

# Deciding on the ideal software environment

---

Personal experiences with Anaconda, Docker, virtual machines, and servers

Raymond Wan and Patrick Yizhi Cai  
Manchester Institute of Biotechnology  
4 September 2024

# Outline

## Introduction

Overview of each

What has gone wrong?

Summary

# Introduction

Prior to project development, two main considerations:

- operating systems (i.e., Microsoft Windows, MacOS, or Linux)
- programming language (i.e., Python, C/C++, Java, etc.)

A third but equally important consideration is the *environment*.

Some options include:

- Anaconda,
- Docker,
- virtual machines, and
- servers

i.e., “inside what” will the program run – the focus of this talk

# Introduction

The decision for operating system, programming language, and environment affects all of this:

- who should be hired to **develop** the project,
- who can be brought on to the project to **help**, and
- even how the software will be **maintained**.

# Outline

## Describe

1. briefly, what each of them are,
2. pros and cons of each, and
3. what has gone wrong?

# Motivation for this talk

Each promises reproducibility and ease of software maintenance.  
Including:

- Reproducing a system from backups on to the same or other system.
- Long-term support of a piece of software, possibly by someone who was not the original developer.

But are they all equal? If they each promise the same thing, how would an RSE choose one over another?

## About me

Obtained a computer science degree in the 1990s and a PhD in computer science in 2004.

Relevance? My degrees pre-date all of these:

- Anaconda,
- Docker, and
- virtual machines

Like many of us, I had to learn them myself. Regrettably, it means I might not know them very well.

# Outline

Introduction

Overview of each

What has gone wrong?

Summary

# Servers

Physical computer (or server):

- It can be networked to another computer both inside (intranet) or outside (Internet) a workplace.
- Otherwise, no dependencies (or interferences) from other computers.



# Servers

## Pros:

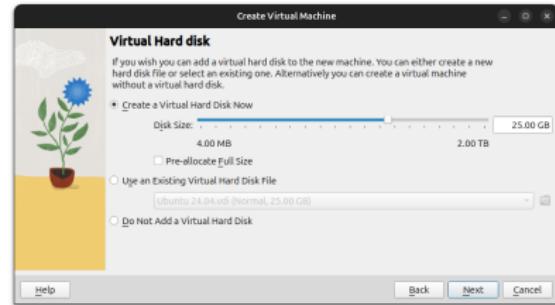
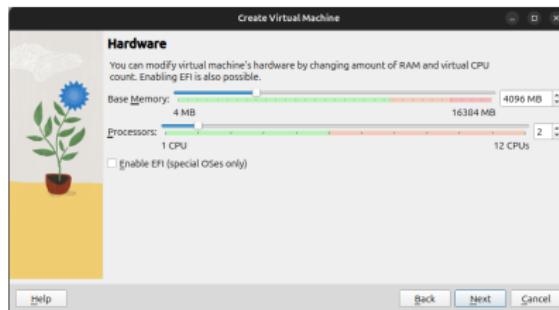
- Reproducible if you buy a similar computer and install the same software.
- Possible to purchase with a research budget since it is a **fixed** cost.

## Cons:

- Buying an entire server is a big investment.
- Keeping it requires physical space as well as electricity, cooling, physical security, etc.
- Someone in-house needs to look after it and act as system administrator.

