VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SPRÁVA RASPBERRY PI 4 CLUSTERU POMOCÍ NIX

RASPBERRY PI 4 CLUSTER MANAGEMENT IN NIX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ADAM ŽIVČÁK**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Živčák Adam, Bc.**
Program: Informační technologie
Obor: Informační systémy
Název: **Správa Raspberry Pi 4 clusteru pomocí Nix**
Raspberry Pi 4 Cluster Management in Nix
Kategorie: Operační systémy
Zadání:

1. Seznamte se s jazykem a systémem správy software Nix, operačním systémem NixOS (jádro linux) a s repozitářem NixPkgs. Prozkoumejte podporu Raspberry v NixOS a možnosti konfigurace clusteru pro distribuované výpočtu v Nix.
2. Navrhněte způsob nasazení, centrální správy a sledování Raspberry Pi 4 clusteru pomocí technologií Nix za účelem distribuovaných výpočtů. Navrhněte také víceuživatelskou webovou aplikaci, která takovou správu usnadní.
3. Po konzultaci s vedoucím řešení implementujte v Nix a NixOS. Řešení otestujte na ukázkových úlohách distribuovaných výpočtů (Hadoop, Spark, OpenMPI, atp.) a změřte výkon clusteru. Vytvořte také podpůrnou webovou aplikaci.
4. Výsledek popište, vyhodnoťte a zveřejněte jako open-source.

Literatura:

- Bruno Bzeznik, Oliver Henriot, Valentin Reis, Olivier Richard, a Laure Tavard. Nix as HPC package management system. In Proceedings of the Fourth International Workshop on HPC User Support Tools (HUST'17). Association for Computing Machinery, New York, NY, USA, 2017. Dostupné z: [https://doi.org/10.1145/3152493.3152556]
- Eelco Dolstra a Andres Löh. "NixOS: A purely functional Linux distribution." ACM Sigplan Notices. Vol. 43. No. 9. ACM, 2008. Dostupné z: [https://doi.org/10.1145/1411204.1411255]
- NixOS Manual. *NixOS* [online]. 2020 [cit. 2020-10-26]. Dostupné z: [https://nixos.org/manual/nixos/stable/]
- René Peinl, Florian Holzschuher a Florian Pfitzer. "Docker Cluster Management for the Cloud - Survey Results and Own Solution". J Grid Computing 14, s. 265-282, 2016. Dostupné z: [https://doi.org/10.1007/s10723-016-9366-y]
- Raspberry Pi 4 support: Issue #63720. *NixOS/nixpkgs* [online]. 2020 [cit. 2020-10-26]. Dostupné z: [https://github.com/NixOS/nixpkgs/issues/63720]

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2 a rozpracovaný bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 27. října 2020

Abstrakt

Náplňou tejto diplomovej práce je návrh a implementácia systému pre nasadenie, správu a monitoring Raspberry Pi klastra pomocou technológií Nix. Práca popisuje výhody funkcionálneho prístupu systému Nix a podsystémov, ktoré z neho vychádzajú. Výsledkom práce je tiež podporná webová aplikácia, poskytujúca intuitívne prostredie pre prácu s nasadeniami konfigurácii klastra a prehľadné zobrazovanie informácii o vyťažení jednotlivých uzlov s využitím dashboardov. Záverečná časť práce sa venuje testovaniu výkonu klastra pomocou ukážkových úloh distribuovaných výpočtov.

Abstract

The scope of this thesis is to design and implement a system for deploying, managing and monitoring a Raspberry Pi cluster using Nix technologies. The thesis describes the benefits of the functional approach of Nix and the subsystems that are based on it. The thesis also results in a supporting web application, providing an intuitive environment for working with cluster configuration deployments and clearly displaying information about the utilization of individual nodes using dashboards. The final part of the thesis is devoted to testing cluster performance using sample distributed computing jobs.

Kľúčové slová

Nix, NixOS, NixOps, správa klastru, monitoring klastru, distribuované výpočty

Keywords

Nix, NixOS, NixOps, cluster management, cluster monitoring, distributed computing

Citácia

ŽIVČÁK, Adam. *Správa Raspberry Pi 4 clusteru pomocí Nix*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Správa Raspberry Pi 4 clusteru pomocí Nix

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána RNDr. Mareka Rychlého, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Adam Živčák
19. mája 2021

Poďakovanie

Ďakujem Všemohúcemu Bohu za všetky milosti a silu, ktorou ma počas celého štúdia napĺňal. Moja veľká vďaka patrí vedúcemu tejto práce, pánovi RNDr. Marekovi Rychlému, Ph.D., za cenné rady a ústretový prístup pri tvorbe tejto práce. Ďakujem tiež mojej snúbenici za trpezlivosť a podporu v tých najťažších chvíľach a taktiež mojej rodine a blízkym, ktorí pri mne stáli a podporovali ma.

Obsah

Kapitola 1

Úvod

Nároky na výpočtový výkon kladené výskumnými skupinami, inštitúciami, alebo priemyselnými organizáciami stále narastajú. Použitie klastrov pre distribuované výpočty dokáže splniť tieto požiadavky, ale vznikajú pri tom ďalšie problémy, ktoré je treba vyriešiť. Vyvíja sa množstvo aplikácií podľa potrieb používateľov, a zároveň sa výrobcovia hardvéru snažia prichádzať s novými špecializovanými technológiami. Životný cyklus aplikácií je zvyčajne dlhší ako životnosť hardvéru, na ktorom tieto aplikácie bežia. Udržiavanie systémov v aktuálnom stave a poskytovanie stabilného prostredia pre používateľov môže byť taktiež náročné [1]. Častokrát si to vyžaduje nasadzovanie starších knižníc a závislostí, ktoré sú s nimi kompatibilné. Špecifické úlohy si vyžadujú špecifický, neraz proprietárny softvér, ktorý nie je súčasťou štandardných balíčkovacích systémov.

Distribuované výpočty umožňujú rozdeliť riešený problém na niekoľko menších paralelných úloh, ktoré môžu byť riešené jedným, alebo viacerými počítačmi v jednej sieti. Distribuované výpočty môžu fungovať aj na globálnej úrovni, kedy sa do výpočtov veľmi náročných úloh môžu zapojiť počítače naprieč celým svetom. Príkladom takého projektu je Folding@home[1], ktorý sa zaoberá simuláciami proteínov, a v súčasnosti (rok 2021) aj skúmaním štruktúr vírusu SARS-CoV-2.

Raspberry Pi 4 je malý počítač vyvinutý britskou spoločnosťou Raspberry Pi Foundation, ktorej pôvodným zámerom bolo rozšíriť záujem ľudí na celom svete o informačné technológie [17]. Klaster zostavený z týchto počítačov je vhodným prostredím pre vývoj aplikácii a testovanie systémov, ktoré môžu byť následne nasadené na výkonnejšie zariadenia.

Prvá časť tejto práce popisuje princípy funkcionálneho systému Nix pre správu balíčkov a ďalšie systémy, ktoré z neho vychádzajú. Ich výhody spočívajú v dobrej reprodukovateľnosti riešení a odstraňujú nevýhody imperatívnych prístupov, ktoré sú popísané v kapitole 2. V ďalšej kapitole je predstavený návrh systému pre správu klastra a koncept podpornej webovej aplikácie pre tento systém. Kapitola 4 obsahuje popis implementácie navrhnutého systému a spôsob používania webovej aplikácie. Záverečná časť je venovaná testovaniu a meraniu výkonu klastra.

[1] https://foldingathome.org

Kapitola 2

Prehľad technológií Nix

V tejto kapitole bude poukázané na nevýhody imperatívneho prístupu, predstavený čiste funkcionálny balíčkovací systém Nix, ktorý využíva deklaratívny prístup, a ďalšie podsystémy vychádzajúce z neho, akými sú operačný systém NixOS, repozitár balíčkov NixPkgs alebo nástroj pre nasadzovanie a správu systémov NixOps.

2.1 Imperatívne balíčkovacie systémy

Vo väčšine dnešných balíčkovacích systémov je použitý imperatívny model. Zmeny, ktoré sú vykonávané, sú stavové a deštruktívne. Aktualizáciou softvéru dochádza k nahradeniu aktuálne využívaných, zastaraných súborov a konfigurácií, bez možnosti návratu k pôvodnej verzii. V prípade, že nejaká časť nového systému nie je spätne kompatibilná, môže dochádzať k poškodeniu komponentov, ktoré boli na danej verzii závislé. Ďalší problém vzniká potrebou dvoch rôznych verzií balíčka. Ak sú dva balíčky závislé na navzájom konfliktných verziách softvérového komponentu, je potrebné pracne vytvoriť samostatné lokality pre jednotlivé verzie a závislosti ručne nastaviť. Rovnaký problém môže nastať s konfiguračnými riadiacimi skriptami.

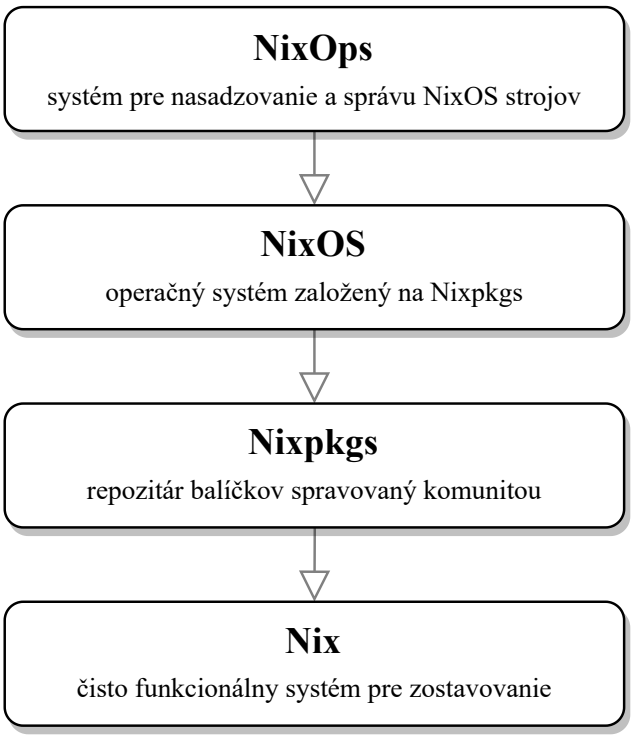
Preklad zdrojových súborov a vytváranie nových balíčkov (tzv. *build*) je taktiež stavový. Zvyčajne je pre zostavenie balíčka potrebný súbor so špecifikáciou. Ten popisuje akcie, ktoré sa musia vykonať, a ďalšie informácie, akými sú napríklad závislosti. Problémom pri zostavovaní týchto súborov môže byť ich úplnosť. Systém, na ktorom sa balíček zostavuje, už môže mať nainštalované niektoré z nevyhnutných závislostí, takže sa balíček podarí zostaviť, aj keď nebola závislosť správne definovaná. Avšak pri prenesení na iný systém, ktorý túto závislosť nebude obsahovať, sa zostavenie nepodarí a administrátor sa pravdepodobne ani nedozvie, z akého dôvodu. Špecifikácia závislosti je naviac nominálna, teda vyjadrená len názvom. V prípade, že zostavovaný balíček vychádza z určitej verzie komponenty, môže hostiteľský systém obsahovať rovnakú komponentu nižšej verzie. V tom prípade je špecifikácia splnená, avšak zostavenie aj napriek tomu zlyhá.

Stavové vykonávanie zmien sa týka aj konfiguračných súborov, ktoré bývajú uložené v adresári `/etc`. Pri prvej inštalácii súboru – keď jeho inštancia v systéme ešte neexistuje – je vytvorený potrebný obsah. Avšak v momente, keď dôjde k aktualizácii, nestačí, aby nový súbor prepísal pôvodný. V práve používanej verzii súboru mohol používateľ vykonať zmeny, ktoré by boli stratené. Existuje niekoľko riešení tohto problému: ignorovanie nového súboru dúfajúc, že starý bude fungovať; prepísanie pôvodného súboru s vytvorením jeho zálohy; pokus o spojenie oboch súborov *merge*. Zlým aspektom je tiež to, že zmeny

v konfiguračných súboroch často nespadajú pod kontrolu balíčkovacích systémov. Za vykonané zmeny zodpovedá administrátor systému, a výsledný konfiguračný súbor je neraz výsledkom viacerých čiastočných úprav. Trasovanie zmien je veľmi obťažne. Samozrejme, bolo by možné zahrnúť konfiguračné súbory pod správu verziovacích nástrojov. To však tiež nevyrieši všetky problémy. Zmeny sú totiž vytvárané tak používateľmi, ako aj samotným systémom, a je zložité všetky zmeny odpozorovať. Naviac, niektoré súbory môžu byť závislé na konkrétnych verziách programov či knižníc a bolo by potrebné verziovať aj tie [3].

2.2 Funkcionálny balíčkovací systém Nix

Nix je čisto funkcionálny balíčkovací systém. To znamená, že balíčky sú vnímané ako hodnoty v čisto funkcionálnych jazykoch – sú zostavované funkciami bez vedľajších efektov a po zostavení sa nemenia [10]. Balíčky sú zostavované podľa výrazov Nix a ukladané v úložisku Nix store, adresovanom s využitím hashov. Repozitár balíčkov systému Nix sa nazýva Nixpkgs. Nad balíčkovacím systémom je postavený operačný systém NixOS, a nad ním systém pre správu strojov s týmto operačným systémom NixOps. Na obrázku 2.1 je znázornená táto hierarchia.

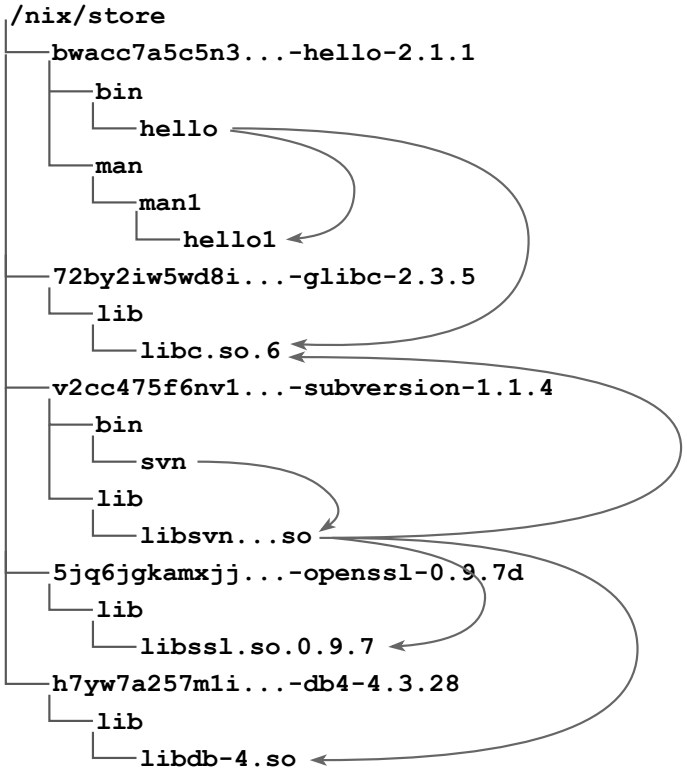


Obr. 2.1: Hierarchia častí ekosystému Nix.

