

Nix : Révolutionner la gestion de paquets

Emeric Laberge

31 octobre 2024

Plan de la présentation

Introduction

Comment l'installer ?

Fonctionnement Technique

Point Original

Réflexion

Conclusion

Introduction

Qu'est-ce qu'un gestionnaire de paquets ?

- Outil de gestion des dépendances logicielles
- Permet d'installer, mettre à jour et désinstaller des paquets

En quoi c'est utile ?

- On peut installer des logiciels sans avoir à les compiler
- Facilite la gestion des dépendances (un programme peut en dépendre de plusieurs)
- Permet de garder les logiciels à jour

- Conflits de dépendances, notamment entre plusieurs versions d'une même dépendance,
- Environnements non reproductibles, un programme peut fonctionner sur une machine mais pas sur une autre.
- On doit utiliser des outils supplémentaires pour gérer ces problèmes (virtualenv, conda, docker, etc.)

Nix



Qu'est-ce que Nix ?

- **Quoi** : Gestionnaire de paquets purement fonctionnel avec plus de **100 000** paquets¹
- **Où** : Disponible sur Linux, MacOS et autres *Unix-like*
- **Auteur** : Eelco Dolstra
- **Date de sortie** : 15 juin 2003
- **Contexte** : Développé dans le cadre de sa thèse de doctorat *The Purely Functional Software Deployment Model* (2006)²
- **Autres détails** : Langage de programmation à part entière, logiciel libre (GNU LGPLv2.1)

1. Site officiel de Nix

2. The Purely Functional Software Deployment Model

Avantages de Nix

Reproductibilité :

- Paquets sont *build* de manière isolée les uns des autres.

Déclaratif :

- Facile de partager des n'importe quel type d'environnement production pour des projets et ce **peu importe la machine et les langages de programmation utilisés**
- Les dépendances sont déclarées dans un fichier *.nix*

Fiabilité :

- L'ajout ou la mise à jour d'un paquet **ne peut pas** affecter les autres paquets.
- Possibilité de **rollback** en cas de problème.

Comment l'installer ?

Sur Linux :

```
curl -L https://nixos.org/nix/install | sh
```

Sur MacOS :

```
sh <(curl -L https://nixos.org/nix/install)
```

Sur Windows (WSL2) :

```
sh <(curl -L https://nixos.org/nix/install) -daemon
```

Fonctionnement Technique

Aperçu de la syntaxe Nix

- **Valeurs de base :**
 - Chaînes : `"hello world"`
 - Booléens : `true`, `false`
 - Nombres : `123`, `3.141`
 - Chemins : `/etc`, `./foo.png`
- **Valeurs composées :**
 - Ensemble : `{ x = 1; y = 2; }`
 - Liste : `["foo" "bar" "baz"]`
- **Opérateurs et Contrôles :**
 - Concaténation : `"foo" + "bar"`
 - Sélection : `{ x = 1; }.x → 1`
 - Condition : `if 1 + 1 == 2 then "yes!" else "no!"`
 - Variables locales : `let x = "foo"; in x + "bar"`
- **Fonctions :**
 - Lambda simple : `x: x + 1`
 - Avec paramètres : `{ x, y }: x + y`
 - Utilisation de map : `map (x: x * 2) [1 2 3] → [2 4 6]`

- Outil pour créer des environnements isolés
- Exemple d'utilisation :

```
nix-shell -p python3 python3Packages.numpy
```

- **Effet** : Crée un environnement avec Python 3 et NumPy
- **Utilisation** : `python3 code.py`

Encore mieux

Déclarons les dépendances dans un fichier `shell.nix` :

```
#shell.nix
let
  pkgs = import <nixpkgs> {};
in pkgs.mkShell {
  packages = [
    (pkgs.python3.withPackages (python-pkgs: [
      python-pkgs.numpy
      python-pkgs.pandas
      python-pkgs.matplotlib
    ]))
  ];
  shellHook = ''
    python3 code.py
  '';
}
```

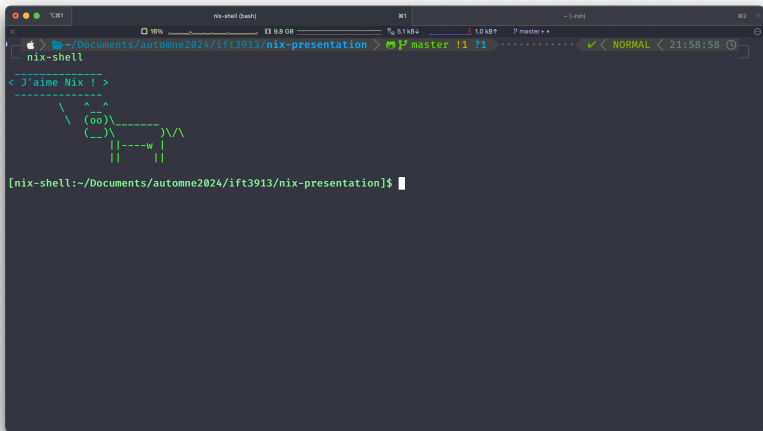
- **Commande** : `nix-shell`
- **Effet** : Crée un environnement avec les dépendances et exécute le `shellHook`, ici `'python3 code.py'`

TA-DA !

Autre exemple

```
#shell.nix
let
  pkgs = import <nixpkgs> {};
in pkgs.mkShell {
  packages = [
    lolcat
    cowsay
  ];
  shellHook = ''
    echo "J'aime Nix !" | cowsay | lolcat
  '';
}
```

Résultat



```
nix-shell (bash)
16% 9.8 GB 5.1 KB 1.0 KB master !1 71 NORMAL 21:58:58
< J'aime Nix ! >
  \  ^ ^
   (oo)\_____ )
    (__)\       )\/\
       ||----w |
       ||     ||

[nix-shell:~/Documents/automne2024/ift3913/nix-presentation]$
```

Installation de Paquets

- Installation isolée :

```
$ nix-env -iA nixpkgs.hello
```

- Aucun impact sur les autres paquets
- Désinstallation propre

Point Original

- Chaque utilisateur peut avoir son propre profil
- Partage efficace des ressources communes
- Sécurité renforcée

Réflexion

Avantages :

- Environnements cohérents et reproductibles.
- Déploiement simplifié pour les équipes de développement.

Défis :

- Courbe d'apprentissage pour les nouveaux utilisateurs.
- Intégration avec les systèmes existants peut être complexe.
- Nécessité de comprendre le modèle fonctionnel pour maximiser son potentiel.

Bien que Nix offre des avantages significatifs, il nécessite un investissement initial en temps d'apprentissage.

Conclusion

Pourquoi adopter Nix ?

- **Contrôle total** sur les environnements, pour éviter les erreurs de version.
- **Flexibilité multi-utilisateurs** et gestion facile des dépendances.
- **Reproductibilité** assurée, pour éviter le fameux "ça marche sur ma machine".

Merci de votre attention ! N'hésitez pas à poser des questions.

Des questions ?

Sources

Sources

1. Site officiel de Nix
2. Manuel de Nix
3. Guide Nixpkgs