Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ

MATURITNÍ PRÁCE – HARDWARE A POČÍTAČOVÉ SÍTĚ

NÁVRH A REALIZACE OSOBNÍ TECHNOLOGICKÉ INFRASTRUKTURY

březen 2027 Ondřej Kučera, 4.D

„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.

Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“

V Pardubicích dne ....

podpis

TADY BUDE PRVNÍ STRANA ORIGINÁLU ZADÁNÍ

S PODPISEM ŘEDITELE

TADY BUDE DRUHÁ STRANA ORIGINÁLU ZADÁNÍ

S PODPISEM ŘEDITELE

Anotace

Práce byla vytvořena s cílem prokázat, že lze na vlastním či propůjčeném hardwaru sestavit moderní osobní řešení softwarových služeb, kterým mohou být v mnohých aspektech plně nahrazeny komerční cloudové služby. Komerčním řešením však nikterak není zabráněno využívat stejných principů. Klíčovým bodem bylo vytvoření silné bezpečnostní vrstvy mezi jednotlivými zařízeními, uživateli a konkrétními službami. Řešení je navrženo jako plně modulární a po zničení hardwaru může být obnoveno do původního stavu. V práci jsou podrobně popsány koncepty využité při realizaci a je předložen návod pro tvorbu obdobného řešení.

Klíčová slova: automatizace, infrastruktura, linux, mesh síť, monitoring, NixOS, server, ssh, síťové připojení, virtualizace, vzdálená plocha, vzdálený přístup, zálohování dat

Annotation

The thesis was created to demonstrate that a modern personal solution for software services can be built on one's own or borrowed hardware, by which commercial cloud services can be fully replaced in many aspects. Commercial solutions are, however, in no way prevented from using the same principles. The key element was the creation of a strong security layer between individual devices, users and specific services. The solution is designed to be fully modular and can be restored to its original state after the hardware is destroyed. The concepts used in the implementation are described in detail, and a guide for building a similar solution is provided.

Keywords: automation, data backup, infrastructure, linux, mesh network, monitoring, network connection, NixOS, remote access, remote desktop, server, ssh, virtualization

Obsah

ÚVOD....8

1 ANALÝZA....10

1.1 OPERAČNÍ SYSTÉMY....10

1.1.1 Linux Kernel....10

1.1.2 Distribuce OS....11

1.1.3 Drift konfigurace....11

1.1.4 Aktualizační modely....13

1.2 DEKLARATIVNÍ SESTAVOVÁNÍ....13

1.2.1 Imperativnost a deklarativnost....13

1.3 KOMBINACE DEKLARATIVNOSTI A OPERAČNÍHO SYSTÉMU....16

ZÁVĚR....19

SEZNAM POUŽITÉ LITERATURY A ZDROJŮ INFORMACÍ....20

SEZNAM POUŽITÝCH ZKRATEK....25

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ....26

PŘÍLOHY....27

Úvod

Infrastrukturou se rozumí soubor více než jednoho zařízení, která společně tvoří funkční celek.

V současné době je zcela běžné, že uživatel vlastní více než jedno elektronické zařízení zároveň. Z pohledu konzumenta takových prostředků, jakými mohou být mobilní telefon, nositelná elektronika a laptop, je běžnou praxí tendence přetrvávat u jednoho poskytovatele služeb či hardwaru, a to i po mnoho let či desetiletí. Pro uživatele je to nejsnazší cesta. Kompletní řešení nabízené technologickými firmami šetří uživatelův čas. Pro něj tak nevzniká povinnost učit se hlubší koncepty počítačových sítí, kybernetické bezpečnosti ani programovacích jazyků a porozumět jim. Pro firmy se jednotné nabízené řešení stává dobře marketingově propagovatelným (viz Apple Inc.). Za dobu prosazování konceptu centralizace si uživatelé od velkých korporací vybudovali očekávání luxusu v této kategorii. Uživatel se tak stává závislý na dodavateli služeb. Tento způsob čerpání výhod však má i svá úskalí.

Uzavřením do jednoho ekosystému se uživatel vystavuje nebezpečí nekontrolovaného nárůstu výdajů. Společnosti využívají centralizace k vytvoření psychické iluze a emocionální vazby k produktu či ekosystému. Často se tak v praxi setkáváme s tolerancí uživatele k nadsazeným cenám poskytovaných služeb a fenoménu „Nebudeš nic vlastnit. A budeš šťastný.“, tedy předpovědi Světového ekonomického fóra z roku 2016 [1], jejíž důsledky pro vlastnictví kriticky rozebírají Stewart, Charles a Page [2]. Rozpor mezi skutečným vlastnictvím a pouhým přístupem ke službě je přitom měřitelný. V reprezentativním experimentu s 1 299 účastníky se více než 83 % z nich po kliknutí na tlačítko „Buy Now“ považovalo za vlastníky zakoupeného digitálního obsahu, přestože jim licenční podmínky přiznávaly pouze omezené právo jej užívat [3].

Do jisté míry je zmíněným fenoménem zasažen každý z nás. V západní společnosti se za poslední dekádu setkáváme s normalizací trendu primárně u mladé generace, což si lze vysvětlit tím, že mladí lidé nemají možnost vlastnictví prožít. To je primárním důvodem, proč se tato práce zaobírá hlubšími koncepty self-hosted služeb, decentralizací a technologické nezávislosti.

