Introduction to Containers for Science

Hans Fangohr
European XFEL GmbH
University of Southampton

September 9, 2018

Contents

1 Introduction to Containers for Science

This chapter motivates why containers are a powerful tool for science. If you want to jump straight into the examples, jump to the `Hello world` container in the next chapter.

1.1 Computational science challenge: unusual installation requirements

Computational science projects often relay on unusual libraries, or rely on particular versions of a library, or a combination of particular libraries. Often, these tools have only been developed to be used on a particular operating system (say SuSE, Debian, Ubuntu, ...) and depend on the packages one of the versions of these distribution provides.

This makes the code difficult to port (to other machines), to use in the future (if operating system upgrades are required and the updated operating system breaks one of the requirements of the tool, and difficult to re-use by others.

Containers offer a practical way of allowing installation of such tools inside the container image, execution inside the container, and tus portability (as the container can be run on other machines) and also the ability to run the container in the future.

1.2 Computational science challenge: reproducibility

Reproducibility of computational science has become a widely debated topic. We have made good progress with, for example, the use of versioning tools for software (so that it is possible to go back to particular versions of code that may have been used for a particular study). Github and bitbucket, for example, have helped researches to have cloud hosted version control of their research software.

One area of computational science that in my view is not sufficiently addressed at the moment is that of the computational environment: it is not enough to have the source code, we also need to know (i) how to install it and (ii) which libraries the code may have used to compute a particular result.

Containers offer a good way of documenting this. The file that species how the container image should be built (the `Dockerfile` in the coming chapters), documents *exactly* the compute environment in which the software should execute.

1.3 Computational science challenge: execution of code on cluster

High Performance Computing installations (ofter a Linux cluster) are generally maintained by specialist IT staff, and the scientists are users on the system. As such, the scientists generally have no administrator rights, and IT staff install software on the cluster on behalf of the users. As alluded to above, the users may have quite specific requirements on the particular versions of tools they need, leading to tools to manage different versions of software (`module load ...`) and lengthy processes to install software: for example the operating system is generally fixed, and thus occasionally it is not possible (within the staff time available) to port a tool onto an supercomputer for the user.

Containers have a solution to this problem: the user can provide their own container with the installed software, and thus completely (i) elimenate the need to have multiple versions of libraries of the supercomputer and (ii) reduce IT staff time required to install the software. In the following chapters, I use Docker as an application to run containers. In the High Performance Computing community other container applications are likely to be more widely spread, such as for example 'singularity'.

1.4 So what are these containers?

Containers are a slim version of virtual machines: they allow us to use an operating system of our choice, install software of our choice, and execute this software 'inside' the container. This can be done seamlessly, i.e. it is possible to just run one command inside the container and it doesn't take long to start the container (sub seconds). It is possible to execute multiple different containers on the same machine to execute tasks in their respectively preferred compute environment.

The subsequent chapters give more details on the use of containers, creation of (container) images, mounting of file systems, and sharing of ports between containers and host.

For the quick tour here, we assume the reader has installed Docker [1] on their computer.

[1] http://docker.com

1.5 This document

The document will grow over time. It is available:

- sources at http://github.com/fangohr/containers-for-science
- notebooks (one for each chapter at https://github.com/fangohr/containers-for-science/tree/master/notebooks)
- pdf version at https://github.com/fangohr/containers-for-science/tree/master/pdf/containers-for-science.pdf
- html version at https://github.com/fangohr/containers-for-science/tree/master/html/index.html

The pdf and html versions will be updated from the source occasionally.

If you know the Jupyter notebook, you may want to clone the repository, and execute the notebooks in the `notebooks` chapter to walk through the commands interactively.

```
In [ ]:
```

2 Hello World programme in container

2.1 Example 1: Executing a bash command inside an Ubuntu container

First, we will *pull* the latest Ubuntu (long term support) image from dockerhub.com. If you have carried out this operation before, it will be very quick, otherwise it depends a little on the bandwidth of your Internet connection:

```
In [1]: # NBVAL_IGNORE_OUTPUT
        !docker pull ubuntu:latest
```

```
latest: Pulling from library/ubuntu
Digest: sha256:de774a3145f7ca4f0bd144c7d4ffb2931e06634f11529653b23eba85aef8e378
Status: Downloaded newer image for ubuntu:latest
```

Second, we can now execute commands inside that container. For example, the file `/etc/issue` contains information about the version of the Linux distribution. We use the unix command `cat` to conCATenate the contents of that file to the standard output:

```
In [2]: !docker run ubuntu:latest cat /etc/issue
```

```
Ubuntu 18.04.1 LTS \n \l
```

As you may guess, the structure of the docker command is - `docker`: the Docker application on your host system - `run`: the command for the docker application - we want to run an image - `ubuntu:latest` : the image that we want to start - all remaining arguments (here `cat /etc/issue`) are being executed inside the running container

One way of 'writing' a container hello world program would be this:

```
In [3]: !docker run ubuntu:latest bash -c "echo 'Hello World'"

Hello World
```

We have just executed the "echo 'Hello World'" command inside the Ubuntu Linux distribution container! We can check how large the image of the container is:

```
In [9]: !docker images ubuntu:latest

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              cd6d8154f1e1        2 days ago          84.1MB
```

