Singularity Workshop

28. and 29.7.2021

In this hands-on workshop we address the following questions:
Day 1:
- what is a container?
- why using a container?
- how do I build a container?
- how do I build a singularity container?
- what do I need to bear in mind when I am creating and using a container?
- I got a docker image, can I use that one on Rosalind?
- how do I run a container on Rosalind?
- questions?

Day 2:
- why do I want to open up a container?
- how do I open up a container?
- how do I install additional software inside a container?
- how do I build the container again?
- questions?

What is a container?

Probably the most simplest definition of a container is a piece of software, which is a closed, self-contained entity. Thus it can be created on one computer and then lift-and-shift to a different computer to run there. Think of a shipping container which can be filled at one point and then moved to another point where it will be emptied again.

Why using a container?

There are several reasons why a container might be a good solution for you.
One is from the **installation** point of view. You can build the container once and then copy it to several computers and run it there. So you only need to do build the container once, and use it several times. This also means the work you are doing is **reproducible**. Because you are using **exactly** the same environment inside the container, it does not matter much where the container is running.
Like always in life, there are some things one needs to bear in mind with this.

How to build a container?

For the two most common types of containers, Docker and Singularity, there are only a few requirments which are needed in order to build a container.
For Docker, a Docker engine needs to run with root privileges! You then only need the Docker Definition File to build the container.
Note: there are also a number of already build containers on Dockerhub, you can download them as well. However, the question was about building a Docker container.
For Singularity, you only need Singularity being installed. This could either be a installation from the repository of your operating system, or you are building it manually. The latter approach has the advantage you got the latest version installed. Unlike Docker, there is no engine running with root privileges and thus there is a better security aspect. This is the main reason why you cannot use Docker on Rosalind.
It should be noted you only need root privileges when you are **installing** Singularity, but not when you are **building** or **running** a container. There are, again, some things to bear in mind here.

How to build a Singularity container?

Once Singularity is installed building a container requires two things:
- a Singularity Definition File
- sometimes a bit of time, depending on what you want to build.

The first problem is where to get the Singularity Definition File from and what is the content of it.
There are a few ways to get them:
- you get them from a colleague, download it from some repository
- make it yourself

As this workshop is about doing things yourself, we are actually creating our own Singularity Definition File. However, fear not, you don't need to know the ins-and-outs of how the file is constructed. In fact, you can get away without even ever looking into that file at all!
It is, however, good to know what the file is made of so if there are any problems, or you want to amend it, you know where to look out for.

Inside a Singularity Definition File

There are a few sections in the Singularity Definition File which are useful to know about.
First there is the head section, which is right at the top of the file. A typical example is here:

```
Bootstrap: debootstrap
OSVersion: bullseye
MirrorURL: http://httpredir.debian.org/debian
```

These lines tell Singularity to used a 'bootstrap' was to install the container, using the version 'Bullseye' from Debian and use the URL as specified. This will create a very minimalistic container with Debian Bullseye in it. However, a bit more is needed to actualy use it.

```
%post
apt update
...
```

This will instruct Singularity that in a post-installation process to do several things. Only one line is shown here. Here you basically can use normal Bash type commands.

Inside a Singularity Definition File

The next important bit is this section:

```
%runscript
eval "$@"
```

Here we instruct Singularity to run the command which you will put after the name of the container file. In other words, what to do when you are running the container. In this instance, we are simply saying: do what I am telling you to do.

```
%environment
...
```

Here we are setting the environment. This could be for example any specific paths, the used language inside the container, these things.

```
%labels
Author J. Sassmannshausen <jorg.sassmannshausen@kcl.ac.uk>
bzip2-1.0.6
```

This section is adding meta-data to the container. This is useful if you want to publish your Singularity Definition File. This way, both the conatiner and the file contain your name and email address for example. Other entries are possible and some metadata fields, like the used operating system, are automatically completed by Singularity.

How to build a Singularity container?

Fortunately, we don't need to know all of the working mechanism of the Definition File as long as we know how to make one. This is very easy. The two main requirements are:
- have Singularity installed on the machine you want to use in such a way you can build a container without the need to be root.
- know how to find your way around a GitHub page.