V komerční sféře softwarových firem je obdobný přístup běžnou praxí. Firmy shledávají provoz vlastních služeb nejen finančně výhodnějším [4], ale i bezpečnějším a efektivnějším. V praxi k tomu využívají vlastní hardware, síťovou infrastrukturu a software. Firmám tento způsob nabízí absolutní kontrolu nad jejich fungováním. Takový přístup je však pro běžného uživatele příliš složitý a nákladný.

Cílem této práce je nabídnout uživateli možnost sestavit si vlastní infrastrukturu, která bude plně funkční a bezpečná, a to i na základě vlastního hardwaru. Práce je shrnutím autorovy cesty s deklarativními systémy včetně blíže rozvedených rozhodnutí, vyvstalých úskalí a klíčových výhod. Blíže je popsán návrh, realizace a testování.

1 Analýza

Výběr technologie pro tvorbu infrastruktury se stává již zprvu nejklíčovějším rozhodnutím, protože další postup navazuje. Ačkoliv se rozdíly v technologiích nemusejí zdát velké, chyba při výběru může vést ke zdlouhavému procesu migrace dat a služeb na jiný substrát (rozumí se platformu) [5]. Vyvarování se migraci vhodným výběrem je v praxi preferováno, protože se tím šetří náklady, ale ne úplně běžné. Technologie se ve svých vlasnostech překrývají. Architekt tak musí pochopit kladené požadavky a nároky na infrastrukturu, zpracovat je, predikovat budoucí vývoj, a občasně i tvořit kompromisy.

Řešení je mnoho. Problém přetrvává od vzniku internetu a nepolevuje, naopak se škáluje, a to jak horizontálně (výpočetní výkon v datacentrech pokrývá větší geografickou plochu, větší počet datacenter), tak i vertikálně (jednotlivé uzly jsou výkonnější, mají rychlejší paměť i propojení) [6]. Přenosy dat tak mnohdy dosahují až TB/s, viz ultrazvukový skener Midjourney Medical [7], nebo současná AI datacentra [8] s akcelerátory TPU [9] či NPU [10]. Software se musí nárokům přizpůsobit. Může se zdát, že problém je řešitelný pouze na straně síťového připojení, kde jsou kladeny extrémní nároky na přenosová média (optické kabely, metalické spoje, on-die čipové spoje[1]) [11]. Pokud se však opomene software, vyvstane problém škálovatelnosti mnohem obtížněji řešitelný.

Vytvoření systému, který by se dokázal dle potřeby a vytíženosti adaptovat [12] a zároveň by byl plně deterministický [13], je těžkým inženýrským problémem, jehož existenci potvrzuje mnoho lepších i horších odpovědí.

1.1 Operační systémy

1.1.1 Linux Kernel

Operační systém (OS) tvoří vrstvu mezi hardwarem a aplikacemi. Úkolem OS je zajistit, aby aplikace mohly sdílet procesor (CPU), operační paměť (RAM), disk i síťové

[1] On-die spoje tvoří propojení mezi klíčovými komponenty přímo v čipu pro extrémně velký objem přenesených dat ve vyšších řádech stovek GB/s.

rozhraní, aniž by o sobě musely nutně vzájemně vědět. Základem systémů, kterými se tato práce zabývá, je jádro Linux (kernel). [14].

Linux je dominantem především na serverech. Podle průzkumu běží na Linuxu 63,7 % webových stránek, u nichž je OS serveru znám [15], a v žebříčku 500 nejvýkonnějších superpočítačů světa jej využívají všechny [16]. Na stolních počítačích je trend opačný, Linuxu připadá jen asi 2 % provozu [17]. Kernel však používá i Android [18], který je s podílem 44,6 % nejrozšířenějším operačním systémem na světě [17].

1.1.2 Distribuce OS

Samotný kernel však k provozu nestačí. Obsluhuje hardware a spouští procesy. O tom, jaké programy jsou nainstalovány, jejich konkrétní verze a jak jsou nastaveny, ale nerozhoduje. To je úkolem distribuce[2], jakou je například Ubuntu, nebo Fedora. Ty na kernelu staví. Rozdíly nevznikají v kernelu samotném, ale ve způsobu, jakým distribuce spravuje vše, co nad ním běží.

Nejrozšířenější je imperativní[3] způsob správy distribuce. Pro demonstrační účely lze využít OS Ubuntu na nejž se aplikace a jiný software instalují a aktualizují s pomocí nástroje APT. [19]. Každý příkaz mění soubory v systému za běhu na místě. Obdobné použití aplikuje i OS Windows: instalátory aplikací, systémové aktualizace i změny nastavení se postupně vrší. Výsledný stav počítače tak není nikde centralizovaně popsán. Je dán pouze výchozí instalací a celou historií změn, kterou uživatel v průběhu užívání uskutečnil.

1.1.3 Drift konfigurace

Problém vyvstane ve chvíli, kdy je strojů více než jeden. Dva počítače nainstalované ze stejného média se po několika měsících běžného používání téměř jistě liší, protože každý prošel jinou posloupností aktualizací, instalací a oprav. Tento jev se

[2]Celek složený z jádra, základních nástrojů, správce balíčků a výchozí konfigurace.

