Run whatever on Hercules with Docker and Singularity

Henning Hilmarsson

MPIfR Bonn

henning@mpifr-bonn.mpg.de

July 7, 2017

Overview

The aim of this tutorial is to get you to be able to run your desired versions of your programs on e.g. the Hercules cluster, where permissions will be limited. This requires creating a Docker image, converting it to a Singularity image, and then using that Singularity image on Hercules.
This will in no way be a technical tutorial, where everything is explained from a-z. The purpose is to get you to run your stuff with singularity by basically telling you what to do step by step, with minimal fluff.
The Dockerfile I will be using in this tutorial is a Dockerfile for a modified version of PRESTO, and can be found at
`https://github.com/ghenning/DockerfilePrestoMod.git`

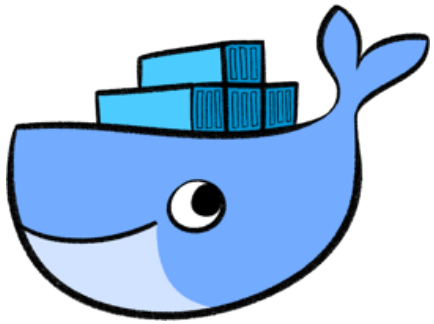
The annoying stuff...

- Can't install this or that
- The version of X that I want is not installed
- Other infuriating things that might arise (permission issues etc.)

The solution!

Docker

1. Create (or borrow) DockerFile
2. Manipulate DockerFile to install whatever you want
3. Or install things from within container and commit to image



Singularity

1. Issues with Docker on Hercules
2. Convert Docker image to Singularity image with one line
3. Use Singularity image on Hercules

How to...

- Create (borrow) Dockerfile?
- Create Docker image?
- Commit changes to Docker image?
- Use Docker image?
- Convert Docker image to Singularity image?
- Use Singularity image?

Installing Docker and Singularity on your machine should be simple, there is a "how to" on their webpages, but you will need sudo privileges (at least for singularity), so you're advised to do this on your own machine.

How does the Dockerfile look like?

The Dockerfile is just a text document containing the commands required to build the image. You can get a complete Dockerfile from somewhere, and then modify it to your needs and/or make changes to the image after you've built it, which we wil go through in this tutorial.

```
# Copyright (C) 2016 by Ewan Barr
# Licensed under the Academic Free License version 3.0
# This program comes with ABSOLUTELY NO WARRANTY.
# You are free to modify and redistribute this code as long
# as you do not remove the above attribution and reasonably
# inform receipients that you have modified the original work.

FROM ubuntu:xenial-20160923.1

MAINTAINER Ewan Barr "ebarr@mpifr-bonn.mpg.de"

# Suppress debconf warnings
ENV DEBIAN_FRONTEND noninteractive

# Switch account to root and adding user accounts and password
USER root
RUN echo "root:root" | chpasswd && \
    mkdir -p /root/.ssh

# Create psr user which will be used to run commands with reduced privileges.
RUN adduser --disabled-password --gecos 'unprivileged user' psr && \
    echo "psr:psr" | chpasswd && \
    mkdir -p /home/psr/.ssh && \
    chown -R psr:psr /home/psr/.ssh

# Create space for ssh daemon and update the system
RUN echo 'deb http://us.archive.ubuntu.com/ubuntu trusty main multiverse' >> /etc/apt/sources.list &&
\
    apt-get -y update && \
    apt-get install -y apt-utils apt-transport-https software-properties-common python-software-proper
ties && \

# Install dependencies
```

Dockerfile and image

Get your Dockerfile, e.g. from Ewan:

`https://hub.docker.com/u/mpifrpsr/`

then cd into the dir where your Dockerfile is, and run

```
$ docker build -t "name of image" .
```

and make sure your directory has only one Dockerfile. I did this with a modified PRESTO build, so I ran

```
$ docker build -t prestomod .
```

When it's done compiling the image, the last line in the terminal output should look something like this:

Docker Images

Let's see what images we have available, using

```
$ docker images
```

which lists all the images you have available:

```
user@n1305:~/work/docker/DockerfilePrestoMod$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
prestomod           latest              b01201fc07e3        3 minutes ago
1.45 GB
hello-world         latest              48b5124b2768        5 months ago
1.84 kB
ubuntu              xenial-20160923.1   c73a085dc378        9 months ago
127 MB
```

Now let's mess around with our new image a bit.

