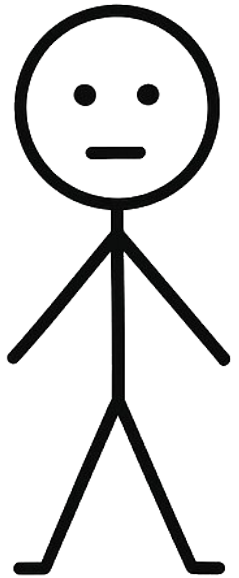
Introduction to containers

Isac Pasianotto

1. **Containers... Why?**
2. **Main concepts**
3. **Podman quick-start**
4. **Practical session**

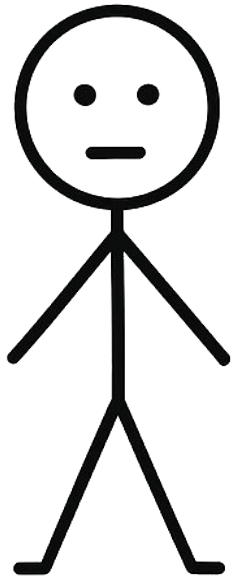
Meet Bill

- This is Bill.
- Bill needs to try a new software



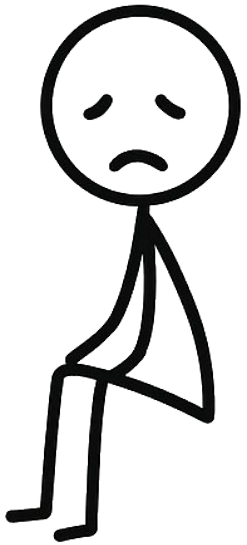
Meet Bill

- This is Bill.
- Bill needs to try a new software
- Unfortunately it does not exists any precompiled binary for his OS
- Bill ask google for help but he gets only useless advise like “compile it by yourself!”
- Bill start to download libraries and for the development, some of those require a upgrade/downgrade of its OS.



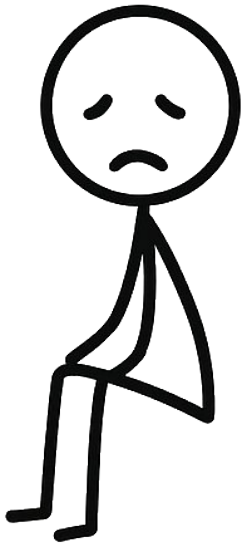
Meet Bill

- This is Bill.
- Bill needs to try a new software
- Unfortunately it does not exists any precompiled binary for his OS
- Bill ask google for help but he gets only useless advise like “compile it by yourself!”
- Bill start to download libraries and for the development, some of those require a upgrade/downgrade of its OS.
- ***Something goes terribly wrong***



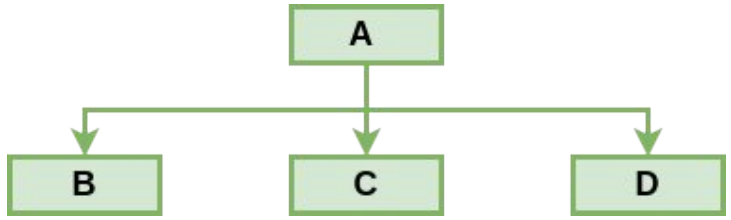
Meet Bill

- This is Bill.
- Bill needs to try a new software
- Unfortunately it does not exists any precompiled binary for his OS
- Bill ask google for help but he gets only useless advise like “compile it by yourself!”
- Bill start to download libraries and for the development, some of those require a upgrade/downgrade of its OS.
- ***Something goes terribly wrong***
- Bill spends the entire day only to fix his own OS, and the following day to compile that software.
- Eventually Bill discovers that that software is not suitable for his use case.