[3]Sekvence příkazů $p_1, ..., p_n$ aplikovaná na aktuální stav: $S_n = p_n(...p_2(p_1(S_0)))$. Výsledek závisí na výchozim stavu S_0 a na pořadí příkazů.

označuje jako drift konfigurace (z angl. *configuration drift*). Důsledkem je, že nikdo nedokáže s jistotou říct, co přesně na daném stroji je a hlavně, proč.

S tím souvisí i obnova. Pokud aktualizace neproběhne, jak se předpokládalo, imperativní systém obvykle nenabízí jednoduchou cestu zpět. Předchozí verze souborů byly přepsány. Windows tento problém částečně řeší funkcí Obnovení systému, která pracuje s body obnovy[4].

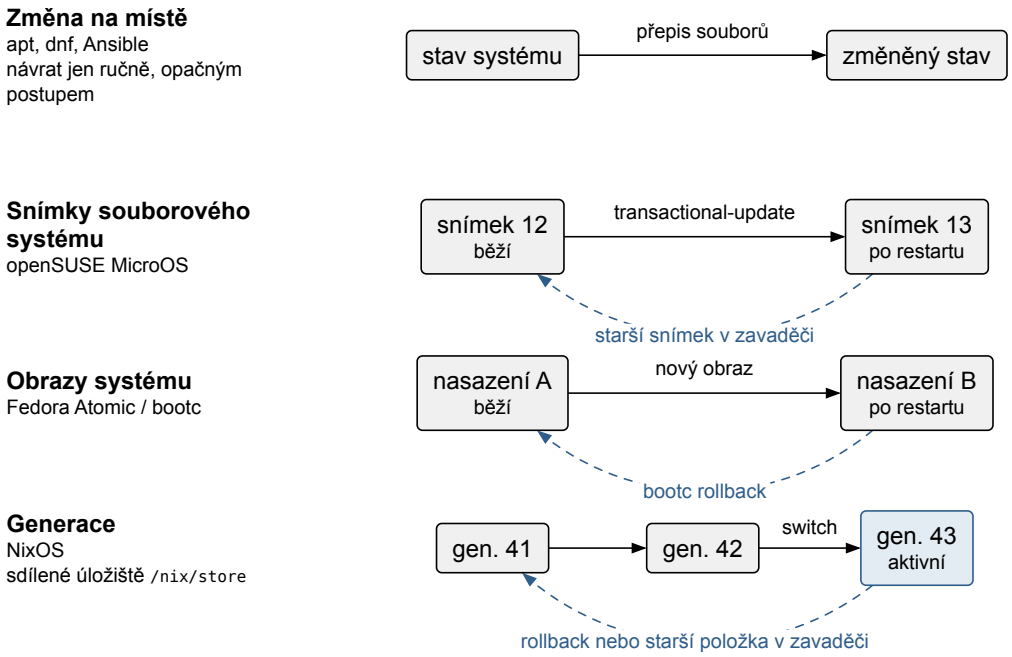
Přirozenou reakcí na drift by mohl být zápis postupu do předpisu, který lze spustit na libovolném počtu strojů. Tímto směrem jdou nástroje pro správu konfigurace, z nichž nejznámější je Ansible. Základem je idempotence[5]. Dokumentace Ansible uvádí, že uvede systém do požadovaného stavu bez ohledu na aktuální stav, místo aby uživatel musel popisovat, jak se k němu dostat [21]. Ansible tedy není čistě imperativní. Požadovaný stav ale popisuje pouze pro ty části systému, které jsou v předpisu uvedeny. Co bylo na stroj nainstalováno mimo předpis, Ansible neodstraní, protože o tom neví.

Jiná skupina systémů řeší obnovu uchováváním předchozích stavů. Distribuce openSUSE MicroOS využívá nástroj transactional-update. Před každou aktualizací vytvoří pomocí nástroje snapper nový snímek kořenového souborového systému a aktualizaci provede v něm. Změny se projeví až po restartu a příkaz `rollback` umožňuje nastavit jako výchozí některý z předchozích snímků [22]. Projekt bootc, na kterém staví Fedora Atomic, jde ještě o kus dál a aktualizuje celý operační systém pomocí obrazů kontejnerů ve formátu OCI [23]. Nová verze systému se připraví jako další nasazení vedle stávajícího a příkaz `bootc rollback` pouze změní pořadí položek v zavaděči, takže se při dalším startu spustí nasazení předchozí [24].

[4](z angl. *snapshot*), Snímky konfigurace a nastavení systému v určitém okamžiku [20]

[5]Binární operace je idempotentní, pokud je výsledek jednoho jejího provedení stejný jako výsledek opakovaného provedení: $x * x = x \quad \forall x \in S$

1.1.4 Aktualizační modely



Obrázek 1: Modely aktualizace a obnovy stavu operačního systému. Zdroj: vlastní zpracování dle [22], [24] a [25].

Rozdíly mezi popsanými přístupy řešení driftu konfigurace shrnuje obrázek 1. Snímky i obrazy přinášejí to, co klasickým systémům chybí, to jest atomickou změnu a možnost návratu. Stále však platí, že systém nepopisují, pouze uchovávají jeho stavy. Snímek je kopií výsledku, nikoli předpisem, jak k němu dojít, a obraz systému vzniká sestavením, které samo může být imperativní.

