Конкетики в модулната система на NixOS

Универсални конфигурации с Nix

Павел Атанасов Камен Младенов

08.05.2025

Преговор

- Какво е нужно за операционна система, базирана на Nix
- Запознахме се с модули и опции

- Модул е функция, която приема и връща атрибутно множество
- Приетото има 2 *нужни* атрибута
  - `config` - **всички константни** стойности на атрибути в системата
  - `pkgs` - пакетите в системата
- Върнатото има 3 задължителни атрибута
  - `imports` - списък с пътища към други модули
  - `options` - имена и типове на позволените стойности
  - `config` - атрибути и стойности в системата

Примерен модул

```
{ config, pkgs, ... }: {
  options = {
    includeCurl = pkgs.lib.mkOption {
      type = pkgs.lib.types.bool;
      description = "Should curl be added as a global package";
    };
  };
  config = {
    environment.systemPackages = if config.includeCurl
      then [ pkgs.curl ]
      else [];
  };
}
```

- За момента всичко са примерни модули
- Сега ще разгледаме истински възможни стойности
- Ще създаваме NixOS конфигурация!

Раздел 1

Как ще тестваме NixOS конфигурация?

Как ще тестваме NixOS конфигурация?

- Nix предоставя лесен начин да създаваме виртуални машини, спрямо подадени конфигурации
- Достатъчно е да изпълним следната команда:

```
nix-build '<nixpkgs/nixos>'
    -A vm
    -I nixpkgs=channel:nixos-24.11
    -I nixos-config=./configuration.nix
```

- **Не е нужно да конфигурираме хардуера!**

Раздел 2

Елементарен `configuration.nix`

Елементарен `configuration.nix`

```
{ config, pkgs, ... }:
{
  # За създаване на виртуална машина, това не е нужно
  # imports =  [ ./hardware-configuration.nix ];

  boot.loader.systemd-boot.enable = true;
  boot.loader.efi.canTouchEfiVariables = true;

  system.stateVersion = "24.11";
}
```

Какво значи всеки ред

- `boot.loader.systemd-boot.enable = true;`

  Използваме `systemd-boot` като bootloader

- `boot.loader.efi.canTouchEfiVariables = true;`

  Добавя този bootloader като опция в BIOS-а

- `system.stateVersion = "24.11";`

  Първата NixOS версия от която създаваме конфигурацията. Използва се за поддържане на съвместимост. **Никога** не трябва да се променя!

Да пуснем виртуална машина!

1. Записваме предходния модул в `configuration.nix` израз
2. Пускаме командата
3. Пускаме `./result/bin/run-nixos-vm`

Потребители

- Проблем! Как да се логнем?
- Трябва да добавим в конфигурацията си потребител:

```
users.users.ПОТРЕБИТЕЛСКО_ИМЕ = {
  isNormalUser = true;          # Дали е истински потребител
  extraGroups = [ "wheel" ];    # Дали има административни права
  initialPassword = "test";     # Парола
};
```

- Нека за целта на слайдовете да го кръстим `john`

- Как да сменим паролата? Не искаме всички да я виждат наяве...

1. `users.mutableUsers = true;` (което е стойността по подразбиране)

   Можем да използваме обикновенни команди като `passwd`.
   Nix **няма** да менажира паролите!

2. `john.hashedPassword = "...";`

   Създаваме хеш на парола като пуснем `mkpasswd` командата. Този хеш добавяме в конфигурацията:

   ```
   users.users.john.hashedPassword = "$y$j9T...";
   ```

- Ако пробваме двата варианта един след друг, втория няма да работи (след като наново сме пуснали `nix-build`)
- За паролата Nix трябва да генерира файл във файловата система на виртуалната машина
- Тази файлова система се намира в `./nixos.qcow2`
- Обаче, `nix-build` не бърника в `nixos.qcow2`, когато съществува
- Трябва да го изтрием и след това да пуснем `nix-build`

Раздел 3

Развиване на конфигурацията

Добавяне на пакети

- Нека да добавим командовото приложение `fastfetch`. Има две места на които можем да го вмъкнем:
  1. `users.users.john.packages`

     В сички програми са налични само за потребителя
  2. `environment.systemPackages`

     В сички програми са налични глобално, за всички потребители

Добавяне на графична среда

- Графичните среди изискват много повече неща, за да се подкарат
- *Включваме* ги чрез специфични атрибути:
  - `services.xserver.desktopManager.gnome` за Gnome
  - `services.xserver.desktopManager.cinnamon` за Cinnamon
  - `services.desktopManager.plasma6` за Plasma desktop (KDE)
    (трябват още някои атрибути, подробности тук)
  - ...