3 Creating our own docker container

To create our own container images, we need to write a file that defines what should be inside the environment. The contents are similar to a bash script, although the syntax is determined by the application to run the container. Here we use docker, and the file is (by default) called `Dockerfile`:

```
In [1]: %%file Dockerfile
        FROM ubuntu:latest

        RUN apt-get update
        RUN apt-get install -y cowsay

        # cowsay installs into /usr/games. Make avaible in PATH:
        RUN ln -s /usr/games/cowsay /usr/local/bin

Overwriting Dockerfile
```

Build a docker image based on the `Dockerfile` above (in the current directory). Call it `cowimage`.

```
In [2]: #NBVAL_IGNORE_OUTPUT
        !docker build -t cowimage .

Sending build context to Docker daemon    150kB
Step 1/4 : FROM ubuntu:latest
 ---> cd6d8154f1e1
Step 2/4 : RUN apt-get update
 ---> Running in 94ff97dc933d
```

```
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/universe Sources [11.5 MB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/universe Sources [17.4 kB]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [203 kB]
Get:8 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [1363 B]
Get:9 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [69.0 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:13 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/universe Sources [70.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [226 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [401 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [3925 B]
Get:18 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [2807 B]
Fetched 25.9 MB in 4s (6374 kB/s)
Reading package lists...
Removing intermediate container 94ff97dc933d
 ---> 864c48282361
Step 3/4 : RUN apt-get install -y cowsay
 ---> Running in 74d8be92f3c3
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libgdbm-compat4 libgdbm5 libperl5.26 libtext-charwidth-perl netbase perl
  perl-modules-5.26
Suggested packages:
  filters cowsay-off gdbm-l10n perl-doc libterm-readline-gnu-perl
  | libterm-readline-perl-perl make
The following NEW packages will be installed:
  cowsay libgdbm-compat4 libgdbm5 libperl5.26 libtext-charwidth-perl netbase
  perl perl-modules-5.26
0 upgraded, 8 newly installed, 0 to remove and 1 not upgraded.
Need to get 6567 kB of archives.
After this operation, 41.7 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 perl-modules-5.26 all 5.26.1
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 libgdbm5 amd64 1.14.1-6 [26.0 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 libgdbm-compat4 amd64 1.14.1-6 [6084
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libperl5.26 amd64 5.26.1-6ub
Get:5 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 perl amd64 5.26.1-6ubuntu0.2
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 libtext-charwidth-perl amd64 0.04-7.
Get:7 http://archive.ubuntu.com/ubuntu bionic/main amd64 netbase all 5.4 [12.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic/universe amd64 cowsay all 3.03+dfsg2-4 [17.7 kB
debconf: delaying package configuration, since apt-utils is not installedFetched 6567 kB in 1
Selecting previously unselected package perl-modules-5.26.
(Reading database ... 4037 files and directories currently installed.)
```

```
Preparing to unpack .../0-perl-modules-5.26_5.26.1-6ubuntu0.2_all.deb ...
Unpacking perl-modules-5.26 (5.26.1-6ubuntu0.2) ...
Selecting previously unselected package libgdbm5:amd64.
Preparing to unpack .../1-libgdbm5_1.14.1-6_amd64.deb ...
Unpacking libgdbm5:amd64 (1.14.1-6) ...
Selecting previously unselected package libgdbm-compat4:amd64.
Preparing to unpack .../2-libgdbm-compat4_1.14.1-6_amd64.deb ...
Unpacking libgdbm-compat4:amd64 (1.14.1-6) ...
Selecting previously unselected package libperl5.26:amd64.
Preparing to unpack .../3-libperl5.26_5.26.1-6ubuntu0.2_amd64.deb ...
Unpacking libperl5.26:amd64 (5.26.1-6ubuntu0.2) ...
Selecting previously unselected package perl.
Preparing to unpack .../4-perl_5.26.1-6ubuntu0.2_amd64.deb ...
Unpacking perl (5.26.1-6ubuntu0.2) ...
Selecting previously unselected package libtext-charwidth-perl.
Preparing to unpack .../5-libtext-charwidth-perl_0.04-7.1_amd64.deb ...
Unpacking libtext-charwidth-perl (0.04-7.1) ...
Selecting previously unselected package netbase.
Preparing to unpack .../6-netbase_5.4_all.deb ...
Unpacking netbase (5.4) ...
Selecting previously unselected package cowsay.
Preparing to unpack .../7-cowsay_3.03+dfsg2-4_all.deb ...
Unpacking cowsay (3.03+dfsg2-4) ...
Setting up perl-modules-5.26 (5.26.1-6ubuntu0.2) ...
Setting up libgdbm5:amd64 (1.14.1-6) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Setting up libtext-charwidth-perl (0.04-7.1) ...
Setting up libgdbm-compat4:amd64 (1.14.1-6) ...
Setting up netbase (5.4) ...
Setting up libperl5.26:amd64 (5.26.1-6ubuntu0.2) ...
Setting up perl (5.26.1-6ubuntu0.2) ...
Setting up cowsay (3.03+dfsg2-4) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Removing intermediate container 74d8be92f3c3
 ---> 951b8495700a
Step 4/4 : RUN ln -s /usr/games/cowsay /usr/local/bin
 ---> Running in 7e1aeea049be
Removing intermediate container 7e1aeea049be
 ---> 532aedd4e4e6
Successfully built 532aedd4e4e6
Successfully tagged cowimage:latest
```

Then we can use this image to run commands, for example cowsay:

```
In [3]: !docker run cowimage cowsay "Hello"

 -------
< Hello >
 -------
        \   ^__^
```

```
\  (oo)\_______
   (__)\       )\/\
       ||----w |
       ||     ||
```

3.1 Exporting an image to a file

Often, having a Dockefile as shown above is sufficient to create a computational environment, in particular if all support libraries and the main code itself are open source, and ideally available online.