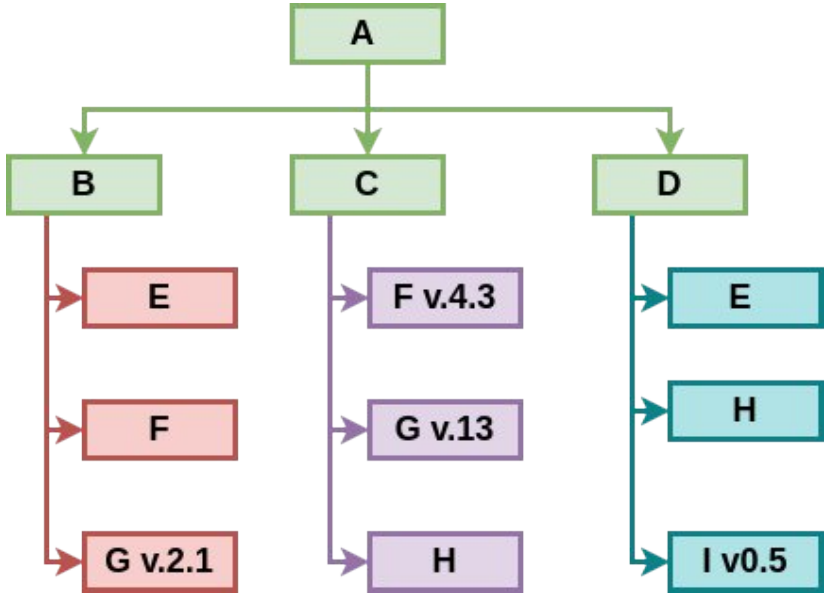
The dependency hell

1. An application, to work properly requires some libraries to be installed.



The dependency hell

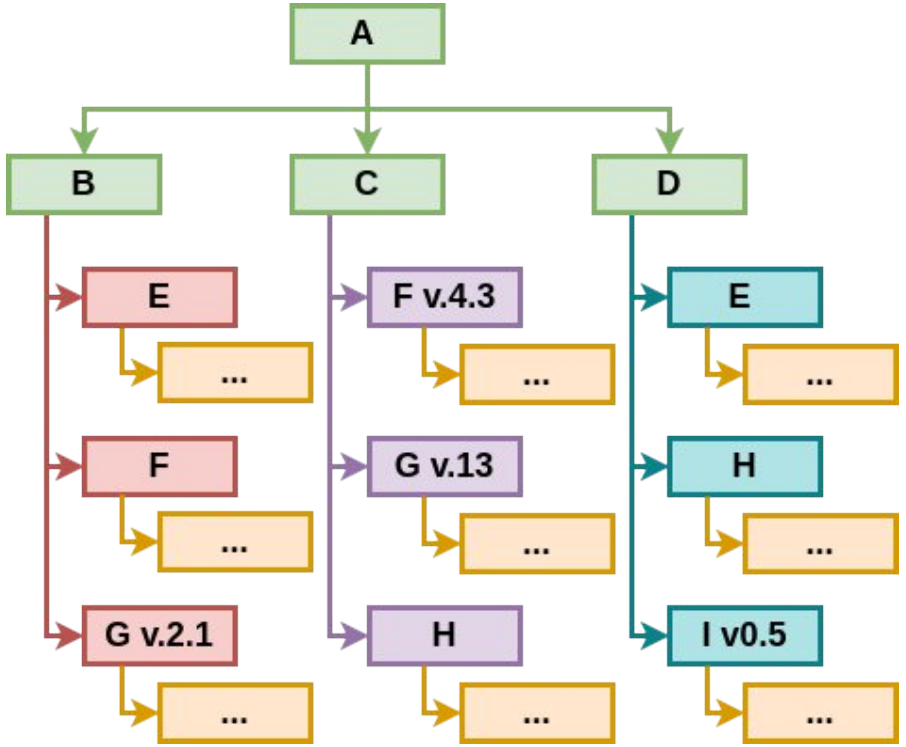
1. An application, to work properly requires some libraries to be installed.
2. Each of these libraries certainly depends on other libraries and other software that has be installed.



The dependency hell

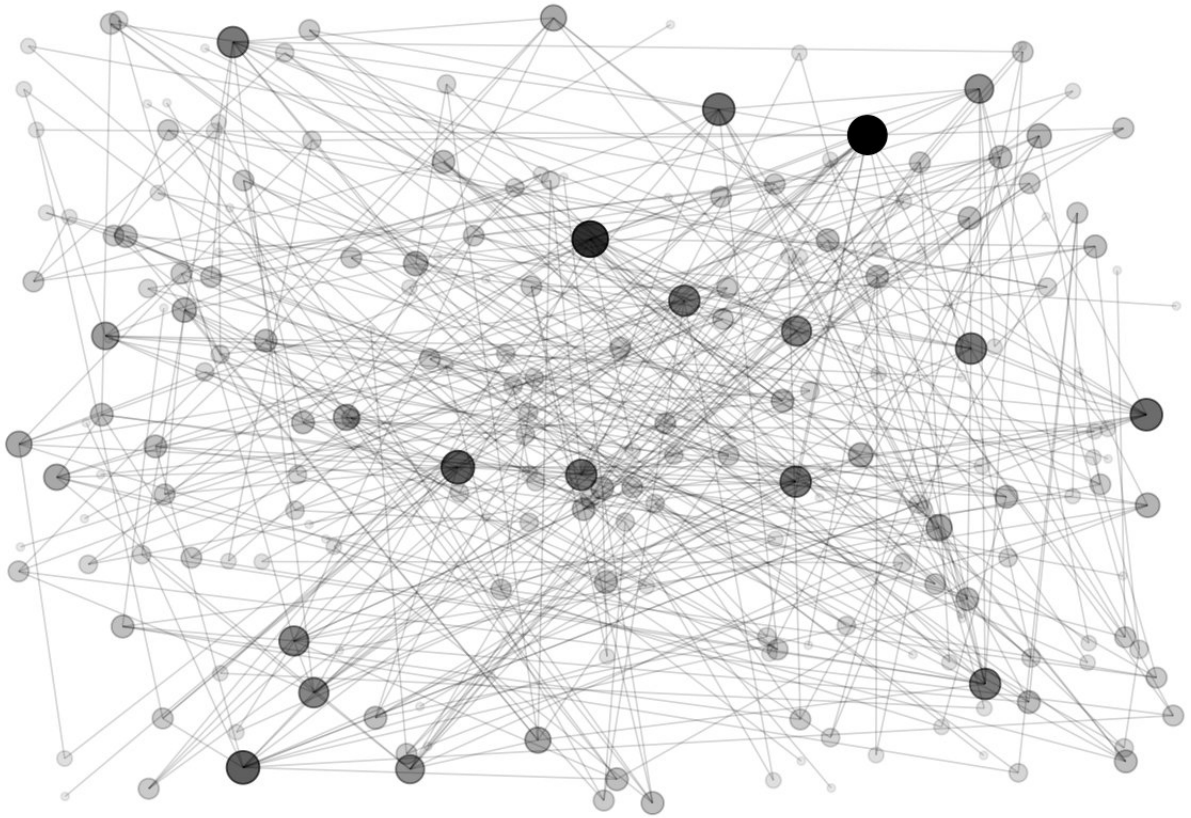
1. An application, to work properly requires some libraries to be installed.
2. Each of these libraries certainly depends on other libraries and other software that has be installed.
3. Read again point 2.
4. Read again point 3.
5. . . .

