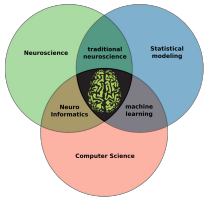
Docker 101 - The build strikes back



- the same procedure as every error ….

```
docker build -t millennium_falcon .

docker run -it --rm \
           -v /Users/peerherholz/Desktop/galaxy/:/rebel_base:ro\
           millennium_falcon

python /rebel_base/fancy_DTI_analyzes.py
```

Docker 101 - The build strikes back



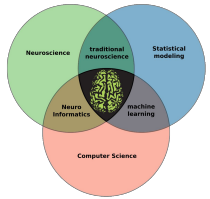
- waaaaaaiiiiiit … no errors? Let's check if the output is there:

  `ls`

- fellow virtualization academy folks, we did it. We successfully encapsulated an entire OS with software dependencies required to run our fancy DTI analysis no matter where, producing the exact same results!

- the only thing left to do before you can emerge from the ranks of padawan to become a master of virtualization is to share the fully functional `docker image`

Docker 101 - The build strikes back

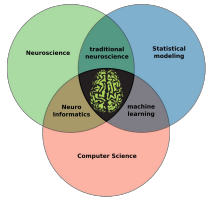


- while we could use the `docker save` and `load` commands to share our `image` locally via e.g. USB, we are going to use this thing called the internet to share it on `docker hub`
- to do so, we need essentially two lines of code, one to `tag` our `image` and one to actually `push` it:

  ```
  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

Docker 101 - 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:

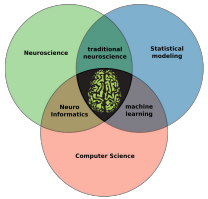
```
docker images
```

- we can finally push it out into the galaxy:

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

- and then we wait ...

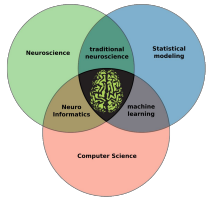
Docker 101 - The build strikes back



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

