Guix@MESONET

...ou comment mutualiser la gestion des environnements logiciels

Guix@MESONET - 27/06/2024

P.-A. Bouttier

TOC

TOC

Le contexte à GRICAD

- ~~4~~ 3 clusters de calcul en propre (un peu plus de 6500 coeurs CPU, environ 100 GPUs)…
- …reliés entre eux et à des clusters de labos par une grille de calcul locale
- Utilisations :
  - HPC, HTC
  - Traitement de données
  - Visu, formation & développement
- **Grande hétérogénéité** des usages, des communautés, des niveaux de compétences utilisateur et, du coup, des piles logicielles
- **Le défi** : trouver un outil de gestion d'environnement logiciel qui réponde à des contraintes très hétérogènes

Le contexte de la politique scientifique

- **La Science Ouverte** : Comment assurer la reproductibilité des traitements numériques ?
  - Bonnes pratiques de développement logiciel
  - Et l'environnement logiciel ?
- Changements de pratique
  - Utilisation croissantes des plateformes non-HPC : jupyterhub, binderhub, cloud computing...
  - ...dans la même chaîne de traitement
  - Développement de plateformes calcul/données distribuées

The Good Ol' days

- Jusqu'à 2015, utilisation de module
  - Classique, bien connu
  - Utilisé au tiers-1 et au tiers-2
  - 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, binaires)
  - ...

L'utopie des environnements logiciels ?

- **Isolation par rapport au système hôte**, à l'exécution ET à la construction
- **Maintenance** (arbre des dépendances bien géré), **reproductibilité** (version unique d'un paquet - src, compilation, desc, déf,...- a la même sortie où que ce soit), **portabilité**
- **Complètement fonctionnel en espace utilisateur**
- **Workflow automatisé** : paquets **custom**, rebuilds automatiques, du PC perso aux clusters, CI
- **OS-indépendant**
- Installation de **Nix** en 2015 puis **GUIX** en 2018.

TOC

Les mécanismes fondamentaux des FPM

Les gestionnaires de paquets fonctionnels :

- Langage fonctionnel pour décrire la construction des paquets
- Hash crypto sur (sources, dépendances, expressions pour la construction)
- Notion de profil/shell via les variables d'environnement
- Transactionnel (rollback possible)
- Isolation à la construction

Quelques définitions

- **Un paquet Guix** : Une **définition** (=code source, fichier texte brut) de l'ensemble des instructions et dépendances pour installer un logiciel. On peut regrouper des définitions de paquets en **modules** (au sens Python, pas au sens `module`).
- **Un channel Guix** : un **dépôt git** contenant **un ensemble de définitions de paquets** (et quelques fichiers de configurations). Un numéro de commit particulier (donc un état bien identifié des définitions) de ce dépôt peut être appelé révision.

Le graphe de dépendance

Un commit précis d'un channel Guix peut être vu comme un instantané (unique) du graphe de dépendances entier des dizaines de milliers de logiciels empaquetés !

En pratique

À la demande de construction d'un logiciel (e.g. `guix install`, `guix shell`)

- Guix vérifie dans `/gnu/store` s'il n'est pas déjà construit
- Si non, il va vérifier les **caches binaires distants**
- Si non, il construit le paquet de façon isolée et place les sorties (binaires, libs, headers, doc, etc.) dans un répertoire unique de `/gnu/store`
- Des **liens symboliques** sont créés dans le `$HOME` de l'utilisateur
- **Les variables d'environnements** font le reste c$oté utilisateur

TOC

Guix, briefly

Les commandes lancées par les utilisateurs :

```
$ guix pull # équivalent d'apt update
$ guix upgrade # équivalent d'apt ugrade
$ guix search monpaquet
$ guix install monpaquet
$ guix shell monpaquet # équivalent de venv mais pour toute pile logicielle
$ guix pack monpaquet # création d'une image de conteneur
```

Déployer et reproduire facilement une pile logicielle -- First step

Si vous connaissez l'ensemble des paquets dont vous avez besoin, vous pouvez écrire un fichier `manifest.scm` qui contiendra les noms des paquets :

```
(specifications->manifest
  (list "gcc-toolchain"
        "openmpi"
        "hdf5-openmpi"))
```

- **Description centralisée** de tout votre env. log.
- Fichier que vous pouvez intégrer à votre code, vos scripts et aussi **versionner** !

