Comment rater la reproductibilité de ses expériences ?

Compas 2023

Quentin GUILLOTEAU , Adrien FAURE , Olivier RICHARD , Millian POQUET*

Univ. Grenoble Alpes, INRIA, CNRS, LIG
firstname.lastname@inria.fr
*IRIT, Université de Toulouse
firstname.lastname@irit.fr

2023-07-06

La reproductibilité en informatique

- des petit exemples marrants :
  - Mytkowicz et al. (2009): l'ordre des options de compilation influe sur les perfs d'une app
  - Stodden et al. (2018): le nombre de variables d'environnement UNIX influe sur les perfs d'une app
- Collberg et al. (2015)
  - étudient 402 papiers de confs + journaux système
  - $\rightsquigarrow$ 46% pas reproductibles
  - les principales raisons:
    - code pas accessible
    - **code ne compile pas**
    - nécessite du hardware spécifique

Et aujourd'hui ?

La réponse de la communauté

ACM

- **Répétabilité** ⇝ Mêmes personnes, mêmes conditions
- **Reproductibilité** ⇝ Personnes différentes, même conditions
- **Réplicable** ⇝ Personnes différentes, autres conditions

Évaluation des artefacts et Badges :

La réponse de la communauté

ACM

- **Répétabilité** ⇝ Mêmes personnes, mêmes conditions
- **Reproductibilité** ⇝ Personnes différentes, même conditions
- **Réplicable** ⇝ Personnes différentes, autres conditions

Évaluation des artefacts et Badges :

Mais...

- Problématiques pas vraiment comprises
- "Reproductibilité" ≠ stocker des images et rejouer un Jupyter !
- Souvent trop rigides et non entendables... Besoin de **variation**.
- Images non reconstructibles...

Comment (bien) partager un environnement logiciel ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` $\rightsquigarrow$ quid des autres dépendances ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` $\rightsquigarrow$ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) $\rightsquigarrow$ oublis ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹
  - Conteneur ⇝ quid de l'OS/Kernel/Drivers ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹
  - Conteneur ⇝ quid de l'OS/Kernel/Drivers ?
  - ⇝ la recette plutôt ! (`Dockerfile`, Kameleon, etc.)

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹
  - Conteneur ⇝ quid de l'OS/Kernel/Drivers ?
  - ⇝ la recette plutôt ! (`Dockerfile`, Kameleon, etc.)
- Spack ? ⇝ mieux, mais pas (totalement) reproductible

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹
  - Conteneur ⇝ quid de l'OS/Kernel/Drivers ?
  - ⇝ la recette plutôt ! (`Dockerfile`, Kameleon, etc.)
- Spack ? ⇝ mieux, mais pas (totalement) reproductible
- Pérennité du partage ?

Comment (bien) partager un environnement logiciel ?

- `requirements.txt` ⇝ quid des autres dépendances ?
- Liste de paquets (`apt-get install ...`) ⇝ oublis ?
- Module ⇝ modification ? pérennité ? partage ?
- Image (conteneur, VM, système)
  - `tgz-g5k`, `cc-snapshot` ☹
  - Format binaire ? ☹
  - Conteneur ⇝ quid de l'OS/Kernel/Drivers ?
  - ⇝ la recette plutôt ! (`Dockerfile`, Kameleon, etc.)
- Spack ? ⇝ mieux, mais pas (totalement) reproductible
- Pérennité du partage ?
- Comment introduire de la variation **précise** ?

Exemple de Dockerfile rencontré

```
FROM ubuntu
RUN apt-get update -y && \
    apt-get install -y \
      build-essential\
      ...\
      simgrid
RUN git clone https://.../chord.git
WORKDIR chord
RUN curl -L https://tinyurl.com/patchchord\
    -o increase-timeout.patch
RUN git apply increase-timeout.patch
RUN cmake .
RUN make
ENTRYPOINT ["./chord"]
```

Saurez-vous trouver toutes les erreurs ? ☺

Problèmes - Image de base

Quelle version ?!

```
FROM ubuntu
```

```
FROM ubuntu:latest
```

Traçabilité ? Quid d'une reconstruction future ?
↪ Dépendance à un état extérieur **incontrôlable** !

Mieux ?

```
FROM ubuntu:23.04
```

Pérennité/Reconstructibilité de l'image de base ?
Tag actualisé par le mainteneur ?
↪ Dépendance à un état extérieur **incontrôlable** !