```
docker images -a
```

- 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 and keep track of tremendously large software in order to keep your *docker image* as small as possible



Remember you must:

https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6FwNg26vVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

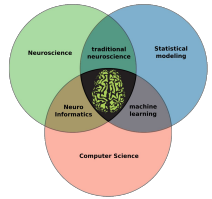
`docker build` and `push` everything you need is.

`Dockerfiles` very quickly very complex can get.

Combining `python` and `container virtualization` you can and should.

Tools like `Neurodocker` helpful for this purpose are.

By default, no `window`/`screen` available is within a container.



Remember you must:

https://ffhmb.tqn.com/5Jqt2NdU5fitcwhv6FwNq26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

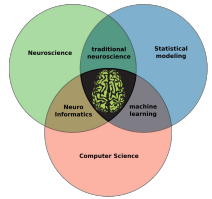
`Dockerfile` creation the same rules as linux based OS installation follows.

A learning process with a steep curve the build of `Docker images` is, but helping you and others tremendously, saving countless hours of installation and reproducibility related problems it will.

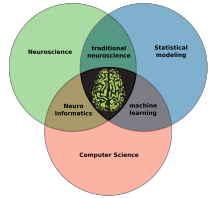
If possible, your `image` with the whole galaxy on `docker hub` should be shared.

`Naming` and `tagging` your `image` necessary is.

Some important points



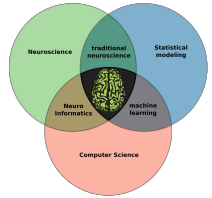
- these things take time, especially in the beginning
- no one expects you to create virtual environments or build containers all day after this 3 h lecture
- the examples within this lecture were picked to showcase different important aspects of virtualization
- building the right and fitting environment is most likely going to be a trial and error process, with some things going to be comparable (thus copy-past-able) while others will require more work and searching the internet
- just start using virtualization for different aspects of your scientific workflow and get to know the benefits along the way



The problem statement

We’re virtualization masters now, but this isn’t the end. A seat on the virtualization council awaits after proofing we’re also capable of advanced container concepts.

https://starwars.fandom.com/wiki/Jedi_Council?file=Councilrots.jpg



With something bright on the horizon, let's relax for 10 min

https://i.pinimg.com/600x315/e5/74/6a/e5746a117cbc6c0886090ac21536a7ab.jpg

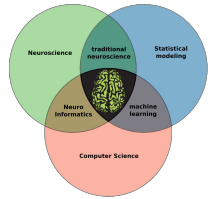
VIRTUALIZATION WARS

Docker 101 - The return of advanced concepts

Based on your accomplishments and bravery a virtualization council seat awaits. However, you must face one final challenge to defeat the non-existent-reproducibility empire for good

. . .

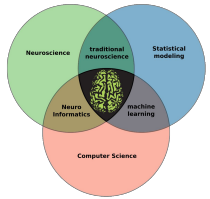
Docker 101 - The return of advanced concepts



- to secure our seats on the virtualization council, we need to learn two advanced concepts within the utilization of containers:

- ***automated builds*** through GitHub

- ***automated functionality*** of containers

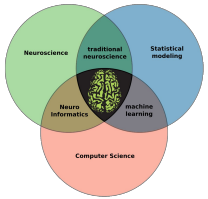


Docker 101 - The return of advanced concepts

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

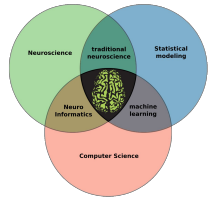
- the respective process is highly customizable and of course completely free
- all you need: a docker id and a *GitHub* account
- as usual, the setup is super easy and straightforward, as we just need to create a new *GitHub repository* in which we store our *Dockerfile*:
  - please create a new *Github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

Docker 101 - The return of advanced functionality



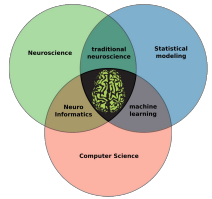
- now, on `docker hub`, go to your just uploaded `docker image` and within that, click on `builds`
- next, click on `Configure automated builds`
- we're now asked to indicate the `source repository` on `GitHub` and indicate some `build rules`, including `automated tests`, `branch` to `build` from, `tag` 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

Docker 101 - The return of advanced concepts



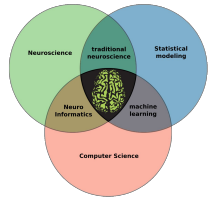
- that’s it, that’s all we need
- the cool thing is, that every `push` or `commit` to this `GitHub` repository triggers a new `build`, hence your `docker image` remains nicely up to date
- you'll get email notifications wrt the currently running processes, also letting you know that the `automated build`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

Docker 101 - The return of advanced concepts



- so far, we (hopefully) got to know how `docker` works, how `images` can be `pulled`, `run`, `build` and `pushed`
- however, we're missing one super cool part: the ***automatization of functionality***
- `docker images` can be setup and configured to do a certain ***task*** upon starting / running
- these ***automatized*** commands can nearly be everything you want, from simple to incredibly complex tasks
- for us, this could be ***automatically*** `run`ning a certain `script/function` using `python` upon starting the `container`

Docker 101 - The return of advanced concepts

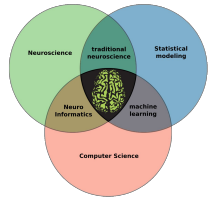


- so how can we achieve that using our `docker image`?
- one way would be to include the functionality we want in the `docker run command`:

```
docker run -it --rm \
           -v /Users/peerherholz/Desktop/galaxy/:/rebel_base:ro\
           millennium_falcon python
```

- while this is indeed starting python, we're opening a python terminal and also have no easy access to our fancy DTI analysis
- also it's not really the automated functionality we want ...

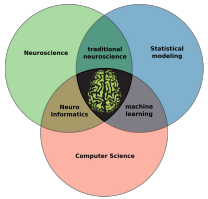
Docker 101 - The return of advanced concepts



- we can instead directly `configure` our `docker image` to `run python` as a default any time it's started
- to do so, we need to change the `build` of our `docker image`, adding `python` to the `entrypoint`
- this is done through `Neurodocker`'s `--add-to-entrypoint` argument:

```
--add-to-entrypoint "python"
```

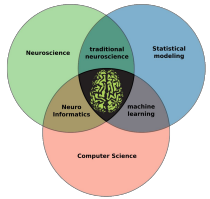
Docker 101 - The return of advanced concepts




```
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 \
             libgl1-mesa-glx libsm6 xvfb \
           --miniconda conda_install="python=3.7 dipy" \
             pip_install="fury xvfbwrapper" create_env="r2d2" \
             activate=true \
           --add-to-entrypoint "python" \
           > Dockerfile
```

- `rebuild` and `run` the `container` again

Docker 101 - The return of advanced concepts

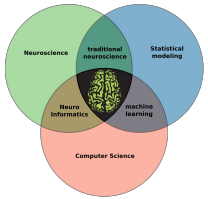


- after `rebuild`ing and `run`ning the `container` again, we `automatically` start in a `python` shell