→ ***What could possibly go wrong?***

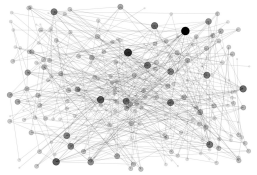


Short answer:

Everything!



Short answer:

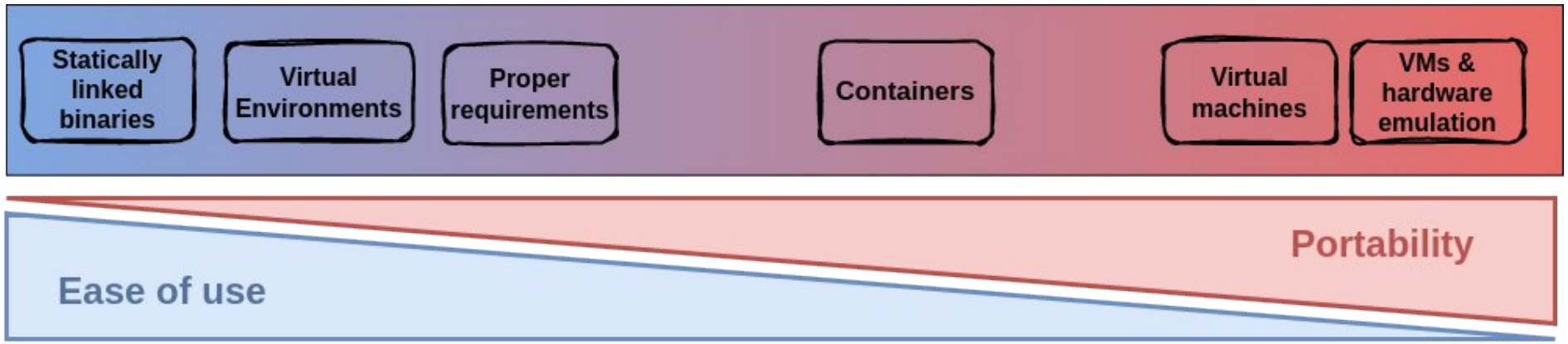


Everything!

Typical issues:

- Different modules require the same dependency, but at different versions.
  - Various versions of the same software may be incompatible.
- Circular dependency:
  - A depends on B, B depends on C, C depends on A
- What about software update?

Dependency hell: The solutions spectrum




Proper requirements
Virtual Environments
Statically linked binaries
Containers
Virtual machines
VMs & hardware emulation

Proper requirements

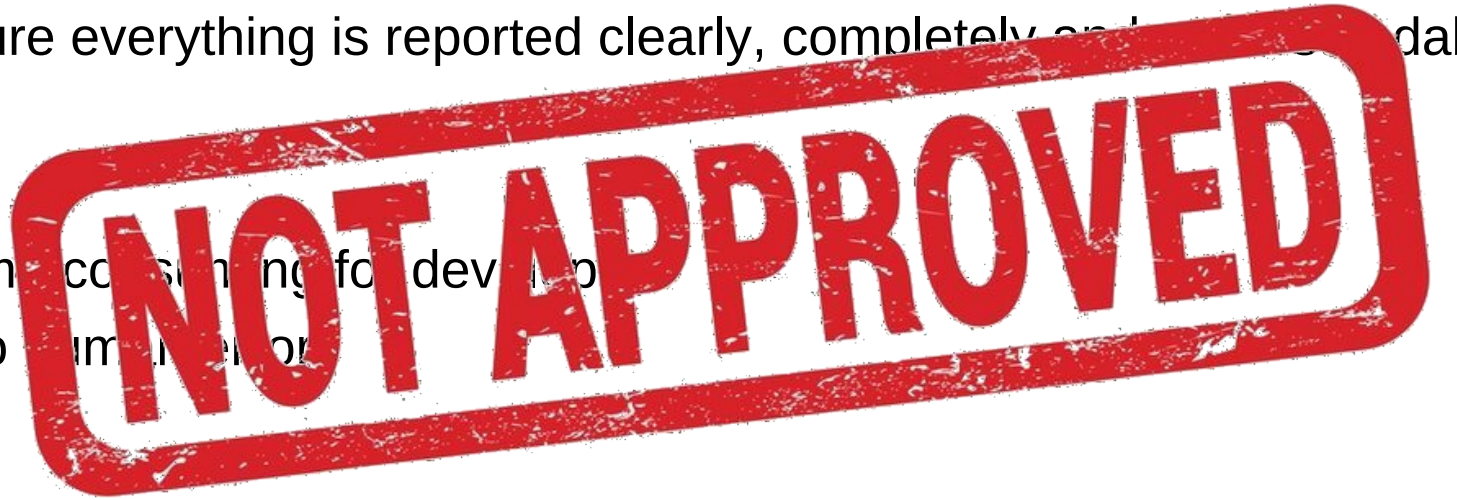
- Write down meticulously *every* single library, OS-feature used during the development.
- Report everything on the documentation, for each release.
- Make sure everything is reported clearly, completely and understandably.