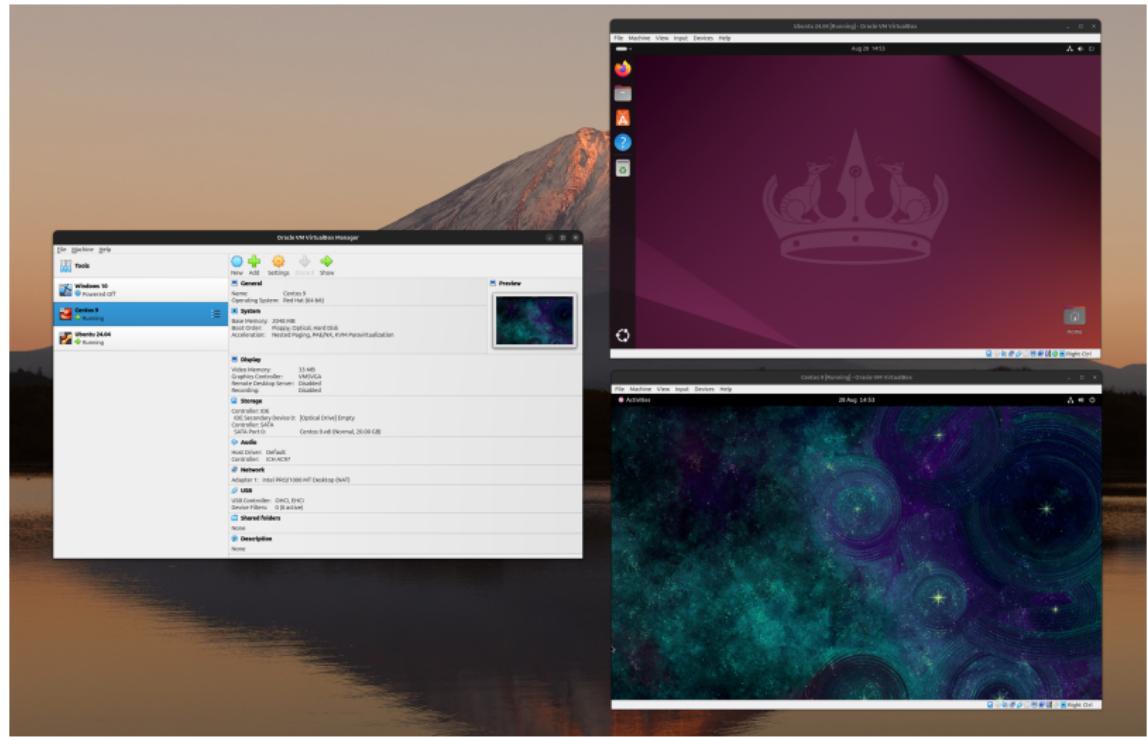
# Virtual machines

- Virtualisation<sup>1</sup> at the level of the physical computer.
- Host multiple **virtual** computers on **one** computer.
- On the virtualisation program, you allocate how much memory should be used, disk space, network, etc.
- Some **programs**: Oracle VirtualBox or Microsoft Windows Hyper-V. Amazon AWS employs the same approach.



1 Dividing of physical computer resources into separate units

# Virtual machines



# Virtual machines

## Pros:

- Each VM exists independently of other VMs.
- Virtual hard disks can be backed up, copied, and restored on another system.
- A snapshot on a live system is possible (with some caveats).

## Cons:

- Common memory, hard disk, and network means that a computationally intensive VM can take up resources from other VMs and the host system.
- Some projects require the location of the server and data to be known. This might rule out Amazon AWS.

# Docker

- Virtualisation at the level of the operating system, one level above virtual machines.
- Unlike before, **one** physical computer running **one** operating system. On this operating system, Docker Engine is continuously running.
- Docker Engine hosts various containers, which are “self-contained” software and configuration files.
- Multiple Docker containers can be running simultaneously that can communicate with each other.

# Docker

Example `docker-compose.yml` for starting up applications based on multiple containers:

```
version: '3.0'  
services:  
    helloworld:  
        container_name: hello-world  
        working_dir: /hello  
        volumes:  
            - /var/logs/helloworld:/myapp/logs  
        ...
```

- `/hello` and `/var/logs/helloworld` are directories on the **host** computer
- `/myapps/logs` is from the point of view of the application

# Docker

## Pros:

- Good for programs that start up, run, and end (i.e., for data processing/analysis, etc.).
- Some developers use it to write web applications.

## Cons:

- With less separation from the host compared to virtual machines, diagnosing problems (i.e., why does it work on this computer but not that one) can be frustrating.
- **Personally**, a steeper learning curve than the other options.

# Anaconda

- Able to manage separate instances of Python and R, as well as other open source programs.
- Software packages are grouped together into **Anaconda environments**.
- Same program at different versions can be installed in multiple environments by the same user.

Example setup on Ubuntu host:

Command	What it does
<u>sudo apt-get install bwa</u>	Install bwa on host
conda create --name rsecon24	Create an environment
conda activate rsecon24	Activate environment
conda install bwa	Install bwa in environment

# Anaconda

```
rwan@xps8930:~$ bwa 2>&1 | head -n 4

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.17-r1188
Contact: Heng Li <lh3@sanger.ac.uk>
rwan@xps8930:~$ which bwa
/usr/bin/bwa
rwan@xps8930:~$ env | grep ^PATH
PATH=/opt/miniconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:/snap/bin
rwan@xps8930:~$
rwan@xps8930:~$
rwan@xps8930:~$ conda activate rsecon24
(rsecon24) rwan@xps8930:~$ bwa 2>&1 | head -n 4

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.18-r1243-dirty
Contact: Heng Li <hli@ds.dfci.harvard.edu>
(rsecon24) rwan@xps8930:~$ which bwa
/home/rwan/.conda/envs/rsecon24/bin/bwa
(rsecon24) rwan@xps8930:~$ env | grep ^PATH
PATH=/home/rwan/.conda/envs/rsecon24/bin:/opt/miniconda3/condabin:/usr/local/sbin:/usr/local/
/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
(rsecon24) rwan@xps8930:~$
```