Nix store

Komponenty softvéru, o ktoré sa Nix stará, ukladá do tzv. skladu komponentov, nazývaného tiež *Nix store*. Sklad komponentov je adresár, resp. priečinok, určený špeciálne k tomuto účelu, zvyčajne umiestnený v `/nix/store`. Komponenty sú v tomto adresári ukladané izolovane, čo znamená, že názvy žiadnych dvoch komponentov nie sú rovnaké. Na obrázku 2.2

je znázornená podmnožina úložiska Nix store, obsahujúca niekoľko aplikácií, vrátane ich závislostí. Nix store nie je len množinou súborov. Súbory sú navzájom prepojené v stromovej štruktúre. Jednotlivé prepojenia predstavujú závislosti medzi balíčkami, knižnicami, a inými softvérovými komponentmi.



Obr. 2.2: Znázornenie štruktúry podmnožiny repozitára balíčkov `/nix/store` a závislostí medzi jednotlivými balíčkami. Obrázok bol vytvorený podľa obrázku v dizertačnej práci o systéme Nix [2].

Izolácia (zamedzenie interferencií názvov) jednotlivých balíčkov je zabezpečená pomocou hashovania. Plne definovaný (tzv. *fully-qualified*) názov balíčku sa označuje ako *store path*. Príklad takého názvu je: `v2cc475f6nv1lie47ouf7r5vwi7r4gu-subversion-1.1.4`. Časť názvu `v2cc475f6nv1...` je kryptografický hash, ktorý je vytvorený z názvov všetkých vstupov potrebných pre zostavenie daného balíčka (zdrojové súbory, riadiace skripty zostavenia, argumenty a premenné prostredia, závislosti, prekladač, linker, knižnice a pod.). Systém hashovania zabezpečuje odolnosť voči kolíziám názvov balíčkov, keďže je výpočtovo nemožné nájsť dve vstupné množiny hodnôt, ktoré by poskytovali rovnaký výsledok hahsovacej funkcie. Nix používa 160-bitové hashe reprezentované v 32-bitovej notácii, takže hash, ktorý je súčasťou názvu, je 32 znakov dlhý. Hash je vypočítavaný zo všetkých vstupov zostavovacej funkcie. Teda aj najmenšia zmena na vstupe sa premietne do názvu balíčka. Hash je vypočítavaný rekurzívne. V prípade, že dôjde napríklad k aktualizácii knižnice, budú prepočítané názvy všetkých balíčkov používajúcich túto knižnicu. Je dôležité poznamenať, že zmeny vykonané pri aktualizácii balíčka nie sú deštruktívne. Staršie verzie ostávajú prítomné v systéme, môžu byť aj naďalej využívané inými balíčkami, alebo ostávajú dostupné pre prípad návratu k staršej konfigurácii systému (tzv. *rollback*).

Kryptografické hashe naviac umožňujú úplnú identifikáciu závislostí, čím odstraňujú ďalší nedostatok tradičných balíčkovacích systémov. Zvyčajne komponentom systému nič nebráni v prístupe k iným komponentom počas zostavovania balíčkov, či počas ich behu. V prípade, že sa zostavuje balíček, ktorý pre správne fungovanie vyžaduje nejakú knižnicu, prekladač prehľadá lokality, kde by sa táto knižnica mohla nachádzať, ako napríklad `/usr/lib`. Ak prekladač uspeje, vznikne závislosť, avšak neexistuje mechanizmus zabezpečujúci pridanie popisu tejto závislosti do súboru, ktorý špecifikuje zostavenie balíčka. Využitie hashovania tomu zabraňuje tým, že neukladá komponenty globálne. Ak dôjde k zostavovaniu balíčka a závislosť nebude správne zadefinovaná, zostavenie zlyhá.

S úplnou identifikáciou závislostí komponentov súvisí uzáver nad reláciou závislostí. To znamená, že ak komponent X závisí na komponente Y a ten závisí na komponente Z, musia byť v závislostiach definované a následne nasadené všetky tieto závislosti. Tým vzniká graf závislostí komponentov, z ktorých Nix vypočítava uzávery. Uzáver sa dá zostaviť jednoducho rekurzívnym nasledovaním šípok, ktoré reprezentujú závislosti medzi komponentmi.

Balíčkovací systém Nix

Nix je multiplatformový balíčkovací systém. Je podporovaný v Linuxe i macOS. Môže pracovať v dvoch základných režimoch – režim jedného používateľa (`single-user mode`) alebo režim niekoľkých používateľov (`multi-user mode`). Režim jedného používateľa je podobný klasickým balíčkovacím systémom. Existuje v ňom jeden používateľ (zvyčajne `root`), ktorý spravuje všetky balíčky (od inštalácie, cez aktualizácie, až po odstránenie) a ostatní používatelia môžu tieto balíčky len používať. Zmeny nad Nix store môže vykonávať len vlastník daného priečinka, teda obvykle `root`. Režim viacerých používateľov dovoľuje každému z nich spravovať svoje vlastné balíčky bez potreby špeciálnych privilégií. Nix garantuje, že je tento systém bezpečný a nehrozí riziko zavedenia vírusu z profilu jedného používateľa do iného. Nedovoľuje používateľom zostavovať balíčky svojvoľným spôsobom, ani zasahovať do priebehu iných, práve prebiehajúcich zostavovaní. V tomto režime je vlastníkom úložiska privilegovaný používateľ, rovnako ako v predošlom režime. Naviac existuje mimoriadna skupina používateľov (`nixbld`), ktorá sa vytvorí pri inštalácii Nixu. Bežní používatelia nemôžu patriť do tejto skupiny. V momente, keď spustí používateľ v tomto režime nejakú udalosť, zaznamená to démon služby Nix, ktorý predá vykonanie tejto služby špeciálnemu používateľovi zo skupiny `nixbld`. Tento mechanizmus sa stará o bezpečnosť celého systému a nedovolí, aby zostavovanie balíčka jedného používateľa zasahovalo do zostavovania iného balíčka, pričom štandardný používateľ nad procesom stratí kontrolu, keďže ho riadi démon. Súbežne môže prebiehať toľko zostavovaní, koľko existuje v systéme používateľov v skupine `nixbld`.

Inštaláciu Nixu v Linuxe v režime jedného používateľa je možné spustiť pomocou nasledujúceho príkazu, bez použitia privilégií správcu:

```
$ sh <(curl -L https://nixos.org/nix/install) --no-daemon
```

V prípade potreby správcovských oprávnení si ich skript sám vyžiada. Inštalácia režimu viacerých používateľov sa líši len v použitom prepínači. Namiesto `-no-daemon` je potrebné zadať `-daemon`. Počas inštalácie sa vytvoria účty špeciálnych používateľov pre zostavovanie balíčkov a ich skupina.

Za základné funkcie vykonávané pomocou balíčkovacích systémov možno považovať získavanie balíčkov, ich inštalovanie a aktualizovanie, alebo odstraňovanie. Pohľady používateľov na balíčky nainštalované s využitím systému Nix sa môžu líšiť. Každý používateľ môže mať vlastnú množinu aktívnych balíčkov, aj keď ich je v systéme nainštalovaných omnoho

viac. Umiestnenia aktívnych balíčkov sa pridávajú do premennej `PATH` daného používateľa, a vytvárajú tak jedinečné užívateľské prostredie. V skutočnosti sú to len symbolické odkazy na balíčky, ktoré majú v systéme vždy len jednu inštanciu. Balíčky sú inštalované za pomoci *výrazov Nix*, ktoré určujú, ako sa inštalácia vykoná, prípadne určujú ďalšie závislosti. Súbor týchto výrazov a balíčkov, a teda obdoba repozitára, sa nazýva Nixpkgs.

Nixpkgs je možné stiahnuť ručne z webových stránok Nixu[1]. Výhodnejšie je však využiť tzv. kanál balíčkov Nix (*Nix channel*). Nix kanál je v podstate odkaz na vždy aktuálnu verziu úložiska s balíčkami. Zabezpečuje, že inštalované balíčky sú aktuálne, bez nutnosti manuálnej aktualizácie. Pre používanie kanálu je potrebné prihlásiť sa k jeho odberu, napríklad pomocou príkazu:

```
$ nix-channel --add https://nixos.org/channels/nixpkgs-stable
```

pričom URL adresa sa môže líšiť podľa toho, či chce používateľ najnovšie verzie balíčkov bez ohľadu na ich stabilitu (`nixpkgs-unstable`), najnovšie stabilné balíčky (`nixpks-stable`), alebo balíčky pre konkrétnu verziu operačného systému (`nixos-21.03`), a pod.

Nix umožňuje zisťovať, aké balíčky sú dostupné k inštalácii, aké balíčky sú nainštalované v systéme, a ktoré z nich sú aktívne pre konkrétneho používateľa. Príklad príkazu pre výpis dostupných balíčkov, ktorých názov začína slovom ätom":

```
$ nix-env -qa 'atom*'
```

Inštalácia balíčka sa tiež vykonáva pomocou príkazu `nix-env`. Napríklad nainštalovanie textového editora Atom je možné pomocou príkazu:

```
$ nix-env -i atom
```

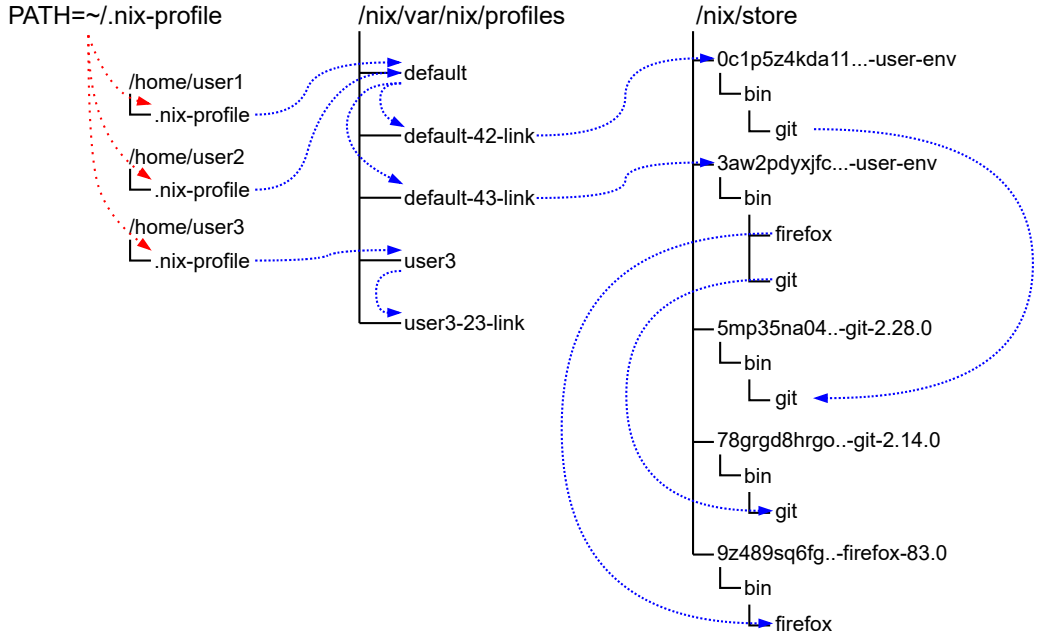
Nix najprv overí dostupnosť predpripraveného balíčku v binárnej cache. V prípade, že cache požadovaný balíček neobsahuje, stiahnu sa zdrojové súbory a balíček sa zostaví lokálne. Odinštalovanie balíčka sa vykoná rovnakým príkazom, s tým rozdielom, že namiesto prepínača `-i` (install) sa použije `-e` (erase). Balíček už viac nebude pre daného používateľa dostupný, v systéme však ostane. Jedným z dôvodov môže byť to, že balíček používa iný používateľ, alebo na ňom závisia iné balíčky. To znamená, že na tento balíček stále existujú symbolické odkazy. Ponechanie v systéme je tiež z dôvodu možného návratu k predchádzajúcej konfigurácii, tzv. *rollback*.

Mechanizmy umožňujúce atomické aktualizácie a rollbacky v Nixe sú profily a používateľské prostredia. Dovoľujú tiež to, aby mohol mať každý používateľ vlastnú konfiguráciu, nezávisle od ostatných používateľov. Ako už bolo uvedené, balíčky sú ukladané v Nix store pomocou jedinečných názvov zostavených z kryptografických hashov. Názvy všetkých balíčkov by sa mohli pridať do premennej prostredia `PATH`. To je však málo používateľsky prijateľné, a preto Nix zavádza symbolické odkazy, ktoré vytvárajú stromovú štruktúru aktívnych balíčkov [10]. Odkazy z používateľského prostredia do Nix store následne vytvárajú *generácie*. Názov je odvodený od princípu, kedy sa po každom použití príkazu `nix-env` generuje nová konfigurácia prostredia, pretože bola vykonaná istá zmena. Skupiny generácii jednotlivých používateľov vytvárajú *profily*. Tie slúžia k tomu, aby nedochádzalo ku vzájomným konfliktom medzi konfiguráciami používateľov. Príklad profilov a generácii je zobrazený na obrázku 2.3. Vytváranie nových generácii je atomické, pretože nové súbory neprepisujú pôvodné, ale existujú súbežne, pričom záleží len na verzii, na ktorú sa odkazuje. K predchádzajúcej generácii je možné vrátiť sa pomocou príkazu

```
$ nix-env --rollback
```

[1] http://nixos.org/nixpkgs/download.html

alebo s využitím prepínača `-switch-generation x`, kde `x` predstavuje číslo generácie. Zoznam existujúcich generácii je možné vypísať príkazom `nix-env -list generations`. Ďalšie príklady práce s generáciami a profilmi sú uvedené v manuáli k Nix [10].



Obr. 2.3: Na obrázku je štruktúra profilov, generácii a balíčkov v Nix. Ľavý stĺpec obsahuje profily jednotlivých používateľov. Každý profil sa odkazuje na nejakú generáciu konfigurácie systému (napr. `default`), čo môže byť symbolický odkaz na konkrétnu generáciu (napr. `default-43-link`). Konfigurácie potom obsahujú odkazy na balíčky, ktoré má daný používateľ aktívne, teda si ich nainštaloval. Obrázok bol vytvorený podľa obrázku z manuálu k systému Nix [10].