Cons:

- Very time consuming for developer
- Prone to human error

Proper requirements

- Write down meticulously *every* single library, OS-feature used during the development.
- Report everything on the documentation, <u>for each release</u>.
- Make sure everything is reported clearly, completely and [illegible]dably.



Cons:

- Very time consuming for development
- Prone to human error

Virtual environments

- Work within a reproducible environment where libraries remain consistent for both developers and end users.
- Each release provides a definition of its virtual environment.

e.g., venv, conda, pixi, ...

Cons:

- Requires the user to set up and activate its own environment
- Not a comprehensive solution
- prone to human error

Virtual environments

- Work within a reproducible environment where libraries remain consistent for both developers and end users.
- Each release provides a definition of the virtual environment.

e.g., venv, conda, pixi, ...

Cons:

- Requires the user to set up and activate its own environment
- Not a comprehensive solution
- prone to human error

Statically linked binaries

- Distribute statically linked binaries that bundle the application with all required dependencies.
- Ensures the program runs identically across systems without relying on external libraries.

Cons:

- ➢ Works only for compiled languages

Statically linked binaries

- Distribute statically linked binaries that bundle the application with all required dependencies.
- Ensures the program runs identically across systems without dependency on external libraries.

Cons:

- Works only for compiled languages


Proper requirements
Virtual Environments
Statically linked binaries
Containers
Virtual machines
VMs & hardware emulation

VMs with hardware emulation

Let’s be honest... It is an absolute overkill!

Cons:

- High resource usage
- Poor performance and long startup times
- More complex life-cycle management

VMs with hardware emulation

Let’s be honest... It is an absolute overkill!

Cons:

- High resource usage
- Poor performance and long startup times
- More complex life-cycle management

Virtual Machines (VMs)

- Each VM includes its own OS, libraries, and dependencies.

Pros:

✓ Do not mess up the host OS: strong isolation.

✓ Plug and play: easy to setup and allow to quickly test.

Cons:

➢ Overhead: higher resource usage than native execution.

➢ Requires to pre-allocate a certain amount of resources (cpus and ram).

➢ You will not find much software packaged in this way

➢ No source code: usually you need to download a (huge) pre-built trusted images

Virtual Machines (VMs)

- Each VM includes its own OS, libraries, and dependencies.

Pros:

✓ Do not mess up the host OS: strong isolation.

✓ Plug and play: easy to setup and allow to quickly te

Cons:

➢ Overhead: higher resource usage (an entire guest OS)

➢ Requires to pre-allocate a certain amount of resources (cpus and ram)

➢ You will not find most software packaged in this

➢ No source code: need to download a (huge) pre-built trusted images

Cons > Pros

→ ***But we're moving in the right direction...***

Can we have that level of isolation with more reasonable compromises?

Containers

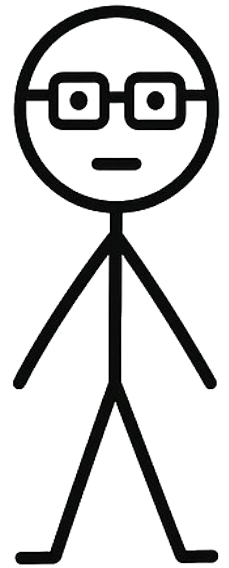
- As first approximation, you can think of a container as a lightweight executable which comes with all the things needed:
  - Code
  - Dependencies and libraries
  - Other stuff needed to setup the environment
- Because everything travels together, containers make it simple to move software between machines, without the overhead of virtual machines.

Containers

- As first approximation, you can think of a container as a lightweight executable which comes with all the things needed:
  - Code
  - Dependencies and libraries
  - Other stuff needed to setup the environment
- Because everything travels together, containers make it simple to move software between machines, without the overhead of virtual machines.

