Sem vložte zadání Vaší práce.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Podpora automatické správy virtualizačního kontejneru Solaris Zones na platformě Solaris

Bc. Tomáš Šimáček

Katedra počítačových systémů
Vedoucí práce: Ing. Michal Šoch, Ph.D.

4. května 2018

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2018 .....

Odkaz na tuto práci

Šimáček, Tomáš. *Podpora automatické správy virtualizačního kontejneru Solaris Zones na platformě Solaris*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Abstract v češtině

Klíčová slova Solaris, Solaris Zones, virtualizace, automatická správa

Abstract

Abstract in english

Keywords Solaris, Solaris Zones, virtualization, automatic management

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódů

Úvod

Virtualizace je technika, se kterou se dnes v IT můžeme setkat v mnoha podobách. Jednou z hlavních oblastí využití, je virtualizace serverů. Virtualizace se také objevuje i v oblasti komunikačních sítí nebo desktopů. Tato technologie umožňuje vytvářet virtuální prostředky a poskytovat tak kompletní virtuální prostředí. Toto prostředí umožňuje provozovat systémy na jiných fyzických architekturách, než pro jaké jsou určeny.

Hlavním tématem této práce je virtualizace serverů, která umožňuje rozdělit jeden fyzický systém do několika nezávislých virtuálních prostředí, ve který jsou spouštěny virtuální počítače. Možnost vytváření virtuálních počítačů v rámci jednoho fyzického systému značně snižuje náklady na pořizování a provoz fyzických serverů. Díky virtualizaci již není třeba pořizovat dedikovaný server pro každou instanci operačního systému, který chce společnost provozovat. Správným rozdělením virtuálních počítačů na fyzické servery může být docíleno ideální rozdělení zátěže a tím mohou být dostupné fyzické prostředky efektivně využity.

Rostoucí počet virtualizovaných serverů může mít za následek obtížnější správu. Automatizované nasazení, instalace nebo zálohování virtuálních počítačů může být značným ulehčením správy počítačové infrastruktury, která využívá virtualizačních technik. Díky tomuto ulehčení lze jednoduše vytvářet předem definovaná virtuální prostředí, která mohou složit pro vývoj software, testování nebo nasazení aplikací do produkčního prostředí.

V dnešní době existuje mnoho operačních systémů, které nějakým způsobem poskytují virtualizaci v rámci svých služeb. Jedním ze zástupců takovýchto systémů je operační systém Solaris. Exkluzivně pro tento operační systém byla vytvořena virtualizační technika Solaris Zones, která umožňuje v rámci jedné instance operačního systému Solaris vytvářet virtuální počítače nazývané zóny. Nástroje pro správu této virtualizační technologie umožňují manipulovat se zónami pouze v rámci lokálního serveru. Tato diplomová práce se věnuje automatizaci procesu vytváření, zálohy a migrace těchto zón. Poskytuje také služby pro správu zón, které se nacházejí na vzdálených serverech.

Cíle práce

Cílem této diplomové práce je seznámení s operačním systémem Solaris a jeho aktuální stabilní verzí 11.3. Součástí popisu tohoto operačního systému je i představení podporovaných architektur a jeho základních služeb pro správu softwarových balíčku nebo souborového systému ZFS. Především jde však o popis základních principů virtualizační techniky Solaris Zones, která umožňuje běh více zón v rámci jedné instance operačního systému Solaris. Nebude chybět ani porovnání běžně používaných virtualizačních technik.

Diplomové práce také souvisí s virtualizační technikou Solaris Zones a má za úkol detailně popsat možnosti konfigurace a instalace zón. Součástí tohoto popisu bude i popis administrátorských procesů pro zálohování, obnovu a migraci. Popis se také věnuje integraci techniky Solaris Zones s ostatními službami operačního systému Solaris.

Hlavním cílem této diplomové práce je návrh a implementace nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones na platformě Solaris. Jelikož základní nástroje pro správu této virtualizační technologie neumožňují správu vzdálených zón, implementovaný nástroj bude tuto funkcionalitu podporovat. Dále tento nástroj bude umožňovat provádění základních administrátorských procesů pro větší množství lokálních i vzdálených zón. Mezi těmito procesy bude zahrnuta automatická konfigurace, instalace, náhrada, záloha, obnova a migrace zón v rámci několika virtualizačních serverů. Nástroj bude umožňovat definici softwarových balíků, které mají být při instalaci do zóny zahrnuty a to pomocí šablon nebo interaktivně pomocí uživatelského rozhraní.

Posledním cílem této diplomové práce je otestovat implementovaný nástroj. Testovány budou hlavní scénáře využití výsledného nástroje a bude změřena doba běhu pro určité funkce nástroje.

Struktura práce

Struktura této diplomové práce se skládá ze šesti hlavním kapitol a příloh. První kapitola práce se zabývá obecným popisem virtualizace a jejím využitím v informačních technologiích. Dále tato kapitola definuje pojem virtuálního stroje a představuje jednotlivé druhy virtualizačních technik. V závěru první kapitoly je zmíněno několik hlavních scénářů pro nasazení virtuální infrastruktury. Druhá kapitola stručně představuje operační systém Solaris. Důraz je kladen především na podporované platformy a služby, které tento operační systém poskytuje. Třetí kapitola obsahuje podrobný popis virtualizační techniky Solaris Zones. Úvod této kapitoly popisuje základní principy a typy zón, které je možné v rámci této technologie vytvářet. Ve zbytku kapitoly jsou představeny konkrétní způsoby správy této virtualizační techniky, které se zaměřují na popis konfigurace, instalace, zálohy a migrace zón. Čtvrtá kapitola se za-

bývá návrhem nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones. Hlavním obsahem této kapitoly je stanovení požadavků na výsledný nástroj a navržení jeho architektury. V závěru této kapitoly jsou popsány některé bezpečnostní aspekty, které by měl uživatel nástroje splňovat. Pátá kapitola popisuje způsob implementace nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones. V úvodu této kapitoly je popsána volba programovacího jazyka, pomocí kterého byl výsledný nástroj implementován. Zbytek kapitoly popisuje implementaci šablon, uživatelského rozhraní a ostatních funkčních komponent výsledného nástroje. Poslední kapitola je věnována testování implementovaného nástroje. Hlavním obsahem kapitoly je popis testování hlavních scénářů využití nástroje pro správu Solaris Zones. Praktické ukázky testování jsou obsaženy v příloze. Závěr této kapitoly se zabývá měřením doby běhu některých funkcí výsledného nástroje a rozebírá výsledky měření.

Zdrojové kódy celé diplomové práce a implementovaného nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones jsou dostupné na přiloženém médiu.

Kapitola 1

Virtualizace

Jak je z názvu kapitoly patrné, hlavním obsahem následující části práce je právě virtualizace a to především odvětví, které se věnuje výpočetní technice. Virtualizace je velice komplexní téma, které je nutné řádně specifikovat.

Po představení obecného konceptu virtualizace v úvodu této kapitole, je představeno několik oblastí informačních technologií, které tuto techniku využívají. Detailní popis všech oblastí virtualizace není předmětem této práce. Tato diplomová práce se věnuje oblasti virtualizace serverů. I takto specifikované téma však obsahuje mnoho virtualizačních principů a technik, které budou u jednotlivých typů virtuálních strojů představeny. Jelikož virtualizace zažívá v dnešní době velký rozvoj, budou v závěru kapitoly zmíněny některé scénáře nasazení serverů využívající virtualizaci.

1.1 Obecná definice virtualizace

Před popisem jednotlivých virtualizačních technik je nutné definovat pojem virtualizace v obecném slova smyslu. Slovo virtuální je v anglickém jazyce dle slovníku [1] definováno následovně:

Definice 1 (Virtual) *Almost or nearly as described, but not completely or according to strict definition.*

Ve výpočetní technice má tento výraz podle stejného zdroje [1] podobnou definici:

Definice 2 (Virtual in computing) *Not physically existing as such but made by software to appear to do so.*

Proces virtualizace ve výpočetní technice je možné definovat jako vytváření virtuálních prostředků, které skrývají nebo upravují podstatu fyzických prostředků před uživatelem. Tento proces zahrnuje vytváření více virtuálních prostředků z jednoho fyzického prostředku. Jako příklad této virtualizace je

možné použít paměť počítače, kdy se virtuální paměť více procesů mapuje do hlavní (fyzické) paměti počítače. V druhém případě může jít i o vytvoření jednoho virtuálního prostředku z více fyzických prostředků. Příkladem pro tento typ virtualizace může být vytvoření jednoho logického disku z několika fyzických a to například v konfiguraci RAID.

Virtualizace se zcela jistě objevuje i v jiných oblastech, ale pro účely této diplomové práce bude důležité, jak se tento koncept využívá k virtualizaci výpočetní techniky a konkrétně jeho využití v sítích, operačních systémech a také v počítačovém HW.

1.2 Virtualizace ve výpočetní technice

Virtualizace se v dnešní době stala důležitou součástí návrhu počítačových systémů a zdárně se využívá v mnoha oblastech informačních technologií. Velkého rozvoje dosáhla především v oblastech virtualizace operačních systémů, procesorů, sítí a programovacích jazyků.

Součástí dnešních moderních řešení počítačových systémů už nejsou jenom samotné počítače. Nezbytnou součástí architektury je například počítačová síť, která umožňuje počítačům vzájemně komunikovat. Z pohledu architektury a typů zařízení, která se v ní vyskytují, je možné hned najít několik oblastí pro využití virtualizace.

1.2.1 Virtualizace serverů

V dnešní době je pro většinu společností téměř nutností nějakým způsobem využívat výpočetních prostředků. Důvodem pro jejich použití může být potřeba ukládání a zálohy obchodních záznamů, poskytování interních nebo externích služeb či provozování výpočetně náročné aplikace. Ať tak či onak, hlavním poskytovatelem výpočetního výkonu v dnešních počítačových systémech je výpočetní server.

Klasický výpočetní server je fyzický počítač (HW), který poskytuje své výpočetní prostředky řídícímu programu. Řídícím programem se standardně myslí operační systém. Servery můžeme rozdělit dle typu běžících uživatelských programů v operačním systému. Konkrétně se jedná o aplikační, souborové nebo výpočetní servery, které se specializují na poskytování různých druhů služeb, jak je patrné z jejich názvu.

Proces virtualizace serverů spočívá v přenesení operačního systému a jeho služeb do virtuálního prostředí, které napodobuje chování HW, ale není závislé na nižších vrstvách. Dochází tedy ke zvýšení přenositelnosti a možnosti současného běhu více instancí OS na jednom fyzickém stroji. Vytváření tohoto virtuálního prostředí je zajištěno speciální softwarovou vrstvou, která se ve většině případů nazývá virtualizační monitor. Tento software hraje roli řídícího programu a pracuje mezi HW a jednotlivými instancemi operačních

systémů. Virtualizační monitor nebo také VMM je detailněji popsán v kapitole 1.6, kde jsou představeny jeho hlavní funkce a jednotlivé typy.

Počítačové systémy využívající virtualizace se skládají z dalšího typu serverů, tzv. virtualizačních serverů, které slouží jako zdroj fyzických prostředků pro instance operačních systémů a jejich uživatelské programy. Tyto servery se vyznačují především velkým množstvím operační paměti a vysokým výpočetním výkonem, který je díky VMM rozdělován mezi operační systémy. Výhody nasazení virtuální infrastruktury jsou popsány v kapitole 1.5.

Tato práce se zaměřuje právě na techniky virtualizace, které jsou v dnešní době aktuální a využívají se k virtualizaci serverů. Práce podrobně představuje virtualizační techniku Solaris Zones od firmy Oracle, která slouží pro vyvářeení virtuálních strojů (zón) v rámci jedné instance operačního systému Solaris.

1.2.2 Využití virtualizace v sítích

Oblast komunikačních sítí je další významnou oblastí pro využití virtualizačních technik. Bez síťové infrastruktury by mezi sebou počítače nemohly komunikovat, a tudíž by jejich využití nemělo takový potenciál. V dnešní době je tato infrastruktura značně rozsáhlá a to v některých případech ztěžuje její správu. S virtualizací přichází do sítí možnost dynamické konfigurace sítě, což umožňuje rychle měnit její topologii. To vše lze uskutečnit z jednoho místa a bez nutnosti zasahovat do fyzických zařízení sítě.

Virtualizace sítí je koncept, který se v mnoha ohledech podobá virtualizaci serverů. V případě serverů, se VMM stará o reprodukci vlastností fyzických prostředků v SW. Podobně je to tomu i v případě virtualizace sítí, kde existuje funkční ekvivalent VMM, který reprodukuje síťové komponenty v SW. Administrátor má tak možnost za chodu vytvářet virtuální síťové komponenty, jako je *switch*, *router*, *firewall* nebo *load balancer*. To vše v rámci desítek sekund. Tento síťový VMM také umožňuje spravovat nové virtuální sítě, které zahrnují všechny standardní síťové služby a kvalitu služeb [2].

1.2.3 Virtualizace desktopu

Společně s virtualizací serverů a sítí je virtualizace desktopu posledním typem virtualizace, která stojí za zmínění. Pojem desktop značí klasický stolní počítač, který má obrazovku, myš a klávesnici.

S desktopem je klasicky spojeno grafické uživatelské prostředí, pomocí kterého uživatel ovládá počítač, instaluje aplikace nebo přizpůsobuje prostředí. Bez využití virtualizace nebo dalších podpůrných systémů jsou všechny informace o uživatelském nastavení uloženy lokálně na počítači a uživatel se k nim dostane pouze z konkrétního stroje. Virtualizací desktopu se míní oddělení uživatelského prostředí a nastavení od lokálního počítače. Jednou z možností je přesunutí tohoto prostředí do virtuálního stroje, který je centrálně spravován a spouštěn v případě potřeby uživatele. Tento koncept umožňuje uživateli

přístup ke svému prostředí téměř bez ohledu na lokalitu nebo platformu. Mezi další benefity zavedení virtualizovaného desktopu patří zvýšení bezpečnosti a zjednodušení správy celého systému. Tyto výhody pramení především z centralizaci tohoto řešení.

1.3 Virtuální stroj

V kapitole 1.1 je pojem virtualizace definován jako virtualizace fyzických (HW) prostředků. Virtualizace celého systému nebo komponenty na určité vrstvě architektury počítače znamená mapování jejich rozhraní na rozhraní nižší vrstvy. Mezi tyto komponenty může patřit procesor, paměť nebo I/O zařízení. Způsob mapování může reprezentovat komponentu počítače v jiném smyslu než fyzicky existuje.

Tento koncept virtualizace nemusí být aplikován pouze na jednotlivé subsystémy jako například disky, ale může být obecně použit na celý systém. Pro tento účel je zavedena speciální SW vrstva, která operuje mezi konkrétními vrstvami počítačového systému, aby bylo dosaženo požadované funkcionality. Tato vrstva poskytuje vyšším vrstvám všechny prostředky nižší vrstvy tak, že vyšší vrstvy nemají o existenci této vrstvy ponětí a přitom může docházet k virtualizaci celého systému. Tímto způsobem může virtuální stroj obejít kompatibilitu některých komponent fyzického stroje nebo omezení HW prostředků.

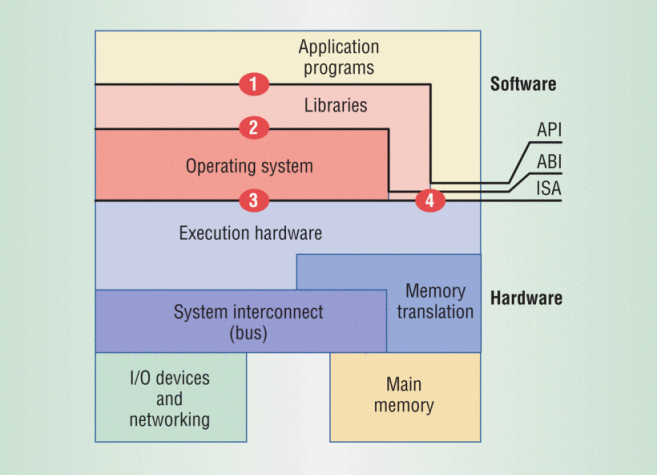
Pro účely klasifikace virtuálních strojů je v následující kapitole představena architektura klasického počítačového systému.

1.3.1 Architektura počítačového systému

Jelikož virtuální stroje operují na rozhraních jednotlivých vrstev architektury počítačového systému, je nutné tyto vrstvy řádně představit. Tyto vrstvy reprezentují několik úrovní abstrakce v počítačovém systému, které mají za úkol odstínit složité implementační detaily některých rozhraní. Čím výše se daná vrstva v hierarchii nachází, tím abstraktnější a jednodušší funkce poskytuje.

Každá z vrstev má dobře definované rozhraní, což umožňuje vývoj vyšších vrstev nezávisle na implementaci nižších vrstev. Pro příklad mohou být uvedeni výrobci Intel a AMD, kteří vyrábějí mikroprocesory implementující instrukční sadu IA-32 (x86) [3]. Zatímco nezávisle na vývoji těchto procesorů mohou vývojáři softwaru vyvíjet aplikace, které se kompilují do této instrukční sady. Takto zkompilovaný program pak může být bez problému spuštěn na každém počítači s procesorem architektury IA-32.

Na druhou stranu komponenty navržené pro jeden typ rozhraní nebudou fungovat s rozhraním jiného typu. Jednoduše řečeno program sestavený pomocí instrukcí x86 se nebude dát spustit na počítači s procesorem architektury SPARC. Nicméně díky některým technikám virtualizace je tohoto přenosu možno dosáhnout.



Obrázek 1.1: Architektura počítačového systému [3]

Obrázek 1.1 ukazuje hierarchii počítačového systému a některé jeho SW i HW vrstvy. Dále jsou na obrázku vyznačeny následující rozhraní:

- Instruction set architecture - **ISA**,
- Application binary interface - **ABI**,
- Application programming interface - **API**.

Tyto tři rozhraní jasně definují rozmezí mezi HW a SW a určují architekturu počítačového systému. Uživatelské programy jsou zcela odkázány na funkcionalitu, která je jim poskytnuta kombinací těchto rozhraní.

1.3.1.1 Instruction set architecture

Instrukční sada neboli **ISA** definuje rozhraní mezi HW a SW. Tuto sadu je možné rozdělit na dvě části, a to na systémovou a uživatelskou instrukční sadu. Rozhraní s číslem 4 na obrázku 1.1 reprezentuje uživatelskou instrukční sadu, která obsahuje instrukce dostupné pro všechny uživatelské programy i knihovny. Rozhraní s číslem 3 na stejném obrázku pak reprezentuje systémovou instrukční sadu a zahrnuje instrukce dostupné pouze operačnímu systému. Tyto instrukce je možné vykonávat pouze v privilegovaném režimu procesoru a jsou zodpovědné za správu HW prostředků.

1.3.1.2 Application binary interface

Fyzické prostředky a zařízení dostupné ve fyzickém systému spravuje operační systém, který k nim poskytuje přístup ostatním programům skrze svoje rozhraní. Toto rozhraní s číslem 2 na obrázku 1.1 se nazývá systémové a společně

s uživatelskou částí **ISA** tvoří tzv. *application binary interface* neboli **ABI**. Toto rozhraní tedy neposkytuje aplikačním programům přímý přístup k HW prostředkům, ale zprostředkovává je skrze systémová volání. Operační systém tak může udržovat kompletní přehled o využití jednotlivých prostředků.

1.3.1.3 Application programming interface

Důležitou vrstvou softwarového vybavení počítače jsou uživatelské knihovny. Tyto knihovny skrývají implementační detaily systémových volání a poskytují rozhraní pro vyšší programovací jazyky jako je C nebo C++. Společně s uživatelskou částí instrukční sady tvoří toto rozhraní nazývané *application programming interface* neboli **API**. Toto rozhraní je na obrázku 1.1 označeno číslem 1. Uživatelské programy pak mohou využívat tohoto rozhraní, což přináší výhody v přenositelnosti na systémy, které nabízejí stejné **API**.

1.4 Klasifikace virtuálních strojů

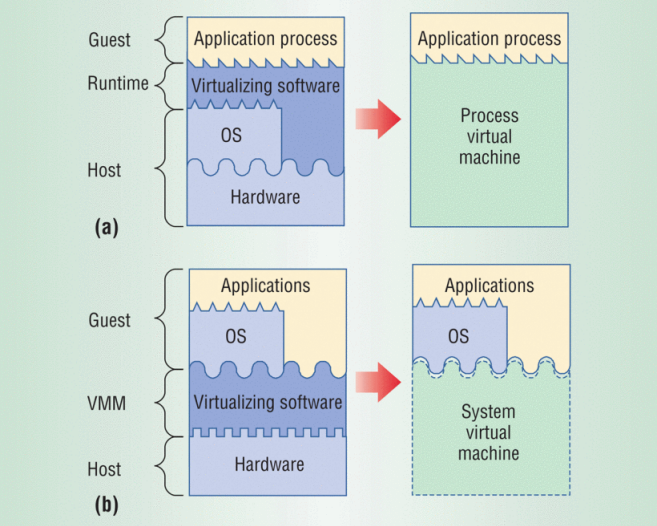
Aby bylo možné rozlišit jednotlivé typy virtuálních strojů, je nutné se podívat v jaké části počítačové architektury operují a tedy jakou vrstvu virtualizují. V části 1.3.1 byly představeny tři dobře definovaná rozhraní počítačového systému a je logické, že virtualizační software bude virtualizovat nějaké z nich. Dle rozdělení [3] mohou být virtuální stroje obecně rozděleny na následující dva druhy v závislosti na tom, které rozhraní počítačové architektury virtualizují:

- *systémové virtuální stroje*,
- *virtuální stroje v procesech.*

Jak může být z názvu patrné, *systémové virtuální stroje* poskytují kompletní systémové prostředí, které podporuje operační systém a jeho aplikace. Operační systém využívá ke svému běhu **ISA** fyzického systému. Systémový virtuální stroj tedy musí poskytovat operačnímu systému stejné rozhraní jako OS očekává. Tímto způsobem nabízí operačnímu systému přístup k HW prostředkům fyzického stroje a součástí tohoto způsobu zprostředkování může být i virtualizace těchto prostředků.

