

Software management



introduction

What are we defining as software management?

Controlling what is installed, which version, and how it can be recreated.

It's the difference between "it worked" and "it works".

Modern programming requires a lot of dependencies – code you did not create, but which you rely on. However, dependencies change with time, causing issues that you may not have anticipated.

The state of your machine, or of computers in general, also changes with time. Good software management ensures any future change will not impact the outcome of your pipelines.

introduction

Pharma industry

Analyses supporting a drug submission must be reproducible on demand, sometimes years later, by people who weren't there

Audit trails record who ran what, using which software version, on which data

introduction

Pharma industry

Analyses supporting a drug submission must be reproducible on demand, sometimes years later, by people who weren't there

Audit trails record who ran what, using which software version, on which data

Software industry

CI (continuous integration): every code change automatically triggers a build and a test suite, in a clean, defined environment

CD (continuous delivery/deployment): code that passes is packaged into a reproducible artifact (often a container) and shipped the same way every time.

introduction

The script wrangler

- Installs samtools, STAR and DESeq2 system-wide, one at a time, fixing errors as they appear
- Runs each step by hand, in order, remembering which output feeds which input
- Paths hardcoded to
`/home/maria/project_final_v3_actualfinal/`
- When a reviewer asks for a re-run, gets called up by their old lab and needs to get it to work again

introduction

The script wrangler

- Installs samtools, STAR and DESeq2 system-wide, one at a time, fixing errors as they appear
- Runs each step by hand, in order, remembering which output feeds which input
- Paths hardcoded to `/home/maria/project_final_v3_actualfinal/`
- When a reviewer asks for a re-run, gets called up by their old lab and needs to get it to work again

The pipeline manager

- Tools and exact versions declared in one environment file or container image
- Steps defined in a workflow, with explicit inputs, outputs and ordering
- Can share their repo with anyone in the world, who can then reproduce their results
- Re-running the code is possible 20 years later, even if they already died

libraries

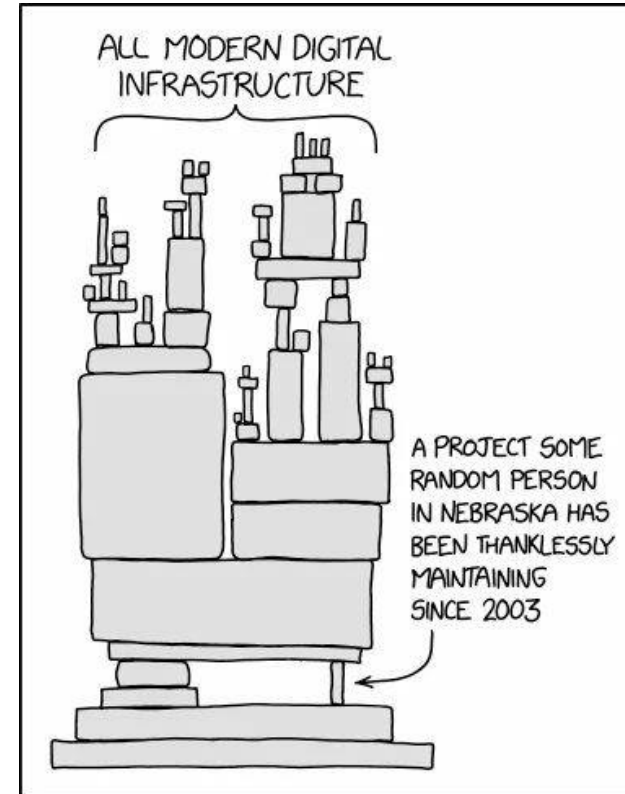
In a software context, libraries (or packages) are sets of reusable code someone else wrote and shared. Their complexity can run from anything as simple as formatting a string to full-blown analytical tools like `seurat`.

Libraries are usually tested, optimized and maintained by communities far larger than your lab. Even something simple might need maintenance because of security vulnerabilities, or to add compatibility with operating system updates.

dependencies

A dependency is any external library your code needs to run. While it would be best for reproducibility to not rely on any of them, in practice it would be prohibitive and potentially dangerous to do so.

Each dependency often has dependencies of their own. This means that while you are using library A, they might require library B to work. Worse, if you now want to use library C, this could require a different version of library B in order for it to work.



managing dependencies

Local installations

Installing things directly into a system (ie apt, pip, manual builds) can be very easy for prototyping, but painful for long-term maintenance and reproducibility, particularly if you have multiple projects running in the same system.

managing dependencies

Local installations

Installing things directly into a system (ie apt, pip, manual builds) can be very easy for prototyping, but painful for long-term maintenance and reproducibility, particularly if you have multiple projects running in the same system.

Environment managers

Isolated, declared environments (conda, venv, uv, pixi, etc.) take some more effort to set up. However, they allow you to easily recreate conditions for a project to run, at least in the same OS and architecture.

managing dependencies

Local installations

Installing things directly into a system (ie apt, pip, manual builds) can be very easy for prototyping, but painful for long-term maintenance and reproducibility, particularly if you have multiple projects running in the same system.

Environment managers

Isolated, declared environments (conda, venv, uv, pixi, etc.) take some more effort to set up. However, they allow you to easily recreate conditions for a project to run, at least in the same OS and architecture.

Containers

Packages the entire environment, OS tools and anything else needed to get the code to run (Docker, Apptainer, etc.). It is the best practice for reproducibility and portability, with the drawback that it can be cumbersome to set up and maintain.

dependencies

The left-pad incident

In 2016, a programmer took down a package called leftpad from npm (a package manager for Javascript) in protest against one of their decisions. Several giant companies like Facebook, Paypal and Netflix were using this dependency, and in some cases their CI/CD pipelines failed, causing delays and expenses.

