Guix (& Nix)

Retour d'expérience GRICAD

Pierre-Antoine Bouttier

Contexte et historique

Notre contexte

- 3 clusters de calcul en propre (environ 10000 coeurs CPU)…
- …reliés entre eux et à des serveurs de labos par une grille de calcul locale (*CiGri*)
- Utilisations :
  - HPC
  - Traitement de données
  - HTC
  - Visu, formation & développement
- **Grande hétérogénéité** des usages, des communautés, des niveaux de compétences utilisateur

The Good Ol' days

- Jusqu'à 2015, utilisation de module
  - Classique, bien connu
  - Utilisé au tiers-1
  - Usages et communautés (beaucoup) plus homogènes
- Mais des gênes se font sentir :
  - **Portabilité très moyenne** (indispensable pour notre petite grille !)
  - **Duplication des efforts** proportionnelle au nombre de clusters
  - **Aspect communautaire marginal** (*e.g.* pas de partage direct de paquets)
  - ...

L'utopie logicielle pour le HPC

- **Maintenance** (arbre des dépendances bien géré, système-indépendant), **reproductibilité** (version unique d'un paquet - src, compilation, desc, déf,...- a la même sortie où que ce soit), **portabilité**
- **Gestion avancée des permissions** (environnement utilisateur, paquets/environnements spécifiques à une communauté, licences)
- **Workflow automatisé** : paquets *custom*, rebuilds automatiques, du PC perso aux clusters, CI
- Options de compilation avancée
- **OS-indépendant**

Welcome to the jungle (out-of-date?)

	easybuild	Spack	/ GuixHPC
Build Reproducibility	up to system libs	up to system libs	almost binary
User Environments	module-based	module-based	pure, isolated
Runtime env	module/os-based	module/os-based	static deps
Portability	no	no	yes
Packaging Language	Python/Easyconfig/tcl	Python	Nix
Multiple versions	yes	yes	rebuild/grafts
Binary packages	no	yes	yes
Isolated build env.	no	no	yes
External/Legacy Modules	yes	?	no
Who packages	anyone?	no	anyone
Who builds	admin and user?	admin or user	admin and user on head (daemon+store)
Community/Doc	HPC	HPC	large
Custom Packages	?	?	source-based

Here comes a new challenger

- En développant une solution maison (à base de liens symboliques et une glibc embarquée…), Bruno Bzeznik est tombé sur **Nix** :
  - Orienté reproductibilité
  - Très faible dépendance au système d'exploitation hôte
  - Communauté jeune mais active
  - En 2015, on (enfin *il*) se lance

Nix (& Guix), briefly

- Les paquets et environnements sont définis par une expression Nix (fichier source)
- OS-indépendant
- Mise à niveau ou rollbacks atomiques
- Plusieurs versions d'un même paquet pour le même utilisateur
- Pas de privilège pour installer les paquets
- Fourni des environnements **isolés** de construction et d'exécution
- Reproductibilité à partir des sources
- Le binary cache
- Un garbage collector

L'installation de Guix

- Montages NFS `/gnu/store` et `/var/guix` sur tous les clusters
- Install "classique" depuis la doc de Guix, en root, sur un cluster
- Utilisateurs guixbuilder01 à guixbuilder10 et le groupe guixbuild sur le même cluster
- Lancement du démon guix sur le même cluster et écoute en tcp depuis les autres clusters.
- Un script à sourcer, côté utilisateur, sur chaque cluster à chaque connexion

Ce qui a changé avec Guix (& Nix) depuis module

Côté utilisateur

- Débutants :
  - Pas de problèmes spécifiques, ils suivent la doc.
- Historiques :
  - Adaptation par rapport à module (assez rapide)
  - Mais quelques soucis pour certains softs… (j'y reviendrai)
- Utilisateurs puissants :
  - Prise en main OK
  - (Rares) appropriations de l'outil (nix-shell, guix environmeent, manifest.scm, guix describe, etc.)

Côté support

- Langage déroutant
- Courbe d'apprentissage raide (peu de chances que l'utilisateur soit développeur)
- Peut demander beaucoup de temps pour empaqueter binaires/soft proprios/aux archi. logicielles exotiques
- Mais plein de bénéfices ! (aucun regret)
  - base énorme de paquets existants
  - Guix environment, profiles, etc.
  - Portabilité
  - Reproductibilité, etc.

The dark side (Nix & Guix)

- Communauté HPC encore petite et peu de centralisation (channel dédié ?)
- Compliqué dès qu'on sort de l'open source
  - Suite Intel (compilateurs, MPI, MKL)
  - Driver NVIDIA+CUDA (et calcul sur GPU)
  - Casse-tête avec certains softs python et visu
- Manque (?) de docs type "tutos" pour faire découvrir aux utilisateurs (et développeurs, et ASR) les potentialités de l'outil

Guix VS Nix

Installation de paquets

Nix

```
$ nix-env -i gcc # Installe les compilos GNU
$ nix-env -q # Liste des paquets installés dans l'environnement courant
gcc-9.2.0
$ nix-env -e gcc # Désinstalle le paquet
```

Guix

```
$ guix install gcc-toolchain
$ guix package -I
gcc-toolchain   11.1.0  out     /gnu/store/ji09aab6as5qqyda9h0yvif55zla0rdi-gcc-toolchain-11.1.0
$ guix remove gcc-toolchain
```

Définition de paquets NIX

```
{ 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 stdenv.lib; {
    description = "A program that produces a familiar, friendly greeting";
    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;
  };
}
```

Définition de paquets GUIX

```
(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.")
    (home-page "https://www.gnu.org/software/hello/")
    (license gpl3+)))
```

Gestion des profiles

Nix

- `nix-env --switch-profile monProfil`
- Même dossier, liens qui changent
- Pratique pour le calcul multinoeud

Gestion des profiles

Guix

- `. "$GUIX_PROFILE"/etc/profile`
- Dossiers (et var. d'env.) qui changent
- Moins *direct* pour les utilisateurs…
- …mais plus facilement versatile (*empilement* des profiles, gestion par dossier plus explicite)
- Manque une interface plus explicite ?

Et le reste

- Environnement python : avantage GUIX
- Compilation à la main : avantage GUIX
- Communautés : Plus large pour Nix, plus proche pour GUIX
- Support MacOS : avantage Nix
- Préférence personnelle : GUIX (*Comment ça, ça se voit ?*)

Pour finir

Résumé de notre retour d'exp - Part One

- Se passer de module se fait bien, pas du tout bloquant en soi.
- Les fonctionnalités de Nix & Guix sont de vrais atouts pour le calcul scientifique (au sens large)
- Guix nous semble plus adapté que Nix, en particulier pour les utilisateurs et le support (plus accessible)
- Quelques problèmes encore sur la table : GPU (conda environment...), intel-suite.

Résumé de notre retour d'exp - Part Two

- Adopter un nouvel outil prend du temps (surtout quand il est jeune, comme Nix ou Guix)
- On ne profite/alimente pas encore assez l'aspect communautaire de ces outils
- On doit encore monter en compétence sur guix
- La cathédrale (*module*) et le bazar (*module, nix, guix, conda, singularity*...)

Quelques suggestions

Comment casser la dépendance au chemin ?

- Une base de connaissance, orientée pratique (tutos, HOWTO), qui s'adresse à différents niveaux (utilisateurs, support, développeurs/mainteneurs de paquets)
- Un dépôt central HPC ? (guix-science ?)
- un fléchage clair et centralisé :
  - Packaging
  - Documentation
  - Canaux de communication
- Qu'on (GRICAD) communique plus (questions, besoins, passage en upstream) !
- Montrer et répéter les nombreux avantages à proposer et à s'approprier GUIX !

Merci de votre attention !

Questions/Discussion