Теми за проекти

Универсални конфигурации с Nix

Павел Атанасов Камен Младенов

10.06.2025

Какво научихме до сега

- Nix езикът
- Деривации, пакетиране за няколко програмни езика
- NixOS, модулната система
- flake-ове, overlay функции
- flake-parts
- DevOps процеси, DevOps чрез Nix

Раздел 1

Теми за финални проекти

Организационно

- По един човек на тема
- Реализиране на Nix
- Да е публично качено в GitHub (или подобна платформа). Ще ви поискаме линк по време на защитата.

- Контролно 3 и (директно след това) защита на проекти
- Ще трае около 3 часа: 45 минути контролно, 15 минути да проверим, защити по 30-45 минути
- Ще се състои в офиса на blocksense, на Христо Белчев 1 (ще качим инструкции в обявленията, някой ако е забравил от входния тест)
- За защитата, сядате с лаптоп (или ако не носите, отваряме наш, но трябва да е качено онлайн) и обяснявате проекта какъв е, как се ползва, показвате как сте го реализирали, питаме въпроси и т.н.

Кога и в колко часа ще се състои контролно 3 + защити?

- Кога?

Понеделник	Вторник	Сряда	Четвъртък	Петък	Събота	Неделя
23	24	25	26	27	28	29

- От колко часа (общо 3 часа заетост с минимални почивки)?

Общи условия за всички теми

- **Трябва да разбирате всеки ред Nix код, който сте написали!** Ще има намаляване на оценката в противен случай!
- Имплементацията на всяка тема трябва да присъства във flake
- Освен това трябва да реализирате втори flake, който използва първия по смислен начин (мислете го като демо)
- Трябва да изкарвате смислени грешки за логически проблеми при употреба
- Начина по който го реализирате е по избор: функция, деривация, модул, ...
- Употребата на библиотеки (Nix код извън `builtins` и `nixpkgs`) **не** са позволени

I тема – Агрегиране на NixOS конфигурации

Съдържат се *две* директории: “компоненти” и “конфигурации”.

Всяка под-директория на “компоненти” определя името на компонент и съдържа NixOS модул(и). Всяка под-директория на “конфигурации” определя името на системна конфигурация, и съдържа такава.

Във всяка конфигурация трябва да се определят компоненти по избор *(формата за това е по избор)*. Освен това, трябва да има класове/видове на машини, които вмъкват множество компоненти (примерно: клас “лаптоп”, клас “сървър”).

Някои опции трябва да се задават автоматично (примерно: `hostname` спрямо името на конфигурацията) и други трябва да идват с всички конфигурации (примерно: `system.stateVersion`).

II тема – Менажиране на HOME директорията

Чрез Nix описваме и контролираме файлове в HOME директорията на определения потребител.

Съдържанието на файл може да се дефинира като път или като низ. Самите файлове трябва да бъдат символични връзки към read-only вариантите им, запазени в nix-store.

Някои файлове предвиждат допълнителна обработка (примерно: ако е определен вид скрипт, да се добави неговото изпълнение в `.bashrc`). Трябва да се направят поне 3 уникални такива обработки, на желани програми (конфигурационни файлове).

III тема – Автоматична генерация на пакети

Да се реализира *метод*, чрез който може да се приеме **само** източникът на даден проект и се връща низ, който е пакет за източника.

Примерно, приема само URL и хеш, и връща атрибутно множество, което съдържа името на пакета, версия, зависимости, ... Не се очаква този уред да бъде перфектен, идеята е повече да произведе примерен пакет, от който разработчика може да почне.

Трябва да поддържа поне 3 програмни езика и няколко различни формата изход: като flake, като самостоятелен файл пакет (който да се извиква с `callPackage`), като nix-2 пакет (тоест с `import <nixpkgs>`), ...

IV тема – Модуляризирани flake-ове

Да се реализира система за удобно разбиване на flake-ове. Нека всяко такова парче наричаме компонент.

Всеки компонент е във формата на функция (или атрибутно множество, ако не са нужни аргументи), която връща атрибутно множество. Входовете наподобяват тези на `outputs`, където обаче:

- са премахнати системни под-атрибути (`a."x86_64-linux".b` става `a.b`)
- self реферира към самия компонент
- съдържа и атрибут `config`, който реферира към целия `outputs` (тоест можем да достъпим други компоненти)

Допълнително, компонент може да връща атрибут `inputs`, което е списък с други компоненти (или пътища към компоненти). Те трябва да се включват.

Накрая, всички компоненти трябва да се обединят в един кохерентен `outputs`. Ако няколко компонента се опитат да презапишат стойност, трябва да се хвърли грешка, освен ако не е посочено да се използва единия от тях по специален начин.

V тема – NixOS pipelines

Начин, по който дефинираме задачи и тяхната последователност на изпълнение.

В началото описваме вход по който да работим (GitHub хранилище, ...). След това дефинираме задачите, като за всяка задача пишем:

- име
- скрипт (низ или път), който описва всички стъпки по време на изпълнение на задачата
- артефакт (ако има такъв), който да се предаде на следващата стъпка и **само** той да е наличен на следваща стъпка
- зависими задачи, тоест списък с имената на задачи които трябва да се изпълнят преди текущата

Резултатът трябва да бъде деривация, чието пускане ще изпълни целия pipeline в коректния ред на задачи.

Раздел 2

Въпроси?