# Anaconda

```
rwan@xps8930:~$ bwa 2>&1 | head -n 4

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.17-r1188
Contact: Heng Li <lh3@sanger.ac.uk>
rwan@xps8930:~$ which bwa
/usr/bin/bwa
rwan@xps8930:~$ env | grep ^PATH
PATH=/opt/miniconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:/snap/bin
rwan@xps8930:~$
rwan@xps8930:~$
rwan@xps8930:~$ conda activate rsecon24
(rsecon24) rwan@xps8930:~$ bwa 2>&1 | head -n 4

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.18-r1243-dirty
Contact: Heng Li <hli@ds.dfci.harvard.edu>
(rsecon24) rwan@xps8930:~$ which bwa
/home/rwan/.conda/envs/rsecon24/bin/bwa
(rsecon24) rwan@xps8930:~$ env | grep ^PATH
PATH=/home/rwan/.conda/envs/rsecon24/bin:/opt/miniconda3/condabin:/usr/local/sbin:/usr/local/
/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
(rsecon24) rwan@xps8930:~$ 
```

# Anaconda

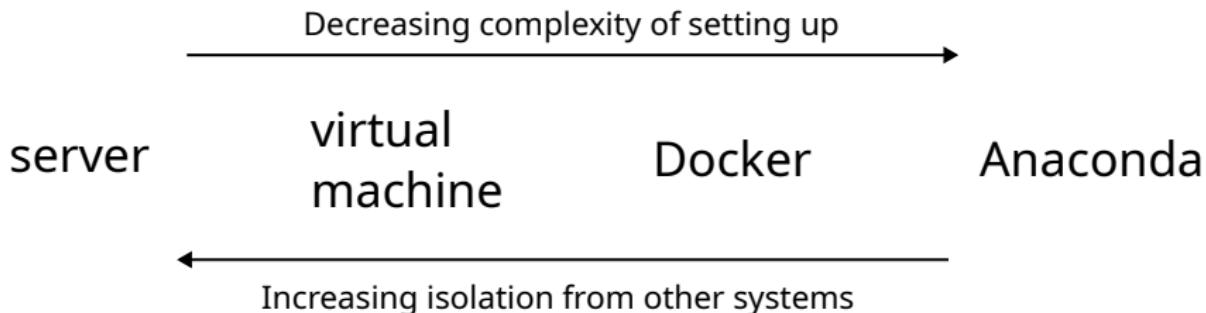
## Pros:

- Relatively very little setup. Anaconda can be installed by a user or the system administrator. Then, the user can create and delete their own Anaconda environments.
- Neither Oracle VirtualBox nor a Docker Engine-like system is running.
- Conceptually easier to understand; it's "just" updating the user's PATH.
- Ideal for data analysis; not useful for writing a web application.

## Cons:

- No separation between environments and between environments and the host system.

# Summary



We all want less complexity:

- Lower financial costs
- Anaconda environments can be created and maintained without system administrator access

More isolation makes it easier to reproduce an environment and/or troubleshoot.

# Outline

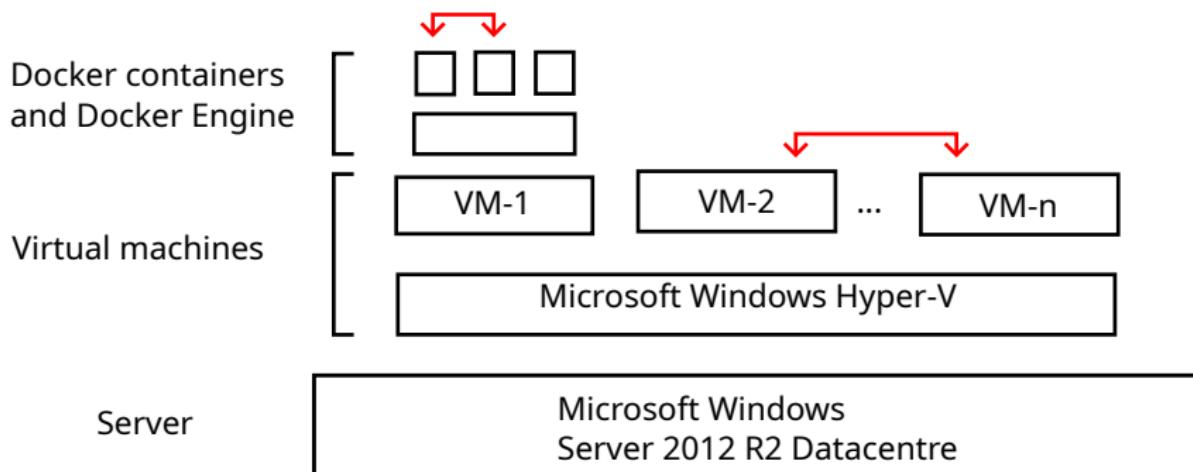
Introduction

Overview of each

What has gone wrong?

Summary

# Layered organisation



With communication (in red) between Docker containers and virtual machines.

At first glance, it should help with reproducibility...

