Working with Containers and Pipelines

Please open this in your browser to follow along:

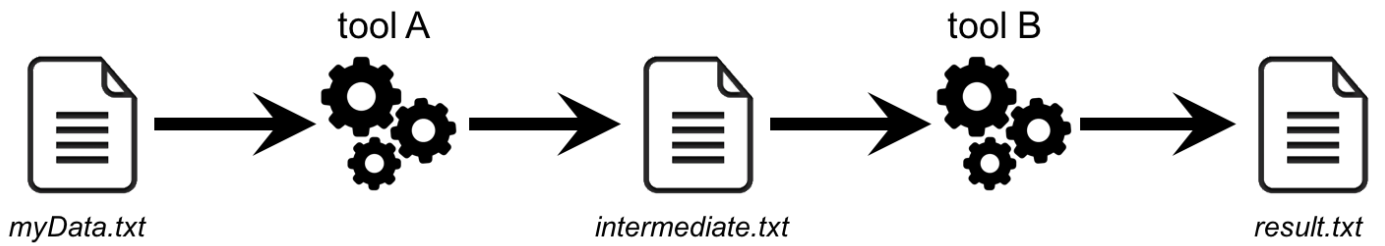
bit.ly/pipeline_container_tutorial

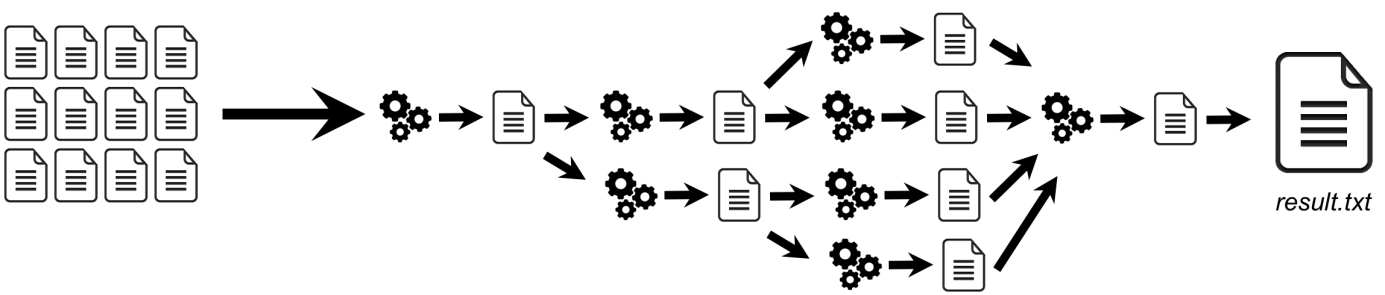
Agenda

- **Introduction** about pipelines
  - What is a **pipeline**?
  - A basic example of **workflow manager**
- **Hands-on**
  - Building a **self-contained** pipeline `(ex01)`
  - Running a pipeline on **different data** `(ex02)`
  - **Modular** containers for pipelines `(ex03a` & `ex03b)`
  - Running **out of the box** `(ex04)`
  - **Multiple** containers `(ex05)`
  - **Validating** a pipeline and its container `(ex06)`
  - Testing it **all** `(ex07)`
  - A different **base image** `(ex08)`
  - Singularity containers via a **Dockerfile** `(ex09` & `ex10)`
- A real use case: **MelArray**
  - The *MelArray* **pipeline**
  - **Snakemake** and **Singularity**
  - The *MelArray* **container**
  - A note about container sizes
- Docker, Singularity, Shifter

What is a pipeline ?

Pipeline: chain of data-processing software





A basic example of **workflow manager**

Pipeline: chain of data-processing software

Workflow manager: software to create **reproducible** and **scalable** data analysis *pipeline* (or *workflow*)

Snakemake
a Python workflow manager


```
rule all:
    input:
        '/data/output2.dat'
    shell:
        'echo "Pipeline \'ex1\' complete.'

rule step1:
    input:
        '/data/input.dat'
    output:
        '/data/output1.dat'
    shell:
        'step1.py {input} {output}'

rule step2:
    input:
        '/data/output1.dat'
    output:
        '/data/output2.dat'
    shell:
        'step2.py {input} {output}'
```

ex01

Building a **self-contained** pipeline

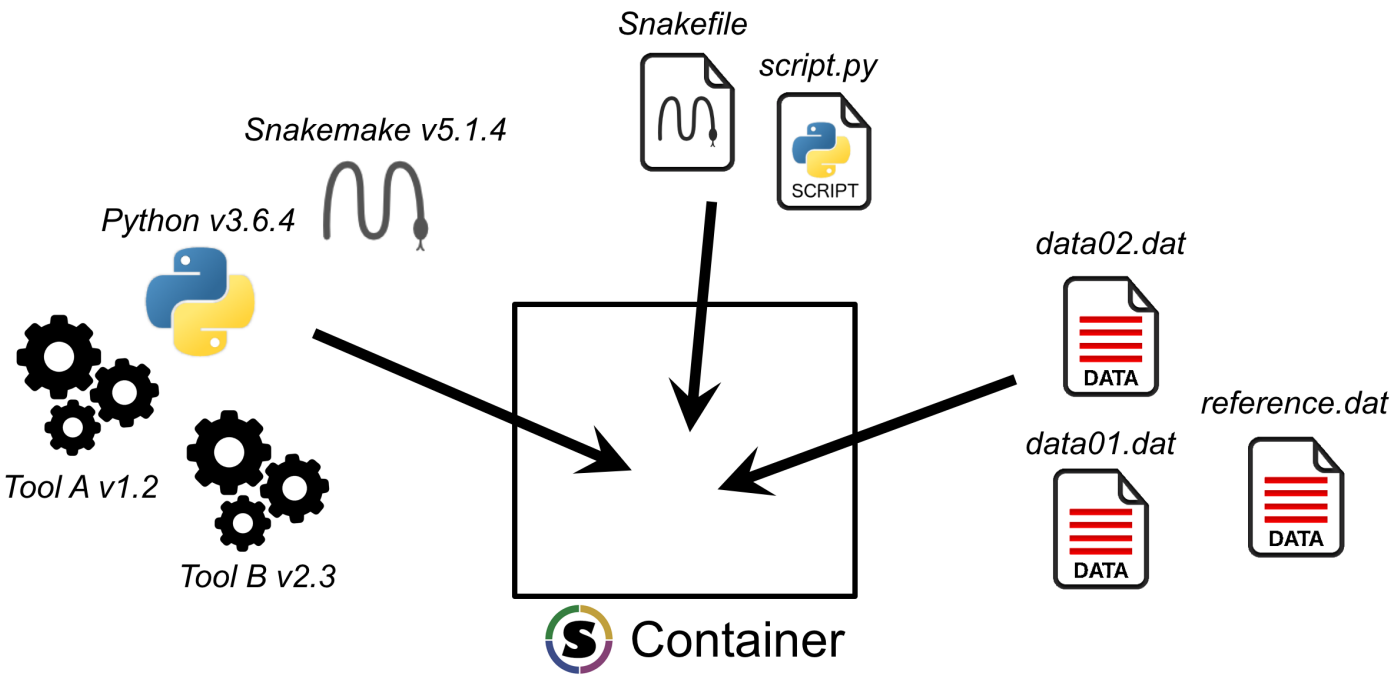
The most simple way of *packaging* a pipeline into a reproducible container is to **put everything inside** one (big) container:

- **Data**: all the data needed to re-create the results
- **Dependencies**: all the code / scripts, tools, libraries to run the pipeline
- **Pipeline definition file**: the definition file(s) describing how to run the pipeline

ex01

Building a **self-contained** pipeline

The most simple way of *packaging* a pipeline into a reproducible container is to **put everything inside** one (big) container:



ex01

Building a **self-contained** pipeline

Exercise:

- Go to `exercises/ex01`
- Have a look at the **data**: `input.dat`
- Have a look at the **pipeline definition**: `Snakefile`
- Have a look at the **scripts**: `step1.py` & `step2.py`
- Have a look at the `Singularity` **recipe file**

ex01

Building a **self-contained** pipeline

- **`input.dat`**: a simple text file containing 1 number
- **`Snakefile`**: a simple 2-step pipeline going from **`input.dat`** to **`output1.dat`** to **`output2.dat`**
- **`step1.py`** & **`step2.py`**: scripts reading in the number, doing simple math on it, and writing the result out to a file
- **`Singularity`**: recipe to build a container with `Python3`, `Snakemake`, the scripts, the `Snakefile` and the data. Upon running, the container executes this:

```
snakemake -q -s /pipeline/Snakefile
```

ex01

Building a **self-contained** pipeline

Let's build our container:

```
user@host:~/exercises/ex01$ sudo singularity build ex01.img Singularity
[...]
```

After a short build time, you should see your container:

```
user@host:~/exercises/ex01$ ls -alh
total 150M
drwxr-xr-x 1 user user  238 Jun  8 09:29 .
drwxr-xr-x 1 user user  272 Jun  7 20:28 ..
drwxr-xr-x 1 user user  102 Jun  6 17:01 data
-rwxr-xr-x 1 root root 150M Jun  7 07:47 ex01.img
drwxr-xr-x 1 user user  136 Jun  6 22:15 scripts
-rw-r--r-- 1 user user 1.4K Jun  7 07:54 Singularity
-rw-r--r-- 1 user user  435 Jun  7 20:19 Snakefile
```

ex01

Building a **self-contained** pipeline

We can now run our pipeline inside the container:

```
user@host:~/exercises/ex01$ singularity run ex01.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:01:36 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:01:36 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/10-10:01:36 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/10-10:01:36 | step1.py : Read input data: -2.0
#2018/06/10-10:01:36 | step1.py : Output data after processing: -8.0
#2018/06/10-10:01:36 | step1.py : Step 1 complete.
#2018/06/10-10:01:36 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:01:36 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/10-10:01:36 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/10-10:01:36 | step2.py : Read input data: -8.0
#2018/06/10-10:01:36 | step2.py : Output data after processing: -14.0
#2018/06/10-10:01:36 | step2.py : Step 2 complete.
Pipeline 'ex01' complete. Final output: -14.0
Complete log: /home/user/exercises/ex01/.snakemake/log/2018-06-10T220135.972359.snakemake.log
```

ex01

Building a **self-contained** pipeline

This is the equivalent of calling `Snakemake` inside the container using Singularity's `exec` command:

```
user@host:~/exercises/ex01$ singularity exec ex01.img snakemake -q -s /pipeline/Snakefile
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:06:45 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:06:45 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/10-10:06:45 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/10-10:06:45 | step1.py : Read input data: -2.0
#2018/06/10-10:06:45 | step1.py : Output data after processing: -8.0
#2018/06/10-10:06:45 | step1.py : Step 1 complete.
#2018/06/10-10:06:45 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:06:45 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/10-10:06:45 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/10-10:06:45 | step2.py : Read input data: -8.0
#2018/06/10-10:06:45 | step2.py : Output data after processing: -14.0
#2018/06/10-10:06:45 | step2.py : Step 2 complete.
Pipeline 'ex01' complete. Final output: -14.0
Complete log: /home/user/exercises/ex01/.snakemake/log/2018-06-10T220644.982334.snakemake.log
```

ex01

Building a **self-contained** pipeline

Great! We have now a **reproducible pipeline** in a simple container file that can re-run the *exact same* analysis anywhere where `Singularity` is installed!

Building a **self-contained** pipeline

Great! We have now a **reproducible pipeline** in a simple container file that can re-run the *exact same* analysis anywhere where `Singularity` is installed!

Problems:

- Where is the output?

```
user@host:~/exercises/ex01$ ls data
input.dat
user@host:~/exercises/ex01$ singularity shell ex01.img
Singularity: Invoking an interactive shell within container...

Singularity ex01.img:~> ls /data
input.dat
```

- What if I want to run on different data?

ex02

Running a pipeline on **different data**

A better approach to have data available in a container is to **bind** the data folder into the container.

Exercise:

- Go to `exercises/ex02`
- Have a look at the **data folders**
- Have a look at the `Singularity` **recipe file**

ex02

Running a pipeline on **different data**

The data is **not** included in this container:

```
user@host:~/exercises/ex02$ singularity run ex02.img
Building DAG of jobs...
MissingInputException in line 7 of /pipeline/Snakefile:
Missing input files for rule step1:
/data/input.dat
```

```
user@host:~/exercises/ex02$ singularity shell ex02.img
Singularity: Invoking an interactive shell within container...

Singularity ex02.img:~> ls -alh /data
ls: cannot access /data: No such file or directory
```

ex02

Running a pipeline on **different data**

However, we can **bind** (= mount) folders inside the container using the `-B host:container` parameter:

```
user@host:~/exercises/ex02$ singularity run -B data01:/data ex02.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:08:25 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:08:25 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/10-10:08:25 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/10-10:08:25 | step1.py : Read input data: 13.0
#2018/06/10-10:08:25 | step1.py : Output data after processing: 37.0
#2018/06/10-10:08:25 | step1.py : Step 1 complete.
#2018/06/10-10:08:25 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:08:25 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/10-10:08:25 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/10-10:08:25 | step2.py : Read input data: 37.0
#2018/06/10-10:08:25 | step2.py : Output data after processing: 76.0
#2018/06/10-10:08:25 | step2.py : Step 2 complete.
Pipeline 'ex02' complete. Final output: 76.0
Complete log: /home/user/exercises/ex02/...
```

ex02

Running a pipeline on **different data**

This way, we can run the pipeline in the container on different data:

```
user@host:~/exercises/ex02$ singularity run -B data02:/data ex02.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:09:10 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:09:10 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/10-10:09:10 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/10-10:09:10 | step1.py : Read input data: 7.0
#2018/06/10-10:09:10 | step1.py : Output data after processing: 19.0
#2018/06/10-10:09:10 | step1.py : Step 1 complete.
#2018/06/10-10:09:10 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:09:10 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/10-10:09:10 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/10-10:09:10 | step2.py : Read input data: 19.0
#2018/06/10-10:09:10 | step2.py : Output data after processing: 40.0
#2018/06/10-10:09:10 | step2.py : Step 2 complete.
Pipeline 'ex02' complete. Final output: 40.0
Complete log: /home/user/exercises/ex02/...
```

ex02

Running a pipeline on **different data**

Since we are binding the data folder to the container, the output is now also available *outside* the container:

```
user@host:~/exercises/ex02$ ls -alh data0*
data01:
total 12K
drwxr-xr-x 1 user user 170 Jun  8 11:31 .
drwxr-xr-x 1 user user 272 Jun  8 11:31 ..
-rw-r--r-- 1 user user   2 Jun  6 09:37 input.dat
-rw-r--r-- 1 user user   4 Jun  8 11:31 output1.dat
-rw-r--r-- 1 user user   4 Jun  8 11:31 output2.dat

data02:
total 12K
drwxr-xr-x 1 user user 170 Jun  8 11:35 .
drwxr-xr-x 1 user user 272 Jun  8 11:31 ..
-rw-r--r-- 1 user user   1 Jun  6 09:37 input.dat
-rw-r--r-- 1 user user   4 Jun  8 11:35 output1.dat
-rw-r--r-- 1 user user   4 Jun  8 11:35 output2.dat
```

Running a pipeline on **different data**

Note: running the container will **not** actually re-run the pipeline:

```
user@host:~/exercises/ex02$ ls data01
input.dat  output1.dat  output2.dat

user@host:~/exercises/ex02$ singularity run -B data01:/data ex02.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1
Pipeline 'ex02' complete. Final output: 76.0
Complete log: /home/user/exercises/ex02/...
```

This is because `Snakemake` has a detection system and will only re-run steps where the **output is missing**.

Running a pipeline on **different data**

However, if you **delete the output** and then re-run the container, the pipeline is executed again.

```
user@host:~/exercises/ex02$ rm data01/output*
user@host:~/exercises/ex02$ ls data01
input.dat

user@host:~/exercises/ex02$ singularity run -B data01:/data ex02.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:11:12 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:11:12 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'

[...]

#2018/06/10-10:11:12 | step2.py : Step 2 complete.
Pipeline 'ex02' complete. Final output: 76.0
Complete log: /home/user/exercises/ex02/...

user@host:~/exercises/ex02$ ls data01
input.dat  output1.dat  output2.dat
```

ex03a

Modular containers for pipelines

Imagine now that you want to **modify one of the scripts** in your pipeline, or modify the steps of your pipeline in the pipeline definition file.

You would need to **rebuild** the container every time you do a modification, since the script on the host is not the same script that is inside the container.