In cases where we need to transport a created container image, we can use the following commands:

```
In [4]: !docker save cowimage > cowimage.tar
```

```
In [5]: !ls -hl cowimage.tar
```

```
-rw-r--r--  1 fangohr  staff   163M Sep  8 21:36 cowimage.tar
```

It's worth trying to compress the image:

```
In [6]: !gzip -f cowimage.tar
```

```
In [7]: !ls -hl cowimage.tar.gz
```

```
-rw-r--r--  1 fangohr  staff    69M Sep  8 21:36 cowimage.tar.gz
```

3.2 Importing an image from a file

```
In [8]: !docker load < cowimage.tar.gz
```

```
Loaded image: cowimage:latest
```

Let's tidy up and remove the large file we just created:

```
In [9]: !rm -f cowimage.tar.gz
```

4 Working on data on host system (mount directory)

4.1 Non-persistance of file system changes in container

While the container is a nice environment to provide our tailored installation environment and computational tools, it does not retain any changes to the disk system: as soon as the container process is stopped, all changes to the files within the container are forgotten:

```
In [1]: !docker run ubuntu:18.04 echo "Hello" > hello.txt && ls -l hello.txt
```

```
-rw-r--r--  1 fangohr  staff  6 Sep  8 21:40 hello.txt
```

In the example above, we create the file `hello.txt` and store `Hello` in it. The `&&` means that if that command was executed successfully, we will carry on and also execute the next one. The next command (`ls -l hello.txt`) tries to display the file `hello.txt`, and the command succeeds.

At this point, our container sessions stops, and all changes in the container are forgotten. We can confirm this by trying to use `ls -l` exactly as we did before, but find that the file `hello.txt` is not there:

```
In [2]: !docker run ubuntu:18.04 ls -l hello.txt

ls: cannot access 'hello.txt': No such file or directory
```

While it is possible to create special data containers for persistent data, I find it more straight forward to mount a directory from the host system into the container, and to save any output data on this mounted directory.

4.2 Mounting a directory from the host to be available in the container

Let's create a new container to demonstrate this. We will use `cowsay` as the application we install in the container, and we want it to "say" something that comes from an input file (on the host system) and to produce output in the container, which should be saved to the host file system so we can make use of this when the container execution has completed.

```
In [3]: %%file Dockerfile
        FROM ubuntu:18.04

        RUN apt-get update
        RUN apt-get install -y cowsay

        # cowsay installs into /usr/games. Make avaible in PATH:
        RUN ln -s /usr/games/cowsay /usr/local/bin

        # create directory we use for input and output
        RUN mkdir /io

        # change into that direcotry
        WORKDIR /io

Overwriting Dockerfile


In [4]: !docker build -t cowimage-mount .

Sending build context to Docker daemon  163.3kB
Step 1/6 : FROM ubuntu:18.04
 ---> cd6d8154f1e1
Step 2/6 : RUN apt-get update
 ---> Using cache
 ---> 864c48282361
Step 3/6 : RUN apt-get install -y cowsay
 ---> Using cache
```

```
 ---> 951b8495700a
Step 4/6 : RUN ln -s /usr/games/cowsay /usr/local/bin
 ---> Using cache
 ---> 532aedd4e4e6
Step 5/6 : RUN mkdir /io
 ---> Using cache
 ---> a9d42e44c301
Step 6/6 : WORKDIR /io
 ---> Using cache
 ---> 426382c2da8b
Successfully built 426382c2da8b
Successfully tagged cowimage-mount:latest
```

Let's check that we start in `/io` if we use the container:

```
In [5]: !docker run cowimage-mount pwd

/io
```

Now we need to mount our local directory to `/io` when we call docker. Let's first create an itput data file:

```
In [6]: %%file cow-input.txt
        Hello from file

Overwriting cow-input.txt
```

Let's also make sure no file `cow-output.txt` is on the disk. (We create the file in the next command.)

```
In [7]: !rm -f cow-output.txt

In [8]: !docker run -v `pwd`:/io cowimage-mount  cowsay `cat cow-input.txt` > cow-output.txt
```

Let's first check that this has created our output file `cow-output.txt`, and that the file is available on the host system:

```
In [9]: !ls -l cow-output.txt

-rw-r--r--  1 fangohr  staff  181 Sep  8 21:40 cow-output.txt
```

The file exists. What does it contain?

```
In [10]: !cat cow-output.txt

 -----------------
< Hello from file >
 -----------------
        \   ^__^
```

```
\  (oo)\_______
   (__)\       )\/\
       ||----w |
       ||     ||
```

This is good. We discuss all parts of the command line now.