Balíčky nie sú po odinštalovaní používateľom okamžite odstránené zo systému. Odstránia sa symbolické odkazy z profilu a vytvorí sa nová generácia, takže z pohľadu používateľa je balíček odinštalovaný a nedokáže ho používať. Balíček ostáva v systéme, pretože naň môžu mať vytvorené odkazy iný používatelia, prípadne na ňom môžu závisieť iné balíčky. Ak aj neexistujú žiadne väzby k balíčku, stále existuje dôvod ponechania v systéme, ktorým je možnosť návratu k predchádzajúcej konfigurácii. Kapacita systému však nie je neobmedzená, a preto Nix zavádza mechanizmus čistenia (tzv. *garbage collection*), ktorý odstraňuje balíčky bez väzieb. Aby bol balíček odstránený mechanizmom čistenia, nesmie existovať žiadna väzba – ani k starým generáciám konfigurácii akéhokoľvek používateľa. Staršie generácie môžu byť odstránené, čím sa zrušia väzby k balíčkom, avšak zároveň definitívne zanikne možnosť návratu k týmto generáciám. Príkaz pre odstránenie starších generácii môže byť napríklad:

```
$ nix-env --delete-generations old
```

Tento príkaz odstráni všetky generácie daného používateľa okrem jednej – práve používanej. Namiesto argumentu `old` možno presne špecifikovať, ktoré generácie majú byť odstránené, alebo definovať časový údaj. Generácie staršie ako definovaný čas budú zmazané. Proces čistenia sa spúšťa príkazom: `nix-env -gc`.

Výrazy Nix

Jazyk výrazov Nix je pomerne jednoduchý, dynamický typovaný, čiste funkcionálny jazyk [3]. Jeho vyjadrovacia sila nie je úplná. Je zostavený pre potreby špecifikácie spôsobu, akým sa majú zostavovať balíčky. Jazyk obsahuje niekoľko základných dátových typov: boolovské výrazy, čísla, reťazce a cesty. Z nich je možné zostaviť zoznamy, množiny atribútov a funkcie. Jazyk tiež obsahuje niekoľko vstavaných funkcii.

Reťazce môžu byť uzatvorené pomocou úvodzoviek, napríklad `"retazec"`, alebo do dvojice apostrofov (`"retazec"`), pričom môžu presahovať cez viacero riadkov. Do reťazcov môžu byť vkladané výsledky výrazov, ktoré sú uvedené pomocou konštrukcie `${ ... }`. Vložené výsledky musí byť možné previesť na reťazec, takže výsledkom musí byť iný reťazec, cesta, alebo derivácia. Výrazy a reťazce môžu byť do seba ľubovoľne vnárané. Ďalším typom reťazcov sú URI odkazy, ktoré môžu byť uvedené pomocou úvodzoviek, ale aj bez nich. Čísla môžu byť celočíselného typu (integer) alebo typu s pohyblivou rádovou čiarkou (float). Výrazy nad číslami rovnakého typu vracajú ako výsledok číslo toho istého typu, ak však výraz obsahuje float i integer, výsledok bude typu float. Ďalším základným dátovým typom sú cesty. Cesty môžu byť relatívne alebo absolútne. Relatívne cesty sa vyhodnocujú voči umiestneniu súboru, v ktorom sú uvedené. Cesty sa zapisujú bez úvodzoviek a musia obsahovať aspoň jednu lomku (`/`) – takže odkaz na súbor v rovnakom priečinku musí byť zapísaný ako `./subor`. Hodnoty typu boolean sa zapisujú ako `true` a `false`.

Zoznamy sú definované pomocou hranatých zátvoriek, medzi ktorými sú položky oddelené medzerami. Zoznamy môžu byť heterogénne. Príklad validného zoznamu v jazyku Nix: `[ [1 2 "text"] true 6.2]`. Veľmi dôležitým typom v Nixe je množina atribútov. Atribút je dvojica názov — hodnota, medzi ktorými je uvedený znak rovnosti a každý atribút je ukončený bodkočiarkou. Atribúty množiny sú uzatvorené v zložených zátvorkách, na ich poradí nezáleží, avšak názov atribútu musí byť v rámci množiny unikátny. Príklad množiny atribútov:

```
{ a = 123.4;
  t = "string";
  p = /home/file;
}
```

K hodnotám atribútov možno pristupovať pomocou operátora bodky. Napríklad výsledok nasledujúceho výrazu bude vyhodnotený ako reťazec `"svet"`:

```
{a="ahoj"; b="svet"}.b
```

Atribúty sa na seba môžu odkazovať aj rekurzívne, ak je množina uvedená kľúčovým slovom `rec`. U rekurzívnych množín vzniká riziko nekonečnej rekurzie, ak napríklad prvok `a` odkazuje na prvok `b`, ktorý odkazuje na prvok `a`. Konštrukcia výrazu `let ...in ...` umožňuje za slovíčkom `let` definovať lokálne premenné pre výraz uvedený za kľúčovým slovom `in`. Pri definovaní množín alebo výrazov `let` môže byť výhodné kopírovať premenné zo susedného menného priestoru, pričom dochádza k tzv. dedeniu atribútov. Dedenie môže byť skrátené pomocou kľúčového slova `inherit`, čo je syntaktický cukor pre zápis s využitím znaku rovnosti. Nasledujúce dva výrazy sú teda ekvivalentné [10].

```
let x = 123;
in {
  inherit x;
  y = 456;
}
```

```
let x = 123;
in {
  x = x;
  y = 456;
}
```

Funkcie v Nix majú tvar: `vzor : telo`. Ak je vo vzore jeden identifikátor, funkcia sa vždy vyhodnotí. Funkcia negácie, ktorá pracuje len s jedným argumentom, môže vyzerať nasledovne: `x : !x`. Tá istá funkcia ale môže vyzerať aj takto: `x: y: x + y`. Táto funkcia spracováva jeden argument a ako výsledok vracia inú funkciu, spracovávajúcu iný argument. Funkcie nemajú meno, jediný spôsob pomenovania je vyrobiť z nich atribút. Druhým spôsobom definovania vzoru je tvar `arg1, arg2, ..., argN`. Príkladom takejto funkcie môže byť: `{ x, y, z }: x + y + z`. Táto funkcia musí byť zavolaná presne s troma argumentami, inak nebude vyhodnotená.

2.3 NixOS

NixOS je Linuxová distribúcia operačného systému postavená na čisto funkcionálnom balíčkovacom systéme Nix. NixOS používa Nix nie len na zostavovanie balíčkov, ale aj ďalších statických častí systému. Keďže sa konfiguračné súbory Unixových systémov zvyčajne nemenia za behu, môžu byť zostavované s využitím derivácii, ako je tomu pri zostavovaní balíčkov.

Vďaka odloženému vyhodnocovaniu Nix výrazov (lazy evaluation) sa zostavujú len nevyhnutné systémové komponenty a moduly. To je výhodné hlavne v prípade zostavovania veľkých modulov, akými sú napríklad XServer, či desktopové prostredia (KDE).

Inštalácia systému NixOS je v princípe podobná inštalácii inej linuxovej distribúcie. Rozdiel je ale v tom, že NixOS neposkytuje grafické používateľské rozhranie pre inštaláciu. Systém po nabootovaní z inštalačného média nabehne to tzv. live verzie operačného systému, kde je potrebné vykonať úkony v príkazovom riadku. Najprv je potrebné vytvoriť na cieľovom disku zväzky, naformátovať ich a namapovať. Následne je potrebné vytvoriť konfiguračný súbor systému, ktorý je popísaný nižšie. Predvolenú konfiguráciu je možné vytvoriť pomocou príkazu `nixos-generate config`. Konfiguráciu je možné po automatickom vygenerovaní upraviť tak, aby spĺňala všetky požiadavky na systém. Po vykonaní všetkých zmien sa spustí inštalácia príkazom `nixos-install`. V prípade chýb v konfigurácii nebude inštalácia úspešná a bude potrebné opraviť chyby, na ktoré systém upozorní. Ak je konfigurácia správna, systém sa nainštaluje a pred dokončením požiada o vyplnenie nového administrátorského hesla. Po dokončení je nutné systém reštartovať. Detailné informácie k procesu inštalácie NixOS sú v manuáli dostupnom na webových stránkach [11] alebo v live systéme po zadaní príkazu `nixos-help`.

Konfigurácia systému je popísaná pomocou Nix výrazov v súbore `configuration.nix` ktorý je umiestnený v priečinku `/etc/nixos/`. Tento súbor obsahuje množiny atribútov, ktoré sú do seba rôzne vnorené a tvoria tak špecifikáciu celého systému. Príklad jednoduchého konfiguračného súboru je vo výpise 2.1. Možno tu vidieť nastavenie zavádzača operačného systému, moduly systémového jadra, vytvorenie a mapovanie súborového systému na disku, nastavenie sieťového rozhrania a povolenie služby SSH, vytvorenie používateľa `nixosuser` a výpis balíčkov, ktoré majú byť do systému nainštalované.

```
{ config, pkgs, ... }:

{
  boot.loader.grub.enable = true;
  boot.loader.grub.version = 2;
  boot.kernelModules = [ "kvm-intel" ];

  fileSystems."/" = {
    device = "/dev/disk/by-name/sda";
    fsType = "ext4";
  };

  networking.interfaces.eth0.useDHCP = true;

  services.openssh.enable = true;

  users.users.nixosuser = {
    isNormalUser = true;
    extraGroups = [ "wheel" ];
  };

  environment.systemPackages = with pkgs; [ wget vim firefox ];
}
```

Výpis 2.1: Príklad konfiguračného súboru `configuration.nix`

Zmeny vykonané v konfiguračnom súbore sa uplatnia až po znovu-zostavení systému pomocou príkazu:

```
$ nixos-rebuild switch
```

Tento príkaz vyhodnotí atribúty v konfiguračnom súbore, nainštaluje chýbajúce balíčky a závislosti a spustí aktivačný skript. Aktivačný skript aplikuje novú konfiguráciu systému a reštartuje systémové služby, u ktorých boli vykonané zmeny. Celý proces rekonfigurácie, prípadne upgrade systému je atomický. To znamená, že konfigurácia aktívna počas zostavovania novej konfigurácie nie je nijako ovplyvňovaná – žiadne súbory sa neprepisujú novými verziami. Zostavenie novej konfigurácie systému s prepínačom `switch` vytvorí nový záznam v zavádzači GRUB a nastaví ho ako predvolený pre bootovanie. Inou možnosťou je použiť prepínač `test`, s ktorým sa vykoná rovnaká činnosť, ale nový záznam v zavádzači sa nenastaví ako predvolený. Existujú aj ďalšie možnosti zostavenia pre otestovanie správnosti konfigurácie a pod. o ktorých sa píše v dokumentácii k NixOS [11].

Novovytvorené záznamy v zavádzači GRUB je možné využiť pre návrat k starším konfiguráciám systému (tzv. rollback). Keďže zmeny v NixOS nie sú deštruktívne, rovnako ako to je v Nix, návrat k predchádzajúcej verzii systému je veľmi jednoduchý. Stačí z menu zavádzacieho programu vybrať požadovanú verziu konfigurácie, ktorá je stále dostupná v úložisku Nix store. Iným spôsobom je použitie príkazu pre návrat k predchádzajúcej verzii:

```
$ nixos-rebuild --rollback switch
```

Konfigurácie sú uchovávané po celý čas existencie systému a nie sú automatický odstraňované. Používateľ môže nechcené konfigurácie odstrániť podobne ako generácie v Nix a následne spustiť proces čistenia (garbage collector).

```
$ nix-env -p /nix/var/nix/profiles/system --delete-generations old
$ nix-env --gc
```

Systém je zložený najmä zo služieb, ktoré majú svoje funkcie – napríklad DHCP klient, SSH démon, sieťové služby a pod. Tieto služby sú v NixOS počas ich behu monitorované pomocou služby Upstart, ktorá ich tiež zapína pri bootovaní systému a ukončuje pri vypínaní systému. Jednotlivé služby na sebe rôzne závisia. Napríklad služba SSH môže byť spustená až po tom, čo úspešne bežia sieťové služby. Upstart spravuje jednotlivé služby pomocou úloh (tzv. jobs), ktoré sú zostavované staticky z konfiguračných súborov pomocou derivácii. Úlohy sú ukladané v `/nix/store/` podobne ako balíčky, a ich názvy sa generujú s využitím hashovania pre zachytenie všetkých závislosti. Výhodou tohto prístupu je možnosť paralelného spustenia rovnakej služby s inou konfiguráciou, napríklad pre účely testovania. V systéme ale tiež existujú služby, ktoré nemôžu byť zostavené staticky pomocou derivácii Nix, pretože využívajú konfiguračné súbory, ktoré sa počas behu systému menia. Príkladom je konfiguračný súbor DNS služby `resolv.conf` v adresári `/etc/`, ktorý je počas behu systému modifikovaný DHCP klientom. Ďalšími typmi konfiguračných súborov sú také, ktoré môžu byť vygenerované pomocou derivácii, a následne sú k týmto súborom pomocou aktivačného skriptu vytvorené symbolické odkazy zo zložky `/etc/` do `/nix/store/`. Používanie konfiguračných súborov v adresári `/etc` narúša čisto funkcionálny prístup, nakoľko súbory a symbolické odkazy sú stavovou záležitosťou [4].

2.4 NixOps

Nasadzovanie, správa a sledovanie služieb na množine strojov nie je jednoduchou úlohou. Administrátor systému musí vynakladať nemalé úsilie a ani reprodukcia rovnakých krokov nemusí viesť vždy k rovnakému výsledku. NixOps je nástroj na nasadzovanie (tzv. *deployment*) služieb na stroje v spoločnej sieti alebo cloude. Podľa zadanej špecifikácie strojov vykonáva potrebné úkony, aby bola táto špecifikácia na cieľovej platforme dosiahnutá – inštaluje balíčky, zostavuje závislosti, spúšťa a spravuje systémové služby, a pod. Je založený na Nixe a stavia na jeho prednostiach. Nasadzovanie je atomické, umožňuje rollbacky, a zaručuje kompletnosť závislostí.

Nasadzovanie je deklaratívne. Riadi sa podľa špecifikácie v konfiguračnom súbore. Nezáleží, či sa konfigurácia aplikuje na nový stroj, alebo sa upravuje už existujúci. Výsledok nasadenia podľa toho istého konfiguračného súboru bude totožný. Proces je plne automatický a reprodukovateľný. NixOps dokáže zabezpečovať tzv. *provisioning* strojov – pridávať virtuálne zariadenia, či konfigurovať logické disky, a taktiež nasadiť systémy rovnakej konfigurácie na rôzne platformy (virtuálny stroj, lokálny stroj alebo aj do cloudu). NixOps vie nasadzovať konfiguráciu na akýkoľvek fyzický stroj s NixOS, a podporuje tiež Amazon EC2, Google Compute Engine, Hetzner a VirtualBox [5].