While this instance was somewhat harmless, this case illustrates the risk of a supply-chain attack, in which the code is replaced by something harmful instead.

```
module.exports = leftpad;

function leftpad (str, len, ch) {
  str = String(str);

  var i = -1;

  ch || (ch = ' ');
  len = len - str.length;

  while (++i < len) {
    str = ch + str;
  }

  return str;
}
```

package managers

CRAN

1997 (R)

Bioconductor

2001 (R)

pip

2008 (Python)

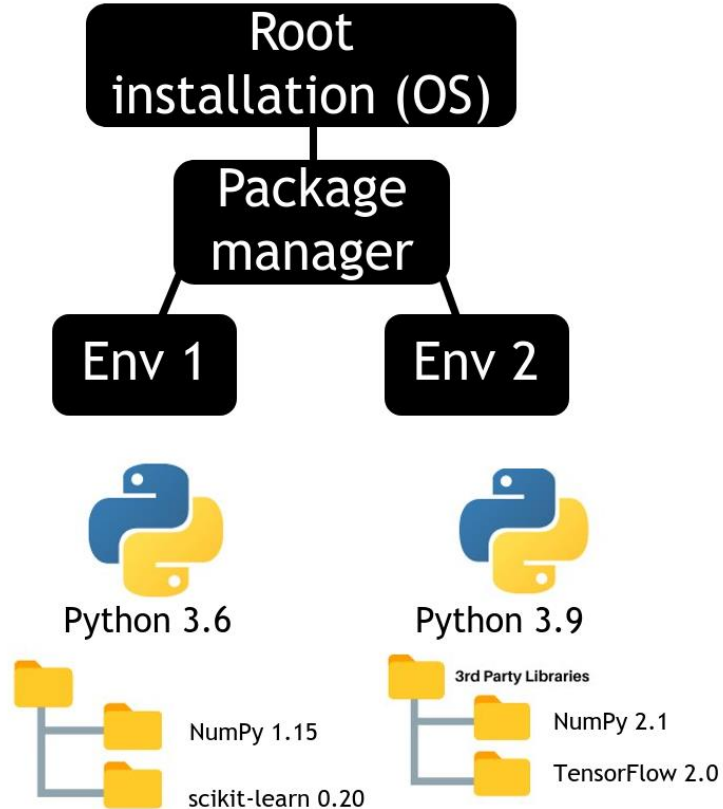
conda

2012 (multilanguage)

uv

2024 (Python)

environments



What is an environment?

An environment is a directory with a specific set of installed packages and tools. Since installation is self-contained and isolated, changes in one environment doesn't affect others.

This allows users to activate / deactivate to switch between them, and lets you keep conflicting versions side-by-side.

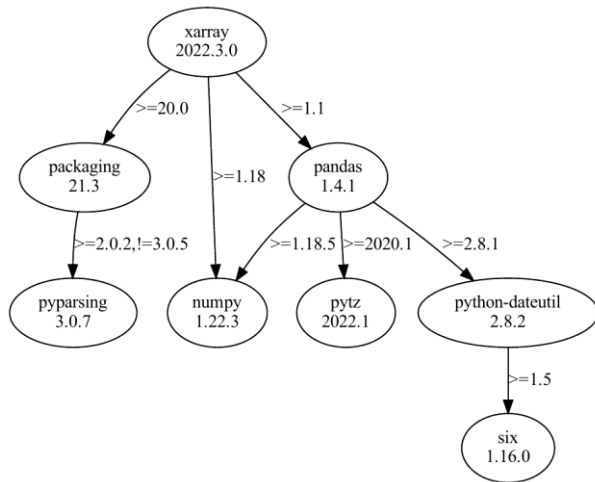
conda

As of today, conda is still one of the most popular package managers for data science, scientific computing and bioinformatics. It is open source, language-agnostic and free to use for academic purposes*.

We will focus on conda in this course because of its popularity. However, you should also check out pixi (which we will also mention) and uv (if using only python), which are more modern attempts that provide similar functionalities, with much faster performance.

While conda itself is free for academic use, Anaconda (the company that created it) is aggressive about enforcing its license in other contexts. Its default installation is also bloated. We recommend using Miniforge instead, which is slim and defaults to free channels.

conda



myenv_1.yml

channels:

- conda-forge
- nodefaults

dependencies:

- pandas==0.20.3
- statsmodels==0.8.0
- r-dplyr==0.7.0
- r-base==3.4.1
- python==3.6.0

Conda is useful for managing environments, because it takes care of many things in the background:

- Allows package installations on a host without admin privileges
- Assembles the dependency tree to find compatible versions of everything
- Capture all dependencies in one file, creating self-contained, reproducible projects

conda cheatsheet

```
# create a new environment
conda create -name mynewenv
```

```
# activate it
conda activate mynewenv
```

```
# leave it
conda deactivate
```

```
# remove it permanently
conda env remove -n mynewenv
```

```
#List all available environments (* indicates
active env)
conda env list
    Base /home/username/miniconda3/
    myenv * /home/username/miniconda3/envs/myenv
```

```
# export environment to a YAML file, capturing
complete env setup
conda env export --from-history > env.yml
```

```
# install a new environment from a file
conda env create -f env.yml -n mynewenv
```



~ 15 min



`hds-sandbox.github.io/HPC-lab`

workshop > HPC Pipes > Day 1 >

Package managers: conda
(first exercise)

Now is YOUR time!

- General knowledge about workflow managers
- Conda exercise: Understanding an existing environment

Imagine a colleague shares a conda environment with you—let's explore it and see what's installed.





Problems?
Comments?




conda channels

The software installation "recipes" used by conda are maintained by large communities of software developers. These communities are organized by channels. Here are two of the most popular and important ones, but there are many more.


Conda-forge: for scientific computing software
Bioconda: bioinformatics software (>8000 packages)



Search 

Packages in conda-forge

Filter items... 25651 packages loaded

 **TIP**
The following packages have recently received updates in [Anaconda.org](#). Check [conda-forge/feedstocks](#) for an overview of the latest commits in our feedstocks.