Na druhou stranu procesy využívají mimo uživatelské části **ISA** také **ABI**. Vyšší programovací jazyky využívají ještě abstraktnější rozhraní **API**. Virtuální stroje, které se specializují na virtualizaci těchto dvou systémových rozhraní, se nazývají *virtuální stroje v procesech.* Hlavním účelem tohoto typu virtuálního stroje je podpora jednoho procesu. Její činnost začíná v okamžiku vytvoření procesu a končí v okamžiku jeho ukončení.

V následujících kapitolách jsou popsané jednotlivé typy virtuálních strojů. Tato klasifikace [3] byla převzata a upravena podle požadavků práce.



Obrázek 1.2: Virtuální stroje v procesech (a) versus systémové virtuální stroje (b) [3]

1.4.1 Terminologie

Pro účely diplomové práce zde budou definovány některé pojmy, které se ve virtualizované architektuře počítačového systému vyskytují.

Přidání virtualizačního SW mezi dvě vrstvy počítačového systému vytvoří tři oddělené části. Tuto skutečnost popisuje obrázek 1.2, který zároveň ukazuje cílové umístění virtualizačního software v případě systémových virtuální strojů (b) a virtuálních strojů v procesech (a). Softwarové vrstvy, které se v hierarchii počítačového systému nachází nad virtualizačním SW (je jim poskytováno rozhraní), se souhrnně nazývají *guest*. V případě systémových virtuálních strojů se dá také mluvit o *guest OS*, což je operační systém běžící ve virtuální prostředí. Vrstvy nacházející se pod vritualizačním SW se nazývají *host*. Tyto vrstvy poskytují rozhraní virtualizačnímu SW, který ho zprostředkovává vyšším vrstvám hierarchie.

Poslední vrstvou systému je samotný virtualizační SW. V případě systémových virtuálních strojů se standardně nazývá virtualizační monitor neboli VMM. Pro virtualizační SW v druhém typu virtuálních strojů se používá název *runtime*. Tato vrstva tedy virtualizuje prostředky nižších vrstev a prezentuje tak povahu systému jiným způsobem.

Jak je naznačeno na obrázku 1.2, typ virtuálního stroje je určen strukturou hosta a umístěním virtualizačního SW. Systémové virtuální stroje se tedy skládají z HW vrstvy a virtualizačního monitoru, který přímo operuje nad fyzickými prostředky systému. V případě virtuálních strojů v procesech se vrstva hosta skládá z HW a hostitelského operačního systému, ve kterém je spouštěn *runtime*.

1.4.2 Virtuální stroje v procesech

Jak již bylo zmíněno, tento typ virtuálního stroje poskytuje uživatelským programům virtuální **ABI** nebo **API**. Různé implementace těchto virtuálních strojů si kladou za cíl splnění různých kritérií. Některé se snaží zajistit přenositelnost mezi různými počítačovými platformami a jiné se snaží optimalizovat instrukce uživatelského programu. Následovat bude stručný výčet jednotlivých typů virtuálních strojů, které spadají do této kategorie virtualizace.

1.4.2.1 Virtualizace na úrovni OS

Jako virtuální stroj je možné považovat operační systémy, které umožňují současný běh více procesů současně. Operační systém poskytuje každému procesu iluzi, že na systému běží sám. Z tohoto důvodu je nutné sdílet fyzickou paměť, CPU a jiná HW zařízení mezi běžícími procesy. Operační systém poskytne každému procesu stejně velký izolovaný virtuální adresní prostor, který je mapován do fyzické paměti počítače. Procesor je sdílen mezi procesy tak, že dochází k tzv. přepínání kontextu, což znamená uložení všech registrů a načtení registrů procesu, který má přidělený procesor. Přístup k ostatním HW zařízením systému je řízen skrze systémové volání operačního systému. Z výše uvedených důvodů je možné říct, že OS poskytuje virtuální prostředí pro každý proces v systému.

Dalším zástupcem virtualizace na úrovni OS je rozdělování zdrojů operačního systému nebo také vytváření zón. Pro tento účel operační systém vytvoří pomocí procesů virtuální platformu, které je možné přidělit různé množství dostupných fyzických prostředků. Takto vytvořené virtuální stroje jsou většinou izolované na úrovni souborového systému, sítě a procesů. Jednotlivé zóny se chovají jako nezávislé systémy a sdílejí jádro operačního systému s hlavním operačním systémem. Jednotlivé úrovně izolace zajišťují, že jednotlivé zóny nemají žádné informace o jiných zónách a ani je nemohou přímo ovlivnit.

1.4.2.2 Emulátory a překladače

Jak bylo zmíněno výše, některé implementace těchto virtuální strojů jsou zaměřeny na přenositelnost programů mezi jednotlivými počítačovými architekturami. Tyto virtuální stroje zpracovávají instrukce jiné instrukční sady, než které vykonává systém hosta a poté je dynamicky překládají do této instrukční sady. Konkrétní implementace by mohla například umožňovat vykonávat programy zkompilované pro architekturu IA-32 na systému s architekturou SPARC. Tyto virtuální stroje se nazývají překladače nebo emulátory, jelikož emulují prostředí dostupné na jiných architekturách a mapují ho do architektury hosta.

1.4.2.3 Virtuální stroje v HLL

Virtuální stroje napsané ve vyšších programovacích jazycích přímo navazují na výše popsané téma přenositelnosti mezi platformami. Nevýhoda emulátorů spočívá ve skutečnosti, že se specializují na překlad jedné instrukční sady do druhé. Pokud by bylo nutné docílit přenositelnosti mezi všemi platformami, musel by být vytvořen emulátory pro každou kombinaci instrukčních sad. Jelikož je toto řešení poněkud složité a náročné na implementaci, může být k vytvoření virtuálního stroje použit právě vyšší programovací jazyk. Virtuální stroj napsaný v konkrétním HLL se neopírá o konkrétní architekturu, ale o samotný programovací jazyk a jeho výhody. Takto vytvořený virtuální stroj přijímá vlastní jazyk a využívá konkrétního HLL k jeho provádění. Takto je zajištěna přenositelnost programů, které jsou napsané v jazyce virtuálního stroje, mezi systémy, na kterých jsou dostupné potřebné knihovny a samotná implementace virtuálního stroje.

Nejznámějším zástupcem této kategorie virtuálních strojů je Java virtual machine od společnosti Sun Microsystems. Tento virtuální stroj provádí instrukce vyššího programovacího jazyka zvaného Java a dnes existuje mnoho implementací v nejrůznějších programovacích jazycích. Implementace HotSpot [4], která je napsaná v jazyce C++ a kterou v dnešní době vyvíjí a udržuje společnost Oracle, je pravděpodobně neznámější a nejvíce využívanou implementací JVM.

1.4.3 Systémové virtuální stroje

Druhým typem virtuálních strojů jsou tzv. *systémové virtuální stroje*, které kompletně virtualizují celou HW platformu. Virtualizační software je většinou umístěn ihned nad HW vrstvu počítače a jeho hlavním úkolem je virtualizovat **ISA**. Tímto způsobem *systémové virtuální stroje* vytvářejí virtuální prostředí pro běh operačního systému a jeho aplikací a dokonce umožňují současný běh různých operačních systémů na jednom HW stroji.

1.4.3.1 Virtuální počítače

Nejznámějším a pravděpodobně nejpoužívanějším zástupcem z této kategorie virtuálních strojů jsou tzv. *virtuální počítače*. Tento typ virtuálního stroje umožňuje současný běh více operačních systémů na jednom fyzickém počítači. Všechny operační systémy musí využívat instrukce stejné instrukční sady, kterou vykonává host. Tento typ virtuálního stroje žádným způsobem nepřekládá instrukce do jiné instrukční sady.

Dle umístnění virtualizačního monitoru v architektuře počítače, může být tento typ virtuálního stroje rozdělen do následujících kategorií:

- nativní VMM,
- hosted VMM.

Rozdíl mezi těmito dvěma typy spočívá v umístění virtualizačního monitoru v architektuře počítače. Zatímco *nativní VMM* má přímou kontrolu nad HW počítače, *hosted VMM* běží uvnitř operačního systému, který spravuje HW prostředky. Oba tyto typy VMM jsou detailně popsány v kapitole 1.6.

1.4.3.2 Virtualizace celého systému

V případě, kdy virtualizovaný operační systém používá stejnou instrukční sadu jako fyzická platforma, má virtualizační monitor za úkol spravovat přístup k HW a sdílet ho mezi virtuální systémy. Pokud by bylo vyžadováno spouštění operačního systému s odlišnou instrukční sadou, musí být zajištěna emulace prostředí. Jinými slovy VMM musí vytvořit virtuální prostředí pro konkrétní OS tak, aby si OS myslel, že je spuštěn na odpovídající HW platformě. Tento typ virtuálního stroje se nazývá *virtualizace celého systému*. Mimo instrukční sady mohou být virtualizovány i některá I/O zařízení, která jsou specifická pro konkrétní platformu.

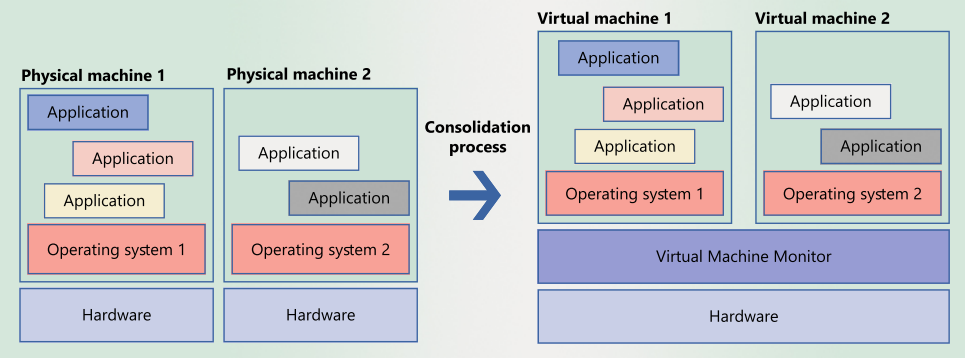
Virtualizace celého systému se hojně využívá případech, kdy je nutné udržet v chodu systémy, který běží na speciálních HW platformách. V tomto případě se vytvoří emulace celého prostředí a systém se přesune do takto vytvořeného prostředí. Systém i jeho aplikace nepoznají žádný rozdíl, jelikož je celé prostředí virtualizováno. Tento typ virtuálního stroje je v mnoha ohledech podobný virtuálnímu stroji popsanému v 1.4.2.2.

1.4.3.3 Resource partitioning

Frekvence dnešních procesorů již není hlavním měřítkem jejich výkonnosti. Moderní procesory škálují svůj výkon s počtem výpočetních jader a díky tomu může být nezávisle spouštěno více výpočetních úloh najednou. Tato jádra jsou propojena sdílenou pamětí, která zajišťuje prostor pro ukládání dat a komunikaci. Tohoto faktu využívá technika zvaná *resource partitioning*, která prostředky vícejádrového systému spojuje do nezávislých částí.

Hard partitioning je technika, kdy jsou fyzické prostředky počítače rozděleny do disjunktních částí. Každá tato část se chová jako nezávislý systém a většinou má vlastní CPU (jádro), paměť, I/O zařízení i adaptér pro připojení k síti. Jednotlivé fyzické komponenty nebo jejich části jsou tedy přímo přiřazeny konkrétní části (partition) systému. V takto rozděleném systému pak může být současně provozováno několik instancí operačních systémů. Tato technika zajišťuje velkou míru izolace pro běžící operační systémy, neboť každý OS má k dispozici vlastní disjunktní prostředí, ve kterém může operovat. Pokud dojde k SW nebo HW chybě v jedné části (partition) systému, aplikace a OS běžící v ostatních částech nejsou nijak omezeny.

Druhým přístupem k rozdělování zdrojů je technika zvaná *logical partitioning*. V tomto případě fyzické prostředky nejsou exkluzivně přiděleny jednotlivým operačním systémům, ale dochází k jejich sdílení na SW úrovni. Ke sdí-



Obrázek 1.3: Konsolidace serverů

lení procesoru a jeho jader se například může používat časový multiplex, který umožňuje střídání OS ve využívání procesoru. Tato technika umožňuje lepší využití fyzických prostředků než *hard partitioning*, kde některé prostředky mohou kvůli malé zátěži zůstat nevyužity.

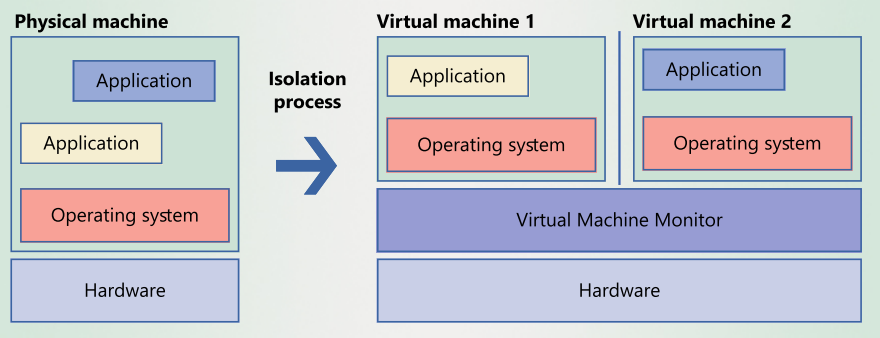
1.5 Nasazení virtuální infrastruktury

Pro přechod k virtuální infrastruktuře serverů existuje v dnešní době několik důvodů. Jedním z hlavních benefitů virtualizace pro dnešní firmy a organizace je značná finanční úspora. Tato úspora se projevuje především ve snížení nákladů organizace na pořizování a provoz fyzických zařízení.

Mezi další benefity virtualizace patří především efektivní využití výpočetních zdrojů, vysoká dostupnost běžících aplikací nebo vytvoření oddělených a nezávislých prostředí pro vývoj, testování a nasazení software. Výhody zavedení virtuální infrastruktury jsou podrobněji popsány v následujících kapitolách, které se zabývají základními scénáři pro nasazení virtuální infrastruktury.

1.5.1 Konsolidace

Konsolidace serverů je proces sjednocování systémů z více fyzických serverů na jeden fyzický server, který pro tyto systémy poskytne virtuální prostředí pro jejich běh. Vstupem tohoto procesu je tedy několik systémů na fyzických serverech, na kterých jsou spuštěny různé aplikace. Vstup procesu je zobrazen na obrázku 1.5.1 vlevo. Výstupem konsolidace je jeden fyzický server s dostatečnými prostředky, na kterém jsou konsolidované systémy spuštěny jako virtuální počítače. Výstup tohoto procesu je zobrazen na obrázku 1.5.1 vpravo.



Obrázek 1.4: Izolace aplikací

1.5.1.1 Využití scénáře

Dnes je zcela běžnou praxí provozovat jednu aplikaci na jednom dedikovaném serveru. Pokud aplikace využívá jen malé procento výpočetních zdrojů daného serveru, může administrátor sjednotit více takovýchto serverů do jednoho. Pro organizaci, která vlastní tisíce takových serverů, může konsolidace výrazně zmenšit požadavky na prostor, spotřebu energie a provoz fyzických serverů. Správnou konsolidací serverů může společnost docílit efektivního využití dostupných prostředků a tím výrazně snížit vynaložené finanční prostředky [5].

Rychlý vývoj technologií v oblasti hardware je příčinou rychlého stárnutí některých systémů. Přechod ze staršího na nový může být složitý, obzvláště v případě, kdy systém potřebuje ke svému běhu speciální hardware. Aby bylo možné provozovat služby poskytované těmito zastaralými systémy, můžeme je spustit jako virtuální počítač na modernějším hardware. Systém se bude chovat stejně jako kdyby byl spuštěn na zastaralém hardware, zatímco výkonnost služby může těžit z novější a výkonnější hardwarové vrstvy [5].

1.5.2 Izolace

Dalším ze scénářů využití virtualizované infrastruktury je izolace aplikací. Proces izolace aplikací spočívá v oddělení dvou a více kritických aplikací spuštěných na jednom systému do nezávislých virtuálních prostředí. Vstupem je jeden systém s aplikacemi, které se mohou negativně ovlivňovat. Vstup izolačního scénáře je zobrazen na obrázku 1.5.2 vlevo. Výstupem je několik nezávislých virtuálních počítačů, ve kterých jsou spuštěny jednotlivé aplikace. Výstup izolace aplikací je zobrazen na obrázku 1.5.2 vpravo.

1.5.2.1 Využití scénáře

V dnešní době jsou útoky na aplikace vystavené do internetu běžnou záležitostí. Pokud útočník využije nějaké zranitelnosti aplikace, může v některých

případech získat kontrolu nad celým systémem. V takovém případě jsou ohrožena všechna data a aplikace, které jsou na daném systému spuštěny. V tomto případě je vhodné využít virtualizaci a rozdělit aplikace do nezávislých prostředí.

Jedním z příkladů ohrožení systému může být útok na výpočetní zdroje. Podstatou útoku je vyčerpání fyzických zdrojů systému, což má za následek nedostupnost jeho služeb a v některých případech i pád celého systému. Ve virtualizovaném prostředí lze přidělit každému VM pouze určitou část prostředků a tím chránit celý systém před jejich vyčerpáním. V případě napadení jednoho VM sice dojde k jeho vyřazení, ale ostatní VM a jejich služby mohou dále pokračovat v provozu.

1.5.3 Migrace

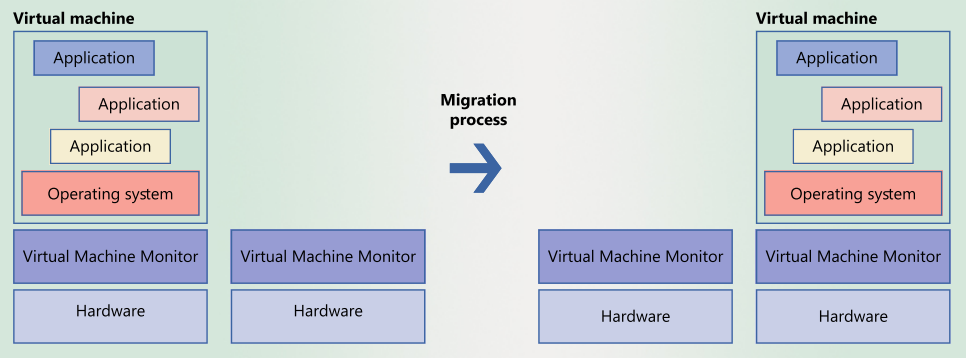
Posledním diskutovaným scénářem nasazení virtualizované infrastruktury je migrace. Jedná se o proces přesunutí systému z jednoho počítače na druhý. V rámci virtualizace bude předmětem migrace systému na počítač s běžícím VMM, který zprostředkovává virtuální prostředí. Výstupem procesu je systém, který do něj zároveň vstupuje. Rozdíl je v tom, že daný systém je na konci procesu spuštěn ve virtuálním prostředí jiného VMM. Dle typu migrovaného systému je možné rozdělit scénář na dva typy.

1.5.3.1 Migrace VM

Migrací virtuálního stroje se rozumí přesun VM mezi dvěma různými fyzickými stroji s VMM. Tento přesun byl dříve možný pouze v případě, kdy oba stroje měly stejný HW, operační systém a procesor [5]. Tato možnost administrátorovi umožňuje přesouvat virtualizované systémy na výkonnější hosty a tím umožňuje dynamicky regulovat využití fyzických prostředků v závislosti na aktuální zátěži systému.

Další výhodou zavedení virtualizované architektury je zajištění vysoké dostupnosti služeb. Virtualizace umožňuje zajistit redundanci ve smyslu spuštění služby na více serverech najednou. Ve virtualizované architektuře můžou nastat dva typy selhání. Prvním typem je selhání virtuálního počítače. Pokud dojde k selhání některého VM, jiný VM převezme obsluhu požadavků a v minimálním čase dojde k obnovení služby. Druhým typem je selhání celého VMM nebo fyzického hosta. V tomto případě je nutné provozovat více redundantních hostů pro virtuální počítače, které v případě HW chyby převezmou obsluhu služby.

Proces migrace VM je demonstrován na obrázku 1.5.3.2, kde je zobrazena konfigurace před migrací a po provedení migrace VM.



Obrázek 1.5: Migrace virtuálního stroje mezi virtualizačními monitory

1.5.3.2 Migrace fyzického stoje na VM

Migrace fyzického stroje na virtuální stroj je proces, kdy dochází k přesunu a virtualizaci systému z fyzického stroje. Vstupem je systém, který je provozován na fyzickém stroji bez VMM. Výstupem tohoto procesu je opět virtuální stroj běžící ve virtuálním prostředí VMM.

Virtualizací serverů dochází k uvolňování fyzického HW a stejně jako v případě konsolidace, může společnost značně ušetřit na nákladech nutných k provozu a správě fyzických serverů. Obecně lze říct, že tento scénář přináší podobné benefity jako v případě konsolidace popsané v kapitole 1.5.1.

1.6 Virtualizační monitor

Jak již bylo zmíněno, virtualizační monitor je softwarová vrstva starající se o virtualizaci fyzických prostředků hosta. Vytváří virtuální prostředí pro běh virtuálních počítačů. Virtualizační monitor tedy vytváří iluzi pro každý virtuální počítač, která mu umožňuje transparentně přistupovat k fyzickým prostředkům počítače.

1.6.1 Režimy ochrany CPU

Moderní procesory mají čtyři režimy ochrany paměti, které se značí čísly od 0 do 3. Nejvyšší režim ochrany má číslo 0 a je označován jako režim *kernel*. Procesor běžící v tomto režimu má přístup ke všem instrukcím instrukční sady a může přistupovat k jakékoli paměti. Číslo 3 naopak znamená režim nejnižší ochrany a může využívat jen část instrukční sady, která se nazývá uživatelská **ISA**. Tomuto režimu se říká *user*. Většina operačních systémů používá právě dva výše zmíněné režimy ochrany a zbylé dva jsou nevyužité.