NixOps je modulárny systém, vďaka čomu je možné oddeliť špecifikáciu logických a fyzických častí. Modul `A` môže potom špecifikovať softvérové služby, ako napríklad webový server a jeho súčasti, a modul `B` zasa hardvérové časti, akou je napríklad popis cieľovej platformy. Modul `A` teda popisuje *čo* sa má nasadiť, a modul B *kam* sa to má nasadiť. Jednotlivé moduly sa potom môžu kombinovať a dopĺňať existujúci systém. Výhodou je, že NixOps používa jednotný formalizmus (výrazy Nix) pre špecifikovanie balíčkov i systémových nastavení.

Vybrané príkazy nástroja NixOps

Medzi základné príkazy nástroja NixOps patria `create`, `deploy`, `info`, `check`, `destroy` a `delete`.

Príkaz `nixops create` vytvorí v databáze záznam o novej konfigurácii pre nasadenie. Ako parametre vyžaduje názov pre nový deployment a Nix výrazy s konfiguráciou. Výrazy môžu byť zapísané priamo v príkaze. Častejším a prehľadnejším riešením je použitie súborov s danou konfiguráciou zapísanou v jazyku Nix. Výstupom tohto príkazu je identifikátor (UUID) vytvoreného deploymentu. Identifikátor je na rozdiel od názvu vždy unikátny a jednoznačne označuje daný deployment.

Spustením príkazu `nixops deploy` prebehne nasadenie vytvorenej konfigurácie na cieľovú stanicu. Prvým krokom nasadenia je vytvorenie páru SSH kľúčov pre zabezpečenie komunikácie medzi riadiacim a cieľovým uzlom. Následne za zostaví konfigurácia a systém vyhodnotí, aké balíčky je potrebné stiahnuť z globálneho repozitára a prípadne preložiť. Po úspešnom dokončení dôjde k nakopírovaniu balíčkov na cieľovú stanicu a aktivácii konfigurácie. Prebehne tiež spustenie požadovaných služieb, prípadne reštart, ak sa menila ich konfigurácia. Ďalej nepožadované služby sú zastavené.

Príkazy `nixops info` a `nixops check` zobrazujú informácie o stave daného deploymentu. Prvý príkaz zobrazí všeobecné informácie, akými sú napríklad názov, UUID, a výrazy, z ktorých deployment vychádza. Tiež zobrazí základné informácie o každom stroji v danom deploymente – názov stroja, jeho stav, typ a IP adresu. Druhý uvedený príkaz zobrazuje podrobnejšie informácie o každom stroji – jeho názov, dostupnosť v rámci siete, informáciu o tom či je zapnutý alebo nie je, ďalej môže zobrazovať stav diskov, vyťaženie procesora, a informácie o stave služieb, ktoré na daných strojoch zlyhali.

Nepoužívané konfigurácie nasadených systémov je možné z databázy odstrániť pomocou príkazu `nixops delete`. V prípade, že chceme odstrániť konfigurácie, ktoré stále obsahujú dostupné zdroje, je najprv potrebné tieto zdroje zneškodniť (zastaviť ich beh) pomocou príkazu `nixops destroy`. Ďalšie príkazy a presnú syntax spomínaných je možné nájsť v manuále nástroja NixOps [5].

2.5 Nix Flakes

Systém Nix a nástroje, ktoré z neho vychádzajú, stavajú na reprodukovateľnosti. Výsledkom nasadenia rovnakej konfigurácie na dvoch na sebe nezávislých strojoch by mali byť dva totožné stroje. To však platí len v prípade, že je celá konfigurácia uložená v jednom zdrojovom súbore, alebo je na oboch strojoch rovnaká štruktúra súborov s konfiguráciou. Dosiahnutie tejto vlastnosti je veľmi náročné, a v prípade skutočne komplexných riešení priam nemožné. Riešením tohto problému je použitie Nix Flakes.

Flake je adresárová štruktúra (napríklad Git repozitár), ktorá obsahuje súbor `flake.nix`. Obsahom tohto súboru sú výrazy v jazyku Nix, ktoré popisujú daný Flake. V súbore je špecifikovaná zdrojová adresa alebo repozitár, popis, a výstup. Výstupom sa v tomto prípade rozumieme popis derivácie, ktorá bude pomocou príkazu `nix build` zostavená. Uvedenie URL adresy repozitára ako zdroja pre deriváciu sa môže javiť nepresné, keďže repozitár môže byť v budúcnosti zmenený. To môže viesť k problémom v reprodukovateľnosti. Z tohto dôvodu sa počas zostavovania derivácie vytvorí súbor `flake.lock`, do ktorého budú uložené podrobné informácie o revíziách repozitárov, ktoré sa použili.

Nástroj Nix Flakes je zatiaľ dostupný len medzi experimentálnymi nástrojmi v repozitári `nixUnstable`. Je to však už veľmi obľúbený a v komunite používateľov rozšírený nástroj.

Očakáva sa, že bude čoskoro zaradený medzi stabilné balíčky Nix systému. Aktuálne je pre používanie flakes potrebné do konfigurácie systému pridať nasledujúce výrazy [9]:

```
nix = {
    package = pkgs.nixUnstable;
    extraOptions = ''
        experimental-features = nix-command flakes
    '';
};
```

2.6 Podpora NixOS na Raspberry Pi

Podpora platformy ARM, kam spadá aj Raspberry Pi, je stále vo fáze *work-in-progress*. To znamená, že sa na nej stále intenzívne pracuje a podpora nemusí byť úplná. Podpora sa líši podľa architektúry, aj konkrétneho modelu dosky. Plnú (tzv. *upstream*) podporu má aktuálne len model Raspberry Pi 3. Pre model Raspberry Pi 4B existuje podporovaná, avšak stále experimentálna verzia, ktorá stále obsahuje isté nedostatky. Ostatné modely Raspberry Pi sú podporované komunitou podobne ako model 4B [12].

Inštalácia NixOS na Raspberry Pi 4 si vyžaduje vykonať špecifické kroky. Na webových stránkach [13] sú dostupné zostavené obrazy operačného systému, ktoré je potrebné nakopírovať na SD kartu (napríklad pomocou nástroja `dd`). Po nabootovaní nie je potrebné vytvárať zväzky ani mapovať súborový systém, ako pri klasickej inštalácii. Je potrebné vygenerovať konfiguračný súbor systému. Automaticky vygenerovaný konfiguračný súbor však nie je správny a inštalácia systému by nebola úspešná. Preto je potrebné súbor upraviť pre potreby platformy Raspberry Pi. Príklad základnej konfigurácie je dostupný na Wiki stránkach NixOS [13]. Súbor je ešte možné upraviť a následne spustiť zostavenie systému podľa zadanej špecifikácie pomocou príkazu `nixos-rebuild switch`. Bootovanie z USB a zo siete, ktoré Raspberry Pi 4 podporuje v iných operačných systémoch, žiaľ v NixOS nefunguje spoľahlivo [8].

Kapitola 3

Návrh systému pre správu klastra

V tejto kapitole budú popísané existujúce riešenia správy klastrov a špecifikované požiadavky na systém pre nasadenie, správu a sledovanie klastra. Tiež bude predstavený návrh takéhoto systému a jeho podpornej webovej aplikácie.

3.1 Existujúce riešenia

Virtuálizácia a kontajnery Virtuálizácia a kontajnery umožňujú plnú kontrolu nad nasadzovaným systémom. Dnes patria medzi veľmi populárne techniky nasadzovania služieb. Patria tu napríklad technológie Docker či VMware. Nevýhodou je veľká réžia a náročné vytváranie virtuálnych strojov. Pri virtuálizácii môže dochádzať k neefektívnemu využívaniu pridelených zdrojov jednotlivými aplikáciami [19].

Nástroj enviroment modules Tento systémový nástroj umožňuje správcom pridávať do systému moduly knižníc alebo aplikácií, ktoré sú zdieľané naprieč uzlami. Používatelia systému si potom môžu prispôsobovať svoje pracovné prostredie s využitím týchto modulov, ktoré si sami aktivujú. Udržiavanie repozitára modulov s rôznymi verziami rovnakých balíčkov a ich závislosťami je neľahkou úlohou pre administrátorov systému. Problémy môžu nastať aj pri aktualizáciách balíčkov, ktoré môžu spôsobiť problémy s kompatibilitou závislostí [1].

3.2 Požiadavky na systém

Navrhovaný systém by mal umožňovať nasadenie, správu a sledovanie klastra pomocou technológií Nix. Okrem terminálového rozhrania by mala byť dostupná aj webová aplikácia, ktorá by správu zjednodušovala. V nasledujúcich bodoch sú popísané požiadavky na samotný systém i webovú aplikáciu. Systém by mal umožniť:

- nasadzovať konfiguráciu na uzly klastra,
- upravovať existujúcu konfiguráciu klastra,
- vytvárať nové konfigurácie pre jednotlivé uzly i celý klaster,
- meniť systémové nastavenia klastra – napr. hostname uzlov, IP adresy, časové pásmo, konfiguráciu DNS serveru a pod.,

- spravovať zoznam používateľov a skupiny, do ktorých sú priradení,
- aktualizovať firmware uzlov v klastri,
- sledovať zaťaženie klastra – využívanie CPU a pamäte, množstvo dát prenesených na sieťovom rozhraní a využitie dostupnej kapacity na disku,
- sledovať teplotu jednotlivých uzlov v klastri,
- spravovať používateľov webovej aplikácie – pridávať nových používateľov, meniť pridelené roly, upravovať ich údaje a odoberať ich zo systému.

Podporná webová aplikácia by mala poskytovať intuitívne rozhranie pre najčastejšie vykonávané úkony nad klastrom. Mala by prehľadne vizualizovať vyťaženie systému a upozorňovať používateľa na možné chyby v systéme. Webová aplikácia by nemala poskytovať plnohodnotnú náhradu správy systému pomocou textového rozhrania.

3.3 Návrh systému a podpornej webovej aplikácie

Všetky riadiace komponenty systému pre správu klastra budú nainštalované na hlavnom (riadiacom) uzle (tzv. *master node*). Pomocou nich bude možné nasadzovať, spravovať a monitorovať ostatné výpočtové uzly v klastri. Schéma architektúry systému je znázornená na obrázku 3.1. Operačný systém na všetkých uzloch bude NixOS, ktorý umožňuje deklaratívnu, stavovú a nedeštruktívnu správu konfigurácie systému. Komunikácia riadiaceho uzla s ostatnými uzlami bude prebiehať s využitím technológie NixOps. Systémové služby Grafana a Prometheus budú mať na starosti monitorovanie klastra a vytváranie dashboardov pre prehľadné zobrazenie informácií vo webovej aplikácii. Webová aplikácia bude zostavená z klientskej časti, v ktorej budú použité technológie HTML a CSS a dashboardy s grafmi z nástroja Grafana. Serverová časť bude postavená na technológii PHP, ktorá bude komunikovať s klientskou časťou aplikácie, a tiež s ostatnými nástrojmi pre správu klastra.

Prometheus a Grafana

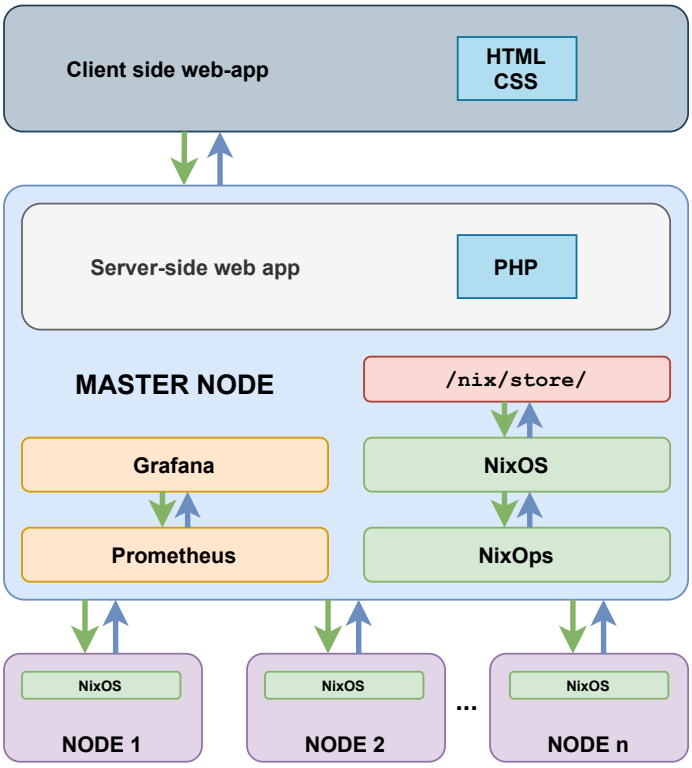
Prometheus je open-source sada nástrojov na monitorovanie a oznamovanie systémových udalostí. Dnes už samostatný projekt bol pôvodne založený spoločnosťou SoundCloud[1].

Hlavným komponentom systému je Prometheus server, ktorý získava a ukladá dáta získané v čase. Hlavnými prednosťami systému Prometheus sú multidimenzionálny dátový model, dotazovací jazyk PromQL a nezávislosť od ostatných systémov[16]. Prometheus získava dáta jednotlivých metrík na cieľových uzloch. Na cieľových uzloch musí byť rovnako spustená Prometheus služba a nastavený export dát, ktoré budú môcť byť zozbierané hlavným uzlom. V našom prípade bude použitý exportér *node* a v rámci neho kolektor *systemd*. Dáta v tejto kolekcii obsahujú metriky operačného systému, i hardvéru uzlov.

Grafana umožňuje získavať dáta, vykresľovať ich a vytvárať upozornenia, bez ohľadu na miesto uloženia týchto dát. Tento systém je možné ľubovoľne rozširovať vďaka veľkému množstvu pluginov. Podporuje mnoho rozšírených databáz dát ako sú Graphite, Prometheus, Elasticsearch, či InfluxDB [6].

Zdrojom dát bude v tejto práci databáza serveru Prometheus, ktorá je uložená na hlavnom uzle klastra. Z týchto dát budú vytvorené grafy a vizualizácie, ktoré budú zobrazovať

[1] https://soundcloud.com/



Obr. 3.1: Schéma architektúry systému pre správu klastra.

jeho aktuálny stav. Konkrétne budú vykresľovať vyťaženie procesorov a využitie pamäte uzlov klastra, objem dát prenesených na sieťových rozhraniach, teplotu procesorov a dostupnú kapacitu súborového systému.