The `-v` option tells docker to mount a Volume. In particular, the notation `-v A:B` asks to mount the path `A` from the host file system to the path `B` in the container. As we would like to mount our current directory from the host, we use the `pwd` command (which stands for Print Working Directory). By enclosing `pwd` in backticks (`), the output of the `pwd` command is used to represent `A`.

The path `B` to which we mount the directory is `/io`. Note that we have asked in the Dockerfile that the process in the container should start in this directory.

The actual command to be executed within the container is

```
cowsay `cat cow-input.txt` > cow-output.txt
```

- `cowsay` is the name of the executable.
- `cow-input.txt` is a file on the host file system, which is available within the container in `/io` because we mounted the directory
- with `` `cat cow-input.txt` `` we, we take the content of the `cow-input.txt` file (which is `hello from file` as we have created the file earlier in this notebook), and pass this content to `cowsay`. As a result, the cow prints this in the speech bubble.
- with `> cow-output.txt` we send the (standard) output from the process into the file `cow-output.txt`. As the file doesn't exist, it is created (in the container) in our directory `/io`) and as our host directory is mounted to `/io` in the container, the file is actually saved in the host directory. And therefore available after the docker container process has completed.

5 Connecting to a port in the container

Another common use case is that we start some kind of long-running process in the container, and talk to it through a port. That process could be a Jupyter Notebook, for example.

5.1 Example http.server

For demonstration purposes, we will use Python's in-built web server. To run it from the host, we could use

```
In [1]: # NBVAL_SKIP
        !python -m http.server
```

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [08/Sep/2018 21:41:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Sep/2018 21:41:27] code 404, message File not found
127.0.0.1 - - [08/Sep/2018 21:41:27] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [08/Sep/2018 21:41:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Sep/2018 21:41:31] code 404, message File not found
127.0.0.1 - - [08/Sep/2018 21:41:31] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [08/Sep/2018 21:41:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [08/Sep/2018 21:41:34] code 404, message File not found
127.0.0.1 - - [08/Sep/2018 21:41:34] "GET /favicon.ico HTTP/1.1" 404 -
```

```
^C

Keyboard interrupt received, exiting.
```

This program will show the contents of the current file system in a webbrowser interface at port 8000 of this machine. So typically at one or some of these links http://127.0.0.1:8000 or http://localhost:8000 or http://0.0.0.0:8000).

(If you have executed the above cell by pressing SHIFT+RETURN, you need to interrupt the http.server process to get the control back in the notebook. This can be done by choosing from the menu "Kernel" -> "Interrupt".)

We will now create a container and run this server inside the container. We like to use a web-browser on the host machine to inspect the files.

First, we create the Dockerfile:

```
In [2]: %%file Dockerfile
        FROM ubuntu:18.04

        RUN apt-get -y update
        RUN apt-get -y install python3

        CMD python3 -m http.server

Overwriting Dockerfile
```

The last line starts the `http.server` when the container is run.

```
In [3]: #NBVAL_IGNORE_OUTPUT
        !docker build -t portdemo .

Sending build context to Docker daemon  163.3kB
Step 1/4 : FROM ubuntu:18.04
 ---> cd6d8154f1e1
Step 2/4 : RUN apt-get -y update
 ---> Running in bd009d15a49b
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/universe Sources [11.5 MB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/universe Sources [17.4 kB]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [1363 B]
Get:8 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [69.0 kB]
Get:9 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [203 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/universe Sources [70.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [226 kB]
```

```
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [3925 B]
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [401 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [2807 B]
Fetched 25.9 MB in 3s (7665 kB/s)
Reading package lists...
Removing intermediate container bd009d15a49b
 ---> 171f7e7d44c9
Step 3/4 : RUN apt-get -y install python3
 ---> Running in 63358399b4f5
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  file libexpat1 libmagic-mgc libmagic1 libmpdec2 libpython3-stdlib
  libpython3.6-minimal libpython3.6-stdlib libreadline7 libsqlite3-0 libssl1.1
  mime-support python3-minimal python3.6 python3.6-minimal readline-common
  xz-utils
Suggested packages:
  python3-doc python3-tk python3-venv python3.6-venv python3.6-doc binutils
  binfmt-support readline-doc
The following NEW packages will be installed:
  file libexpat1 libmagic-mgc libmagic1 libmpdec2 libpython3-stdlib
  libpython3.6-minimal libpython3.6-stdlib libreadline7 libsqlite3-0 libssl1.1
  mime-support python3 python3-minimal python3.6 python3.6-minimal
  readline-common xz-utils
0 upgraded, 18 newly installed, 0 to remove and 1 not upgraded.
Need to get 6184 kB of archives.
After this operation, 33.4 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libssl1.1 amd64 1.1.0g-2ubun
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 libpython3.6-minimal amd64 3.6.5-3 [
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 libexpat1 amd64 2.2.5-3 [80.2 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main amd64 python3.6-minimal amd64 3.6.5-3 [142
Get:5 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 python3-minimal amd64 3.6.5-
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 mime-support all 3.60ubuntu1 [30.1 k
Get:7 http://archive.ubuntu.com/ubuntu bionic/main amd64 libmpdec2 amd64 2.4.2-1ubuntu1 [84.1
Get:8 http://archive.ubuntu.com/ubuntu bionic/main amd64 readline-common all 7.0-3 [52.9 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic/main amd64 libreadline7 amd64 7.0-3 [124 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/main amd64 libsqlite3-0 amd64 3.22.0-1 [496 kB
Get:11 http://archive.ubuntu.com/ubuntu bionic/main amd64 libpython3.6-stdlib amd64 3.6.5-3 [
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 python3.6 amd64 3.6.5-3 [186 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libpython3-stdlib amd64 3.6
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 python3 amd64 3.6.5-3ubuntu
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagic-mgc amd64 1:5.32-2
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagic1 amd64 1:5.32-2ubu
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 file amd64 1:5.32-2ubuntu0.
Get:18 http://archive.ubuntu.com/ubuntu bionic/main amd64 xz-utils amd64 5.2.2-1.3 [83.8 kB]
debconf: delaying package configuration, since apt-utils is not installedFetched 6184 kB in 1
Selecting previously unselected package libssl1.1:amd64.
(Reading database ... 4037 files and directories currently installed.)
Preparing to unpack .../libssl1.1_1.1.0g-2ubuntu4.1_amd64.deb ...
```

