Nix пакетния мениджър и команди

Универсални конфигурации с Nix

Павел Атанасов Камен Младенов

03.04.2025

Раздел 1

Преговор

Преговор

- Nix е пакетен мениджър, който разрешава големите трудности с менажиране на зависимости и варианти
- Инсталирахме Nix

- Nix трябва не само да инсталира и премахва програми
  - Менажиране по уникални директории спрямо хеш
  - Създаване на чиста среда
  - Инструкции за компилиране в тази среда
  - Премахване на неизползвани зависимости чрез апроксимиращия метод

Раздел 2

Командите

Основни

- `nix-store` - управление на всички директории под `/nix/store`
- `nix-build` - компилира програма, спрямо подадените инструкции, в чистата среда
- `nix-env` - стандартните операции за пакетни мениджъри

Част от останалите

- `nix-channel` - менажиране на “уеб адресите” в които пакетния мениджър търси “програми”
- `nix-hash` - изчислява хешът на дадена директория
- `nix-collect-garbage` - изтрива всички програми, които не са използвани (спрямо апроксимиращия метод)
- `nix-shell` - създава нов интерактивен shell, в който са налични само предопределените програми (чрез `PATH` променливата)

Раздел 3

V2 срещу V3

V2 срещу V3

- Намираме се в междинен период
- Тези команди се водят компоненти от “Версия 2 Nix системата”
- В момента активно се разработва Версия 3, която е **значително** по-елегантна и мощна
- Обаче все още е “експериментална”, и изисква материал от към средата на академията *(въпреки че почти цялата общност я използва вече)*

- Ще използваме командите на двете версии
- Версия 2 в началото, защото са по-простички и по-близко до концепциите
- Версия 3, защото са по-добри, предоставят неща които ги няма в предходната версия

Разлики

- Под Версия 2, всяка команда е различен изпълним файл.

  `nix-build`, `nix-shell`, `nix-store`, ...
- Под Версия 3, всичко е обединено като под-команди на една команда

  `nix build`, `nix shell`, `nix store`, ...

- Версия 2 притежава командата `nix-env` (и `nix-channel`)
- Във Версия 3 имаме тотални алтернативни механизми, които ни освобождават от нуждата от команди като `nix-env` (има `nix profile` за крайни цели, но няма да го обсъждаме сега)
- Все пак Версия 2 е официалната, затова документациите описват как да работим с `nix-env` (и `nix-channel`)
- **Ние няма да разглеждаме `nix-env` и `nix-channel`!**

Включване на Версия 3 командите

1. Временно:

   ```
   nix --extra-experimental-features "nix-command flakes" ...
   ```

2. Постоянно - в `~/.config/nix/nix.conf` или `/etc/nix/nix.conf` се добавя:

   ```
   experimental-features = nix-command flakes
   ```

Раздел 4

Кратък увод в Nix езика

Защо?

- Намекнахме по-рано, че трябва да подадем инструкции на `nix-shell`, чрез които програми се компилират
- Тези инструкции се описват чрез специален език, също наречен `Nix`

  (програмистите са добри в наименуване на неща)
- Най-важните ни команди са `nix-build`, `nix-shell` и `nix-store`

  Всичките се нуждаят от инструкции, описани на `Nix` езика

Основи

- Функционален, лениво-оценен, динамично-типизиран

Поддържани стойности

- Булеви: `true`, `false`
- Числа: `2847`, `-301`, `3.14159`
- Низове: `"I am a piece of text!"`

Контейнери

- Атрибутно множество:
  `{ attribute_name1 = value1; attribute_name2 = value2; }`
- Списък (с елементи от различен тип):
  `[ 3 "Hello" { x = 10; } true ]`

Извикване на функции

- Пишем името, и след това всеки аргумент, разделен с шпация
- `my_function argument1 argument2`

Раздел 5

Създаване на пакети

Как описване инструкциите, нужни за компилиране на програми?

- За наше удобство са създадени няколко функции, които ни улесняват работата (так. нар. “trivial builders”)
- Трябва да бъдат внесени чрез `with import <nixpkgs> { };` (за сега не е важно какво прави това)
- Инструкции на Nix езика ще пишем в **текстови** файлове, завършващи на `.nix`

writeTextFile

- Подаваме (атрибутно множество с) име на текстов файл и съдържание.
- `nix-build` ще създаде този текстов файл и ще го запише в `/nix/store`

Пример

```
with import <nixpkgs> { };
writeTextFile { name = "something.txt"; text = "Hello!"; }
```

- Ако искаме `text` да е на много редове, низът става грозен:

```
with import <nixpkgs> { };
writeTextFile { name = "something.txt"; text = "First line
second line
third line"; }
```

- В Nix езикът има и так. нар. “многоредов низ”, който автоматично премахва водещата идентация, когато тя не е значима:

```
with import <nixpkgs> { };
writeTextFile { name = "something.txt"; text = ''
    First line
      second line
    third line
  ''; }
```

writeShellApplication

- Аналогично на `writeTextFile`, обаче създава изпълним shell скрипт

Пример

```
with import <nixpkgs> { };
writeShellApplication { name = "script.sh"; text = "echo 'Hello World!'"; }
```

- Главната разлика е, че можем да определим зависимости

Това ще хвърли грешка

```
with import <nixpkgs> { };
writeShellApplication { name = "script.sh"; text = "fastfetch"; }
```

Така добавяме зависимостта

```
with import <nixpkgs> { };
writeShellApplication { name = "script.sh";
runtimeInputs = [ fastfetch ]; text = "fastfetch"; }
```

- Всички възможни програми идват от този магически ред с `import`
- Можем да ги търсим/разглеждаме лесно на search.nixos.org/packages

Раздел 6

Още команди и тяхната употреба

nix-instantiate

- Изпълнението на инструкции за компилиране на програми най-накрая се свежда в един голям shell скрипт, който изпълнява нужните команди за дадената цел
- Nix код се превръща в този скрипт (+ някои допълнителни данни), наричаме резултата **деривация**
- `nix-instantiate` създава деривация от подаден `.nix` файл и я запазва в `/nix/store`, като файл завършващ на `.drv`

nix-store и деривации

- Съответно, `nix-store --realize` изпълнява (скрипта в) дадената деривация
- `nix-build ./file.nix` е почти същото като `nix-instantiate ./file.nix && nix-store --realize /nix/store/....drv`
- `nix-store` също може да ни покаже самия скрипт, чрез флага `--print-env`

nix-shell

- Позволява ни да влезем в shell с всички зависимости на програмата

Пример

```
with import <nixpkgs> { };
writeShellApplication {
    name = "page_lines";
    runtimeInputs = [ curl cloc ];
    text = "curl -s \"$1\" | cloc --force-lang=html -";
}
```

- Много често искаме shell с някаква редица програми, без да сме определили пакет
- Постига се с `-p` флага

```
nix-shell -p curl cloc
```

- Даже е толкова често срещана нужда, създадена е специална функция за деривация `mkShell`
- `packages` определя наличните програми, `shellHook` изпълнява shell команди веднага преди съответния shell да се отвори

Пример

```
with import <nixpkgs> { };
mkShell { packages = [ curl cloc ]; shellHook = "echo Hi"; }
```

nix-collect-garbage

- Една от най-значимите операции е да изтриваме програми
- Не го правим като казваме “изтрий тази програма”, ами извикваме `nix-collect-garbage`, която чрез апроксимиращия алгоритъм намира всички неизползвани пакети и ги премахва
- Всъщност `nix-collect-garbage` е друго име за `nix-store --gc`

Раздел 7

Въпроси?