Containers for use with Docker, Apptainer, and Compute4PUNCH

PUNCH Young Academy training

Nicola Malavasi - Marie Curie Fellow @ MPE - 16/04/2024

Tutorial outline

- Introduction
- What are containers?
- How to use pre-made containers
- How to create your own container
- Exercise: container creation
- Tips and tricks to optimize container creation
- Using containers in C4P

Speaker introduction

Who am I?

- Astronomer (i.e. not computer scientist).
- Ph.D. in 2016, PUNCH4NFDI postdoc for 2 years (2022-23).
- Did not know containers before starting to work for PUNCH. Now can confidently use containers in everyday scientific life (i.e. container use does not imply a steep learning curve).
- Using containers (in a non-expert way, for everyday science) is accessible to everyone, plenty of resources to move to advanced stage.

Why this tutorial?

Why learn about containers?

- Containers are a powerful tool, can be useful in many circumstances.
- Simplify life, streamline analysis (e.g. code installation, migration to new machines).
- More and more HPC infrastructures require the use of containers to some extent.
- Compute4PUNCH requires mandatory use of containers.
- There is no need to know everything about container use to start experiencing the benefits.
- Offer the perspective of a non-expert, science-centric, container user.

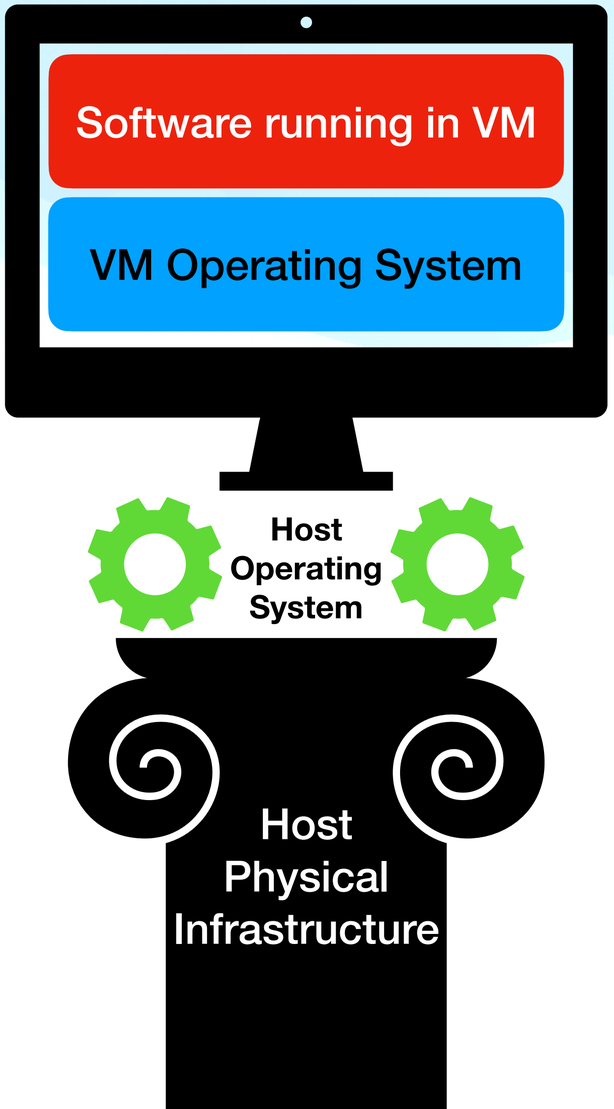
Part one

What are containers?
How to use pre-made containers.

Nicola Malavasi - Marie Curie Fellow @ MPE - 16/04/2024

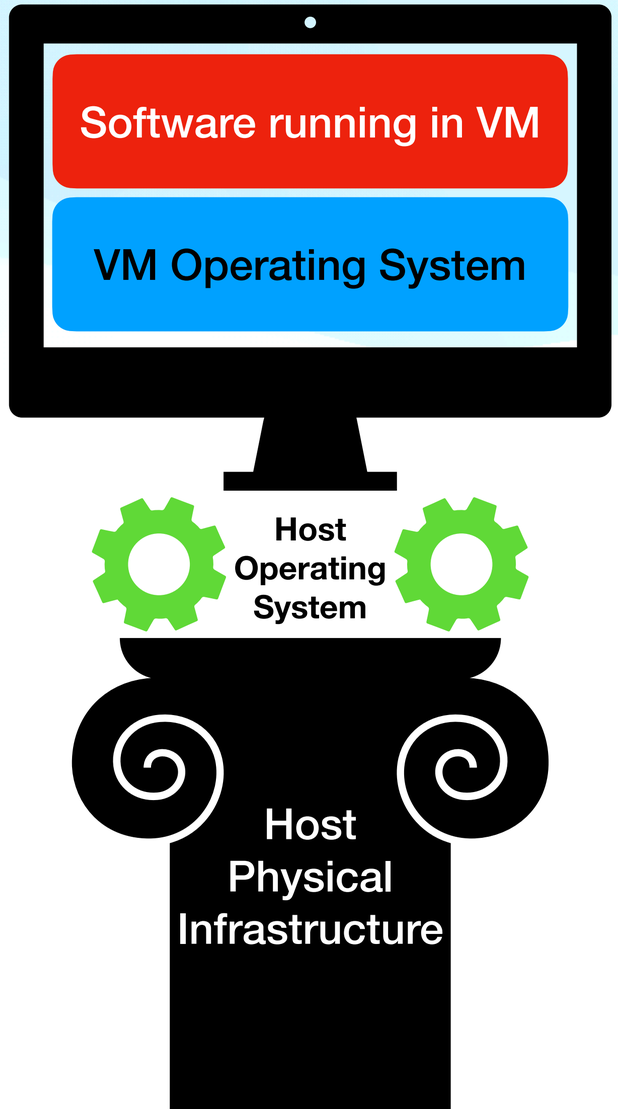
Virtual Machines

A VM is a reproduction of a physical computer running on a host system. Host system devolves resources to VM.
It runs on the host physical infrastructure, on top of the host OS. It contains a full OS, reproduces functionality of physical machine.



Virtual Machines

A VM is a reproduction of a physical computer running on a host system. Host system devolves resources to VM.
It runs on the host physical infrastructure, on top of the host OS. It contains a full OS, reproduces functionality of physical machine.

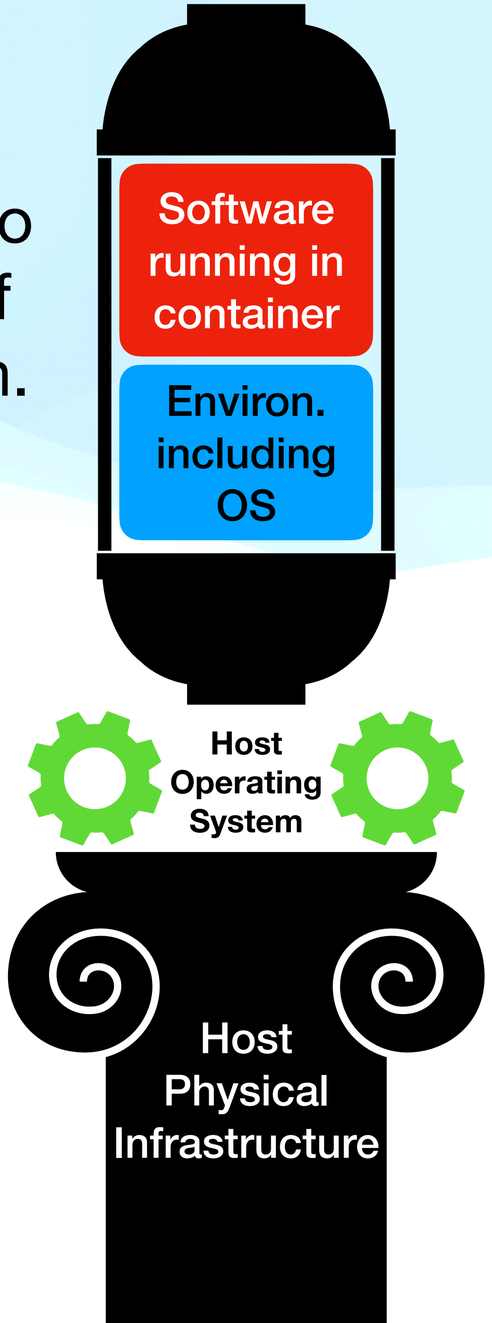


Powerful and user-friendly, yet cumbersome:

- Easy to use: “computer in computer”, graphical interface, familiar system.
- Full functionality of OS: easy to install code, run software as you would on normal computer.
- Heavy on resources: host must support full additional OS and provide CPU/RAM, VM filesystem is stored on host filesystem (can use significant memory).