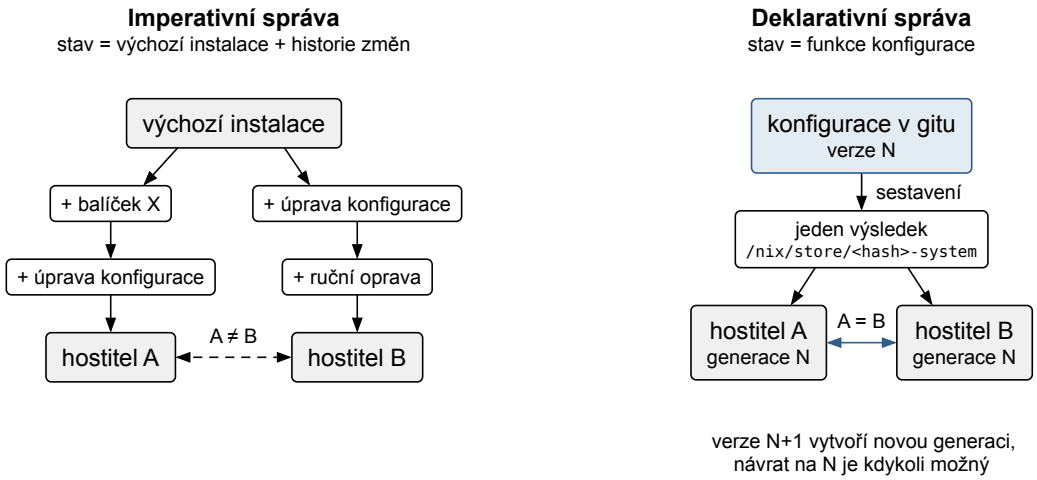
1.2 Deklarativní sestavování

1.2.1 Imperativnost a deklarativnost

Rozdíl mezi imperativním a deklarativním[6] přístupem lze shrnout jednou otázkou: popisuji, jak se k výsledku dostat, nebo jaký má výsledek být? Imperativní předpis je posloupnost kroků (nainstaluj, uprav, restartuj), jejichž výsledek závisí na tom, v jakém

[6]Systém S je derivován z konfigurace K funkcí: $S = \text{sestav}(K)$. Aplikování funkce vrátí jakýkoliv výchozí stav, $\text{aplikuj}(S_0, K) = \text{sestav}(K) \quad \forall S_0$, opakovaným spuštěním se nic nezmění: $\text{aplikuj}(\text{aplikuj}(S_0, K), K) = \text{aplikuj}(S_0, K)$.

stavu systém na začátku byl. Deklarativní předpis popisuje požadovaný výsledek a cestu k němu ponechává nástroji. Dokumentace Kubernetes tento rozdíl vysvětluje na příkladu termostatu: uživatel nastaví požadovanou teplotu, tedy požadovaný stav, a termostat zapíná a vypíná topení tak, aby se aktuální stav požadovanému přiblížil [26].



Obrázek 2: Imperativní a deklarativní správa více hostitelů. Zdroj: vlastní zpracování.

Pro správu více strojů má tento rozdíl zásadní důsledek, který ilustruje obrázek 2. Při imperativní správě je stav každého stroje dán jeho historií, a dva stroje se proto mohou lišit, i když vznikly ze stejné instalace. Při deklarativní správě je stav funkcí konfigurace: ze stejné konfigurace vznikne stejný výsledek, a pokud stroje sdílejí konfiguraci, sdílejí i stav. To ovšem platí jen za předpokladu, že je samotné sestavení reprodukovatelné.

Projekt Reproducible Builds definuje sestavení jako reprodukovatelné, pokud může kdokoli ze stejného zdrojového kódu, stejného sestavovacího prostředí a stejných instrukcí znovu vytvořit bit po bitu identické kopie všech určených výstupů [27]. Význam této vlastnosti přesahuje pohodlí správce: pokud lze výsledek sestavení nezávisle ověřit, zvyšuje se důvěryhodnost celého dodavatelského řetězce softwaru [28].

Aby bylo sestavení reprodukovatelné, nesmí jeho výsledek záviset na ničem, co není výslovně deklarováno. Tuto vlastnost označuje dokumentace nástroje Bazel jako hermetičnost: hermetický sestavovací systém při stejném vstupním kódu a stejné konfiguraci vždy vrátí stejný výstup, protože sestavení izoluje od změn hostitelského systému [29]. Pokud by sestavení mohlo nahlédnout do hostitelského systému (například použít knihovnu, která na něm náhodou je nainstalována), jeho výsledek by se lišil podle toho, na jakém stroji proběhlo.

Dobrým příkladem, že deklarativní vzhled ještě neznamená deklarativní chování, je Dockerfile. Na první pohled popisuje obraz kontejneru: z jakého základního obrazu vychází, jaké soubory obsahuje a jaký program se v něm spustí. Základní obraz se ale obvykle uvádí pomocí značky (angl. *tag*), která není neměnná. Dokumentace Dockeru proto doporučuje připnout obraz ke konkrétnímu otisku (angl. *digest*), protože jedině tak je zaručeno použití stále stejné verze, i když vydavatel pod stejnou značku vydá obraz nový [30]. Obdobně se chovají i instrukce `RUN`, které typicky spouštějí správce balíčků (například `apt-get install`) a stahují balíčky v té verzi, která je dostupná v okamžiku sestavení. Tato druhá část je mou vlastní úvahou, nikoli tvrzením dokumentace. Dockerfile je tak ve skutečnosti spíše skriptem, jehož výsledek se ukládá po vrstvách, a stejný Dockerfile může za měsíc vytvořit jiný obraz.