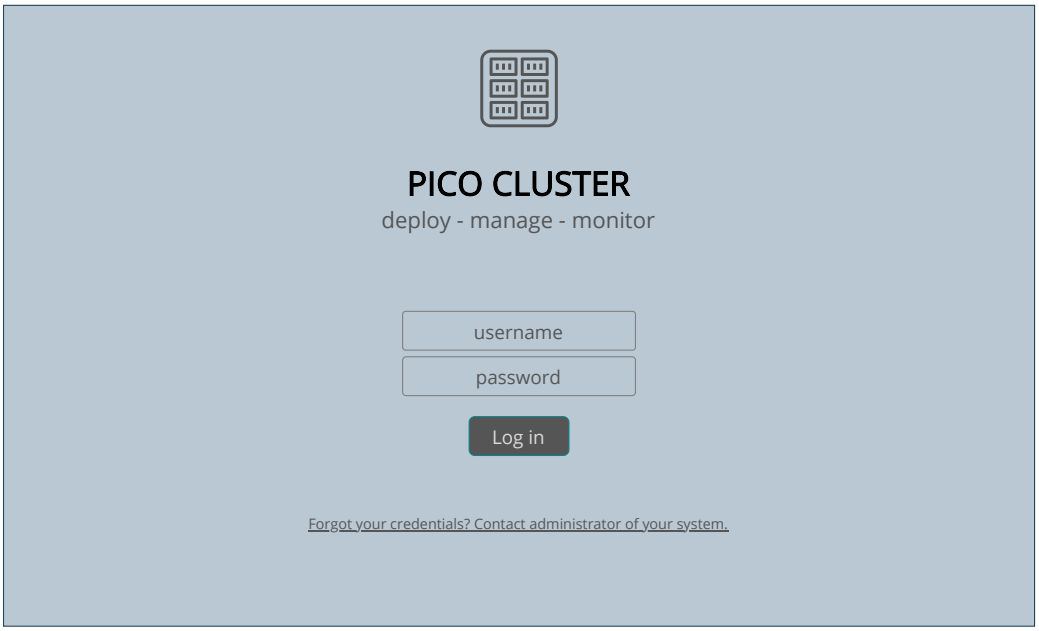
Návrh GUI

Grafické užívateľské rozhranie aplikácie by malo byť intuitívne a malo by pre používateľov predstavovať príjemnejšiu alternatívu pre správu klastra v porovnaní s textovým rozhraním príkazového riadku.

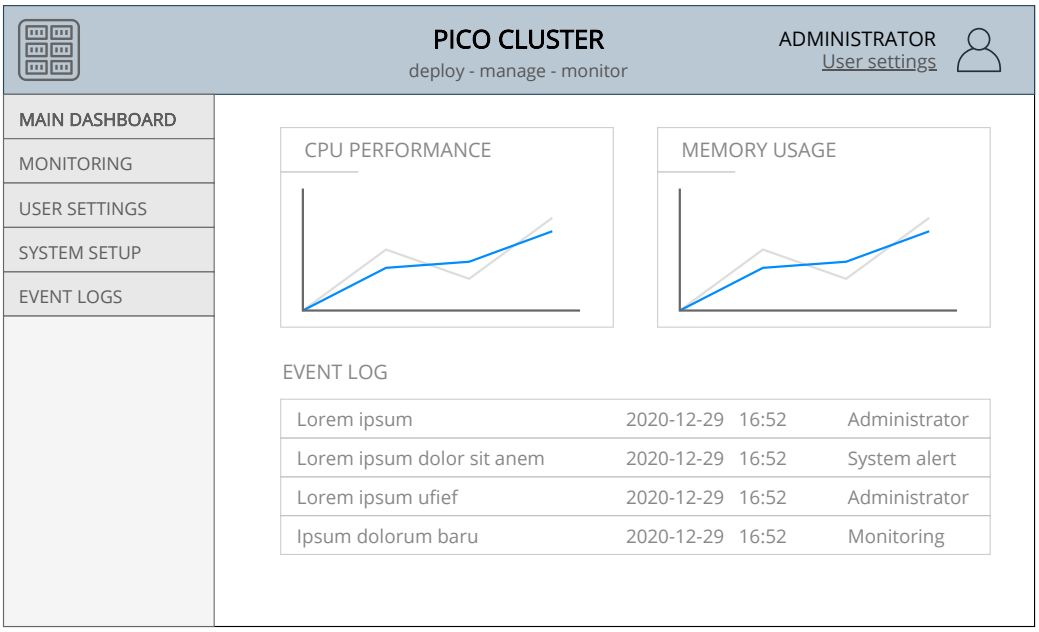
Na obrázku 3.2 je znázornený návrh úvodnej obrazovky aplikácie, kde bude potrebné prihlásiť sa pomocou užívateľského mena a hesla. V prípade straty prihlasovacích údajov bude potrebné kontaktovať administrátora systému, nakoľko účet v aplikácii nebude naviazaný na žiadnu e-mailovú adresu, a teda nebude možné obnoviť heslo bez jeho zásahu.

Obrázok 3.3 znázorňuje domovskú obrazovku aplikácie, ktorá sa zobrazí po prihlásení. Zobrazujú sa tu náhľady zaťaženia klastra, prehľad systémových udalostí. V pravom hornom rohu je možné vidieť názov užívateľského účtu a možnosť zmeniť nastavenia užívateľa. Menu na ľavej strane umožňuje prepínať medzi časťami aplikácie.

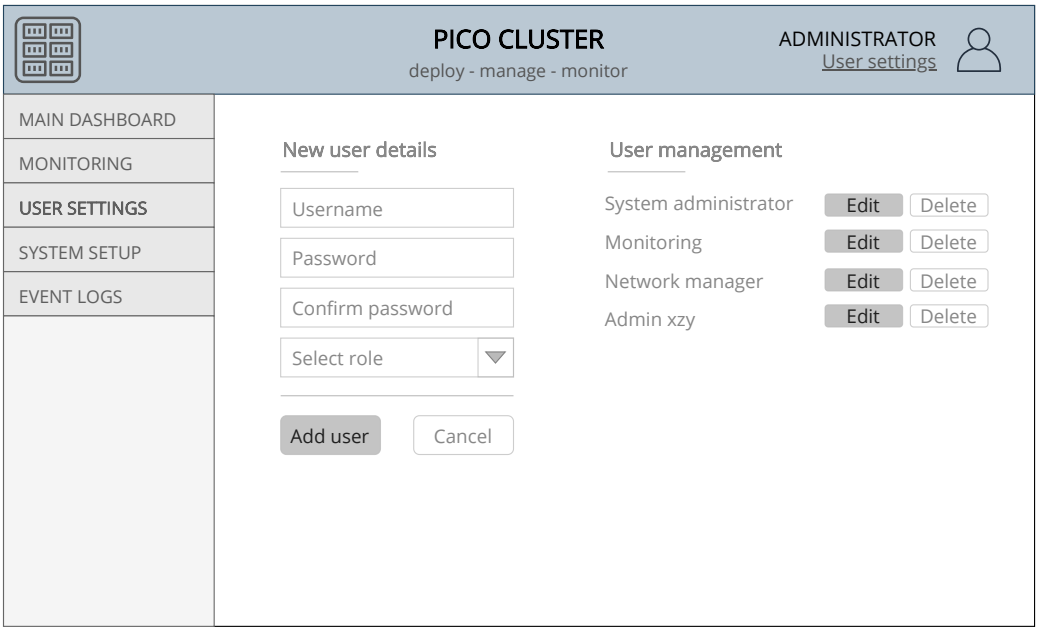
Návrh rozhrania správy používateľských účtov je znázornený na obrázku 3.4. Umožňuje pridávať do systému nových používateľov a priraďovať im príslušné roly. Tiež zobrazuje všetkých používateľov, ktorí v systéme existujú, a ponúka možnosť upraviť ich alebo odoberať zo systému.



Obr. 3.2: Návrh úvodnej obrazovky pre prihlásenie do aplikácie.



Obr. 3.3: Návrh domovskej obrazovky aplikácie, ktorá sa zobrazí po prihlásení.



Obr. 3.4: Návrh obrazovky znázorňujúci správu používateľov systému.

Kapitola 4

Implementácia systému

V tejto kapitole budú popísané najdôležitejšie koncepty použité pri vytváraní a nasadzovaní konfigurácie klastra, aké zmeny bolo potrebné vykonať oproti návrhu systému a aké boli dôvody pre vykonanie týchto zmien. Taktiež budú popísané hlavné časti implementácie podpornej webovej aplikácie.

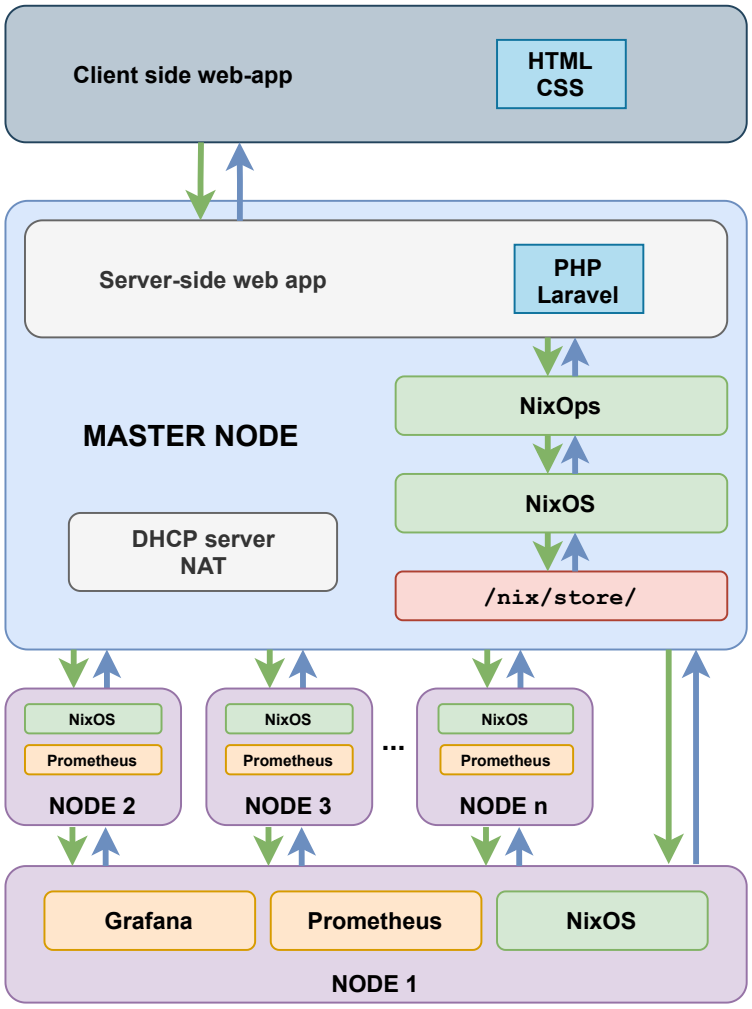
4.1 Zmena architektúry systému

V sekcii 3.3 venovanej návrhu systému bolo uvedené, že všetky riadiace komponenty budú nainštalované na hlavnom riadiacom uzle. Za tento uzol je v danej kapitole považovaný jeden z uzlov v rámci klastra, teda Raspberry Pi. Počas implementácie sa však ukázalo, že nástroj NixOps nie je v súčasnosti možné nainštalovať na Raspberry Pi, ani na iný stroj s architektúrou aarch64. Dôvodom je, že súčasná verzia NixOps, ktorá je dostupná v repozitári, je 1.7[1]. Táto verzia využíva Python 2.7, ktorého podpora skončila v roku 2020. Závislosti potrebné pre zostavenie a beh NixOps na Raspberry Pi už nie sú v repozitári pre túto architektúru dostupné. Aktuálne vyvíjaná verzia 2.0 využíva Python 3, ale je zatiaľ dostupná len v GitHub repozitári. Spustenie tejto verzie je možné len s využitím `nix-shell`. Toto riešenie ale nie je vhodné pre integráciu do webovej časti aplikácie, a preto bolo nevyhnutné pozmeniť architektúru celého systému.

Súčasťou výsledného systému je teda okrem Raspberry Pi uzlov aj stroj s architektúrou x86 (ďalej označovaný ako *riadiaci uzol*). Na tento stroj bola presunutá časť systému zodpovedná za nasadzovanie a správu klastra pomocou nástroja NixOps. Keďže NixOps príkazy sú spúšťané aj z prostredia webovej aplikácie, bolo nevyhnutné presunúť na tento stroj aj webový server aplikácie. Na hlavnom Raspberry Pi uzle ostali služby starajúce sa o monitorovanie – Prometheus a Grafana. Schéma výslednej architektúry je zobrazená na obrázku 4.1. Na riadiacom uzle je nainštalovaný operačný systém NixOS, aby bola zachovaná možnosť jednoduchej reprodukovateľnosti tohto riešenia.

V okamihu, kedy bude verzia 2.0 nástroja NixOps dostupná v repozitári, bude možné preniesť zodpovednosti z riadiaceho uzla architektúry x86 na hlavný uzol v klastri a daný riadiaci uzol odstrániť zo systému.

[1] https://github.com/NixOS/nixops



Obr. 4.1: Schéma upravenej architektúry systému pre správu klastra.

Topológia siete / sieťová architektúra

Z dôvodu zvýšenia robustnosti riešenia bola pre klaster vytvorená lokálna sieť v rámci privátneho rozsahu. Riadiaci uzol disponuje dvoma sieťovými rozhraniami. Jedno z nich (`enp3s0`) bolo využité na komunikáciu medzi riadiacim uzlom a ostatnými uzlami v klastri. Prostredníctvom druhého rozhrania (`wlp4s0`) bol umožnený prístup na internet. Na riadiaci uzol bol nainštalovaný DHCP server, ktorý sa stará o dynamické prideľovanie IP adries jednotlivých uzlov klastra. Pre spustenie DHCP služby je potrebné do konfiguračného súboru systému pridať nasledujúce parametre:

```
services.dhcpd4 = {
  enable = true;
  interfaces = ["enp3s0"];
  extraConfig = ''
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    option subnet-mask 255.255.255.0;
    option routers 10.10.0.1;
    default-lease-time 10000;
```

```
      subnet 10.10.0.0 netmask 255.255.255.0
      {
        ...
      }
    '';
};
```

Týmto nastavením sa povolí DHCP služba a nastaví sa sieťové rozhranie, na ktorom bude služba fungovať. V sekcii `extraConfig` sú uvedené ďalšie parametre služby, pre ktoré neexistujú voľby v rámci NixOS systému, ale je potrebné ich nastaviť. Tieto nastavenia budú pridané na koniec konfiguračného súboru DHCPv4 služby. Nastaví sa adresa DNS serverov, maska podsiete, router pre danú sieť, doba platnosti pridelených IP adries DHCP serverom, a samotná podsieť. Ďalej je špecifikovaný rozsah dynamicky prideľovaných IP adries, a tiež zoznam statických adries rezervovaných pre jednotlivé uzly klastra, definované MAC adresou príslušného rozhrania.