Docker Images continued

Let's run a Docker container on our new image, using "run"

```
$ docker run --rm -ti prestomod bash
```

This opens a container on the image "prestomod", the "-ti" gives us an interactive container, for exact jargon, you can run

```
$ docker run --help
```

and the "–rm" command removes the container after we've closed it (more on that later). Now we're inside the container, and we can do whatever we please inside of it.

```
user@n1305:~$ docker run --rm -ti prestomod bash
root@0a823e725d7d:~# echo "I'm working in a docker container now!"
I'm working in a docker container now!
root@0a823e725d7d:~#
```

Making changes

Let's open python within the container. I want to import filterbank, rawdata, and waterfaller from PRESTO, so let's try it out.

```
root@0a823e725d7d:~# python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import filterbank
>>> import rawdata
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/psr/software/presto1bitmod/lib/python/rawdata.py", line 5, in <module>
    from psrfits import PsrfitsFile
  File "/home/psr/software/presto1bitmod/lib/python/psrfits.py", line 18, in <module>
    import astropy.io.fits as pyfits
ImportError: No module named astropy.io.fits
>>> import waterfaller
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/psr/software/presto1bitmod/lib/python/waterfaller.py", line 18, in <module>
    import matplotlib.pyplot as plt
  File "/usr/local/lib/python2.7/dist-packages/matplotlib/pyplot.py", line 115, in <module>
    _backend_mod, new_figure_manager, draw_if_interactive, _show = pylab_setup()
  File "/usr/local/lib/python2.7/dist-packages/matplotlib/backends/__init__.py", line 32, in pylab_setup
    globals(),locals(),[backend_name],0)
  File "/usr/local/lib/python2.7/dist-packages/matplotlib/backends/backend_tkagg.py", line 6, in <module>
    from six.moves import tkinter as Tk
  File "/usr/local/lib/python2.7/dist-packages/six.py", line 203, in load_module
    mod = mod._resolve()
  File "/usr/local/lib/python2.7/dist-packages/six.py", line 115, in _resolve
    return _import_module(self.mod)
  File "/usr/local/lib/python2.7/dist-packages/six.py", line 82, in _import_module
    __import__(name)
  File "/usr/lib/python2.7/lib-tk/Tkinter.py", line 42, in <module>
    raise ImportError, str(msg) + ', please install the python-tk package'
ImportError: No module named _tkinter, please install the python-tk package
```

whoops, looks like astropy and python-tk are missing. Let's install those packages inside the container, and commit those changes to the image.

Making changes continued

Exit python in the container and run (you might need to run apt-get update if this doesn't work)

```
$ pip install --no-deps astropy
$ apt-get install python-tk
```

Did this work?

```
root@0a823e725d7d:~# python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import filterbank
>>> import waterfaller
>>> import rawdata
>>>
```

It sure did! Now we need to commit these changes to the image to make this change permanent.

Making changes continued

Detach from the container by pressing ctrl+p and crtl+q. To see the list of running containers, run

```
$ docker ps
```

and you will see something similar to this

```
root@0a823e725d7d:~# user@n1305:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
0a823e725d7d        prestomod           "bash"              32 minutes ago      Up 32 minutes       22/tcp              stoic_davinci
```

Docker gives containers random (and weird) names, it's possible to give your container a name when you do "docker run" by adding "–name NAME", where NAME is of course whatever name you want to give the container.
To commit the changes we made in the container to the image, we run

```
$ docker commit stoic_davinci prestomod
```

or in general

```
$ docker commit <container name> <image name>
```

Did the changes work?