For the first point: In order to build a container, either 'root' or 'sudo' is required unless a helper program 'fakeroot' is installed. As neither of these requirements are fulfilled on Rosalind, you cannot **build** containers there.

The second point is important as we need to tell Singularity what to build. So here we need to know the **name** of the file from the EasyBuild Github page, we do normally **not** need to know the content.

How to build a Singularity container?

In order to find out the **name** of the EasyBuild file we need to use, we simply navigate in our web browser to this page:

https://github.com/easybuilders/easybuild-easyconfigs/tree/main/easybuild/easyconfigs

Make sure you are using the main branche and not another one!

So for example, we want to install 'bzip2' we would navigate into the 'b' folder and in there to 'bzip2'. Here you will find all the various versions of bzip2 you could install. You only need to be concerned with files which are ending with '.eb' as these are the EasyConfing (EC) files. You also might wonder what the difference is between for example 'bzip2-1.0.6.eb' and 'bzip2-1.0.6-GCCcore-8.2.0.eb' for example. Without going into too much details, anything which comes after the version number of the program you want to install, here it is '1.0.6' has to do with the used toolchain which consist out of compiler, additional programs and suchlike. For now, we can ignore that.

How to build a Singularity container?

So why the fuss about EasyBuild? The answer is that more complex programs, like for example R, a lot of dependencies are needed. EasyBuild is one way to take care of all of that, so you don't have to and usually that is working really well.

The final bit we need is a set of scripts to create Definition File and build the container. Fortunately, this can be done automatically using the scripts from this GitHub page:

https://github.com/sassy-crick/Singularity-Easybuild

The set of scripts allows you to create the Definition File, build a container and, if required, even build a sandbox.
For the Day 1 workshop, we are looking into how to install the scripts and how to do a simple example.

Exercise: build my first container

The GitHub page for the scripts is here: https://github.com/sassy-crick/Singularity-Easybuild
So we could to the familiar

```
$ git clone https://github.com/sassy-crick/Singularity-Easybuild.git
```

Which would clone the GitHub project. However, that is not desirable as any work done on the *main* branch on the GitHub page means that your installation will no longer be reproducible. Thus, it is better to install either a tag or a release from the page. Thus, we do it like this:

```
$ wget https://github.com/sassy-crick/Singularity-Easybuild/archive/refs/tags/
v1.0.1.tar.gz
```

This way, our installation is reproducible and after all that is one of the reasons why we are using containers in the first place. To make things a bit easier to follow, we are creating two directories:

```
$ mkdir github singularity
```

We move the downloaded tarball into the directory called 'github' and unpack it there:

```
$ tar -xvf v1.0.1.tar.gz
```

Exercise: build my first container

Once the tarball is unpacked, we change into the subdirectory where the scripts are:

```
$ cd Singularity-Easybuild-1.0.1/scripts
```

The script we are interested in is called 'install.sh'. This will install the scripts which are needed in order to create the Definition File and / or build the container as well.
It is a good idea to have these scripts not in the current directory, but in a different one so we are not messing up the current one. Thus, we are doing that in our 'singularity' directory:

```
$ cd ~/singularity
$ ~/github/Singularity-Easybuild-1.0.1/scripts/install.sh
```

You will be asked a number of questions when you are executing the script, like:
- Build/Create
- Debian, Ubuntu, CentOS
- (version of the used distribution)
- Lmod, Envmod
- create symbolic links in ~/bin directory

All of that can be put conveniently into one command line like this example:

```
$ ~/github/Singularity-Easybuild-1.0.1/scripts/install.sh Create Ubuntu Focal Lmod n
```

This would only *Create* the Definition Files, using *Ubuntu Focal*, with *Lmod*, and install the script in your current directory. Please play around a bit with the options.