#	Package	Feedstock(s)	Metadata	Last updated
1	xsel	xsel-feedstock	Browse	Thu, 03 Oct 2024 12:23:29 GMT
2	xorg-xauth	xorg-xauth-feedstock	Browse	Thu, 03 Oct 2024 11:38:06 GMT

BIOCONDA® Package Index

Navigation

[FAQs](#)
[Contributing to Bioconda](#)
[Developer Docs](#)
[Tutorials](#)
[Browse packages](#)
[Bioconda @ Github](#)
[chat](#) [on gitter](#)

Search
packages & docs

[1](#) | [2](#) | [3](#) | [a](#) | [b](#) | [c](#) | [d](#) | [e](#) | [f](#) | [g](#) | [h](#) | [i](#) | [j](#) | [k](#) | [l](#) | [m](#) | [n](#)

1

[10x_bamtofastq](#)

2

[2pg_cartesian](#)

3

[3d-dna](#)
[3seq](#)

conda channels

two ways to add channels to your conda

```
conda config --append channels bioconda
conda config --append channels conda-forge
conda config --add channels genomedk
```

note that --add prepends:

```
conda config --show channels
channels:
```

- genomedk # highest priority
- bioconda
- conda-forge # lowest priority

disable the automatic activation of the base env when you open a new terminal session

```
conda config --set auto_activate_base false
```

only install packages from the highest-priority channel that contains the requested package

```
conda config --set channel_priority strict
```

Conda channels are not saved on an environment basis, but rather on a general config file (~/.condarc by default)

packages

```
# list all installed packages in the  
currently active env
```

```
conda list
```

```
# search for all available versions of a  
certain package
```

```
conda search samtools
```

```
Loading channels: done
```

```
# Name Version Build Channel
```

```
samtools 0.1.12 0 bioconda
```

```
samtools 0.1.12 1 bioconda
```

```
samtools 0.1.12 2 bioconda
```

```
""""""""""
```

```
samtools 1.9 h91753b0_3 bioconda
```

```
samtools 1.9 h91753b0_4 bioconda
```

```
samtools 1.9 h91753b0_5 bioconda
```

packages

```
# install package e.g. pandas
conda install pandas

# specify which version to install
conda install pandas=2.3.3

# update package
conda update pandas

# remove package
conda remove pandas
```

```
# save list of conda installed software
# in a particular env to a file (e.g.
# requirements.txt)
conda list --export > requirements.txt

# install the same software in your
# local environment
conda install --file=requirements.txt
```

pixi

pixi is a newer package manager built on top of the conda ecosystem. It pulls from the same channels (conda-forge, bioconda, etc.). It is written in Rust so it is **much** faster than conda.

There are some more differences under the hood, but the take-home message is that if you are already familiar with conda, you should give pixi a try as the logic and commands are quite similar.

Task	conda	pixi
Start a new environment/ project	<code>conda create -n myenv</code>	<code>pixi init myproject</code>
Add a (conda) package	<code>conda install numpy</code>	<code>pixi add numpy</code>
Add a PyPI package	<code>pip install requests</code>	<code>pixi add --pypi requests</code>
Remove a package	<code>conda remove numpy</code>	<code>pixi remove numpy</code>
Enter the environment	<code>conda activate myenv</code>	<code>pixi shell</code>
Run one command in it	<i>(activate first)</i>	<code>pixi run python script.py</code>
Leave the environment	<code>conda deactivate</code>	<code>exit</code>
List installed packages	<code>conda list</code>	<code>pixi list</code>
Recreate from a shared project	<code>conda env create -f environment.yml</code>	<code>pixi install</code> (<i>uses pixi.lock</i>)

pixi



Pixi

next-gen package manager for reproducible development setups

Easily switch between Conda and Pixi using YAML files

.pixi/ directory contains the installed environments

pixi.toml

pixi.lock

pixi.toml contains info pixi use to create the environment

pixi.toml

[workspace]

channels = ["conda-forge"]

name = "example"

platforms = ["linux-64"]

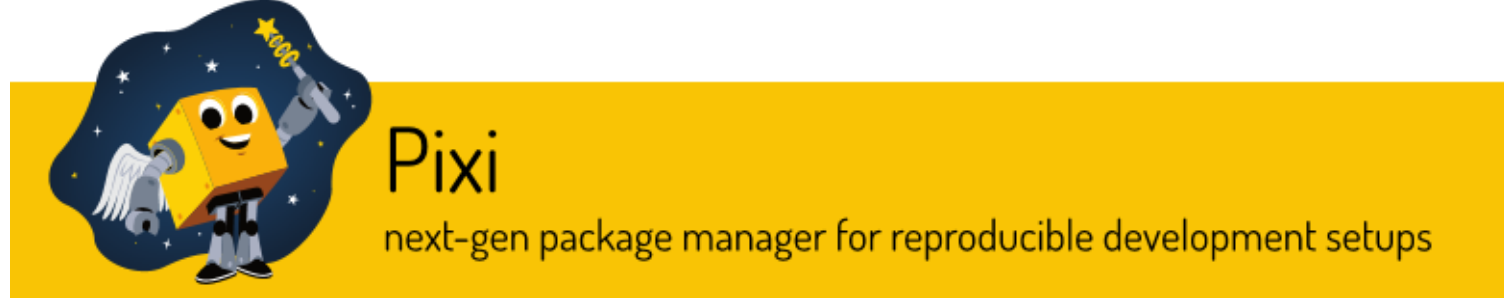
version = "0.1.0"

[tasks]

[dependencies]

python = ">=3.13.5,<3.14"

pixi



- **pixi.lock**: contains the definition of environments and packages
 - Environment: every environment in the workspace for every platform
 - Packages: package's URL on the conda package index and digests (e.g. sha256, md5).

Multi-platform projects

```
pixi workspace platform add linux-64 osx-arm64 win-64
```

Multi-environments

You can use features for building complex environment structures.