Problèmes - Version du miroir

Quelle version ?!

```
RUN apt-get update
```

Traçabilité ? Quid d'une reconstruction future ?
↪ Dépendance à un état extérieur **incontrôlable** !

Mieux ?

```
RUN deb https://snapshot.debian.org/... lenny
```

Pérennité ? Introduction de variation ?
↪ Trop rigide ?

Problèmes - Commit utilisé

Quelle version ?!

```
RUN git clone https://.../mon_repo.git
```

Traçabilité ? Quid d'une reconstruction future ?
↪ Dépendance à un état extérieur **incontrôlable** !

Mieux ?

```
RUN git clone https://.../mon_repo.git?ref=...
RUN git clone https://.../mon_repo.git?rev=...
```

Pérennité ?
↪ Software-Heritage

Problèmes - Objet téléchargé

Quelle version ?!

```
RUN curl https://tinyurl.com/ma_config
```

Traçabilité ? Quid d'une reconstruction future ? Pérennité ?
↪ Dépendance à un état extérieur **incontrôlable** !

Mieux ?

```
RUN curl https://tinyurl.com/ma_config
RUN md5sum --check attendu.md5
```

Plus important de ne pas construire une image plutôt que d'en construire une erronée !

Problèmes - En vrac

Jupyter/OrgMode/...

- Attention à l'ordre des cellules
- Attention aux dépendances à votre config ! (*e.g.*, `.emacs`)

Moteur d'expérience / Moteur de workflow

- EnOSlib, Execo, Snakemake, etc.
- Fait aussi partit de l'environnement logiciel !

Triste réalisation... ☹

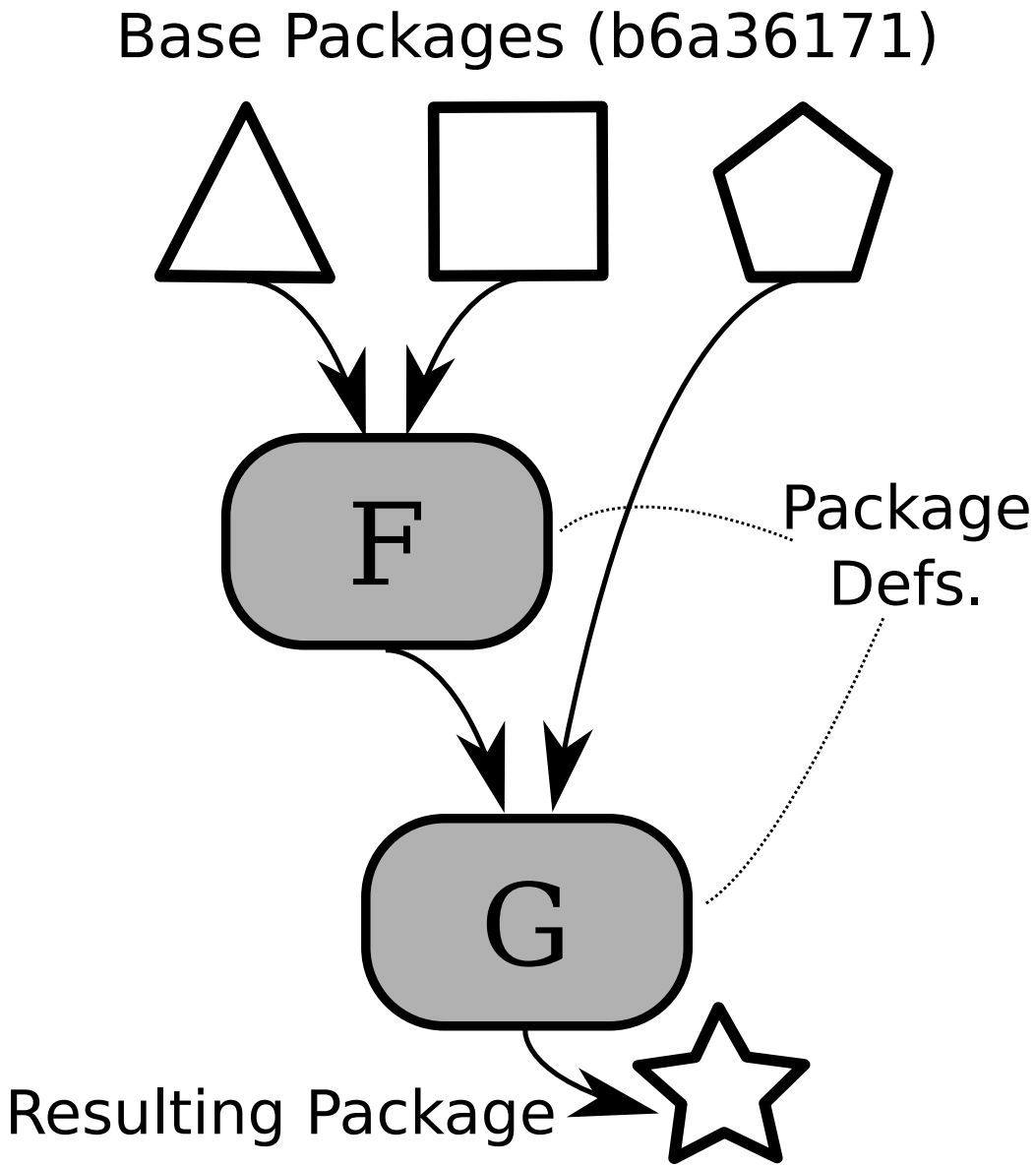
- avec les outils et pratiques usuels :
  - faire des env logiciels reproductibles $\rightsquigarrow$ pas facile
  - introduire de la variation précise $\rightsquigarrow$ pas facile
- les évaluations d'artefacts sont faites juste après la création des envs
  - $\rightsquigarrow$ tout à peu près dans le même état
  - mais dans 1 an ? 5 ans ? 20 ans ?