```
Unpacking libssl1.1:amd64 (1.1.0g-2ubuntu4.1) ...
Selecting previously unselected package libpython3.6-minimal:amd64.
Preparing to unpack .../libpython3.6-minimal_3.6.5-3_amd64.deb ...
Unpacking libpython3.6-minimal:amd64 (3.6.5-3) ...
Selecting previously unselected package libexpat1:amd64.
Preparing to unpack .../libexpat1_2.2.5-3_amd64.deb ...
Unpacking libexpat1:amd64 (2.2.5-3) ...
Selecting previously unselected package python3.6-minimal.
Preparing to unpack .../python3.6-minimal_3.6.5-3_amd64.deb ...
Unpacking python3.6-minimal (3.6.5-3) ...
Setting up libssl1.1:amd64 (1.1.0g-2ubuntu4.1) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine m
debconf: falling back to frontend: Teletype
Setting up libpython3.6-minimal:amd64 (3.6.5-3) ...
Setting up libexpat1:amd64 (2.2.5-3) ...
Setting up python3.6-minimal (3.6.5-3) ...
Selecting previously unselected package python3-minimal.
(Reading database ... 4292 files and directories currently installed.)
Preparing to unpack .../0-python3-minimal_3.6.5-3ubuntu1_amd64.deb ...
Unpacking python3-minimal (3.6.5-3ubuntu1) ...
Selecting previously unselected package mime-support.
Preparing to unpack .../1-mime-support_3.60ubuntu1_all.deb ...
Unpacking mime-support (3.60ubuntu1) ...
Selecting previously unselected package libmpdec2:amd64.
Preparing to unpack .../2-libmpdec2_2.4.2-1ubuntu1_amd64.deb ...
Unpacking libmpdec2:amd64 (2.4.2-1ubuntu1) ...
Selecting previously unselected package readline-common.
Preparing to unpack .../3-readline-common_7.0-3_all.deb ...
Unpacking readline-common (7.0-3) ...
Selecting previously unselected package libreadline7:amd64.
Preparing to unpack .../4-libreadline7_7.0-3_amd64.deb ...
Unpacking libreadline7:amd64 (7.0-3) ...
Selecting previously unselected package libsqlite3-0:amd64.
Preparing to unpack .../5-libsqlite3-0_3.22.0-1_amd64.deb ...
Unpacking libsqlite3-0:amd64 (3.22.0-1) ...
Selecting previously unselected package libpython3.6-stdlib:amd64.
Preparing to unpack .../6-libpython3.6-stdlib_3.6.5-3_amd64.deb ...
Unpacking libpython3.6-stdlib:amd64 (3.6.5-3) ...
Selecting previously unselected package python3.6.
Preparing to unpack .../7-python3.6_3.6.5-3_amd64.deb ...
Unpacking python3.6 (3.6.5-3) ...
Selecting previously unselected package libpython3-stdlib:amd64.
Preparing to unpack .../8-libpython3-stdlib_3.6.5-3ubuntu1_amd64.deb ...
Unpacking libpython3-stdlib:amd64 (3.6.5-3ubuntu1) ...
Setting up python3-minimal (3.6.5-3ubuntu1) ...
Selecting previously unselected package python3.
```