Let's go back into the container with

```
$ docker attach stoic_davinci
```

You might need to hit enter a couple of times afterwards. Now just type exit inside the container to exit. The reason we wrote "–rm" when we ran "docker run" was so that when we exit the container like we just did, docker will remove the stopped container. Otherwise we would have to remove it manually. To see all containers (running and stopped), run

```
$ docker ps -a
```

Note that containers are identified with both "Container ID" and "Name", and you can use either one when running commands on a container (attach, commit, remove, etc.). To remove a container, run

```
$ docker rm <container name or ID>
```

Did the changes work? continued

To see if this worked the way we thought, lets open a new container on our image, and give it a name this time

```
$ docker run --rm -ti --name hamster prestomod bash
```

and let's see if we can import the packages we want in python

```
user@n1305:~$ docker run --rm -ti --name hamster prestomod bash
root@521b10200cef:~# python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import filterbank
>>> import waterfaller
>>> import rawdata
>>>
```

Super duper!
We've gone through the Docker part of this tutorial, but this has just scratched the surface of what Docker is capable of. You can run things without entering containers, mount volumes, and more. But that's outside of the scope of this tutorial, and there is plenty of documentation online for these purposes. Before entering the realm of Singularity, let's review some common Docker commands.

Docker commands

Help

```
$ docker --help
```

Help for certain commands, e.g. "run"

```
$ docker run --help
```

Build image (cd into Dockerfile dir)

```
$ docker build -t <image name> .
```

See images

```
$ docker images
```

See containers (add "-a" to see stopped containers)

```
$ docker ps -a
```

Docker commands continued

Run a container from image

```
$ docker run --rm -ti --name <name> <img name> bash
```

Detach from container without stopping it

```
ctrl+p ctrl+q
```

Get back into container (attach)

```
$ docker attach <container name or ID>
```

Commit changes to image (remember to detach from container first)

```
$ docker commit <container name> <image name>
```

Remove container or image

```
$ docker rm <container name or ID>
$ docker rmi <image name or ID>
```

Converting Docker to Singularity

Some nice folks had a hunch that others might want to convert their Docker images to Singularity images, so they did the work for us. All we have to do is to run the following

```
docker run \
-v /var/run/docker.sock:/var/run/docker.sock \
-v <local location of singularity image>:/output \
--privileged -t --rm \
singularityware/docker2singularity \
<your image name>
```

And continuing with the image we built it will look like this (I will have two singularity images, as I did this also a while ago)

```
user@n1305:~/work/singularity$ docker run -v /var/run/docker.sock:/var/run/docker.sock -v /home/
user/work/singularity:/output --privileged -t --rm singularityware/docker2singularity prestomod
Unable to find image 'singularityware/docker2singularity:latest' locally
latest: Pulling from singularityware/docker2singularity
```

```
(9/9) Moving the image to the output folder...
  2,447,376,416 100%   35.53MB/s    0:01:05 (xfr#1, to-chk=0/1)
user@n1305:~/work/singularity$ ls
prestomod-2017-05-11-29e5097df53c.img  prestomod-2017-07-05-9a5bb0f02b98.img
```

Using Singularity

Now we have our Singularity image, let's use it!
Note that if you installed software in "home" in your Docker image, you'll need to mount your own home directory to home1 when running singularity, because singularity auto-mounts home to home in the container, which overwrites what was there before (in your Docker image).
The workaround for this is to add