# Layered organisation

In practice, when something goes wrong (i.e., network connection failed or one Docker container not communicating with another), finding the cause is difficult...

Occasionally, the server's operating system or a virtual machine's OS should be upgraded. If it fails, then anything that depends on it will be affected. Thus, many systems can fail at the **same time**.

Also, see [this](#).

As part of “software reuse”, most software depend on modules, libraries, packages, etc. created by others.

If one of those dependencies is ever removed or made redundant, then restoring the system from backups would be difficult. This is less likely when the system is first completed; and more likely during the software’s **maintenance period**.

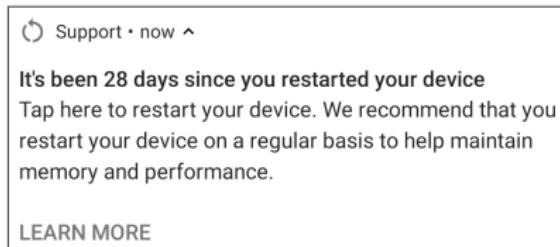
My own experience is that this is less likely at the operating system level (i.e., Debian/Ubuntu packages) and more likely when:

- Re-creating Docker containers
- Re-creating Anaconda environments

See [this](#) for some personal examples.

# Rebooting systems

Message on my phone:



Restarting computers and software as a solution used to be a punch-line to a joke. Now, it is actually recommended...

Due to the two previous reasons, we avoided restarting the entire system. Ultimately, we were delaying the inevitable.

# Outline

Introduction

Overview of each

What has gone wrong?

Summary

# Which to choose?

- Each has its pros and cons.
- If one option has issues, then it would have disappeared by now.
- Best is to know when to use one over another, instead of using one for everything.

# Which to choose?

- Server – If server or data location is an issue **and** if physical space and someone to look after it is available.
- Virtual machine – Alternative to buying a new server; but the host computer needs to be **powerful** enough to support multiple running operating systems.
- Docker – Slightly steeper learning curve than the other options, but worth the time invested as an **intermediate** option between virtual machines and Anaconda.
- Anaconda – Allow users without system administrator's access to create their own environments and install their own software. They can install **multiple** versions of Python or R (i.e., 4.4.1, 3.6.1, ...) in separate environments to accommodate their code.

## Other advice

Good to periodically **re-create** Docker containers and Anaconda environments from scratch on another system. Just to check that its external dependencies are all fine. Doing this when we're "free" means we can confront these problems on our own terms.

Lastly, I am not yet convinced if layers of environments is a good approach.

# Conclusion

The University of Manchester

None of these environments is a good substitute for good documentation.

That is, step-by-step instructions that explains how everything fits together.

# Acknowledgements

We thank

- colleagues in my current and all previous workplaces for their patience while I learned all this,
- reviewers for the helpful suggestions, and
- the organising committee for the opportunity to present our experiences.

Funding:

- Wellcome Trust Director's Discretionary Award (#221267/Z/20/Z)
- BBSRC Partner with international researchers on AI for Bioscience (#BB/Y513921/1)

# Contact details

Raymond Wan and Patrick Yizhi Cai

{raymond.wan,yizhi.cai}@manchester.ac.uk

Manchester Institute of Biotechnology, University of Manchester

LATEX slides: [https://github.com/rwanwork/Conf\\_RSECon\\_2024](https://github.com/rwanwork/Conf_RSECon_2024)

Homepage of the lab: <https://www.caillab.org/>

Thank you!

# License

LATEX Slides for RSECon 2024 © 2024 by Raymond Wan is licensed under Creative Commons Attribution 4.0 International.



See <https://creativecommons.org/licenses/by/4.0/> for further details.

## Additional credits

- $\text{\LaTeX}$  template from:  
<https://www.overleaf.com/latex/templates/university-of-manchester-presentation-beamer-template/rwcrzjmzcdyn>.
- Server clip art from [Clipart Library](#).
- Other figures made using [Inkscape](#) version 1.2.2

# End of Life

End of Life of software occurs when it is no longer supported by the original developer. No more support is given and no security updates are made available.

In our case, [support](#) for Microsoft Windows Server 2012 R2 Datacentre would have ended on 10 October 2023. So, a major update had to occur or else we would lose security updates and, likewise, the ability to connect to our organisation's network.

[Back](#)

## Removed packages

When a module, a library, or a package is no longer available, substituting it for a newer version might not be possible if the newer version is **not** backward compatible.

Also, another module that depends on it in turn might also have to be upgraded, causing a **ripple** effect.

In the past, solving such problems has taken me hours or days.

# Removed packages

In our case, a web application developed using Ruby on Rails had one of its dependencies (`mimemagic`) removed from all repositories due to a licensing issue. This occurred after our software had been developed and during its post-development maintenance period.

See this [page](#) for an explanation.

[Back](#)