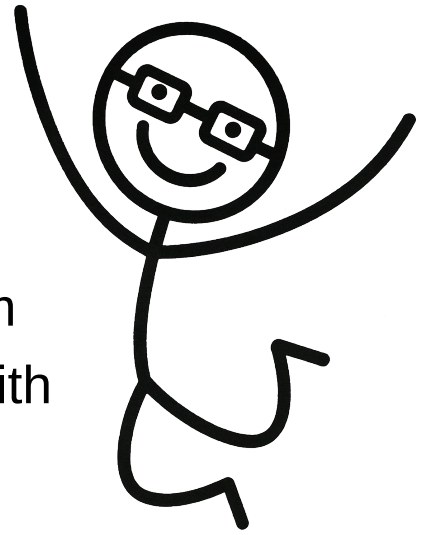
Meet Bob

- This is Bob.
- Bob also needs to try a new software, again no precompiled binary is available.
- Bob search if a container exists for that software, and the answer is yes.
- Bob pull that container and runs it in seconds.



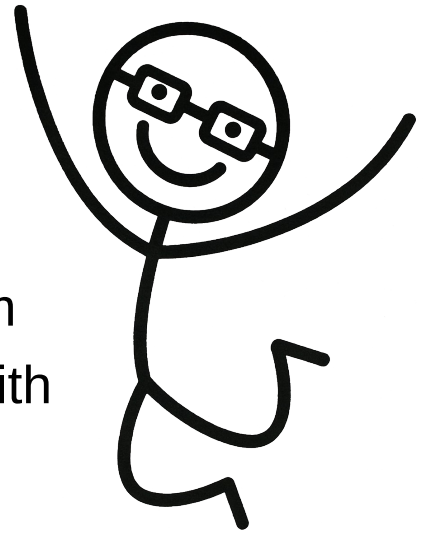
Meet Bob

- This is Bob.
- Bob also needs to try a new software, again no precompiled binary is available.
- Bob search if a container exists for that software, and the answer is yes.
- Bob pull that container and runs it in seconds.
- ***Bob immediately realize that he needed something different,***
  after that he find another software, always containerized, which is perfect for him
- Bob spends the rest of the day working on his much more important business with the right software, even finding time for a walk.



Meet Bob

- This is Bob.
- Bob also needs to try a new software, again no precompiled binary is available.
- Bob search if a container exists for that software, and the answer is yes.
- Bob pull that container and runs it in seconds.
- ***Bob immediately realize that he needed something different,***
  after that he find another software, always containerized, which is perfect for him
- Bob spends the rest of the day working on his much more important business with the right software, even finding time for a walk.
- **Please, be like bob!**



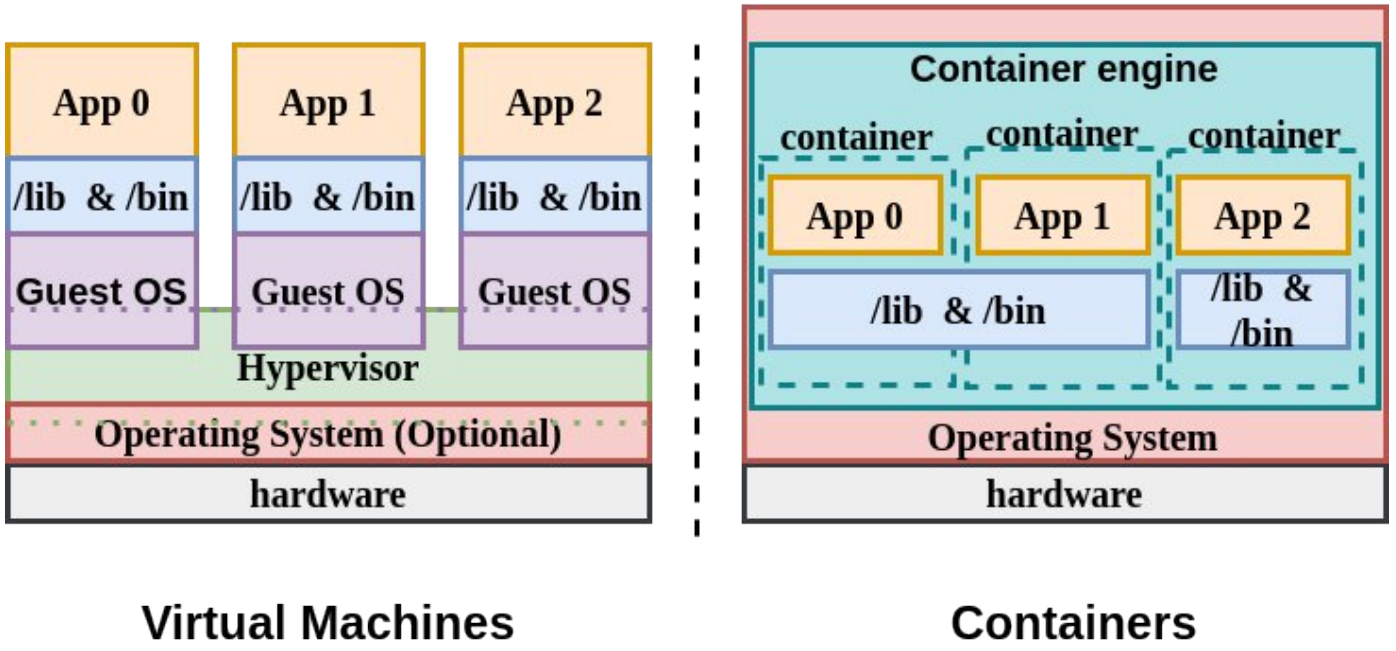
1. Containers... Why?
2. **Main concepts**
3. Podman quick-start
4. Practical session

Containers, why we like it?