```
(Reading database ... 4748 files and directories currently installed.)
Preparing to unpack .../python3_3.6.5-3ubuntu1_amd64.deb ...
Unpacking python3 (3.6.5-3ubuntu1) ...
Selecting previously unselected package libmagic-mgc.
Preparing to unpack .../libmagic-mgc_1%3a5.32-2ubuntu0.1_amd64.deb ...
Unpacking libmagic-mgc (1:5.32-2ubuntu0.1) ...
Selecting previously unselected package libmagic1:amd64.
Preparing to unpack .../libmagic1_1%3a5.32-2ubuntu0.1_amd64.deb ...
Unpacking libmagic1:amd64 (1:5.32-2ubuntu0.1) ...
Selecting previously unselected package file.
Preparing to unpack .../file_1%3a5.32-2ubuntu0.1_amd64.deb ...
Unpacking file (1:5.32-2ubuntu0.1) ...
Selecting previously unselected package xz-utils.
Preparing to unpack .../xz-utils_5.2.2-1.3_amd64.deb ...
Unpacking xz-utils (5.2.2-1.3) ...
Setting up readline-common (7.0-3) ...
Setting up mime-support (3.60ubuntu1) ...
Setting up libreadline7:amd64 (7.0-3) ...
Setting up libmagic-mgc (1:5.32-2ubuntu0.1) ...
Setting up libmagic1:amd64 (1:5.32-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Setting up xz-utils (5.2.2-1.3) ...
update-alternatives: using /usr/bin/xz to provide /usr/bin/lzma (lzma) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/lzma.1.gz because associat
update-alternatives: warning: skip creation of /usr/share/man/man1/unlzma.1.gz because associ
update-alternatives: warning: skip creation of /usr/share/man/man1/lzcat.1.gz because associa
update-alternatives: warning: skip creation of /usr/share/man/man1/lzmore.1.gz because associ
update-alternatives: warning: skip creation of /usr/share/man/man1/lzless.1.gz because associ
update-alternatives: warning: skip creation of /usr/share/man/man1/lzdiff.1.gz because associ
update-alternatives: warning: skip creation of /usr/share/man/man1/lzcmp.1.gz because associa
update-alternatives: warning: skip creation of /usr/share/man/man1/lzgrep.1.gz because associ
update-alternatives: warning: skip creation of /usr/share/man/man1/lzegrep.1.gz because assoc
update-alternatives: warning: skip creation of /usr/share/man/man1/lzfgrep.1.gz because assoc
Setting up libsqlite3-0:amd64 (3.22.0-1) ...
Setting up libmpdec2:amd64 (2.4.2-1ubuntu1) ...
Setting up libpython3.6-stdlib:amd64 (3.6.5-3) ...
Setting up python3.6 (3.6.5-3) ...
Setting up file (1:5.32-2ubuntu0.1) ...
Setting up libpython3-stdlib:amd64 (3.6.5-3ubuntu1) ...
Setting up python3 (3.6.5-3ubuntu1) ...
running python rtupdate hooks for python3.6...
running python post-rtupdate hooks for python3.6...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Removing intermediate container 63358399b4f5
 ---> 1e9c6b97586e
Step 4/4 : CMD python3 -m http.server
 ---> Running in cbe4f046ed6e
Removing intermediate container cbe4f046ed6e
 ---> e5d48bb20260
Successfully built e5d48bb20260
```

```
Successfully tagged portdemo:latest
```

We now need to export the port 8000 in the container. We can do this using:

```
In [4]: #NBVAL_SKIP
        !docker run -p 8123:8000 portdemo
```

```
^C
```

The numbers `8123:8000` mean that the internal port (8000) of the container should be connected to the port (8123) on the host system.

Once the above command is executing, we should be able to browse the file system in the container by going to the link http://localhost:8123 (or http://127.0.0.1:8123 or http://0.0.0.0:8123) on the host system.

We could have mapped port 8000 in the container to port 8000 on the host as well (`-p 8000:8000`).

As before, to stop the process, select `Kernel->Interrupt`.

5.2 Jupyter Notebook

A common application of exposing ports is to install computational or data analysis software inside the container, and to control it from a Jupyter notebook running inside the container, but to use a webbrowser from the host system to interact with the notebook. In that case, the above example of exposing the port is in principle the right way to go, too. However, as this is a common use-case, there are a number of prepared Dockerfiles to install the notebook inside the container available at https://github.com/jupyter/docker-stacks, so that one can start the Dockerfile with `FROM jupyter/...`, (instead of `FROM ubuntu/...`) and in this way build on the Dockerfiles that the Jupyter team provides already.

The container image for https://github.com/jupyter/docker-stacks/tree/master/scipy-notebook might be a good starting point for work based on the Scientific Python stack.

6 Re-connect to container, or throw away

We can use the computational environment in a container in many ways, including:

- Option 1: to execute one command that produces some output, and writes this to a mounted folder (so the data is available on the host file system). We don't need the computational environment any more (all our output is on the host file system). If we need to run the command again, we can start with a fresh container again.
- Option 2: to start a container as a process, to which we can connect repeatedly to carry out tasks inside the same container instance. This is known as running the container in a "detached" state.

In this notebook, we will explore both options.

6.1 Opion 1: Run command in container, then throw container away

This is what we have done in previous examples.

However, there is one detail that I'd like to introduce now: we executed our command, for example using `docker run cowimage cowsay "Hello"`. Every time we run this command, docker saves

the state of the container after the execution of the command, and if we really wanted, we could connect to it and carry on working with this container.

Here is some evidence. First, let's show that a new container is created every time we run the command:

```
In [1]: !docker run cowimage cowsay "Hello 1"
```

```
 ---------
< Hello 1 >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
In [2]: !docker run cowimage cowsay "Hello 2"
```

```
 ---------
< Hello 2 >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Now we ask docker to display the directory (`ls`) of the `last 2` containers that have been created:

```
In [3]: !docker container ls --last 2
```

```
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS
d3d165e8858c        cowimage            "cowsay 'Hello 2'"    13 seconds ago      Exited (0) 1
416e170dc3ad        cowimage            "cowsay 'Hello 1'"    15 seconds ago      Exited (0) 1
```

We can see that there are two images, and by looking at the command and the time ago, we can see that we have just created those two containers.

