Разработка на flake-ове чрез flake-parts

Универсални конфигурации с Nix

Павел Атанасов Камен Младенов

27.05.2025

Преговор

- Писахме flake-ове съдържащи:
  - Пакети
  - NixOS конфигурации
  - Наименувани модули
  - Наименувани overlay функции

Раздел 1

flake-parts

Какво е flake-parts?

- `flake-parts` е библиотека, която ни дава начин да използваме (вече добре познатата ни) NixOS модулна система в контекста на flake-ове
- Сега ще разгледаме как можем да експлоатираме тази библиотека, че да опростим големите си flake-ове

Базов Пример

```
{
  inputs = {
    nixpkgs = {
      url = "github:NixOS/nixpkgs/nixos-unstable";
    };
    flake-parts = {
      url = "github:hercules-ci/flake-parts";
    };
  };

  # outputs...
}
```

:::

```
{
  # inputs...

  outputs = inputs: inputs.flake-parts.lib.mkFlake { inherit inputs; } (
    { config, inputs, self, ... }: {
      systems = [ "x86_64-linux" "aarch64-linux" ];
      imports = [ ];

      flake = { top = "level"; };
      perSystem = { config, inputs', self', pkgs, ... }: {
        packages.hello = pkgs.hello;

        devShells.default = pkgs.mkShell { buildInputs = [ pkgs.hello ]; };
      };
    });
}
```

До какво се оценява това?

```
{
  # ...
  devShells = {
    aarch64-linux.default = «derivation /nix/store/...-nix-shell.drv»;
    x86_64-linux.default = «derivation /nix/store/...-nix-shell.drv»;
  };
  # ...
  packages = {
    aarch64-linux.hello = «derivation /nix/store/...-hello-2.12.1.drv»;
    x86_64-linux.hello = «derivation /nix/store/...-hello-2.12.1.drv»;
  };
  # ...
  top = "level";
}
```

Какво се случи тук?

- Стойността на `outputs` се определя изцяло от резултата на извикването на `mkFlake` от `flake-parts`, на която ѝ подаваме `inputs`
- Оттам то автомагически си извлича `nixpkgs`, от което си създава канонична `pkgs` инстанция (може да се конфигурира с `_module.args`, my beloved)

- И финално, ние му подаваме атрибутното множество*, което дефинира няколко атрибута (има и още поддържани), а иммено:
  - `flake` дефинира *`toplevel` неща*, които директно ще се появят във финалния резултат
  - `perSystem` дефинира `system`-зависими неща като `packages`, `devShells`, т.н.

`mkFlake` и аргументите му

- `config` - подобно на `NixOS` модулите, *финалната* версия на отварата в казана след обединяване на всички модули
- `self` - нещо *като* `config`, ама е как ни изглежда `flake`-а отвън (без `flake` и `perSystem` абстракции)

`perSystem` и аргументите му

- `config` - като глобалното `config`, ама е само за `perSystem` нещата, за текущия `system`
- `inputs'` - като глобалното `inputs`, но `system`-зависимите неща са *смачкани* (няма `${system}` атрибут, през който да минаваме)
- `self'` - като глобалното `self`, но нещата, които са дошли от `perSystem` са (подобно на `inputs'`) *смачкани*
- `pkgs` - както предполагаме, това е инстанциран `nixpkgs` с *правилния* `system`
- ...

Раздел 2

Въпроси?