Aby bolo možné do verejnej siete internetu pristupovať aj z uzlov klastra, bol nastavený preklad sieťových adries (NAT). Do konfigurácie systému boli pridané nasledujúce parametre, ktoré povolia preklad adries a určia zdrojové a cieľové rozhranie pre preklad.

```
networking.nat = {
    enable = true;
    externalInterface = "wlp4s0";
    internalInterfaces = ["enp3s0"];
};
```

4.2 Konfigurácia klastra

Klaster pozostáva z niekoľkých výpočtových a jedného hlavného uzla. Výpočtové uzly sú v konfigurácii označované ako *node* a hlavný uzol ako *head-node*. Ako bolo uvedené v predošlej sekcii, značná časť zodpovedností hlavného uzla bola presunutá na stroj mimo klaster. Hlavnému uzlu zostala zodpovednosť za monitorovanie ostatných uzlov a vytváranie dashboardov.

V konfiguračnom súbore pre nasadenie pomocou NixOps môže byť definovaný popis daného nasadenia a povolenie, resp. zakázanie návratu k predchádzajúcim verziám (tzv. *rollback*). Ďalej súbor obsahuje sekciu `defaults` a sekcie s parametrami strojov, ktoré budú nasadené. Parametre uvedené v sekcii defaults sú spoločné pre všetky stroje v danom nasadení. V našom prípade je v tejto časti špecifikované nastavenie SSH služby, časové pásmo a používatelia systému.

Konfigurácia strojov obsahuje dve rozdielne konfigurácie – jednu pre výpočtové uzly a druhú pre hlavný uzol. Obe z nich sú funkciami, ktoré ako parameter vyžadujú hostname stroja. Parametrizácia umožňuje prehľadnejší zápis konfigurácie, prípadné zmeny je nutné vykonať len na jednom mieste v súbore, aj v prípade nastavovania veľkého počtu uzlov. Pomocou parametra predaného funkcii zodpovednej za vytvorenie konfigurácie pre stroj sa odkazujeme do štruktúry v súbore `nodes-config.nix`, ktorá je načítaná do premennej `nodes`. Táto štruktúra obsahuje záznam o každom stroji, ktorý chceme, aby bol nasadený. V zázname je definovaná IP adresa a hostname. IP adresa je použitá pre nastavenie parametra `deployment.targetHost`, ktorým určujeme cieľové zariadenie pre nasadenie danej konfigurácie.

Konfigurácia výpočtového uzlu

Prvým veľmi dôležitým parametrom, ktorý musel byť pre správne fungovanie nasadenia nastavený, je `nixpkgs.system`. Tento parameter špecifikuje architektúru systému nasadzovaného stroja. Určuje aké verzie balíčkov budú zostavené a nainštalované tak, aby boli spustiteľné. Pre nasadenie na Raspberry Pi je potrebné tento parameter nastaviť na hodnotu `"aarch64-linux"`.

Nastavenie parametrov bootovania, ako sú moduly jadra, moduly pre počiatočný ramdisk (initrd), alebo špecifické parametre pre platformu Raspberry Pi, sú uložené v súbore `hw-rpi.nix`. Taktiež je v ňom uvedená špecifikácia súborového systému a systém regulácie napájania. Obsah tohto súboru je následne naimportovaný do hlavnej konfigurácie.

Ďalšími dôležitými parametrami je upravené nastavenie siete. Názov hostiteľa (hostname) je nastavený parametricky zo súboru so záznamami o jednotlivých uzloch. Služba DHCP klienta na ethernetovom rozhraní `eth0` je povolená, keďže sú uzly v privátnej sieti s vlastným DHCP serverom. Bezdrôtové rozhranie uzlov sa nepoužíva.

Na každom výpočtovom uzle beží Prometheus služba, ktorá ukladá do databázy metriky systému. Okrem samotnej služby je nastavený aj tzv. *exportér*, ktorý určuje, aké dáta budú zbierané a uchovávané. Pre potreby tejto práce je vyhovujúci *Node Exporter* s kolektorom *systemd*. Táto konfigurácia umožňuje získavať metriky z operačného systému a jeho jadra.

```
services.prometheus = {
  enable = true;
  port = 9001;
  exporters.node = {
    enable = true;
    enabledCollectors = [ "systemd" ];
    port = 9002;
};};
```

Konfigurácia hlavného uzla

Hlavný uzol sa od výpočtového odlišuje len nastavením Prometheus služby a tým, že je na ňom spustená služba Grafana servera. Do konfigurácie Prometheus servera pribudla sekcia `scrapeConfigs`, ktorá zabezpečuje, že tento uzol bude zbierať metriky zo všetkých ostatných uzlov (v konfigurácii označované ako `targets`), a následne ich sprístupní Grafana službe.

Parametre Grafany určujú na akej IP adrese a porte má byť služba dostupná a ďalšie dve voľby potrebné pre správne fungovanie vo webovej aplikácii. Aby bolo možné vizualizácie integrovať do webovej aplikácie za použitia HTML bloku `<iframe>`, musí byť nastavený parameter `extraOptions.SECURITY_ALLOW_EMBEDDING` na hodnotu `true`. Možnosť zobrazovať metriky aj pre používateľov, ktorí následne nie sú v aplikácii prihlásení, sa povolí voľbou `auth.anonymous.enable`.

4.3 Webová aplikácia

Webová aplikácia by mala používateľom poskytovať intuitívnejšie prostredie pre prácu s klastrom. Hlavnými časťami aplikácie sú dashboard s prehľadom využívania zdrojov klastra, správa nasadení (deploymentov) a správa používateľov. Jednotlivé časti aplikácie budú podrobnejšie popísané v nasledujúcich odsekoch.

Základom front-end časti aplikácie je Bootstrap téma *SB Admin 2*[2], ktorá bola následne upravovaná. Hlavnými prvkami, ktoré boli z tejto témy použité, sú bočný navigačný panel a vrchný panel. Šablóna má licenciu MIT, ktorá umožňuje so softvérom nakladať bez obmedzení, kopírovať, používať a modifikovať program za predpokladu uvedenia autora a informácií o licencii, spolu s upozornením na zrieknutie sa zodpovednosti za dielo [7]. Back-end časť aplikácie je postavená na PHP frameworku Laravel[3], ktorý používa návrhový vzor MVC.

Nasadenie aplikácie do prevádzky vyžaduje nastavenie Apache HTTP serveru a MySQL databázy, úpravu PHP prostredia a pridanie používateľského účtu, pod ktorým budú spúšťané príkazy z webovej aplikácie. Podrobný návod na inštaláciu webovej aplikácie je uvedený v prílohe C.

Dashboard

Dashboard, ktorý je zobrazený na obrázku 4.2, sa používateľovi zobrazí hneď po prihlásení do aplikácie. Prehľadne zobrazuje využívanie zdrojov klastra. Jednotlivé grafy v dashboarde sú vyexportované z aplikácie Grafana, ktorej služba beží na hlavnom uzle klastra. Grafana ponúka niekoľko možností zdieľania grafov. Pre účely webovej aplikácie bola využitá možnosť tzv. *embeded* panelu, ktorý bol následne vložený do aplikácie ako HTML blok `<iframe>`.

Nastavenie týchto grafov prebieha vo webovom rozhraní tejto aplikácie. Prvým krokom nastavenia je vytvorenie zdroja dát, ktorým je Prometheus server bežiaci na rovnakom uzle. Toto nastavenie sa nachádza v sekcii *Data Sources*, a je potrebné uviesť IP adresu a port, na ktorom je služba Prometheus spustená. Jednotlivé grafy sú následne vytvárané v rámci dashboardu ako panely.

Graf je vykresľovaný podľa dát metriky, ktoré sú získané z databázy Prometheus servera. Dáta sa získavajú dotazovaním do tejto databázy. Príklad dotazu množstva prenesených dát na rozhraní `eth0`:

```
'rate(node_network_transmit_bytes_total{device="eth0"}[5m]'
```

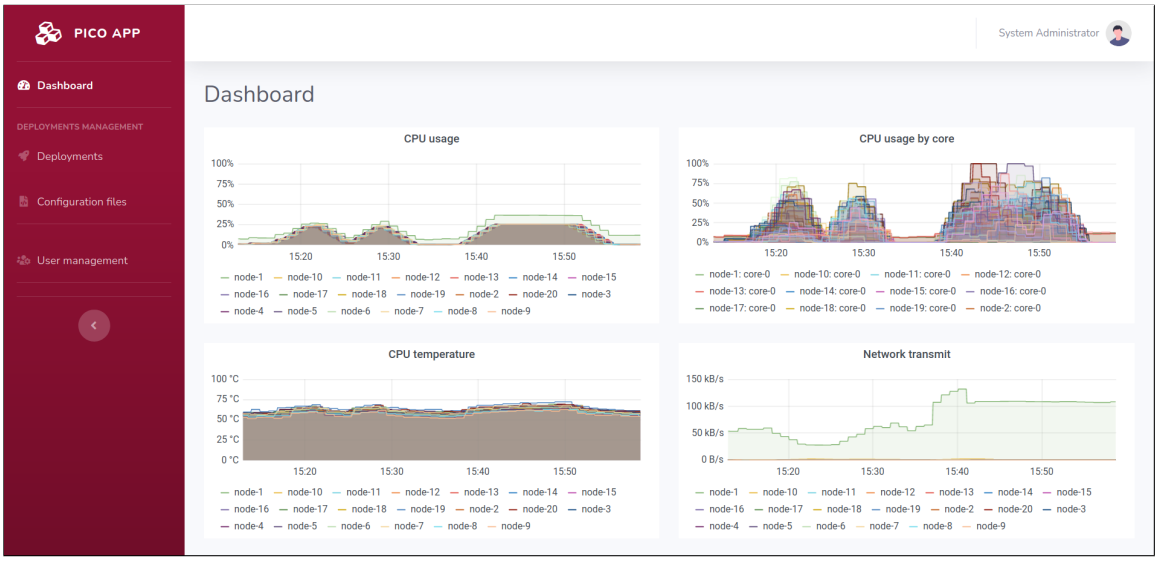
Výsledok dotazu sa okamžite prenesie do grafu, ktorý môže byť následne prispôsobovaný. Grafana poskytuje veľké množstvo nastavení osí, legiend, popisov a ďalších súčastí vizualizácií. Ukážka prostredia webovej aplikácie Grafana je zobrazená v prílohe B na obrázku B.1.

Správa konfiguračných súborov

Konfiguračné súbory sú nevyhnutné pre vytváranie zložitejších nasadení, medzi ktoré rozhodne patrí nasadenie klastra. Na obrázku 4.3 je zobrazená časť aplikácie správy konfiguračných súborov. V tejto časti je zobrazený zoznam existujúcich súborov, ktoré je možné zobraziť, upraviť alebo odstrániť. Taktiež je možné vytvárať nové konfiguračné súbory.

Pre konfiguračný súbor je v aplikácii vytvorený model `Configfie`. Model uchováva informácie o názve, popise a umiestnení konfiguračného súboru. Tieto informácie sú po vyvolaní metódy `store()`, ktorá je súčasťou kontroléra `ConfigfilesController`, uložené do databázy, a následne je na disku vytvorený súbor s obsahom, ktorý zadal používateľ v aplikácii. Ak sa používateľ rozhodne odstrániť konfiguračný súbor z aplikácie, vyvolá sa metóda `destory()`, ktorá odstráni súbor zo systému a následne vymaže záznam z databázy.

[2] https://startbootstrap.com/theme/sb-admin-2
[3] https://laravel.com/



Obr. 4.2: Snímka obrazovky s dashboardom, ktorá sa zobrazí po prihlásení sa do aplikácie.

Správa nasadení

Nasadenia konfigurácií (označované tiež ako *deployments*) sú hlavnou podstatou aplikácie. Používateľom je umožnené vytvárať záznamy o týchto nasadeniach, nasadzovať ich na zariadenia, kontrolovať stav zariadení v daných nasadeniach, a taktiež ich rušiť a odstraňovať zo systému.

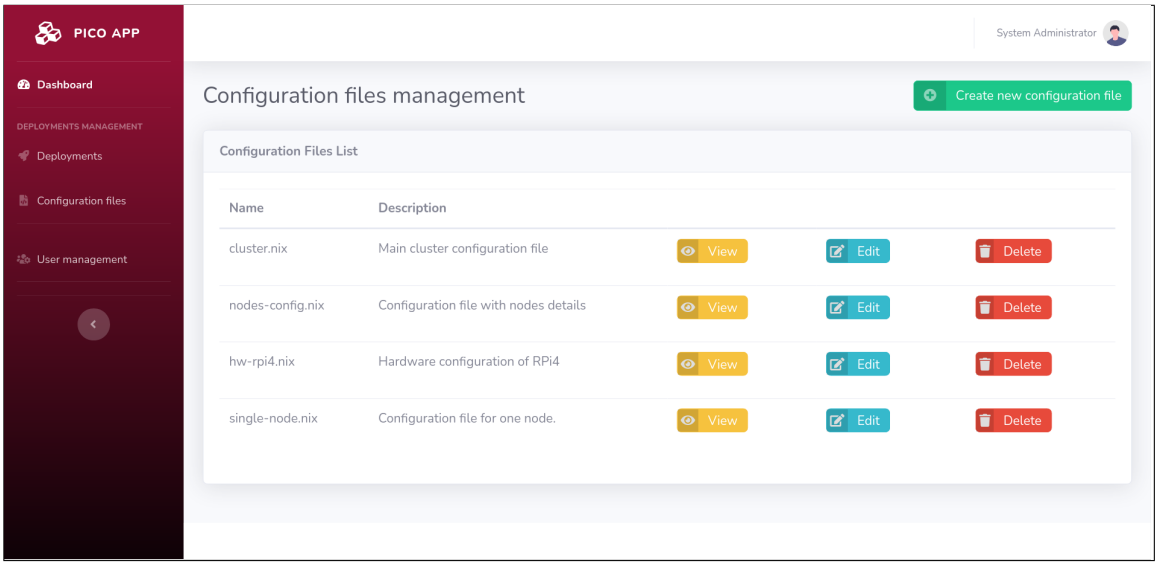
Spomínané akcie sú ekvivalentné s príkazmi nástroja NixOps – `create`, `deploy`, `info`, `check`, `destroy` a `delete`. Zvolením niektorej z týchto akcií vo webovej aplikácii sa vyvolá príslušná metóda v kontroléri nasadení. Z tejto funkcie je volaná PHP funkcia `exec()`, ktorá spustí zadaný príkaz na hostiteľskom systéme a vráti výstup, ktorý by bol vypísaný na štandardný výstup textového rozhrania. Výstup je následne spracovaný a v podobe asociatívneho poľa predaný do časti aplikácie zodpovednej za zobrazovanie.

Príkaz `deploy`, ktorého dĺžka behu závisí na komplexnosti nasadzovaného systému, je spúšťaný na pozadí a jeho výstup je presmerovaný do súboru. Obsah tohto súboru je periodicky získavaný a zobrazovaný používateľovi v aplikácii bez nutnosti obnovovať stránku, aby mal prehľad o priebehu nasadenia. Ukážka priebehu nasadenia v aplikácii je zobrazená na obrázku 4.4.