We can delete those using the `docker container rm` command, but need to provide the `CONTAINER ID` as an argument. The `CONTAINER ID` is given in the first column of the output above. There is a switch `-q` to only return that:

```
In [4]: !docker container ls --last 2 -q
```

```
d3d165e8858c
416e170dc3ad
```

With this information, we can delete those containers, for example:

```
In [5]: !docker container rm $(docker container ls --last 2 -q)
```

```
d3d165e8858c
416e170dc3ad
```

Let's check they have disappeared:

```
In [6]: !docker container ls --last 2
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

6.2 The `run --rm` option

For one-time execution of a command, we can use the `--rm` option to run, which means that the container will be destroyed as soon as our use of it has stopped:

```
In [7]: !docker container ls --last 1
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

```
In [8]: !docker run --rm cowimage cowsay "Hello 3"
```

```
 ---------
< Hello 3 >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
In [9]: !docker container ls --last 1
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

6.3 Option 2: attach to running container

6.3.1 Start the container in detached mode using `run -d`

To demonstrate this, we start a `bash` process in our container. The `-d` switch means to Detach the container, meaning that it will run it the background. The `-t` switch is necessary to attach a pseudo terminal to the `bash` command. The return value is the ID of the container.

```
In [10]: !docker run -d -t cowimage bash
```

```
6ba4f1e46f452b8be22ba86531b2307f3799c718ee8865149d320660cbf70924
```

For this demonstration, it is convenient to know the name of the container in a variable. We thus use Jupyter Notebook python magic and save the output from the `!docker ...` command in the variable `output`:

```
In [11]: output = !docker run -d -t cowimage bash
```

Turns out that `output` is a list:

```
In [12]: output

Out[12]: ['227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927']
```

So we take the first element to be able to refer to the container ID:

```
In [13]: ID = output[0]

In [14]: ID

Out[14]: '227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927'
```

Let's check that the container is running:

```
In [15]: !docker container ls --last 1

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
227518a56581        cowimage            "bash"              7 seconds ago       Up 6 seconds
```

As we have stored the name of the container in the variable `ID`, we can pass the value of the variable to the `!docker` command using `$ID`. Here is an example, asking the container for the current date and time (in its timezone):

```
In [16]: !docker exec $ID date

Sun Sep  9 06:55:49 UTC 2018
```

6.3.2 Execute arbitrary commands in the running container using `exec`

Here is the plan: using subsequent commands, we show the files in /tmp in the container, create a file in the container in /tmp, then confirm it exists, then stop the container, then start the container again, and confirm the file is still there.

Confirm there are no files in the `/tmp/` directory in the container:

```
In [17]: !docker exec $ID ls -l /tmp

total 0
```

Now we create a file:

```
In [18]: !docker exec $ID touch /tmp/my-file.txt
```

We confirm the file exists:

```
In [19]: !docker exec $ID ls -l /tmp
```

```
total 0
-rw-r--r-- 1 root root 0 Sep  9 06:56 my-file.txt
```

As we have seen earlier, carrying out the same sequence of commands using `run` instead of `exec` would not work: as the second `run` command will start from the original docker *image* again and create a new container for the execution.

We can also stop our running container using `stop`:

```
In [20]: !docker stop $ID
```

```
227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927
```

We cannot execute further commands in this container because it is stopped:

```
In [21]: !docker exec $ID ls -l /tmp
```

```
Error response from daemon: Container 227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513b
```

However, we can start it again using `start`:

```
In [22]: !docker start $ID
```

```
227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927
```

```
In [23]: !docker exec $ID ls -l /tmp
```

```
total 0
-rw-r--r-- 1 root root 0 Sep  9 06:56 my-file.txt
```

6.4 The `container ls` command (`ps`)

This is a good opportunity for some more detail on the `docker container ls` command:

Reminder: the `ls` command lists all *containers*. Containers are instances of images that are created when an image is being executed.

The `ls` command shows all running containers:

```
In [24]: !docker container ls
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
227518a56581        cowimage            "bash"              37 seconds ago      Up 4 seconds
6ba4f1e46f45        cowimage            "bash"              48 seconds ago      Up 48 seconds
```

If we stop our container, it will not be displayed anymore using `ls`:

```
In [25]: !docker stop $ID
```

```
227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927
```

```
In [26]: !docker container ls
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
6ba4f1e46f45        cowimage            "bash"              56 seconds ago      Up 55 seconds
```

6.4.1 `ls -a`

We can see all containers (even those not executing) using the `ls -a` option. Note that a container that is run running has the status `Exited`, whereas the active processes show `Up [some time]`:

```
In [27]: !docker container ls -a
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS
227518a56581        cowimage            "bash"              49 seconds ago       Exited (0) 4
6ba4f1e46f45        cowimage            "bash"              About a minute ago   Up 59 second
```

6.4.2 `--last N`

The `--last N` option shows the last N containers that have been created (irrespective of if they are running or not). For example the last 3 containers:

```
In [28]: !docker container ls --last 3
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS
227518a56581        cowimage            "bash"              51 seconds ago       Exited (0) 6
6ba4f1e46f45        cowimage            "bash"              About a minute ago   Up About a m
```

6.4.3 `container ls` == `ps`

As the `container ls` command behaves a little bit like the Linux `ps` command, we can use `docker ps` instead of `docker container ls`:

```
In [29]: !docker container ls --last 1
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS
227518a56581        cowimage            "bash"              58 seconds ago       Exited (0) 14
```

```
In [30]: !docker ps --last 1
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS
227518a56581        cowimage            "bash"              About a minute ago   Exited (0) 1
```

6.5 Remove all exited containers

To remove an exited container from our host, we can use `docker container rm` command:

```
In [31]: !docker container rm $ID
```

```
227518a5658159522c7f377cd45cd92782aff361d0c07a580cb513bcc660e927
```

What is the container that has been created most recently now?

```
In [32]: !docker container ls -a --last 1
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
6ba4f1e46f45        cowimage            "bash"              About a minute ago  Up About a m
```

If we don't need any state that maybe saved in our exited containers, we can remove them from our host system. We can ask for the container ids (`-q`) for all containers:

```
In [33]: !docker container ls -a -q

6ba4f1e46f45
```

and we can use that information to pass all the container ids to the `rm` command:

```
In [34]: !docker container rm $(docker container ls -a -q)

Error response from daemon: You cannot remove a running container 6ba4f1e46f452b8be22ba86531b


In [35]: !docker container ls -a

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
6ba4f1e46f45        cowimage            "bash"              About a minute ago  Up About a m
```

We can stop individual containers using the `stop` command (see above). To stop all running containers conveniently, we can use the `kill` command:

```
In [36]: !docker container kill $(docker container ls -q)

6ba4f1e46f45
```

6.6 More convenience: naming containers

In the example above, we have used the ID of the container. We can also give containers a name ourselves (ideally one that is easier to remember) and use that to refer to the container:

```
In [37]: !docker run -d -t --name myname cowimage bash

04c4e348b76fc1a6cfbbe30d902241faf4eca032360f70552dc0002dadee8cc4


In [38]: !docker container ls

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
04c4e348b76f        cowimage            "bash"              2 seconds ago       Up 1 second


In [39]: !docker exec myname date

Sun Sep  9 06:57:23 UTC 2018
```

```
In [40]: !docker container stop myname

myname


In [41]: !docker container rm myname

myname
```

It is also worth noting that we don't need to use the full hash of the container ID when referring to them; we can just use the leading few characters as long as they are unique:

```
In [42]: output = !docker run -dt --name myname cowimage bash

In [43]: IDfull = output[0]
         print(IDfull)

e14eba99c7b0121fe5168e59362cbd5a6ed5786795f83f0be57a3eafb435fe8d


In [44]: IDshort = IDfull[0:3]
         print(IDshort)

e14


In [45]: !docker container exec $IDshort date

Sun Sep  9 06:57:37 UTC 2018


In [46]: !docker container exec $IDfull date

Sun Sep  9 06:57:37 UTC 2018
```

And some tidying up:

```
In [47]: !docker container stop $IDshort

e14


In [48]: !docker container rm $IDshort

e14
```

6.7 More convenience: dropping `container` from commands

In many places where we have used `container` above, this can be dropped from the command:

```
In [49]: !docker run -dt --name myname2 cowimage bash
```

```
88dac8aaa738d89d3e6706328076db43bdaac9d48252c0f8f54b6b7a87da2c93
```

```
In [50]: !docker exec myname2 date
```

```
Sun Sep  9 06:57:44 UTC 2018
```

```
In [51]: !docker stop myname2
```

```
myname2
```

```
In [52]: !docker rm myname2
```

```
myname2
```

Using `docker container exec` and `docker container stop` is sligthly clearer that `docker exec` and `docker stop` and done here for education purposes, but typically people will omit the `container` from the commands.

7 Connect X to container

https://cntnr.io/running-guis-with-docker-on-mac-os-x-a14df6a76efc

```
In [ ]:
```

8 Clean up

```
In [ ]:
```

9 References

A number of useful references I came across while putting together these notebooks.

- Carl Boettiger

```
@article{DBLP:journals/corr/Boettiger14,
  author    = {Carl Boettiger},
  title     = {An introduction to Docker for reproducible research, with examples
               from the {R} environment},
  journal   = {CoRR},
  volume    = {abs/1410.0846},
  year      = {2014},
```

```
  url       = {http://arxiv.org/abs/1410.0846},
  timestamp = {Sun, 02 Nov 2014 11:25:59 +0100},
  biburl    = {http://dblp.uni-trier.de/rec/bib/journals/corr/Boettiger14},
  bibsource = {dblp computer science bibliography, http://dblp.org}
}
```

- Titus Brown

http://angus.readthedocs.io/en/2016/week3/CTB_docker.html
http://ivory.idyll.org/blog/2015-docker-and-replicating-papers.html

- David Koop

http://www.cis.umassd.edu/~dkoop/cis602-2016fa/lectures/lecture15.pdf

In [ ]: