

HCS7004 – Session 13

Software and containers

September 16th, 2020

CFAES



THE OHIO STATE UNIVERSITY
COLLEGE OF FOOD, AGRICULTURAL,
AND ENVIRONMENTAL SCIENCES

Agenda for today

- Some commands and concepts:
sudo recap, the **\$PATH** environment variable, and remote file transfer
- Avoiding dependency hell
- Increasing reproducibility and portability of software environments
using **conda** and **containers**

sudo recap

- Use **sudo** to temporarily gain administrative rights for tasks such as software installation with a package manager
- To run a command in sudo mode, simply prepend “sudo”:

```
jelmer@jpi:~$ apt-get update
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
```

sudo recap

- Use **sudo** to temporarily gain administrative rights for tasks such as software installation with a package manager
- To run a command in sudo mode, simply prepend “sudo”:

```
jelmer@jpi:~$ apt-get update  
Reading package lists... Done  
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)  
  
jelmer@jpi:~$ sudo apt-get update  
[sudo] password for jelmer:  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
```



sudo recap

- Use **sudo** to temporarily gain administrative rights for tasks such as software installation with a package manager
- To run a command in sudo mode, simply prepend “sudo”:

```
jelmer@jpi:~$ apt-get update  
Reading package lists... Done  
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)  
  
jelmer@jpi:~$ sudo apt-get update  
[sudo] password for jelmer:  
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
```



- At OSC, you will **not** be able to use sudo

The \$PATH environment variable

- \$PATH is one of several *environment variables*.
- It lists the directories that will be searched when you type a program's name. This allows you to run programs without specifying the full path to the executable.

```
jelmer@jpi:~$ vcftools
```

```
VCFtools (0.1.16)
```

```
© Adam Auton and Anthony Marcketta 2009
```

The \$PATH environment variable

- **\$PATH** is one of several *environment variables*.
- It lists the directories that will be searched when you type a program's name. This allows you to run programs without specifying the full path to the executable.

```
jelmer@jpi:~$ vcftools
```

```
VCFtools (0.1.16)
```

```
© Adam Auton and Anthony Marcketta 2009
```

```
jelmer@jpi:~$ which vcftools  
/usr/bin/vcftools
```

“which” to check location

The \$PATH environment variable

- To check which dirs are in your **\$PATH**:

```
jelmer@jpi:~$ echo $PATH  
/home/jelmer/.local/bin:/home/jelmer/bin:/home/jelmer/.yarn/bin:/usr/local/go/bin
```

The \$PATH environment variable

- To check which dirs are in your **\$PATH**:

```
jelmer@jpi:~$ echo $PATH  
/home/jelmer/.local/bin:/home/jelmer/bin:/home/jelmer/.yarn/bin:/usr/local/go/bin
```

- **\$PATH** is a *variable* (hence the dollar sign) and can be modified directly.
To add a dir to your path:

```
jelmer@jpi:~$ PATH=$PATH:~/new-dir-with-software
```

The \$PATH environment variable

- To check which dirs are in your **\$PATH**:

```
jelmer@jpi:~$ echo $PATH  
/home/jelmer/.local/bin:/home/jelmer/bin:/home/jelmer/.yarn/bin:/usr/local/go/bin
```

- **\$PATH** is a *variable* (hence the dollar sign) and can be modified directly.
To add a dir to your path:

```
jelmer@jpi:~$ PATH=$PATH:~/new-dir-with-software
```

- For *lasting* changes, edit your bash configuration files (like `~/.bashrc`),
from which **\$PATH** is loaded whenever you open a terminal

Remote file transfer

- Use sftp to establish contact between a local and a remote (OSC) computer:

```
jelmer@jpi:~$ sftp jelmer@sftp.osc.edu
Connected to sftp.osc.edu.
sftp> █
```

Remote file transfer

- Use sftp to establish contact between a local and a remote (OSC) computer:

```
jelmer@jpi:~$ sftp jelmer@sftp.osc.edu  
Connected to sftp.osc.edu.  
sftp> █
```

- Use “put <file> [target-dir]” to upload to remote:

```
sftp> put multiqc_1.9--py_1.sif          # No target dir specified: goes to home  
Uploading multiqc_1.9--py_1.sif to /users/PAS0471/jelmer/multiqc_1.9--py_1.sif
```

Remote file transfer

- Use sftp to establish contact between a local and a remote (OSC) computer:

```
jelmer@jpi:~$ sftp jelmer@sftp.osc.edu  
Connected to sftp.osc.edu.  
sftp> █
```

- Use “put <file> [target-dir]” to upload to remote:

```
sftp> put multiqc_1.9--py_1.sif          # No target dir specified: goes to home  
Uploading multiqc_1.9--py_1.sif to /users/PAS0471/jelmer/multiqc_1.9--py_1.sif  
  
sftp> put multiqc_1.9--py_1.sif /fs/scratch/PAS1752/jelmer/  
Uploading multiqc_1.9--py_1.sif to /fs/scratch/PAS1752/jelmer/multiqc_1.9--py_1.sif
```

Remote file transfer

- Use sftp to establish contact between a local and a remote (OSC) computer.
- Other, even simpler methods include the command “scp” (and “rsync”), but OSC prefers you to use sftp so as to use nodes specifically reserved for file transfer.

Dependency hell

- As you heard in the previous session, many pieces of software have several **dependencies**, sometimes so many that installation can become tricky
- It gets worse: different (versions of) software can have mutually exclusive dependencies – “**dependency hell**”

Software management in Ubuntu

- In Ubuntu, the Advance Package Tool automatically handles dependencies and updates – for instance with the **apt-get** command.

```
jelmer@jpi:~$ sudo apt-get update      # Update package index first
```

```
jelmer@jpi:~$ sudo apt-get install -y vcftools bcftools  # Install
```

Software management in Ubuntu

- In Ubuntu, the Advance Package Tool automatically handles dependencies and updates – for instance with the **apt-get** command.

```
jelmer@jpi:~$ sudo apt-get update          # Update package index first
```

```
jelmer@jpi:~$ sudo apt-get install -y vcftools bcftools # Install
```

-y: don't prompt for confirmation

>1 package at once works

Software management in Ubuntu

- In Ubuntu, the Advance Package Tool automatically handles dependencies and updates – for instance with the **apt-get** command.

```
jelmer@jpi:~$ sudo apt-get update      # Update package index first
```

```
jelmer@jpi:~$ sudo apt-get install -y vcftools bcftools  # Install
```

```
jelmer@jpi:~$ sudo apt-get upgrade vcftools  # Update a package
```

```
jelmer@jpi:~$ sudo apt-get upgrade          # Update all packages
```

```
jelmer@jpi:~$ apt-cache search vcftools    # Search for a package
```

vcftools - Collection of tools to work with VCF files

Software management in Ubuntu

- In Ubuntu, the Advance Package Tool automatically handles dependencies and updates – for instance with the **apt-get** command.
- This is very useful, **but what if:**
 - Software isn't available in the package manager?
 - You need a specific (older) version of software?
 - You need different versions at different times?
 - There are mutually exclusive dependencies for software you all want?

Similarly, at OSC

- Using specific versions & version switching is made easy with module
- But, when software isn't installed, lack of admin rights makes software installation at OSC even more challenging than usual...

The alternatives

- Two main alternatives:
 - **conda**
 - **containers** (*Docker* and *Singularity*)
- These alternatives are useful *not only* at OSC – but more generally, for:
 - Reproducible environments
 - Portable environments (especially containers)
 - Avoidance of dependency hell

conda: a software and environment manager

- Creates *environments* with one (= best-practice) or more software packages
- Lots of bioinformatics software is available as a conda package
- Handles dependencies but does not require admin rights
- Environments are activated and deactivated – much like *Lmod* at OSC

conda: benefits and limits

- Benefits of conda:
 - Easy to use specific software versions
 - Easy to have multiple versions of the same software available
 - Easy to use mutually incompatible software
 - Save and share environment instructions
 - Not much of a learning curve
 - Available at OSC

conda: benefits and limits

- **Benefits of conda:**
 - Easy to use specific software versions
 - Easy to have multiple versions of the same software available
 - Easy to use mutually incompatible software
 - Save and share environment instructions
 - Not much of a learning curve
 - Available at OSC
- **Limits of conda** – it does not:
 - Create an isolated environment that can be exported and used anywhere
 - Allow running software requiring a different OS / OS version