Rebuilding the container for every change would be rather cumbersome and would make the **development process longer** and much more tedious.

The **solution** is to use the same **binding mechanism** for your scripts and pipeline definition file as for the data.

ex03a

Modular containers for pipelines

Exercise:

- Go to `exercises/ex03a`
- Have a look at the `Singularity` **recipe file**
- Have a look at the `Snakefile` **definition file**

ex03a

Modular containers for pipelines

In this container, only the **dependencies** for our pipeline are present (`Python3` and `Snakemake`), but the **pipeline definition file** and the **scripts** to run our pipeline are not:

```
user@host:~/exercises/ex03a$ singularity run ex03a.img
Error: Snakefile "/pipeline/Snakefile" not present.

user@host:~/exercises/ex03a$ singularity shell ex03a.img
Singularity: Invoking an interactive shell within container...

Singularity ex03a.img:~> python3.6 --version
Python 3.6.5
Singularity ex03a.img:~> snakemake --version
5.1.4

Singularity ex03a.img:~> ls /pipeline
ls: cannot access /pipeline: No such file or directory
Singularity ex03a.img:~> ls /pipeline/scripts
ls: cannot access /pipeline/scripts: No such file or directory

Singularity ex03a.img:~> ls /data
ls: cannot access /data: No such file or directory
```

ex03a

Modular containers for pipelines

Therefore, in order to run our pipeline in this container, we have to **bind** these additional script files:

```
user@host:~/exercises/ex03a$ singularity run -B Snakefile:/pipeline/Snakefile \
    -B scripts:/pipeline/scripts -B data:/data ex03a.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:12:14 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:12:14 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/10-10:12:14 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/10-10:12:14 | step1.py : Read input data: -2.0
#2018/06/10-10:12:14 | step1.py : Output data after processing: -8.0
#2018/06/10-10:12:14 | step1.py : Step 1 complete.
#2018/06/10-10:12:14 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:12:14 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/10-10:12:14 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/10-10:12:14 | step2.py : Read input data: -8.0
#2018/06/10-10:12:14 | step2.py : Output data after processing: -14.0
#2018/06/10-10:12:14 | step2.py : Step 2 complete.
Pipeline 'ex03a' complete. Final output: -14.0
Complete log: /home/user/exercises/ex03a/...
```

ex03b

Modular containers for pipelines

Even simpler, one can use a great feature of `Singularity`: the **automatic binding of the current folder**.

Indeed, `Singularity` will try to mount the current directory in the working directory of the container:

```
user@host:~/exercises/ex03b$ ls
data  ex03b.img  scripts  Singularity  Snakefile

user@host:~/exercises/ex03b$ singularity shell ex03b.img
Singularity: Invoking an interactive shell within container...

Singularity ex03b.img:~/exercises/ex03b> ls
data  ex03b.img  scripts    Singularity  Snakefile
```

ex03b

Modular containers for pipelines

Therefore, one can modify our pipeline to arrange that all paths are **relative to the current folder**, and use directly the folders in the current directory both inside and outside the container:

```
rule step1:
     input:
         'data/input.dat'
     output:
         'data/output1.dat'
     shell:
         'scripts/step1.py {input} {output}'
```

```
%runscript

    snakemake -q -s Snakefile
```

*Note that the paths are now relative to the current directory and **not absolute** anymore.*

ex03b

Modular containers for pipelines

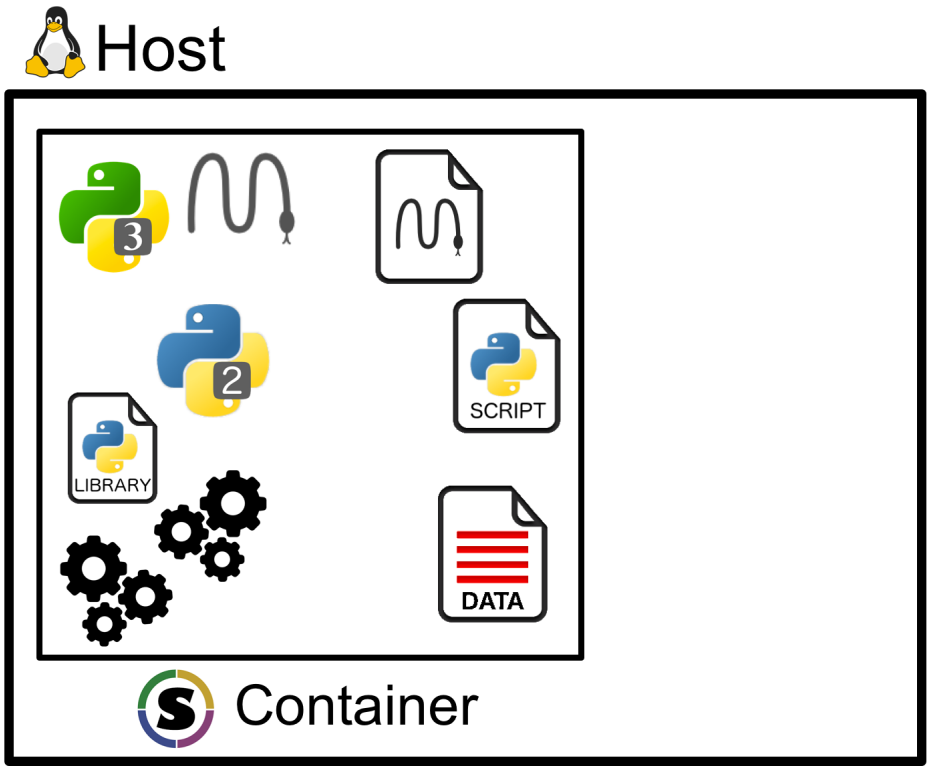
This way, our execution of the pipeline inside the container becomes again as simple as:

```
user@host:~/exercises/ex03b$ singularity run ex03b.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/10-10:13:12 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/10-10:13:12 | step1.py : Input: 'data/input.dat', output: 'data/output1.dat'

[...]

Pipeline 'ex03b' complete. Final output: -14.0
Complete log: /home/user/exercises/ex03b/...
```

While still retaining the **flexibility** of changing the scripts, the pipeline and the data easily, **without rebuilding the container**.

ex01

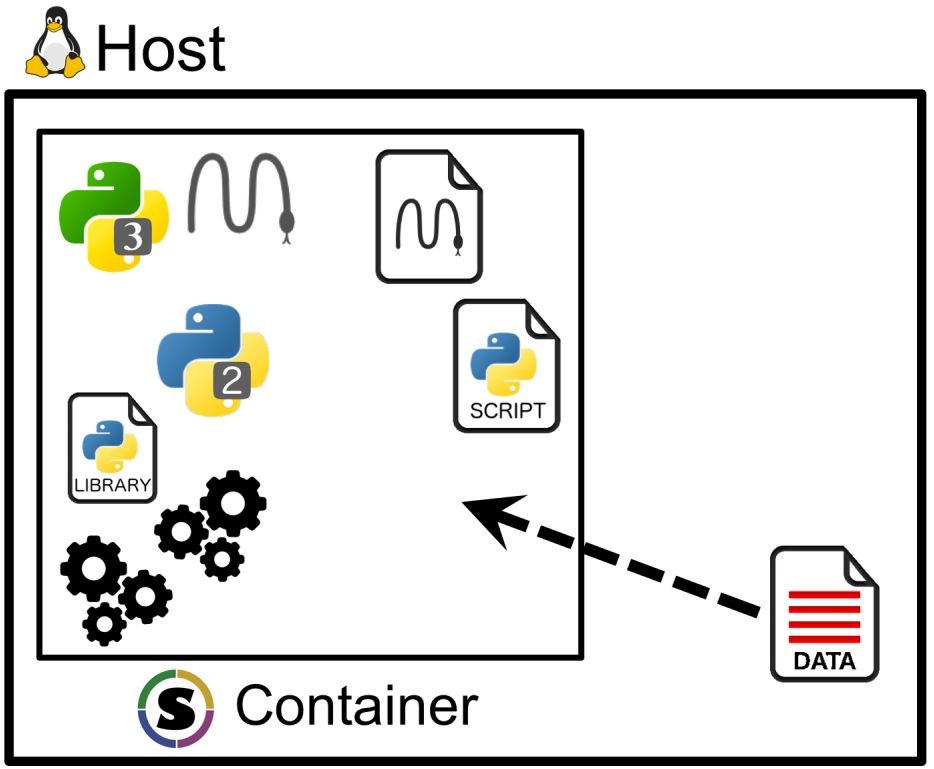
Summary so far



DATA	Data **copied** in the container
2	`Python2` **running scripts** in the container
3	`Python3` **running Snakemake** in the container
	`Snakemake` **running** in the container
	`Snakefile` **copied** in the container
SCRIPT	Scripts **copied** in the container

ex02

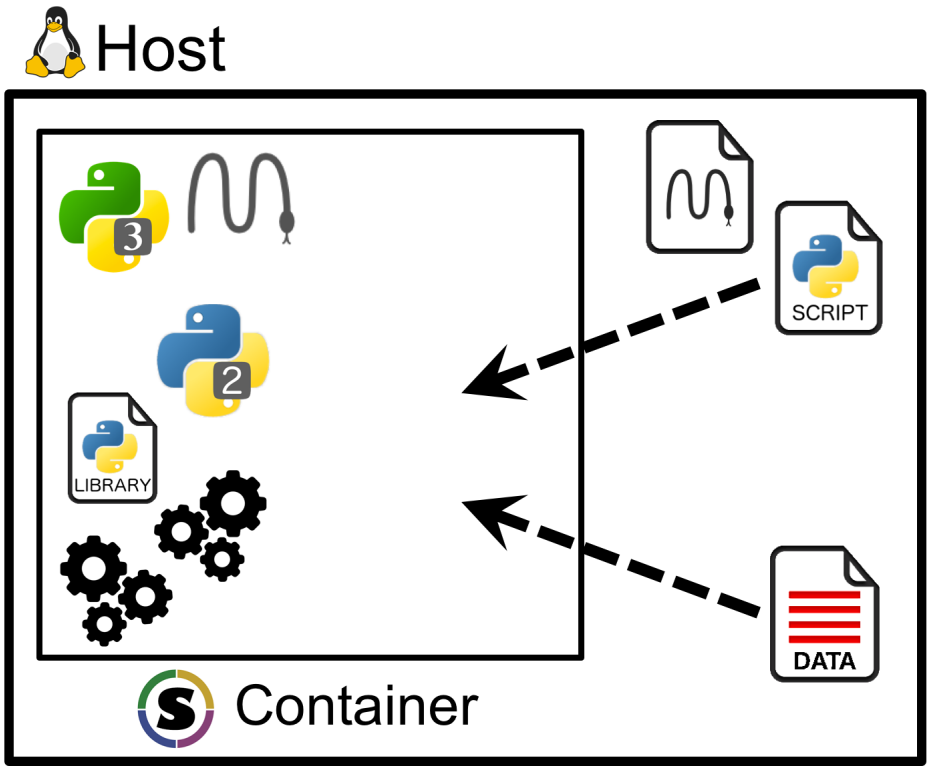
Summary so far



DATA	Data **bound** in the container
2	`Python2` **running scripts** in the container
3	`Python3` **running Snakemake** in the container
	`Snakemake` **running** in the container
	`Snakefile` **copied** in the container
SCRIPT	Scripts **copied** in the container

ex03

Summary so far



DATA	Data **bound** in the container
2	`Python2` **running scripts** in the container
3	`Python3` **running Snakemake** in the container
	`Snakemake` **running** in the container
	`Snakefile` **bound** in the container
SCRIPT	Scripts **bound** in the container

ex04

Running **out of the box**

In all the previous exercises, we **mounted** (or copied) all data & scripts **inside** the container and **executed** our pipeline there.

This is **convenient** as everything runs in a well-contained "box" and it is **easy to share** and understand what is going on.

However, there is a **drawback** to this approach of "**running inside**": we only see what is inside the container. Any software present on the host is **not visible**:

```
user@host:~/exercises/ex04$ which python
/usr/bin/python
user@host:~/exercises/ex04$ singularity shell ex04.img
Singularity: Invoking an interactive shell within container...

Singularity ex04.img:~> which python
bash: which: command not found
```

ex04

Running **out of the box**

Therefore, **"running inside"** can be a problem in a few situations:

- Some tools cannot be installed inside a container due to **licensing**
- On **HPC clusters**, the **scheduler** is (often) only installed on the login nodes and cannot be accessed from the container.

Thus, it is often needed that the pipeline is **running outside** of the container and only **calling tools** that are *packaged* inside the container.

In other words, one can say that the **orchestration** of the tools (≈ the pipeline) can be done either **inside** or **outside** the container.

ex04

Running **out of the box**

In the container of exercise 4, there is no more `Python3` or `Snakemake` in the container:

```
user@host:~/exercises/ex04$ singularity run ex04.img
Software installed in this container:
Python 2.7.15rc1

user@host:~/exercises/ex04$ singularity exec ex04.img snakemake
/.singularity.d/actions/exec: line 9: exec: snakemake: not found
```

But, luckily, we have them on the host:

```
user@host:~/exercises/ex04$ python3 --version
Python 3.6.5

user@host:~/exercises/ex04$ snakemake --version
5.1.4
```

ex04

Running **out of the box**

Exercise:

- Go to `exercises/ex04`
- Have a look at the `Singularity` recipe file
- Have a look at the `Snakefile`
- Have a look at the scripts: `step1.py` & `step2.py`

ex04

Running **out of the box**

Our `Singularity` file defines now a very empty and **sad container**:

```
Bootstrap: docker
From: centos:7.4.1708

%help
    This is the container for exercise 4.

%setup

%post

%files

%runscript

    echo "Software installed in this container:"
    python --version

%labels

    MAINTAINER Balazs Laurenczy (ETHZ)
    VERSION ex04

%test
```

ex04

Running **out of the box**

Our `Snakefile` is now **run by `Snakemake` on the host** and it is calling the scripts **in two different ways**:

```
rule step1:
     input:
          'data/input.dat'
     output:
          'data/output1.dat'
     shell:
          'singularity exec ex04.img python2.7 scripts/step1.py {input} {output}'

rule step2:
     input:
          'data/output1.dat'
     output:
          'data/output2.dat'
     shell:
          'python2.7 scripts/step2.py {input} {output}'
```

The scripts are now also showing the OS on which they are running:

```
logging.info("Running on %s %s %s", *platform.linux_distribution())
```

ex04

Running **out of the box**

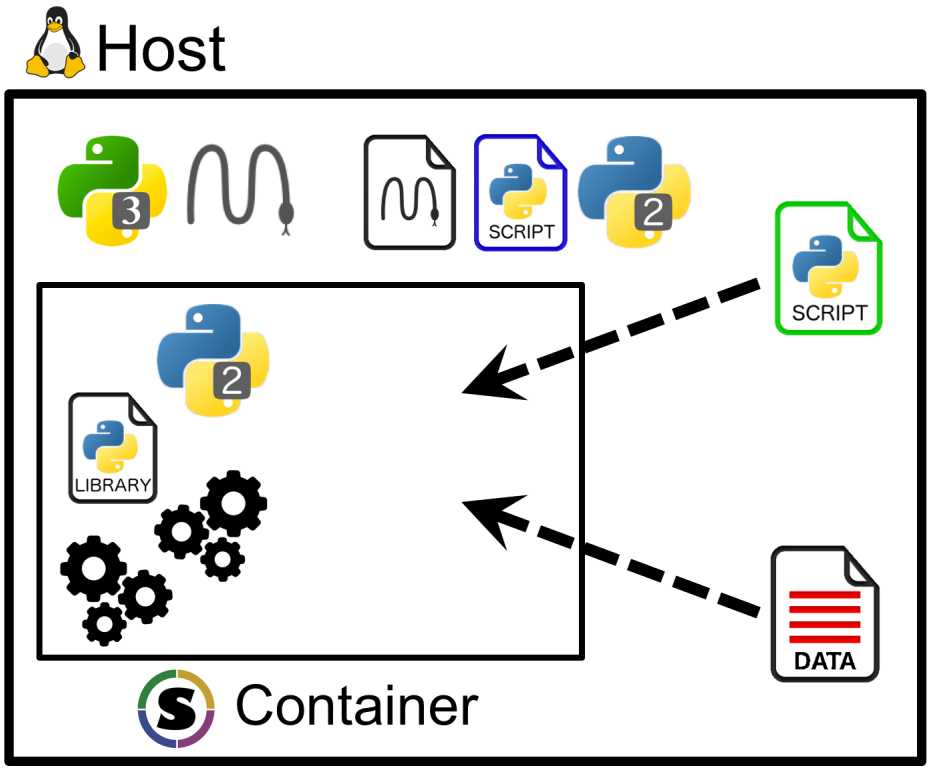
Running the pipeline now requires us to **call directly** `Snakemake` on the host, rather than running the container:

```
user@host:~/exercises/ex04$ snakemake -q
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/09-06:45:59 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/09-06:45:59 | step1.py : Input: 'data/input.dat', output: 'data/output1.dat'
[...]
#2018/06/09-06:45:59 | step1.py : Step 1 complete.
#2018/06/09-06:45:59 | step2.py : Running on Ubuntu 16.04 xenial
#2018/06/09-06:45:59 | step2.py : Input: 'data/output1.dat', output: 'data/output2.dat'
[...]
#2018/06/09-06:45:59 | step2.py : Step 2 complete.
Pipeline 'ex04' complete. Final output: -14.0
Complete log: /home/user/exercises/ex04/...
```

***Note** how the two rules did not see the same OS.*

ex04

Summary



	Data **bound** in the container
	`Python2` **running scripts** in the container **and** on the host
	`Python3` **running `Snakemake`** on the host
	`Snakemake` **running** on the host
	`Snakefile` on the host
	`script1` **bound** in the container, `script2` on the host

ex05

Multiple containers

Doing the **orchestration outside** gives us some more **flexibility**, at the cost of a bit **less portability**.

