Reproductibilité: apport des gestionnaires de paquet fonctionnels

Olivier Richard [1]

[25 Nov 2021]

1/ DATAMOVE Team, LIG, INRIA, Univ. Grenoble Alpes

Agenda

- Reproductibilité: retour d'expérience Grid'5000
- Présentation succinte des principes des gestionnaires de paquet fonctionnels (FPM)
- Guix et Nix
- Exemples
- Quelques succés
- des difficultés
- Discussions
- Conclusion

Reproductibilité (éléments de définition)

Réproductibilité

- La capacité de reproduire une expérience dans des conditions *similaires* et d'aboutir aux mêmes observations (même statistique).

Répétabilité

- Mêmes conditions d'expérience mêmes observations.

Réproductibilité bit-à-bit

- La recompilation d'un code produit un binaire identique bit-à-bit [1].

[1] Reproducible Builds

Retour d'expérience: Grid'5000 (les débuts)

En 2003-

- Réaliser des expériences *propres* à grande échelle

Aventure collective

- Mutualisation de moyens matériels et humains

Permettre toutes expériences et leur répétabiliité

- Garantir l'accès bas niveaux aux noeuds: reset distant, console série, boot sur réseau (*netboot*)
- Outil de déploiement
- Annuaire des machines (conservation des caractéristiques)
- Partage des ressources

Pré-occupation *de l'époque*

- Isolation expé = résa. exclusive, Complexité = nombre de noeuds

Générer / maitriser des images systèmes (*software appliance*)

Générer, conserver, reconstruire, étudier les images systèmes, piles logicielles, *software appliance*...

Plusieurs approches

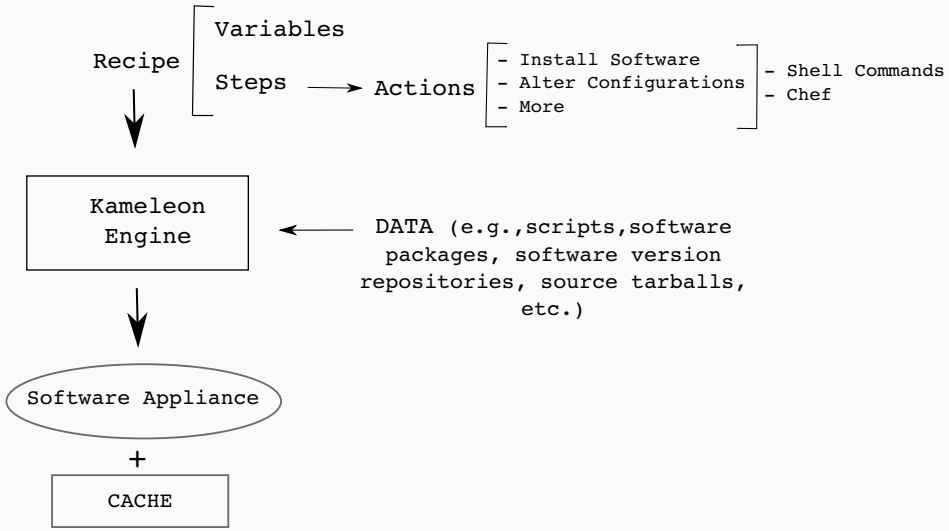
- CDE: capturer l'environnement à travers le suivi des appels systèmes.
- Machine virtuelle
- Outils de gestion de configuration et de déploiement (CFEngine / Ansible / Chef / Puppet / Salt)
- Container (*à la* Docker)

Limitations

- Capture partielle, perte des sources, difficulté de maintenance, dégradation bas bruit

Kameleon:

Un générateur de *software appliance* (pile logicielle)



Permettre la (re)construction de toutes images

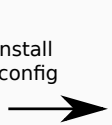
- Un séquenceur de commandes shell, des recettes, un cache, une archive
- Notion de contextes d'exécution, ex: VM (construction)
- Limiter les besoins de *bootstrap*[2], faciliter dev/mise au point

[2] Guix Reduces Bootstrap Seed by 50%

Kameleon: recette

Fichier Yaml décrivant les opérations dans chaque contexte

```
global:
  workdir: /tmp/kameleon
  distrib: debian
  debian_version_name: etch
  distrib_repository:  http://archive.debian.org/debian-archive/debian/
  output_environment_file_system_type: ext3
  arch: i386
  network_hostname: "test"
  extra_packages: "mysql-server mysql-client mingetty "
  oar_repository: "deb http://oar-ftp.imag.fr/oar/2.2/debian/stable/ ./"
steps:
  - bootstrap
  - system_config
  - mount_proc
  - software_install:
  - extra_packages
  - oar_2.2/oar_debian_install
  - oar_2.2/oar_system_config
  - oar_2.2/oar_config
  - autologin
  - kernel_install
  - umount_proc
  - build_appliance_kpartx:
    - create_raw_image
    - attach_kpartx_device
    - mkfs
    - mount_image
    - copy_system_tree
    - install_extlinux
    - umount_image
    - save_as_vdi
```

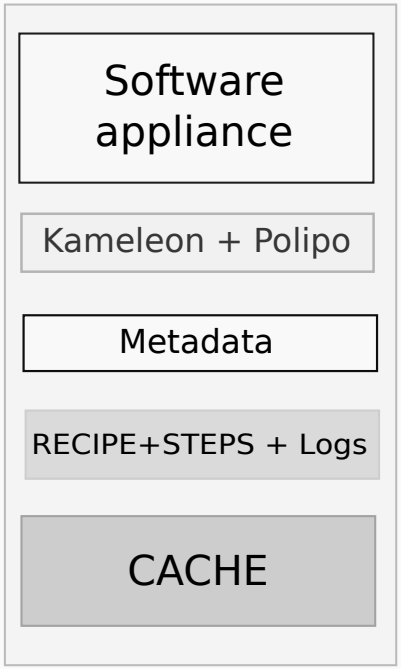


```
oar_config:
  - config_mysql:
      - exec_chroot: /etc/init.d/mysql start || service mysql start || true
      - exec_on_clean: chroot $$chroot bash -c "/etc/init.d/mysql stop || true"
  - mysql_db_init:
      - exec_appliance: cp $$stepdir/data/oar_mysql_db_init $$chroot/usr/lib/oar/
      - exec_chroot: oar_mysql_db_init
  - update_hostfile:
      - append_file:
          - /etc/hosts
          - |
             127.0.0.1 node1 node2
  - create_resources:
      - exec_chroot: oarnodesetting -a -h node1
```

Kameleon: garantir la reconstruction - archive

Une archive *presque autosuffisante*

Kameleon Archive



Limitations

- Le développement d'une nouvelle recette est délicat
- Le temps d'une contruction $>$ 5 min
- Peu adapté pour des modifications fréquentes, les tests...

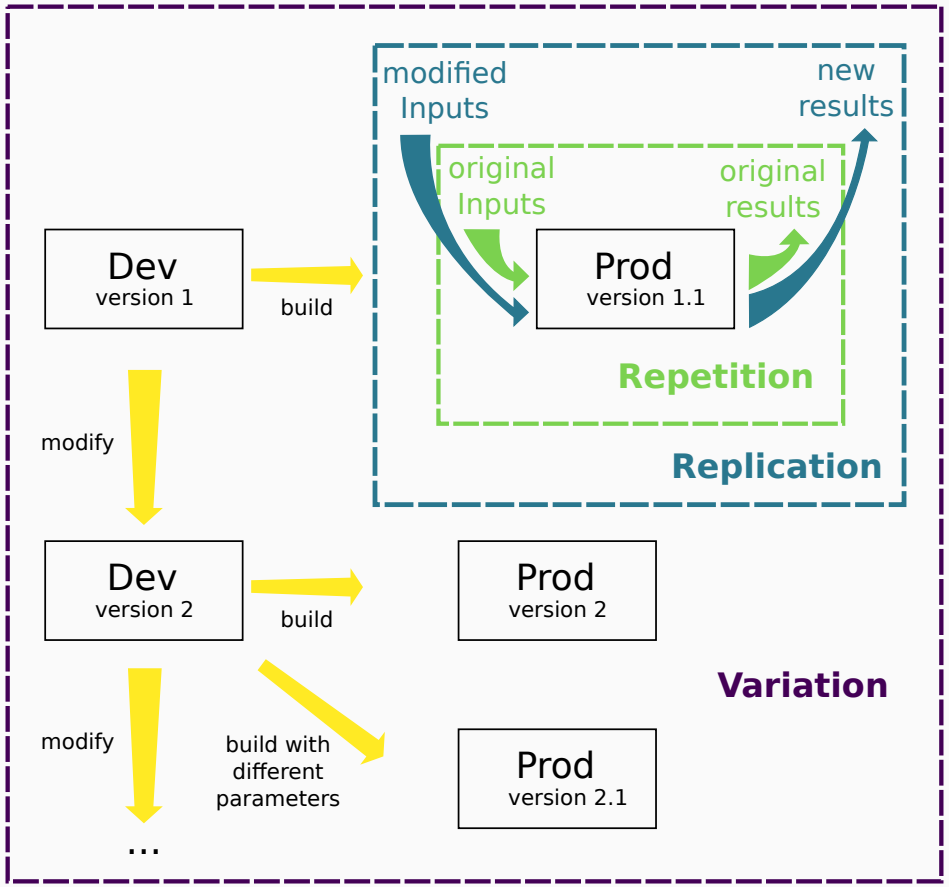
Modifications et variations (1)

Les composés d'une expérience sont des cibles mouvantes

Figer les code + data pour une publication n'est pas suffisant

Modifications et variations (2)

Répéter / Répliquer / Varier



- Reproductibilité se transforme en un besoin d'élaboration de processus (similarité forte avec les problématique de DevOps)

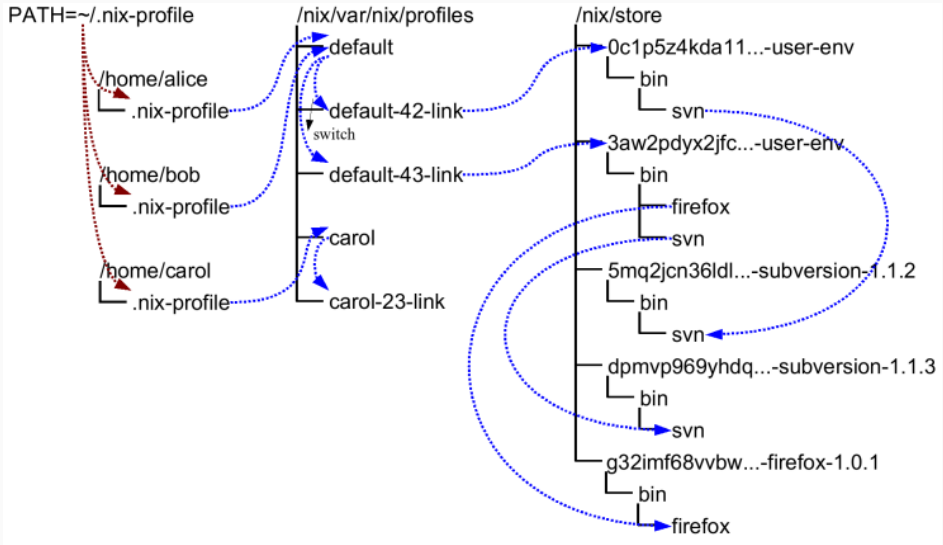
Et les gestionnaires de paquet fonctionnels ???

- Une nouvelle facon d'aborder la réalistation de paquets, de piles de logicielles, images systèmes...
- Support pour les processus aidant à la maitrise de la reproductibilité, avec répétition dans différents contextes, varitions, extensions...

L'approche par gestionnaire de paquets fonctionnel

Nix (2003) et Guix (2012)

- Langage fonctionnel décrivant la construction des paquets
- Une zone de stockage (store) en lecture seule (read only)
- Hash crypto sur (source, dépendances, expressions pour la construction)
- Notion de profil (vers quoi pointe le $PATH)
- Transactionnel (retour en arriére, à la demande ou en cas d'erreur)



- Résout le *dependencies hell* problème

Mais encore

- Notion de dérivation -> unité de construction bas niveau
- Cache binaire (on ne recompile pas tout)
- Profiter des propriétés des langages fonctionels (composabilité, peu/pas d'effet de bord)
- Multi-architecture: x86, power, arm64, riscv
- Multi-plateforme: Linux, Darwin (Nix), Gnu Hurd (Guix)
- Distributions completes: Nixos / GNU Guix System
- Nix langage: un DSL domain-specific lang. (un sorte de Json + λ)
- Guix langage: Guile (Lisp de la famille Scheme) general purpose language

Nix: paquet Hello

```
{ lib, stdenv, fetchurl }:

stdenv.mkDerivation rec {
  pname = "hello";
  version = "2.10";

  src = fetchurl {
    url = "mirror://gnu/hello/${pname}-${version}.tar.gz";
    sha256 = "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i";
  };

  doCheck = true;

  meta = with lib; {
    description = "A program that produces a familiar, friendly greeting";
    longDescription = ''
      GNU Hello is a program that prints "Hello, world!" when you run it.
      It is fully customizable.
    '';
    homepage = "https://www.gnu.org/software/hello/manual/";
    changelog = "https://git.savannah.gnu.org/cgit/hello.git/plain/NEWS?h=v${version}";
    license = licenses.gpl3Plus;
    maintainers = [ maintainers.eelco ];
    platforms = platforms.all;
  };
}
```

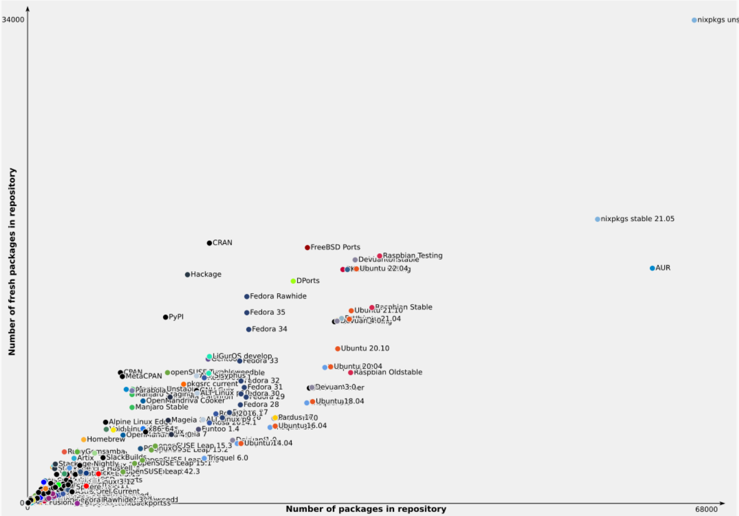
Guix: paquet Hello

```
(define-public hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                  ".tar.gz"))
              (sha256
               (base32
                "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
    (build-system gnu-build-system)
    (synopsis "Hello, GNU world: An example GNU package")
    (description
     "GNU Hello prints the message \"Hello, world!\" and then exits.  It
serves as an example of standard GNU coding practices.  As such, it supports
command-line arguments, multiple languages, and so on.")
    (home-page "https://www.gnu.org/software/hello/")
    (license gpl3+)))
```

Des trucs cools

- Garantie forte de reproductibilité, de (re)-construction avec variation
- Générer simplement des images VM, container, ramdisk, des archives...
- Des environnemets (profils) temporaires (pour le dev, tests)
- Travailler sur la sobriété (viser des piles logicielles réduites)
- Envergures techniques et applicatives plus vastes que les approches comme Spack, Flatpack, Snap...
- Satisfaction lorsque l'on contribue à une expression/partage

Popularités, diffusions



`https://repology.org/repositories/graphs`

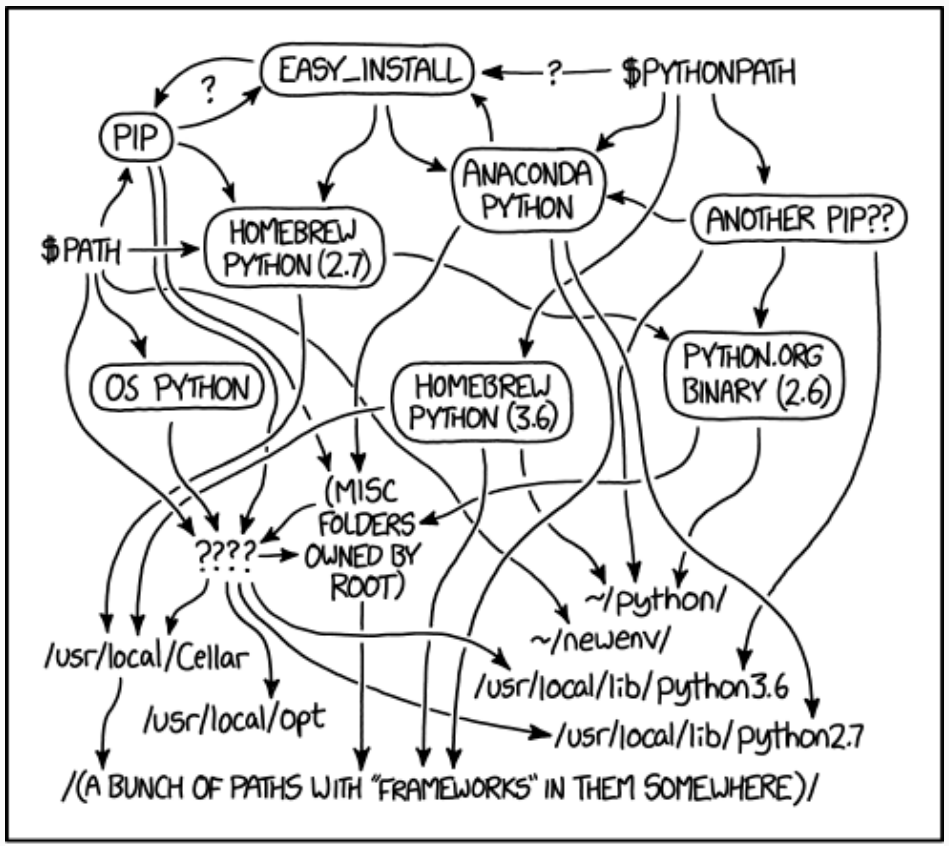
- Guix 19,315 paquets (au 24/11/21)
- Nix/NixOS/Nixpkgs Nombre de mainteneurs: 1955 (Debian 3593)

Crise de croissance pour Nix/NixOS

- 60 K paquets c'est beaucoup (nixpkgs: 333K commits / taille 30Mo)
  - Mais la centralisation a ses avantages (*curating*)
- Le projet NUR: Nix User Repository
  - exemple équipe DataMove Inria/LIG: nur-kapack - 60 paquets, une 10 dizaines de services (*systemd*)
- Décentralisation / composabilité via les flakes
- Mécanisme de RFC pour la gouvernance
- Développement organique

Les trucs difficiles:

- Certains langages... Python: quel est la bonne méthode ???
  - setup.py, poetry, wheel, conda, pip, virtualenv, flit, requirement.txt
    ...
- Installation system wide / vendoring
- Developpement vs usage/paquet final
- Projet avec compilation complexe, multi-langages



En pratique

- Hors utilisation de base l'apprentissage est (parfois) rude pour les débutants
  - Principes, langage et problèmatique de l'empaquetage
  - Rappellez-vous pour le passage à Git ;)
- Fonctionne bien au sein d'une équipe:
  - Tout les membres n'ont pas le / besoin du même niveau d'expertise
    - Expertise automake, autoconf...
    - Mise en place de bonne pratique, cookbook...
- Le gain sur moyen et long terme est important
  - Reproductibilité !
  - Processus devops, CI/CD, testing... (curating)
  - L'accueil des stagiaires, demo, diffusion...

Et après

Infrastructure as Code

```
import ./make-test-python.nix ({
  nodes = {
    one =
      { pkgs, lib, ... }: {
          environment.systemPackages = [ pkgs.jq ];
          services = {
            logstash.enable = true;
            elasticsearch.enable = true;
            kibana.enable = true;

            elasticsearch-curator = {
              enable = true;
              actionYAML = ''
              ---
              actions:
                1:
               ...
              '';
            };};};
    };
  testScript = ''
    start_all()
    one.wait_for_unit("elasticsearch.service")
    one.wait_for_open_port(9200)
    ...
  '';
```

Et après

- Les languages Guile et Nix ont leurs limites (+ pour Nix)
- Principalement le support du typage
  - C'est une dette technique bas bruit (cf. Python)
  - Des questions similaires sur les langages pour la configuration: CUE, Jsonnet, Dhall
  - Le langage Nickel `https://nickel-lang.org` pour être un début de response
  - CONFLANG 2021: `https://2021.splashcon.org/home/conflang-2021`

Conclusion

- Reproductibilité: un/des processus
- Complexité des piles locigielles grandissante
- Les FPM (Guix et Nix) apportent des éléments de réponse élégants
  - Bien installés dans le paysage
- Requiert l'acquisition de nouvelles *expertises*
  - Courbe d'apprentissage
- Importance des communautés / des équipes / des collectifs

Questions ?

- Merci pour votre attention...
- Café Guix: `https://hpc.guix.info/events/2021/café-guix/`
- Discourse Nix/NixOS: `https://discourse.nixos.org/`
- Mattermost (local Grenoble) `https://framateam.org/nix-tm`

example nginx

```
import  ./make-test-python.nix ({ pkgs, ... }: {
  nodes = {
    server = { pkgs, ... }: {
      services.nginx = {
        enable = true;
        # a minimal site with one page
        virtualHosts.default = {
          root = pkgs.runCommand "testdir" { } ''
            mkdir "$out"
            echo hello world > "$out/index.html"
          '';
        };
      };
      networking.firewall.enable = false;
    };
    client = { ... }: { };
  };
  testScript = ''
    server.wait_for_unit("nginx.service")
    client.wait_for_unit("network.target")
    assert "hello world" in client.succeed("curl -sSf http://server/")
  '';
})
```