A **feature** can define *a part* of an environment and, by itself, is not useable. Features can be composed together to create an environment and they can be used across multiple environments.

```
> pixi add --feature A python
> pixi run --environment dev python
--version
> pixi shell --environment dev
```

pixi.toml

```
[feature.A.dependencies]
...

[feature.B.dependencies]
...

[feature.C.dependencies]
...

[environments]
A = ["A"]
two = ["A", "C"]
three = ["A", "B", "C"]
alternative = ["B", "A"]
```

summary

- **conda package installation**
- **conda environments**
- **portability of conda envs**
- **conda channels**
- **pixi**



~ 50 min



`hds-sandbox.github.io/HPC-lab`

workshop > HPC Pipes > Day 1 >

Package managers: conda
OR

Package managers: pixi

Now is YOUR time!

Option 1

- Build a new conda environment from scratch
- Install an environment from a YAML file and modify the included software and tools

Option 2

- Creating and sharing envs with Pixi

Bonus exercises (*If you have time*)



When to use a Package Manager?



[Conda cheat sheet](#)

- Same HPC platform (sharing OS and architecture)
- Quickly install and manage software
- Projects are relatively simple and do not require complex isolation
- To conduct repetitive data analyses using a pipeline



Advantages

Easily search for packages in a centralised **repository/registry** (e.g., Bioconda)

Do not require **administrative privileges** to install software & manage dependencies automatically

Can encapsulate different software version in environments

Allow recreation of the environment on other systems

Limitations

Not every software is available through Conda

Cannot address complex dependency conflicts (circular dependency and version conflict)

Can take a lot of **space** (tip: run **conda clean** to remove unused and cached packages)

Reproducibility: some packages may have different versions or may not be available on other operating systems and architectures.



Functional package managers

Provide a reproducible build process that avoids circular dependencies by **building each package in a sandboxed environment**.

The source code and dependencies are **uniquely identified by a hash value**, ensuring that packages with different hashes are built in separate, isolated environments.

This prevents dependency conflicts and allows for concurrent, independent builds of packages, even if they have conflicting dependencies or different versions of the same software.

Examples: Nix and Guix



Software
management

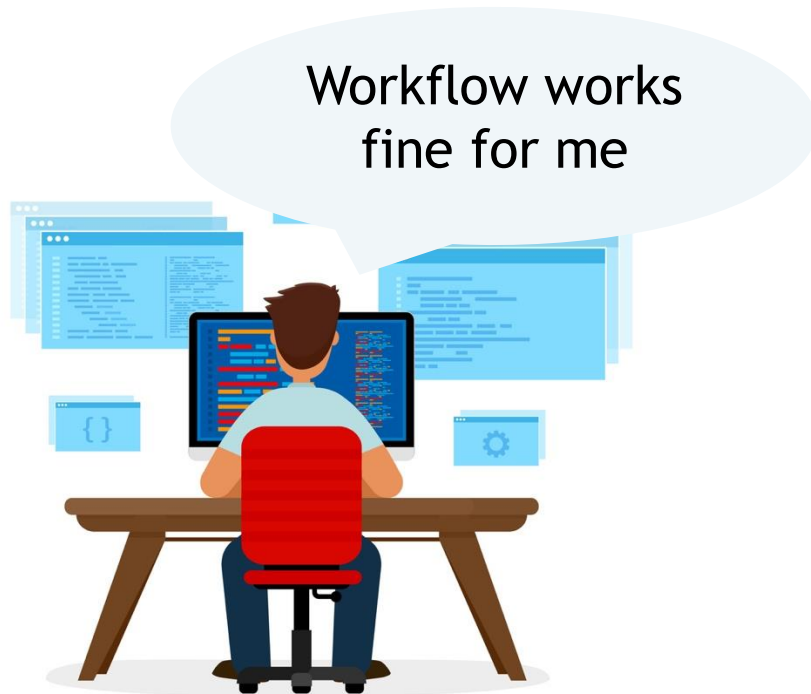
Package
managers

Containers &
isolation

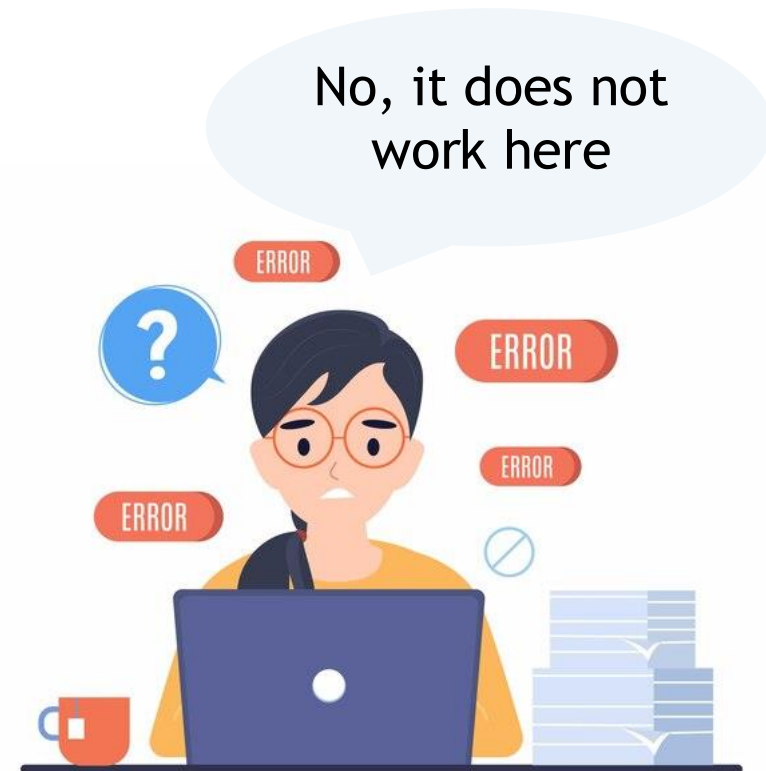


Why containers in workflows?