It is indeed **easier to share** a single container doing everything - *no additional files or instructions* - rather than giving a container, some scripts and the need of having `Python3` and `Snakemake` installed on the host.

However, our pipeline can now use **all the tools available** on the host (licensed software, scheduler) and still escape the need of installing *all* **dependencies** required by the pipeline, since they are nicely **packaged in the container**

ex05

Multiple containers

Furthermore, this saves us from on additional trouble: having only a **single gigantic container** that packs all the tools in a beautifully **chaotic toolbox**.

ex05

Multiple containers

A better approach would be either:

1. To have an **organised toolbox** using **SCI-F**, a container format for packing several applications in a single container (see documentation here).

ex05

Multiple containers

A better approach would be either:

2. To have **multiple containers**, using one container per tool (or per step of your pipeline).

ex05

Multiple containers

Exercise:

- Go to `exercises/ex05`
- Have a look at the `Singularity` recipe files (5A & 5B)
- Have a look at the `Snakefile`
- Have a look at the scripts: `step1.py` & `step2.py`

ex05

Multiple containers

In this exercise, each of the script is installed in its **own container**, and our pipeline is calling the two different containers at different steps:

```
rule step1:
     input:
         'data/input.dat'
     output:
         'data/output1.dat'
     shell:
         'singularity exec ex05A.img step1.py {input} {output}'

rule step2:
     input:
         'data/output1.dat'
     output:
         'data/output2.dat'
     shell:
         'singularity exec ex05B.img step2.py {input} {output}'
```

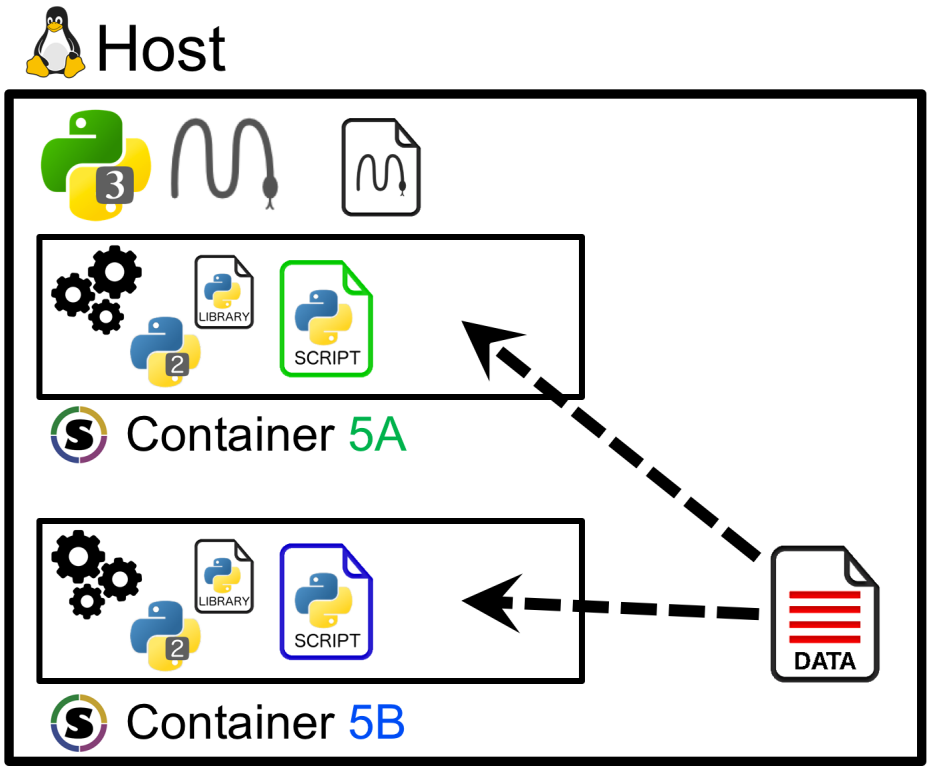
Multiple containers

In this exercise, each of the script is installed in its **own container**, and our pipeline is calling the two different containers at different steps:

```
user@host:~/exercises/ex05$ snakemake -q
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/13-08:29:22 | step1.py : Running in container 'ex05A.img' on CentOS Linux 7.4.1708 Core
#2018/06/13-08:29:22 | step1.py : Input: 'data/input.dat', output: 'data/output1.dat'
#2018/06/13-08:29:22 | step1.py : Reading input file: 'data/input.dat' ...
#2018/06/13-08:29:22 | step1.py : Read input data: -2.0
#2018/06/13-08:29:22 | step1.py : Output data after processing: -8.0
#2018/06/13-08:29:22 | step1.py : Step 1 complete.
#2018/06/13-08:29:22 | step2.py : Running in container 'ex05B.img' on CentOS Linux 7.4.1708 Core
#2018/06/13-08:29:22 | step2.py : Input: 'data/output1.dat', output: 'data/output2.dat'
#2018/06/13-08:29:22 | step2.py : Reading input file: 'data/output1.dat' ...
#2018/06/13-08:29:22 | step2.py : Read input data: -8.0
#2018/06/13-08:29:22 | step2.py : Output data after processing: -14.0
#2018/06/13-08:29:22 | step2.py : Step 2 complete.
Pipeline 'ex05' complete. Final output: -14.0
Complete log: /home/user/exercises/ex05/...
```

ex05

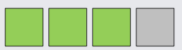
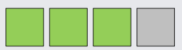
Summary



DATA	Data **bound** in the container
Python2	`Python2` **running scripts** in both **container 5A** and **container 5B**
Python3	`Python3` **running Snakemake** on the host
Snakemake	`Snakemake` **running** on the host
Snakefile	`Snakefile` on the host
SCRIPT	`script1` **copied** in the **container 5A**, `script2` **copied** in the **container 5B**

ex01-5

Summary

Use case	Flexibility	Portability	Ease of use	Scalability	Reproducibility
Single container with **everything inside**, pipeline **running inside** the container					
Single container with everything inside **but the data**, pipeline **running inside** the container					
Single container with pipeline **running inside** the container, scripts and data **mounted** in a flexible way					
Pipeline running outside of the container using a **single container** packaging all the tools					
Pipeline running outside of the container using **multiple** containers (~ one container per tool or step)					

ex06

Validating a pipeline and its container

When sharing a pipeline and a container, it is **good practice** to check whether they are **valid**, meaning whether the pipeline and container are the same and will **run the same way** on our machine as it was on the original machine.

The best way to confirm this is to **run the pipeline** on some defined **test data** and see whether we get the expected **pre-defined result**.

ex06

Validating a pipeline and its container

In all the previous exercises, the **input and output** of our pipeline was very **simple**, it was just a simple number.

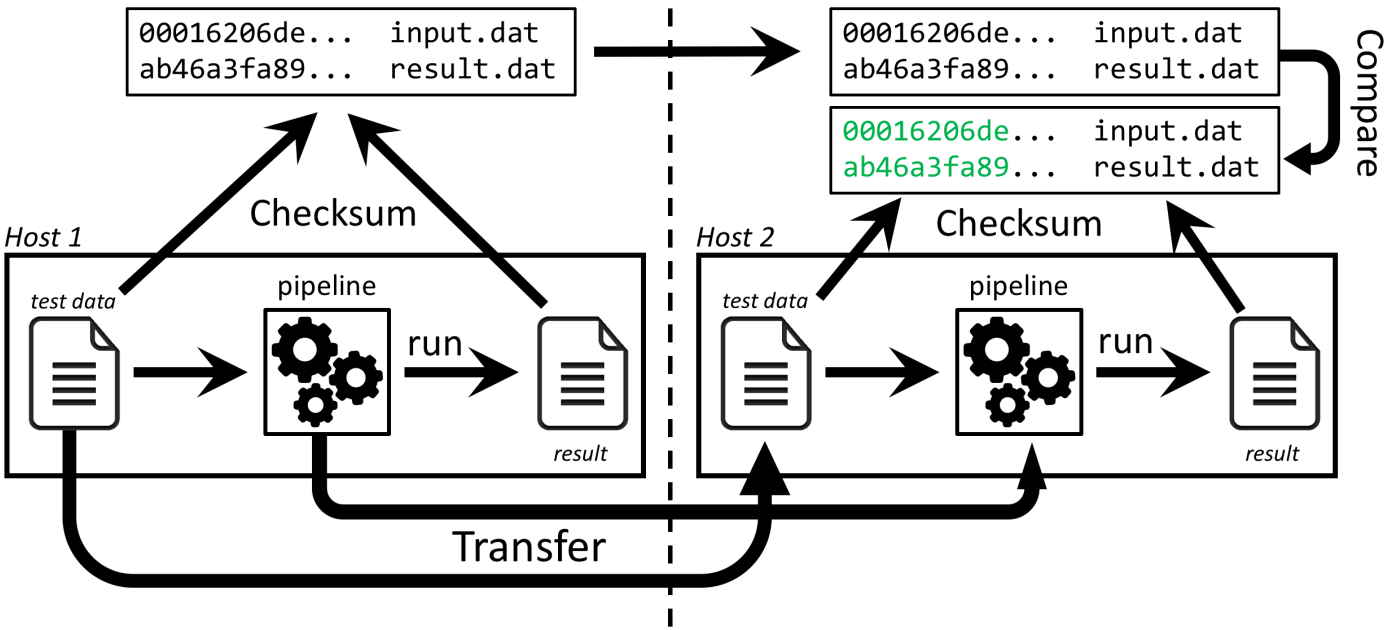
However, in real use cases, the input and output are often much **more complex** and checking whether the output is the same can be more difficult.

A common practice is therefore to use **checksums** (like MD5 or SHA-1) on the **initial input** (test data) and on the **final output**, like MD5 or SHA-1.

ex06

Validating a pipeline and its container

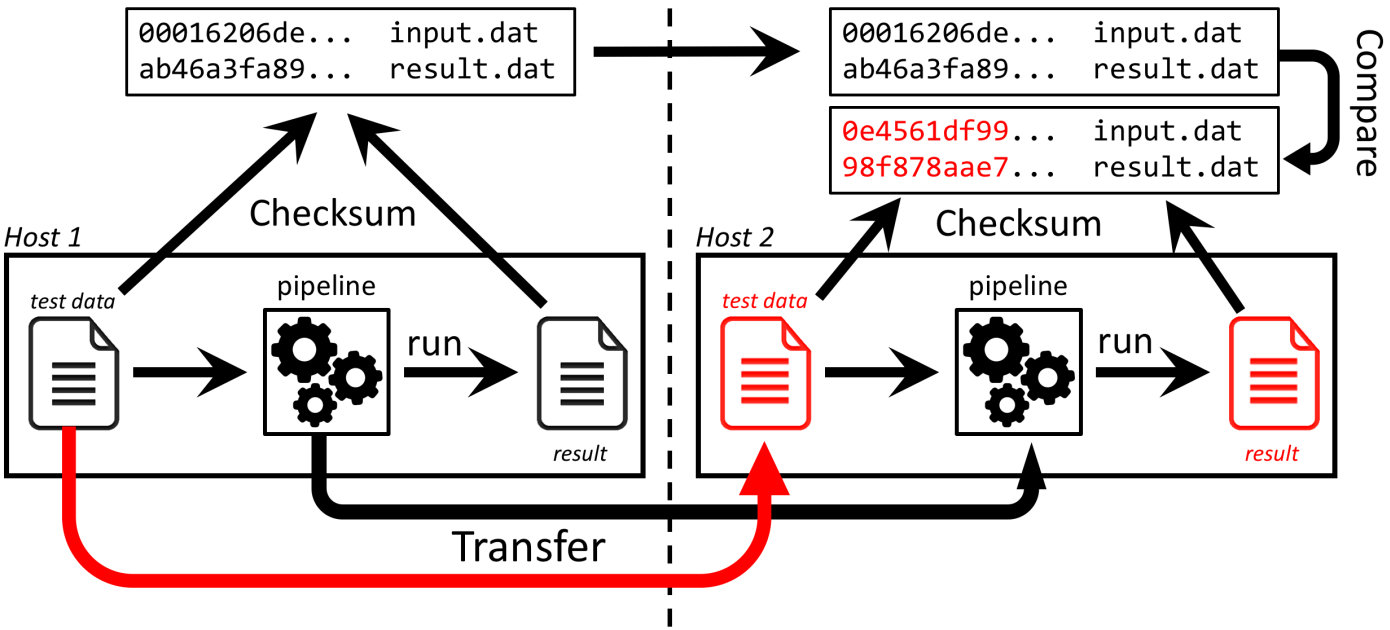
Here is the validation process when everything is running without problems:



ex06

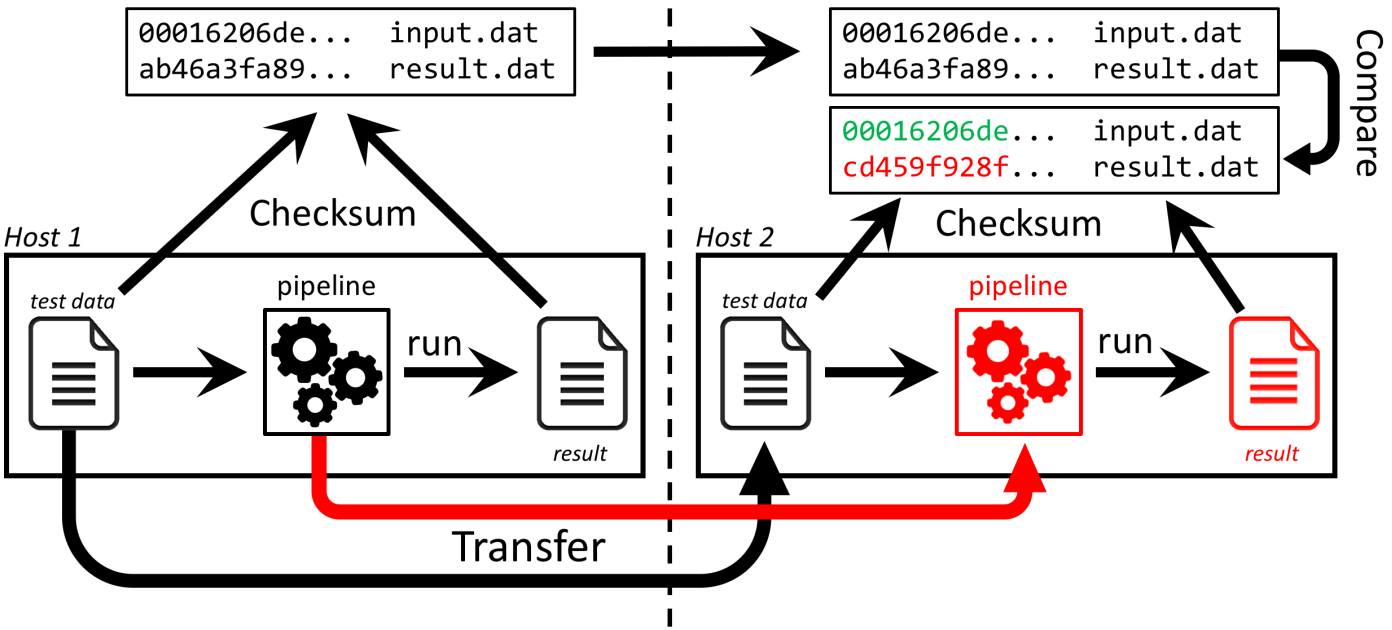
Validating a pipeline and its container

Here is the validation process in case of a problem with the input data:



Validating a pipeline and its container

Here is the validation process in case of a problem with the pipeline or the container:



ex06

Validating a pipeline and its container

Exercise:

- Go to `exercises/ex06`
- Have a look at the "`*.snake`" `Snakemake` files
- Have a look at the scripts: `step1.py` & `step2.py`
- Run through the different pipelines by using the different `Snakefiles`:

```
snakemake -q -s pipeline_XXX.snake
```

- Check whether the pipeline is valid by running a checksum on the output:

```
sha1sum test_data/*
```

- In case your pipeline is not valid, find out why!

ex06

Validating a pipeline and its container

Running with the **valid** pipeline:

```
user@host:~/exercises/ex06$ rm -rf test_data/output*

user@host:~/exercises/ex06$ snakemake -q -s pipeline_valid.snake
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex06' complete. Final output: 94.0 -488.0 [...]
Complete log: /home/user/exercises/ex06/...

user@host:~/exercises/ex06$ sha1sum test_data/*
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  b534f96cf3812474b386915ea38d6a2fbf91a1d7  test_data/output2.dat

user@host:~/exercises/ex06$ cat checksums.sha1
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  b534f96cf3812474b386915ea38d6a2fbf91a1d7  test_data/output2.dat
```

ex06

Validating a pipeline and its container

Running with **bad input data**:

```
user@host:~/exercises/ex06$ rm -rf bad_test_data/output*

user@host:~/exercises/ex06$ snakemake -q -s pipeline_bad_data.snake
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex06' complete. Final output: 88.0 -488.0 [...]
Complete log: /home/user/exercises/ex06/...

user@host:~/exercises/ex06$ sha1sum bad_test_data/*
  7393d136a8ff4d5dbae9c1712eb8c27fe129b8d6  bad_test_data/input.dat
  90474a941b58db46eeff568dc559325b076d0749  bad_test_data/output1.dat
  00273d709f71c8652fe0451654a35379661ac466  bad_test_data/output2.dat

user@host:~/exercises/ex06$ cat checksums_sha1.txt
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  b534f96cf3812474b386915ea38d6a2fbf91a1d7  test_data/output2.dat
```

ex06

Validating a pipeline and its container

Running with a **bad pipeline**:

```
user@host:~/exercises/ex06$ rm -rf test_data/output*

user@host:~/exercises/ex06$ snakemake -q -s pipeline_bad_pipeline.snake
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex06' complete. Final output: 100.0 -482.0 [...]
Complete log: /home/user/exercises/ex06/...

user@host:~/exercises/ex06$ sha1sum test_data/*
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  24817e0f1a58f6651884d26b83866410064aff93  test_data/output1.dat
  f891870ddac97306b25fc5359aae97582731599f  test_data/output2.dat

user@host:~/exercises/ex06$ cat checksums_sha1.txt
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  b534f96cf3812474b386915ea38d6a2fbf91a1d7  test_data/output2.dat
```

ex06

Validating a pipeline and its container

Running with **bad containers**:

```
user@host:~/exercises/ex06$ rm -rf test_data/output*

user@host:~/exercises/ex06$ snakemake -q -s pipeline_bad_container.snake
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex06' complete. Final output: 136.0 -737.0 [...]
Complete log: /home/user/exercises/ex06/...

user@host:~/exercises/ex06$ sha1sum test_data/*
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  f5478d868eded37c9d100bab8e9a59a0144761e8  test_data/output2.dat

user@host:~/exercises/ex06$ cat checksums_sha1.txt
  b833473ecf183d5e34d839a7ead59b944c571318  test_data/input.dat
  cbe8b7a4dc2c3e0761d070f116ae44c51b860d2c  test_data/output1.dat
  b534f96cf3812474b386915ea38d6a2fbf91a1d7  test_data/output2.dat
```

ex07

Testing it **all**

Let us now do a *real* **reproducibility & portability test** with our own container and pipeline:

Exercise:

- Go to `exercises/ex07`
- Create your own input data
- Modify the scripts with your own "computation"
- Modify the version in the `Singularity` file
- Build your container
- Run your container
- Create the checksums for your container
- Share your container with your neighbour:
  test data, container, pipeline & checksums
- Run their container (they run yours)
- Validate their container (they validate yours)

ex07

Testing it **all**

Generate your **input data**:

```
user@host:~$ echo "from random import random
for i in range(0, 30):
    print(round(random() * 198 -99))
" | python3 > input.dat

user@host:~$ head input.dat
-64
-43
-65
93
65
-85
-9
-66
-7
95
```

ex07

Testing it **all**

Modify the **scripts** with your own "computation":

`step1.py`:

```
# CHANGE THE COMPUTATION HERE
output_data = [(x * 42.0) - 56.0 for x in input_data]
```

`step2.py`:

```
# CHANGE THE COMPUTATION HERE
output_data = [(x + 11.0) * 42.0 for x in input_data]
```

ex07

Testing it **all**

Modify the **version** in the `Singularity` file:

From:

```
%labels

    MAINTAINER [Last-name first-name (institution)]
    VERSION ex07_[your-container-name]
```

To:

```
%labels

    MAINTAINER Luke Skywalker (Jedi Academy)
    VERSION ex07_luke
```

ex07

Testing it **all**

Build your container:

```
user@host:~/exercises/ex07$ sudo singularity build ex07.img Singularity
Using container recipe deffile: Singularity
Sanitizing environment
Adding base Singularity environment to container
Docker image path: index.docker.io/library/centos:7.4.1708

[...]

Singularity container built: ex07.img
Cleaning up...

user@host:~/exercises/ex07$ ls -alh
total 137M
drwxr-xr-x  5 user user 4.0K Jun 14 19:30 .
drwxr-xr-x 12 user user 4.0K Jun 14 19:15 ..
drwxr-xr-x  2 user user 4.0K Jun 14 19:24 data
-rwxr-xr-x  1 root root  69M Jun 14 19:24 ex07.img
drwxr-xr-x  2 user user 4.0K Jun 14 19:15 scripts
-rw-r--r--  1 user user  732 Jun 14 19:23 Singularity
-rw-r--r--  1 user user  479 Jun 14 19:23 Snakefile
```

ex07

Testing it **all**

Run your container & create the **checksums**:

```
user@host:~/exercises/ex07$ snakemake -q
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex07' complete. Final output: -114786.0 -77742.0 [...]
Complete log: /home/user/exercises/ex07/...

user@host:~/exercises/ex07$ sha1sum data/* > checksums.sha1

user@host:~/exercises/ex07$ cat checksums.sha1
075454cc29ab6d36b9e59cbe6a4db575302a74a2  data/input.dat
f2cac8544de08c6090b62ddf63b9a4f7b948a23b  data/output1.dat
5c0e1b3e2d88ddda5be2159b9a7fca71fc991ffe  data/output2.dat
```

ex07

Testing it **all**

Share your container with your neighbour:

- Via `scp` and email:

```
# download
local_user@local_host:~$ scp user@host:/path/to/container.img .

# manually send by email

# upload
local_user@local_host:~$ scp container.img user@host:/path/to/container.img
```

- Via the `tmp` folder:

```
# move your container
user@host:~/exercises/ex07$ cp ex07_luke.img /tmp/.

# get the partner's container
user@host:~/exercises/ex07$ cp /tmp/ex07_yoda.img .

# change the owner
user@host:~/exercises/ex07$ sudo chown user:group ex07_yoda.img
```

ex07

Testing it **all**

Run their container & **validate** it:

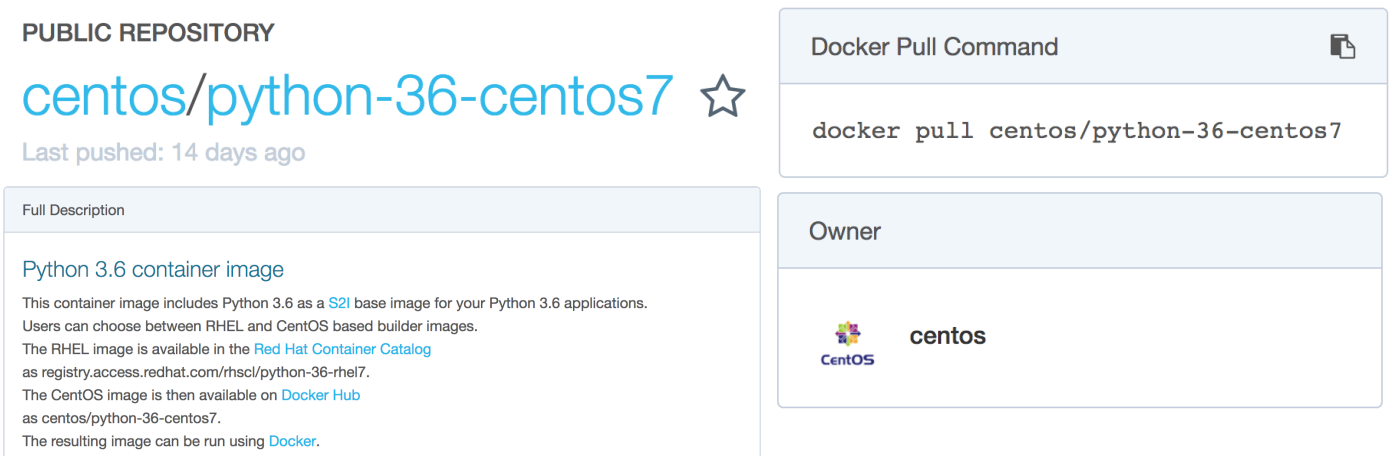
```
user@host:~/exercises/ex07$ snakemake -q
Building DAG of jobs...
Using shell: /bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex07' complete. Final output: -44226.0 -77722.0 [...]
Complete log: /home/user/exercises/ex07/...

user@host:~/exercises/ex07$ sha1sum data/*
  2168e7d05cc31d5927d8f461c08b76ca83186ad3  data/input.dat
  c2a8e2a780ced501569e04714ecf213bac383ee4  data/output1.dat
  7df6f1d442113f29f1e5efb8846f59fdc89e402a  data/output2.dat

user@host:~/exercises/ex07$ cat checksums.sha1
  2168e7d05cc31d5927d8f461c08b76ca83186ad3  data/input.dat
  c2a8e2a780ced501569e04714ecf213bac383ee4  data/output1.dat
  7df6f1d442113f29f1e5efb8846f59fdc89e402a  data/output2.dat
```

ex08

A different **base image**

An important idea to keep in mind when working with containers is the concept of **reusability**.

Before starting to create a new container, have a look whether somebody already created that container (or a similar one) before and check whether you can **use that container** or **base your container** on that one.



ex08

A different **base image**

Exercise:

- Go to `exercises/ex08`
- Have a look at the `Singularity` **recipe file**

ex08

A different **base image**

Building the image:

```
user@host:~/exercises/ex08$ sudo singularity build ex082.img Singularity
Using container recipe deffile: Singularity
Sanitizing environment
Adding base Singularity environment to container
Docker image path: index.docker.io/centos/python-36-centos7:latest
Cache folder set to /root/.singularity/docker

[...]

Singularity container built: ex08.img
Cleaning up...
```

ex08

A different **base image**

Running the container:

```
user@host:~/exercises/ex08$ singularity run ex08.img
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3

[...]

Pipeline 'ex08' complete. Final output: -14.0
Complete log: /home/user/exercises/ex08/...
```

ex09

Singularity containers via a **Dockerfile**

In all the previous examples, we started our container build process with the **`Singularity` recipe file**.

This is the most simple and obvious way to **build `Singularity` containers**.

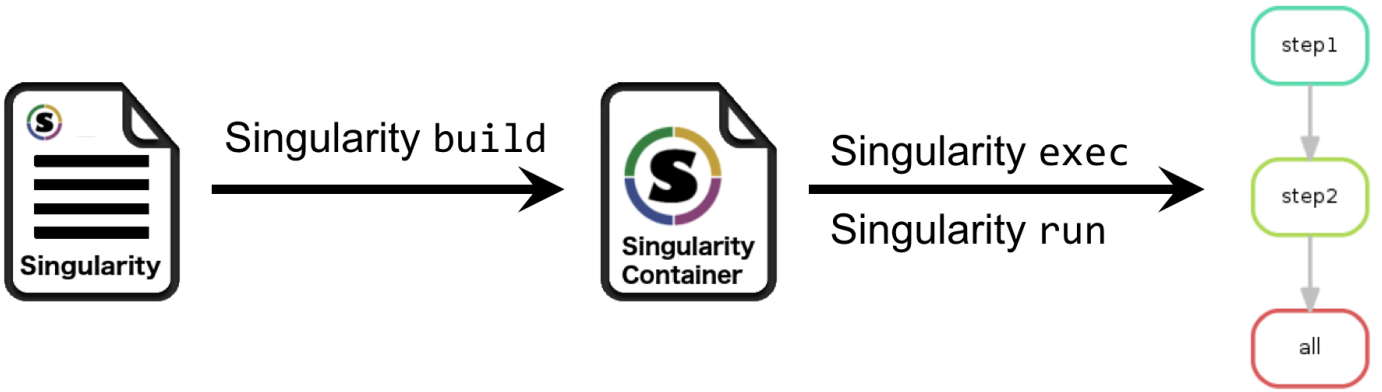
However, `Singularity` has also some level of **compatibility** with **`Docker`**, as `Singularity` can *pull* `Docker` images into `Singularity` containers.

Therefore it is possible to get a `Singularity` container from a **`Dockerfile`**.

ex09

Singularity containers via a **Dockerfile**

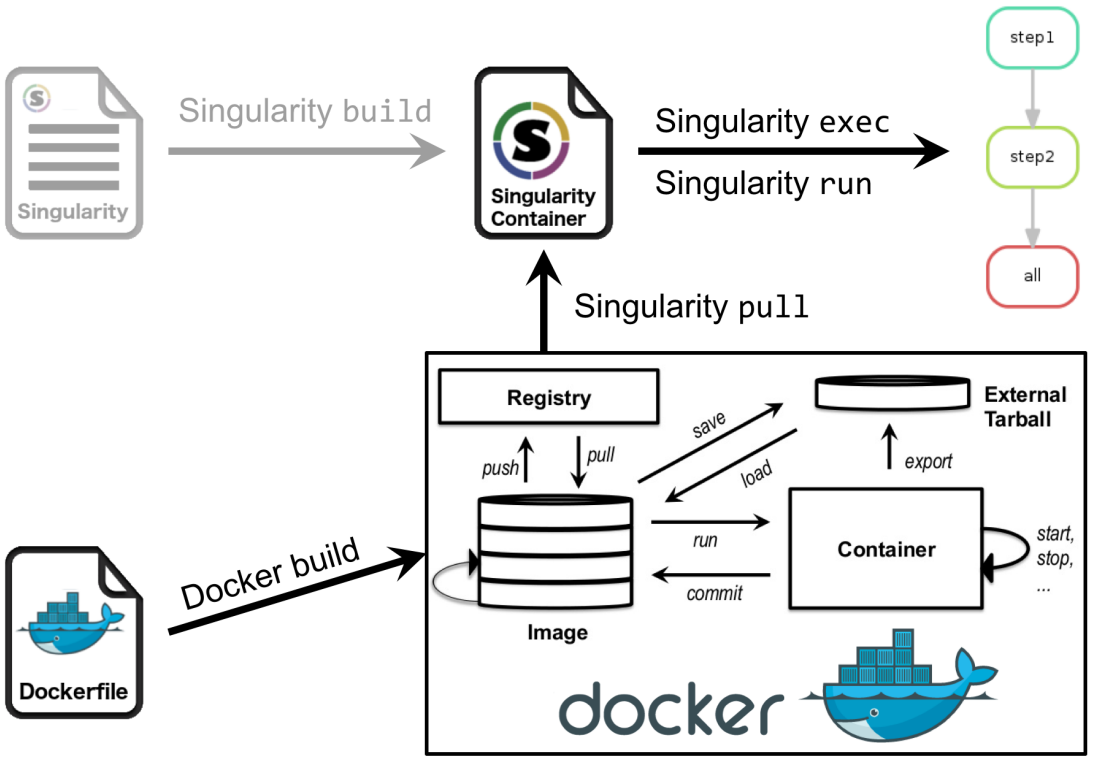
Here is what we have been doing so far:



ex09

Singularity containers via a **Dockerfile**

Here is how it works when starting with a **`Dockerfile`**:



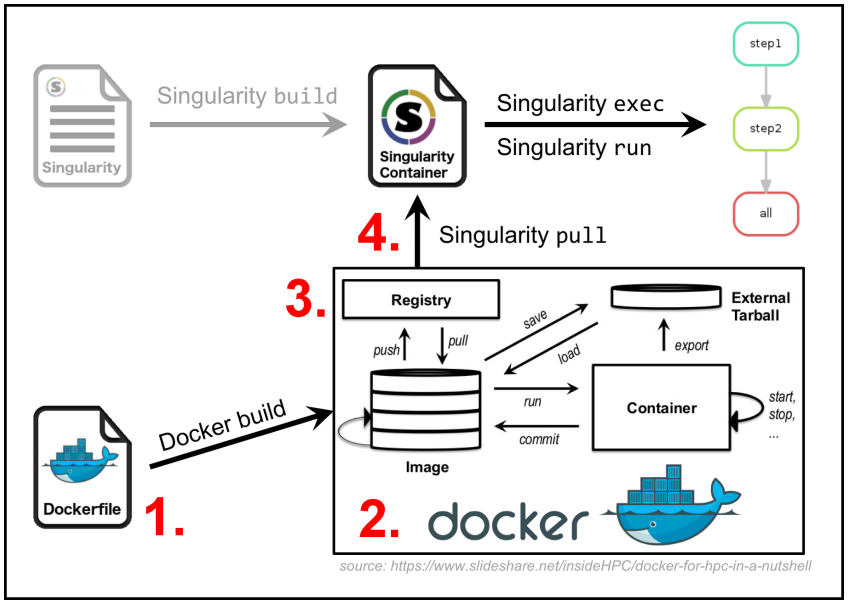
source: https://www.slideshare.net/insideHPC/docker-for-hpc-in-a-nutshell

ex09

Singularity containers via a Dockerfile

We therefore need the following for building `Singularity` containers from a `Dockerfile`:

1. a `Dockerfile`
2. `Docker`
3. a `Docker` registry
4. `Singularity`



Note: These components do not need to be all on the same host.

ex09

Singularity containers via a Dockerfile

Here is what our `Dockerfile` looks like:

```
FROM centos:7.4.1708

# Enable repo & install compilers
[...]

# Install Python 3.6 & Snakemake
RUN yum -y install https://centos7.iuscommunity.org/ius-release.rpm \
    && yum -y install python36u python36u-pip python36u-devel \
    && pip3.6 install snakemake

# Copy test files into the container
COPY data/input.dat /data/
# Copy pipeline scripts into the container
COPY scripts/* /pipeline/scripts/
COPY Snakefile /pipeline/

# Add the pipeline scripts to the $PATH and make sure they are executable
ENV PATH $PATH:/pipeline/scripts
RUN chmod +x /pipeline/scripts/*.py

# Make sure the data is readable even without root
RUN chmod -R 777 /data

# Define the command to execute when running the container
ENTRYPOINT ["snakemake", "-q", "-s", "/pipeline/Snakefile"]

# Define some label for the container
LABEL container.maintainer="Balazs Laurenczy (ETHZ)"
LABEL container.version="ex09"
```

ex09

Singularity containers via a Dockerfile

We can now start the build process:

1. **Build** the `Docker` image
2. **Push** the `Docker` image to a `Docker` registry
3. **Pull** the `Docker` image as a `Singularity` container from the registry

ex09

Singularity containers via a **Dockerfile**

1. **Build** the `Docker` image - status before the build

```
user@host:~/exercises/ex09$ ls -alh
total 8.0K
drwxr-xr-x  6 bal staff  204 Jun 19 09:57 .
drwxr-xr-x 15 bal staff  510 Jun 15 15:34 ..
-rw-r--r--  1 bal staff 1019 Jun 19 09:57 Dockerfile
-rw-r--r--  1 bal staff  477 Jun  7 18:51 Snakefile
drwxr-xr-x  3 bal staff  102 Jun  6 11:29 data
drwxr-xr-x  4 bal staff  136 Jun  6 12:09 scripts

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY  TAG  IMAGE ID  CREATED  SIZE


user@host:~/exercises/ex09$
```

ex09

Singularity containers via a **Dockerfile**

1. **Build** the `Docker` image - launching the build ...

```
user@host:~/exercises/ex09$ sudo docker build -t ex09:latest .
Sending build context to Docker daemon  10.24kB
Step 1/12 : FROM centos:7.4.1708
7.4.1708: Pulling from library/centos
18b8eb7e7f01: Already exists
Digest: sha256:2a61f8abd6250751c4b1dd3384a2bdd8f87e0e60d11c064b8a90e2e552fee2d7
Status: Downloaded newer image for centos:7.4.1708
 ---> 3afd47092a0e
Step 2/12 : RUN yum -y install epel-release && yum-config-manager --enable epel
 ---> Running in 8acc6ceebb12

[...]

Step 11/13 : ENTRYPOINT ["snakemake", "-q", "-s", "/pipeline/Snakefile"]
 ---> Running in 98f79cfe7632
Removing intermediate container 98f79cfe7632
 ---> 5552b1cd0543
Step 12/13 : LABEL container.maintainer="Balazs Laurenczy (ETHZ)"
 ---> Running in 03dadf0e0806
Removing intermediate container 03dadf0e0806
 ---> d3872319711b
Step 13/13 : LABEL container.version="ex09"
 ---> Running in 59154c8b79d0
Removing intermediate container 59154c8b79d0
 ---> 8f11e10d5764
Successfully built 8f11e10d5764
Successfully tagged ex09:latest
```

ex09

Singularity containers via a **Dockerfile**

1. **Build** the `Docker` image - status after the build

```
user@host:~/exercises/ex09$ ls -alh
total 8.0K
drwxr-xr-x  6 bal staff  204 Jun 19 09:57 .
drwxr-xr-x 15 bal staff  510 Jun 15 15:34 ..
-rw-r--r--  1 bal staff 1019 Jun 19 09:57 Dockerfile
-rw-r--r--  1 bal staff  477 Jun  7 18:51 Snakefile
drwxr-xr-x  3 bal staff  102 Jun  6 11:29 data
drwxr-xr-x  4 bal staff  136 Jun  6 12:09 scripts

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ex09                latest              8f11e10d5764        28 seconds ago      718MB
centos              7.4.1708            3afd47092a0e        7 months ago        197MB
```

ex09

Singularity containers via a **Dockerfile**

2. **Push** the `Docker` image to **`Docker` registry**

There are several options to use as `Docker` registry:

A. **`Docker Hub`**: this is easiest and most common place to push images, but the images are **public** and anyone can pull them.

B. **A *private* registry**: it is possible to run your own `Docker` registry on your own server. SIS has its own (internal) registry via **our SIS `GitLab`** (see https://sissource.ethz.ch).

   But there are also publicly available *free* options, like **Google's container registry** (see https://cloud.google.com/container-registry).

C. **Local registry**: it is possible to run a `Docker` *container* that runs a registry on the same host where you are building your images.

   If you have `Singularity` installed on the same machine, you can **directly build, push and pull** on the same machine, without ever going online.

Singularity containers via a **Dockerfile**

2A. **Push** the `Docker` image to **`Docker Hub`** - tagging ...

```
user@host:~/exercises/ex09$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ex09                latest              8f11e10d5764        28 seconds ago      718MB
centos              7.4.1708            3afd47092a0e        7 months ago        197MB

user@host:~/exercises/ex09$ sudo docker tag ex09:latest blaurenczy/container_pipeline_tutorials:ex09

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
blaurenczy/container_pipeline_tutorials   ex09                8f11e10d5764        2 minutes ago       718MB
ex09                                      latest              8f11e10d5764        2 minutes ago       718MB
centos                                    7.4.1708            3afd47092a0e        7 months ago        197MB
```

Tagging is just *renaming* the image so that `Docker` knows **where** to push it.

Singularity containers via a **Dockerfile**

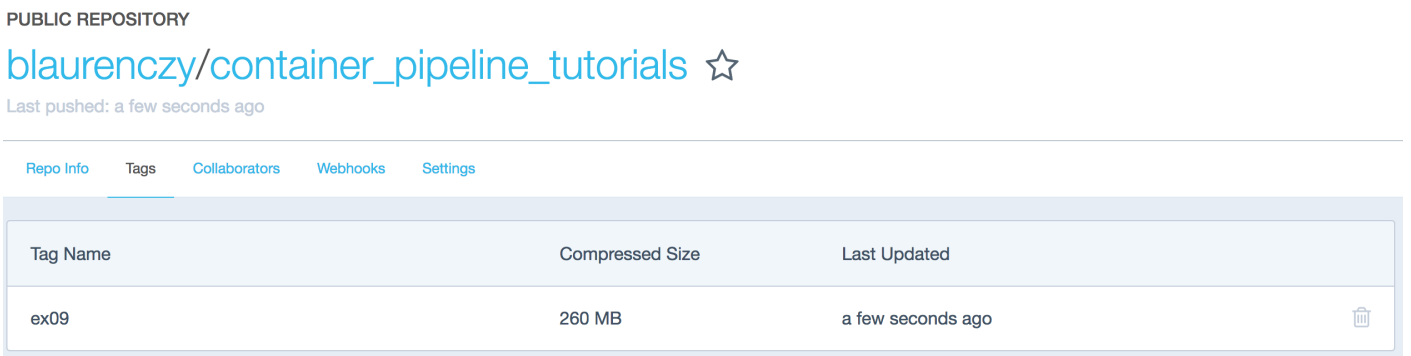
2A. **Push** the `Docker` image to **`Docker Hub`** - pushing ...

```
user@host:~/exercises/ex09$ sudo docker push blaurenczy/container_pipeline_tutorials:ex09
The push refers to repository [docker.io/blaurenczy/container_pipeline_tutorials]
c3133bfd5bdb: Preparing
2d852d7e8784: Preparing
b26bfe971df1: Preparing
775ec8ab46da: Preparing
8bc74bddfa2a: Preparing
d23effa2a292: Waiting
854329f71a7b: Waiting
129b697f70e9: Waiting
  denied: requested access to the resource is denied

user@host:~/exercises/ex09$ sudo docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't [...]
Username: blaurenczy
Password: 123456password
 Login Succeeded

user@host:~/exercises/ex09$ sudo docker push blaurenczy/container_pipeline_tutorials:ex09
The push refers to repository [docker.io/blaurenczy/container_pipeline_tutorials]
c3133bfd5bdb: Pushed
2d852d7e8784: Pushed
b26bfe971df1: Pushed
775ec8ab46da: Pushed
8bc74bddfa2a: Pushed
d23effa2a292: Pushed
854329f71a7b: Pushed
129b697f70e9: Mounted from library/centos
ex09: digest: sha256:11befba6d377ab53af7353a7674762c3676d5e4bd881fd0b5d411c5171af4743 size: 2200
```

ex09

Singularity containers via a **Dockerfile**

2A. **Push** the `Docker` image to **`Docker Hub`** - checking ...

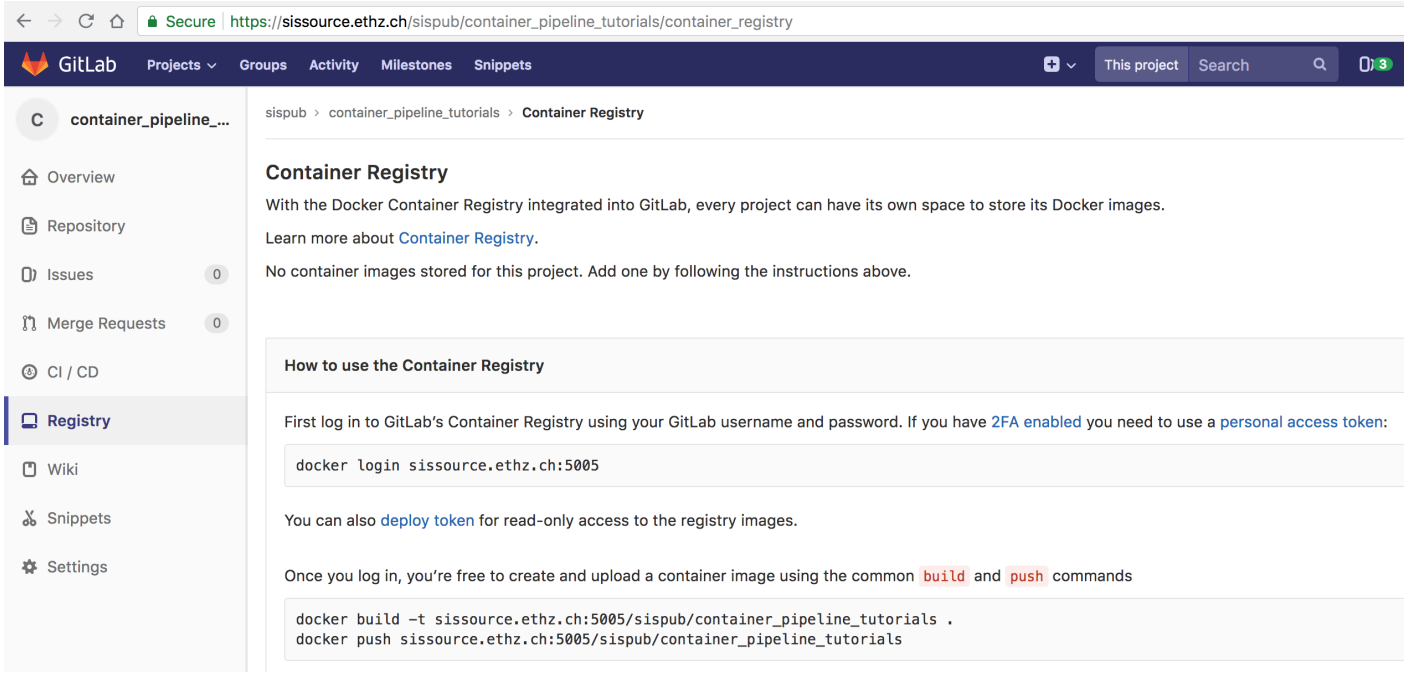


https://hub.docker.com/r/blaurenczy/container_pipeline_tutorials/tags

ex09

Singularity containers via a **Dockerfile**

2B. **Push** the `Docker` image to **the SIS `GitLab` registry** - the registry



Singularity containers via a **Dockerfile**

2B. **Push** the `Docker` image to **the SIS `GitLab` registry** - tagging ...

```
user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
blaurenczy/container_pipeline_tutorials   ex09                8f11e10d5764        6 minutes ago       718MB
ex09                                      latest              8f11e10d5764        6 minutes ago       718MB
centos                                    7.4.1708            3afd47092a0e        7 months ago        197MB

user@host:~/exercises/ex09$ sudo docker tag ex09:latest \
    sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09:latest

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                                  TAG                 IMAGE ID
blaurenczy/container_pipeline_tutorials                     ex09                8f11e10d5764
ex09                                                        latest              8f11e10d5764
sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09  latest              8f11e10d5764
centos                                                      7.4.1708            3afd47092a0e
```

ex09

Singularity containers via a **Dockerfile**

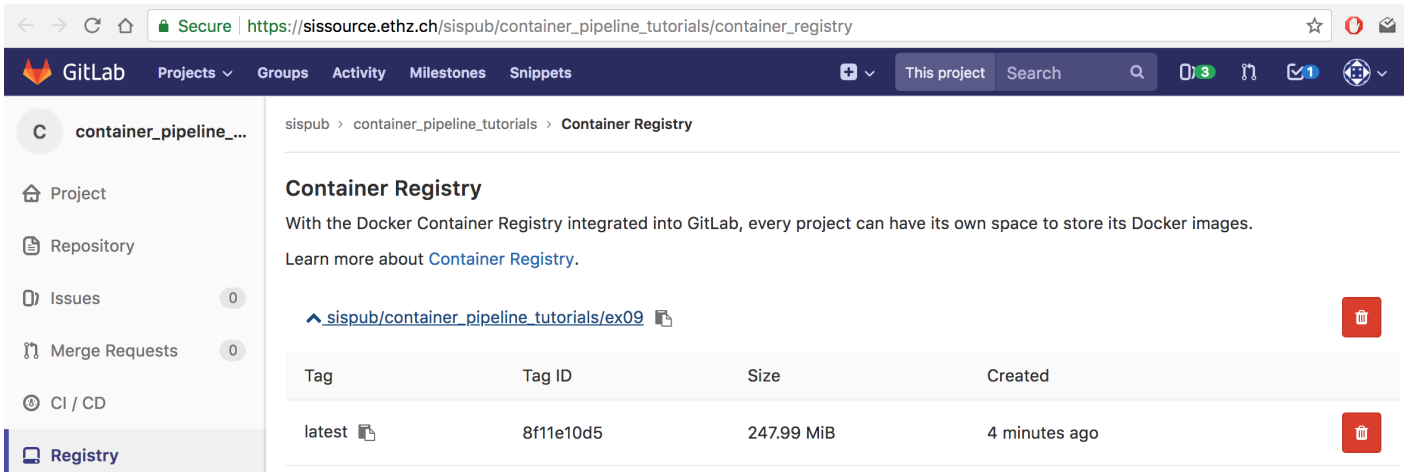
2B. **Push** the `Docker` image to **the SIS `GitLab` registry** - pushing ...

```
user@host:~/exercises/ex09$ sudo docker push \
    sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09:latest
The push refers to repository [sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09]
c3133bfd5bdb: Preparing

[...]

129b697f70e9: Waiting
  denied: requested access to the resource is denied

user@host:~/exercises/ex09$ sudo docker login sissource.ethz.ch:5005
Username: balazsl
Password: 7ru57n01
 Login Succeeded

user@host:~/exercises/ex09$ sudo docker push \
    sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09:latest
The push refers to repository [sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09]
c3133bfd5bdb: Pushed

[...]

129b697f70e9: Pushed
latest: digest: sha256:11befba6d377ab53af7353a7674762c3676d5e4bd881fd0b5d411c5171af4743 size: 2200
```

ex09

Singularity containers via a **Dockerfile**

2B. **Push** the `Docker` image to **the SIS `GitLab` registry** - checking ...



Singularity containers via a **Dockerfile**

2C. **Push** the `Docker` image to **a local `Docker` registry** - preparing ...

```
user@host:~/exercises/ex09$ sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
81033e7c1d6a: Pull complete
b235084c2315: Pull complete
c692f3a6894b: Pull complete
ba2177f3a70e: Pull complete
a8d793620947: Pull complete
Digest: sha256:672d519d7fd7bbc7a448d17956ebeefe225d5eb27509d8dc5ce67ecb4a0bce54
Status: Downloaded newer image for registry:2
df140832bf9481f436a27f95fa09e2be0bd833bf49e3aaeb0098adfec946b467

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                                  TAG                 IMAGE ID
sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09  latest              8f11e10d5764
blaurenczy/container_pipeline_tutorials                     ex09                8f11e10d5764
ex09                                                        latest              8f11e10d5764
registry                                                    2                   d1fd7d86a825
centos                                                      7.4.1708            3afd47092a0e

user@host:~/exercises/ex09$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
df140832bf94        registry:2          "/entrypoint.sh /etc…"   3 seconds ago       Up 6 seconds
```

ex09

Singularity containers via a **Dockerfile**

2C. **Push** the `Docker` image to **a local `Docker` registry** - tagging ...

```
user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                                      TAG                 IMAGE ID
sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09  latest              8f11e10d5764
blaurenczy/container_pipeline_tutorials                         ex09                8f11e10d5764
ex09                                                            latest              8f11e10d5764
registry                                                        2                   d1fd7d86a825
centos                                                          7.4.1708            3afd47092a0e

user@host:~/exercises/ex09$ sudo docker tag ex09:latest \
    localhost:5000/container_pipeline_tutorials/ex09:latest

user@host:~/exercises/ex09$ sudo docker images
REPOSITORY                                                      TAG                 IMAGE ID
blaurenczy/container_pipeline_tutorials                         ex09                8f11e10d5764
ex09                                                            latest              8f11e10d5764
localhost:5000/container_pipeline_tutorials/ex09                latest              8f11e10d5764
sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09  latest              8f11e10d5764
registry                                                        2                   d1fd7d86a825
centos                                                          7.4.1708            3afd47092a0e
```

Singularity containers via a **Dockerfile**

2C. **Push** the `Docker` image to **a local `Docker` registry** - pushing ...

```
user@host:~/exercises/ex09$ sudo docker push localhost:5000/container_pipeline_tutorials/ex09:latest
The push refers to repository [localhost:5000/container_pipeline_tutorials/ex09]
c3133bfd5bdb: Pushed
2d852d7e8784: Pushed
b26bfe971df1: Pushed
775ec8ab46da: Pushed
8bc74bddfa2a: Pushed
d23effa2a292: Pushed
854329f71a7b: Pushed
129b697f70e9: Pushed
latest: digest: sha256:11befba6d377ab53af7353a7674762c3676d5e4bd881fd0b5d411c5171af4743 size: 2200
```

Since we used minimal configuration for this local registry, there is no login needed. You can find more about the *registry container* at https://hub.docker.com/_/registry or https://docs.docker.com/registry

Singularity containers via a **Dockerfile**

2C. **Push** the `Docker` image to **a local `Docker` registry** - checking ...

```
user@host:~/exercises/ex09$ curl -s -X GET localhost:5000/v2/_catalog
{"repositories":["container_pipeline_tutorials/ex09"]}

user@host:~/exercises/ex09$ curl -s -X GET \
    localhost:5000/v2/container_pipeline_tutorials/ex09/tags/list | json_pp
{
   "name" : "container_pipeline_tutorials/ex09",
   "tags" : [
      "latest"
   ]
}
```

This means that our *local* `Docker` registry contains our image!

ex09

Singularity containers via a **Dockerfile**

3A. **Pull** the `Docker` image as a **`Singularity` container** (`Docker Hub`)

```
user@host:~/exercises/ex09$ singularity pull -n dockerhub-ex09.simg \
    docker://blaurenczy/container_pipeline_tutorials:ex09
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Docker image path: index.docker.io/blaurenczy/container_pipeline_tutorials:ex09
Cache folder set to /home/user/.singularity/docker

[...]

Exploding layer: sha256:dc609884789517476b0fd1d1b9765007b624f8563b2b0c889230f998f62a6b79.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ./container_pipeline_tutorials-ex09.simg
Cleaning up...
Done. Container is at: ./dockerhub-ex09.simg

user@host:~/exercises/ex09$ ls -alh
total 167M
drwxr-xr-x 1 user user  238 Jun 20  2018 .
drwxr-xr-x 1 user user  510 Jun 20 07:25 ..
drwxr-xr-x 1 user user  102 Jun  6 09:29 data
-rw-r--r-- 1 user user 1.1K Jun 20 08:26 Dockerfile
-rwxr-xr-x 1 user user 167M Jun 20  2018 dockerhub-ex09.simg
drwxr-xr-x 1 user user  136 Jun 20 07:14 scripts
-rw-r--r-- 1 user user  434 Jun 20 08:28 Snakefile
```

Singularity containers via a **Dockerfile**

3B. **Pull** the `Docker` image as a **`Singularity` container** (SIS `GitLab`)

```
user@host:~/exercises/ex09$ singularity pull -n sisgitlab-ex09.simg \
    docker://sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09:latest
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Docker image path: sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09:latest
Cache folder set to /home/user/.singularity/docker

[...]

WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ./sisgitlab-ex09.simg
Cleaning up...
Done. Container is at: ./sisgitlab-ex09.simg

user@host:~/exercises/ex09$ ls -alh
total 333M
drwxr-xr-x 1 user user  272 Jun 20  2018 .
drwxr-xr-x 1 user user  510 Jun 20 07:25 ..
drwxr-xr-x 1 user user  102 Jun  6 09:29 data
-rw-r--r-- 1 user user 1.1K Jun 20 08:26 Dockerfile
-rwxr-xr-x 1 user user 167M Jun 20  2018 dockerhub-ex09.simg
drwxr-xr-x 1 user user  136 Jun 20 07:14 scripts
-rwxr-xr-x 1 user user 167M Jun 20  2018 sisgitlab-ex09.simg
-rw-r--r-- 1 user user  434 Jun 20 08:28 Snakefile
```

Singularity containers via a **Dockerfile**

3C. **Pull** the `Docker` image as a **`Singularity` container** (local registry)

```
user@host:~/exercises/ex09$ singularity pull -n localhost-ex09.simg \
    docker://localhost:5000/container_pipeline_tutorials/ex09:latest
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Traceback (most recent call last):

[...]

  File "/usr/lib/python2.7/urllib2.py", line 1198, in do_open
    raise URLError(err)
  urllib2.URLError: <urlopen error [SSL: UNKNOWN_PROTOCOL] unknown protocol (_ssl.c:590)>
Cleaning up...
  ERROR: pulling container failed!
```

By default, our local registry does not have `https` enabled. Therefore we need to use the `SINGULARITY_NOHTTPS` variable to force `Singularity` to **not** use `https` when interacting with a Docker registry.

ex09

Singularity containers via a **Dockerfile**

3. **Pull** the `Docker` image as a **`Singularity` container** (local registry)

```
user@host:~/exercises/ex09$ SINGULARITY_NOHTTPS=true singularity pull -n localhost-ex09.simg \
    docker://localhost:5000/container_pipeline_tutorials/ex09:latest
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Docker image path: localhost:5000/container_pipeline_tutorials/ex09:latest

[...]

Building Singularity image...
Singularity container built: ./localhost-ex09.simg
Cleaning up...
Done. Container is at: ./localhost-ex09.simg

user@host:~/exercises/ex09$ ls -alh
total 499M
drwxr-xr-x 1 user user  306 Jun 20  2018 .
drwxr-xr-x 1 user user  510 Jun 20 07:25 ..
drwxr-xr-x 1 user user  102 Jun  6 09:29 data
-rw-r--r-- 1 user user 1.1K Jun 20 08:26 Dockerfile
-rwxr-xr-x 1 user user 167M Jun 20  2018 dockerhub-ex09.simg
-rwxr-xr-x 1 user user 167M Jun 20  2018 localhost-ex09.simg
drwxr-xr-x 1 user user  136 Jun 20 07:14 scripts
-rwxr-xr-x 1 user user 167M Jun 20  2018 sisgitlab-ex09.simg
-rw-r--r-- 1 user user  434 Jun 20 08:28 Snakefile
```

ex09

Singularity containers via a **Dockerfile**

We now created 3 `Singularity` containers starting with a `Dockerfile`.

These containers are similar in size and functionality, but they are **not identical** files.

```
user@host:~/exercises/ex09$ ls -al *.simg
-rwxr-xr-x 1 user user 174071839 Jun 20  2018 dockerhub-ex09.simg
-rwxr-xr-x 1 user user 174071839 Jun 20  2018 localhost-ex09.simg
-rwxr-xr-x 1 user user 174071839 Jun 20  2018 sisgitlab-ex09.simg

user@host:~/exercises/ex09$ sha1sum *.simg
5e430870127a08616d2265ef421ca827c5806e30  dockerhub-ex09.simg
2950f718552d4d8d57c32709501bc93deb2d4149  localhost-ex09.simg
697a722145582641f72c47e8e87e4da87292690d  sisgitlab-ex09.simg

user@host:~/exercises/ex09$ singularity inspect localhost-ex09.simg
{
    "container.maintainer": "Balazs Laurenczy (ETHZ)",
    "vendor": "CentOS",
    "name": "CentOS Base Image",
    "license": "GPLv2",
    "container.version": "ex09",
    "build-date": "20170911"
}
```

ex09

Singularity containers via a **Dockerfile**

They run the same pipeline in the same way inside the container:

```
user@host:~/exercises/ex09$ docker run localhost:5000/container_pipeline_tutorials/ex09:latest
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/20-08:15:42 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-08:15:42 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/20-08:15:42 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/20-08:15:42 | step1.py : Read input data: -2.0
#2018/06/20-08:15:42 | step1.py : Output data after processing: -8.0
#2018/06/20-08:15:42 | step1.py : Step 1 complete.
#2018/06/20-08:15:42 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-08:15:42 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/20-08:15:42 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/20-08:15:42 | step2.py : Read input data: -8.0
#2018/06/20-08:15:42 | step2.py : Output data after processing: -14.0
#2018/06/20-08:15:42 | step2.py : Step 2 complete.
Pipeline 'ex09' complete. Final output: -14.0
Complete log: /.snakemake/log/2018-06-20T081542.666721.snakemake.log
```

ex09

Singularity containers via a **Dockerfile**

They run the same pipeline in the same way inside the container:

```
user@host:~/exercises/ex09$ singularity run localhost-ex09.simg
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/20-08:17:31 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-08:17:31 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/20-08:17:31 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/20-08:17:31 | step1.py : Read input data: -2.0
#2018/06/20-08:17:31 | step1.py : Output data after processing: -8.0
#2018/06/20-08:17:31 | step1.py : Step 1 complete.
#2018/06/20-08:17:31 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-08:17:31 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/20-08:17:31 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/20-08:17:31 | step2.py : Read input data: -8.0
#2018/06/20-08:17:31 | step2.py : Output data after processing: -14.0
#2018/06/20-08:17:31 | step2.py : Step 2 complete.
Pipeline 'ex09' complete. Final output: -14.0
Complete log: /home/user/exercises/ex09/...
```

ex09

Singularity containers via a **Dockerfile**

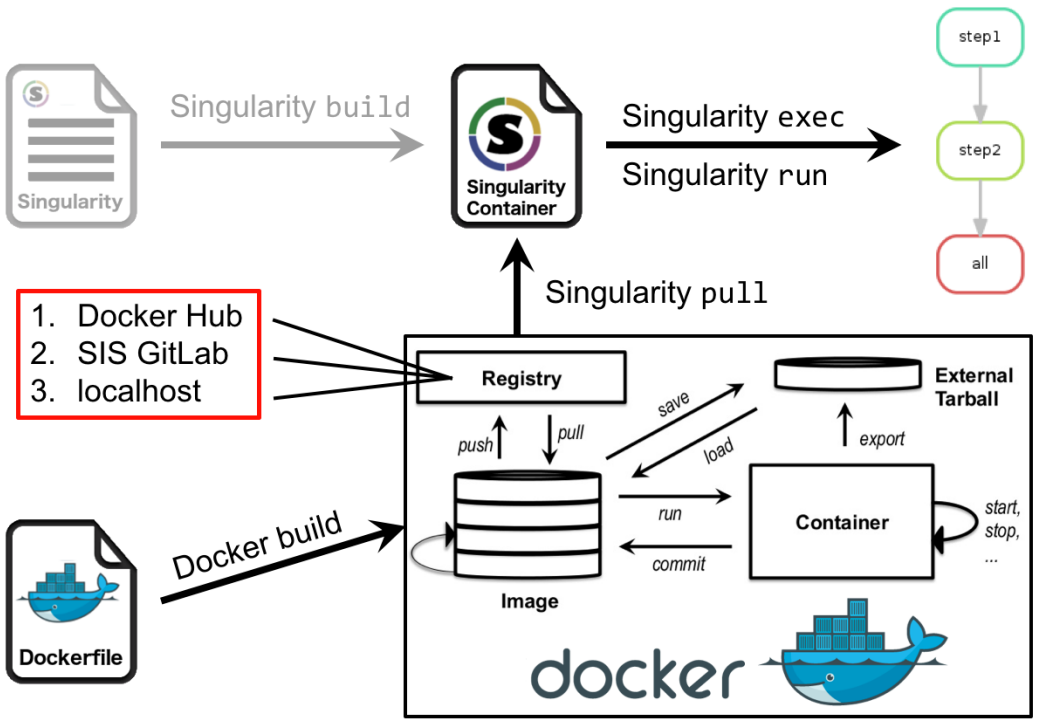
They run the same pipeline in the same way inside the container:

```
user@host:~/exercises/ex09$ singularity run dockerhub-ex09.simg
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1    step1
    1    step2
    3
#2018/06/20-09:30:59 | step1.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-09:30:59 | step1.py : Input: '/data/input.dat', output: '/data/output1.dat'
#2018/06/20-09:30:59 | step1.py : Reading input file: '/data/input.dat' ...
#2018/06/20-09:30:59 | step1.py : Read input data: -2.0
#2018/06/20-09:30:59 | step1.py : Output data after processing: -8.0
#2018/06/20-09:30:59 | step1.py : Step 1 complete.
#2018/06/20-09:30:59 | step2.py : Running on CentOS Linux 7.4.1708 Core
#2018/06/20-09:30:59 | step2.py : Input: '/data/output1.dat', output: '/data/output2.dat'
#2018/06/20-09:30:59 | step2.py : Reading input file: '/data/output1.dat' ...
#2018/06/20-09:30:59 | step2.py : Read input data: -8.0
#2018/06/20-09:30:59 | step2.py : Output data after processing: -14.0
#2018/06/20-09:30:59 | step2.py : Step 2 complete.
Pipeline 'ex09' complete. Final output: -14.0
Complete log: /home/user/exercises/ex09/...
```

ex09

Singularity containers via a **Dockerfile**

In summary, we have created a `Singularity` container from a `Dockerfile`, that runs the **same way** as a *pure* Singularity container. In addition, there are several options of *registry*.