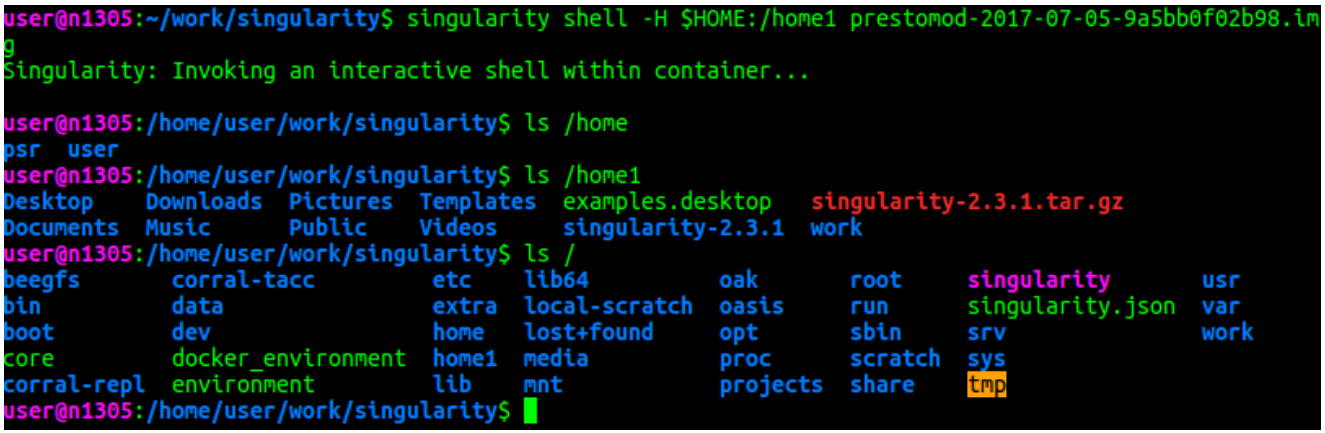
```
-H $HOME:/home1
```

in your singularity exec or shell command
Let's open a shell in our new singularity image and see what's the deal with home and home1

```
$ singularity shell -H $HOME:/home1 \
<path to singularity img>
```

Using Singularity continued

We can see in this figure that home is the home directory of the image, and home1 is our own local home directory. Also, there are other directories in the root directory which singularity creates for us, which can be really useful, like data and work. These directories can be mounted with your data and code, making it really simple to navigate and work in the singularity image. To exit singularity just write exit, or press ctrl+d (same as in Docker)



Using Singularity continued

Before we mount and run stuff with Singularity, we need to make sure we can import the PRESTO packages we want, like we did in Docker.

```
user@n1305:~/work/singularity$ singularity shell -H $HOME:/home1 prestomod-2017-07-05-9a5bb0f02b98.im
g
Singularity: Invoking an interactive shell within container...

user@n1305:/home/user/work/singularity$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import filterbank
>>> import waterfaller
>>> import rawdata
>>>
```

And indeed it works, just like in the Docker container.

Singularity exec

To run things with singularity without entering a container, we use the command "exec" like this

```
$ singularity exec -H $HOME:/home1 \
/path/to/singularity/image.img <command>
```

I made a python script (called reader.py) which takes in a textfile (called txtfile.txt) as an argument, and put the python file and the textfile in separate directories. Let's mount the python script directory in "work" and the textfile in "data", and run the script with singularity. The directory mounting command is "-B local path:singularity path".

```
$ singularity exec -H $HOME:/home1 \
-B /home/user/work/garbage/:/work \
-B /home/user/work/garbage/afile/:/data \
prestomod-2017-07-05-9a5bb0f02b98.img \
python /work/reader.py --file /data/txtfile.txt
```

Singularity exec continued

And this is what happens

```
user@n1305:~/work/singularity$ singularity exec -H $HOME:/home1 -B /home/user/work/garbage/:/work -B
/home/user/work/garbage/afile/:/data prestomod-2017-07-05-9a5bb0f02b98.img python /work/reader.py --f
ile /data/txtfile.txt
Ich
bin
ein
Auslander
und
spcreche
nicht
gut
Deutsch
user@n1305:~/work/singularity$
```

I highly recommend that you mount your data directory and the directory where your code is like this

```
-B /path/to/data/:/data
-B /path/to/code/:/work
```

because it makes directory handling so much easier within Singularity.

The grand finale!

A typical command with Singularity could look something like this

```
$ singularity exec -H $HOME:/home1 \
-B /path/to/data/:/data \
-B /path/to/code/:/work \
/path/to/singularity/image.img \
python /work/main.py --inputfiles /data/
```

All that's left to do is to copy your singularity image to where you want to use it (e.g. Hercules). Singularity is installed on hercules03, so you should be able to play around with your images there.
When you use Singularity on Hercules, you simply put the Singularity command where you would put your normal command in your batch script

```
srun singularity exec ...
```