Developer



Tester/User



What was the context or situation in which you used a container?



Isolation & containerization, what does it mean?

Software containerization provides a **system for running software in isolated compartments** that minimize the interaction with the host machine.

It operates **independently** from the host machine ensuring that software runs **consistently across different environments (OS and architecture)**.



What is a container?

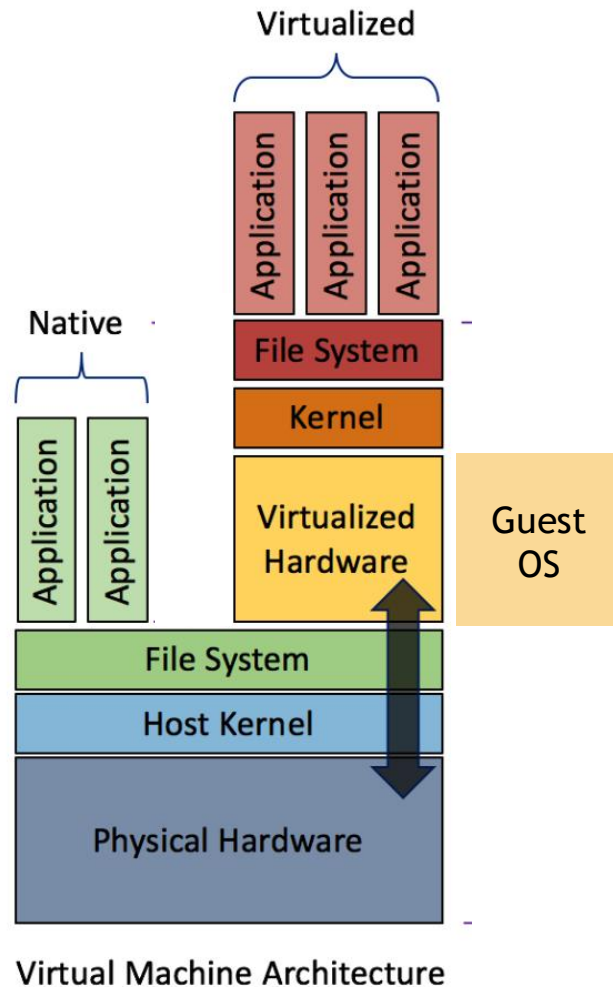
It is a lightweight, self-sufficient package containing everything needed to run software (including application code, system libraries, system tools).

Why are containers useful?

- **Portability:** run in many different environments with different operating systems and hardware platforms
- **Consistency:** you can be sure that they operate the same, regardless of where they are deployed
- **Speed to deploy:** no virtualization of hardware



Virtual machines vs. containers



VMs are created by virtualising the host machine's underlying hardware (processing, memory and disk)

- substantial computational overhead.

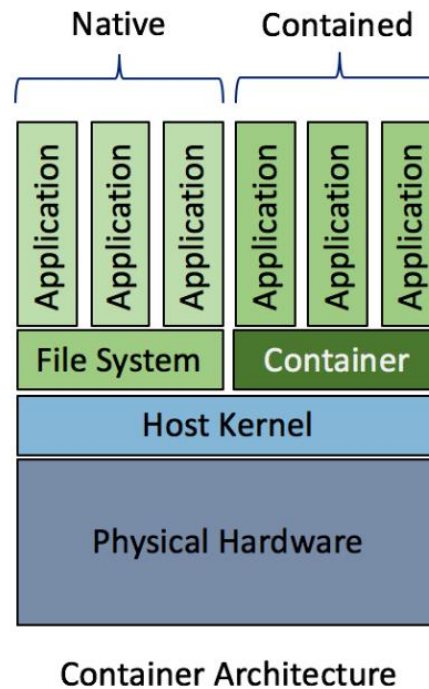
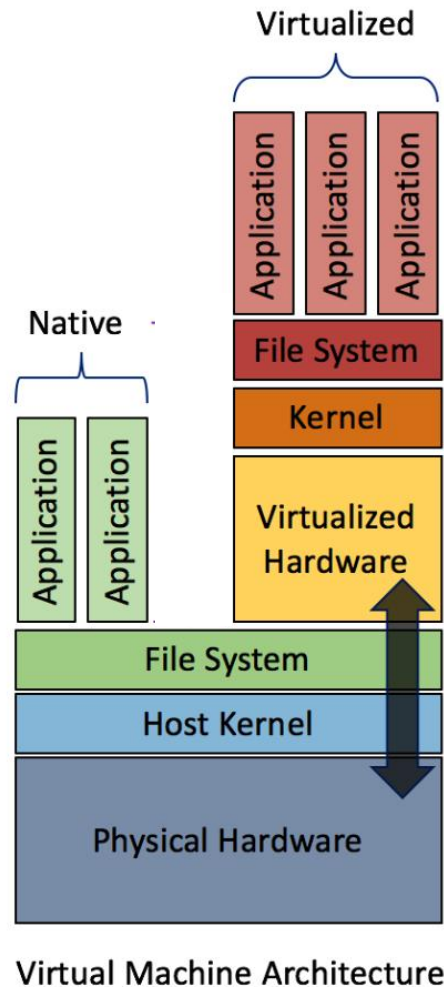
Accessible to less technical users via '**hypervisor**' software which virtualises the host's hardware and acts as the broker:

- managing the lifecycle of the virtualised hardware
- feeding resources to the VMs.

Example hypervisor: VirtualBox, Vmware Workstation Player and Parallels Desktop.



Virtual machines vs. containers



Containers don't need to have their own OS, making them much more **lightweight** than VMs

Multiple containers run on the host OS while sharing the **same kernel**

Example Docker daemon: manage the lifecycle of the container



Docker containers



Docker is an **open-source containerization platform** that enables users to **build, deploy and manage** containerised applications.

It “**packages**” an **entire software environment**, including dependencies and configuration settings, into a portable container that can **run on any system** with Docker installed.