- Ние ще използваме Mate, защото е много леко на виртуалната машина:

```
services.xserver.desktopManager.mate.enable = true;
```

- Но можем и да увеличим хардуерните ограничения на тази машина:

```
virtualisation.vmVariant.virtualisation = {
  memorySize = 2048; # В мегабайти
  cores = 2;
};
```

- Включвайки машината пак ни вкарва в терминала (дори след регенериране на `nixos.qcow2`)
- Тези неща се наричат “desktop environment”: предоставят графичната среда и програми с нея
- Обаче в една система може да имаме няколко графични среди. Затова отделно се правят менижиращи “display manager” програми, които ни логват и ни позволяват да си изберем среда.
- Най-популярния от тези е LightDM. Той е включен по подразбиране, само трябва да добавим:

```
services.xserver.enable = true;
```

- Нека да добавим и две от тях - `firefox` (уеб браузър) и `copyq` (история на копиранията)

```
users.users.john.packages = with pkgs; [
  firefox copyq
];
```

Български език

- Българо-говорящи сме! Трябва ни кирилица!
- По принцип се използва програмата `setxkbmap`
- Има Nix атрибути над нея:

```
services.xserver.xkb = {
  layout = "us,bg";                  # Английска и българска клавиатура
  variant = ",phonetic";             # QWERTY и ЯВЕРТЪ варианти
  options = "grp:alt_shift_toggle"; # Смяна на език с Alt+Shift
};
```

Раздел 4

Писане на модули

Разбиване на конфигурацията

- В момента нашата конфигурация е малка, но с времето ще расте. По-добре е от рано да я разделим на няколко модула.
- Нека да преместим:
  - `boot.loader` атрибутното множество в модул `boot.nix`
  - `users.uses.john` в модул `john.nix`
  - `services.xserver` в модул `graphics_environment.nix`
- В нашия `configuration.nix` ще "извикаме" тези модули:

```
import = [ ./boot.nix ./john.nix ./graphics_environment.nix ];
```

Модул с нов атрибут

- Нека да реализираме модул, който добавя опцията `hello`
- Ако тя е `true`, добавяме в `environment.systemPackages` програмата `hello`, иначе не я добавяме

Решение

```
{ config, pkgs, ... }:
let
  lib = pkgs.lib;
in {
  options = with lib; {
    hello = mkOption {
      type = types.bool;
    };
  };
  config = {
    environment.systemPackages =
      if config.hello then [ pkgs.hello ]
      else [];
  };
}
```

Модул с демо потребител

- Нека да направим модул, който добавя опциите `addDemoUser` и `demoUserName` Когато `addDemoUser` е истина, добавяме потребител на име `demoUserName`, който не е администратор и има парола “demo”.
- `addDemoUser` е булева и е `false` по подразбиране. `demoUserName` е низова и е `"demo"` по подразбиране.

Решение

```
{ config, pkgs, ... }:
let
  lib = pkgs.lib;
in {
  options = with lib; {
    addDemoUser = mkOption { type = types.bool; default = false; };
    demoUserName = mkOption { type = types.string; default = "demo"; };
  };
  config = {
    users.users.${config.demoUserName} = lib.mkIf config.addDemoUser {
      isNormalUser = true;
      password = "demo";
    };
  };
}
```

Раздел 5

Сървиси и systemd

Какво е сървис?

- Това е програма, която се пуска на заден план
- Може да се пуска със системата, по време на някакво събитие (компютъра се изключва примерно), може да се пуска ръчно
- Писането и менажирането на сървиси е сложен проблем сам по себе си
- Създадени са специални програми за тази цел
- Най-мощната и използвана се нарича systemd

Бърз увод в systemd сървиси

- Всеки сървис (по-принцип) се дефинира в един текстов файл
- Този текстов файл съдържа
  - секции, нова секция започва след `[ИМЕ НА СЕКЦИЯ]` ред
  - полета със записани стойности във формата `ИМЕ=СТОЙНОСТ`
  - празни редове и коментари (едноредови, започващи с `#`)
- Важните секции са
  - `[Unit]`, която описва самия сървис и зависимости (трябва да се пусне преди/след друг сървис)
  - `[Service]` - какво се изпълнява
  - `[Install]` - кога сървиса се изпълнява

Пример

```
[Unit]
After=network.target
Before=nextcloud.service

[Service]
ExecStart=/usr/local/apache2/bin/httpd -D FOREGROUND -k start
ExecReload=/usr/local/apache2/bin/httpd -k graceful
Type=notify
Restart=always

[Install]
WantedBy=default.target
RequiredBy=network.target
```

Бърз увод в systemctl

- Командата чрез която *комуникираме* с `systemd` се нарича `systemctl`
  - `systemctl status SERVICE` - общ статус на сървиса: пуснат ли е, къде се намира, лог, ...
  - `systemctl start SERVICE` - стартиране на сървис
  - `systemctl stop SERVICE` - спиране на сървис
  - `systemctl restart SERVICE` - рестартиране на сървис

Писане на сървиси през Nix

- Подобно на `setxkbmap`, можем да пишем systemd сървиси чрез Nix изрази:

```
systemd.services = {
  lxqt-policykit = {
    description = "lxqt-policykit runner";
    after = [ "graphical-session.target" ];

    serviceConfig = {
      Type = "simple";
      ExecStart = "${pkgs.lxqt.lxqt-policykit}/bin/lxqt-policykit-agent";
      Restart = "on-failure";
      RestartSec = 1;
    };
  };
```

Раздел 6

Въпроси?