Operační systémy v nevirtualizované architektuře hrají roli řídícího programu, a proto běží v režimu *kernel*. Ve virtualizované architektuře tuto roli přebírá VMM, a proto je potřeba, aby běžel v nejvíce privilegovaném režimu.

Řešením může být posunutí OS virtuálního stroje do méně privilegovaného režimu, například na úroveň 1, zatímco VMM poběží na úrovni 0. Druhým řešením je vytvoření nového režimu ochrany s číslem -1, který bude speciálně pro VMM. Současné procesory řeší ochranu právě tímto způsobem a podporují plnou virtualizaci v HW [6].

Výsledkem je tedy znemožnění přístupu programů s nižší úrovní ochrany k paměti alokované programem z vyšší úrovně ochrany. Paměť VMM je ve virtualizované architektuře chráněna proti přístupu z OS virtuálního počítače a stejně tak je chráněna paměť OS virtuálního počítače před přístupem z jeho procesů.

1.6.2 Požadavky na VMM

Z úlohy virtualizačního monitoru při virtualizaci systémů plyne několik základních požadavků, které by měl VMM splňovat. Specifikace [6] těchto požadavků byla převzata.

1.6.2.1 Transparentnost

Operační systém běžící ve virtuálním prostředí virtualizačího monitoru by se neměl dozvědět o existenci VMM ani jiných VM, se kterými ve skutečnosti sdílí prostředky hosta.

VMM tedy zachytává systémové volání operačních systémů na HW zařízení nebo na čtení z paměti a transparentně je vyřizuje. Operační systém virtuálního počítače si tak myslí, že komunikuje přímo s HW. V této fázi VMM využívá některých technik virtualizace CPU, paměti nebo I/O zařízení, aby mohl sdílet prostředky mezi virtuální stroje.

1.6.2.2 Izolace

Virtualizační monitor vytváří virtuální prostředí, které by mělo izolovat jednotlivé instance OS. Každý virtuální stroj má svůj kontext procesoru i svoji virtuální paměť mapovanou do jiných oblastí fyzické paměti. Jednotlivé VM tedy nemohou vzájemně ovlivňovat svoji činnost a pád jedné VM by neměl ovlivnit činnost ostatních.

1.6.2.3 Ochrana

Virtualizační monitor by měl být chráněn proti přístupu z virtuálních počítačů. Operační systém virtuálního stroje je provozován v privilegovaném režimu úrovně 1 (ring 1). Virtualizační monitor je provozován v nejvíce privilegovaném režimu [1] úrovně 0. Pro operační systém virtuálního počítače to

[1] Současné procesory podporují plnou virtualizaci v HW. Mají speciální režim ochrany úrovně -1 speciálně pro VMM [6]

znamená, že nemůže přistupovat do paměti mapované pro VMM a privilegované instrukce vyvolají systémové přerušení, které obsluhuje virtualizační monitor. V tomto okamžiku může docházet k emulaci chování těchto privilegovaných instrukcí.

1.6.3 Typy VMM

V následujících kapitolách jsou definovány tři základní druhy typické architektury virtualizačního monitoru, které se liší ve způsobu přístupu k fyzickým prostředkům systému.

1.6.3.1 Nativní VMM

Virtualizační monitor se nazývá *nativní* nebo také *bare-metal*, pokud má přímý přístup k HW pomocí vlastních ovladačů. Obvykle se *nativní* VMM implementuje ve firmwaru počítače nebo jako hlavní řídící program, který se po startu systému zavede do hlavní paměti a je mu předáno řízení. Tento přístup poskytuje nejvíce kontroly nad systémem a je také nejefektivnější, protože VMM přístup k HW zařizuje přímo.

Pokud virtualizační monitor nepodporuje určité typy procesorů nebo periferních zařízení některých HW platforem, může softwarové vybavení VMM limitovat přenositelnost na tyto platformy [7].

1.6.3.2 Hostovaný VMM

Druhým typem architektury je *hostovaný* VMM. Tento název vychází z faktu, že VMM využívá ke svému běhu existující operační systém. Tento operační systém mu poskytuje rozhraní s HW. Virtualizační monitor v tomto případě neobsahuje ovladače k HW, protože jsou součástí operačního systému hosta. Aby mohl *hostovaný* VMM správně pracovat, běží některé jeho části (jádro) v privilegovaném režimu procesoru společně s operačním systémem hosta. Požadavky na HW zařízení jsou jádrem VMM přesměrovány do komponenty VMM, která neběží v privilegovaném režimu. Tato komponenta dále zavolá nativní rozhraní OS a požadavek je vyřízen operačním systémem hosta [7].

Tento typ virtualizace je v dnešní době velice populární, protože umožňuje používat virtualizaci na systémech s běžícím operačním systémem. Přenositelnost *hostovaného* VMM je zcela určena přenositelností dané implementace. Na druhou stranu tento typ VMM není vhodný pro nasazení do prostředí s vysokými nároky výpočetní výkon, jelikož díky další vrstvě mezi HW a VMM není tak efektivní jako *nativní* VMM.

1.6.3.3 Servisní VM

Některé typy *nativních* VMM vyžadují pro svoji plnou funkčnost jednu nebo více speciálních instancí VM. Tato instance většinou slouží pro správu VMM

a ostatních instancí VM. Nicméně existují i virtualizační monitory, které využívají výhody existujících OS a především jejich podpory velké škály HW ovladačů. V tomto případě je OS spuštěn ve speciální instanci VM a je společně s jeho ovladači využíván jako komponenta VMM [7].

Kapitola 2

Solaris

Solaris nebo dříve SunOS je operační systém původně vytvořený firmou Sun Microsystems. V současné době je vyvíjený a podporovaný firmou Oracle. Je to komplexní unixový operační systém, který v sobě integruje pokročilé technologie pro virtualizaci, moderní souborový systém ZFS, vlastní systém pro instalaci a správu SW a v neposlední řadě také podporu cloudu. Díky integraci těchto technologií poskytuje Solaris stabilní a rychlé prostředí pro různé scénáře nasazení aplikací a navíc tato integrace vytváří pohodlné rozhraní pro správu tohoto OS.

2.1 Verze Solarisu

Nejaktuálnější stabilní verze operačního systému Solaris je verze s označením 11.3. Ke dni 30. ledna 2018 byla uvolněna beta verze 11.4 [8]. Pro účely popisu operačního systému Solaris a jeho služeb, zejména služby Solaris Zones, bude použita stabilní verze 11.3. Existují i starší verze 11.2 a 11.1, které ale nebudou předmětem zkoumání.

2.2 Podporované architektury

Operační systém Solaris v současné době podporuje následující dvě HW architektury počítačových systémů:

- x86,
- SPARC.

Jelikož architektura SPARC není běžně dostupná, bude pro účely této diplomové práce použita architektura x86, přesněji její 64 bitová verze.

2.2.1 Architektura x86

První počítačovou architekturou podporovanou operačním systémem Solaris je x86. Tato architektura je v dnešní době velmi rozšířená především v oblasti osobních počítačů a je podporována nejznámějšími OS jako Windows, Linux a Mac. Solaris tuto architekturu podporuje jak v 32 bitové verzi *x86-32* tak i v 64 bitové verzi *x86-64*.

2.2.2 Architektura SPARC

Scalable Processor Architecture neboli SPARC je z pohledu operačního systému Solaris domovská architektura. Architektura SPARC byla stejně jako Solaris původně navržena společností Sun Microsystems a nyní ji spravuje společnost Oracle. Tato architektura je tedy od začátku své existence úzce spojena s operačním systémem Solaris, který se snaží využívat všechny její výhody. Uplatnění této architektury je především v komerčním sektoru, který klade vysoké nároky na přizpůsobivost a možnosti škálování potřebného řešení.

2.3 Služby

Hlavní předností operačního systému je kvantita a kvalita jeho služeb. Tyto služby umožňují nasazení tohoto OS i ve scénářích, kdy by ostatní OS selhaly nebo by nemohly být vůbec použity.

2.3.1 Service Management Facility

Service Management Facility neboli SMF je systém, který v operačním systému Solaris spravuje systémové služby. Nahrazuje tím tradiční způsob spravování služeb pomocí tzv. *init* skriptů, který byl běžný v ostatních unixových operačních systémech a dokonce i v dřívějších verzích OS Solaris. Hlavním rozdílem oproti staršímu způsobu je možnost u služby definovat závislosti na ostatních službách. Na rozdíl od sériového spouštění *init* skriptů z adresáře je díky tomuto zlepšení možné při startu systému paralelně spouštět více nezávislých služeb najednou, a tím urychlit start systému [9]. Pro účel startu jsou v systému definovány speciální služby tzv. *milestone*. Tyto služby mají definovaný pouze seznam závislostí, který určuje jaké služby se mají spustit. Při startu se určí, do kterého *milestone* má systém nastartovat a tím je přesně určeno, které služby se mají spustit.

2.3.2 Souborový systém ZettaByte

Pro ukládání na disk používá Solaris souborový systém ZettaByte neboli ZFS. Je to pokročilý systém, který byl vyvinut společností Sun Microsystems a integrován do operačního systému Solaris. ZFS dokáže spravovat velké množství

dat díky své 128-bitové architektuře [10]. Mezi hlavní funkce ZFS patří ověřování integrity dat, vlastní softwarový RAID nebo šifrování dat. Díky principu *copy on write* dokáže udržet data neustále konzistentní, což některé souborové systémy nedokážou nebo tento problém řeší složitě. Architektura tohoto souborového systému umožňuje klonování jednotlivých svazků nebo rychlou a elegantní tvorbu obrazů disku tzv. *snapshot*, které z počátku zabírají minimální místo na disku. Datové bloky jsou totiž zduplikovány až v okamžiku, kdy se zdrojový blok nebo jeho klon změní. Tento způsob uchovávání dat společně s možností *deduplikace* stejných datových bloků značně snižuje nároky na diskové místo.

Principů a funkcí ZFS hojně využívají další služby operačního systému Solaris. Příkladem může být uvedena virtualizační technika Solaris Zones, která je hlavním tématem této diplomové práce.

2.3.3 Virtualizace

Dle specifikace [11] nabízí operační systém Solaris ve verzi 11.3 následující techniky virtualizace:

- virtualizace na úrovni OS,
- virtuální počítače,
- hardware partitions.

Tyto modely se liší zejména ve způsobu izolace virtualizovaných prostředí a ve flexibilitě přidělování prostředků těmto prostředím. Čím více model izoluje prostředí od sebe, tím nabízí menší flexibilitu v přidělování prostředků.

2.3.3.1 Solaris Zones

Jedním z modelů virtualizace nabízené operačním systémem Solaris je *virtualizace na úrovni OS*. Tento model umožňuje vytvořit jedno nebo více izolovaných prostředí (zón) pro běh programů v rámci jedné instance OS. Takto vytvořená prostředí jsou izolována na úrovni procesů, souborového systému a síťových rozhraní. Každá zóna má vlastní lokální pohled na systémové prostředky, které mohou být dále virtualizované operačním systémem. Virtualizace na úrovni operačního systému poskytuje vysoký výkon a flexibilitu, protože nezanechává tak velkou stopu na disku, paměti nebo CPU na rozdíl od ostatních dvou modelů virtualizace.

Operační systém Solaris poskytuje tento model virtualizace skrze službu Solaris Zones, která je přímo integrována to jádra OS.

2.3.3.2 Virtuální počítače

Model *virtuálních počítačů* popsaný v kapitole 1.4.3.1 umožňuje souběžný běh více instancí operačního systému na jednom fyzickém stroji. Každý virtuální

počítač má svojí instanci operačního systému, který nemusí být stejný ve všech virtuálních strojích. Tento typ virtualizace je umožněn díky virtualizačnímu monitoru, který vytváří pro operační systémy iluzi, která izoluje jednotlivé virtuální počítače. Virtuální počítače poskytují na rozdíl od virtualizace na úrovni OS menší flexibilitu rozdělování prostředků, ale naopak poskytuje větší úroveň izolace.

Tento typ virtualizace je v OS Solaris 11.3 podporován produkty Oracle VM Server for x86, Oracle VM Server for SPARC a Oracle VM VirtualBox [12]. Každá z těchto implementací se zaměřuje na jinou architekturu nebo používá jiný typ virtualizačního monitoru.

2.3.3.3 Hardware partitions

Posledním modelem, který je nepřímo podporovaný operačním systémem Solaris, jsou tzv. *hardware partitions*. Je to technika, která fyzicky odděluje běh OS na oddělených částech fyzických prostředků. Tohoto způsobu virtualizace je dosaženo bez pomocí virtualizačního monitoru, a proto tato technika poskytuje reálný výkon systému. *Hardware partitions* je technika poskytující běžícím operačním systémům největší izolaci, ale není tak flexibilní v konfiguraci prostředků jako výše zmíněné modely.

Tento model virtualizace není z logických důvodů poskytován operačním systémem Solaris, jelikož se jedná o virtualizaci na HW úrovni. Pro účely nasazení tohoto OS s touto virtualizační technikou používá Oracle speciální servery SPARC M-Series [12].

Kapitola 3

Solaris Zones

V úvodu následující kapitoly jsou popsány základní principy a struktura virtualizační techniky Solaris Zones od firmy Oracle. Dále se tato kapitola zabývá popisem datových struktur, postupů a nástrojů, které slouží ke správě a manipulaci se zónami. Detailnější popis je věnován především administrátorským rutinám pro konfiguraci, instalaci a zálohování zón.

3.1 Virtualizační technika

Oracle Solaris Zones je virtualizační technika, která umožňuje běh více virtuálních počítačů na jednom fyzickém stroji. Jak bylo již, tato technika je standardní součástí operačního systému Oracle Solaris od verze systému Solaris 10. Aktuální verze je Solaris 11.3, která přináší novou funkcionalitu v podobě podpory starších verzí operačního systému Solaris.

Virtualizační techniku Solaris Zones je možné zařadit v klasifikaci virtuálních strojů popsané v kapitole 1.4 do sekce *virtualizace na úrovni OS*. Tato technika tedy rozděluje zdroje hostitelského operačního systému jako CPU, paměť nebo I/O zařízení mezi běžící virtuální stroje a zajišťuje izolaci na úrovni procesů, souborového systému a sítě. Zónu je možné definovat jako virtuální kontejner běžící v hostitelském operačním systému, který využívá zdrojů hostitelského operačního systému a je izolovaný od ostatních zón.

Standardně se po instalaci operačního systému Solaris nachází v systému jedna zóna. Je to vlastní instance operačního systému a nazývá se **globální** zóna. Je to zóna, která běží přímo na hardwaru počítače nebo ve virtualizovaném prostředí. Tato zóna má dvě hlavní funkce. Plní funkci hlavního operačního systému a přebírá kontrolu nad fyzickými prostředky po startu systému. Dále je také hlavním centrálním prvkem pro administraci celého systému a ostatních zón. Globální zóna poskytuje globální pohled na celý systém a má přehled o všech systémových zdrojích a aktivitách ostatních zón. Její role v rámci Solaris Zones je zásadní a její chyba může zapříčinit pád ostat-

ních zón. Z tohoto důvodu je doporučené používat globální zónu pouze pro účely administrace systému a managementu ostatních zón.

Neglobální zóny jsou takové, které se spouštějí v rámci globální zóny. Tyto zóny jsou navzájem izolované na několika úrovních. První úroveň izolace je izolace na úrovni sítě. Každá neglobální zóna může mít svůj vlastní logický síťový adaptér, který je vytvořený nad fyzickým síťovým rozhraním a je dostupný pouze pro konkrétní zónu. Z jiné neglobální zóny tento adaptér není přístupný. Takto vytvořený síťový adaptér může být spravován pouze z globální zóny a ze zóny, ke které byl přiřazen v průběhu jejího vytváření. Pomocí virtuálního adaptéru je možné zóně nastavovat IP adresu a připojit ji tak do konkrétní sítě.

Druhou úrovní izolace zón je souborový systém. Globální zóna spravuje svůj vlastní souborový systém, ve kterém se nachází standardní adresářová struktura operačních systému typu UNIX. Můžeme v něm nalézt adresář */etc* sloužící pro globální systémovou konfiguraci, adresář */bin* obsahující uživatelsky spustitelné programy nebo například adresář */sbin*, který uchovává programy spustitelné pod privilegovaným uživatelem. Každá neglobální zóna potřebuje pro svůj běh velmi podobné prostředí, a proto má svůj vlastní souborový systém, který se nachází v hierarchii souborového systému globální zóny. Podle typu zón popsaných v kapitole 3.1.1 může být tento souborový systém částečně sdílený se souborovým systémem globální zóny a nebo může obsahovat kompletně nezávislý obraz zóny. Při spouštění zóny pak dojde pomocí příkazu `chroot(1)` k přepnutí kořenu souborového systému a zóna pracuje pouze se svojí částí souborového systému. V případě sdílené části souborového systému jsou tyto části připojeny v režimu read-only [7]. Tím je zajištěno, že souborové systémy jednotlivých zón jsou vzájemně izolované a nemohou se vzájemně ovlivnit. Správu těchto souborových systémů je opět možné provádět pouze z konkrétní zóny a nebo ze zóny globální.

Mimo izolace na úrovni sítě a souborového systému Solaris Zones implementuje ještě izolaci na úrovni procesů. Každá neglobální zóna má svůj plánovač a může spouštět svoje vlastní procesy. Procesy běžící v jedné neglobální zóně nejsou žádným způsobem viditelné ani přímo ovlivnitelné z jiných neglobálních zón. Pokud chce proces z jedné zóny komunikovat s procesem druhé zóny, nemůže k tomu využít mezi procesovou komunikaci, ale musí použít počítačovou síť. Naopak procesy, které běží v rámci jedné zóny spolu mohou komunikovat pomocí signálů, sdílené paměti a jiných prostředků. Všechny procesy běžící v systému mohou být spravovány z globální zóny. Globální zóna tedy nabízí globální přehled všech procesů, které jsou spuštěny ve všech běžících neglobálních zónách v systému. Výstup příkazu `ps(1)` v globální zóně zobrazí všechny procesy, zatímco v neglobální zóně budou zobrazeny pouze procesy příslušící dané zóně.

3.1.1 Typy zón

Jak bylo zmíněno výše, virtualizační technika Solaris Zones umožňuje v rámci jedné globální zóny spouštět mnoho neglobálních zón. Každá neglobální zóna má vlastnost zvanou *brand*, která určuje její typ. Tato vlastnost se specifikuje při konfiguraci zóny a dle specifikace [13] může mít následující hodnoty:

- *solaris*,
- *solaris-kz*,
- *solaris10*.

Typ zóny neboli *brand* určuje jakým způsobem se zóna bude po spuštění chovat. Implicitním typem zóny v Solaris Zones je *solaris*, který se označuje jako nativní zóna nebo také tenká zóna. Dalším typem zóny je *solaris-kz*, kde zkratka za pomlčkou v názvu odpovídá slovnímu spojením kernel zone. Tato zóna má vlastní jádro operačního systému a někdy se se také označuje jako plná nebo tlustá zóna. Posledním typem zón, kterou Solaris Zones umí vytvářet, je *solaris10*. Hlavním úkolem této zóny je zajišťovat zpětnou kompatibilitu s operačním systémem Solaris 10.

3.1.1.1 Nativní zóna

Nativní neboli tenká zóna umožňuje administrátorovi vytvořit zónu, která má sdílené jádro operačního systému s globální zónou. Verze jádra operačního systému musí tedy být stejná jako v globální zóně. Tento typ zóny je izolovaný pouze nad svým souborovým systémem a standardně nemá k dispozici informace o žádném fyzickém zařízení systému. Souborové systémy ostatních zón jsou nedostupné a konkrétní neglobální zóna o nich nemá žádné informace. Z jejího pohledu existuje pouze její kořenový souborový systém. Jak již bylo popsáno výše, kořenový sytém nativní zóny může být sdílený se souborovým systémem globální zóny a sdílet tak základní systémové nástroje. Neglobální zóně je možné delegovat nějaký typ zařízení. Tento krok musí být učiněn při konfiguraci zóny v globální zóně. Tímto způsobem je možné nativní zóně zpřístupnit souborové systémy, ZFS pool nebo ZFS dataset. Takto definované prostředky jsou po instalaci dostupné uvnitř konkrétní zóny.

Tento typ zóny má svoji vlastní databázi produktů, která obsahuje informace o všech nainstalovaných softwarových komponentech v konkrétní neglobální zóně. Opět platí, že konkrétní neglobální zóna vidí pouze své balíky. Díky tomu je možné instalovat dodatečné softwarové balíky do neglobálních zón, které nemusí být nainstalované v globální zóně [13]. Některé softwarové balíky jsou však společné s globální zónou (jádro OS) a nelze provádět jejich kompletní aktualizaci bez zásahu do globální zóny.

Nativní zóna podporuje dva typy síťových rozhraní, které mohou být zóně při konfiguraci přiřazeny. Prvním typem je sdílená adresa neboli *shared-ip*.

Tento typ síťového rozhraní sdílí IP adresu s konkrétním fyzickým rozhraním globální zóny. Pokud chce neglobální zóna komunikovat s okolím, bude v hlavičce paketu IP adresa globální zóny a při obdržení odpovědi globální zóna přesměruje paket na virtuální síťové rozhraní konkrétní globální zóny. Zde je možné pozorovat podobnost s technikou NAT v počítačových sítích. Jako druhý typ adresy je možné použít exkluzivní rozhraní nebo také *exclusive-ip*, které nesdílí IP adresu s globální zónou, ale má svoji vlastní. V tomto případě veškerý síťový provoz generovaný touto zónou bude mít v hlavičce paketu jinou IP adresu, než kterou má zóna globální.

Tento typ zóny nepodporuje vytváření další neglobálních zón. Nativní neglobální zóna se tedy nemůže chovat jako globální zóna a vytvářet nové zóny uvnitř sebe. Stejně tak z nativní zóny nemůže být vytvořena ani spravována jiná neglobální zóna.