It ensures that a software environment remains **consistent** across different machines, facilitating the **reproduction of computational analyses and experiments**.



Docker elements



Image

App blueprints that define the basis and configuration for containers

Container

Instance of a Docker image and what runs the actual app

Docker daemon

Background service running on the host that manages images (building, running and distributing containers)

Docker client

Command-line tool

Docker Hub

Registry



Running an official Docker image example: hello-world

```
(base) gsd818@SUN1016678 ~ % docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
```

```
latest: Pulling from library/hello-world
```

```
c1ec31eb5944: Pull complete
```

```
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
```

```
Status: Downloaded newer image for hello-world:latest
```

Name:tag (latest <- default)

Repository: Docker Hub

Download image

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

Run instance of the image (“container”)

```
To generate this message, Docker took the following steps:
```

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

```
To try something more ambitious, you can run an Ubuntu container with:
```

```
$ docker run -it ubuntu bash
```

```
Share images, automate workflows, and more with a free Docker ID:
```

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



Build a Docker image

DIR (project directory)

Dockerfile ← plain-text “build recipe” file

./omics-pipeline

./omics-pipeline/data_init.sh

./omics-pipeline/data_anal.sh

The image is **immutable** and needed for the container to run - to change it you need to modify your Dockerfile and **build a new version** of your image for the new container to run.



Dockerfile

```
FROM ubuntu:20.04

RUN apt-get update &&
apt-get install wget

wget path/to/data

COPY ./omics-pipeline

RUN ./data_init.sh

CMD [“./data_anal.sh”]
```

If tag not specified, Docker will use latest - version change over time

Ubuntu image, 20.04 tag

- Runs command on the image ubuntu

Download data to image

Copy the pipeline dir into the image

Run script in our image

Default command in our docker image - when this container runs, it'll execute it



How is it all connected?

dockerfile

```
Dockerfile x
FROM microsoft/dotnet:sdk AS build-env
WORKDIR /Docker

# Copy csproj and restore as distinct layers
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Instructions for
building an image

> docker build

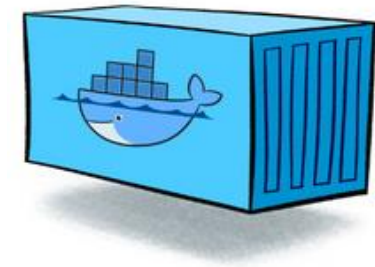
Docker image



Docker will run the
image and spin up a
new container in our
system

> docker run

Docker container

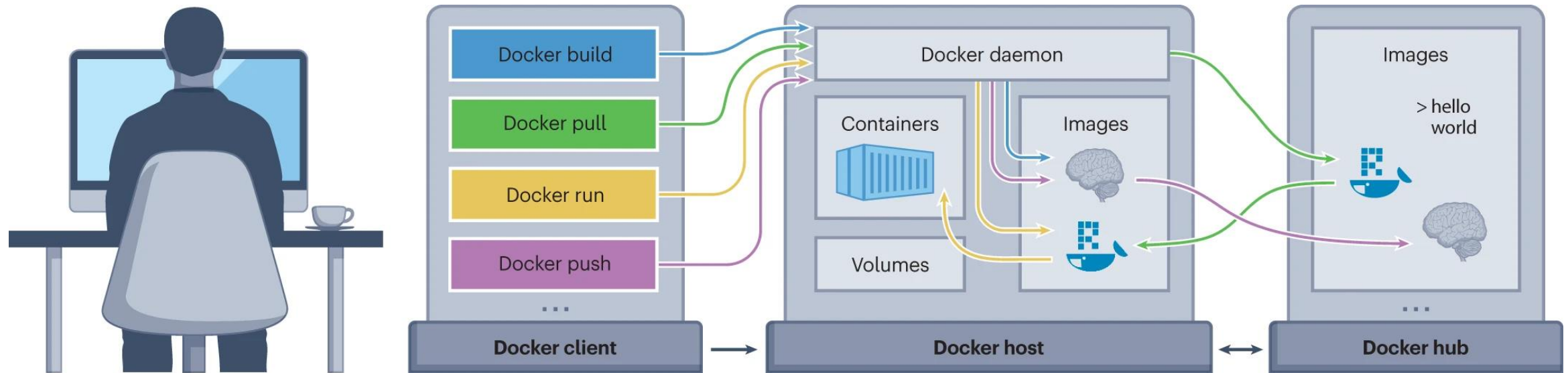


Use the appropriate
command-line options to
execute the commands for
running our processes

docker run my-image python app.py



Docker architecture



OS agnostic



Let's build a simple container from scratch



Create a directory with a Dockerfile & bash script designed to print text banners

`docker-example/`

`Dockerfile`
`print-message.sh`



Let's build a simple container



Create a directory with a Dockerfile & bash script designed to print text banners

Dockerfile

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y figlet
COPY print-message.sh /print-message.sh
RUN chmod +x /print-message.sh

CMD ["/print-message.sh"]
```

print-message.sh

```
#!/bin/bash
# Define phrases
PHRASES=("Welcome to HPC-Pipes", "This container is cool")

# Randomly select 1 phrase
RANDOM_IDX=$(( RANDOM % ${#PHRASES[@]} ))
SELECTED_PHRASE=${PHRASES[$RANDOM_IDX]}

# Print message
figlet -w 100 "$SELECTED_PHRASE"
```

Program that
prints a text
banner

Instructions to create the
containerised environment to
print banners (using figlet)