Kubernetes je deklarativní v plném smyslu slova, ale na jiné úrovni. Objekty v něm lze vytvářet, aktualizovat a mazat uložením konfiguračních souborů do adresáře a jejich předáním příkazu `kubectl apply` [31] a takzvané kontrolery pak průběžně přibližují aktuální stav požadovanému [26]. Kubernetes však spravuje běžící kontejnery, nikoli sestavení softwaru ani operační systém strojů, na kterých kontejnery běží. Stejný princip, tedy deklarativní popis uložený ve verzovaném úložišti, který si agenti sami stahují a průběžně s ním srovnávají skutečný stav, formalizuje přístup GitOps [32]. K němu se vrátím při popisu nasazování vlastní infrastruktury.

Bazel naopak řeší právě sestavení. Izoluje ho od hostitelského systému [29], operační systém ani jeho konfiguraci však nespravuje, protože to není jeho účelem.

Zda deklarativní a hermetický přístup funguje i ve velkém, zkoumala studie, která znovu sestavila 709 816 balíčků z historických snímků repozitáře nixpkgs z let 2017 až 2023. Bitově identický výsledek autoři získali v 69 až 91 % případů s rostoucím trendem a úspěšnost samotného opětovného sestavení přesáhla 99 % [33]. Přibližně 15 % neúspěchů přitom způsobilo pouhé vložení data sestavení do výsledku.

Tabulka 1: Role ve správě infrastruktury a nástroje, které je pokrývají. Zdroj: vlastní zpracování.

Role	Nix / NixOS	Specializované alternativy
Správa balíčků	nixpkgs	apt / dnf · Homebrew · conda / pixi
Sestavování softwaru	derivace, `nix build`	Bazel · Buck2 · Dockerfile
Vývojové prostředí	`nix develop`	Dev Containers · mise / asdf
Konfigurace systému	moduly NixOS	Ansible · Puppet · Salt
Neměnný systém a návrat	generace NixOS	Fedora Atomic / bootc · openSUSE MicroOS · Talos
Nasazování na více strojů	deploy-rs, comin	Kubernetes · Ansible · bootc switch

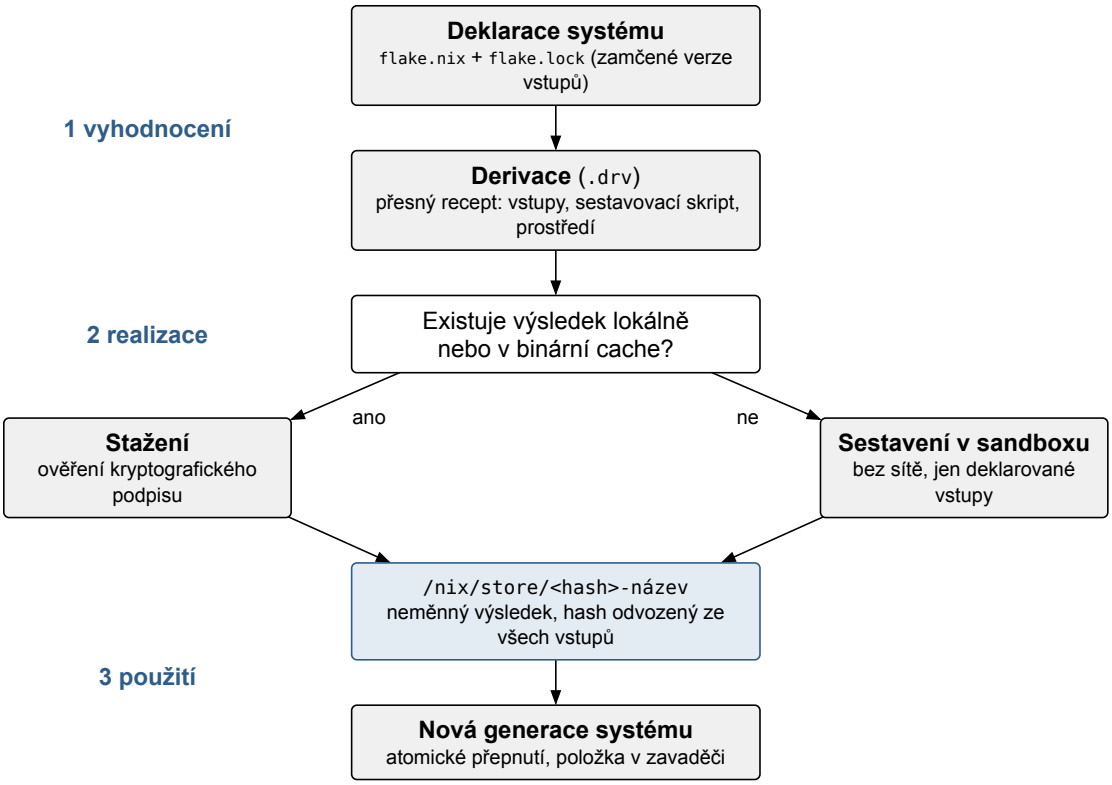
Přehled nástrojů zmíněných v této a předchozí kapitole podává tabulka 1. Je z ní patrné, že každý z nich pokrývá jednu roli: Ansible konfiguraci systému, Bazel a Dockerfile sestavení, Kubernetes nasazování, bootc neměnný systém. Jedinými nástroji, které tyto role spojují do jednoho modelu, jsou Nix a jemu příbuzný GNU Guix. Jejich spojení s operačním systémem se věnuje následující kapitola.