Nativní zóna je implicitní typ zóny v Solaris Zones a pokud administrátor nespecifikuje jinak při vytváření zóny, bude nově vytvořená zóna právě typu *solaris*. Tento typ zóny může být provozován na všech systémech, které podporují operační systém Oracle Solaris 11.3 [13].

3.1.1.2 Kernel zóna

Druhým typem zóny, který virtualizační technika Solaris Zones umožňuje vytvářet, je kernel zóna nebo také tlustá zóna. Tento typ zóny obsahuje vlastní jádro operačního systému a na rozdíl od nativní zóny ho nesdílí s globální zónou. Kernel zóna tedy může být provozována na jiné verzi jádra než globální zóna. V důsledku toho kernel zóna podporuje funkcionalitu, které nelze pomocí nativní zóny dosáhnout.

Stejně jako v případě nativní zóny i kernel zóna obsahuje vlastní databázi instalovaných softwarových balíků. Jelikož i kernel zóna je neglobální, nelze z ní žádným způsobem vidět balíky ostatních zón. Na rozdíl od nativní zóny administrátor může provádět aktualizaci všech balíků, protože kernel zóna nesdílí jádro operačního systému s globální zónou. Žádné balíky tedy nejsou závislé na balících globální zóny.

V případě síťových rozhraní kernel zóny podporují pouze rozhraní typu *exclusive-ip* a neumožňuje sdílet síťovou adresu s globální zónou. Rozhraní typu *shared* je pro tento typ nedostupné.

Na rozdíl od nativní zóny se kernel zóny mohou chovat jako globální zóny uvnitř hostitelské globální zóny. Uvnitř kernel zóny je tedy možné vytvářet další neglobální zóny a vytvářet tak hierarchickou strukturu virtuálních strojů. Je na zvážení administrátora, jestli daný scénář použití vyžaduje tuto strukturu.

Požadavky

Jelikož provoz kernel zóny se liší od provozu standardní nativní zóny, liší se i požadavky na hostitelský systém. Požadavky se liší v závislosti na platformě. Pro jednoduchost budou uvedeny pouze požadavky pro systémy s architekturou x86. Podle specifikace [14] se na hostitelský systém kladou následující požadavky:

- procesor typu Nehalem (nebo novější),
- virtualizace CPU (VT-x) ,
- podpora virtualizace paměti (RVI, EPT),
- ochrana paměti.

Výše zmíněné požadavky kladou nároky na HW vybavení hostitelského systému. Spolu s těmito požadavky musí být v globální zóně nainstalovaný softwarový balík *brand/brand-solaris-kz*, který umožňuje vytváření kernel zón. Administrátor může pomocí příkazu `virtinfo(1)` zjistit, jaký typ virtualizace je v globální zóně podporován. Výpis programu ve virtualizované platformě VMvare, kde jsou splněné výše zmíněné požadavky je zobrazen na v ukázce výpisu programu 3.1.

Výpis kódu 3.1: Výpis příkazu `virtinfo`

```
zadmin@shost:~$ virtinfo
NAME            CLASS
vmware          current
non-global-zone supported
kernel-zone     supported
```

3.1.1.3 Branded zóna

Branded zóny byly vytvořeny pro zpětné zajištění kompatibility se staršími verzemi operačního sytému Solaris. Díky technologii *BrandZ* [13] umožňují spouštění aplikací určených pro operační systém Solaris 10 na systému s OS Solaris 11. Aplikace mohou běžet v nezměněné formě v bezpečném prostředí, které je zajištěno neglobální zónou.

Z pohledu administrátora se tento typ zóny chová stejně jako nativní zóna a má stejné vlastnosti, které jsou popsané v kapitole 3.1.1.1.

3.1.1.4 Shrnutí

Virtualizační technologie Solaris Zones umožňuje vytvářet neglobální zóny uvnitř primární globální zóny. Neglobální zóny poskytují izolované prostředí pro nezávislý a bezpečný běh aplikací. Zóny jsou izolovány na úrovni počítačové sítě, souborového systému a běžících procesů, čímž je zajištěno, že se vzájemně nemohou přímo ovlivňovat. Jediný způsob komunikace procesů z jiných zón je pomocí počítačové sítě.

3.2 Administrace

Globální zóna slouží hlavně pro účely správy hostitelského systému a všech neglobálních zón. Poskytuje nainstalovaným zónám prostředky pro jejich běh a spravuje informace o jejich stavu. Je to klíčové místo pro správu celého systému. Správný chod neglobálních zón vyžaduje bezproblémový chod globální zóny. Administrátor takového systému by tento fakt měl vzít v úvahu a nepoužívat globální zónu jako zdroj pro spouštění uživatelských aplikací.

Proces vytváření zón se skládá ze dvou částí. První částí je konfigurace neglobální zóny, kdy administrátor specifikuje jaké parametry má zóna mít a jaké prostředky bude moci využívat. Tento proces zavede zónu do databáze globální zóny a od této chvíle je registrovaná v systému. K tomuto účelu nabízí Solaris Zones nástroj `zonecfg(1)`. Tento nástroj a možnosti konfigurace zón jsou popsány v kapitole 3.3.

Z předchozího procesu vznikne předpis na to, jak danou neglobální zónu vytvořit. Nyní je třeba vytvořit souborový systém zóny se všemi balíky, které zóna bude potřebovat ke svému běhu. Této fázi se říká instalace zóny a v jejím průběhu se do kořenového souborového systému nahrávají určené balíky a nastavuje se systémový profil zón. Pro tento účel slouží nástroj `zoneadm(1)`. Výstupem tohoto procesu je nainstalovaná zóna připravená ke spuštění. Nástroj pro instalaci zón a proces instalace popisuje kapitola 3.4.

Nainstalovaná zóna může být opět pomocí nástroje `zoneadm(1)` spuštěna a následně se do ní může privilegovaný uživatel přihlásit pomocí nástroje `zlogin(1)`.

3.2.1 Administrátor

Jelikož neglobálních zón může být v systému provozováno velké množství, může být pro administrátora globální zóny složité spravovat celou globální zónu a přitom se starat o všechny neglobální zóny. Pro tento účel může být vytvořen uživatel, který se bude starat výhradně jenom o neglobální zóny. Tento uživatel bude mít práva na správu všech neglobálních zón.

Pokud by i tak bylo neglobálních zón mnoho, je možné jednotlivým neglobálním zónám přiřadit vlastního administrátora. Ten bude mít možnost se

do zóny přihlásit pomocí příkazu `zlogin(1)` a provádět údržbu a správu systému.

3.2.2 Stavový model zón

V Solaris Zones má neglobální zóna definovaný stavový model. Jsou to stavy, ve kterých se zóna během jejího životního cyklu může nacházet. Zóna může být převedena z jednoho stavu do druhého pouze použitím nástrojů `zonecfg(1)` a `zoneadm(1)`. Podle specifikace [15] se neglobální zóna může nacházet v jednom z následujících sedmi stavů:

- *configured*,
- *incomplete*,
- *unavailable*,
- *installed*,
- *ready*,
- *running*,
- *shutting down/down*.

V každém z těchto stavů může administrátor používat pouze konkrétní podmnožinu příkazů, které zónu ovládají. Pro příklad může být uvedeno, že zónu nelze spustit pokud se nachází například ve stavu *configured*. Pro spuštění se zóna musí nacházet ve stavu *installed* nebo *ready*.

3.2.2.1 Configured

Stav *configured* značí, že konfigurace zóny je hotová a uložená na perzistentním úložišti. V tuto chvíli se zónou ještě nebyl spojen žádný diskový obraz a tedy nemá připojený kořenový souborový systém. Tento stav je v pořadí prvním stavem, ve kterém se zóna může od svého vzniku nacházet. Nachází se v něm buď bezprostředně po vytvoření konfigurace pomocí `zonecfg(1)` nebo pokud je zóna odinstalována nebo odpojena.

3.2.2.2 Incomplete

Zóna se nachází ve stavu *incomplete* během procesu instalace a odinstalace. Je to přechodný stav, ale v případě poškození nainstalované zóny může být v tomto stavu stále. V případě úspěchu procesu instalace pomocí nástroje `zoneadm(1)` je stav zóny změněn na stav *installed*. Pokud uspěje proces odinstalování zóny pomocí stejného nástroje, je stav změněn na stav *configured*.

3.2.2.3 Unavailable

Ve stavu *unavailable* se zóna nachází v případě, kdy zóna byla v minulosti nainstalována, ale momentálně nemůže být spuštěna, přesunuta nebo její validace značí chybu. Tento stav může mít několik příčin. Jednou z nich může být nedostupnost zdrojového souborového systému zóny. Souborový systém může být nedostupný chybou administrátora nebo například chybou diskového zařízení. Další příčinou může být nekompatibilita softwarového vybavení globální zóny a neglobální zóny. To se může stát například ve chvíli, kdy dochází k migraci neglobální nativní zóny z jednoho systému na druhý a tyto dva systémy mají odlišnou verzi jádra.

3.2.2.4 Installed

Stav *installed* signalizuje, že zóna s danou konfigurací je nainstalovaná ve svém kořenovém souborovém systému, ale nemá alokovanou žádnou virtuální platformu pro svůj běh. Nemůže být tedy přímo spuštěna. Zóna ve stavu *installed* již může být zálohována nebo migrována mezi různými hosty.

3.2.2.5 Ready

Zóna se nachází ve stavu *ready*, právě když je pro ni alokována virtuální platforma pro její běh. To znamená, že jádro hostitelského operačního systému Solaris vytvořilo proces `zsched`. Vytvořilo také virtuální síťové rozhraní a zpřístupnilo je neglobální zóně. Obecně jádro inicializovalo všechny prostředky specifikované v konfiguraci zóny a zpřístupnilo je dané neglobální zóně. V tomto stavu ještě nebyl spuštěn žádný uživatelský proces asociovaný s konkrétní zónou [15]. Tento stav je tranzitní a nastává v okamžiku, kdy je zahájen boot zóny pomocí příkazu `zoneadm(1)`.

3.2.2.6 Running

Ve stavu *running* se zóna nachází, pokud je spuštěn první uživatelský proces. Většinou se jedná o proces `init(1)`, který inicializuje celou zónu a umožňuje spouštění procesů uvnitř dané neglobální zóny. Zóna ve stavu *running* má tedy alokovanou virtuální platformu v jádru hostitelského operačního systému, inicializovaná všechna zařízení a spuštěné uživatelské procesy.

3.2.2.7 Shutting down/Down

Posledním stavem respektive dvojicí stavů, ve kterých se může neglobální zóna nacházet jsou stavy *shutting down* resp. *down*. Tyto stavy jsou tranzitní a nastávají ve chvíli, kdy daná zóna zastavuje svůj běh. V případě, kdy nelze zónu z nějakého důvodu zastavit, může daná zóna setrvat v některém z těchto dvou stavů [15].

3.2.2.8 Doplňkové stavy kernel zón

Výše zmíněné stavy jsou společné pro všechny typy zón. Nastávají tedy u nativních zón, kernel zón i branded zón. Pro kernel zóny existují ještě další stavy. Jedná se o stavy *suspended*, *debugging*, *panicked*, *migrating-out* a *migrating-in*.

3.3 Konfigurace

Prvním krokem k vytvoření zóny je zaregistrování její konfigurace do systému Solaris Zones. K tomuto účelu se používá nástroj `zonecfg(1)`, který umožňuje vytvářet, měnit nebo mazat konfigurace jednotlivých neglobálních zón. Dle manuálových stránek [16] tento nástroj umožňuje administrátorovi zadávat konfiguraci ve třech následujících režimech:

- interaktivně,
- dávkově,
- pomocí souboru s příkazy.

První režim zadávání konfigurace vyžaduje aktivní účast uživatele a umožňuje interaktivně zadávat příkazy, které definují konfiguraci dané zóny. Další dva režimy načítají příkazy k definici konfigurace z jiného zdroje než je interaktivní vstup uživatele. V případě dávkového režimu se jedná o vstup příkazů na příkazové řádce, kdy jsou příkazy zřetězeny za sebe a předány nástroji `zonecfg(1)` jako parametr. V druhém případě jsou příkazy načteny ze souboru, kde je každý příkaz na samostatné řádce.

Všechny režimy mají jednu věc společnou a to příkazy, kterými předávají nástroji informace o definici konfigurace zóny. Nástroj tedy očekává přesně definovanou syntaxi příkazů, které umí zpracovávat. Konfigurace zóny se skládá z konfigurace globálních atributů a prostředků. Prostředky mohou reprezentovat síťové rozhraní nebo jiný typ zařízení a mají svoje vlastní lokální atributy. Popis všech příkazů, které nástroj `zonecfg(1)` umí zpracovávat je zbytečný, a proto je syntaxe názorně zobrazena ve výpisu programu 3.2. Z výpisu je patrné, že každá řádka začíná příkazem. Příkaz `create` vytvoří v paměti reprezentaci konfigurace, která se po dokončení procesu konfigurace uloží do souboru. Následuje sekvence příkazů `set`, které nastavují konkrétní globální atributy zóny. Dále je možné ve výpisu pozorovat příkaz `add`, který reprezentuje přidání zdroje k zóně. V tomto případě se jedná o přidání síťového rozhraní s automatickou konfigurací. Následuje opět sekvence příkazů `set`, která se však nyní váže k předchozímu příkazu `add` a nastavuje tak lokální atributy daného síťového rozhraní. Celý konfigurační soubor je ukončený příkazem `end`, který reprezentuje výstup z konfigurace prostředku.

Nástroj `zonecfg(1)` umožňuje dva módy editace konfigurace zóny. První mód je editace konfigurace uložené v souborovém systému. Změna konfigurace

Výpis kódu 3.2: Ukázka konfiguračního souboru zóny

```
zadmin@shost:~$ cat /var/tmp/rzone-test_hu67.zonecfg
create -b
set brand=solaris
set zonepath=/system/zones/rzone-test
set autoboot=false
set autoshutdown=shutdown
set ip-type=exclusive
add anet
set linkname=net0
set lower-link=auto
set configure-allowed-address=true
set link-protection=mac-nospoof
set mac-address=auto
end
```

v tomto módu žádným způsobem neovlivní běžící zónu. Druhým způsobem je editace konfigurace v takzvaném živém módu, který umí ovlivňovat nastavení zóny ve stavu *running*. V tomto případě je například možné dočasně přidat běžící zóně síťové rozhraní.

3.3.1 Globální atributy

Globální atributy popisují globální vlastnosti zóny jako celku. Neváží se tedy ke konkrétnímu prostředku, ale k zóně jako takové. Podle manuálových stránek [16] umožňuje příkaz `zonecfg(1)` konfigurovat třináct globálních atributů zóny. V této kapitole nebudou popsány všechny, ale důraz bude kladen na ty nejdůležitější.

3.3.1.1 Jméno zóny

Jedním z hlavních atributů zóny je její jméno. Tento atribut je hlavním identifikátorem zóny v rámci systému a používá se pro její specifikaci v rámci nástrojů `zonecfg(1)` a `zoneadm(1)`. Nastavuje se pomocí vlastnosti *zonename*, nemá žádnou implicitní hodnotu a je povinným atributem pro vytvoření neglobální zóny.

3.3.1.2 Cesta k souborovému systému zóny

Klíčovým atributem zóny je cesta k adresáři, kde je připojený kořenový souborový systém neglobální zóny. Tento adresář obsahuje všechny nezbytné softwarové balíky pro běh zóny. V operačním systému Solaris 11 se pro kořenové

souborové systémy zón používá souborový systém ZFS. Tato skutečnost umožňuje využívat pokročilé funkce ZFS, jako například snapshot nebo klonování při správě a instalaci neglobálních zón. Tento atribut se nastavuje pomocí vlastnosti *zonepath* a je povinným atributem pro vytvoření zóny. V jeho definici je možné používat proměnou *zonename* a jeho implicitní hodnota je nastavena na */system/zones/%{zonename}*.

3.3.1.3 Typ zóny

Jak již bylo zmíněno, typ neglobální zóny určuje jakým způsobem s ní bude globální zóna zacházet. Konfigurace typu zóny se provádí pomocí atributu *brand*, který je povinným atributem pro vytvoření zóny. Může nabývat hodnot *solaris*, *solaris-kz* nebo *solaris10* a jeho implicitní hodnota je *solaris*.

3.3.1.4 Typ IP adresy

Atribut určující typ síťové adresy byl již specifikován v kapitole 3.1.1. Tento atribut určuje zda bude síťová adresa sdílená s adresou globální zóny či nikoli. Typ adresy je možné nastavit pomocí atributu *ip-type* a může mít hodnoty *shared* a *exclusive*, což je zároveň implicitní hodnota.

3.3.1.5 Automatické spouštění a vypínání

Jako poslední budou zmíněny dva atributy, které souvisí se spouštěním a vypínáním neglobálních zón. Atribut *autoboot* vyjadřuje, zda se má daná neglobální zóna spustit při startu zóny globální. Tento atribut může nabývat dvou hodnot. Jestliže je vyžadováno, aby se zóna spustila při startu globální zóny, musí být hodnota tohoto atributu nastavená na *true*. V opačném případě musí být hodnota nastavená na *false*. O automatické spouštění neglobálních zón se stará systémová služba `svc:/system/zones:default`. Proto je nutné, aby byla tato služba aktivní [16].

Druhým atributem je atribut *autoshutdown*, který se uplatňuje při vypínání globální zóny a určuje co se má stát s danou neglobální zónou. Jeho hodnoty mohou být *shutdown* pro korektní vypnutí zóny nebo *halt* a *suspend*. Implicitně se při vypínání globální zóny používá korektní vypnutí neglobální zóny.

3.3.2 Zdroje

Mimo globálních atributů umožňuje nástroj `zonecfg(1)` přidávat do konfigurace zóny také zdroje. Zdroj je objekt konkrétního typu, který má svoje lokální atributy a do konfigurace zóny se přidává pomocí příkazu `add`. Zdroj reprezentuje většinou dané zařízení, souborový systém, prostředky pro přidělování zdrojů zónám nebo specifikuje uživatele, který může zónu administrovat. Některé zdroje mohou být do konfigurace zóny přidány vícekrát. V takovém

případě jim nástroj `zonecfg(1)` automaticky přidělí číselný identifikátor, který daný zdroj jednoznačně určuje. Podle manuálových stránek [16] umožňuje konfigurace zón přidávat až jednadvacet různých typů zdrojů. Z tohoto důvodu budou představeny pouze nejdůležitější zdroje, které jsou nezbytné pro vytvoření zóny.

3.3.2.1 Zařízení

Prvním typem zdroje, který může být delegován neglobální zóně, je obecné zařízení. Takovým zařízením může být například disk nebo disková partition. V případě operačního systému Solaris se jedná o takzvané *slice*, které jsou alternativou k diskovým partition.

Použití tohoto zdroje se liší v závislosti na typu neglobální zóny, ke které ho chceme přiřadit. V případě nativní zóny má tento zdroj následující atributy:

- *match*,
- *allow-partition*,
- *allow-raw-io*.

Atribut *match* odpovídá jménu zařízení, které chceme delegovat zóně. Hodnotou může být absolutní cesta k zařízení nebo regulární výraz, který může specifikovat více zařízení najednou. Druhý atribut *allow-partition* určuje, zda bude moci zóna používat nástroj `format(1)`, který umožňuje rozdělování disku na jednotlivé partition. Přímý přístup na disk může být povolen pomocí atributu *allow-raw-io*.

V případě kernel zóny je podle manuálových stránek [17] povinné přidat alespoň jedno diskové zařízení, které bude sloužit jako hlavní disk. Implicitně je pro kernel zónu vytvořen souborový systém ZFS, který je vyexportovaný jako zařízení. Toto zařízení se přidá v průběhu konfigurace a nastaví se mu atribut *bootpri* na hodnotu 0 (primární disk). Část konfigurace specifikující úložné zařízení kernel zóny je naznačena v kódu 3.3 Podle doporučení v pří-

Výpis kódu 3.3: Ukázka konfigurace zařízení kernel zóny

```
add device
set storage=/dev/zvol/dsk/rpool/VARSHARE/zones/z1/disk
set bootpri=0
set id=0
end
```

ručce [16] může být nebezpečné delegovat zónám obecné zařízení a povolit na ně přímý přístup. Tento krok může vést k nestabilitě a ohrožení globální

zóny, jelikož uživatelské procesy neglobálních zón mohou přímo ovlivňovat delegovaná zařízení.

3.3.2.2 Síťové rozhraní

Jelikož spolu neglobální zóny nemohou komunikovat jinak než pomocí počítačové sítě, nástroj `zonecfg(1)` umožňuje delegovat neglobálním zónám následující dva typy síťových rozhraní:

- Fyzické síťové rozhraní - *net*,
- Automatické síťové rozhraní - *anet*.

Fyzické síťové rozhraní neboli zdroj *net* umožňuje delegovat do neglobální zóny existující fyzické síťové rozhraní z globální zóny. Tento typ zdroje má několik lokálních atributů, které definují jeho zdroj a chování. Hlavním atributem je *physical*, který reprezentuje jméno fyzického rozhraní v globální zóně. Jedno fyzické rozhraní nemůže být sdíleno napříč více neglobálními zónami. Pomocí dalších atributů je možné specifikovat například IP adresy, které se mohou k danému rozhraní připojovat, gateway nebo IP adresu rozhraní.