`docker build -t test-message:v01 .`

check the output
to understand
what docker does

```
(base) gsd818@SUN1016678 docker-example % docker build -t "test-message" .
[+] Building 1.2s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 241B                             0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04  0.9s
=> [auth] library/ubuntu:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                   0.0s
=> [1/4] FROM docker.io/library/ubuntu:20.04@sha256:6d8d9799fe6ab3221965efac00b4c34a2bcc102c086a58dff9e19a08b913c7ef 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 329B                                 0.0s
=> CACHED [2/4] RUN apt-get update && apt-get install -y figlet 0.0s
=> [3/4] COPY print-message.sh /print-message.sh              0.0s
=> [4/4] RUN chmod +x /print-message.sh                       0.2s
=> exporting to image                                           0.0s
=> => exporting layers                                          0.0s
=> => writing image sha256:e7a0b80f31f5978d858786d8ae8b7fe5dafdf982fc04bae0a0baa39adac9460b 0.0s
=> => naming to docker.io/library/test-message                 0.0s
```



```
(base) gsd818@SUN1016678 docker-example % docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test-message	latest	e7a0b80f31f5	2 minutes ago	128MB
my-test-image	latest	f9af426f16ae	5 weeks ago	262MB
docker/welcome-to-docker	latest	c1f619b6477e	11 months ago	18.6MB
dreg.cloud.sdu.dk/ucloud-apps/ubuntu	Oct2023-xfce	8497e2544612	12 months ago	5GB

```
docker run test-message
```

```
(base) gsd818@SUN1016678 docker-example % docker run test-message
```

A 5x10 grid of 50 small, stylized, abstract shapes. Each shape is composed of black lines forming various geometric patterns, including lines, curves, and enclosed spaces. The shapes are arranged in five rows and ten columns, with each row containing ten unique patterns.

[illegible]

Containers

Images

Volumes

Builds

Docker Scout

Extensions

Images

LocalHub

390.25 MB / 16.93 GB in use8 images

Last refresh: 6 hours ago

Search

Name	Tag	Status	Created	Size	Actions
770df7d7d795	2023.03.01	Unused	1 year ago	9.66 GB	
debian	stable-20230320	Unused	2 years ago	124.11 MB	
hello-world	latest	In use	1 year ago	13.25 KB	
test-message	latest	In use	7 minutes ago	128.41 MB	

Showing 8 items

Terminal

(base) gsd818@SUN1016678 ~ % docker run test-message

THE DOCKER WAY

(base) gsd818@SUN1016678 ~ % docker run test-message

WELCOME TO

(base) gsd818@SUN1016678 ~ %

Engine running

RAM 0.77 GB CPU 0.08% Disk 38.88 GB avail. of 62.67 GB

BETA Terminal v4.34.2

Docker desktop



Repositories for distributing packages



cloud-based repository for
storing and sharing Docker
images



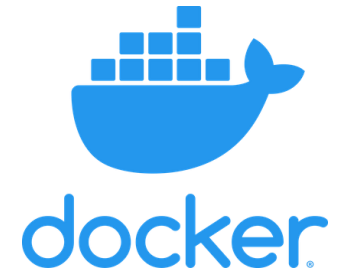
bioinformatics Docker images

Other repositories

- depot.galaxyproject.org
- [Sylabs](https://sylabs.io)



Running blast (Basic Local Alignment Search tool) compares biological sequence information



```
$ docker pull biocontainers/blast:2.2.31
```

```
$ docker run biocontainers/blast:2.2.31 blastp -help
```

```
$ docker run biocontainers/blast:2.2.31 curl  
https://www.uniprot.org/uniprot/P04156.fasta >> P04156.fasta
```

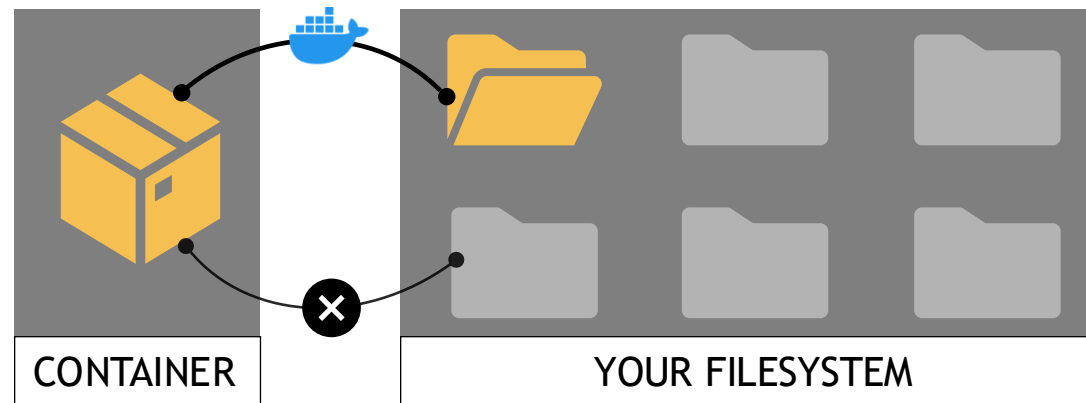
Multiple **command line options**!

Containers typically come with various software packages commonly available on Linux systems.



Shared volumes

- Location in your local filesystem managed by Docker
- Key to **preserve** data generated using a container (the content in a container is deleted with the container itself) and used by the container



Let's mount the data so we can save the results on our computer



```
$ docker run -v `pwd`/host-data/:/data/  
biocontainers/blast:2.2.31 curl -O  
ftp://ftp.ncbi.nih.gov/refseq/D_rerio/mRNA_Prot/zebrafish.1  
.protein.faa.gz
```

```
$ docker run -v `pwd`/host-data/:/data/  
biocontainers/blast:2.2.31 gunzip  
zebrafish.1.protein.faa.gz
```

← You will find it in host-data

TIP: Containers typically come with various software packages commonly available on Linux systems.



Advantages

Package all dependencies into a single image (ensuring compatibility with host OS)

Developers can deploy images for different OSs and architectures, facilitating **software distribution**

Containers minimize the virtualization overhead in Linux, enabling multiple containers to run concurrently with low CPU and memory usage

Limitations

Docker containers run on a shared OS with high-level system access

Root privileges

Alternative to deal with administrative privileges?
Singularity/Apptainer



Singularity/Apptainer

- **Popularity:** designed for use in **HPC**
- **Free and open-source** computer program performs operating-system-level virtualization
- Images created with Docker are compatible with Apptainer and *vice versa (repositories)*
 - *Difference: Image.sif*
- Key difference: runs containers as non-root users by default (improved security)

Simple, fast and secure.



Command-line examples

Pull an image from biocontainers

