Лекция 1) Проблемите със стандартното менижиране на пакети в Linux-базирани операционни системи. История на Nix. Идеи и концепции на Nix.

1. Защо - Проблеми

2. Кога - История

- Кога е създаден Nix, защо, от кого
- сейм за NixOS
- nixpkgs, **the** monorepo (total github death?)

3. Какво - Идеи и концепции

4. Как - въведение в езика отгоре-отгоре

5. Инсталация на `Nix`

1. Standard

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

2. Determinate System's Installer

Differences from standard installer

```
curl --proto '=https' --tlsv1.2 -sSf -L https://install.determinate.systems/nix | sh -s -- i
```

Лекция 2) Nix - езикът

Типове данни, конструкции, оператори, интерполация

- Какво е декларативно/функционално програмиране, отгоре-отгоре, примери с Nix, може би споменаване на други функционални езици/езици с функционални елементи в тях (ламбди, immutable data)
- Синтаксис и *Граматика*, отново, сравнение с други езици (`attrset - (hash)map`, `list - array`)
- `builtins`, `nixpkgs.lib`, https://noogle.dev

Лекция 3) Nix - пакетният мениджър

- `/nix/store`
- Съществени командни опции и техните функционалности
- Интерпретиране и "компилиране" на Nix програми
- Съвместимост с Linux дистрибуции, MacOS, Windows (WSL), Android

NOTE: Може би да споменем за channels vs flakes отгоре отгоре, че ги има, ма ще ползваме flakes, или е твърде рано?

1. Деривации

концептуално, без да навлизаме в какво и как е `builtins.derivation`, концептуално как се правят `/nix/store/aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-package-version.drv`, безплатен caching, сигурност за reproducibility

2. Примерни (прости) пакети

нещо тривиално, като

```
{ writeShellApplication
, curl
, ...
}:

writeShellApplication {
  name = "weather";
  runtimeInputs = [ curl ];
  text = ''
    curl -s "wttr.in/sofia"
  '';
}
```

3. Примери за екосистеми с интеграция за пакетиране с Nix (`rust`, `haskell`, etc.)

Лекция 4) Създаване/дефиниране на (сложни) пакети

- Какво правят в `nixpkgs`, структура на един *сложен* пакет (`mkDerivation` и Сие)
- Специфики при програми написани на C++, Python, …

1. `devShells`

- като начин за “дебъгване” на пакети
- като начин за влизане в generic shell с development tools в `${PATH}`-а (и още)

Лекция 5) Nix(OS) - операционната система. Обща структура, разлики със стандартни Linux-базирани ОС.

1. Що е NixOS, защо е, защо не е, сравнение с други (FHS-enabled) дистрота

(този момент когато)

```
$ ls /bin /usr/bin
/bin:
sh

/usr/bin:
env
```

2. Системна конфигурация чрез Nix: структура, наименования, reproduceability, модули

- `configuration.nix` as the source of all truth
- разцепване на неща в модули

1. Модули

1. Аргументите `config`, `pkgs`, `lib`

- Overlays (for modifying `pkgs`)
- `_module.args` (for modifying everything)
- `specialArgs` (for adding extra arguments, e.g. flake `inputs`)

2. Атрибутите `imports`, `options`, `config` (+ имплицитността му)

3. Какво са `options`, как ги *декларираме*, как ги *дефинираме*, как merge-ваме *дефиниции*

1. Инсталация

Линк

Лекция 6) Nix flakes - концепция, употреба в проекти, употреба в NixOS

1. nix (v3) experimental-options (`nix-command`, `flakes`)

Default from `DeterminateSystems's Installer`

2. Защо са готини, защо channel-ите не ги тачим твърде много

Reproducibility, lock files are cool

3. Структура

Линк

- pure function, `inputs -> outputs`, `flake.lock`, `nix flake _` commands
- Какво може да връщаме (от всичко по много) - `nixosConfigurations`, `packages`, `devShells`, …

4. `flake-parts`

`NixOS` modules system става и не за `NixOS` модули ?? pog Отдолу е просто купчина `Nix` код, нищо специално Също (преди това) може да си hand-roll-нем `forEachSystem` type beat нещо

Лекция 7) DevOps чрез Nix

1. `disko` && `nixos-anywhere`

Деклариране на дискови конфигурации през Nix, headless инсталиране на `NixOS` на отдалечени машини (само с работещ `sshd` на каквото и да е буутнато дистро (може би ще спомена за `kexec`-а, нз колко е релевантно))

2. terranix

Линк

С това не сме бачкали още, та ще трябва по документация да вадим неща, трябва да се почете повече, че да измислим какво да {,по}кажем

3. `process-compose`

process-compose process-compose-flake services-flake

Декларативно и системно-агностично дефиниране на нужните *service*-и, за спомагане локалната разработка на проект (бази, … кво още реално?)

С това аз (Павел) съм леко запознат, но не съм си играл достатъчно, пак (както за `terranix`) трябва да се чете от документация, няма да отказваме примери от вас :Д

4. `nixos-rebuild build-vm`

За локално тествене на конфигурации, трябва да спомена за `virtualisation.vmVariant`

nixpkgs's `build-vm.nix`

```
{
  virtualisation.vmVariant = {
    # following configuration is added only when building VM with build-vm
    virtualisation = {
      memorySize = 2048; # in MiB
      cores = 3;
    };

    #                    NixOS
    #                 override-          user-
  };
}
```

5. Secrets management

Две големи опции:

- agenix (с agenix-rekey е много яко) По-базираното IMO, заключва поотделно всеки secret във `.age` файл с публичните ключове на всеки host, който реферира ключа (с вариращо ниво на автоматизация, и пак, `agenix-rekey` е много готино тука, може да навлезнем в детайли)
- sops-nix Всичко е в един дебел `.yaml` файл, in-place криптирано

Име бая още други, добре са документирани в Comparison of secret managing schemes, ама тва са 2-те най-ползвани

Лекция 8) Other complementary tools

1. `home-manager`

Като `NixOS` конфигурация (с модули), но с цел конфигурарине на програми, не толкова на service-и. Бачка и под други системи, като generix Linux, Android (под `nix-on-droid`), Darwin (макбоклуци)

2. caches, remote builders

Няма какво толкова да се каже, освен, че `Nix` позволява специфициране на други машини, от които да: 1. тегли вече-компилирани пакети - `cache`-ове 2. праща `.drv`-тата за билдване - `(remote) builder`-и

3. Editor tools (`LSP`s, etc.)

- Не е най-развитото нещо, има няколко `LSP` сървъра, сега май най-готин е nil, може да допълва `NixOS` module опции (IIRC засега само от `nixpkgs`, трябва да се почете)
- shameless plug) Аз (Павел) преди N време накодих ендо плъгинче за `neovim`, което помага при update-ването на `hash`-овете на разни `fetch`-ове след промяна на source-овете им (github repo, revision, etc.) - ей го