- ***Simplify the deployment****:* sandboxed view of the OS logically isolated from other applications.
- ***Portability:*** Registries makes simple to share and retrieve container images, usually also for different architectures.
- ***Versioning:*** containers images comes with a tag to distinguish various versions, similar to what happens with git.
- ***Popularity:*** Since containers are (one of the most) popular solution, it is easier to find a container for the app/software you are looking for.

Containers, why we like it?

Like VMs but better!



Containers lexicon... some clarity

In practice, the terms *“container”* is overloaded...

Definition:

“ A container is a standard Linux process typically created through a clone() system call instead of fork() or exec().

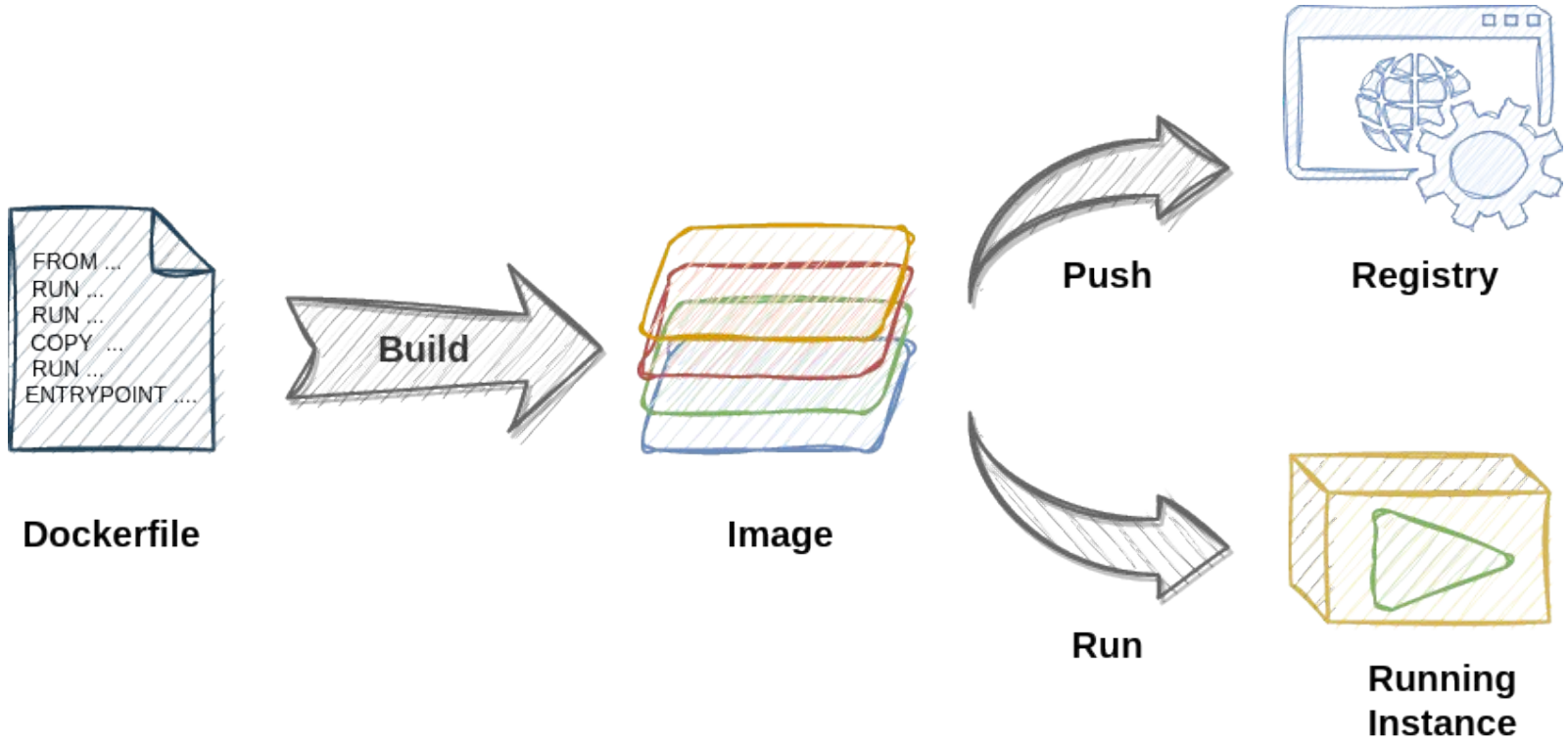
Also, containers are often isolated further through the use of cgroups, SELinux or AppArmor. ”

Containers lexicon... some clarity (cont’d)