Using conda

- To create a new environment for multiqc, activate it, and install multiqc:

```
[jelmer@owens-login01 ~]$ module load python/3.6-conda5.2 # Load Python+conda module  
[jelmer@owens-login01 ~]$ conda create -n multiqc arbitrary environment name # Create a new environment
```

Using conda

- To create a new environment for multiqc, activate it, and install multiqc:

```
[jelmer@owens-login01 ~]$ module load python/3.6-conda5.2 # Load Python+conda module
[jelmer@owens-login01 ~]$ conda create -n multiqc          # Create a new environment
                                                               arbitrary environment name
```

```
[jelmer@owens-login01 ~]$ conda activate multiqc          # Activate the environment
(multiqc) [jelmer@owens-login01 ~]$ conda install -c bioconda multiqc # Install multiqc
Solving environment: / █
```

Using conda

- To create a new environment for multiqc, activate it, and install multiqc:

```
[jelmer@owens-login01 ~]$ module load python/3.6-conda5.2 # Load Python+conda module  
[jelmer@owens-login01 ~]$ conda create -n multiqc # Create a new environment  
arbitrary environment name
```

```
[jelmer@owens-login01 ~]$ conda activate multiqc # Activate the environment  
(multiqc) [jelmer@owens-login01 ~]$ conda install -c bioconda multiqc # Install multiqc  
Solving environment: /
```

Tells us the conda environment we're in

specify "channel" – bioconda has
bioinformatics software

Using conda

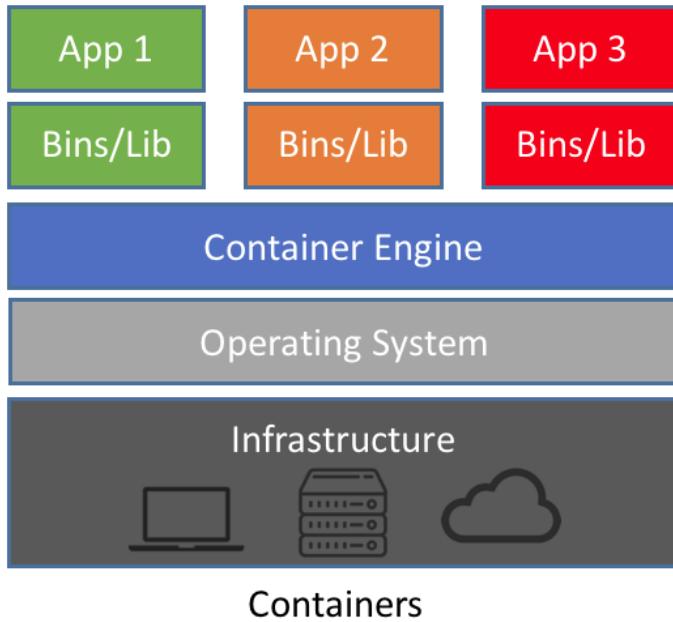
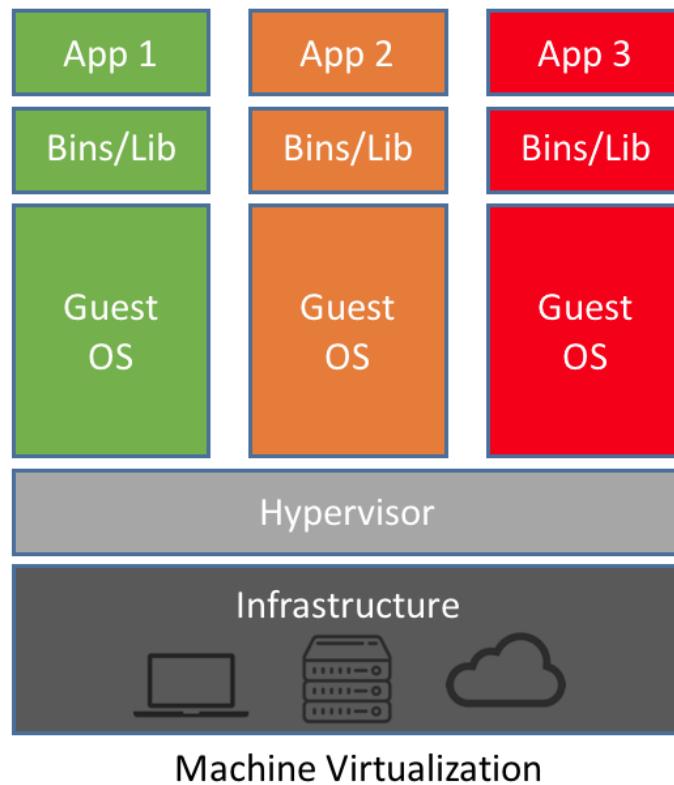
- To create a new environment for multiqc, activate it, and install multiqc:

```
[jelmer@owens-login01 ~]$ module load python/3.6-conda5.2 # Load Python+conda module  
[jelmer@owens-login01 ~]$ conda create -n multiqc # Create a new environment  
                                         arbitrary environment name
```

```
[jelmer@owens-login01 ~]$ conda activate multiqc # Activate the environment  
(multiqc) [jelmer@owens-login01 ~]$ conda install -c bioconda multiqc # Install multiqc  
Solving environment: / █
```

```
(multiqc) [jelmer@owens-login01 ~]$ multiqc --help  
Usage: multiqc [OPTIONS] <analysis directory>
```