1.3 Kombinace deklarativnosti a operačního systému

Nix vznikl na Univerzitě v Utrechtu. Jeho autoři ho v roce 2004 představili jako systém pro nasazování softwaru, který řeší dva hlavní problémy tehdejších nástrojů: nemožnost spolehlivě vynutit popis závislostí mezi komponentami a chybějící podporu pro více verzí nebo variant téže komponenty. Řešení stojí na jednoduché myšlence, totiž na použití kryptografických hashů k výpočtu jedinečné cesty pro každou instanci komponenty [34]. Celý model pak Eelco Dolstra podrobně rozpracoval ve své disertační práci [35].

Každý výsledek sestavení se ukládá do adresáře `/nix/store` pod cestou, jejíž součástí je zmíněný hash. Změní-li se vstup, vznikne nová cesta a ta původní zůstane nedotčena. Díky tomu lze mít nainstalováno více verzí nebo variant téhož balíčku současně, a protože Nix stávající balíčky nikdy nepřepisuje, ale pouze přidává nové verze do jiných cest, jsou jeho operace atomické a lze se vrátit k verzi předchozí [36]. Pokud už výsledek existuje v binární cache, tedy na webovém serveru s předem sestavenými balíčky, Nix ho místo sestavování stáhne [36], přičemž stažený obsah musí být podepsán důvěryhodným klíčem [37].

Pokud výsledek v cache není, Nix ho sestaví sám. Sestavení přitom probíhá v izolovaném prostředí (angl. *sandbox*), ve kterém vidí pouze své závislosti v `/nix/store`, dočasný adresář pro sestavení a soukromé verze několika systémových adresářů. Na Linuxu navíc běží ve vlastních jmenných prostorech, mimo jiné i v síťovém, a k síti tak nemá přístup. Výjimkou jsou pouze derivace s pevně daným výstupem, typicky stažení zdrojového kódu [37]. Celý proces od deklarace po hotový výsledek znázorňuje obrázek 3.



Obrázek 3: Cesta od deklarace k nové generaci systému v Nix. Zdroj: vlastní zpracování dle [35].

Pro opakovatelnost je klíčové, aby se neměnily ani samotné vstupy, například repozitář nixpkgs s definicemi balíčků. K tomu slouží takzvané flakes: Nix pro zajištění reprodukovatelnosti automaticky vytváří a používá soubor `flake.lock`, do kterého zapíše uzamčené specifikace všech vstupů [38]. Aktualizace pak znamená vědomou změnu tohoto souboru, kterou lze, stejně jako jakoukoli jinou změnu v repozitáři, kdykoli vrátit. Je ovšem třeba zmínit, že příkaz `nix flake` je dosud označen jako experimentální a jeho rozhraní se může měnit [38].

NixOS přenáší stejný princip z jednotlivých balíčků na celý operační systém. Jeho autoři vycházejí z toho, že stávající nástroje pro správu balíčků a konfigurace trpí imperativním modelem, ve kterém akce správce destruktivně mění stav systému.

To vede k problémům, jako je nemožnost snadno vrátit změny, provozovat více verzí balíčku vedle sebe, deterministicky zopakovat konfiguraci na jiném stroji nebo spolehlivě aktualizovat systém. NixOS proto všechny statické části systému, tedy balíčky, konfigurační soubory i spouštěcí skripty, sestavuje čistými funkcemi a ukládá je jako neměnné [13].

Výsledkem je, že každá sestavená konfigurace je samostatnou generací systému. Do předchozí konfigurace se lze přepnout za běhu příkazem `nixos-rebuild switch --rollback`, případně ji vybrat v nabídce zavaděče při startu a příkazem `switch-to-configuration boot` ji nastavit jako výchozí [25]. NixOS tak spojuje obě vlastnosti, které se v předchozích kapitolách vyskytovaly odděleně: systém je popsán deklarací (jako u Kubernetes, nebo při disciplinovaném použití u Ansible) a zároveň lze kdykoli atomicky přejít na jinou verzi (jako u snímků nebo obrazů v obrázku 1).

Stejný model přebírá i GNU Guix, čistě funkcionální správce balíčků určený pro celou distribuci GNU/Linux. Staví na nízkoúrovňové vrstvě pro sestavování a nasazování ze systému Nix a jako programovací rozhraní používá jazyk Scheme [39]. Transakčním aktualizacím a návratu podléhá i celková konfigurace systému Guix [40].

K Nixu jsem se ale nedostal systematickým srovnáním všech výše popsaných možností. Doporučil mi ho kolega a k celému přístupu jsem se tak dostal spíše náhodou. Výhody popsané v této kapitole jsem zjišťoval až postupně, při jeho používání. Analýzu v této kapitole proto nelze číst jako zdůvodnění rozhodnutí, které předcházelo realizaci. Je spíše zpětným ověřením, že volba, ke které jsem dospěl zčásti náhodou, obstojí i ve srovnání s alternativami.