Déployer et reproduire facilement une pile logicielle -- Second step

Pour reproduire un env.log. il faut donc le décrire le plus complètement possible. Nous avons vu auparavant le fichier `manifest.scm`. Il manque une information : l'état des définitions/du graphe des dépendances (≃versions).

Nous avons vu précdemment qu'en réalité que celui-ci peut être décrit par la **liste des channels Guix utilisés et leur numéro de commit courant**. Guix propose une commande qui regroupe toutes ces informations :

```
guix describe -f channels >> channels.scm
```

Déployer et reproduire facilement une pile logicielle -- Final step

Ailleurs et/ou plus tard :

- On clone le dépôt du code (scripts, doc, `manifest.scm`, `channels.scm`)
- On lance la commande suivante :

```
guix time-machine -C channels.scm -- shell -C -m manifest.scm
```

- `guix time-machine` donne accès à d'autres révisions de guix et lance la commande `guix` indiquée après `--` dans cette révision.
- Nous déployons ici bien l'environnement logiciel indiqué dans l'état spécifié.

Guix, Pros and Cons (1/2) on the user side

Pros :

- Reproductible et donc réellement portable (au contraire de spack ou conda)
- Très pratique pour les environnements de développements (`venv` mais pour tout et partout)
- Commandes Guix faciles à prendre en main
- Grosses bases de paquets (entre 20 000 et 50 000 si l'on rajoute des channels spécifiques)
- Utilisateur autonome (si paquet existe)

Guix, Pros and Cons on the user side

Cons :

- Un nouvel outil : besoin de support (doc, accompagnement)
- Certaines piles logicielles importantes restent à empaqueter : Intel OneApi, frameworks IA et GPU NVIDIA

TOC

Comment nous avons mis Guix à disposition à GRICAD ?

- Première étape : lecture de la doc !
- En très bref :
  - Un noeud principal sur lequel on va installer Guix proprement dit
  - Les noeuds de calcul qui doivent être configurés

Configurer le noeud principal

Quelques recommandations

- Préférer un serveur dédié, bare-metal
- Stockage :
  - un volume RAID avec quelques TB pour être tranquille
  - Des exports NFS
  - Évitez les FS distribués (sensibilité aux stats storm)
- Construction
  - Assez de CPU pour la compilation en parallèle

Configurer le noeud principal : les comptes locaux

- On installe Guix suivant la doc sur ce noeud.
- On créée les utilisateurs chargés de lancer les processus de build :

```
root@guix:~# grep _guixbuilder /etc/passwd
_guixbuilder0:x:996:996:Guix build user 0:/var/empty:/usr/sbin/nologin
_guixbuilder1:x:995:995:Guix build user 1:/var/empty:/usr/sbin/nologin
_guixbuilder2:x:994:994:Guix build user 2:/var/empty:/usr/sbin/nologin
_guixbuilder3:x:993:993:Guix build user 3:/var/empty:/usr/sbin/nologin
_guixbuilder4:x:992:992:Guix build user 4:/var/empty:/usr/sbin/nologin
_guixbuilder5:x:991:991:Guix build user 5:/var/empty:/usr/sbin/nologin
_guixbuilder6:x:990:990:Guix build user 6:/var/empty:/usr/sbin/nologin
_guixbuilder7:x:989:989:Guix build user 7:/var/empty:/usr/sbin/nologin
_guixbuilder8:x:988:988:Guix build user 8:/var/empty:/usr/sbin/nologin
_guixbuilder9:x:987:987:Guix build user 9:/var/empty:/usr/sbin/nologin
```

Configurer le noeud principal : le démon Guix

Editer `/etc/systemd/system/guix-daemon.service` :

```
ExecStart=/var/guix/profiles/per-user/root/current-guix/bin/guix-daemon \
        --build-users-group=guixbuild \
        --listen=/var/guix/daemon-socket/socket \
        --listen=0.0.0.0

# The --listen=0.0.0.0 option will make the daemon
# listen to all your networks (you may restrict to some local network interface)
```

Configurer le noeud principal : les exports NFS

- `/gnu/store *(ro)` : Big storage **read-only**
- `/var/guix *(rw, async)` : User profiles r/w
- `/var/log/guix *(ro)` : Logs from the daemon

Les noeuds de calcul

Montage des répertories Guix :

```
head-node:/gnu/store    /gnu/store    nfs  defaults,_netdev,vers=3 0 0
head-node:/var/guix     /var/guix     nfs  defaults,_netdev,vers=3 0 0
head-node:/var/log/guix /var/log/guix nfs  defaults,_netdev,vers=3 0 0
```