Exercise: build my first container

Once this is done you should have the correct links in your current directory.
Note: there is currently a bug in the GitHub script which is not installing the files in the ~/bin directory correctly.
Executing it like that for example:

```
$ ./container-create-ubuntu20.04-lmod.sh bzip2-1.0.6.eb
```

Will tell you that currently you do not have the file ~/.singularity/sing-eb.conf and it will tell you why that might be imporant. Please have a read here.
You then will be asked if you want to add another EasyConfig (EC) file. For now, we say 'n' here. And that is it, you have successfully created you first Definition File! You now can follow the printout instructions of how to build the container. This is quite useful if you got access to a fast build host, which will build the container for you. You can do this two step process also into one, provided you have Singularity installed:

```
$ container-build-ubuntu20.04-lmod.sh bzip2-1.0.6.eb n
```

Note this time around, I have added the 'n' as I don't want to use a second EC file. Now I am getting asked if I want to build a sandbox as well. We say 'y' here, simply for demonstation purpose. The build will take a moment, so time to grab a drink if you want to.

NIHR | Guy's and St Thomas' Biomedical Research Centre

KING'S College LONDON

Exercise: build my first container

Once the builds are done, you should have the following files and folders inside your singularity folder:

```
$ ls
bzip2-1.0.6-ubuntu-focal-lmod
bzip2-1.0.6-ubuntu-focal-lmod.sif
Singularity.bzip2-1.0.6-ubuntu-focal-lmod
```

Here, the Singularity.bzip2-1.0.6-ubuntu-focal-lmod file is the Definition file which was used to build the container bzip2-1.0.6-ubuntu-focal-lmod.sif and, in our case, also the sandbox folder bzip2-1.0.6-ubuntu-focal-lmod .
Have a look inside the Definition File and try to understand what it does!

```
$ less Singularity.bzip2-1.0.6-ubuntu-focal-lmod
```

As you can see, we are using a 'speaking' filename for the files and folder. Just looking at the name, you will see which Linux distribution and version we are using, which module tools we use and what software we are using. This Definition File can be shared, for example in a publication, so others can repeat what you done.

What do I need to bear in mind when I am creating and using a container?

There are like aways in life some caveats. As we all know, the CPU architecture is changing on a regular base. So there are some implications of that. If you are building your container on a very old machine and you want to run it on a very new one, you might not get the full performance of the new CPU as some new features are not being used. The more worse case is you have a shiny new cutting edge CPU but the machine where you want to use the container is older. There, your program might contain instructions for the CPU which the CPU does not have and thus the program is crashing.
Another, practical consideration is that some programs take a long time to install as there are plenty of requirements. Typical examples, and by no means the only ones, are R, TensorFlow and suchlike. Two things can happen here during the container creation process:

- the URL of the source code is (currently) not available for one reason or another
- some test jobs are failing.

NIHR | Guy's and St Thomas' Biomedical Research Centre

KING'S College LONDON

What do I need to bear in mind when I am creating and using a container?

For the first problem, what the container during the build process does is downloading the source code first. So if there is a problem, the build crashes early and not after hours of build.

The second problem is a bit more tricky. Although the EC are thoroughly tested, there still might be a problem with a particular program on particular hardware etc. If that happens, please open an issue on the GitHub page so the problem can be addressed and hopefully solved.

I got a docker image, can I use that one on Rosalind?

Fortunately, Docker containers can be converted to Singularity images and that can be done on Rosalind like this:

```
$ module add apps/singularity/3.5.3
$ singularity build ubuntu.img docker://ubuntu:latest
```

In this example, we first load the Singularity module, version 3.5.3 in this case, and then build the *ubuntu.img* from the Docker Hub.
You will then end up with a file called which is the same as the .sif files we created ourself.

How do I run a container on Rosalind?

It is easy to use Singularity containers on Rosalind like this:

```
$ module add apps/singularity/3.5.3
$ singularity run bzip2-1.0.6-ubuntu-focal-lmod.sif bzip2 -version
```

In this example, we use the image we just build and check the version of bzip2. This is, of course, a bit of a silly way to use containers but it is a useful example as the container is building quickly.

Any questions?

NIHR | Guy's and St Thomas' Biomedical Research Centre

KING'S College LONDON

Day 2

Why do I want to open up a container?

There might be good reasons for opening a container again. For example, you want to install additional software, which is not in the EasyBuild repository. You might need to install a large software package and you are not sure if all of that is working the way it should be. You might need to install your own program. So there are a number of reasons why you want to do that.