Automatické síťové rozhraní neboli *anet* je druhým typem síťového zdroje, který může být neglobální zóně přiřazeno. Rozdíl oproti předchozímu typu je v tom, že tento typ rozhraní nemusí v globální síti existovat a je vytvořeno jako virtuální síťové rozhraní. Jelikož je toto zařízení virtuální, umožňuje administrátorovi nastavovat mnohem větší škálu atributů. Těchto atributů je podle manuálových stránek [16] velké množství a popis všech není předmětem této práce. Nejdůležitějšími atributy jsou *linkname* a *lower-link*. Atribut *linkname* určuje jméno síťového rozhraní tak, jak se bude jevit v neglobální zóně. Pomocí tohoto jména je možné toto rozhraní konfigurovat. Atribut *lower-link* je v podstatě jméno fyzického nebo virtuálního síťového rozhraní v globální zóně. Přes toto zařízení bude proudit provoz generovaný vytvořeným virtuálním síťovým rozhraním. Ostatní parametry slouží například k nastavení MAC adresy, ochrany linkové vrstvy nebo VLAN. V ukázce 3.4 je naznačeno, jak by mohla vypadat konfigurace automatického síťového rozhraní pomocí nástroje `zonecfg(1)`.

Výpis kódu 3.4: Ukázka konfigurace síťového rozhraní zóny

```
add anet
set lower-link=auto
set configure-allowed-address=true
set link-protection=mac-nospoof
set mac-address=auto
end
```

3.3.2.3 Řízení zdrojů

Jak již bylo zmíněno výše, neglobální zóny využívají fyzických prostředků hostitelského systému (globální zóny). Aby bylo možné zajistit kontrolu nad přidělováním prostředků jednotlivým neglobálním zónám, umožňuje konfigurace specifikovat některá omezení na využívání zdrojů. Tato omezení se do konfigurace přidávají jako kterékoli jiné zdroje. Podle manuálových stránek [16] existují následující typy zdrojů, které umožňují řízení přidělování fyzických zdrojů:

- exkluzivní CPU - *dedicated-cpu*,
- omezení CPU - *capped-cpu*,
- omezení paměti - *capped-memory*.

První ze zdrojů s názvem *dedicated-cpu* slouží pro alokování určitého počtu procesorů a procesorových jader, exkluzivně pro použití danou neglobální zónou. V systému dojde k vytvoření množiny procesorů, která v době běhu dané neglobální zóny může být využívána pouze danou neglobální zónou. Dokonce ani globální zóna nemůže tyto procesory a jádra využívat. Tento typ zdroje umožňuje specifikovat výpočetní zdroje s různou granularitou. Administrátor může zóně přiřadit prostředky na úrovni procesorových jader, procesorů nebo dokonce celých soketů s několika procesory.

Zdroj *capped-cpu* reprezentuje množství procesorového času, které může být danou zónou využíváno. Tento zdroj má pouze jeden numerický atribut, kterým je desetinné číslo určující možné procentuální využití jednoho procesoru. Hodnota 1 znamená, že daná zóna může využít 100% procesorového času.

Posledním zdrojem, který bude zmíněn v rámci řízení zdrojů zón je *capped-memory*. Tento zdroj se podobá *capped-cpu*, ale řídí využití paměti. Tento zdroj má tři atributy *physical*, *swap* a *locked*, které se týkají určitého druhu paměti. Atribut *physical* souvisí s hlavní operační pamětí počítače a umožňuje administrátorovi specifikovat, kolik hlavní paměti může daná zóna využít. Ostatní atributy mají stejný význam, ale týkají se jiné části paměti. Všechny atributy je možné specifikovat v jednotkách kilobyte (K), megabyte (M), gigabyte (G) nebo terabyte (T).

3.3.2.4 ZFS Dataset

Standardně mají neglobální zóny přístup pouze ke svému kořenovému souborovému systému. Pokud chce zóna využívat další úložiště, musí jí administrátor delegovat buď celý disk nebo použít zdroj *dataset*. Tento zdroj reprezentuje existující souborový systém ZFS v globální zóně, který je delegován do neglobální zóny jako virtuální ZFS pool. Tento zdroj má dva atributy specifikující

jméno souborového systému v globální zóně a alias, pod kterým bude vytvořen virtuální ZFS pool v neglobální zóně.

3.3.2.5 Administrátor zóny

Jak již bylo zmíněno výše, administrátor globální zóny může delegovat administraci neglobální zóny na jiného uživatele. K tomuto účelu se požívá zdroj *admin*, který reprezentuje administrátora dané neglobální zóny. Tomuto administrátorovi lze přiřadit různá privilegia, která mu umožňují různým způsobem spravovat zónu. Podle manuálových stránek [16] je možné nastavit následující privilegia:

- *login*,
- *manage*,
- *copyfrom*,
- *config*,
- *liveconfig*.

Názvy jednotlivých privilegií odpovídají akcím, které může daný uživatel se zónou provádět. Za zmínění stojí pouze privilegium *copyfrom*, které umožňuje administrátorovi vytvářet nové zóny jako klony dané neglobální zóny. V ukázce kódu 3.5 je demonstrována konfigurace, která umožňuje uživateli *zadmin* přihlašování a základní správu dané zóny.

Výpis kódu 3.5: Ukázka delegace administrátorských oprávnění uživateli

```
add admin
set user=zadmin
set auths=login,manage
end
```

3.3.3 Vytvoření konfigurace

První krok v procesu vytváření neglobální zóny je vytvoření její konfigurace. Konfigurace zóny se vytváří pomocí nástroje `zonecfg(1)` a jeho příkazu `create`. Tento příkaz vytvoří datovou reprezentaci konfigurace v paměti počítače a po dokončení konfigurace jí uloží do souboru ve formátu XML v systémovém adresáři */etc/zones*. Teoreticky jediné co administrátor potřebuje k vytvoření standardní zóny, je její jméno. K vytvoření konfigurace může administrátor použít následující způsoby:

- přímá konfigurace,
- konfigurace ze šablony,
- konfigurace z archivu.

Přímá konfigurace již byla popsána v kapitole 3.3 a využívá přímé zadávání příkazů, které obsahují definici atributů a zdrojů zóny. Tento proces může probíhat buď interaktivně postupným zadáváním příkazů nebo dávkově na příkazové řádce.

Druhým způsobem je použití takzvané šablony. Šablona je pouze konfigurací zóny, která je již zaregistrovaná v systému. Pokud tedy administrátor již vytvořil předem neglobální zóny a chce jejich konfiguraci znovu využít, stačí specifikovat jejich jméno jako argument příkazu `create`. Tím vznikne úplně nová konfigurace zóny, která má však stejné atributy jako zóna zdrojová. Standardní instalace operačního systému Solaris již obsahuje standardní šablony pro rychlou tvorbu neglobálních zón. Tyto šablony se nachází společně s konfiguracemi ostatních zón v adresáři */etc/zones*. Uživatel může zjistit, že adresář obsahuje například šablony *SYSdefault.xml*, *SYSsolaris10.xml* nebo *SYSsolaris-kz.xml*, které odpovídají jednotlivým typů neglobálních zón. V ukázce výpisu programu 3.6 je demonstrováno rychlé vytvoření konfigurace zóny se jménem *z1*, které využívá systémovou šablonu *SYSdefault*. Z ukázky

Výpis kódu 3.6: Ukázka vytvoření zóny ze systémové šablony

```
zadmin@shost:~$ zonecfg -z z1 create -t SYSdefault
zadmin@shost:~$ zonecfg -z z1 export
create -b
set brand=solaris
set zonepath=/system/zones/%{zonename}
set autoboot=false
set autoshutdown=shutdown
set ip-type=exclusive
add anet
set linkname=net0
set lower-link=auto
set configure-allowed-address=true
set link-protection=mac-nospoof
set mac-address=auto
end
```

je patrné, že šablona *SYSdefault* nastavuje základní atributy zón s použitím implicitních hodnot a přidává jedno síťové rozhraní pro připojení k síti.

Poslední možností pro vytvoření konfigurace zóny je využití archivu. Archiv musí být typu *Unified archive*, což je výstup systémového nástroje pro zálohování a archivaci zón nebo celých systémů. Tento typ archivu je popsán v kapitole 3.6, která se zabývá zálohováním. Druhou možností archivu je adresář s kořenovým souborovým systémem zóny, která byla odpojena pomocí příkazu `zoneadm detach`. V tomto případě je konfigurace odpojované zóny nahrána do kořenového adresáře a nástroj `zonecfg(1)` si jí převezme.

3.3.4 Změna konfigurace

Dalším administrátorským úkonem ve správě Solaris Zones může být změna existující konfigurace zóny. Administrátor může editovat konfiguraci zóny jak ve stavech *configured* a *installed*, tak i ve stavu *running*. Pokud daná zóna není spuštěná, administrátor může editovat pouze konfiguraci, která je uložená v adresáři */etc/zones*. V případě, že je konfigurovaná zóna spuštěná, může si administrátor vybrat zda změny chce propagovat do uložené konfigurace nebo do živé konfigurace běžící zóny. Změny, které jsou provedeny pouze v živé konfiguraci zóny, jsou po vypnutí dané zóny ztraceny. Při opětovném startu si zóna načte konfiguraci z příslušného souboru.

Editace globálních atributů zóny probíhá stejným způsobem jako při jejich vytváření. Pomocí příkazu `set` nástroje `zonecfg(1)` administrátor pouze přepíše danou hodnotu atributu. Pokud chce uživatel měnit lokální atributy konkrétního zdroje, například síťového rozhraní, musí použít příkaz `select` pro jeho výběr, jak je specifikováno v příručce [18]. Dále už může postupovat stejným způsobem jako při nastavování globálních parametrů.

Při ukončování editace konfigurace zóny je nutné použít příkaz `commit`, který dané změny propaguje do perzistentního úložiště nebo do živé konfigurace zóny.

3.3.5 Smazání konfigurace

Posledním typem podporované operace je mazání konfigurace zóny. K tomuto účelu slouží příkaz `delete` nástroje `zonecfg(1)`. Tato akce nemůže být vrácena, a proto by měl administrátor k tomuto úkonu přistupovat zodpovědně. Po provedení příkazu je konfigurace odstraněna jak z paměti počítače tak ze souborového systému. Tento příkaz neodstraní zdrojový souborový systém zóny.

3.4 Instalace

Dalším krokem na cestě za funkční neglobální zónou je proces její instalace. Tento proces vyžaduje, aby daná zóna měla v systému vytvořenou konfiguraci. Účelem procesu instalace je vytvoření kořenového souborového systému zóny, který obsahuje softwarové balíky nezbytné pro její správný chod.

Instalace zóny se provádí pomocí nástroje `zoneadm(1)`, který poskytuje administrátorovi několik způsobů pro vytvoření kořenového souborového systému zóny. Dle specifikace [19] a manuálových stránek [20] jsou podporovány následující typy instalace:

- instalace z repozitáře,
- instalace pomocí archivu,
- klonování zóny.

Výše zmíněné typy instalace se liší hlavně ve způsobu získávání a vytváření zdrojového souborového systému zón. Kritériem pro výběr může být například doba trvání instalace, protože mezi jednotlivými typy je možné pozorovat zásadní rozdíly. Administrátor musí při výběru druhu instalace brát v úvahu dostupné prostředky systému a časový interval trvání instalace. Druhy instalace budou popsány z pohledu nativní neglobální zóny a rozdíly v instalaci pro kernel zóny budou upřesněny.

3.4.1 Instalace z repozitáře

Prvním typem instalace neglobální zóny je instalace z repozitáře. Repozitář je databáze softwarových balíků, které jsou pro operační systém Solaris poskytovány. Standardní poskytovatel této databáze je společnost Oracle, která hlavní databázi balíku poskytuje jako webovou službu dostupnou z repozitáře [21]. Nastavení této služby v systému se provádí pomocí nástroje `pkg(1)`, který slouží jako správce softwarových balíků v operačním systému Solaris 11.

Instalace touto metodou se provádí pomocí příkazu `install` nástroje `zoneadm(1)`. Jediný povinný argument je jméno zóny, pro kterou chce administrátor nainstalovat systém. Tato zóna se musí nacházet ve stavu *configured*. Po puštění příkazu začne instalátor stahovat potřebné balíky z repozitáře a instalovat je do kořenového souborového systému zóny. V případě nativní zóny se implicitně instaluje softwarový balík *pkg:/group/system/solaris-small-server*. Tento balík je virtuální a obsahuje pouze závislosti. Z jeho názvu je patrné, že balíky budou umožňovat provozovat zónu jako malý server. Jelikož nativní zóna sdílí jádro operačního systému s globální zónou, balík s jádrem se do nativní zóny neinstaluje. V případě kernel zóny se instalace liší a balík s jádrem se nainstaluje. Podle článku [22] je možné v rámci tohoto typu instalace použít pouze stejnou verzi jádra jako používá globální zóna.

Administrátor může předat příkazu `install` další dva volitelné parametry. Prvním z nich je tzv. *manifest*, který předává instalátoru informace o tom, jaké softwarové balíky má nainstalovat do kořenového souborového systému. Druhý parametr odpovídá cestě k tzv. systémovému profilu, který specifikuje systémové nastavení. Tento parametr je společný pro všechny typy instalace.

Po úspěšném dokončení instalace jsou všechny softwarové balíky definované v manifestu nainstalované a připravené k použití. Čerstvě nainstalovaná

zóna se nachází ve stavu *installed* a čeká na první spuštění. Tento typ instalace trvá nejdelší dobu. Je to způsoben tím, že se všechny softwarové balíky musí stáhnout pomocí počítačové sítě.

3.4.1.1 Manifest

Manifest je soubor obsahující definici všech softwarových balíků, které má instalátor nainstalovat do souborového systému zóny. Tento soubor je ve formátu XML a je povinnou součástí instalace. V případě, kdy uživatel nespecifikuje cestu k souboru explicitně, je použit předdefinovaný soubor. Podle specifikace [23] se soubor nachází v adresáři */usr/share/auto_install/manifest* a jmenuje se *zone_default.xml*. Právě v tomto souboru je definované, že standardní výbavou každé zóny je softwarový balík *pkg:/group/system/solaris-small-server*.

Administrátor má možnost si vytvořit vlastní XML soubory s definicemi softwarových balíků. Tyto soubory pak může používat při instalaci nových zón. V ukázce výpisu kódu 3.7 je demonstrováno, jak by mohl takový soubor vypadat. Pro přehlednost je zobrazena pouze část s definicí balíků a zbytek XML dokumentu je vynechán.

Výpis kódu 3.7: Ukázková definice softwarových balíků (manifest)

```
<software_data action="install">
    <name>pkg:/group/system/solaris-small-server</name>
    <name>pkg:/developer/versioning/mercurial</name>
    <name>pkg:/developer/versioning/git</name>
</software_data>
```

3.4.2 Instalace z archivu

Pokud administrátor nechce instalovat zónu přímo z repozitáře může využít možnosti instalace z archivu. Archiv je soubor, který obsahuje zdrojový souborový systém zóny neboli diskový obraz. V tomto případě se nemusí softwarové balíky stahovat přes počítačovou síť z repozitáře, ale jsou zkopírovány ze zdrojového archivu. V předchozím případě instalátor zajistí, že se z repozitáře stáhnou správné verze balíků. V tomto případě může nastat problém s nekompatibilitou balíků neglobální a globální zóny. Pokud se jedná o nativní zónu, pak musí souhlasit verze jádra instalované zóny s verzí jádra globální zóny. Administrátor může instalátoru nastavit, aby při instalaci provedl potřebnou aktualizaci všech balíků. Tento krok lze provést jenom v případě, že verze hosta je vyšší než verze neglobální zóny. V opačném případě se nepodaří zónu z archivu nainstalovat.

Instalace z archivu se provádí pomocí příkazu `install`, kde administrátor specifikuje cestu k danému archivu. Jelikož archiv již obsahuje nainstalované

softwarové balíky, není možné použít manifest pro další specifikaci. Stejně jako při minulém typu instalace je možné použít systémový profil pro konfiguraci systému dané zóny. Instalace z archivu je rychlejší než instalace z repozitáře, ale nekompatibilita globální zóny a archivu může vyústit v neúspěch instalace.

3.4.3 Klonování

Posledním druhem instalace je klonování zóny. Vstupem klonování je již existující nakonfigurovaná a nainstalovaná neglobální zóna. Tento proces instalace využívá pokročilé techniky souborového systému ZFS, který umožňuje vytváření klonů z existujících souborových systémů. Klon je read-write kopie zdrojového souborového systému. Ve skutečnosti se nevytváří úplná kopie souborového systému, ale data se kopírují až v okamžiku, kdy se klon nebo jeho obraz změní. Jak říká specifikace [24], vytvoření klonu trvá zlomek času klasické instalace.

Klonování zón se spouští pomocí příkazu `clone`, který bere zdrojovou zónu jako parametr. Pro efektivní využívání tohoto typu instalace je třeba dodržet, aby se zdrojová i cílová zóna nacházely ve stejném ZFS poolu. V opačném případě je zdrojový souborový systém zkopírován a není využito této techniky klonování. Dále je nutné správně nakonfigurovat instalovanou zónu a změnit atributy, které nemohou zůstat stejné. Především se jedná o atribut *zonepath*, který musí být unikátní.

Obrovskou výhodou tohoto typu vytváření zón je rychlost instalace a celková úspora diskového místa systému. Navíc podle specifikace [24] se všechny aktualizace balíků, které jsou provedeny ve zdrojové zóně, automaticky objeví i ve všech klonech.

3.4.4 Systémový profil

Všechny výše zmíněné typy instalace umožňují specifikovat systémový profil, který slouží pro konfiguraci systému. Profil je soubor ve formátu XML (podobně jako manifest), který podle specifikace [25] umožňuje konfigurovat kteroukoli systémovou službu v rámci SMF. Tento soubor se skládá z popisů konfigurace jednotlivých systémových služeb operačního systému Solaris. Prostřednictvím profilu se například specifikuje počáteční heslo uživatele *root*, konfigurace síťových rozhraní nebo časová zóna systému. Příklad konfigurace uživatele *root* je naznačen v ukázce výpisu kódu 3.8. Pro přehlednost jsou vynechány ostatní části souboru. V ukázce výpisu kódu lze pozorovat, že konfigurace uživatele *root* umožňuje specifikovat jeho počáteční heslo, platnost hesla a typ uživatele.

Konfigurace samostatných služeb je možné vygenerovat pomocí nástroje `sysconfig(1)`, který slouží pro konfiguraci systému. Jak je specifikováno v manuálových stránkách [26], pomocí parametru *grouping* lze stanovit typ služby, pro kterou je potřeba vygenerovat systémový profil. Pokud adminis-

Výpis kódu 3.8: Konfigurace uživatele root

```
<property_group type="application" name="root_account">
 <propval type="astring" name="type" value="%{type}"/>
 <propval type="astring" name="login" value="root"/>
 <propval type="astring" name="password" value="%{password}"/>
 <propval name="expire" value="%{expire}"/>
</property_group>
```

trátor při instalaci zóny neuvede soubor se systémovým profilem, je po prvním startu zóny spuštěn právě nástroj `sysconfig(1)`. Tento nástroj pak interaktivně nastaví konfiguraci požadovaných služeb.

3.5 Správa

Po procesu instalace se zóna nachází ve stavu *installed* a je možné ji spravovat. S neglobálními zónami jde provádět několik základních typů operací, které mohou měnit jejich stav. K účelu správy zón se používá nástroj `zoneadm(1)` a jeho příkazy. Podle manuálových stránek [20] může administrátor pro základní správu neglobálních zón používat následující příkazy:

- `list`,
- `ready`,
- `boot`,
- `shutdown`,
- `halt`.

Příkaz `list` umožňuje administrátorovi získat přehled o zónách přítomných v lokálním systému a jejich stavech. Jedná se o zcela zásadní příkaz, jelikož zobrazuje jména zón, které slouží jako identifikátor. V každém dalším příkazu je nutné specifikovat jméno zóny, pro kterou chce administrátor danou akci vykonat. V ukázce výpisu programu 3.9 je zobrazen výpis příkazu `list`. Z výpisu byl vynechán atribut *zonepath*.

3.5.1 Start zóny

Aby se mohla zóna dostat do stavu *running*, je nutné, aby pro ni byla v jádru globální zóny alokována virtuální platforma. Musí existovat procesy *zsched* a *zoneadmd* asociované s konkrétní neglobální zónou. Tyto procesy se starají o plánování a spouštění procesů v dané zóně a o zpracovávání kontrolních příkazů. Virtuální platforma je pro zónu alokována právě, když se nachází ve

Výpis kódu 3.9: Výpis příkazu `zoneadm list`

```
zadmin@shost:~$ zoneadm list -vic
  ID NAME             STATUS      BRAND      IP
   0 global           running     solaris    shared
   1 zweb1b-clone     running     solaris    excl
   - zweb1b           installed   solaris    excl
   - z1               configured  solaris    excl
```

stavu *ready*. Do tohoto stavu může administrátor zónu dostat pomocí příkazu `ready`, který zařídí vytvoření příslušný procesů, ale ještě nespustí proces *init*.

Zóna ve stavu *ready* je připravená pro spuštění a pomocí příkazu `boot` je možné zahájit její start. Se spuštěním procesu *init* a po proběhnutí startu systému je možné se přihlásit k příkazové řádce. Pokud se zóna před spuštěním příkazu `boot` nenacházela ve stavu *ready*, je automaticky spuštěn stejnojmenný příkaz v rámci příkazu `boot`.

Podle manuálových stránek [20] má příkaz `boot` několik volitelných parametrů, které slouží pro ovlivnění startu systému. Jednou z možností je například zapnout mód pro vypisování detailních informací o startu systému (verbose).

3.5.2 Zastavení zóny

Pro vypnutí zóny poskytuje nástroj `zoneadm(1)` hned dva příkazy. Pro standardní zastavení systému slouží příkaz `shutdown`, který je podle manuálových stránek [20] ekvivalentem volání příkazu `/usr/sbin/init 0` v dané zóně. Tento příkaz počká, než se validně ukončí všechny služby systému. Pomocí dalšího přepínače je možné specifikovat, aby se zóna vzápětí opět nastartovala.

Pokud volání příkazu `shutdown` trvá dlouho nebo je nutné zónu rychle zastavit, může být administrátor použít příkaz `halt`. Tento příkaz násilně ukončí všechny procesy zóny a přepne zónu do stavu *installed*.

3.5.3 Konzole

Přímé ovládání zóny je možné provádět dvěma způsoby. Pokud je uživatel privilegovaný k použití příkazu `zlogin`, může ho použít pro připojení k příkazové řádce dané neglobální zóny. Pomocí příkazové řádky může standardně ovládat běžící systém. Pro tento způsob ovládání musí být uživatel administrátorem globální zóny nebo musí být specifikovaný v konfiguraci dané zóny jako její administrátor.

Druhý způsob se dá použít pouze v případě, že daná zóna má nakonfigurované síťové rozhraní a umožňuje se uživatelům přihlašovat pomocí nástroje `ssh(1)`.

3.6 Zálohování a obnova

Při provozu každého počítačového systému je důležité provádět jeho zálohu pro případ nečekaného selhání. Jinak tomu není ani v případě Solaris Zones. Proces vytváření neglobální zóny zahrnoval vytvoření konfigurace a následnou instalaci softwarových balíků do zdrojového souborového systému zóny, a proto je nutné tyto dva objekty zálohovat. Některé typy zálohy umí vytvářet kompaktní archiv, který obsahuje jak konfiguraci tak i obraz souborového systému zdrojové zóny. V ostatní případech je nutné udržovat konfiguraci a obraz disku odděleně nebo v archivu typu *zip* nebo *tar*. Následující kapitoly popisují dvě základní techniky techniky zálohování Solaris Zones. Základní rozdíl těchto technik spočívá v použitém typu archivu.

3.6.1 Záloha a obnova pomocí ZFS

První typ zálohovací procedury Solaris Zones využívá archiv, který produkuje souborový systém ZFS. Tento typ zálohy lze použít jen v případě, že se souborový systém zálohované zóny nachází v ZFS svazku. V novějších verzích operačního systému Solaris je přítomnost souborového systému ZFS standard, ale ve starší verzích tomu tak být nemusí.

Vstupem tohoto typu zálohy je neglobální zóna, která je nakonfigurovaná, nainstalovaná a nachází se ve stavu *installed*. Prvním krokem je vytvoření snapshotu zdrojového souborového systému zálohované zóny pomocí příkazu `zfs snapshot`. Snapshot je read-only souborový systém, který je kopií zdrojového souborového systému. Rozdíl oproti úplné kopii je v tom, že se data kopírují až v okamžiku, kdy se změní vzor (zdrojový souborový systém). Výhodou této techniky je, že z počátku zabírá minimální místo a vytvoření snapshotu je téměř okamžité. Dalším krokem je konstrukce ZFS archivu z vytvořeného snapshotu. K tomuto účelu se využívá příkaz `zfs send`, který na standardní výstup produkuje datový proud reprezentující daný souborový systém. Úkolem administrátora je přesměrovat tento datový proud do archivu, který bude sloužit jako záloha. Pro zmenšení velikosti archivu je možné použit kompresi typu *bzip* nebo *gzip*. Výstupem zálohy je tedy archiv souborového systému ZFS. Konfiguraci zóny musí administrátor zálohovat zvlášť nebo zabalit pomocí nástroje `zip` nebo `tar` do archivu společně se zálohou.

Obnova zóny z tohoto typu archivu spočívá v obnovení konfigurace zóny z archivu a následné spuštění příkazu `zoneadm attach`, který připojí kořenový souborový systém neglobální zóny na správné místo v globální zóně.

Výhodou tohoto typu zálohy je její relativně rychlé vytvoření a možnost jí vykonávat paralelně pro více zón najednou. Další výhodou může být pře-

směrování výstupu příkazu `zfs send` do nástroje `ssh(1)` a rovnou přijímat archiv na vzdáleném serveru.

3.6.2 Záloha a obnova pomocí UAR

Unified archive neboli UAR [27] je nativní nástroj pro archivování diskových obrazů pro operační systém Solaris. Umožňuje archivaci více systému do jednoho unifikovaného archivu. Součástí tohoto archivu může být i záloha neglobálních zón. Záloha se provádí pomocí nástroje `archiveadm(1)` a její vytvoření je velice jednoduché. Administrátor na příkazové řádce specifikuje, jaké zóny chce archivovat a program spustí. Výstupem je archiv s příponou *uar*, který slouží jako záloha specifikovaných zón.

Obnova ze zálohy pomocí UAR je stejně jednoduchá jako její vytvoření. Jelikož archiv obsahuje jak diskový obraz tak i konfiguraci zóny, je možné vše provést pomocí dvou příkazů. Nejprve je obnovena konfigurace zóny přímo z archivu pomocí nástroje `zonecfg(1)` a následovně spuštěn nástroj `zoneadm` a jeho příkaz `install`, kde je jako parametr specifikována cesta k archivu.

Výhodou zálohy pomocí UAR je její jednoduchost. Celý proces je otázkou několika příkazů. Nevýhodou naopak je, že uživatel potřebuje speciální práva pro práci s nástrojem `archiveadm(1)`. Nevýhodou je také fakt, že se nedá vytvářet více archivů současně. Použití tohoto příkazu je totiž v rámci jednoho systému omezené.

3.7 Migrace

V rámci infrastruktury, která poskytuje výpočetní zdroje virtuálním strojům, může občas dojít k odstávkám některých serverů. Tento případ vyžaduje migraci všech virtuálních strojů na jiné servery, aby nedošlo k přerušení provozu služeb běžících ve virtuálních počítačích. Více serverů, které využívají virtualizační techniky Solaris Zones, mohou reprezentovat takovou infrastrukturu. Solaris Zones poskytují několik technik pro migraci neglobálních zón, které může administrátor využít.

Migrace v kontextu Solaris Zones je přesun neglobálních zón z jedné globální zóny do globální zóny na vzdáleném počítači. Jak je zmíněno v manuálových stránkách [20] a administrátorské příručce [28], hlavním nástrojem pro migraci zón je nástroj `zoneadm` a jeho příkazy `attach` a `detach`. Obecně migrace probíhá tak, že se nejdříve odpojí diskový obraz dané zóny pomocí příkazu `detach`. Tento krok převede danou zónu do stavu *configured*, ale její souborový systém zůstane na původním místě. Poté se souborový systém přenese přímo pomocí `zfs send` nebo pomocí archivu na vzdálený počítač. V tomto kroku se využívá technik popsaných v kapitolách 3.6.1 a 3.6.2 s tím rozdílem, že se vytvořený archiv přesune na vzdálený počítač. Před finálním připojením obrazu disku je třeba ještě nakonfigurovat zónu na cílovém stroji.

Konečně spuštěním příkazu `attach` s odpovídajícím parametrem se začne připojovat diskový obraz ke konfiguraci zóny. Výsledkem je přenesení neglobální zóny z jednoho počítačového systému na druhý.

Kapitola 4

Návrh aplikace

Tato část diplomové práce popisuje návrh aplikace pro podporu automatické správy virtualizačního kontejneru Solaris Zones na platformě Solaris. Zaměřuje se především na popis funkcionality a požadavků, které aplikace musí splňovat. V závěrečné části této kapitoly je rozebrána bezpečnost a požadavky na uživatele, který aplikaci bude moci využívat.

4.1 Požadavky na aplikaci

Hlavním cílem této práce je vytvořit aplikaci, která bude administrátorovi operačního systému Solaris ulehčovat správu většího množství neglobálních zón. Na základě účelu aplikace je nutné vytvořit požadavky, které bude muset výsledná implementace aplikace splňovat. Pokud vytvořená aplikace splní stanovené požadavky, bude moci být cíl práce považován za splněný.

Jak bylo uvedeno výše, virtualizační technika Solaris Zones je exkluzivním produktem pro operační systém Solaris. Tomuto faktu musí být přizpůsoben výběr technologií, které budou použity při implementaci výsledné aplikace. Operační systém Solaris není standardní platformou, i když je v dnešní době podporován na více platformách platformě. Hlavní důraz musí být kladen na kompatibilitu programovacího jazyka a jeho knihoven s operačním systémem Solaris. Z výše uvedených důvodů je možné vyvodit první požadavek na administrační nástroj, kterým je podpora na operačním systému **Solaris**.

Účelem nástroje má být podpora automatické správy neglobálních zón. Pod pojmem správa je myšlena podpora základních administračních postupů a technik, které jsou z velké části popsány v kapitole 3. Mezi tyto postupy patří vytváření neglobálních zón, ale také podpora jejich správy, zálohování nebo migrace. Automatickou správou je myšlena hlavně automatizace procesů vytváření zóny, zálohy nebo migrace, které se skládají z několika kroků. Aplikace by měla administrátorovi poskytovat funkce, které umožní provedení výše zmíněných procesů pomocí jednoho příkazu. Požadavky na aplikaci vy-

plývající z účelu nástroje je možné specifikovat jako **podpora správy** Solaris Zones a **automatizace procesů** administrace.

Virtualizační technika Solaris Zones poskytuje administrátorovi skrze příkazy `zonecfg(1)` a `zoneadm(1)` způsob, jak spravovat lokální neglobální zóny. Zcela zde však chybí podpora pro správu zón na vzdálených serverech. V dnešních infrastrukturách počítačových systémů využívajících virtualizace se nachází mnoho serverů. Tyto servery poskytují své výpočetní prostředky virtuálním strojům. Z tohoto pohledu je tedy žádoucí, aby implementovaná aplikace umožňovala správu neglobálních zón, které se nacházejí na **vzdálených** serverech.

Následující požadavek se vztahuje k automatizaci administračních procesů. Jelikož definice zóny se skládá z její konfigurace, softwarového vybavení a systémového nastavení, aplikace by měla umožňovat specifikaci této definice jednotným způsobem. Aplikace tedy musí poskytovat administrátorovi systém pro vytváření definic zón, které bude možné používat pro jejich vytváření. Tento požadavek lze specifikovat jako podpora vytváření **šablon**.

Uživatel musí mít možnost ovládat nástroj pro podporu správy neglobálních zón. To znamená, že aplikace bude poskytovat uživateli své funkce pomocí **uživatelského rozhraní**. Toto uživatelské rozhraní musí být přehledné a poskytovat uživateli všechny informace potřebné pro využívání jeho funkcí. Pomocí tohoto rozhraní bude uživatel zadávat příkazy, které aplikace bude vykonávat. Rozhraní by mělo nabízet izolovaný pohled pro každého uživatele, který bude aplikaci využívat.

Posledním požadavkem, který musí aplikace splňovat, je **bezpečnost**. Na bezpečnost používání aplikace se musí dbát především proto, že nesprávným a neopatrným používáním virtualizační techniky Solaris Zones může dojít k nestabilitě celého systému. K takovým případům dochází především ve chvílích, kdy neglobální zóny vyčerpají všechny fyzické prostředky systému a tím znemožní správný běh globální zóny.

Kompletní požadavky na aplikaci pro podporu automatické správy Solaris Zones můžeme shrnout do následujících bodů:

- operační systém Solaris,
- lokální a vzdálená správa,
- automatizace administračních procesů,
- šablony,
- uživatelské rozhraní,
- bezpečnost.

Splnění těchto požadavků by mělo vést k značnému zjednodušení správy virtualizačního kontejneru Solaris Zones. Výsledná aplikace by měla zajistit přehled

Obrázek 4.1: Funkční bloky aplikace

o neglobálních zónách, které se nacházejí na lokálním serveru a vzdálených serverech. Aplikace by také měla umožňovat správu těchto zón.

4.2 Architektura aplikace

Prvním krokem v návrhu aplikace je její architektura. Architektura aplikace popisuje její strukturu a určuje jakým způsobem spolu jednotlivé funkční bloky budou komunikovat. Jelikož virtualizační technika Solaris Zones neposkytuje žádné aplikační rozhraní pro konkrétní programovací jazyk, bude nutné postavit aplikaci nad nástroji `zonecfg(1)` a `zoneadm(1)`. Pokud uživatel bude chtít provádět akci se zónou, aplikace sestaví z těchto nástrojů potřebný příkaz nebo jejich sekvenci a vykoná je. Mimo výše uvedených nástrojů je pro efektivní administraci Solaris Zones nutné používat příkaz `zfs(1)` umožňující práci se souborovým systémem ZFS. Pro vytváření záloh pomocí techniky popsané v kapitole 3.6.2 je nutné, aby aplikace uměla používat příkaz `archvieadm(1)`. Aplikace bude simulovat práci administrátora při vykonávání základních administračních rutin tím, že bude vykonávat výše zmíněné příkazy na příkazové řádce.

Pro usnadnění vývoje bude aplikace rozdělena do funkčních bloků, které budou mít na starost konkrétní funkcionalitu aplikace. Prvním funkčním blokem aplikace bude knihovna, která bude zprostředkovávat komunikaci mezi klientskou aplikací a hlavní částí aplikace. Další částí aplikace bude modul, který se bude starat o sestavování a provádění příkazů pro správu Solaris Zones. Tato vrstva bude poskytovat základní funkce pro vytvoření, správu, zálohu, obnovu a migraci neglobálních zón. Poslední částí aplikace bude klientská aplikace, která bude skrze knihovnu využívat funkce modulu. Na obrázku 4.1 jsou znázorněny jednotlivé funkční bloky aplikace a jejich vzájemná interakce.

4.2.1 Knihovna

Knihovna je kontejner, který se bude starat o zprostředkování komunikace mezi klientem a modulem knihovny. Součástí knihovny budou jednotlivé moduly, které budou zajišťovat konkrétní funkcionalitu. V tomto případě se bude jednat o modul, který bude implementovat základní funkce pro administraci Solaris Zones. Knihovna pak bude tyto administrační funkce poskytovat klientům. Celá knihovna bude navržená tak, aby se dala jednoduše rozšířit o další modul. Tento modul bude muset implementovat určité rozhraní, pomocí kterého s ním bude knihovna komunikovat. Díky této architektuře bude v budoucnu jednoduché implementovat další modul, který bude zprostředkovávat

administraci jiného virtualizačního nástroje. Dalším kandidátem může být například modul využívající rozhraní aplikace VirtualBox, která také nabízí rozhraní na příkazové řádce.

Mimo funkcí implementovaných v modulech bude knihovna poskytovat i některé vlastní funkce. Většina virtualizačních technik má společnou jednu věc. Tím je specifikace virtuálního stroje, který chce uživatel ve virtualizovaném prostředí spouštět. Tato specifikace by měla obsahovat základní vlastnost a prostředky, které bude moci daný virtuální stroj využívat. Proto bude knihovna implementovat generickou šablonu, která bude sloužit pro specifikaci virtuálních strojů. V podstatě se bude jednat o hlavičku, která bude určovat název a typ specifikace. Podle typu této specifikace pak bude knihovna přesměrovávat požadavky na konkrétní modul.

Architektura knihovny bude typu *standalone* a nebude tedy poskytovat žádné rozhraní, které by bylo dostupné z počítačové sítě. Veškerou funkcionalitu knihovny bude možné využívat pouze ve stejném systému. Tento krok minimalizuje rizika spojená s napadením aplikace prostřednictvím počítačové sítě.

4.2.2 Modul Solaris Zones

Jednou z hlavních částí aplikace bude modul Solaris Zones, který bude součástí výše zmíněné knihovny. Tento modul se bude skládat z několika hierarchicky uspořádaných vrstev, které se budou navzájem využívat. Nejníže v hierarchii se bude nacházet vrstva, která bude poskytovat spouštění základních nástrojů sloužících pro správu Solaris Zones. Pro poskytnutí základní funkcionality musí tato vrstva poskytovat následující nástroje:

- `zonecfg(1)`,
- `zoneadm(1)`,
- `zfs(1)`,
- `archvieadm(1)`.

Tato vrstva bude tedy umožňovat vyšším vrstvám spouštět tyto nástroje s různým nastavením a různými příkazy.

Vyšší vrstvy budou implementovat základní administrátorské rutiny, které budou složené z funkcí nižších vrstev. Tento modul bude zajišťovat smazání všech dočasných souborů, které byly v průběhu rutiny vytvořeny. K tomu bude také zajišťovat konzistenci ve smyslu navrácení všech změn, které byly v průběhu rutiny provedeny. Toto chování bude nastávat pouze v případě, kdy v průběhu rutiny dojde k chybě.

Jelikož bude tento modul součástí výše zmíněné knihovny, bude od něj očekávána implementace daného rozhraní. Mimo jiné bude muset modul imple-

mentovat funkce pro validaci a zpracování šablon, které budou sloužit pro specifikaci vlastností neglobálních zón.

4.2.3 Klientská aplikace

Poslední neméně důležitou částí nástroje pro správu virtualizačního kontejneru Solaris Zones bude klientská aplikace. Tento funkční blok bude mít za úkol zprostředkovat uživateli funkce modulu Solaris Zones. Samostatná knihovna bude pouze prostředkem, jakým způsobem přímo spravovat konkrétní zónu. Z tohoto důvodu bude na klientské aplikaci, aby zařídila možnost správy většího množství neglobálních zón.

Takto vylepšené možnosti správy Solaris Zones bude klientská aplikace nabízet uživateli pomocí uživatelského rozhraní. Toto uživatelské rozhraní by mělo být jednoduché a přehledné, ale přitom nabízet nejdůležitější funkce pro správu zón. Klientská aplikace by si měla udržovat seznam hostů, které chce daný uživatel spravovat. Na základě tohoto seznamu by měla například zobrazovat všechny neglobální zóny, které se na těchto hostech nachází.

Jelikož aplikaci bude moc využívat více uživatelů, bude každému z nich poskytovat nezávislý pohled. K tomuto účelu bude aplikace využívat uživatelův domovský adresář, kde si bude potřebné informace ukládat. Bude se jednat například o zóny, se kterými uživatel nějak manipuloval nebo je vytvářel. Na základě těchto dat pak klientská aplikace může zobrazovat uživateli změny, které nastaly v době jeho nepřítomnosti.

4.3 Uživatelské rozhraní

Požadavky v kapitole 4.1 stanovují, že hlavním ovládacím prvkem implementované aplikace bude uživatelské rozhraní. Uživatelské rozhraní je prvek, který uživateli umožňuje konkrétním způsobem interagovat s danou aplikací. Tento prvek je možné implementovat pomocí mnoha technologií. Ne všechny typy uživatelského rozhraní jsou však vhodné pro konkrétní typy aplikací.

Hlavní podstatou navrhovaného nástroje je podpora automatizace správy. Nástroj má uživateli umožnit zvládnout hodně práce s malým množství příkazů. Příkladem může být vytvoření desítek neglobálních zón pomocí jedné akce v uživatelském rozhraní. Dalším požadavkem na uživatelské rozhraní může být i možnost uživatelské rozhraní skriptovat a využívat ho například v automatických zálohovacích rutinách. Uživatelské rozhraní tedy musí především umět pracovat v dávkovém režimu. Musí však umožňovat i interaktivní instalaci neglobálních zón.

V dnešní době je pravděpodobně nejpopulárnější webové uživatelské rozhraní. Tento způsob definuje uživatelské rozhraní pomocí HTML stránek a pomocí webového serveru nebo Javascriptu, umožňuje reagovat na akce uživatele. Standardně je tento typ uživatelského rozhraní zprostředkováván pomocí

webového serveru bežícího na portech 80 v případě HTTP nebo 443 v případě HTTPS. Aplikace využívající webové rozhraní jsou většinou typu klient-server, kde aplikace běží na serveru. Výhodou je, že klient se k serveru připojuje vzdáleně pomocí webového prohlížeče a danou aplikaci nemusí mít nainstalovanou. Nevýhodou je nutnost uživatelské interakce a špatná možnost skriptování. Z tohoto důvodu je tento typ uživatelského rozhraní nevhodný pro navrhovaný nástroj.

Klientská aplikace bude jako uživatelské rozhraní využívat kombinaci CLI a grafického rozhraní. Situace a důvody pro použití konkrétního typu uživatelského rozhraní jsou vysvětleny níže.

4.3.1 CLI

Pro umožnění jednoduchého skriptování aplikace bude většina její funkcionality prezentována pomocí rozhraní na příkazové řádce. Celé rozhraní by mělo mít jednotný tvar a syntaxe jednotlivých příkazů by se neměla moc lišit. Uživatel si jednoduše bude moci specifikovat cílové zóny a akci, kterou na nich chce provést. Program potom bez dalšího zásahu uživatele provede danou akci a informuje ho o jejím výsledku pomocí informačního výpisu.

Po zadání příkazu na příkazové řádce bude běh programu ve většině případů neinteraktivní a nebude tedy vyžadovat žádný zásah uživatele. Jedinou výjimku bude tvořit interaktivní instalace zóny, kdy bude použito grafického rozhraní. Po ukončení běhu aplikace může uživatel opět zadávat další příkazy pomocí příkazové řádky.

4.3.2 Grafické rozhraní

Grafické rozhraní bude v aplikaci použito ve dvou případech. Prvním případem je již zmiňovaná interaktivní instalace zón. Při tomto typu instalace bude uživateli zobrazeno dialogové okno, které slouží jako formulář pro vyplnění atributů zóny popsaných v kapitole 3.3. Po vyplnění formuláře bude aplikace pokračovat standardním neinteraktivním způsobem.

Výsledný nástroj bude využívat grafické rozhraní ještě v okamžiku, kdy bude uživatel chtít vytvářet nebo upravovat šablony pro specifikaci virtuálního stroje. Pro tento účel bude aplikace poskytovat editor, který uživateli pomůže s vytvořením šablony. Tento editor se bude opět spouštět pomocí příkazové řádky.

Při výběru grafického rozhraní bude nutné brát ohled na fakt, že aplikace je určená pro platformu Solaris. Jelikož Solaris není standardní platformou, nemusí být všechny grafické knihovny na této platformě podporovány.

4.4 Šablony

Podle požadavků specifikovaných v kapitole 4.1, má aplikace umožňovat vytváření šablon. Šablona slouží jako předpis pro vytvoření virtuálního stroje. Knihovna popsaná v kapitole4.2.1 bude definovat generickou šablonu, kterou bude aplikace umět zpracovávat. Knihovna bude muset umět načítat šablony ze souborového systému a provádět základní validaci. Hlavním obsahem šablony bude její jméno a typ. Podle typu se pak knihovna rozhodne jakému modulu šablonu předá na zpracování.

4.4.1 Šablona Solaris Zones

Ve výsledném nástroji bude obsažený pouze modul pro administraci virtualizační techniky Solaris Zones. Tento modul bude definovat typ šablony, který bude specifikovat neglobální zónu. Opět bude poskytovat funkce pro její validaci, ale oproti knihovně tuto šablonu bude umět využívat pro tvorbu virtuálních kontejnerů Solaris Zones.

Jak již bylo zmíněno, pro úspěšnou instalaci zóny je třeba provést konfiguraci, instalaci softwarových balíků a volitelně i konfiguraci systémových služeb. Aby mohla být zóna ze šablony vytvořena, musí šablona obsahovat právě tyto části.

4.4.1.1 Konfigurace zóny

První část šablony bude obsahovat informace o konfiguraci zóny. Bude zde tedy specifikováno o jaký typ zóny se jedná, jaký typ IP adresy má mít, jaké zdroje mají být zóně delegovány a podobně. Obecně lze říct, že v této části budou specifikovány globální atributy zóny popsané v kapitole 3.3.1 a zdroje zóny popsané v kapitole 3.3.2. Šablona nebude umožňovat specifikovat všechny atributy a zdroje popsané v manuálových stránkách [16], ale jejich podmnožinu, která je využitelná ve většině případů použití virtualizační techniky Solaris Zones.

4.4.1.2 Softwarové balíky

Další podstatnou částí šablony bude sekce se softwarovými balíky. Balíky, které zde uživatel definuje, budou při instalaci zóny z této šablony nainstalovány do kořenového souborového systému zóny. Ihned po prvním startu zóny je uživatel bude moci využívat. Tato část šablony umožňuje uživateli jasně definovat softwarové vybavení a umožňuje tak vytváření konzistentního prostředí.

Přehled softwarových balíků dostupných pro konkrétní verzi operačního systému Solaris může uživatel získat pomocí nástroje pro správu softwarových balíků `pkg(1)` nebo z oficiálního repozitáře [21].

4.4.1.3 Systémový profil

Poslední sekcí šablony pro definici neglobální zóny je konfigurace systémového profilu. V této části šablony může uživatel definovat konfiguraci pro základní systémové služby. Touto cestou může uživatel například nastavovat konfiguraci síťových adaptérů definovaných v sekci s konfigurací. Uživatel může také nastavit časovou zónu, jazyk systému nebo uživatelské účty. Tato sekce nebude opět umožňovat konfiguraci všech systémových služeb, ale jejich část nutnou ke správné a funkční konfiguraci systému.

Minimální konfigurace obsahuje definici hesla uživatele *root* a definici počátečního systémového uživatele. Pokud uživatel nespecifikuje konfiguraci systémových služeb v šabloně, nebude instalovaná zóna vůbec nakonfigurována. Při prvním spuštění zóny bude uživatel vyzván k interaktivní konfiguraci systému.

4.5 Automatizace

Automatizaci administračních procesů bude aplikace zajišťovat na úrovni modulu, který slouží pro správu Solaris Zones. Pokročilejší administrační rutiny se skládají ze sekvence několika příkazů, které musí být provedeny po sobě. V případě selhání, některého z nich dojde k selhání celé rutiny. Často je potřeba při provádění těchto činností vytvořit dočasné soubory nebo dočasně vypnout konkrétní zónu. Tyto situace nastávají hlavně v zálohovacích a migračních rutinách. Modul Solaris Zones bude implementovat tyto pokročilejší rutiny a poskytovat je skrze knihovnu klientským aplikacím. Dále bude zajišťovat, že všechny dočasné soubory budou na konci rutiny odstraněny a dočasné akce vráceny v případě neúspěchu. Toto chování se dá přirovnat k chování transakcí.

Dále bude příkazová řádka aplikace umožňovat zadávání většího počtu neglobálních zón, pro které se má daný příkaz vykonat. Aplikace potom paralelně (pokud je to možné) vykoná daný příkaz pro všechny specifikované zóny.

4.6 Vzdálená správa

Jelikož nástroje pro správu virtualizační techniky Solaris Zones neumožňují správu neglobálních zón na vzdálených serverech, bude modul zmíněný v kapitole 4.2.2 implementovat i funkce pro vzdálenou správu. Ideálním nástrojem pro ovládání vzdálených serverů je program `ssh(1)`. Tento nástroj umožňuje používat příkazovou řádku na vzdáleném serveru a s použitím veřejného a privátního klíče umožňuje i neinteraktivní přihlášení. Tyto dvě vlastnosti jsou pro požadovanou funkcionalitu nástroje pro automatickou správu Solaris Zones klíčové.

Prostředí, do kterého je navrhovaná aplikace směřována, se skládá z několika virtualizačních serverů, které používají operační systém Solaris. V rámci těchto serverů je provozována virtualizační technologie Solaris Zones a běží na nich mnoho neglobálních zón. Klientská aplikace zmíněná v kapitole 4.2.3 musí implementovat způsob identifikace těchto zón v rámci většího počtu serverů. Pomocí tohoto identifikátoru bude uživatel schopný danou zónu specifikovat na příkazové řádce a provádět s ní konkrétní akce.

Aplikace bude umožňovat registraci jednotlivých hostů v rámci dané infrastruktury. Ke každému hostu si aplikace bude držet přístupové údaje, které má použít při připojování. Tyto údaje budou zahrnovat především uživatelské jméno a privátní klíč, který má být použit pro šifrování spojení. Hromadné akce nabízené uživatelským rozhraním se pak budou vztahovat právě k registrovaným hostů.

4.7 Bezpečnost

Důležitou součástí návrhu aplikace je její bezpečnost. V případě nástroje pro automatickou správu Solaris Zones je nutné věnovat zabezpečení velkou pozornost. Nástroje specifikované v kapitole 4.2.2 totiž při nesprávném použití mohou způsobit pád systému. V případě příkazu `zfs(1)` je možné kompletně zničit souborový systém všech neglobálních zón a způsobit tak chybu systému. Naopak pomocí příkazů `zoneadm(1)` je možné vytvořit takové množství neglobálních zón, že dojde k vyčerpání fyzických prostředků globální zóny a následné nefunkčnosti celého systému. Autoři těchto nástrojů na tento problém mysleli, a proto jsou všechny tyto příkazy přístupné pouze privilegovaným uživatelům. Pro aplikaci to znamená, že ji bude moci spouštět pouze uživatel s konkrétními právy. Operační systém Solaris poskytuje službu RBAC, která administrátorovi umožňuje jemněji rozdělit práva mezi uživatele [29]. Pomocí této služby je možné vytvořit uživatele, který bude přímo určený pro správu zón a bude moci používat všechny nástroje stanovené v 4.2.2. Pro spouštění aplikace může být použit jeden z následujících dvou uživatelů:

- Uživatel **root**,
- Privilegovaný uživatel (**RBAC**).

V následujících kapitolách jsou popsány důvody, proč není vhodné pro spouštění aplikace používat uživatele *root* a jaké výhody přináší RBAC.

4.7.1 Uživatel root

Uživatel *root* je plně privilegovaný uživatel v rámci operačního sytému Solaris a má potřebná práva pro spouštění všech potřebných nástrojů. Existuje však několik důvodů, proč není vhodné uživatele *root* používat pro spouštění navrhovaného nástroje pro automatickou správu Solaris Zones. Jedním z důvodů je

ctění principu nejnižších privilegií [30]. Tento princip říká, že aplikace má být spuštěna pouze s nejmenší možnou množinou práv, se kterými je ještě schopná plnit svůj účel. Pokud by byla aplikace nějakým způsobem zkompromitována, útočník může využít pouze těchto práv. Pokud by nebyl dodržen princip nejnižších oprávnění a aplikace by byla spouštěna pod uživatelem *root*, útočník by mohl využívat privilegovaného přístupu v celém systému.

Dalším důvodem proč nepoužívat uživatele *root* je standardní systémové nastavení operačního systému Solaris. Standardně je totiž uživatel *root* v systému zaregistrovaný jako role. Role je funkcionalita RBAC [29], která se chová téměř jako uživatel. Je možné ji přiřazovat práva na provádění privilegovaných operací nebo se na ní přepínat pomocí nástroje `su(1)`. Uživatel může mít v systému přiřazeny role, které může používat. Samotná role však nemá v systému žádnou funkci a nedá se na ni dokonce ani přihlásit. Z tohoto důvodu by nebylo možné přihlašovat se na vzdálených systémech jako uživatel *root* a aplikace by nesplňovala požadavek vzdálené správy.

Poslední důvod souvisí s předchozím důvodem a opět se týká standardního nastavení. Tentokrát se však týká standardního nastavení nástroje `ssh(1)`, které nepovoluje vzdálené přihlašování uživatele *root*.

V důsledku používání uživatele *root* by došlo k porušení principu nejnižších privilegií [30] a navíc by muselo dojít k vypnutí některých standardních bezpečnostních opatření. Z výše uvedených důvodů není vhodné tohoto uživatele používat pro spouštění navrhovaného nástroje pro automatickou správu Solaris Zones.

4.7.2 RBAC

Správnou volbou je vytvořit uživatele, kterému pomocí RBAC přiřadíme potřebná oprávnění pro používání potřebných nástrojů. Aplikace bude spouštěna v souladu s principem nejnižších oprávnění a nebude nutné vypínat bezpečnostní opatření nástroje `ssh(1)` ani jiné systémové nastavení.

Kapitola 5

Implementace

Následující část diplomové práce představuje implementaci nástroje pro automatickou správu Solaris Zones. V úvodu kapitoly je představen použitý programovací jazyk a důvody pro jeho použití. Hlavní částí je popis knihovny, modulu Solaris Zones a klientské aplikace. Důraz je kladen na popis funkcionality jednotlivých částí aplikace a jejich vzájemné komunikace.

5.1 Programovací jazyk

Prvním krokem při implementaci bylo zvolení vhodného programovacího jazyka. Požadavky stanovené v kapitole 4.1 vyžadují od zvoleného programovacího jazyka následující dvě podmínky:

- operační systém Solaris,
- možnost tvorby grafického rozhraní.

Nástroj pro automatickou správu virtualizačního kontejneru Solaris Zones využívá nástrojů na příkazové řádce a zpracovává jejich výstup. Pro tento účel bylo vhodné zvolit interpretovaný programovací jazyk, který umožnil jednoduše spustit nástroje a následně snadno zpracovat jejich výstup. Na základě výstupu se pak nástroj rozhodne o dalším průběhu zpracování uživatelského příkazu. První podmínka není pro volbu jazyka tolik omezující. Pro operační systém Solaris existuje implementace standardního kompilátoru `gcc(1)` pro jazyk C a stejně tak implementace virtuálního stroje JVM pro jazyk Java. Většina interpretovaných jazyků staví svůj překladač právě nad jedním z těchto základních programovacích jazyků.

Z výše uvedených důvodů bylo nutné při volbě jazyka dbát hlavně na dostupnost grafických knihoven pro operační systém Solaris. `Shell` je standardním skriptovacím jazykem pro většinu operačních systému typu UNIX. Tento program interpretuje uživatelské příkazy na příkazové řádce a následně

je provádí. Tato volba by splňovala podmínku platformy, ale těžko by se s pomocí tohoto jazyka vytvářelo grafické uživatelské rozhraní. Z tohoto důvodu byl zvolen programovací jazyk Ruby [31], který umožnil splnit obě stanovené podmínky.

5.1.1 Ruby

Ruby je objektově orientovaný programovací jazyk, který má mnoho možností využití. Jedním ze scénářů využití může být právě spouštění příkazů na příkazové řádce a tvorba uživatelského rozhraní. Objektová povaha tohoto jazyka umožňuje programátorovi využívat všech výhod objektově orientovaného programování. Podle dokumentu [32] existuje několik implementací interpretu jazyka Ruby, z nichž nejpoužívanější jsou YARV [33] a JRuby [34]. Obě tyto implementace jsou dostupné i pro operační systém Solaris.

Pokud chce programátor využívat grafické rozhraní pomocí programovacího jazyka Ruby, je nutné, aby byly v systému nainstalované potřebné grafické knihovny. Standardní knihovny pro programovací jazyk Ruby však nejsou na operačním systému Solaris podporované. Z toho důvodu bylo nutné využít grafické rozhraní, které nabízí implementace JRuby. Tato implementace je postavená nad virtuálním strojem JVM a může využívat grafické knihovny v něm implementované. Navrhovaný nástroj pro automatickou správu virtualizačního kontejneru Solaris Zones tedy využívá programovacího jazyka Ruby. Pokud bude chtít uživatel nástroje využívat grafického rozhraní, musí nástroj spouštět pomocí interpretu JRuby. Zbytek nástroje je nezávislý na použitém interpretu programovacího jazyka Ruby.

5.2 Knihovna

Hlavním centrálním prvkem implementace je knihovna, která zprostředkovává komunikaci mezi implementovanými moduly a klientskými aplikacemi. Knihovna je navržená tak, aby se v budoucnosti dala lehce rozšířit o další moduly, které budou poskytovat funkce pro správu jiných virtualizačních technologií. Jedním z takových rozšíření by mohl být například modul pro podporu virtualizační technologie Oracle VirtualBox. Výsledná implementace obsahuje pouze modul pro podporu automatické správy virtualizačního kontejneru Solaris Zones, který bude popsán v kapitole 5.3.

Knihovna poskytuje hlavní rozhraní, pomocí kterého může klient využívat funkcí jednotlivých modulů. Jednotlivé moduly tedy slouží jako hlavní zdroj funkcionality pro knihovnu.

Mimo zprostředkovávání komunikace mezi moduly a klientem slouží knihovna k validaci šablon, které mají specifikovat konkrétní virtuální stroj. V případě šablon knihovna funguje jako vstupní bod, který umí šablonu načíst a provést prvotní validaci. Spouštění těchto operací a jejich výsledky knihovna zprostředkovává klientovi.

Jelikož moduly mohou implementovat různé typy operací pomocí různých technologií, je nutné ponechat vývojářům velkou volnost v možnostech jejich implementace. Pro účel zajištění jednotné komunikace s moduly je nutné, aby každý implementovaný modul splňoval určité rozhraní. Toto rozhraní zajistí, aby všechny moduly mohly jednotně komunikovat s knihovnou a také, aby knihovna mohla zprostředkovávat jejich funkce klientovi.

Poslední funkcí knihovny je udržování hlavní konfigurace. V této konfiguraci je například uchováván seznam implementovaných modulů, kořenový adresář knihovny nebo například jméno knihovny. Klientská aplikace má možnost tuto konfiguraci změnit a docílit tak jiného chování knihovny.

5.2.1 Rozhraní modulu

Povinné rozhraní modulu slouží především ke komunikaci mezi knihovnou a samotným modulem. Funkcionalitu, kterou modul musí poskytovat, je možné shrnout do následujících bodů:

- inicializační rutina,
- rozhraní poskytované klientům,
- funkce pro validaci šablon.

Prvním požadavkem na rozhraní modulu je existence inicializační rutiny. Pomocí této rutiny je do modulu předána hlavní konfigurace knihovny, která umožňuje modulu zjistit kořenový adresář aplikace a další parametry. Hlavním smyslem této rutiny je inicializace daného modulu. Hlavní knihovna v rámci inicializační smyčky spustí tuto rutinu pro každý registrovaný modul. Uvnitř této rutiny může modul provádět inicializaci vlastních datových struktur nebo vytvoření potřebné adresářové struktury. Dále může modul využít hlavní konfiguraci knihovny k doplnění vlastní lokální konfigurace. Tímto způsobem je zajištěno, že všechny registrované moduly knihovny obdrží globální konfiguraci a dojde k jejich inicializaci.

Další nutnou částí rozhraní modulu jsou funkce, které mají být poskytovány klientovi. K tomuto účelu musí modul poskytovat třídu, která bude tyto funkce implementovat nebo je bude pouze zprostředkovávat pomocí jiných tříd modulu. Tato třída je tedy hlavním funkčním rozhraním modulu, které klientské aplikace mohou využívat. V rámci inicializace celé knihovny dojde nejprve k inicializaci jednotlivých modulů. Po této akci knihovna provede registraci těchto tříd a v udržuje si jejich seznam.

Posledním požadavkem na rozhraní modulu je existence funkcí pro validaci šablon. Tyto funkce musí umožňovat validovat šablony, které se týkají konkrétního modulu knihovny. Modul, který podporuje správu virtualizačního kontejneru Solaris Zones, musí poskytovat funkce pro validaci šablon specifikující neglobální zóny.

Obrázek 5.1: Rozhraní modulu

Vlastní implementace modulu není nijak jinak omezena. Jediným logickým omezením je fakt, že tento modul musí být napsaný v programovacím jazyku Ruby. Pokud modul splní výše zmíněné požadavky, může být jednoduše registrován do knihovny a klientské aplikace ho můžou bezprostředně po inicializaci knihovny využívat. Na obrázku 5.1 je názorně zobrazeno, jakým způsobem knihovna využívá rozhraní modulu a jakým způsobem je modul poskytován klientské aplikaci.

5.2.2 Přesměrování požadavků

Mimo inicializace modulů je hlavní funkcí knihovny přesměrovávat požadavky klientských aplikací na funkční rozhraní implementovaných modulů. K tomuto účelu obsahuje knihovna hlavní třídu, která virtuálně reprezentuje rozhraní všech modulů knihovny. Tato třída se nazývá hlavní rozhraní. Jak již bylo zmíněno, knihovna si udržuje odkazy na hlavní třídy modulů, které reprezentují jejich funkční rozhraní.

V okamžiku, kdy klientská aplikace vznese požadavek na zavolání konkrétní funkce, knihovna za běhu zjistí jakému modulu daná funkce přísluší a vyvolá ji. Pokud neexistuje žádný modul, který umí danou funkci provést, dojde k vyvolání výjimky a aplikace se ukončí. Díky tomuto chování může dojít ke kolizi jmen funkcí. V takovém případě by knihovna použila takovou funkci, kterou by našla jako první v pořadí. Z tohoto důvodu je nutné se vyvarovat opakování jmen funkcí a nejlépe používat pro funkce určitého modulu prefix, který daný modul jasně identifikuje, například jeho jméno.

Toto směrování za běhu aplikace je umožněno díky programovacímu jazyku Ruby a jeho možnosti dynamického volání funkcí za běhu programu. Směrování požadavků ke konkrétním modulům knihovny je demonstrováno na obrázku 5.1.

5.2.3 Generická šablona

Poslední funkcí knihovny je definice generické šablony, která má za úkol specifikovat virtuální stroj. Hlavním úkolem generické šablony je specifikovat typ virtuálního stroje. Tento atribut určuje, který modul knihovny je zodpovědný za zpracování a validaci. Generická šablona by dále měla obsahovat jméno, které bude nějakým způsobem vystihovat a popisovat specifikovaný virtuální stroj. Knihovna tedy zajišťuje validaci těchto dvou atributů a v případě úspěchu předá šablonu zodpovědnému modulu.

Knihovna poskytuje klientským aplikacím funkce pro načítání a validaci šablon. V případě úspěšného načtení šablony knihovna vrátí objekt, který je možné použít v rámci konkrétního modulu. Validace šablony je prováděna

ve dvou krocích. Knihovna nejprve zjistí, zdali daná šablona obsahuje atribut jména a typu. Podle typu šablony se knihovna rozhodne jakému modulu ji předá na druhý krok validace.

5.2.3.1 Struktura šablony

Aby mohla být šablona opakovaně používána pro tvorbu virtuálních strojů, musí být perzistentně uložena v souborovém systému. Pro tento účel je použitý datový formát JSON [35], který slouží pro reprezentaci šablony. Hlavním důvodem využití tohoto formátu je relativně dobrá uživatelská čitelnost a především snadné zpracování pomocí programovacího jazyka Ruby. Uživatel může pro konstrukci šablony použít jednoduchý textový editor nebo grafický editor, který je součástí uživatelského rozhraní klientské aplikace.

Struktura šablony se skládá ze dvou částí. První částí je název a definice typu šablony. Tato hlavička určuje způsob zacházení s danou šablonou. V ukázce kódu 5.1 je naznačeno, jakým způsobem by mohla taková šablona vypadat. Atribut *type* určuje o jaký typ virtuálního stroje se jedná a k jakému modulu knihovny přísluší. Druhou povinnou položkou v šabloně je atribut *name*, který má za úkol popsat funkcionalitu virtuálního stroje. Tečky v ukázce 5.1 reprezentují atributy specifické pro konkrétní typ šablony. Tyto atributy jsou z ukázky vynechány.

Výpis kódu 5.1: Demonstrace generické šablony

```
{
  "name": "template_webserver",
  "type": "szones",
  ...
}
```

5.2.3.2 Validace šablony

Šablona musí poskytovat validní definici virtuálního stroje, aby z ní bylo možné konkrétní virtuální stroj zkonstruovat. Pro tento účel je nutné zavést validaci šablon a jejich atributů. Jelikož je pro ukládání šablon použit datový formát JSON, je pro validaci šablony použité tvz. JSON schéma definované ve specifikaci [36]. Tento dokument je opět ve formátu JSON, ale neslouží pro ukládání dat. Jeho funkcí je definovat formát jiného dokumentu JSON. Pomocí tohoto schématu je možné specifikovat atributy a typy jejich hodnot, které má konkrétní typ dokumentu obsahovat.

Tento nástroj umožňuje definovat, jaké atributy může konkrétní typ virtuálního stroje mít. Pokud uživatel sestrojí nevalidní šablonu virtuálního stroje,

knihovna skrze validaci JSON dokumentu pozná, že se jedná o neplatnou konfiguraci. Knihovna implementuje základní schéma, které slouží pro základní validaci šablon. Jak je vidět v ukázce 5.2, toto schéma vyžaduje, aby v dokumentu byly přítomné atributy *name* a *type*. Moduly aplikace musí implementovat podrobnější schéma, které má definovat konkrétní typ virtuálního stroje.