source: https://www.slideshare.net/insideHPC/docker-for-hpc-in-a-nutshell

Singularity containers via a Dockerfile

Here is the example of the `exercise 2` using a `Dockerfile`: building ...

```
user@host:~/exercises/ex10$ sudo docker build \
    -t localhost:5000/container_pipeline_tutorials/ex10:latest .
Sending build context to Docker daemon  11.26kB
Step 1/11 : FROM centos:7.4.1708
 ---> 3afd47092a0e
Step 2/11 : RUN yum -y install epel-release     && yum-config-manager --enable epel
 ---> Using cache
 ---> 5150f5a44f2e

[...]

Step 11/11 : LABEL container.version="ex10"
 ---> Running in fa12a5210633
Removing intermediate container fa12a5210633
 ---> d0be5de1b2e1
Successfully built d0be5de1b2e1
Successfully tagged localhost:5000/container_pipeline_tutorials/ex10:latest

user@host:~/exercises/ex10$ sudo docker images
REPOSITORY                                                      TAG                 IMAGE ID
localhost:5000/container_pipeline_tutorials/ex10                latest              d0be5de1b2e1
sissource.ethz.ch:5005/sispub/container_pipeline_tutorials/ex09  latest              8f11e10d5764
blaurenczy/container_pipeline_tutorials                         ex09                8f11e10d5764
ex09                                                            latest              8f11e10d5764
localhost:5000/container_pipeline_tutorials/ex09                latest              8f11e10d5764
registry                                                        2                   d1fd7d86a825
centos                                                          7.4.1708            3afd47092a0e
```

Note that you can **directly label** the image with the `URL`. Also note how `Docker` is **nicely reusing the layers** that are similar to previous containers.

Singularity containers via a **Dockerfile**

Here is the example of the `exercise 2` using a `Dockerfile`: pushing ...

```
user@host:~/exercises/ex10$ sudo docker push localhost:5000/container_pipeline_tutorials/ex10:latest
The push refers to repository [localhost:5000/container_pipeline_tutorials/ex10]
68c0ad935b21: Pushed
82538eadbb51: Pushed
8bc74bddfa2a: Mounted from container_pipeline_tutorials/ex09
d23effa2a292: Mounted from container_pipeline_tutorials/ex09
854329f71a7b: Mounted from container_pipeline_tutorials/ex09
129b697f70e9: Mounted from container_pipeline_tutorials/ex09
latest: digest: sha256:0b71d07ed4f8f54346af5edefa2214b95a2cca53fac759dd67e25c2f5fcb5547 size: 1786

user@host:~/exercises/ex10$ curl -s -X GET localhost:5000/v2/_catalog
{"repositories":["container_pipeline_tutorials/ex10","container_pipeline_tutorials/ex09"]}

user@host:~/exercises/ex10$ curl -s -X GET \
    localhost:5000/v2/container_pipeline_tutorials/ex10/tags/list | json_pp
{
   "name" : "container_pipeline_tutorials/ex10",
   "tags" : [
      "latest"
   ]
}
```

Note that the existing layers are **not pushed again** to a registry if they already exist there.

ex10

Singularity containers via a **Dockerfile**

Here is the example of the `exercise 2` using a `Dockerfile`: pulling ...

```
user@host:~/exercises/ex10$ SINGULARITY_NOHTTPS=true singularity pull -n localhost-ex10.simg \
    docker://localhost:5000/container_pipeline_tutorials/ex10:latest
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Docker image path: localhost:5000/container_pipeline_tutorials/ex10:latest
Cache folder set to /home/user/.singularity/docker
[3/3] |===================================| 100.0%
Importing: base Singularity environment
Exploding layer: sha256:18b8eb7e7f013bd0e585c8c88c6f24599a5db1cbbe72e54246004dad662582c1.tar.gz

[...]

Exploding layer: sha256:41ae82b8b8edb92d7a46a507121076798f687c757338e7feee90ca4c8a39a7c3.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...

user@host:~/exercises/ex10$ ls -alh
total 167M
drwxr-xr-x 1 user user  272 Jun 20  2018 .
drwxr-xr-x 1 user user  510 Jun 20 07:25 ..
drwxr-xr-x 1 user user  102 Jun  6 10:12 dataset01
drwxr-xr-x 1 user user  102 Jun  6 10:12 dataset02
-rw-r--r-- 1 user user  942 Jun 20 08:29 Dockerfile
-rwxr-xr-x 1 user user 167M Jun 20  2018 localhost-ex10.simg
drwxr-xr-x 1 user user  136 Jun 20 10:05 scripts
-rw-r--r-- 1 user user  435 Jun 20 10:06 Snakefile
```

ex10

Singularity containers via a **Dockerfile**

Here is the example of the `exercise 2` using a `Dockerfile`: running ...

```
user@host:~/exercises/ex10$ singularity run localhost-ex10.simg
Building DAG of jobs...
  MissingInputException in line 7 of /pipeline/Snakefile:
  Missing input files for rule step1:
  /data/input.dat

user@host:~/exercises/ex10$ singularity run -B dataset01:/data localhost-ex10.simg
Building DAG of jobs...
Using shell: /usr/bin/bash

[...]

Pipeline 'ex10' complete. Final output: -14.0
Complete log: /home/user/exercises/ex10/...

user@host:~/exercises/ex10$ ls -alh dataset01/
total 12K
drwxr-xr-x 1 user user 170 Jun 20  2018 .
drwxr-xr-x 1 user user 272 Jun 20  2018 ..
-rw-r--r-- 1 user user   2 Jun  6 09:37 input.dat
-rw-r--r-- 1 user user   4 Jun 20 10:08 output1.dat
-rw-r--r-- 1 user user   4 Jun 20 10:08 output2.dat
```

Singularity containers via a **Dockerfile**

Here is the example of the `exercise 2` using a `Dockerfile`: building ...

```
user@host:~/exercises/ex10$ sudo docker run -v $PWD/dataset02:/data \
    localhost:5000/container_pipeline_tutorials/ex10:latest
Building DAG of jobs...
Using shell: /usr/bin/bash

[...]

Pipeline 'ex10' complete. Final output: 40.0
Complete log: /.snakemake/log/2018-06-20T102249.392181.snakemake.log

user@host:~/exercises/ex10$ ls -alh dataset02
total 12K
drwxr-xr-x 1 user user 170 Jun 20  2018 .
drwxr-xr-x 1 user user 272 Jun 20  2018 ..
-rw-r--r-- 1 user user   1 Jun  6 09:37 input.dat
-rw-r--r-- 1 user user   4 Jun 20 10:22 output1.dat
-rw-r--r-- 1 user user   4 Jun 20 10:22 output2.dat

user@host:~/exercises/ex10$ singularity run -B dataset02:/data localhost-ex10.simg
Building DAG of jobs...
Using shell: /usr/bin/bash
Job counts:
    count    jobs
    1    all
    1
Pipeline 'ex10' complete. Final output: 40.0
Complete log: /home/user/exercises/ex10/...
```

ex09 & ex10

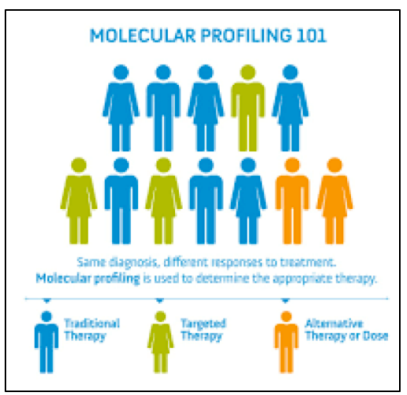
Singularity containers via a **Dockerfile**

In summary:

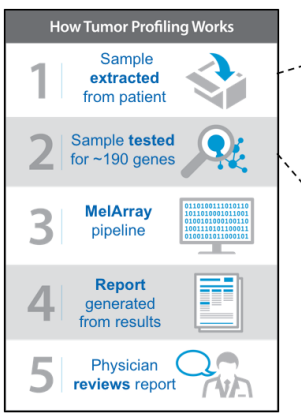
- You can use `Dockerfiles` if you want to keep **compatibility**
- The build process is however *longer*, more *complex* and requires a `Docker` registry
- Writing `Singularity` recipes and `Dockerfile` is **largely similar**, once you have figured out the installation process
- There are however a few differences to keep in mind:
  - `Singularity` produces *files*, whereas `Docker` does not
  - `Docker` installs & runs as **root**, whereas `Singularity` does not run as root by default
  - `Docker` has a better **caching** system during the build

A real use case: **MelArray**

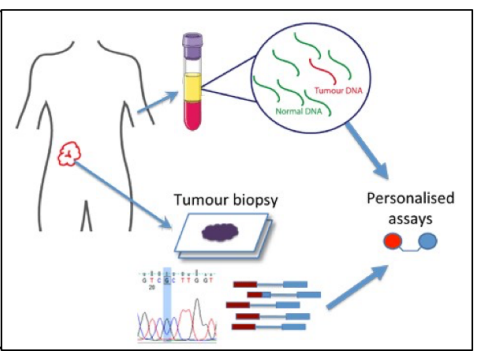
- *MelArray* is a **tumor profiling** pipeline for melanoma patients
- **Idea:** if we know better what **exact type of tumor** patients have, we can give them a therapy or **drug that is more likely to work**
- A panel of about **190 genes** is used in *MelArray* to do the profiling of the tumor



source: https://sarahcannon.com



modified from: https://www.carislifesciences.com



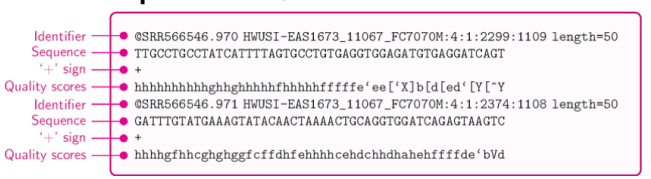
source: https://www.houstonmethodist.org

A real use case: **MelArray**

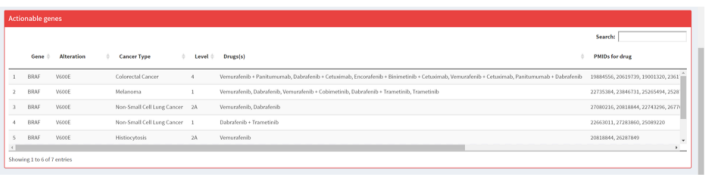
Challenges:

- Convert a bash script based pipeline into something more manageable => **Workflow manager**
- Make the pipeline **portable**, as it will need to run both in the dermatology and pathology departments of UniSpital Zürich => **Containers**

Input: FASTQ files



Output: report



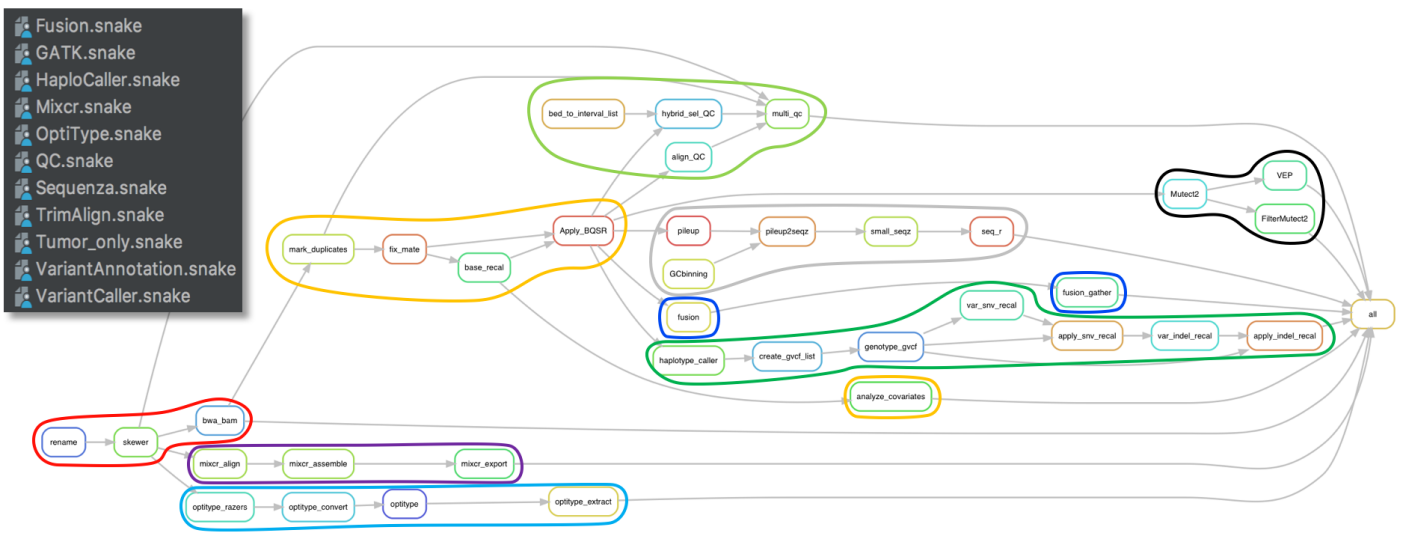
The *MelArray* pipeline

The *MelArray* pipeline *(~37 steps / tools)*:

The *MelArray* pipeline

The *MelArray* pipeline split into **subworkflows**:

Subworkflows:
TrimAlign, Mixcr, OptiType, GATK, HaploCaller, QC, Fusion, Sequenza, VariantAnnotation & VariantCaller



The *MelArray* pipeline

Some *features* of this pipeline:

- `Snakemake` based pipeline
- Organisation into *subworkflows*
- Controlled by a *main* configuration file
- Can be run *inside*, *outside* or *without* its **container**

These features, adding a lot of **flexbility** do however come with the cost of added **complexity**.