Pro úplnost je třeba zmínit i nevýhody. Jazyk Nix a celý model vyžadují nemalé úsilí, než jim člověk porozumí. Rozhraní flakes je, jak bylo zmíněno, stále experimentální [38]. A konečně, neměnné jsou podle autorů NixOS statické části systému [13]. Data, která systém vytváří za běhu (například databáze nebo soubory uživatelů), generace nepokrývají a vyžadují vlastní řešení. Jak tyto principy využívám při stavbě vlastní infrastruktury, včetně síťového propojení jednotlivých strojů a zálohování dat, popisuji v následujících kapitolách.

Závěr

Popište význam projektu, zdůvodněte problémy při řešení daného tématu a odchylky od zadání, popište využití maturitní práce a plány do budoucna.

Seznam použité literatury a zdrojů informací

[1] WORLD ECONOMIC FORUM. *8 predictions for the world in 2030* [online]. 12. listopad 2016 [vid. 2026-10-23]. Dostupné z: https://www.weforum.org/agenda/2016/11/8-predictions-for-the-world-in-2030/

[2] STEWART, Richard, Michael B. CHARLES a John PAGE. A future with no individual ownership is not a happy one: Property theory shows why. *Futures* [online]. 2023, **152**, 103209. ISSN 0016-3287. Dostupné z: doi:10.1016/j.futures.2023.103209

[3] PERZANOWSKI, Aaron a Chris Jay HOOFNAGLE. What we buy when we "buy now". *University of Pennsylvania Law Review* [online]. 2017, **165**, 315–378 [vid. 2026-10-23]. Dostupné z: https://ssrn.com/abstract=2778072

[4] WANG, Sarah a Martin CASADO. *The Cost of Cloud, a Trillion Dollar Paradox* [online]. 27. květen 2021 [vid. 2026-11-12]. Dostupné z: https://a16z.com/the-cost-of-cloud-a-trillion-dollar-paradox/

[5] OPARA-MARTINS, Justice, Reza SAHANDI a Feng TIAN. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing* [online]. 2016, **5**(1), 4. ISSN 2192-113X. Dostupné z: doi:10.1186/s13677-016-0054-z

[6] MICHAEL, Maged et al. Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: *2007 IEEE International Parallel and Distributed Processing Symposium* [online]. B.m.: IEEE, 2007, s. 1–8. Dostupné z: doi:10.1109/IPDPS.2007.370631

[7] MIDJOURNEY. *Frequently Asked Questions — Midjourney Medical* [online]. [vid. 2026-11-12]. Dostupné z: https://www.midjourney.com/medical/faq

[8] VAHDAT, Amin. *Ironwood: The first Google TPU for the age of inference* [online]. 9. duben 2025 [vid. 2026-11-12]. Dostupné z: https://blog.google/products/google-cloud/ironwood-tpu-age-of-inference/

[9] JOUPPI, Norman P. et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture* [online]. New York: ACM, 2017, s. 1–12. Dostupné z: doi:10.1145/3079856.3080246

[10] LIAO, Heng et al. Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* [online]. B.m.: IEEE, 2021, s. 789–801. Dostupné z: doi:10.1109/HPCA51647.2021.00071

[11] POUTIEVSKI, Leon et al. Jupiter evolving. In: *Proceedings of the ACM SIGCOMM 2022 Conference* [online]. New York: ACM, 2022, s. 66–85. Dostupné z: doi:10.1145/3544216.3544265

[12] VERMA, Abhishek et al. Large-scale cluster management at Google with Borg. In: *Proceedings of the Tenth European Conference on Computer Systems* [online]. New York: ACM, 2015, s. 1–17. Dostupné z: doi:10.1145/2741948.2741964

[13] DOLSTRA, Eelco, Andres LÖH a Nicolas PIERRON. NixOS: A purely functional Linux distribution. *Journal of Functional Programming* [online]. 2010, **20**(5–6), 577–615. ISSN 0956-7968. Dostupné z: doi:10.1017/S0956796810000195

[14] THE LINUX KERNEL DOCUMENTATION. *Linux kernel release 6.x* [online]. [vid. 2026-11-12]. Dostupné z: https://docs.kernel.org/admin-guide/README.html

[15] W3TECHS. *Usage statistics and market share of Linux for websites* [online]. [vid. 2026-11-12]. Dostupné z: https://w3techs.com/technologies/details/os-linux

[16] TOP500. *List Statistics: Operating system Family, June 2026* [online]. [vid. 2026-11-12]. Dostupné z: https://www.top500.org/statistics/list/

[17] STATCOUNTER GLOBAL STATS. *Operating System Market Share Worldwide* [online]. [vid. 2026-11-12]. Dostupné z: https://gs.statcounter.com/os-market--share

[18] ANDROID OPEN SOURCE PROJECT. *Kernel overview* [online]. [vid. 2026-11-12]. Dostupné z: https://source.android.com/docs/core/architecture/kernel

[19] UBUNTU SERVER DOCUMENTATION. *Install and manage packages* [online]. [vid. 2026-11-12]. Dostupné z: https://documentation.ubuntu.com/server/how-to/software/package-management/

[20] MICROSOFT. *System Restore* [online]. [vid. 2026-11-12]. Dostupné z: https://support.microsoft.com/en-us/windows/use-system-restore-a5ae3ed9-07c4-fd56--45ee-096777ecd14e