Nécessité d'avoir une autre approche à la gestion d'env logiciel !

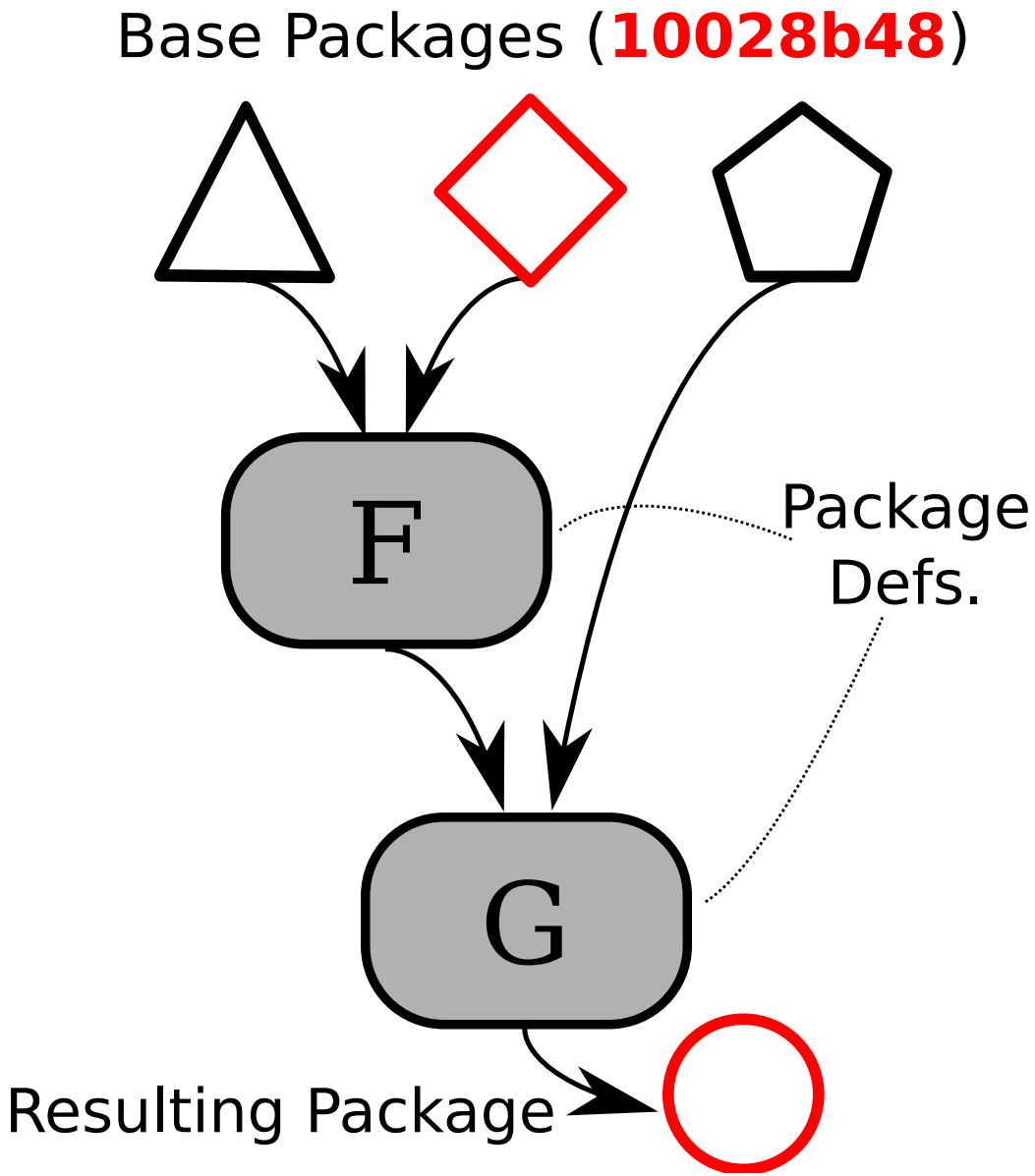
Les gestionnaires de paquets fonctionnels à la rescousse !