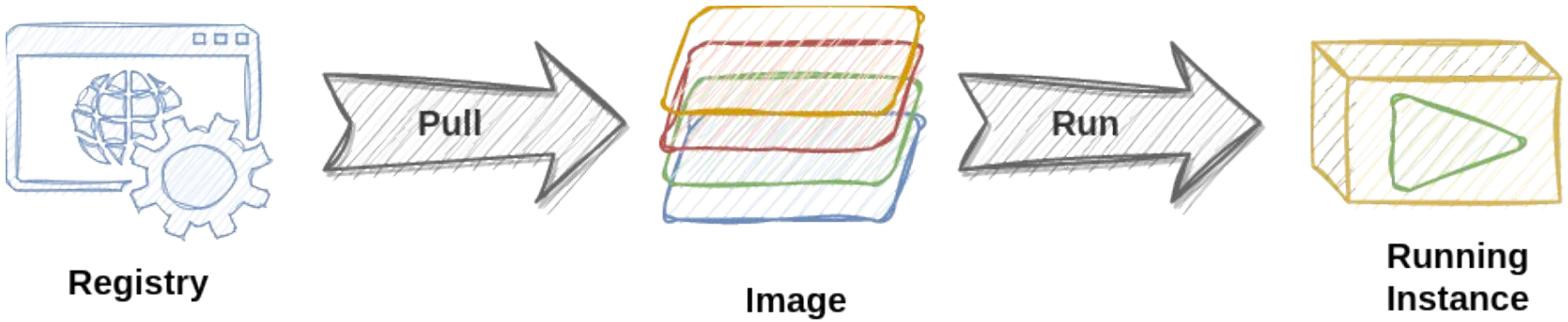
In practice:

- ***Container:*** Running instance of a *container image*
- ***Container image:*** shareable lightweight, stand-alone, executable package that includes everything needed to run a piece of software (code, libraries, ...).
- ***Dockerfile***: text-based build file that defines how to assemble a container image
- ***Container engine***: software that builds, runs, and manages containers from images (e.g., Docker, Podman, Apptainer).

Typical workflow



Typical workflow, option b.



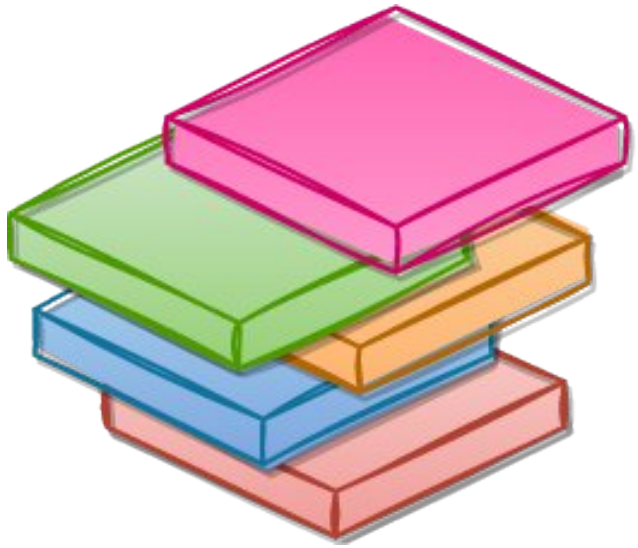
Dockerfile

```
FROM fedora:42

RUN dnf update && \
    dnf install <very_important_package>

ENV SETTING="devel"

COPY my_project_dir /app/

ENTRYPOINT ["/bin/bash"]
```

- Like a “recipe” which describe all the necessary to obtain a container image with a proper setup and environment
- (Almost) always, the starting point is *another container image*
- Each instruction (RUN, COPY, CMD, etc.) creates a new layer, making images modular and cache-friendly.
- Ensure *reproducibility:* the same build always produces the same environment.

Container image

- Images are based on layers
- Each layer depends on the previous one
- *Layer caching:* unchanged layers are reused
  → faster rebuilds.
- *Immutability*: each layer is read-only; the final container adds a writable layer on top.