```
apptainer pull docker://biocontainers/blast:2.2.31
```

Pull an image from dockerhub

```
apptainer pull my_container.sif docker://ubuntu:22.04
```

Run a command inside the container

```
apptainer run blast_2.2.31.sif blastp -version
```

Run an interactive shell within a container

```
apptainer shell <my_container.sif>
```

<https://genome.au.dk/docs/installing-software/>





~ 40 min

Now is YOUR time!

Option 1 - Docker:

Use Docker Desktop for this exercise
(you need run the exercise locally)

Run the software 'fastmixture' inside
the docker container, making sure to
mount the toy dataset

Option 2 - Apptainer:

Use Apptainer on genomedk



`hds-sandbox.github.io/HPC-lab`

workshop > HPC Pipes > Day 1 >

Containers: Apptainer (genomedk)

Containers: Docker (local)



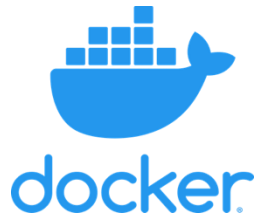
Problems/Issues/Comments

Summary

- `dockerfile`
- `docker build <image>`
- `docker run <image> <somecommand>`
- `docker pull <image>`
- `docker push <image>`
- Shared volumes



When to use a container



[Docker cheat sheet](#)

- When developing software
- Incorporate Docker/Apptainer images into your pipelines when the required software has been containerized. All nf-core pipelines do!
- TIP: If you are using conda and snakemake, containerize your conda-based workflows so that they can run in any OS



Best practices: reproducible environments

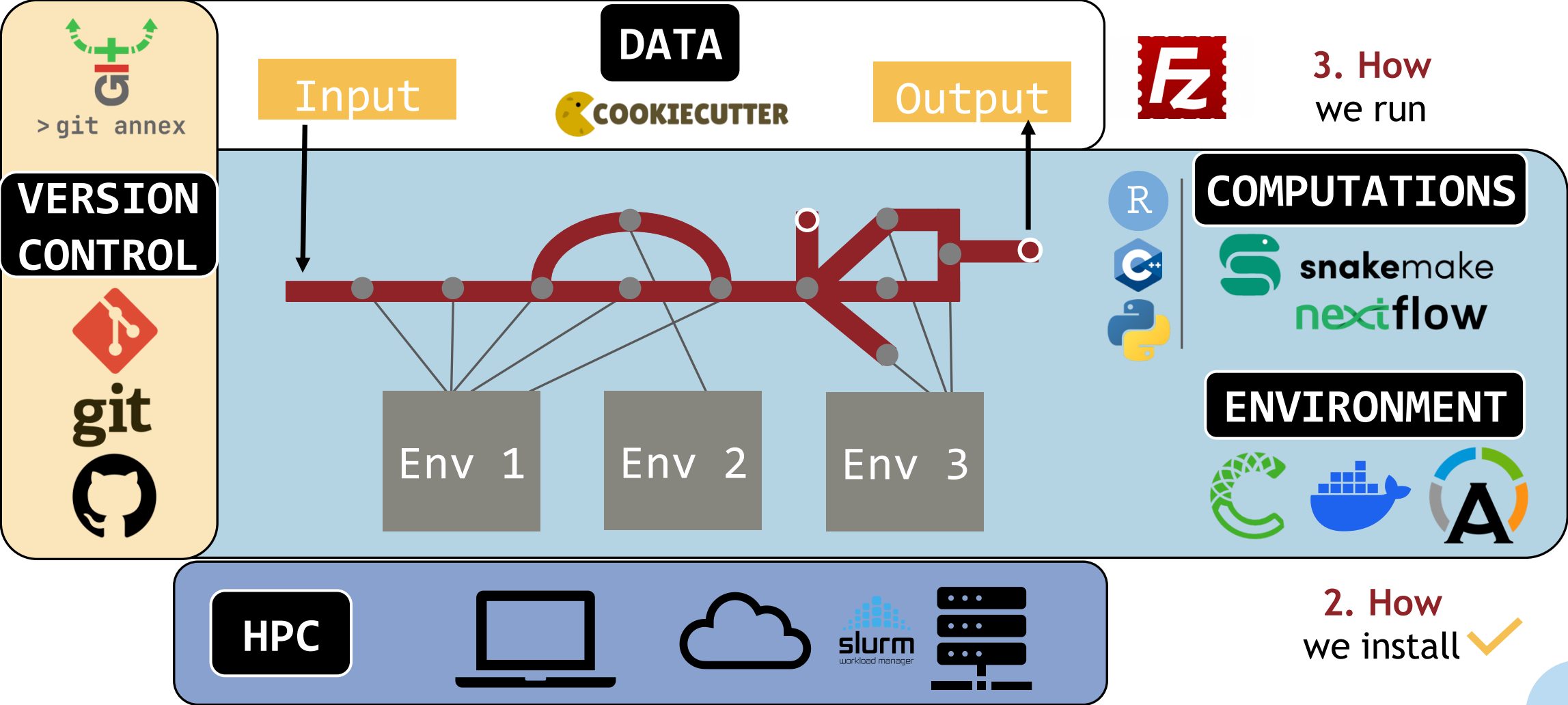
- Choose virtual environments, package managers or containers to manage software, package and dependencies (renv, conda, docker, singularity...)
- One for each project! Isolate the project dependencies
- Document your environment setup
- Version control your environment configuration files
- Test the environment reproducibility
- Share reproducible environments with collaborators

Reproducibility

Interoperability



Components of a bioinformatics pipeline



1. Where we set up ✓

2. How we install ✓

3. How we run