- Nix, Guix (cf. mardi après-midi ☺)
- Reproductible par design
- paquets = fonctions
  - entrées = dépendances
  - corps = commandes pour construire le paquet
  - `chord` = f(`simgrid`, `boost`, `cmake`)
  - $f \simeq$ `cmake` + `make` + ...
- pas d'effets de bords, sandbox
- peut construire : conteneurs, VMs, images système
- Pas une solution magique : **toujours possible de se tromper !**

Vous avez dit “Fonctionnel” ?!



Vous avez dit “Fonctionnel” ?!



Exemple de Paquet

```
{ stdenv, fetchgit, simgrid, boost, cmake }:

stdenv.mkDerivation rec {
  pname = "chord";
  version = "0.1.0";

  src = fetchgit {
    url = "https://gitlab.inria.fr/me/chord";
    rev = "069d2a5bfa4c40...";
    sha256 = "sha256-ff4f...";
  };

  buildInputs = [ simgrid boost cmake ];

  # configurePhase = "cmake .";
  # buildPhase = "make";
  # installPhase = "mkdir -p $out/bin && mv chord $out/bin";
}
```

Dependances

Sources

Build Info

Derivation

Introduire de la variation

```
{ pkgs ? import (fetchTarball {
    url = "https://github.com/NixOS/nixpkgs/[...].tar.gz";
    sha256 = "sha256:[...]";}) {}
}:
let
  packages = rec {
    chord = pkgs.callPackage ./chord.nix { };
    chord_custom = chord.override {
      simgrid = simgrid-330;
      boost = boost-167;
    };

    boost-176 = ...;
    boost-167 = ...;
    boost = boost-176;

    simgrid-330 = ...;
    simgrid-331 = ...;
    simgrid = simgrid-331;
  };
in packages
```

Pinning

Pkg Def

Override

nix-build -A chord_custom

Comment stocker les paquets ?

Usual approach: **Merge them all**

- Conflits
- `PATH=/usr/bin`

```
/usr
├── bin
│   └── myprogram
└── lib
    ├── libc.so
    └── libmylib.so
```

Store approach: **Keep them separated**

+ **Variation**
+ Isolés
+ `PATH` précis
+ Read-only

```
/nix/store
├── y9zg6ryffgc5c9y67fcmfdkyyiivjzpj-glibc-2.27
│   └── lib
│       └── libc.so
└── nc5qbagm3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0
    ├── bin
    │   └── myprogram
    └── lib
        └── libmylib.so
```

Critique

Avantages

+ Pas possible d'oublier des deps.
+ Traçabilité (pinned Nixpkgs)
+ `nix-shell`/`guix shell` = multi-langage `virtualenv`
+ Générer des images (docker, VM, système) minimales $\rightsquigarrow$ trivial

Inconvénients

- Contaminant: toutes les deps en Nix/Guix
- Prise en main + changement de pratique
- Quelques comportements implicites
- Stockage externe (github, gitlab,...)

Conclusion

- Du mouvement sur les questions de reproductibilité... ☺
- ... mais problématiques pas tout à fait comprises ☹
- Gestion de l'env logiciel $\rightsquigarrow$ pas facile
- Très très compliqué d'avoir un env repro avec les solutions usuelles
- Nix/Guix:
  - beaucoup mieux...
  - ...mais toujours possible de se tromper
  - **demande un changement de vision/méthodes** (comme `git`)
  - (pas qqchose à faire juste pour l'eval des artefacts)

Papier ici: `https://hal.science/hal-04132438`