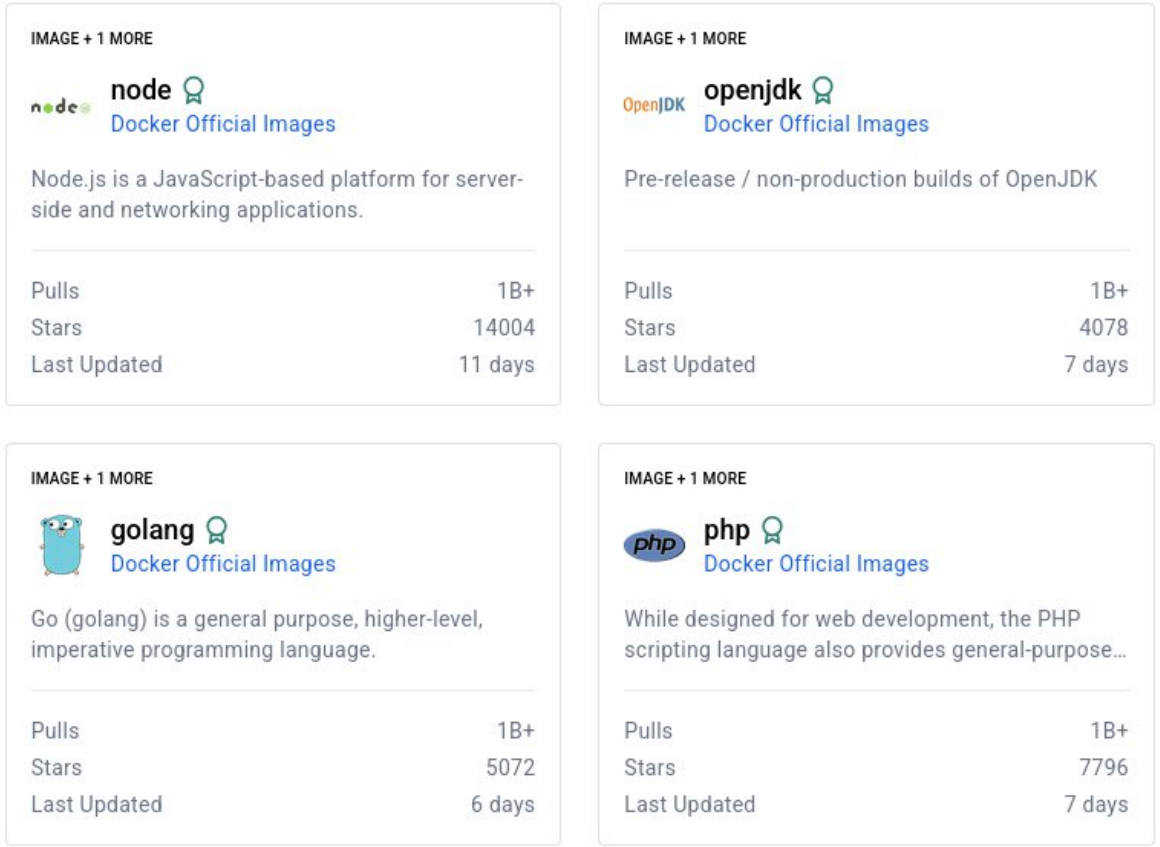


Container Registry

- A container registry is a centralized service to store and distribute container images.
- Images can be pushed (uploaded) and pulled (downloaded) by developers.
- Registries can be public or private (for internal use in organizations).
- They enable versioning!

A very famous and popular one:

https://hub.docker.com



1. Containers... Why?
2. Main concepts
3. **Podman quick-start**
4. Practical session

Container engines

(Too) many alternatives are available...

podman

Podman: one C.E. to rule them all

- Very mature and well-documented solution.
- ***Rootless mode:*** podman does not require sudo privileges for (some very particular exception exists).
- ***Docker-compatible CLI:*** Acts as a drop-in replacement for docker, another extremely popular solution. You can substitute `docker run ...` with `podman run ...` in almost any situation.
- ***Kubernetes friendly & pod support:*** Better integrates some advance features, making the transition to kuberentes smoother (advanced topics not covered in this course!)

Installing podman

```
sudo apt-get update

sudo apt-get -y install podman


# Check that everything is ok

podman run hello-world:latest
```

```
pasianeight@pavilion:~$ podman run hello-world:latest
!... Hello Podman World ...!

         .--"--.
       / -     - \
      / (O)   (O) \
   ~~~| -=(,Y,)=- |
    .---. /`  \   |~~
 ~/  o  o \~~~~.----. ~~
  | =(X)= |~  / (O (O) \
   ~~~~~~~  ~| =(Y_)=-  |
  ~~~~    ~~~|   U      |~~

Project:   https://github.com/containers/podman
Website:   https://podman.io
Desktop:   https://podman-desktop.io
Documents: https://docs.podman.io
YouTube:   https://youtube.com/@Podman
X/Twitter: @Podman_io
Mastodon:  @Podman_io@fosstodon.org
```

```
# on ubuntu, add to  /etc/containers/registries.conf


[registries.search]
registries = ['docker.io', 'quay.io', 'registry.fedoraproject.org']
```

Podman cheat-sheet

podman build	Build a container
Podman pull	Pull a container from a registry
podman run	Run a container (execute default command or a custom one)
podman ps	List all running containers
podman exec	Run a command in running containers
podman stop	Stop a running container
podman rm	Remove a (not running) container
podman image ls	Show all the already-pulled images
podman help	Show the help page, with a lot of useful infos!

Dockerfile cheat-sheet

FROM	Defines the base image to build on
RUN	Executes a command at build time (creates a new layer)
COPY	Copies files/directories from host into the image
ENV	Set environment variables inside the image
WORKDIR	Sets the working directory for subsequent instructions
USER	Sets the user under which the container runs (it must exists!)
VOLUME	Declare mount points for external volumes
ENTRYPOINT	Defines the fixed runtime executable (e.g., /bin/bash)
CMD	Sets teh default arguments

Containers are ephemeral!

- It means that when you exit a container, all changes made to the container's filesystem are lost!
- We will see soon why this is not a problem...

1. Containers... Why?
2. Main concepts
3. Podman quick-start
4. **Practical session**