Les noeuds de calcul : un env Guix minimal

```
$ source /applis/site/guix.sh
```

```
export GUIX_PROFILE=$HOME/.guix-profile/
export GUIX_USER_PROFILE_DIR=/var/guix/profiles/per-user/$USER
export GUIX_DAEMON_SOCKET="guix://head-node"
export PATH=/var/guix/profiles/per-user/root/current-guix/bin:$PATH
export GUIX_LOCPATH=/var/guix/profiles/per-user/root/guix-profile/lib/locale
export USERGUIXPATH=$HOME/.config/guix/current
export INFOPATH="$USERGUIXPATH/share/info:$INFOPATH"
source $USERGUIXPATH/etc/profile
source $USERGUIXPATH/etc/bash_completion.d/guix
```

The very minimal should be:

- guix command in the PATH of the users
- GUIX_DAEMON_SOCKET set to the master host (guix-daemon)

Au niveau réseau

- Le noeud principal agit comme un proxy pour les noeuds de calcul
- Il doit avoir accès au moins à https://ci.guix.gnu.org
- Il doit avoir accès également aux fichiers sources des paquets qui ne sont pas pré-construits, mais possible de le faire offline (cf. https://guix.gnu.org/cookbook/fr/html_node/Acces-reseau-de-la-grappe.html)

TOC

Le coût de Guix

- Pour les utilisateurs finaux :
  - Travailler sous GNU/Linux avec où Guix est disponible
  - Empaquetage des logiciels visés (n'hésitez pas à demander)
  - ?
- Pour les personnels support :
  - Empaqueter les logiciels (prog. fonctionnelle)
  - ?

Les bénéfices de Guix

Au-delà de la reproductibilité :

- Portabilité
- `virtualenv` pour tout type d'env logiciel !
- Le voyage dans le temps et l'espace robuste et fiable…
- …peu importe le système hôte
- Une communauté dynamique, sympa et en plein essor ! (notamment côté calcul scientifique)
- Participer à un projet communautaire
- Plein d'outils ! : Guix Worflow Langage, l'option `--tune`, `guix-jupyter`, **lien avec Software Heritage**, etc.

Pour le HPC

- Des channels dédiés, une communauté existante et accueillante (liste de diffusion, mattermost.univ-nantes.fr, café Guix, docs, tutos)
- Partage des recettes de constructions : un gros gain de temps et de ressources
- Des ressources humaines qui peuvent aider (empaquetage, installation, support)

Guix et Mesonet

- Proposition :
  - Mise à disposition de Guix, où c'est possible (on peut vous aider)
  - Création d'un channel dédié Mesonet (ou HPC), **tiers de confiance**
  - Documentation mutualisée
  - Formation du GSM

Quelques liens utiles

- La doc de guix et la doc d'instalaltion sur un cluster
- La doc utilisateur GRICAD
- Le channel GRICAD
- Le site Guix-HPC
- Le mattermost Guix-HPC

Merci à tous pour votre attention !

Un paquet guix

```
(define-public hello
  (package
    (name "hello")
    (version "2.12.1")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-"$
                                  ".tar.gz"))
              (sha256
               (base32
                "086vqwk2wl8zfs47sq2xpjc9k066ilmb8z6dn0q6ymwj$"
    (build-system gnu-build-system)
    (synopsis "Hello, GNU world: An example GNU package")
    (description
     "GNU Hello prints the message \"Hello, world!\" and then serves as an example of standard GNU coding practices.")
    (home-page "https://www.gnu.org/software/hello/")
    (license gpl3+)))
```

Cheat codes pour créer un paquet python, R, julia, etc.

```
guix import pypi --recursive nomDuPaquetPypi
```

Conteneurs et reproductibilité

2 parties dans les systèmes de conteneurs (**e.g.** docker, singularity):

- La construction de l'image (`docker` ou `singularity` sont plutôt mauvais du point de vue de la reproductibilité)
- L'exécution de l'image

Le Garbage Collector de Guix

Le garbage collector de Guix va vérifier l'ensemble des profiles pour savoir quels paquets (rép. `/gnu/store`) ne sont plus utilisés et les supprimer.

```
$ guix gc
```

La sécurité

Pour tracker les librairies compromises, par exemple, pour la glibc 2.25 avant corrections :

```
guix gc --referrers /gnu/store/...-glibc-2.25
```