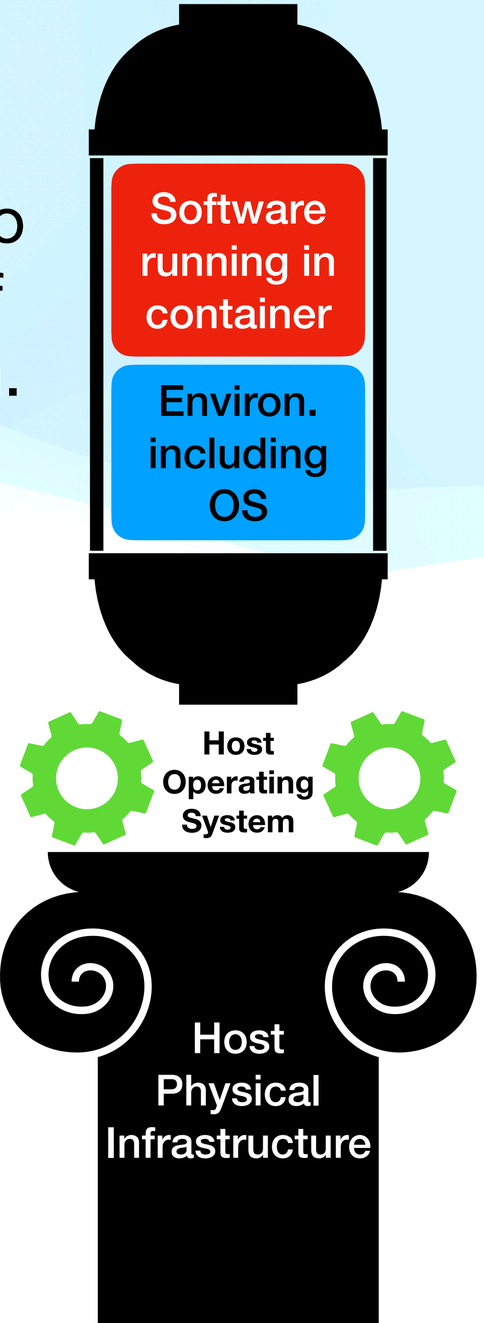
Containers

One can think of containers as similar to VM, but more lightweight. Do not reproduce entire physical machine, but stripped-down version of OS, runtime environment, and what necessary for software execution. More than a conda environment but less than a VM.



Containers

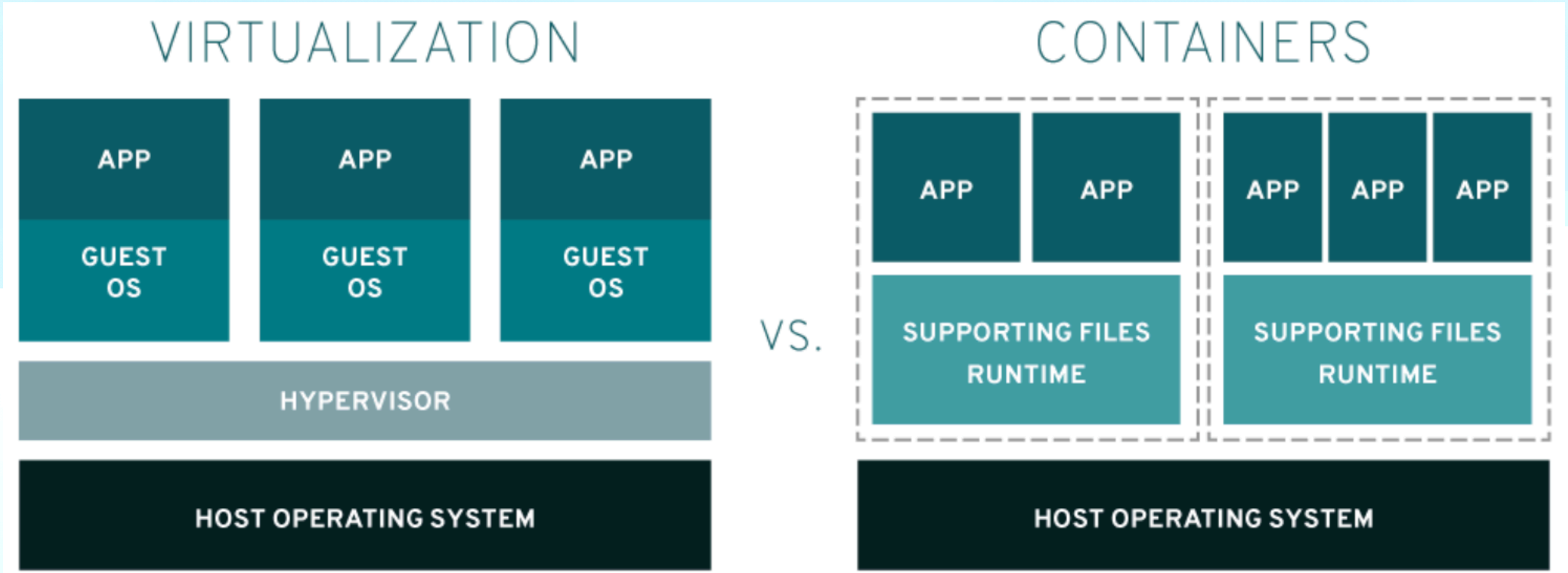
One can think of containers as similar to VM, but more lightweight. Do not reproduce entire physical machine, but stripped-down version of OS, runtime environment, and what necessary for software execution. More than a conda environment but less than a VM.



Lightweight, yet less immediate:

- Isolated execution of software, independent of host architecture.
- Lightweight: only what needed for software to run.
- Container creation and execution can be scripted, single commands can be executed inside the container without the need to boot up full VM.
- Less user-friendly: no graphic interface, new programming language needed to create one, special syntax for command execution.

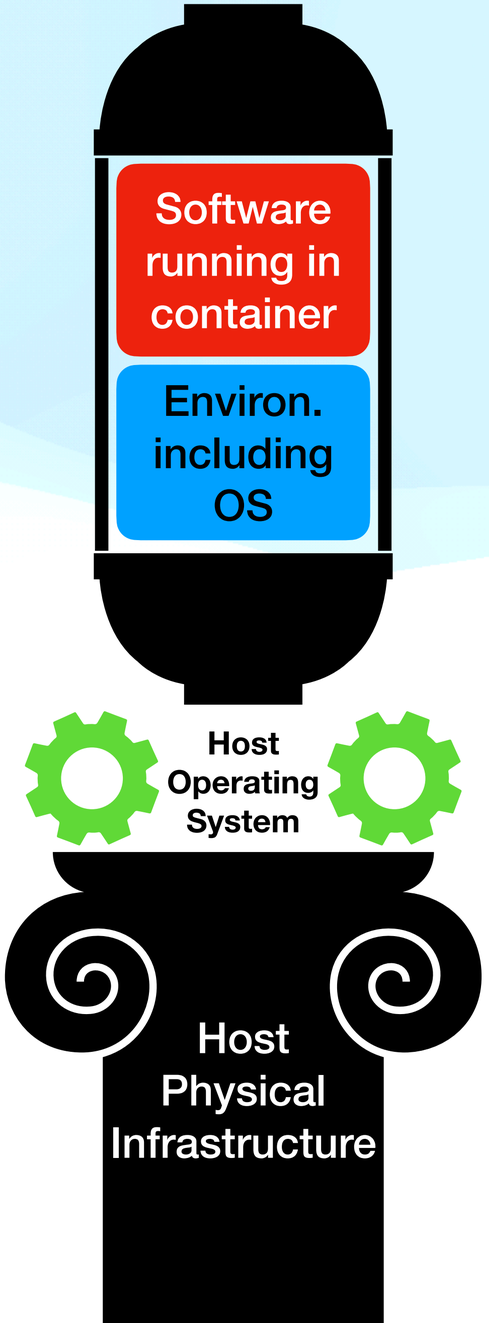
VM vs Containers



Source: https://www.redhat.com/en/topics/containers/containers-vs-vms

Containers

- Isolation: software is run without contact to or from the host OS or other containers.
- Efficiency: many lightweight containers can be run at the same time on same infrastructure.
- Portability: as long as container engine is present, same container can be run anywhere.