Containers: isolated environment for software



VMs versus containers:

- Containers don't virtualize entire Operating Systems:
 - Containers are smaller
 - Containers are more lightweight to run
- Use a VM for long interactive sessions with multiple programs
- Use a container to run one or a few programs non-interactively

What can I do with containers

- Run an application with complicated stack dependencies in a few keystrokes.
- Package a workflow so that it can run on your laptop, in the cloud, and at OSC.
- Publish a paper and include a link to a container with all of the data and software.
- Create a workflow where different programs run on different operating systems.

Docker versus Singularity

- Docker and Singularity are two different container platforms
- Docker is by far the most widely used

Docker versus Singularity

- Docker and Singularity are two different container platforms
- Docker is by far the most widely used
- However, Singularity:
 - Was specifically designed for HPCs & available at OSC
 - Needs no admin rights to run containers
 - Can work with Docker images but not vice versa
 - Integration with several host directories by default
 - Has single-file container images
 - Is open-source

Getting a container

- Container images for most software can be found at these registries:
 - Docker Hub: <https://hub.docker.com/>
 - Sylabs Cloud Library: <https://cloud.sylabs.io/>
 - Biocontainers (for bioinformatics software): <https://biocontainers.pro/>
- If no suitable image is available, you can build your own!

Getting a container

- Container images for most software can be found at these registries:
 - Docker Hub: <https://hub.docker.com/>
 - Sylabs Cloud Library: <https://cloud.sylabs.io/>
 - Biocontainers (for bioinformatics software): <https://biocontainers.pro/>
- If no suitable image is available, you can build your own!

Container terminology

- | | |
|--|--|
| • <u>Image</u> | File(s) containing the container application |
| • <u>Container</u> | A <i>running</i> container image |
| • <u>Definition file</u> (Singularity)
<u>Dockerfile</u> (Docker) | Recipe to build a container image |

Downloading a container image

- Download (“pull”) a container with multiqc:

```
(base) jelmer@jpi:~$ singularity pull docker://quay.io/biocontainers/multiqc:1.9--py_1  
INFO:    Converting OCI blobs to SIF format
```

Downloading a container image

- Download (“pull”) a container with multiqc:

```
(base) jelmer@jpi:~$ singularity pull docker://quay.io/biocontainers/multiqc:1.9--py_1  
INFO:    Converting OCI blobs to SIF format
```

*Registry://user/project/image-name:image-tag
(version)*

Downloading a container image

- Download (“pull”) a container with multiqc:

```
(base) jelmer@jpi:~$ singularity pull docker://quay.io/biocontainers/multiqc:1.9--py_1  
INFO:    Converting OCI blobs to SIF format
```

Conversion necessary because it is a Docker image

Registry://user/project/image-name:image-tag
(version)