Správa používateľov

Vytvorená webová aplikácia je viacužívateľská a rozlišuje dve roly používateľov. Prvou rolou je administrátor, ktorý má neobmedzený prístup ku všetkým funkciám aplikácie. Druhou rolou je používateľ s oprávneniami len na sledovanie stavu klastra - tzv. monitoring rola. Používateľ s rolou monitoringu má prístup k dashboardu, môže si zobraziť stav vytvorených nasadení a prezrieť obsah súborov. Tiež môže upraviť svoj profil a zmeniť si heslo.

Pre používateľov je, podobne ako pre konfiguračné súbory, vytvorený model a informácie o používateľoch sú ukladané do databázy. Aplikácia po nasadení do prevádzky obsahuje dva predvolené používateľské účty. Jeden administrátorský a jeden monitorovací. Ďalších používateľov do systému môže pridávať len administrátor. Žiadny účet v aplikácii nie je

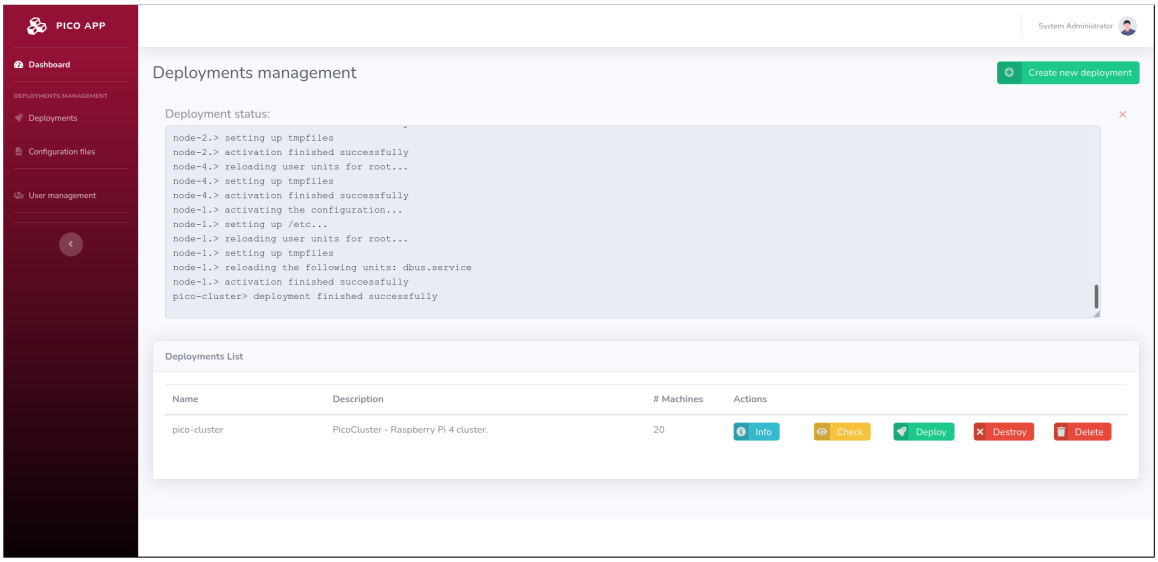


Obr. 4.3: Snímka obrazovky aplikácie – časť správy konfiguračných súborov.

prepojený s e-mailovou adresou a zmenu zabudnutého hesla môže vykonať len používateľ s rolou administrátora.

Overenie, či je používateľ autorizovaný pre vykonávanie zvolenej operácie, prebieha s využitím tzv. brán (*gates*). Brány sú definované v metóde `boot()`, ktorá sa nachádza v triede `\App\Providers\AuthServiceProvider`. V tejto triede je tiež definované asociatívne pole akcií a používateľov oprávnených vykonávať tieto akcie. Rozšírenie aplikácie o ďalšie roly je preto veľmi jednoduché. Stačí do spomínaného asociatívneho poľa k akciám pridať novú rolu a tú potom pridať medzi roly, ktoré môžu byť používateľovi priradené pri jeho vytváraní.

Overovanie oprávnení používateľa prebieha aj pri generovaní front-end časti aplikácie. Pre tento účel je použitá direktíva `@can('action')` a jej uzatváracia časť `@endcan`. Zdrojový kód zapísaný medzi týmito kľúčovými slovami sa vygeneruje a následne zobrazí len používateľom, ktorí sú autorizovaní vykonať danú akciu.



Obr. 4.4: Snímka obrazovky aplikácie – časť správy nasadení s prehľadom aktuálneho stavu procesu nasadzovania.

Kapitola 5

Testovanie a výkon klastra

Táto kapitola popisuje ukážkové úlohy distribuovaných výpočtov v klastri. Za úlohy bol zvolený naivný algoritmus pre hľadanie prvočísel napísaný v jazyku Python a benchmark HPL. V nasledujúcich častiach budú popísané kroky, ktoré je nutné vykonať pre správne fungovanie týchto úloh, priebeh testovania a dosiahnuté výsledky.

5.1 Testovacie prostredie

Pre účely testovania bol dostupný klaster pozostávajúci z 20 uzlov Raspberry Pi 4 s 8 GB operačnej pamäte, ktorý vyrába spoločnosť PicoCluster LLC [15]. Tento klaster je zobrazený na obrázku 5.1. Každé Raspberry Pi v klastri disponuje pasívnym chladičom a celý box má aktívne ventilátory pre dosiahnutie optimálneho prúdenia vzduchu.

Priebeh testovania ukázal, že chladenie klastra nie je pri náročnejších úlohách dostatočné. Po veľmi krátkej dobe (rádovo desiatky sekúnd) došlo k zahriatiu procesorov na úroveň medzi 80°C až 85°C. Hodnota 80°C je hraničná. Túto hodnotu deteguje senzor vo vnútri čipu procesora a po jej prekročení dochádza ku škrteniu výkonu procesora (tzv. throttling), aby sa predišlo trvalému poškodeniu [18]. Ďalšie testovanie prebiehalo z tohto dôvodu len na štvorici uzlov, ktoré bolo možné dostatočne uchladiť aj počas dlhotrvajúcich náročných testov.

5.2 HPL benchmark

High-Performance Linpack (HPL) benchmark je softvérový balík, ktorý rieši náročný lineárny systém so 64-bitovou aritmetickou presnosťou, s využitím distribuovanej architektúry počítačov. HPL poskytuje program na meranie času a určenie presnosti dosiahnutého riešenia úlohy. Dosiahnuteľný výkon je závislý na veľkom množstve parametrov, avšak je škálovateľný s ohľadom na množstvo operačnej pamäte jednotlivých uzlov v sieti [14]. HPL využíva na komunikáciu medzi uzlami knižnicu Open MPI.

Nastavenie parametrov riešenej úlohy sa nachádza v súbore `HPL.dat`. Najdôležitejšie parametre, ktoré je nevyhnutné upraviť pre dosiahnutie čo najlepšieho výsledku, sú: veľkosť problému N, veľkosť bloku NB a rozloženie mriežky, teda parametre P a Q.

Veľkosť riešeného problému N sa určuje podľa množstva dostupnej pamäte v testovanom systéme. Existujú nástroje, ako napríklad HPL Calculator[1]. Nástroj vypočíta niekoľko

[1] http://hpl-calculator.sourceforge.net

Obr. 5.1: Na obrázku je zobrazený klaster Pico 20H od spoločnosti PicoCluster LLC. Obrázok je prevzatý z webových stránok výrobcu [15].

najvhodnejších veľkostí problému podľa toho, koľko pamäte systému sa použije. Hodnota, ktorú odporúčajú autori benchmarku, sa pohybuje na úrovni 80% dostupnej pamäte.

Veľkosť bloku NB určuje pomer medzi distribúciou dát a granularitou výpočtu. Čím bude veľkosť bloku menšia, tým lepšie bude výpočet rozložený medzi jednotlivé uzly. Na druhú stranu, príliš malé bloky dát môžu mať značný vplyv na výpočtový výkon, a taktiež narastá množstvo prenášaných správ medzi uzlami. Násobok parametrov veľkosti mriežky musí odpovedať množstvu procesorov, resp. jadier v celom systéme. Pomer medzi P a Q závisí na topológii siete.

Priebeh a výsledky testovania

Testovanie pomocou HPL benchmarku prebiehalo na jednom samostatnom uzle a následne na štyroch uzloch v klastri. Testy na väčšom množstve uzlov neboli vykonávané z dôvodu nedostatočného chladenia.

Inštalácia softvéru pre spustenie a beh HPL benchmarku je veľmi jednoduchá. Medzi balíčky prostredia je potrebné pridať balíčky `hpl` a `openmpi`.

Výkon jedného uzlu Výkonnostné testy boli spúšťané s rôznymi parametrami pre dosiahnutie čo najlepšieho výsledku. Najlepšou kombináciou parametrov pre jeden uzol sa ukázala byť nasledujúca kombinácia:

- veľkosť problému – $N = 28\,800$
- veľkosť bloku – $NB = 192$
- veľkosť mriežky $P \times Q = 2 \times 2$

Toto nastavenie by malo podľa kalkulačky [20] využívať 88% dostupnej pamäte. Výsledky testov s danými parametrami a výsledky ďalších testov s podobnými výsledkami sú uvedené v tabuľke 5.1. Všetky testy boli spúšťané pomocou nasledujúceho príkazu, pričom obsah súboru `machine1` bol stále rovnaký a obsahoval 4 krát hostname uzlu, na ktorom boli testy spúšťané.

```
$ mpirun -np 4 --machinefile machine1 xhpl
```

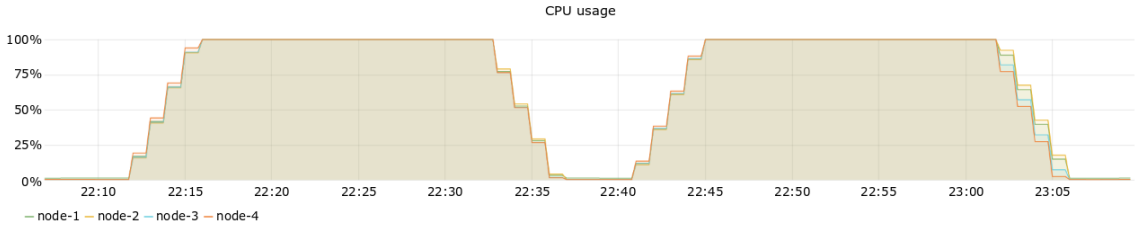
N	**NB**	**P**	**Q**	**čas [s]**	**výsledok [Gflops]**
20000	224	2	2	483,12	11,09
25000	224	2	2	932,51	11,49
28000	224	1	4	1214,36	11,97
28800	192	1	4	1211,32	13,53
28800	192	2	2	1203,71	13,65

Tabuľka 5.1: Výsledky HPL benchmarku spusteného na jednom uzle Raspberry Pi.

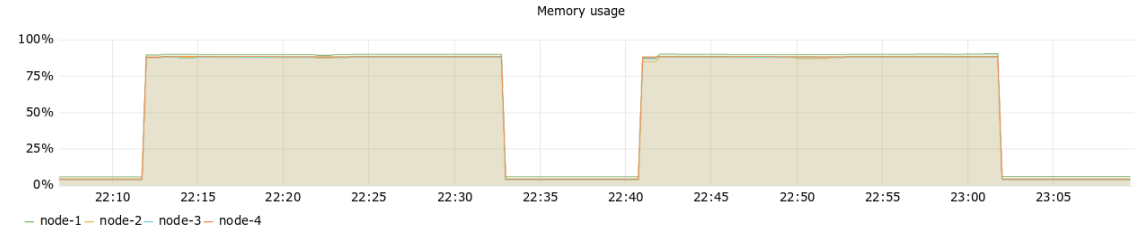
Výkon 4 uzlov Výsledky benchmarku na 4 uzloch boli najlepšie pri použití nasledovnej konfigurácie parametrov:

- veľkosť problému – N = 57 600
- veľkosť bloku – NB = 192
- veľkosť mriežky P × Q = 2 × 8

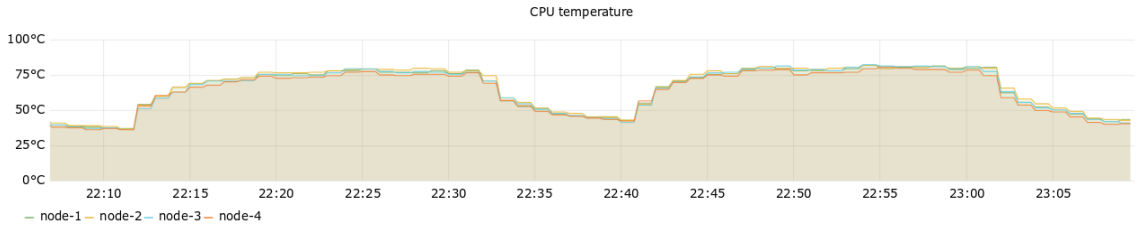
Najlepší nameraný výkon dosahoval hodnotu 25,94 Gflops a testovanie trvalo približne 81 minút. Výsledok toho merania nenaplnil očakávania, keďže sa rovnal len 47,5% maximálneho teoretického výkonu štyroch uzlov. Za najpravdepodobnejšie obmedzenie, ktoré viedlo k značnému poklesu výkonu, možno považovať prepojenie uzlov medzi sebou. Využitie procesorov i pamäte dosahovalo hodnoty porovnateľné s hodnotami pri testovaní jedného uzla. Vyťaženie procesorov bolo 100% a využitie pamäte sa pohybovalo v rozmedzí 86% až 89%. Tieto hodnoty sú vyobrazené na obrázkoch 5.2, 5.3, ktoré sú získané z aplikácie Grafana. Grafy zobrazujú štatistiky dvoch po sebe idúcich behov HPL benchmarku.



Obr. 5.2: Graf vyťaženia procesorov naprieč uzlami klastra získaný z aplikácie Grafana.



Obr. 5.3: Graf využitia operačnej pamäte naprieč uzlami klastru z aplikácie Grafana.



Obr. 5.4: Graf vývoja teplôt procesorov jednotlivých uzlov klastra.

N	**NB**	**P**	**Q**	**time [min]**	**result [Gflops]**
59360	224	2	8	100	22,72
56256	192	4	4	85	23,05
57600	192	4	4	90	23,52
57600	192	2	8	87	24,21
56256	192	2	8	77	25,55
57600	192	2	8	81	25,94

Tabuľka 5.2: Prehľad vybraných výsledkov behov HPL benchmarku na 4 uzloch klastra.

5.3 Hľadanie prvočísel

Problém hľadania prvočísel je často využívaný v rôznych algoritmoch hashovania alebo kryptografie. Existuje mnoho algoritmov pre efektívne hľadanie prvočísel. Problém riešený v tejto úlohe spočíva v nájdení všetkých prvočísel od 2 do n, kde n udáva hornú hranicu prehľadávaného intervalu. Pre riešenie bola zvolená naivná a výpočtovo náročná implementácia dostupná na webe www.the-diy-life.com.