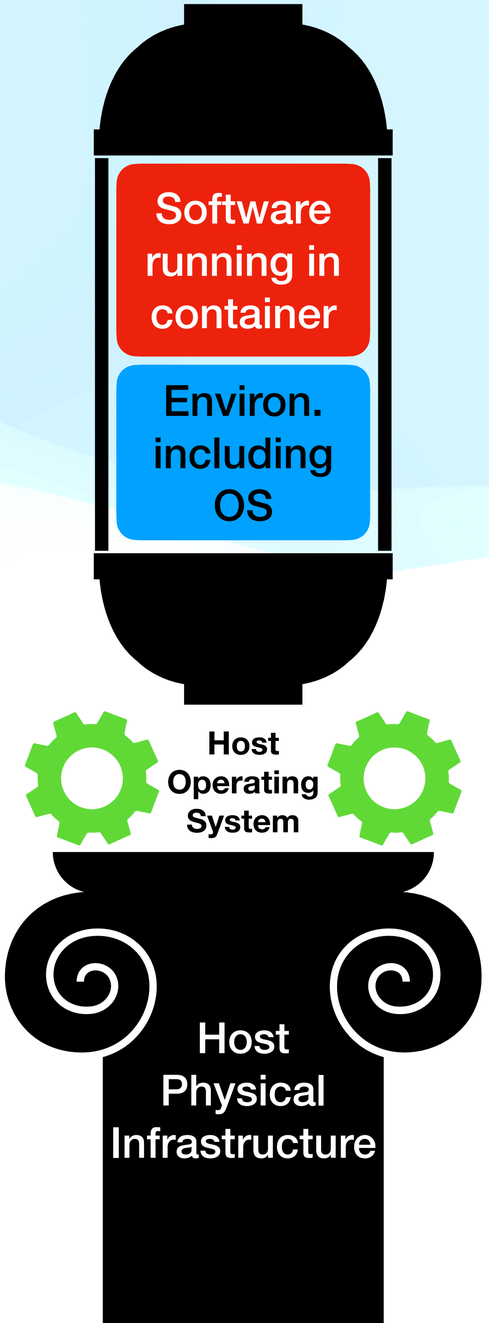


Containers

- Isolation: software is run without contact to or from the host OS or other containers.
- Efficiency: many lightweight containers can be run at the same time on same infrastructure.
- Portability: as long as container engine is present, same container can be run anywhere.

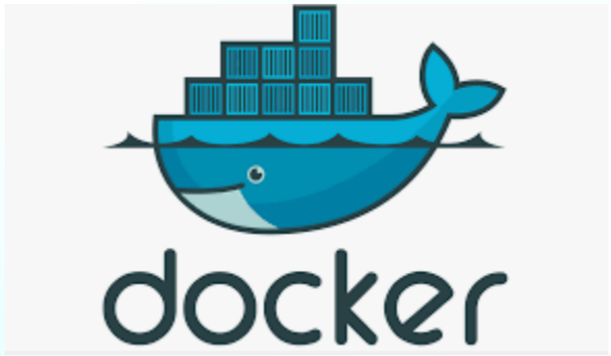
In everyday science:

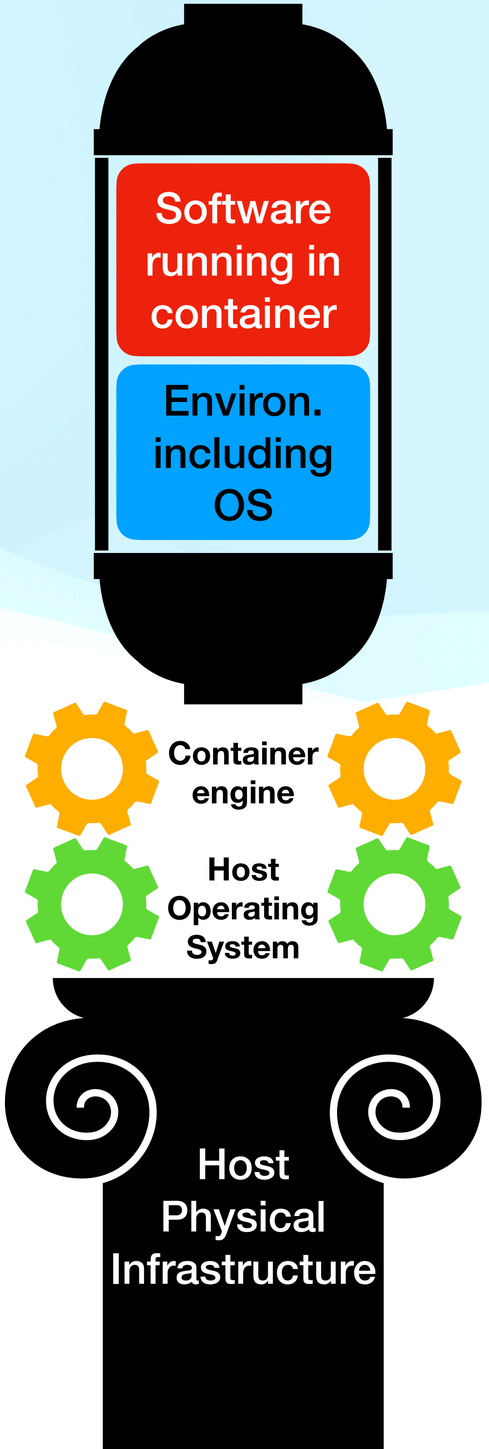
- Code installation: old code can be run in custom container, need only be installed once, can be run anywhere.
- Required/useful for HPC resources.



Container engine

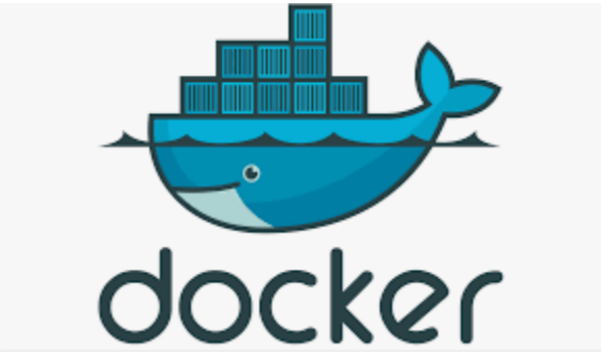
- A piece of software that manages containers, allows to create and operate them, and interface them with host OS.
- Different philosophies and approaches.
- Many exist, will mainly talk about two.





Docker

- Straightforward, well documented, easy to install and learn.
- User-friendly language format to create container creation scripts.
- Community/Docker Hub with pre-made containers.



First steps with containers

First steps with containers

Important terms

First steps with containers

Important terms

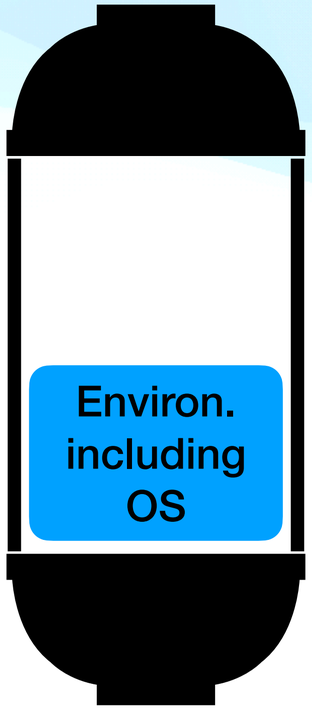
- Container: a process, running in an environment, isolated from the host and everything else. Example: a python instance executing a script.

First steps with containers

Important terms

- Container: a process, running in an environment, isolated from the host and everything else. Example: a python instance executing a script.

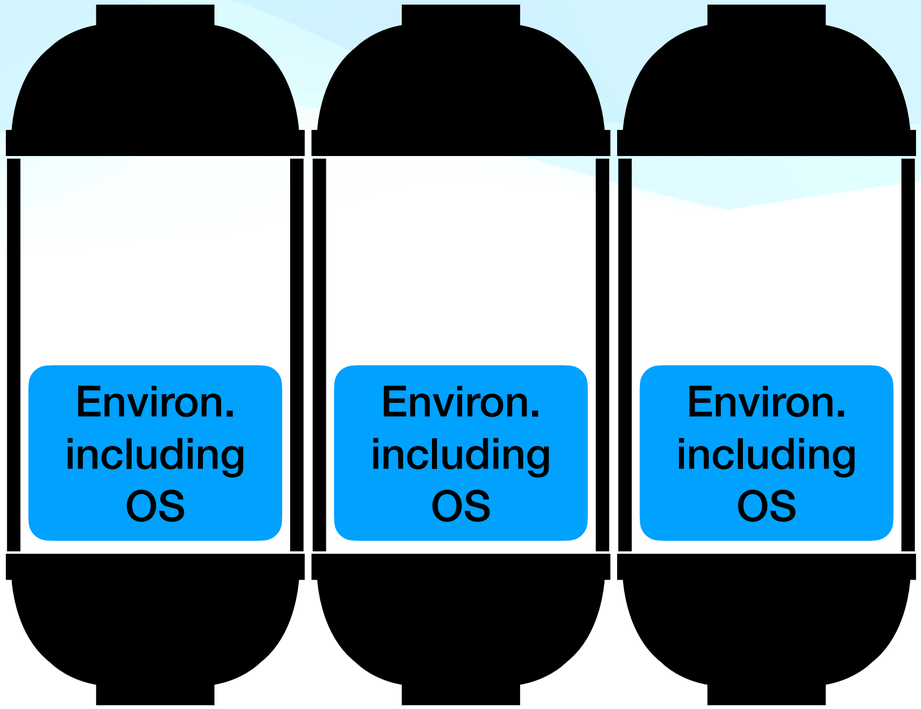
- Image: an immutable file supplying all the information (OS, environment, source, etc.). Example: a stripped-down ubuntu installation, python installed on top of that, with an active conda environment, and a folder with a python script.



First steps with containers

Important terms

- Container: a process, running in an environment, isolated from the host and everything else. Example: a python instance executing a script.
- Image: an immutable file supplying all the information (OS, environment, source, etc.). Example: a stripped-down ubuntu installation, python installed on top of that, with an active conda environment, and a folder with a python script.
- Repository: a storage place for images ready to be run as containers. In case of Docker: an internet page, called Docker Hub.



First steps with containers

First steps with containers

Actions:

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).
- Container run/exec: execute a command in a container.

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).
- Container run/exec: execute a command in a container.
- Container stop: stop a container.

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).
- Container run/exec: execute a command in a container.
- Container stop: stop a container.

Example: Python2.7

Pulling an image

We want to have access to a python 2.7 installation (e.g. because our code needs it).
First: let's check that python 2.7 is not already installed.

```
nmalavasi@ga-lt7982 ~ % python2.7
zsh: command not found: python2.7
nmalavasi@ga-lt7982 ~ % python2
zsh: command not found: python2
nmalavasi@ga-lt7982 ~ % python3
Python 3.9.6 (default, Feb  3 2024, 15:58:27)
[Clang 15.0.0 (clang-1500.3.9.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Example: Python2.7

Pulling an image

Second: let’s download the necessary image. The command is docker image pull.

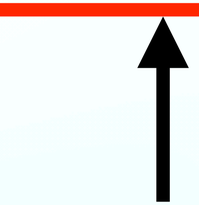
```
docker pull python:2.7.18-stretch
```

Example: Python2.7

Pulling an image

Second: let’s download the necessary image. The command is docker image pull.

```
docker pull python:2.7.18-stretch
```

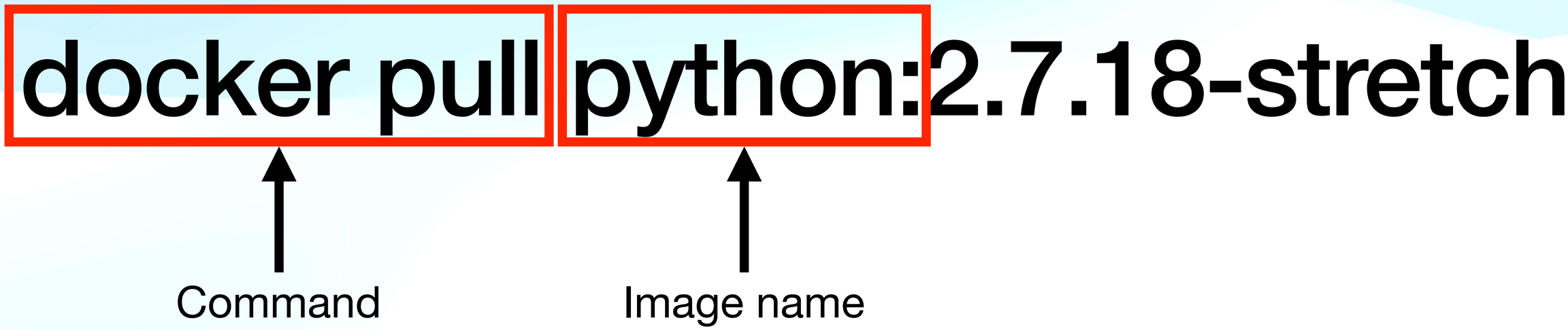


Command

Example: Python2.7

Pulling an image

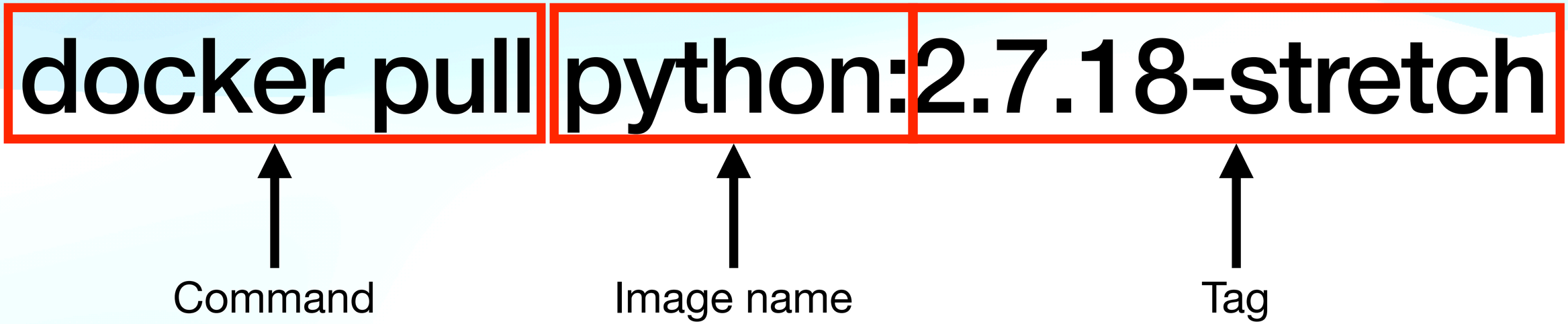
Second: let’s download the necessary image. The command is docker image pull.



Example: Python2.7

Pulling an image

Second: let’s download the necessary image. The command is docker image pull.



Example: Python2.7

Pulling an image

Second: let's download the necessary image. The command is docker image pull.

```
nmalavasi@ga-lt7982 ~ % docker pull python:2.7.18-stretch
2.7.18-stretch: Pulling from library/python
65d54b492d59: Pull complete
7be35cdee43e: Pull complete
b803ac2c0e69: Pull complete
f56b7aa1f85c: Pull complete
6b89e9488b4b: Pull complete
a167bb72b194: Pull complete
3123ab59854f: Pull complete
722c4b126156: Pull complete
29a2bbd5f8f0: Pull complete
Digest: sha256:548e680020444b0f6ddc4c7b0c24964d1af5f47cd2e2b3b44d742852b8b09cfc
Status: Downloaded newer image for python:2.7.18-stretch
docker.io/library/python:2.7.18-stretch
nmalavasi@ga-lt7982 ~ %
```

Example: Ubuntu

Pulling an image

Another example: downloading an ubuntu image.

```
docker pull ubuntu:latest
```

Example: Ubuntu

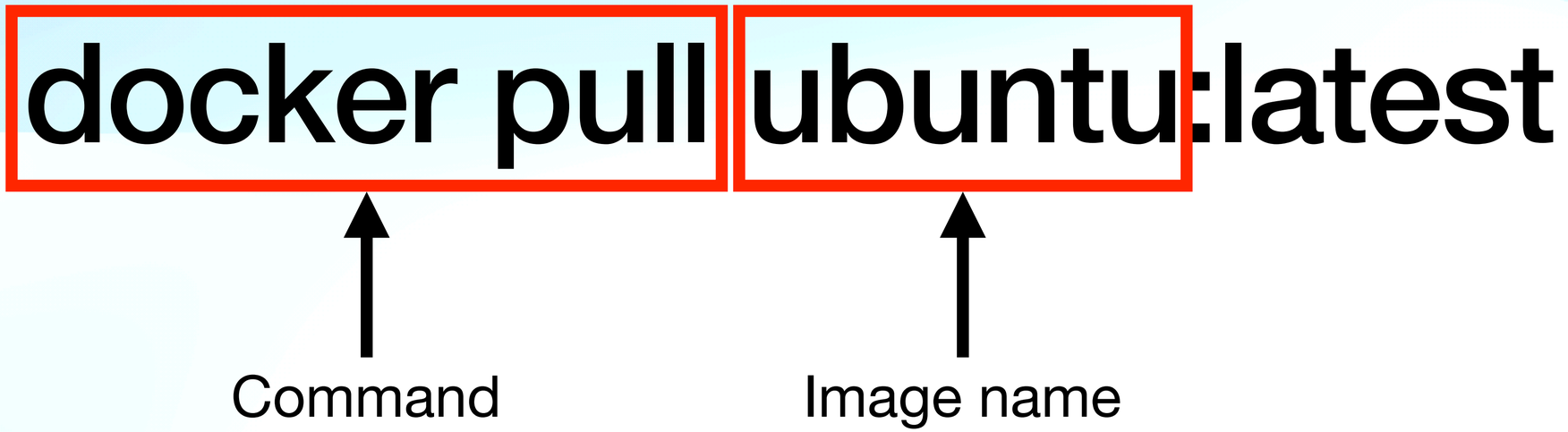
Pulling an image

Another example: downloading an ubuntu image.

Example: Ubuntu

Pulling an image

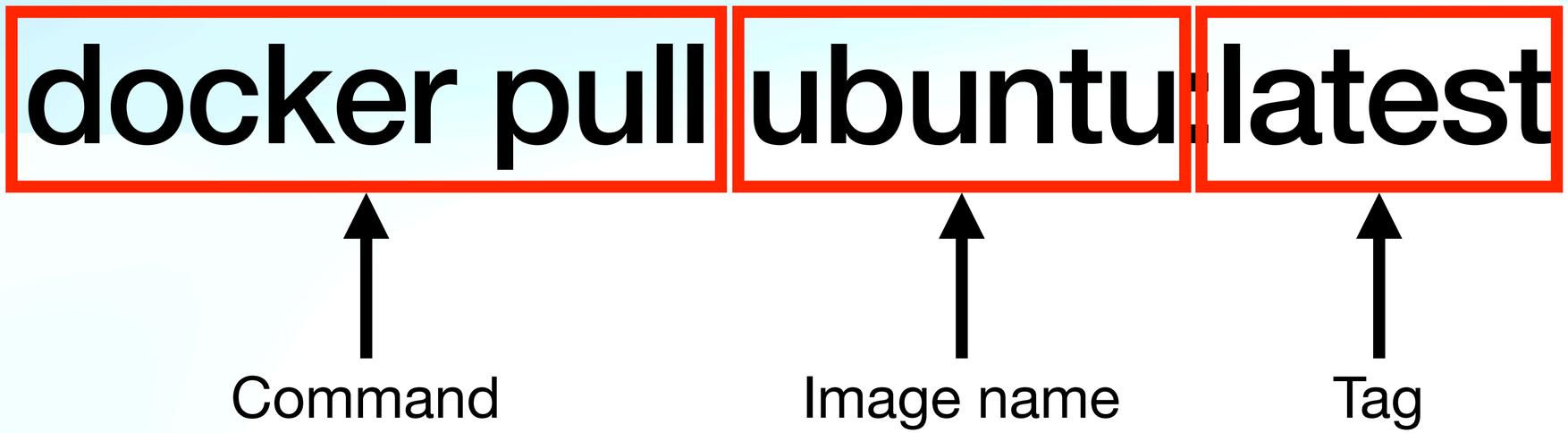
Another example: downloading an ubuntu image.



Example: Ubuntu

Pulling an image

Another example: downloading an ubuntu image.



Example: Ubuntu

Pulling an image

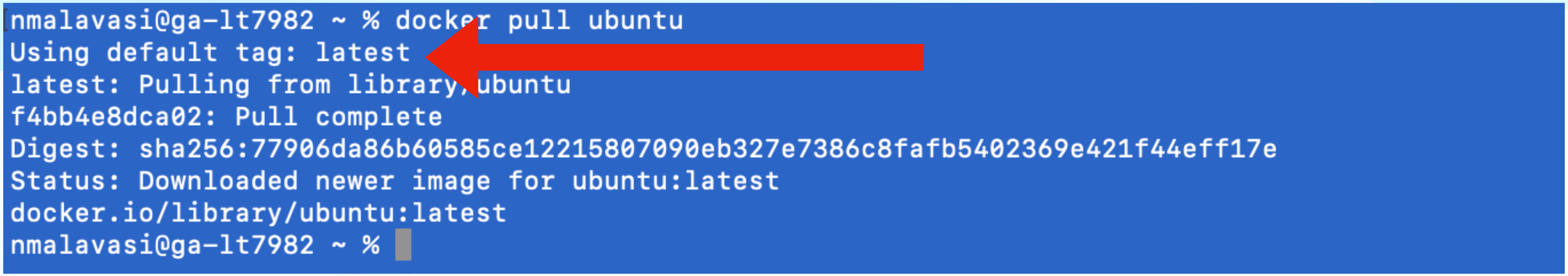
Another example: downloading an ubuntu image.

```
nmalavasi@ga-lt7982 ~ % docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
f4bb4e8dca02: Pull complete
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
nmalavasi@ga-lt7982 ~ %
```

Example: Ubuntu

Pulling an image

Another example: downloading an ubuntu image.



First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).
- Container run/exec: execute a command in a container.
- Container stop: stop a container.

Example: Python2.7

Running a container

We go back to our Python2.7 image. Let’s start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.

```
docker run -it --name python27_c python:2.7.18-stretch
```

Example: Python2.7

Running a container

We go back to our Python2.7 image. Let’s start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.

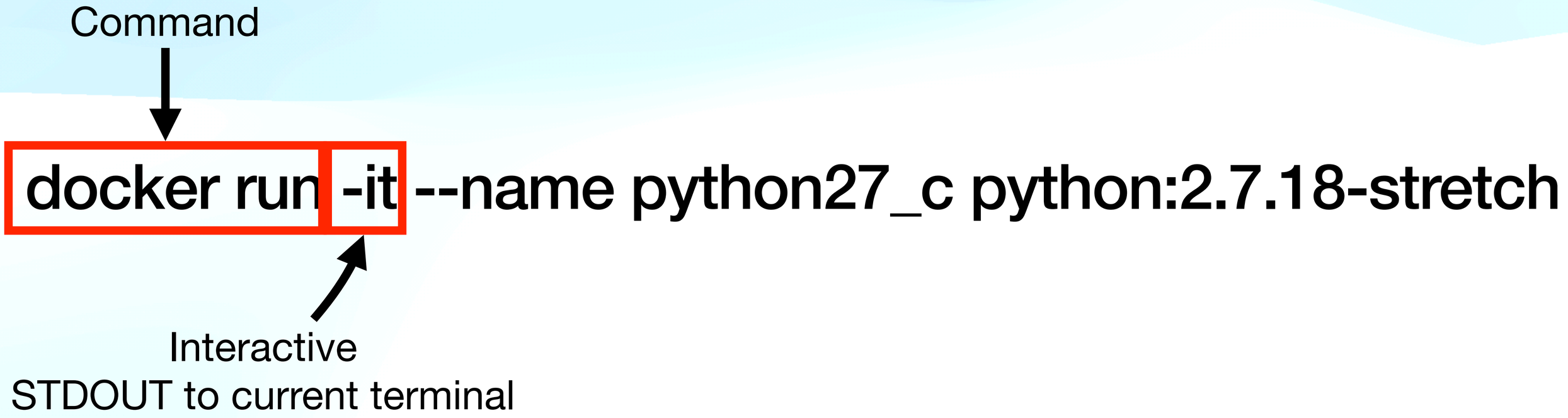
Command

```
docker run -it --name python27_c python:2.7.18-stretch
```

Example: Python2.7

Running a container

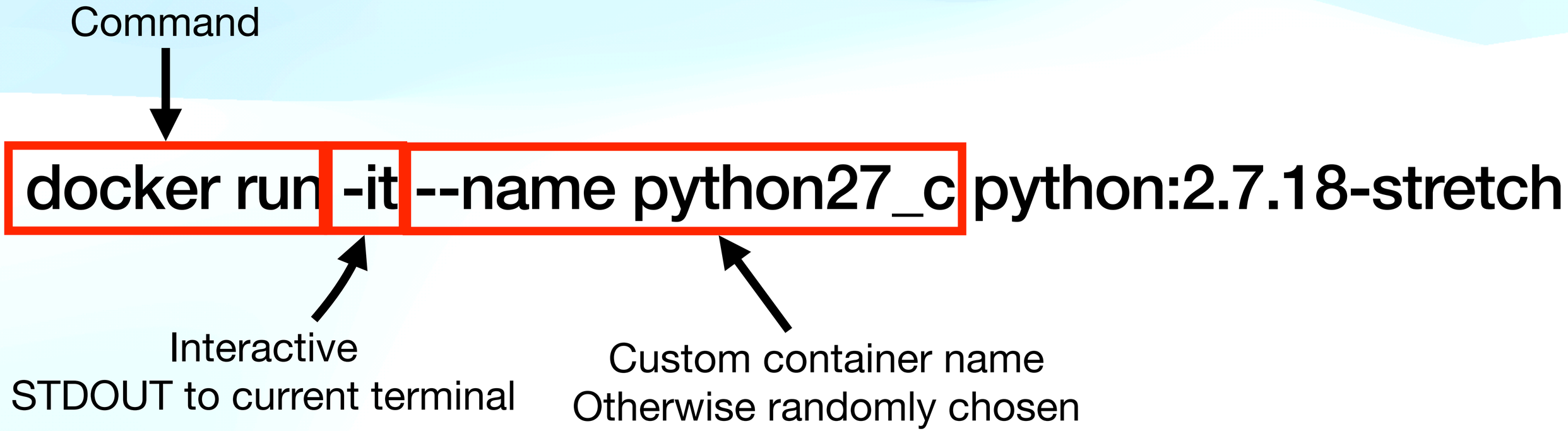
We go back to our Python2.7 image. Let's start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.



Example: Python2.7

Running a container

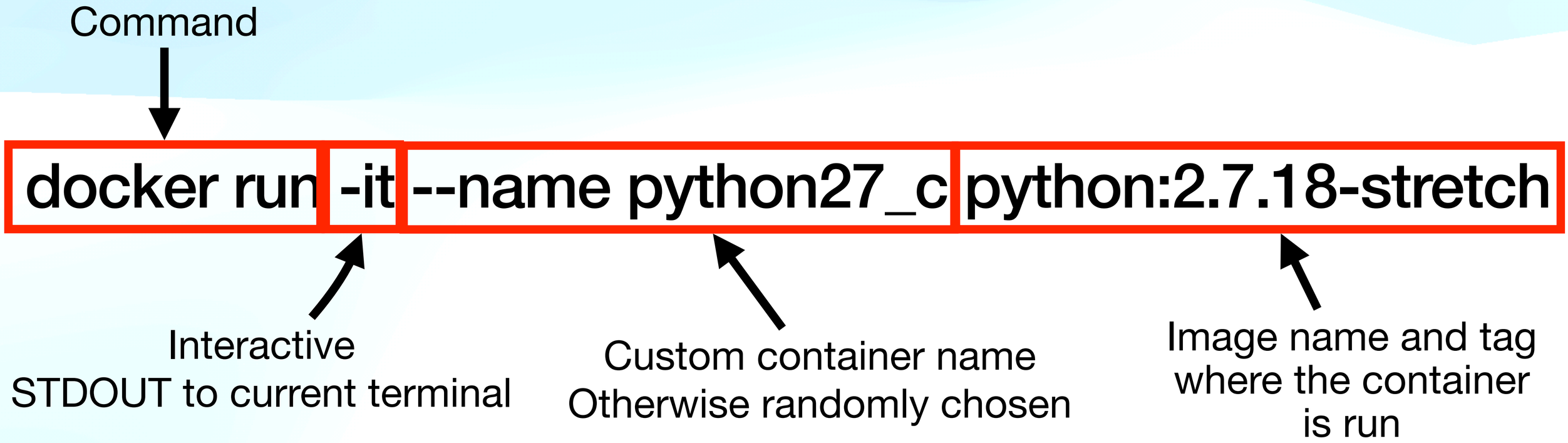
We go back to our Python2.7 image. Let's start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.



Example: Python2.7

Running a container

We go back to our Python2.7 image. Let's start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.



Example: Python2.7

Running a container

We go back to our Python2.7 image. Let's start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.

```
nmalavasi@ga-lt7982 ~ % docker run -it --name python27_c python:2.7.18-stretch
Python 2.7.18 (default, Apr 21 2020, 10:02:18)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Example: Python2.7

Running a container

We go back to our Python2.7 image. Let's start a container (i.e. a self-contained process) using the information saved in the image.
The command is docker run.

```
nmalavasi@ga-lt7982 ~ % docker run -it --name python27_c python:2.7.18-stretch
Python 2.7.18 (default, Apr 21 2020, 10:02:18)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print('a')
a
>>> print 'a'
a
>>>
```

Example: Ubuntu

Running a container

In the case of the Ubuntu image, it is a good way to show container isolation.

```
nmalavasi@ga-lt7982 ~ % docker run -it --name ubuntu_c ubuntu
root@9d341f9b38cc:/# pwd
/
root@9d341f9b38cc:/# whoami
root
root@9d341f9b38cc:/#
```

First steps with containers

Actions:

- Image pull: download a pre-made image from a registry.
- Image build: create image from scratch.
- Container start: start a container from an image (i.e. a contained process in an environment set by an image).
- Container run/exec: execute a command in a container.
- Container stop: stop a container.

Example: Ubuntu

Executing a command in a container

We perform an ls in a non-interactive way. The command is docker exec.

```
docker exec -it ubuntu_c ls
```

Example: Ubuntu

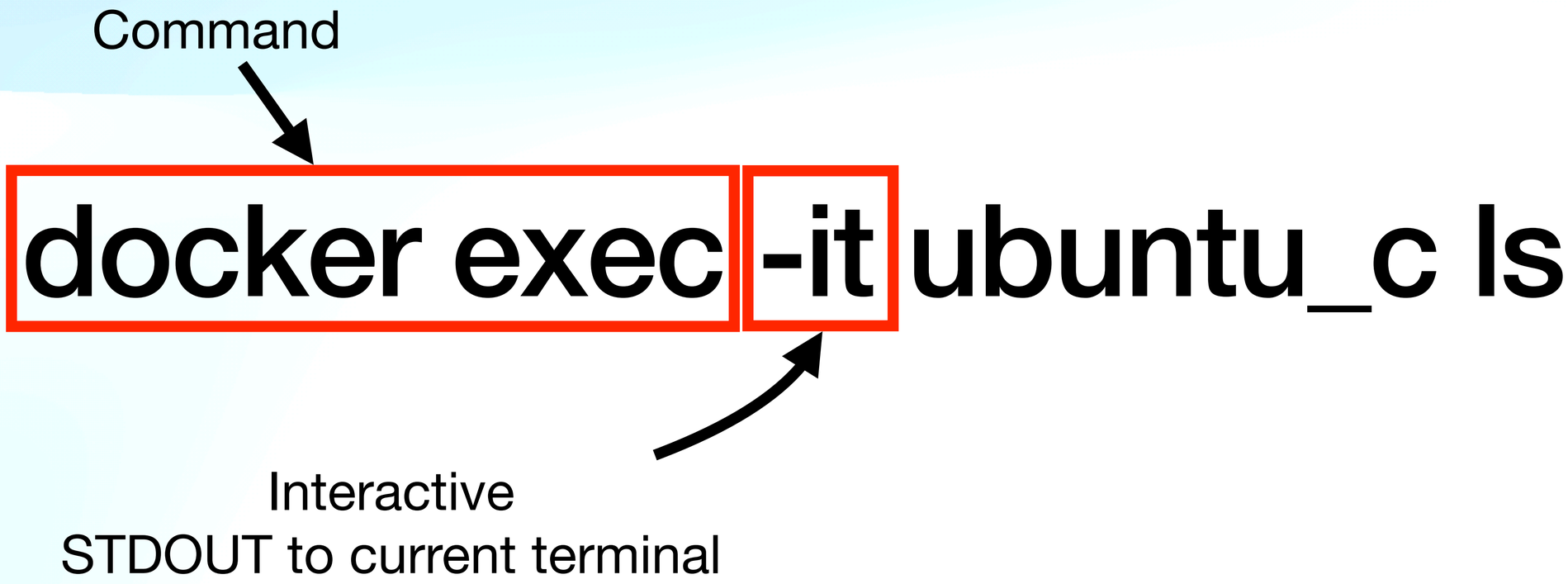
Executing a command in a container

We perform an ls in a non-interactive way. The command is docker exec.

Example: Ubuntu

Executing a command in a container

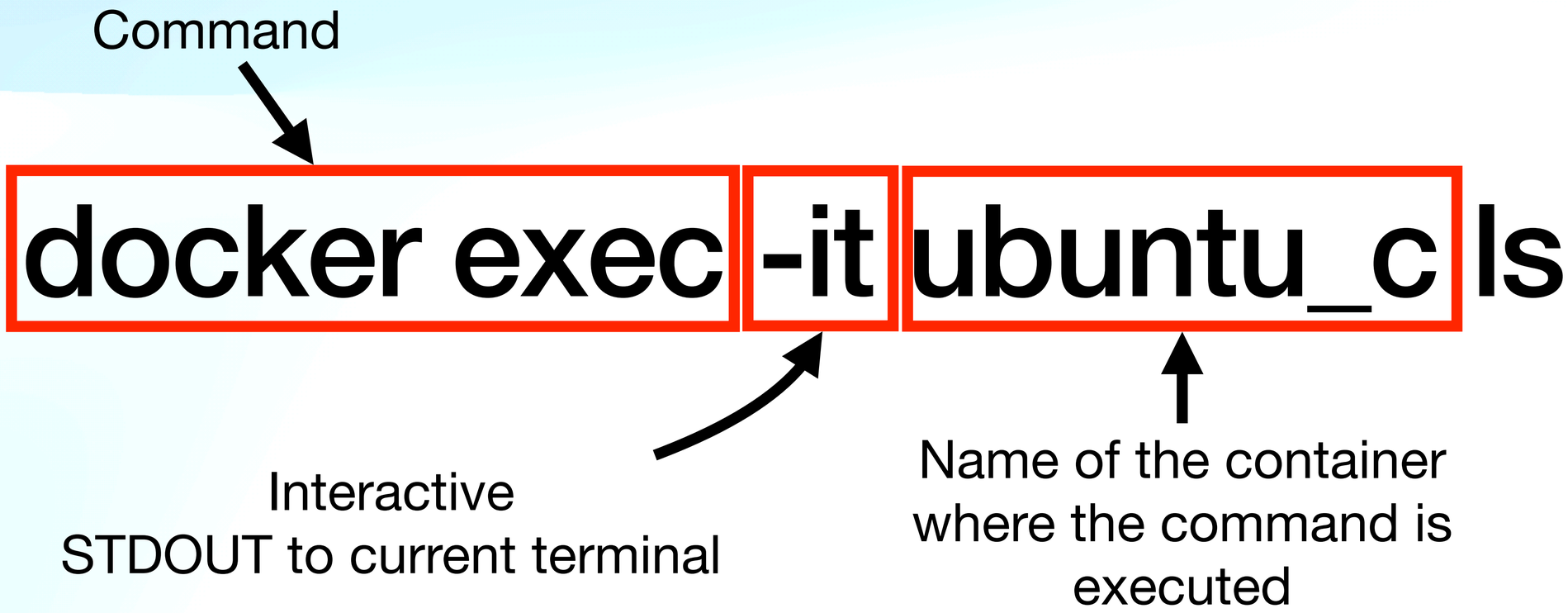
We perform an ls in a non-interactive way. The command is docker exec.



Example: Ubuntu

Executing a command in a container

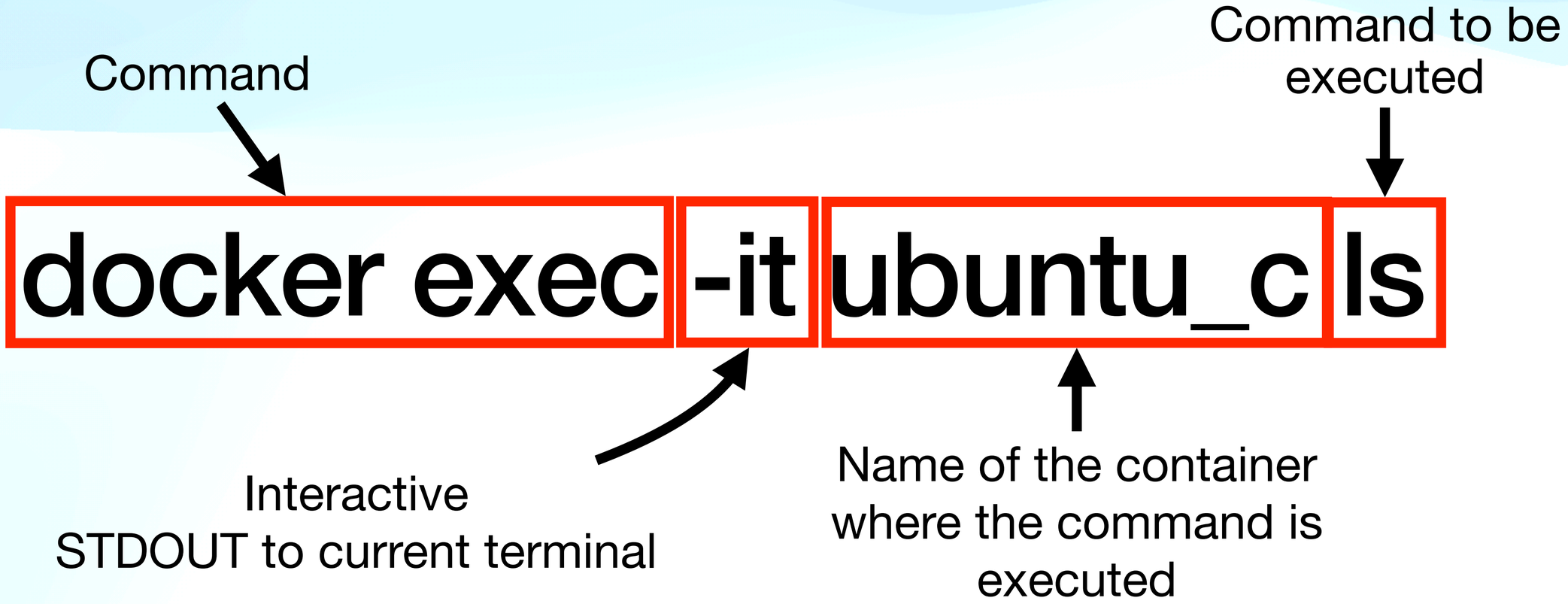
We perform an ls in a non-interactive way. The command is docker exec.



Example: Ubuntu

Executing a command in a container

We perform an ls in a non-interactive way. The command is docker exec.



Example: Ubuntu

Executing a command in a container

We perform an ls in a non-interactive way.

```
nmalavasi@ga-lt7982 ~ % docker exec -it ubuntu_c ls
bin   dev  home  media  opt   root  sbin  sys  usr
boot  etc  lib   mnt    proc  run   srv   tmp  var
nmalavasi@ga-lt7982 ~ %
```

Access to files outside of the container

Mounting volumes

Processes run in containers are isolated. There is no exchange with the host and the container has its own user, filesystem etc.

How to have access to files outside of container? E.g. script is developed locally and only run in container.

Solution: volume (bind) mount. Similar to plugging USB stick in container.

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

```
docker run -it --name my_c --mount type=bind,
source=local_path, target=path_in_container image_name:tag
```

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

Same as before

docker run -it --name my_c --mount type=bind, source=local_path, target=path_in_container image_name:tag

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

Same as before

```
docker run -it --name my_c --mount type=bind,
source=local_path, target=path_in_container image_name:tag
```

Same as before

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

```
docker run -it --name my_c --mount type=bind,
source=local_path, target=path_in_container image_name:tag
```

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

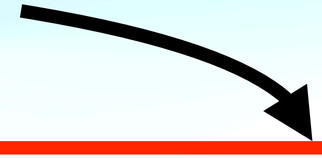
--mount type=bind, source=local_path, target=path_in_container

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

Type of mount: bind is not the only one



```
--mount type=bind, source=local_path,
target=path_in_container
```

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

Type of mount: bind is not the only one

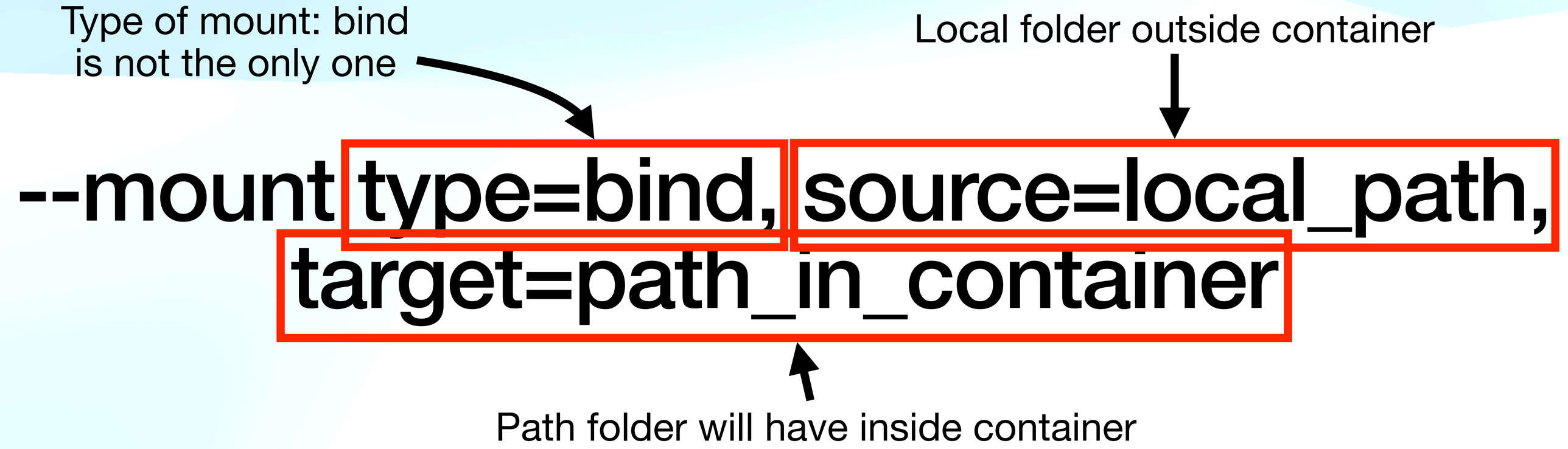
Local folder outside container

```
--mount type=bind, source=local_path,
target=path_in_container
```

Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.



Example: Python2.7

Mounting a volume

Let’s go back to the Python2.7 example. We have to recreate the container with a bind mount to a local folder on the laptop.

```
nmalavasi@ga-lt7982 ~ % docker run -it  --name py_c --mount type=bind,source=/Users/nmalavasi/Desktop/PUNCH_useful/Presentations/PYA_tutorial_container/tutorial_material,target=/place_
of_script python:2.7.18-stretch
Python 2.7.18 (default, Apr 21 2020, 10:02:18)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('ls')
bin  boot  dev  etc  home  lib  media  mnt  opt  place_of_script  proc  root  run  sbin  srv  sys  tmp  usr  var
0
>>> os.system('ls /place_of_script')
example_script_simple.py
0
>>>
```

Example: Python2.7

Mounting a volume

The container can see our script. Let's give it the command to install matplotlib and run our script.

```
nmalavasi@ga-lt7982 ~ % docker start py_c
py_c
nmalavasi@ga-lt7982 ~ % docker exec -it py_c pip install matplotlib
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. A future version of pip will drop support fo
r Python 2.7. More details about Python 2 support in pip, can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support
Collecting matplotlib
  Downloading matplotlib-2.2.5.tar.gz (36.7 MB)
     |████████████████████████████████| 36.7 MB 6.1 MB/s
Processing /root/.cache/pip/wheels/fe/49/d9/dbd8037d1f7b1672ebcdba0c75f1b7000d9e888a33af294c35/numpy-1.16.6-cp27-cp27mu-linux_aarch64.whl
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
     |████████████████████████████████| 67 kB 17.1 MB/s
Collecting python-dateutil>=2.1
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     |████████████████████████████████| 229 kB 59.6 MB/s
Collecting pytz
  Downloading pytz-2024.1-py2.py3-none-any.whl (505 kB)
     |████████████████████████████████| 505 kB 43.0 MB/s
Requirement already satisfied: six>=1.10 in /usr/local/lib/python2.7/site-packages (from matplotlib) (1.14.0)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.1.0.tar.gz (30 kB)
Collecting backports.functools_lru_cache
  Downloading backports.functools_lru_cache-1.6.6-py2.py3-none-any.whl (5.9 kB)
Collecting subprocess32
  Downloading subprocess32-3.5.4.tar.gz (97 kB)
     |████████████████████████████████| 97 kB 23.8 MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python2.7/site-packages (from kiwisolver>=1.0.1->matplotlib) (44.1.0)
Building wheels for collected packages: matplotlib, kiwisolver, subprocess32
  Building wheel for matplotlib (setup.py) ... done
  Created wheel for matplotlib: filename=matplotlib-2.2.5-cp27-cp27mu-linux_aarch64.whl size=10693452 sha256=0dfb1a88cdaa4196877e46ddb122dfdea95241b497d9a06b7f74e78e2d76328c
  Stored in directory: /root/.cache/pip/wheels/20/af/9f/bca6e52dc48188e4068752906758846bd9dfeac4e480d24218
  Building wheel for kiwisolver (setup.py) ... done
  Created wheel for kiwisolver: filename=kiwisolver-1.1.0-cp27-cp27mu-linux_aarch64.whl size=959361 sha256=2c48a01721b1d7d107ddaa2a45fc68b5f0727dff14f87f5b58eb6670ebea11b5
  Stored in directory: /root/.cache/pip/wheels/80/b4/b8/99e51e2ccaadfdf828c133ca0e624cbb737a06ad343d112978
  Building wheel for subprocess32 (setup.py) ... done
  Created wheel for subprocess32: filename=subprocess32-3.5.4-cp27-cp27mu-linux_aarch64.whl size=50848 sha256=3c591ad7c85a91cc2a9d0ecd6eea30621694302c89ce13a32204d5829b238f3f
  Stored in directory: /root/.cache/pip/wheels/e3/c7/6a/434fc8f2936acc4964ded8478435a8ef7c69eb41df7007a49f
Successfully built matplotlib kiwisolver subprocess32
Installing collected packages: numpy, cycler, pyparsing, python-dateutil, pytz, kiwisolver, backports.functools-lru-cache, subprocess32, matplotlib
Successfully installed backports.functools-lru-cache-1.6.6 cycler-0.10.0 kiwisolver-1.1.0 matplotlib-2.2.5 numpy-1.16.6 pyparsing-2.4.7 python-dateutil-2.9.0.post0 pytz-2024.1 subproce
ss32-3.5.4
WARNING: You are using pip version 20.0.2; however, version 20.3.4 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
nmalavasi@ga-lt7982 ~ %
```

Example: Python2.7

Mounting a volume

```
nmalavasi@ga-lt7982 ~ % docker exec -it py_c python /place_of_script/example_script_simple.py
2.2.5
Hello world
nmalavasi@ga-lt7982 ~ %
```

Conclusions

- Very easy to switch between containers (e.g. ubuntu/Python2.7).
- We can have our installation ready in minutes without too much trouble.
- Powerful way to have our code working.