Downloading a container image

- Download (“pull”) a container with multiqc:

```
(base) jelmer@jpi:~$ singularity pull docker://quay.io/biocontainers/multiqc:1.9--py_1  
INFO:    Converting OCI blobs to SIF format
```

```
jelmer@jpi:~$ ls multiqc_1.9--py_1*  
multiqc_1.9--py_1.sif
```

Now we have a “.sif”
container image

Downloading a container image

- Download (“pull”) a container with multiqc:

```
(base) jelmer@jpi:~$ singularity pull docker://quay.io/biocontainers/multiqc:1.9--py_1
INFO:    Converting OCI blobs to SIF format
```

- Download a container from Singularity hub (“shub”):

```
(base) jelmer@jpi:~$ singularity pull shub://singularityhub/hello-world
INFO:    Downloading shub image
59.75 MiB / 59.75 MiB [=====] 100.00%
```

Running a container

- “run” the container (= run the default script):

```
jelmer@jpi:~$ singularity run multiqc_1.9--py_1.sif  
Singularity> █
```

*# In this case, we get a
terminal inside the container*

Running a container

- “run” the container (= run the default script):

```
jelmer@jpi:~$ singularity run multiqc_1.9--py_1.sif  
Singularity> █
```

*# In this case, we get a
terminal inside the container*

```
Singularity> multiqc --help  
Usage: multiqc [OPTIONS] <analysis directory>
```

Running a container

- “run” the container (= run the default script):

```
jelmer@jpi:~$ singularity run multiqc_1.9--py_1.sif  
Singularity> █
```

*# In this case, we get a
terminal inside the container*

- The shell sub-command always opens a terminal in the container:

```
jelmer@jpi:~$ singularity shell multiqc_1.9--py_1.sif  
Singularity> █
```

Running a container

- “run” the container (= run the default script):

```
jelmer@jpi:~$ singularity run multiqc_1.9--py_1.sif  
Singularity> █
```

*# In this case, we get a
terminal inside the container*

- The shell sub-command always opens a terminal in the container:

```
jelmer@jpi:~$ singularity shell multiqc_1.9--py_1.sif  
Singularity> █
```

- The exec sub-command lets you directly execute a shell command – and exits:

```
jelmer@jpi:~$ singularity exec multiqc_1.9--py_1.sif multiqc --help  
Usage: multiqc [OPTIONS] <analysis directory>  
jelmer@jpi:~$ █
```

“Containerizing” an analysis / script

- Running a locally installed program:

```
jelmer@jpi:~$ multiqc --help
```

- Run the same program using a container:

```
jelmer@jpi:~$ singularity exec multiqc_1.9--py_1.sif multiqc --help
```

“Containerizing” an analysis / script

- Running a locally installed program:

```
jelmer@jpi:~$ multiqc --help
```

- Run the same program using a container:

```
jelmer@jpi:~$ singularity exec multiqc_1.9--py_1.sif multiqc --help
```

```
jelmer@jpi:~$ singularity exec multiqc_1.9--py_1.sif \  
>     multiqc --help
```

Building your own container

Building a container does require admin rights –
Need to do this on your own computer or VM with A Linux operating system

- 1) Download a suitable base image (e.g., Ubuntu 20.04)

```
jelmer@jpi:~$ singularity pull docker://ubuntu:20.04
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
```

Building your own container

Building a container does require admin rights –
Need to do this on your own computer or VM with A Linux operating system

- 1) Download a suitable base image (e.g., Ubuntu 18.04)
- 2) Run the image and enter the container in interactive, writeable mode
- 3) Install the required software
- 4) When you exit the container, the image will have been updated

Building your own container

Building a container does require admin rights –

Need to do this on your own computer or VM with A Linux operating system

- 1) Download a suitable base image (e.g., Ubuntu 18.04)
- 2) Run the image and enter the container in interactive, writeable mode
- 3) Install the required software
- 4) When you exit the container, the image will have been updated

- 5) Best-practice – next create a Dockerfile:
 - Reproduce commands that worked & were necessary for installation
 - Add some metadata and formalities

Recap

- sudo, \$PATH, and remote file transfer with sftp
- Use of apt-get and Lmod (module)
- Alternatives to package managers and manual installation:
 - conda
 - Containers – Singularity for working at HPCs such as OSC