Použitá implementácia hľadania prvočísel zo zadaného rozsahu spočíva v postupnom prehľadávaní celého stavového priestoru zadaného intervalu. Každé číslo x z tohto intervalu je vydelené číslami od 2 do $x-1$. V prípade, že bude zvyšok po delení rovný 0, čo znamená, že číslo má viac než dvoch deliteľov, delenie sa ukončí a bude testované nasledujúce číslo.

Implementácia tohto algoritmu v jazyku Python s využitím modulu `mpi4py` bola vytvorená tak, že dokáže využívať len jedno jadro procesora. Tento program ukazuje, ako fungujú aplikácie, ktoré nedokážu naplno využívať všetky dostupné zdroje.

Inštalácia interpreta jazyka Python vrátane modulu `mpi4py` bola dosiahnutá použitím nasledovnej konfigurácie pre všetky uzly klastra:

```
let
  my-py-pkgs = python-packages: with python-packages; [ mpi4py ];
  my-python = python3.withPackages my-py-pkgs;
in
{
    environment.systemPackages = with pkgs; [ my-python ];
}
```

Testy prebiehali postupne na jednom, štyroch a dvadsiatich uzloch klastra. Testovanie na celom klastri bolo tentokrát možné z dôvodu, že zaťaženie jednotlivých uzlov nebolo maximálne. Plne vyťažené bolo vždy len jedno jadro každého procesora. Testovalo sa 5 intervalov od 2 do n, kde n bolo postupne 10 000, 50 000, 100 000, 200 000 a 500 000. Výsledky testov sú uvedené v tabuľke 5.3.

	čas [s]		
n	**1 uzol**	**4 uzly**	**20 uzlov**
10 000	1,81	0,50	0,14
50 000	38,75	10,59	2,82
100 000	144,15	41,51	10,17
200 000	588,93	143,14	43,31
500 000	3 104,53	867,46	236,74

Tabuľka 5.3: Prehľad výsledkov testov hľadania prvočísel v rozsahu 1 až n na 1, 4 a 20 uzloch.

Kapitola 6

Záver

Navrhnutý systém pre správu výpočtových uzlov v klastri odstraňuje nevýhody imperatívnych balíčkovacích systémov, a predstavuje tak zjednodušenie úlohy systémového administrátora. Použitý nástroj NixOps prináša nedeštruktívne vykonávanie zmien a možnosť návratu k predchádzajúcim verziám nasadených konfigurácií. Tvorí tiež jadro mechanizmu pre jednoduché nasadzovanie jednej konfigurácie na mnoho uzlov v klastri.

Prácou bolo úspešne overené a otestované použitie technológií Nix pre nasadenie a monitoring klastru za účelom distribuovaných výpočtov. Nix je komplexný systém, pre ktorého pochopenie je treba vynaložiť značné úsilie, jeho výhody však prevyšujú toto počiatočné úsilie.

Implementovaná webová aplikácia poskytuje prehľadné používateľské rozhranie pre vykonávanie vybraných úkonov v rámci správy klastru a sledovania jeho stavu prostredníctvom dashboardov. Používateľovi je z jedného miesta umožnené pracovať s konfiguračnými súbormi od ich vytvorenia, cez zobrazenie a úpravu, až po nasadenie na cieľové uzly. Priebeh a výsledok nasadenia je možné sledovať v reálnom čase. Výsledný systém bol zverejnený ako open-source na webovej stránke https://github.com/adamzivcak/nix-cluster-app.

Testovanie systému a meranie jeho výkonu overilo funkčnosť nasadeného riešenia na reálnych problémoch. Zaťaženie systému pomocou benchmarku HPL odhalilo poddimenzované chladenie testovaného klastru.

Priestor pre ďalšie vylepšenie implementovanej podpornej aplikácie spočíva v generovaní upozornení pre používateľov v prípade vzniku neočakávaných udalostí v systéme, ako napríklad prehrievanie, či nedostatok voľnej výpočtovej kapacity. Ako spôsob zasielania týchto upozornení sa ponúkajú napr. emailové správy. Užitočným rozšírením bude tiež zaznamenávanie a prehľadné zobrazovanie histórie vykonaných zmien v konfigurácii systému.

Literatúra

[1] Bzeznik, B., Henriot, O., Reis, V., Richard, O. a Tavard, L. Nix as HPC Package Management System. In: *Proceedings of the Fourth International Workshop on HPC User Support Tools*. New York, NY, USA: Association for Computing Machinery, 2017. HUST'17. DOI: 10.1145/3152493.3152556. ISBN 9781450351300. Dostupné z: https://doi.org/10.1145/3152493.3152556.

[2] Dolstra, E. *The Purely Functional Software Deployment Model*. Utrecht, 2006. 281 s. Dizertačná práca. Universiteit Utrecht. ISBN 90-393-4130-3.

[3] Dolstra, E. a Löh, A. NixOS: A Purely Functional Linux Distribution. *SIGPLAN Not.* New York, NY, USA: Association for Computing Machinery. september 2008, zv. 43, č. 9, s. 367–378. DOI: 10.1145/1411203.1411255. ISSN 0362-1340. Dostupné z: https://doi.org/10.1145/1411203.1411255.

[4] Dolstra, E., Löh, A. a Pierron, N. Nixos: A Purely Functional Linux Distribution. *Journal of Functional Programming*. Cambridge University Press. november 2010, zv. 20, 5–6, s. 577–615. DOI: 10.1017/S0956796810000195. ISSN 0956-7968.

[5] Dolstra, E. a Vermaas, R. *NixOps User's Guide* [online]. [cit. 2020-12-21]. Dostupné z: https://nixos.org/nixops/manual.

[6] Grafana Labs. *Get started or start exploring Grafana* [online]. [cit. 2021-2-14]. Dostupné z: https://grafana.com/docs.

[7] Initiative, O. S. *The MIT License* [online]. [cit. 2021-4-24]. Dostupné z: https://opensource.org/licenses/MIT.

[8] Majewsky, S. et al. *Raspberry Pi 4 support* [online]. [cit. 2021-1-24]. Dostupné z: https://github.com/NixOS/nixpkgs/issues/63720.

[9] NixOS. *Flakes- NixOS Wiki* [online]. [cit. 2021-02-12]. Dostupné z: https://nixos.wiki/wiki/Flakes.

[10] NixOS. Nix Package Manager Guide. *Nix 2.3.10 manual* [online]. [cit. 2020-12-10]. Dostupné z: https://nixos.org/manual/nix.

[11] NixOS. NixOS manual. *NixOS 20.09 manual* [online]. [cit. 2020-12-14]. Dostupné z: https://nixos.org/manual/nixos.

[12] NixOS. *NixOS on ARM - NixOS Wiki* [online]. [cit. 2020-12-21]. Dostupné z: https://nixos.wiki/wiki/NixOS_on_ARM.

[13] NixOS. *NixOS on ARM/Raspberry Pi 4 - NixOS Wiki* [online]. [cit. 2020-12-21]. Dostupné z: https://nixos.wiki/wiki/NixOS_on_ARM/Raspberry_Pi_4.

[14] Petitet, A., Whaley, R. C., Dongarra, J. a Cleary, A. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers* [online]. 2018 [cit. 2021-4-18]. Dostupné z: http://www.netlib.org/benchmark/hpl.

[15] PicoCluster LLC. *Pico 20* [online]. [cit. 2021-5-4]. Dostupné z: https://www.picocluster.com/collections/pico-20.

[16] Prometheus. *Overview | Prometheus* [online]. [cit. 2021-2-12]. Dostupné z: https://prometheus.io/docs/introduction/overview.

[17] Raspberry Pi Foundation. *About us* [online]. [cit. 2021-5-10]. Dostupné z: https://www.raspberrypi.org/about.

[18] Raspberry Pi Foundation. *Frequency management and thermal control* [online]. [cit. 2021-4-20]. Dostupné z: https://www.raspberrypi.org/documentation/hardware/raspberrypi/frequency-management.md.

[19] Scheepers, M. J. Virtualization and Containerization of Application Infrastructure: A Comparison. In: *21st Twente Student Conference on IT*. 2014, s. 1–7.

[20] Sindi, M. *Top500 HPL Calculator* [online]. [cit. 2021-4-29]. Dostupné z: http://hpl-calculator.sourceforge.net.

Prílohy

Zoznam príloh

Príloha A

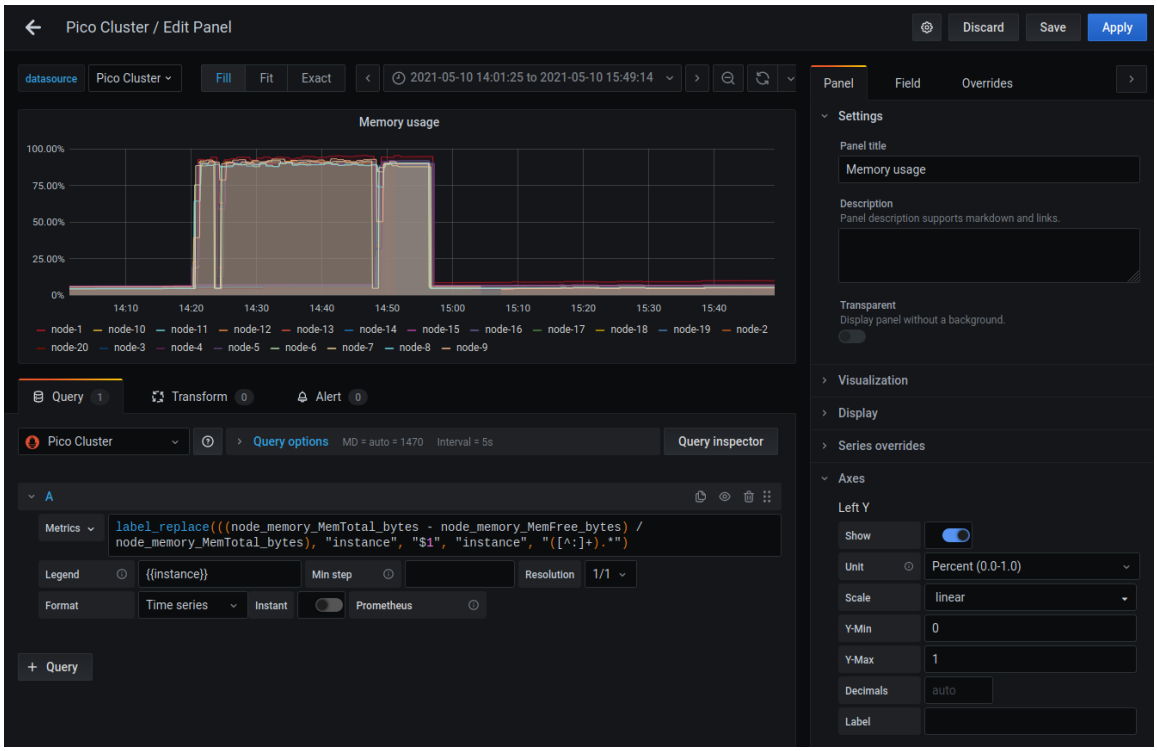
Obsah pamäťového média

K tejto práci je priložené pamäťové médium s touto štruktúrou súborov:

- `/nix-cluster-app/` – zdrojové súbory implementovaného systému
  - `master-node-config/` – konfiguračné súbbory hlavného uzla
  - `picocluster-config/` – príklady konfiguračných súborov pre nasadenia klastra
  - `picoapp/` – zdrojove súbory webovej aplikácie
- `/diplomova-praca/` – zdrojové súbory textovej časti diplomovej práce
- `/diplomova-praca.pdf` – technická správa vo formáte PDF

Príloha B

Snímka obrazovky aplikácie Grafana



Obr. B.1: Ukážka prostredia webovej časti aplikácie Grafana, v ktorej prebieha nastavovanie vizualizácii. V hlavnej časti okna je zobrazený graf vytvorený z dát získaných dotazom, ktorý je zapísaný v sekcii pod týmto grafom. V pravej časti aplikácie je panel pre nastavovanie názvu grafu, typu vizualizácie, parametrov osi a pod.

Príloha C

Návod na inštaláciu webovej aplikácie

Zdrojové súbory webovej aplikácie sú zverejnené ako open-source a dostupne na stránke https://github.com/adamzivcak/nix-cluster-app.

Požiadavky na hostiteľský systém

Hlavnou požiadavkou pre nasadenie aplikácie je operačný systém NixOS. Proces inštalácie aplikácie pracuje s variantom prázdneho systému hneď po jeho inštalácii. V prípade použitia v zabehnutom systéme je treba dbať na vzájomnú kompatibilitu nastavení systémov.

Proces inštalácie

V tejto sekcii budú v bodoch popísané kroky nasadenia webovej aplikácie na hostiteľský systém.

1. Stiahnutie repozitára so zdrojovými súbormi:

   ```
   git clone https://github.com/adamzivcak/nix-cluster-app.git
   ```

2. Nakopírovanie konfiguračných súborov z priečinka `master-node-config` do priečinka `/etc/nixos/`. V prípade zabehnutého systému je treba dať pozor hlavne na časť konfiguračného súboru s informáciami o súborovom systéme a prípadne vykonať ich zálohu.

   ```
   cp nix-cluster-app/master-node-config/* /etc/nixos
   ```

3. Úprava konfigurácie súborového systému a nastavenia siete tak, aby sa podsiete v konfigurácii zhodovali so skutočnosťou. Následne spustenie zostavenia systému.

   ```
   nixos-rebuild switch
   ```

4. Nakopírovanie zdrojových súborov aplikácie do zložky `/var/www/` a vstup do tejto zložky:

   ```
   cp -R nix-cluster-app/picoapp/ /var/www/
   cd /var/www/picoapp
   ```

5. Vytvorenie súboru pre nastavenie prostredia `.env` podľa vzoru `.env.example` a jeho úprava:

```
cp .env.example .env
nano .env
```

6. Spustenie inštalácie závislosti aplikácie:

```
composer install --optimize-autoloader --no-dev
```

7. Vygenerovanie náhodného kľúča pre aplikáciu:

```
php artisan key:generate
```

8. Počiatočné naplnenie databázy používateľov aplikácie:

```
php artisan migrate:refresh --seed
```

9. Optimalizácia načítavania:

```
php artisan route:cache
php artisan view:cache
```

Po vykonaní tejto série príkazov by mala byť aplikácia dostupná na IP adrese hostiteľského systému.

Do aplikácie sa je možné prihlásiť pomocou dvoch predvolených používateľských účtov: **admin** a **monitor**. Prístupové heslá k týmto účtom sú zhodné s prihlasovacími menami používateľov.