[21] ANSIBLE. *Glossary* [online]. [vid. 2026-11-12]. Dostupné z: https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html

[22] OPENSUSE. *transactional-update(8)* [online]. [vid. 2026-11-12]. Dostupné z: https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html

[23] BOOTC-DEV. *bootc: Transactional, in-place operating system updates using OCI/Docker container images* [online]. [vid. 2026-11-12]. Dostupné z: https://github.com/bootc-dev/bootc

[24] BOOTC. *bootc-rollback(8): Change the bootloader entry ordering* [online]. [vid. 2026-11-12]. Dostupné z: https://bootc.dev/bootc/man/bootc-rollback.8.html

[25] NIXOS. *NixOS Manual* [online]. [vid. 2026-11-12]. Dostupné z: https://nixos.org/manual/nixos/stable/

[26] KUBERNETES. *Controllers* [online]. [vid. 2026-11-12]. Dostupné z: https://kubernetes.io/docs/concepts/architecture/controller/

[27] REPRODUCIBLE BUILDS. *Definitions* [online]. [vid. 2026-11-12]. Dostupné z: https://reproducible-builds.org/docs/definition/

[28] LAMB, Chris a Stefano ZACCHIROLI. Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software* [online]. 2022, **39**(2), 62–70. ISSN 0740-7459. Dostupné z: doi:10.1109/MS.2021.3073045

[29] BAZEL. *Hermeticity* [online]. [vid. 2026-11-12]. Dostupné z: https://bazel.build/basics/hermeticity

[30] DOCKER. *Building best practices* [online]. [vid. 2026-11-12]. Dostupné z: https://docs.docker.com/build/building/best-practices/

[31] KUBERNETES. *Declarative Management of Kubernetes Objects Using Configuration Files* [online]. [vid. 2026-11-12]. Dostupné z: https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/

[32] OPENGITOPS. *GitOps Principles v1.0.0* [online]. [vid. 2026-11-12]. Dostupné z: https://opengitops.dev/

[33] MALKA, Julien, Stefano ZACCHIROLI a Théo ZIMMERMANN. Does Functional Package Management Enable Reproducible Builds at Scale? Yes. In: *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)* [online]. B.m.: IEEE, 2025, s. 775–787. Dostupné z: doi:10.1109/MSR66628.2025.00115

[34] DOLSTRA, Eelco, Merijn de JONGE a Eelco VISSER. Nix: A Safe and Policy--Free System for Software Deployment. In: *18th Large Installation System Administration Conference (LISA '04)* [online]. B.m.: USENIX Association, 2004, s. 79–92. Dostupné z: https://www.usenix.org/conference/lisa-04/nix-safe--and-policy-free-system-software-deployment

[35] DOLSTRA, Eelco. *The Purely Functional Software Deployment Model* [online]. Utrecht, 2006. Doctoral dissertation. b.n. Dostupné z: https://edolstra.github.io/pubs/phd-thesis.pdf

[36] NIXOS. *Nix Reference Manual* [online]. [vid. 2026-11-12]. Dostupné z: https://nix.dev/manual/nix/stable/

[37] NIXOS. *nix.conf* [online]. [vid. 2026-11-12]. Dostupné z: https://nix.dev/manual/nix/stable/command-ref/conf-file.html

[38] NIXOS. *nix flake* [online]. [vid. 2026-11-12]. Dostupné z: https://nix.dev/manual/nix/stable/command-ref/new-cli/nix3-flake.html

[39] COURTÈS, Ludovic. *Functional Package Management with Guix* [online]. 20. květen 2013 [vid. 2026-11-12]. Dostupné z: https://arxiv.org/abs/1305.4584

[40] GNU GUIX. *Features (GNU Guix Reference Manual)* [online]. [vid. 2026-11-12]. Dostupné z: https://guix.gnu.org/manual/en/html_node/Features.html

Seznam použitých zkratek

AI	Artificial Intelligence – umělá inteligence
APT	Advanced package tool - pokročilý balíčkový nástroj, užívaný distribucí Ubuntu
CPU	Central Processing Unit - procesor
GB	Gigabyte - jednotka dat: $1\ \mathrm{GB} = 10^9\ \mathrm{B}$
NPU	Neural Processing Unit – neuronová procesorová jednotka, procesor specializovaný na výpočty neuronových sítí
OCI	Open Container Initiative - otevřená kontejnerová iniciativa, soubor standardů pro kontejnerizaci systémů a aplikací
OS	Operační Systém
RAM	Random Access Memory - operační paměť
TB	Terabyte - jednotka dat: $1\ \mathrm{TB} = 10^{12}\ \mathrm{B}$
TPU	Tensor Processing Unit – tenzorová procesorová jednotka, specializovaný čip společnosti Google pro urychlení výpočtů AI

Seznam obrázků, tabulek, grafů

Obrázek 1: Modely aktualizace a obnovy stavu operačního systému. Zdroj: vlastní zpracování dle [22], [24] a [25].....13

Obrázek 2: Imperativní a deklarativní správa více hostitelů. Zdroj: vlastní zpracování… 14

Tabulka 1: Role ve správě infrastruktury a nástroje, které je pokrývají. Zdroj: vlastní zpracování.....16

Obrázek 3: Cesta od deklarace k nové generaci systému v Nix. Zdroj: vlastní zpracování dle [35].....17

Přílohy

Use case diagram