The *MelArray* pipeline

The **configuration file** controls:

- File paths
- Parameters for each tool (arguments, memory allocation, etc.)
- Orchestration strategy

```
main:

    # defines how the pipeline should be run:
    #   'host': the pipeline runs on the host, without using any containers.
    #   'run_outside_single': the pipeline runs on the host, but using a container
    #   'run_inside_single': the pipeline runs inside a single container
    run_mode: "run_outside_single"

    # switches for activating the different subworkflows
    subworkflows:
    - "GATK"
    - "TrimAlign"

    [...]

path:

    # root path for the data folders inside the container, if any used. If not,
    #   this is replace by "host_data_root"
    data_root: "/data"
    # path to the folder where the original fastq files are stored
    fastq: "__DATA_ROOT__/fastq"
    # path to the folder where the renamed fastq files are stored
    fastq_renamed_root: "__DATA_ROOT__/fastq_renamed"
```

The *MelArray* pipeline

As an example, this is one step of a subworkflow of *MelArray*:

```
# trimming step
rule skewer:
    input:
        R1 = paths['renamed_fastq_R1'],
        R2 = paths['renamed_fastq_R2']
    output:
        trimmed_R1 = temp(paths['trimmed_R1']),
        trimmed_R2 = temp(paths['trimmed_R2']),
        trimmed_log = paths['trimmed_log']
    log:
        trimmed_out = paths['trimmed_out']
    shell:
        # wrap the whole command in one string
        (
            # append the Singularity container call (if needed)
            SINGULARITY_CMD +
            # create the command line using the parameters contained in config['software']['skewer']
            'skewer -t {n_threads} -m {trimming_mode} '.format(**soft['skewer']) +
                # specify some more parameters from the config
                '-q {quality_threshold} -z --quiet '.format(**soft['skewer']) +
                # add the output to be in the 'trimmed' folder and to have the [ID] prefix
                '-o {trimmed_root_with_id} '.format(**rule_paths) +
                # add input paths that Snakemake will automatically replace
                '{renamed_fastq_R1} {renamed_fastq_R2} '.format(**rule_paths) +
                # dump all standard and error output to the log file (redir. outside of the container)
                '&> {trimmed_out}'.format(**paths)
        # replace all instances of '{id}' with '{wildcards.id}'
        ).replace('{id}', '{wildcards.id}')
```

ex11

Snakemake and Singularity

In the most recent versions of `Snakemake`, there is some level of **integration for `Singularity`** (see Snakemake's documentation for more information):

```
rule step1:
     input:
         'data/input.dat'
     output:
         'data/output1.dat'
     singularity:
         'docker://blaurenczy/container_pipeline_tutorials:ex11'
     shell:
         'step1.py {input} {output}'
```

You can then executing `Snakemake` with:

```
snakemake --use-singularity
```

Snakemake and Singularity

Here is a testing **without** the container, and then the **building** and **pushing** of the required container to `Docker Hub`:

```
user@host:~/exercises/ex11$ snakemake
Building DAG of jobs...

[...]

  /bin/bash: step1.py: command not found
     Error in rule step1:
         jobid: 2
         output: data/output1.dat

[...]

user@host:~/exercises/ex11$ sudo docker build -t blaurenczy/container_pipeline_tutorials:ex11 .
[...]
Successfully built 2500a63a3a78
Successfully tagged blaurenczy/container_pipeline_tutorials:ex11

user@host:~/exercises/ex11$ sudo docker push blaurenczy/container_pipeline_tutorials:ex11
[...]
ex11: digest: sha256:1c309efe8023162b2aa594547a403facc1f349204787edfef693cb4f258cdba4 size: 1786
```

ex11

Snakemake and Singularity

```
user@host:~/exercises/ex11$ snakemake --use-singularity
Building DAG of jobs...
Pulling singularity image docker://blaurenczy/container_pipeline_tutorials:ex11.

[...]

rule step1:
    input: data/input.dat
    output: data/output1.dat
    jobid: 2

Activating singularity image /home/user/exercises/ex11/...
#2018/06/20-03:39:18 | step1.py : Running on CentOS Linux 7.4.1708 Core

[...]

Pipeline 'ex11' complete. Final output: -14.0
Finished job 0.
3 of 3 steps (100%) done
Complete log: /home/user/exercises/ex11/...

user@host:~/exercises/ex11$ ls -alh data
total 12K
drwxr-xr-x 1 user user 170 Jun 20  2018 .
drwxr-xr-x 1 user user 272 Jun 20 15:38 ..
-rw-r--r-- 1 user user   2 Jun 20 10:34 input.dat
-rw-r--r-- 1 user user   4 Jun 20 15:52 output1.dat
-rw-r--r-- 1 user user   5 Jun 20 15:52 output2.dat
```

The *MelArray* **container**: software content

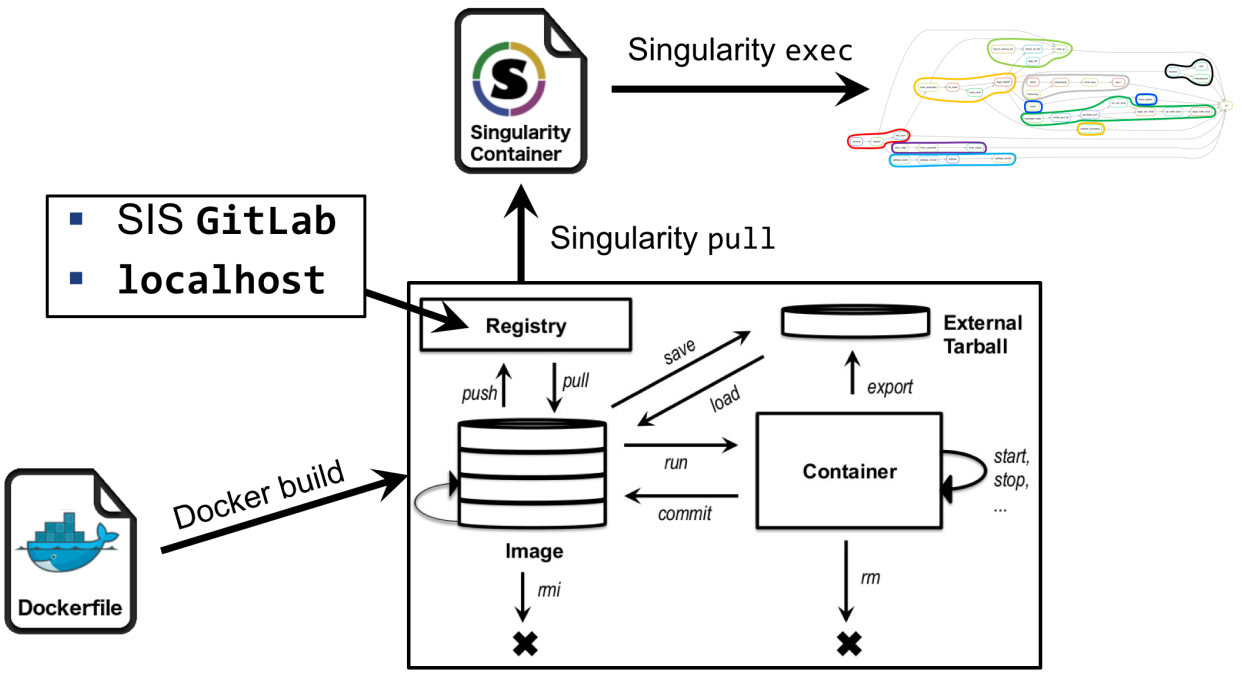
```
# Python 2 is needed for some processing steps
ENV PYTHON2_VERSION "2.7.11"
# Python 3 is needed for Snakemake
ENV PYTHON3_VERSION "3.6.1"
# R is needed for generating plots with GATK
ENV R_VERSION "3.4.4"
# Perl 5 version
ENV PERL_VERSION "5.22.2"
# Java needed for GATK
ENV JAVA_VERSION "1.8_131"

# Snakemake is needed to run the pipeline
ENV SNAKEMAKE_VERSION "3.11.2"
# Pyaml is a YAML file readed and it is needed for loading the configuration file
ENV PYAML_VERSION "16.12.2"
# MultiQC generates the quality control HTML page
ENV MULTIQC_VERSION "0.9"

# skewer is a trimming tool
ENV SKEWER_VERSION "0.2.2"
# BWA is an alignment software package
ENV BWA_VERSION "0.7.13"
# GATK is a toolkit to do variant discovery and genotyping.
ENV GATK_VERSION "4.0.1.1"
# samtools is needed for bam manipulation and pileup file
ENV SAMTOOLS_VERSION "1.3.1"
# MiXCR is an aligner / assembler
ENV MIXCR_VERSION "2.1.3"

[...]
```

The *MelArray* container: *Dockerfile*

```
[...]

RUN echo " - Installing Python 3 ($PYTHON3_VERSION) ..." \
    && cd /usr/local/src \
    && wget -O "Python-$PYTHON3_VERSION.tar.xz" \
    "http://www.python.org/ftp/python/$PYTHON3_VERSION/Python-$PYTHON3_VERSION.tar.xz" \
    && tar xf "Python-$PYTHON3_VERSION.tar.xz" \
    && rm -rf "Python-$PYTHON3_VERSION.tar.xz" \
    && cd "Python-$PYTHON3_VERSION" \
    && ./configure \
    && make \
    && make install \
    && cd .. && rm -rf "Python-$PYTHON3_VERSION"

[...]

RUN echo " --- Installing Perl modules ..." \
    && curl -sL https://cpanmin.us | "/usr/local/src/perl-$PERL_VERSION/bin/perl" - --notest \
    -l "/usr/local/src/perl-$PERL_VERSION" App::cpanminus JSON::Parse LWP LWP::Simple \
    Archive::Extract Archive::Tar Archive::Zip CGI DBI Time::HiRes DBD::mysql Encode \
    File::Copy::Recursive Perl::OSType Module::Metadata Statistics::Lite Tie::Autotie \
    Tie::IxHash Log::Log4perl FindBin::Real Getopt::Long Catalyst::Runtime Catalyst::Devel \
    List::Util Test::XML::Simple Test::XPath IO::String Bio::Perl PerlIO::gzip \
    Set::IntervalTree version

[...]

# specifies what to execute when the container is started with "run"
ENTRYPOINT ["snakemake"]

[...]
```

The *MelArray* **container**: building process

The *MelArray* container is **built** from a `Dockerfile`, **pushed** to either the SIS `GitLab` or to a local registry, and then **pulled** to a `Singularity` container.



source: https://www.slideshare.net/insideHPC/docker-for-hpc-in-a-nutshell

A note about container sizes

Consider the two following `Dockerfile` snippets:

`MelArray v6.0`

```
RUN echo " - Installing Python 3 ($PYTHON3_VERSION) ..."
RUN wget -O "/usr/local/src/Python-$PYTHON3_VERSION.tar.xz"\
    "http://www.python.org/ftp/python/$PYTHON3_VERSION/Python-$PYTHON3_VERSION.tar.xz"
RUN cd "/usr/local/src" && tar xf "Python-$PYTHON3_VERSION.tar.xz"
RUN cd "/usr/local/src" && rm -rf "Python-$PYTHON3_VERSION.tar.xz"
RUN cd "/usr/local/src/Python-$PYTHON3_VERSION" && ./configure
RUN cd "/usr/local/src/Python-$PYTHON3_VERSION" && make
RUN cd "/usr/local/src/Python-$PYTHON3_VERSION" && make install
RUN cd "/usr/local/src" && rm -rf "Python-$PYTHON3_VERSION"
```

`MelArray v6.2`

```
RUN echo " - Installing Python 3 ($PYTHON3_VERSION) ..." \
    && cd "/usr/local/src" \
    && wget -O "Python-$PYTHON3_VERSION.tar.xz" \
    "http://www.python.org/ftp/python/$PYTHON3_VERSION/Python-$PYTHON3_VERSION.tar.xz" \
    && tar xf "Python-$PYTHON3_VERSION.tar.xz" \
    && rm -rf "Python-$PYTHON3_VERSION.tar.xz" \
    && cd "Python-$PYTHON3_VERSION" \
    && ./configure \
    && make \
    && make install \
    && cd .. && rm -rf "Python-$PYTHON3_VERSION"
```

A note about container sizes

Deleting source files and archives helps **reducing the container's size**, but only if the files are removed **in the same layer** where they are downloaded.

- ~ 2GB reduction according to Docker (25% reduction):

```
user@host:~/exercises/MelArray/container$ sudo docker images
REPOSITORY             TAG        IMAGE ID        CREATED          SIZE
balazsl/usz/melarray   6.2        d239ce1e5cc1    3 minutes ago    6.41 GB
balazsl/usz/melarray   6.0        6b1fd02eb717    4 days ago       8.57 GB
```

- ~ 7GB reduction in the Singularity file (33% reduction):

```
user@host:~/exercises/MelArray/container$l s -alh *.img
-rwxr-xr-x 1 wid wheel 21G May 25 10:52 melarray-6.0_merge.img
-rwxr-xr-x 1 wid wheel 14G May 29 22:47 melarray-6.2.img
```

Docker, Singularity, Shifter

There are many container technologies available today.

-  `Singularity`
- 
-  `Shifter`

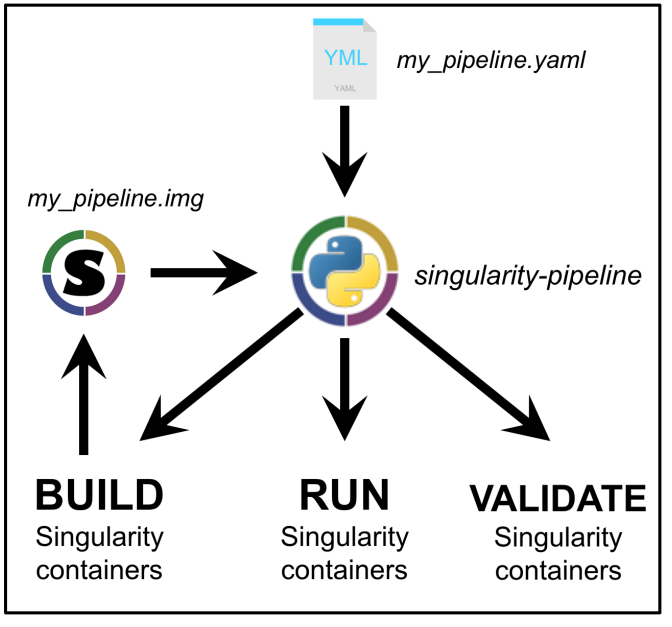
For more information, see
https://tin6150.github.io/psg/blogger_container_hpc.html

singularity-pipeline runner module

The `singularity-pipeline` is a `Python` module to help you **build**, **validate** and **run** your containerised pipelines

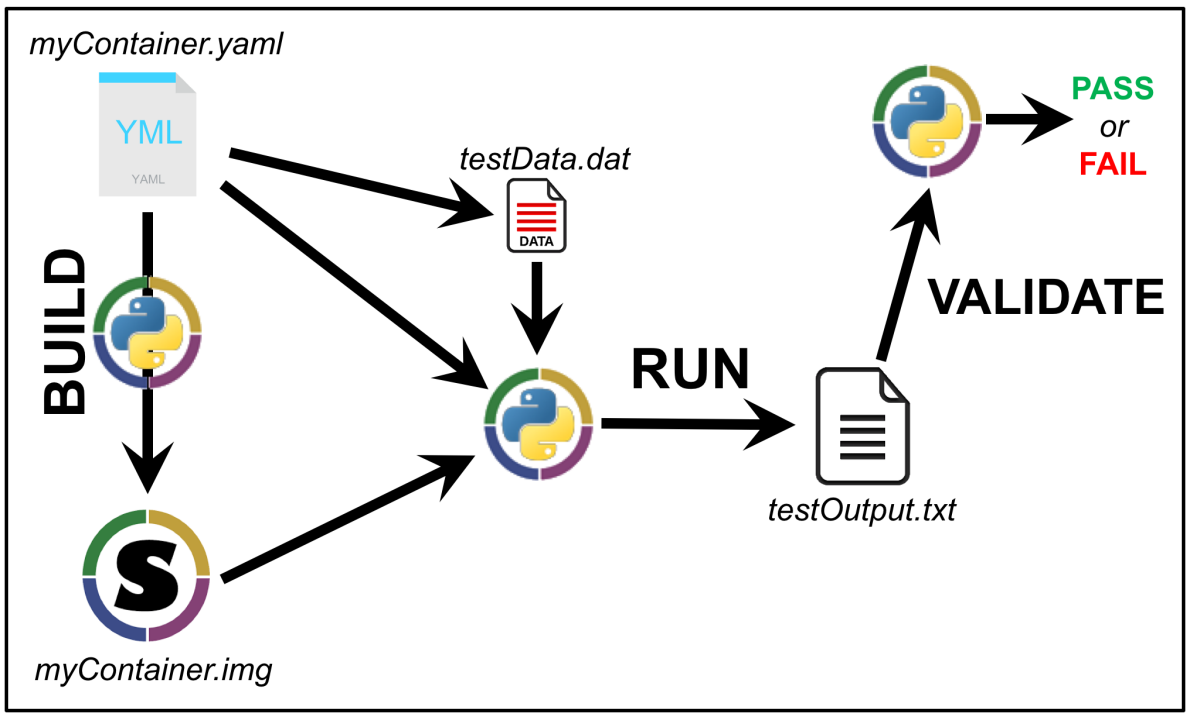
"hello world pipeline" YAML description file

```yaml
name: hello_world_pipeline
version: 1
author: Balazs Laurenczy
author_org: ETHZ
substitutions:
  name: "hello_world_pipeline"
binds:
  - "/usr/lib64:/usr/lib64"
build:
  type: pull
  source: docker://blaurenczy/hello_world:latest
run:
  commands:
    - "{exec} python3 test.py > {name}.out 2> {name}.err"
test:
  validate_commands:
    - "[[ \"$(md5sum {name}.out | cut -f1 -d ' ')\" \
      == \"06c19b37d27dfda293492a4459fe3bc3\" ]] && echo 0"
```



singularity-pipeline runner module

The `singularity-pipeline` is a `Python` module to help you **build**, **validate** and **run** your containerised pipelines



Summary

- Combining pipeline & containers
- *Portability* & *reproducibility* on different levels
- Different *orchestration strategies*
- `Docker` and `Singularity`
- *MelArray* use case