It should be noted, however, if you are doing that you need to bear in mind that the original Definition File is no longer valid for that container. You either need to amend that file, or have keep a log of what you are doing so it can be reproduced!

How do I open up a container?

You might recall from the Day one exercise that at one point you got asked if you want to build a sandbox as well. If you are planning to open a container later, that might be the best way to do it. This way, you will get both, the Singularity container and the sandbox.
However, the build time is twice as much as the whole software needs to be build twice. This is probably not what you want to do, unless you got a quick build.
So probably the better way of doing it is build the Singularity container first, with the base software you want to use, and then open up that container to get the Sandbox. This way, if things are going wrong, you still have the original container. In order to do that without root or sudo access, you will need to have have fakeroot installed!
The way to do that is like this:

```
$ singularity build --sandbox –fakeroot \
bzip2-1.0.6-ubuntu-focal-lmod bzip2-1.0.6-ubuntu-focal-lmod.sif
```

This instructs Singularity to build a *sandbox*, using *fakeroot*, with the name of the sandbox being *bzip2-1.0.6-ubuntu-focal-lmod* based on the container *bzip2-1.0.6-ubuntu-focal-lmod.sif*. Instead of a container you also can use the Definition File as well.

How do I install additional software inside a container?

Now that we have the sanbox, we want to go inside and install additional software in it. After all, that is the whole reason for this exercise. We use again Singularity for that:

```
$ singularity shell -w --fakeroot bzip2-1.0.6-ubuntu-focal-lmod
```

Inside the container, we need to become the user easybuild:

```
> su -l easybuild
```

Now we want to install more software, conveniently done by EasyBuild. For this example, we install another compression program, zlib, like this:

```
$ eb -rd zlib-1.2.8.eb
```

You can check if all of that has been installed ok if you are running this module command:

```
$ ml spider –ignore-cache
```

Which will list all currently installed software. The '--ignore-cache' means that the module-cache will be rebuild. Without that you might not get the expected result. One thing we need to take note of is the module-name of the newly installed software. We will need that in the next step but first we become root again by simply typing exit.

How do I install additional software inside a container?

Now that we have successfully installed our new highly sophisticated software package, we need to make sure the it is actually being loaded when you run the container later. As we are root, we are looking into this file here:

```
> less /environment
```

As the file reads right at the bottom, currently the container is loading the bzip2 module:
module load bzip2/1.0.6
We need to change this with a text editor of your choice. Fortunately, we can do that outside the container! So we leave the container completely and are back as normal user. Here we can use our familiar tools to change that line. Remember the name of the module? I hope you do, as that is the one we need now. Change into the sandbox directory and simply replace the line above in the environment file with:
module load zlib/1.2.8
Leave the sandbox directory.

How do I build the container again?

Having done all the minimalistic changes, we can build the container again. This is similar to the way we created the sandbox, thus:

```
$ singularity build --fakeroot \
zlib-1.2.8-ubuntu-focal-lmod.sif bzip2-1.0.6-ubuntu-focal-lmod
```

This will create the new container named zlib-1.2.8-ubuntu-focal-lmod.sif. As we only installed a library and not a program, we cannot check easily it as before. However, we can do this:

```
$ singularity run zlib-1.2.8-ubuntu-focal-lmod.sif export \
| grep LIBRARY_PATH
```

You should see something like this:
export LD_LIBRARY_PATH='/app/software/zlib/1.2.8/lib:/.singularity.d/libs'
export LIBRARY_PATH='/app/software/zlib/1.2.8/lib'

As we know that the */app/software* path is **inside** the container, all should be well.

SUMMARY

In this two day workshop, we learned about containers. Why they have a right in their own to exist. What makes them special. We learned how to create Singularity Definition Files with the set of scripts which are provided. We learned how to build a container, how to open them up again and add software to it and rebuild it. We learned how to use a container.

I hope you found this workshop useful and you got a few things to take away from.

NIHR | Guy's and St Thomas' Biomedical Research Centre

KING'S College LONDON

Any questions?