```
docker run -it --rm \
           -v /Users/peerherholz/Desktop/galaxy/:/rebel_base:ro\
           millennium_falcon
```

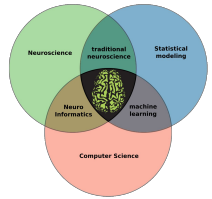
- everything we have to do now, is adding that a certain script should be run
- here, we can utilize the `force bridge`, aka `mapping`, again, as we can set a definite file name inside our `docker image` and map whatever script we want to `run` to that
- again: remember your training!

Docker 101 - The return of advanced concepts




```
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 \
             libgl1-mesa-glx libsm6 xvfb \
           --miniconda conda_install="python=3.7 dipy" \
             pip_install="fury xvfbwrapper" create_env="r2d2" \
             activate=true \
           --add-to-entrypoint "python /rebel_base/ewoks_attack.py" \
           > Dockerfile
```

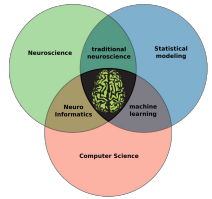
Docker 101 - The return of advanced concepts



- let's rebuild the `image`
- we now have to provide the script `where` and `how` our `docker image` and its `automated functionality` expects it, that is `/rebel_base/ewoks_attack.py`

```
docker run -it --rm \
-v /Users/peerherholz/Desktop/galaxy/fancy_DTI_analysis.py: \
  /rebel_base/ewoks_attack.py:ro\
  millennium_falcon
```

- we can see, that our fancy DTI analysis was `mapped` through the force bridge and `automatically run`



Remember you must:

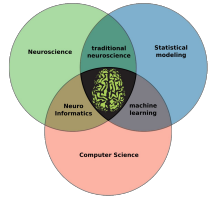
https://ffhmb.tqn.com/5Jqt2NdU5fitcwhv6FwNq26yVRg=/1500x1167/filters:fill%28auto,1%29/yoda-56a8f97a3df78cf772a263b4.jpg

`Dockerfile` creation the same rules as linux based OS installation follows.

A learning process with a steep curve the build of `Docker images` is, but helping you and others tremendously, saving countless hours of installation and reproducibility related problems it will.

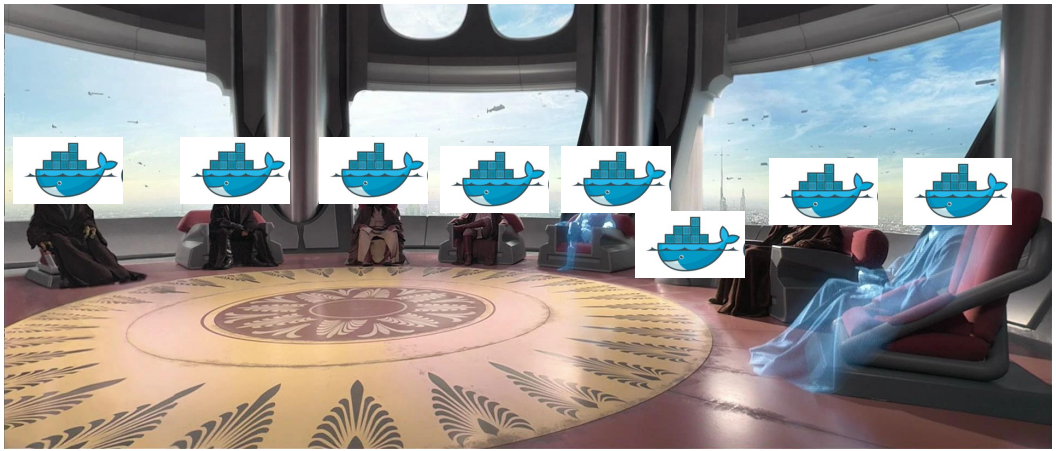
If possible, your `image` with the whole galaxy on `docker hub` should be shared.

`Naming` and `tagging` your `image` necessary is.

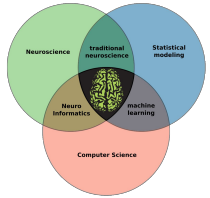


Docker 101 - The return of advanced concepts

- we’ve come a long way over the last couple of hours, from younglings over padawans to masters and now … we are part of the virtualization council

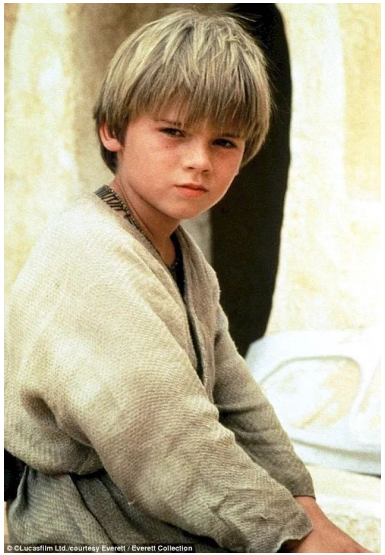


https://starwars.fandom.com/wiki/Jedi_Council?file=CouncilRots.jpg



Where to go from here?

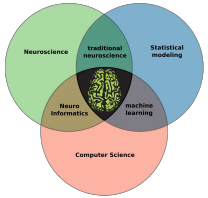
- no matter if you went on a selfish dark path of basically non-existent reproducibility and want to join the light side



https://img.srgcdn.com/e/w:1000/TG5IUThCQmVucVd2d3YzR2MvQjUuanBn.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



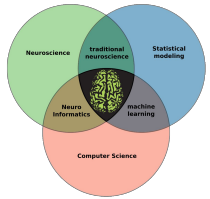
Where to go from here?

- 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

https://fthmb.tqn.com/5Jqt2NdU5fitcwhv6EwNp26yVRg=/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



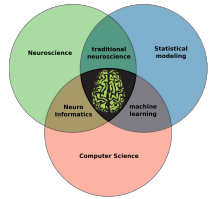
Where to go from here? - Notes on Singularity

- `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://
  ```



Where to go from here? - Notes on Singularity

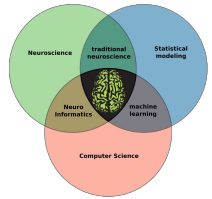
- builds are working similar as well:
- `build` from a singularity file (called recipe):

```
singularity build *image_name*.img Singularity
```

- again, `docker` support is no biggie:

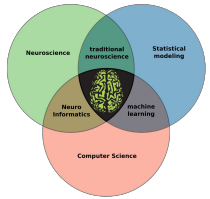
```
singularity build *image_name*.img docker://
```

Where to go from here? - Notes on Singularity



- `Neurodocker` once more has us covered:

```
docker run kaczmarj/neurodocker:0.4.3 \
           generate singularity \
           --base=ubuntu \
           --pkg-manager=apt \
           --install apt-utils bzip2 \
             ca-certificates curl git locales nano unzip \
             libgl1-mesa-glx libsm6 xvfb \
           --miniconda conda_install="python=3.7 dipy" \
             pip_install="fury xvfbwrapper" create_env="r2d2" \
             activate=true \
           --add-to-entrypoint "python /rebel_base/ewoks_attack.py" \
           > Dockerfile
```

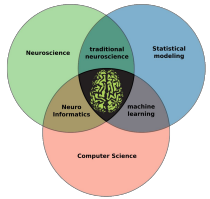


Where to go from here? - Notes on Singularity

- same same, but different: the syntax
- assuming we built a `Singularity` version of our `Docker image`, this is how we would `run` it:

```
singularity run \
          -B /Users/peerherholz/Desktop/galaxy/fancy_DTI_analysis.py: \
          /rebel_base/ewoks_attack.py:ro\
          millennium_falcon.simg
```

Where to go from here?



- we hope that this lecture/session provided you with the necessary details and information that will enable you to utilize virtualization sufficiently and efficiently within your research workflow

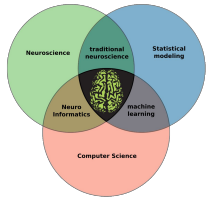
- virtualization is great for:

  - share code/scripts/functions/pipelines with colleagues and everyone else without dependency issues,
  - automize large parts of processing
  - rerunning analyses with identical or changed parameters

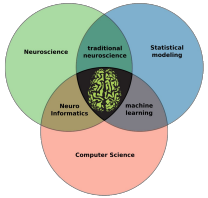
- virtualization is important for:

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

Where to go from here?



- further reading:
    - this course' Zotero group
    - A beginner friendly intro to VMs and Docker
    - Intro to Docker from Neurohackweek
    - Understanding Images
    - Singularity examples
    - one day docker workshop



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