Aby bylo možné v programovacím jazyku Ruby validovat JSON dokumenty, je nutné použít knihovnu, která bude implementovat JSON schéma. Pro tento účel aplikace využívá volně dostupné řešení *json-schema* [37], které implementuje funkce validace dokumentů typu JSON pomocí schémat.

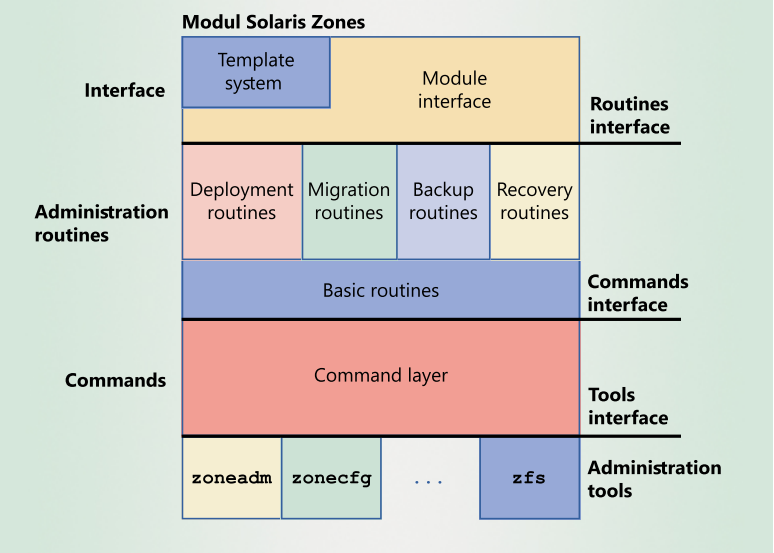
Výpis kódu 5.2: Schéma generické šablony

```
{
  {
  "title": "general-vm-template",
  "description": "Used for general template distinction",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the vm template"
    },
    "type": {
      "enum": [ "szones", "vbox" ],
      "description": "Type of the vm template"
    }

  },
  "required": [ "name", "type" ],
  "additionalProperties": true
}
```

5.3 Modul Solaris Zones

Modul Solaris Zones je hlavním stavebním kamenem celé implementace výsledného nástroje. Tento modul je zařazen do knihovny popsané v kapitole 5.2 a mimo jiné poskytuje základní rutiny pro správu virtualizačního kontejneru Solaris Zones. Aby mohl být tento modul využíván knihovnou, musí implementovat rozhraní definované v kapitole 5.2.1. Takto podmínka zahrnuje především implementaci tříd pro zpracovávání šablon virtuálních strojů.



Obrázek 5.2: Architektura modulu Solaris Zones

V rámci tohoto modulu se jedná o šablony, které specifikují vlastnosti neglobálních zón.

Funkcionalita modulu je rozdělena do několika vrstev, které se vzájemně využívají. Názvy jednotlivých vrstev jsou následující:

- management šablon,
- nástroje pro správu Solaris Zones,
- administrátorské rutiny,
- funkční rozhraní modulu.

Architekturu vrstev modulu je možné pozorovat na obrázku 5.2. V následujících kapitolách bude podrobně popsána funkce jednotlivých vrstev a jejich vzájemná interakce.

5.3.1 Šablona Solaris Zones

Nástroje pro správu Solaris Zones neposkytují možnost vytvářet zóny pomocí jednoho předpisu. Jak bylo popsáno v kapitole 3, k úspěšnému vytvoření neglobální zóny se využívají tři soubory. Prvním a povinným parametrem instalace zóny je její konfigurace. Není podmínkou, aby konfigurace zóny byla v podobě souboru, ale pro automatizaci tohoto procesu je to výhodné. Dále je nutné instalátoru předat definici softwarových balíků, které má nainstalovat. Posledním nepovinným parametrem instalace je konfigurace systémových služeb. Aby bylo možné nainstalovat zónu pomocí jednoho souboru, musí tato šablona kombinovat vlastnosti výše zmíněných souborů.

Kostra šablony pro neglobální zónu je demonstrována v ukázce kódu 5.3. Z ukázky je patrné, že šablona obsahuje tři sekce, které korespondují s jednotlivými soubory vyžadovanými při instalaci. Z ukázky jsou vynechány konkrétní atributy zón. Modul implementuje JSON schéma, které využívá pro validaci šablony a pro mechanizmy jejího zpracování. Úkolem modulu při zpracování šablony je reprodukce souborů s konfigurací, manifestem a systémovým profilem. Tyto soubory nástroj využívá pro instalaci neglobální zóny s parametry, které jsou v dané šabloně specifikovány. Pokud modul zpracovává rutinu, která

Výpis kódu 5.3: Kostra šablony pro neglobální zóny

```
{
  "name": "template_webserver",
  "type": "szones",
  "configuration": {...},
  "manifest": {
    "packages": [...]
   },
   "profile": {...}
}
```

využívá šablonu, je v prvním kroku šablona rozdělena do třech zmíněných částí. V následujícím kroku jsou tyto části převedeny do vnitřní reprezentace a následně zpracovány.

5.3.1.1 Zpracování konfigurace

Konfigurační sekce šablony se skládá z definice globálních atributů zóny popsaných v kapitole 3.3.1 a z definice zdrojů zóny, které jsou popsané v kapitole 3.3.2. Jednoduché globální atributy jsou v šabloně specifikovány přímo pomocí jejich jména a hodnoty. Pro globální atribut typu zóny může definice vypadat následovně `{"brand": "solaris"}`.

Dále tato sekce šablony obsahuje definici zdrojů zóny, které mají složitější strukturu a obsahují několik atributů. Z tohoto důvodu šablona obsahuje speciální atribut `{"resources": []}`, který je typu pole a obsahuje definici všech zdrojů zóny. V rámci jednotlivého zdroje je použita stejná technika definice atributu jako ve výše zmíněném případě globálních atributů.

Cílem zpracování této části šablony je vygenerovat soubor s konfigurací, který má přesně definovanou syntaxi. Pomocí načtené šablony uchované v asociativním poli jsou globální atributy přetransformované do podoby, kterou vyžaduje nástroj `zonecfg(1)`. V případě zdrojů je proveden stejný postup s tím rozdílem, že se před každý zdroj přidá příkaz `add` a jeho definice se ukončí

příkazem `end`. Takto přetransformovaná konfigurace je připravena k použití v nástroji `zonecfg(1)`.

5.3.1.2 Zpracování manifestu

V rámci sekce šablony s názvem `manifest` může uživatel specifikovat softwarové balíky, které má zóna obsahovat. Pro tento účel obsahuje tato část šablony atribut `{"packages": []}`, který je typu pole. Hodnotou každého prvku pole je obyčejný textový řetězec, který obsahuje jméno softwarového balíku. V tomto poli může uživatel specifikovat libovolné množství balíků.

Cílem zpracování této sekce šablony je vytvořit manifest popsaný v kapitole 3.4.1.1. K tomuto účelu si modul drží kopii tohoto souboru, která nese název *manifest_template.xml* a nachází se v kořenovém adresáři aplikace ve složce *szones/manifest*. Při zpracovávání manifestu jsou jednotlivé balíky načteny ze šablony a ve správném formátu vloženy do kopie tohoto souboru. Výsledný soubor je možné použít pro instalaci zóny.

5.3.1.3 Zpracování systémového profilu

Poslední částí zpracovávání šablony je transformace systémového profilu. Tato sekce slouží k nastavení systémových služeb neglobální zóny a má stejnou strukturu jako sekce s konfigurací zóny. Na rozdíl od způsobu zpracování konfigurační sekce je však v tomto případě vyžadován jiný výstup. Cílem této transformace má být soubor v XML formátu popsaný v kapitole 3.4.4.

Pro každou službu existuje korespondující soubor obsahující potřebnou část výsledného XML souboru. Tyto soubory jsou uloženy v kořenovém adresáři aplikace ve složce *szones/profile*. V průběhu zpracovávání šablony jsou jednotlivé soubory načítány a vyplňovány hodnotami ze šablony. Tímto způsobem se zkonstruuje celý soubor, který může být předán instalátoru.

5.3.2 Nástroje

Základním stavebním kamenem modulu je vrstva, která zajišťuje vykonávání potřebných příkazů na příkazové řádce. Tato vrstva poskytuje vyšším vrstvám modulu možnost vykonávání základní administračních příkazů pro správu Solaris Zones a souborového systému ZFS.

Pro účel vykonávání příkazů tato vrstva implementuje třídu, která umožňuje provádění příkazů jak na lokálním tak i na vzdáleném serveru. Tato třída nevykonává daný příkaz okamžitě, ale umožňuje vyšším vrstvám aplikace vykonání odložit. Tento požadavek je na třídu kladen zejména z důvodu efektivity využívání vytvořených SSH spojení. Dále třída umožňuje definovat automatické chování v případě chyby prováděného příkazu. Pro každý nástroj využívaný aplikací je definována sada pravidel, které jsou uplatňovány na chybové výstupy nástrojů a následně jsou vyvolávány příslušné výjimky. Vyšší vrstvy aplikace musí tyto výjimky odchytávat a adekvátně na ně reagovat.

Pomocí výše zmíněné třídy tato vrstva modulu umožňuje volat jednotlivé nástroje s požadovanými parametry a argumenty. Například umožňuje spouštět nástroj `zonecfg(1)` se jménem konfigurované zóny a cestou k souboru s konfigurací. Vyšším vrstvám je poskytnutý standardní výstup a standardní chybový výstup daného nástroje.

5.3.3 Administrátorské rutiny

Hlavní částí modulu je vrstva tříd, které implementují požadované rutiny pro správu Solaris Zones. Rutiny využívají nástrojů implementovaných v nižší vrstvě a pomocí nich vytvářejí sekvence příkazů nutné k vykonání požadované administrátorské činnosti. Tato vrstva poskytuje pokročilejší rutiny pro vytváření neglobálních zón, zálohu, obnovu, ovládání a migraci.

5.3.3.1 Transakce

Jednotlivé rutiny jsou implementované jako transakce. Transakce se skládá z jednotlivých příkazů, které tvoří celek. V případě úspěchu všech příkazů v transakci je celá transakce označena za úspěšnou. Jestliže jakýkoli příkaz v průběhu transakce selže, selže i celá transakce.

V průběhu některých rutin dochází k vytváření dočasných souborů. Tyto soubory slouží například pro přenos konfigurace zóny na vzdálený server nebo pro dočasné uložení diskového obrazu zóny. Soubory, které jsou dočasně vytvořené v průběhu transakce, jsou zaznamenávány a na konci transakce dojde k jejich smazaní. Status transakce nemá na vykonání tohoto procesu vliv. Dále některé transakce vytvářejí nové konfigurace zón, snapshoty konkrétních souborových systémů nebo celé diskové obrazy zón. Některé z těchto entit mají být výstupem transakce. Příkladem může být rutina pro vytváření zóny. V tomto případě má být konfigurace zóny a její diskový obraz výsledkem transakce. Tento případ je nutné rozlišit a tyto entity smazat pouze v případě neúspěchu transakce. Tuto funkcionalitu v rámci modulu zajišťuje třída `Cleanuper`.

Během zálohovacích a migračních rutin je třeba provádět akce, které konkrétním způsobem ovlivní existující zóny. Tyto akce se opět zaznamenávají a v případě neúspěchu transakce se ovlivněné zóny vrátí do původního stavu. Příkladem může být zastavení běžící zóny z důvodu její zálohy nebo migrace. Tuto funkcionalitu v rámci knihovního modulu zajišťuje třída `Rollbacker`.

Rutiny modulu tedy zachovávají stav existujících zón v případě neúspěchu transakce a zajišťují tak konzistentní stav systému. Stejně tak zajišťují, že všechny dočasně vytvořené entity budou po ukončení transakce smazány.

5.3.3.2 Vytváření zón

Hlavní součástí administrátorských rutin jsou funkce pro vytváření zón. Tyto rutiny umožňují vytváření vzdálených i lokálních neglobálních zón z následujících zdrojů:

- ze standardní souborů (konfigurace, manifest, profil),
- ze šablon virtuálních strojů,
- z jiné existující zóny (vzdálené i lokální),
- z archivu zóny.

Prvním způsobem je vytvoření zóny pomocí tří standardních souborů obsahující konfiguraci zóny, manifest a nastavení systémových služeb. V případě vytváření zóny na vzdáleném serveru dojde v první řadě ke zkopírování zdrojových souborů na cílový server. Dále se vytvoří konfigurace zóny pomocí nástroje `zonecfg(1)` ze zdrojového konfiguračního souboru. Následuje spuštění instalace zóny z repozitáře popsané v kapitole 3.4.1, kde se jako parametry předají cesty k souborům s manifestem a systémovým profilem. Tato sekvence příkazů se vykoná lokálně nebo v rámci jednoho SSH spojení s cílovým serverem.

Dále je v rámci modulu poskytnuta podpora pro vytváření zón ze šablon popsaných v kapitole 5.3.1. V průběhu této rutiny nejprve dojde k transformaci šablony, popsané ve stejné kapitole, na tři standardní soubory. Tato transformace proběhne vždy na lokálním serveru. Dále rutina pokračuje stejně jako v případě instalace ze standardních souborů popsané výše.

Výše zmíněné rutiny pro vytváření zóny neměly k dispozici diskový obraz zón a instalace zóny musela vždy probíhat z repozitáře. Následující dva způsoby tvorby neglobálních zón využívají jako zdroj již existující diskový obraz. První z těchto dvou postupů využívá diskový obraz již existující zóny. Kroky této rutiny je možné rozdělit na části získání diskového obrazu a samotné instalace zóny. Obě tyto části mohou být prováděny buď na lokálním serveru nebo na vzdáleném. Server, odkud je zóna získávána, se označuje jako **zdrojový** a server, kde je vytvářená nová zóna, se nazývá **cílový**. Před získáním diskového obrazu se nástroj nejprve musí ujistit, jestli je zóna v konzistentním stavu a zda není spuštěná. Pokud ano, nástroj ji dočasně zastaví. Následuje vytvoření archivu zdrojové zóny pomocí techniky ZFS popsané v kapitole 3.6.1. Vytvořený archiv je následně společně s konfigurací zdrojové zóny přesunut na cílový server a následuje druhá část instalace zóny. Tato část probíhá na cílovém serveru a zde se nejprve nakonfiguruje cílová zóna s pomocí konfigurace zóny zdrojové. Následuje samotná část připojení diskového obrazu z poskytnutého archivu. Po tomto procesu je cílová zóna nainstalována na cílový server. Technika klonování se použije pouze v případě, že zdrojový a cílový server jsou identické stroje.

Poslední podporovanou rutinou pro instalaci neglobálních zón je vytváření z archivu. Tato rutina předpokládá existenci archivu, který může být typu ZFS nebo UAR. V případě tvorby zóny na vzdáleném serveru se nejprve zkopíruje archiv do dočasného adresáře na cílovém serveru. Na cílovém serveru se z archivu extrahuje konfigurační soubor a pomocí něj se zóna nakonfiguruje.

Následuje proces vytvoření kořenového souborového systému cílové zóny z archivu. Po dokončení tohoto procesu je cílová zóna úspěšně nainstalována na cílovém serveru.

Všechny výše zmíněné typy rutin mají několik společných parametrů, které specifikují chování v krajních situacích. Prvním takovým parametrem je *force*. Tento parametr ovlivní rutiny v případě, kdy již existuje zóna se jménem zóny, kterou chce rutina vytvořit. V případě, že je tento parametr zapnutý, rutina danou existující zónu smaže a místo ní nainstaluje zónu novou. V opačném případě skončí rutina s chybou, že se nepodařilo zónu nainstalovat. Druhým parametrem rutin je *boot*. Tento parametr určuje, zda se vytvořená zóna má rovnou spustit. Implicitní hodnota obou parametrů je nastavená na *false*. Všechny rutiny pro vytváření neglobálních zón jsou implementované v rámci třídy `DeploymentRoutines`.

5.3.3.3 Záloha a obnova zón

Záloha zón může podle kapitoly 3.6 probíhat dvojím způsobem. Tyto dva způsoby se liší především v technologii, která vytváří danou zálohu. V obou případech se jedná o vytvoření archivu kořenového souborového systému neglobální zóny. Jeden způsob používá standardní techniku systémové archivace pomocí UAR a druhý způsob používá přímo nástroje souborového systému ZFS. Rutiny pro zálohu podporují oba tyto způsoby a liší se pouze v technice vytvoření daného archivu.

Oba typy zálohovacích rutin je možné provádět na lokálních i vzdálených serverech. V prvním kroku je nutné uvést danou neglobální zónu do konzistentního stavu a případně ji zastavit. Následuje proces vytváření archivu, který se liší v závislosti na použité technice. Po dokončení archivace je záloha hotová.

V obou případech zálohovacích rutin je možné použít volitelný parametr *archive_destination* určující, na který server se má záloha zkopírovat. Implicitně se záloha vytváří na serveru, kde se daná zóna nachází.

Obnova zóny předpokládá existenci její zálohy. Jelikož se jedná o vytvoření zóny z archivu, je tento proces stejný s procesem vytváření zón z archivu popsaným v předchozí kapitole. Všechny zálohovací rutiny jsou v modulu implementované ve třídě `BackupRoutines`.

5.3.3.4 Migrace zón

Migrace v rámci Solaris Zones je přesun neglobální zóny z jednoho virtualizačního serveru na druhý. Jedná se jak o přesun konfigurace zóny tak i o přesun diskového obrazu. Tato administrační rutina se v mnoha ohledech podobá rutině pro vytvoření neglobální zóny z jiné zóny, která již existuje. Podstatou je vytvoření archivu zóny na zdrojovém serveru a přesun tohoto archivu na server cílový. Vytvoření archivu je tedy možné provést dvojím způsobem.

Migrační rutiny implementovaného nástroje budou umožňovat migraci zón jak s použitím archivu ZFS tak i s použitím archivu UAR.

Hlavním rozdílem migrace oproti vytváření zóny z již existující zóny je v tom, že původní zóna se v případě úspěchu transakce smaže. Zóna na zdrojovém serveru se smaže až v případě, kdy je zóna kompletně nakonfigurována a zdárně nainstalována na cílovém serveru. Dřívější smazání zdrojové zóny by mohlo vést ke ztrátě dat.

Dalším rozdílem migrace oproti vytváření zóny z již existující zóny je typ používaných příkazů. Před procesem vytváření archivu jakéhokoli typu dojde na zdrojovém serveru k použití nástroje `zoneadm(1)` a jeho příkazu `detach`, který bezpečně odpojí diskový obraz zóny od její konfigurace. Po tomto kroku následuje vytvoření archivu a jeho přesun na cílový server. Jakmile se dokončí přenos archivu je zóna na cílovém serveru nakonfigurována a následně je její obraz připojen pomocí příkazu `attach`, který má jako argument vytvořený archiv. V případě úspěchu může být konfigurace zóny i jejího obrazu smazána ze zdrojového serveru.

Migrační rutiny poskytují ještě jeden typ přenosu diskového obrazu zóny a to přímo pomocí příkazu `zfs send` a `zfs recv`. Spuštění prvního příkazu na zdrojovém server a druhého příkazu na druhém serveru v rámci SSH spojení zajistí přenos zdrojového souborového systému z jednoho serveru na druhý. Příkaz `attach` po tomto přenosu nemá téměř žádnou práci, protože nemusí souborový systém extrahovat z archivu.

Všechny výše zmíněné funkce umožňují migraci mezi všemi hosty dané infrastruktury. Těmito funkcemi je umožněno migrovat lokální zónu na vzdálený server a naopak. Také je umožněna migrace zóny mezi dvěma vzdálenými hosty. V rámci migračních rutin je možné specifikovat, že zóna na cílovém serveru se může jmenovat jinak než na zdrojovém. Migrační rutiny jsou v implementovaném modulu zahrnuty ve třídě s názvem `MigrationRoutines`.

5.3.3.5 Rutiny pro správu zón

Modul Solaris Zones mimo výše zmíněných pokročilejších rutin poskytuje i základní rutiny pro manipulaci a správu neglobálních zón. V těchto rutinách je zahrnuto spouštění, násilné i nenásilné vypnutí, restart nebo úplné odstranění zóny ze systému. Všechny tyto akce je možné provádět jak na lokálním tak i na vzdáleném serveru. Třída poskytující tyto rutiny v rámci implementovaného modulu se nazývá `BasicRoutines`.

5.3.4 Funkční rozhraní modulu

Poslední částí modulu pro správu virtualizačního kontejneru Solaris Zones je rozhraní, které je nabízeno klientským aplikacím prostřednictvím knihovny. Pro tyto účely je vytvořena speciální třída, která zprostředkovává administrátorské rutiny popsané výše. Princip rozhraní funguje stejně jako v případě

knihovny. Toto rozhraní má zaregistrované všechny třídy, jejichž metody chce veřejně poskytovat knihovně a klientským aplikacím. Pokud knihovna obdrží požadavek na volání konkrétní rutiny, přesměruje tento požadavek právě na toto rozhraní. Rozhraní vyhledá v rámci zaregistrovaných tříd, zad umí obsloužit konkrétní požadavek. Pokud nějaká z tříd modulu umí danou rutinu provést, rozhraní vrátí její návratovou hodnotu. V opačném případě je vyvolána výjimka, kterou knihovna odchytí a případně bude vyhledávat v ostatních implementovaných modulech.

Ve výsledné implementaci je rozhraní reprezentováno třídou `SZONESAPI`, která má zaregistrované třídy korespondující s rutinami popsanými v kapitole 5.3.3.

5.4 Klientská aplikace

Další významnou částí implementovaného nástroje je klientská aplikace. Hlavním úkolem tohoto funkčního bloku nástroje je skrývat implementační detaily knihovny a jejích modulů. Funkcionalitu knihovny tato klientská aplikace zprostředkovává uživateli pomocí následujících komponent:

- uživatelské rozhraní (CLI),
- správa vzdálených hostů,
- uživatelský žurnál.

Klientská aplikace poskytuje rutiny pro správu Solaris Zones popsané v kapitole 5.3 Tyto rutiny může uživatel využívat pomocí uživatelského rozhraní aplikace. Hlavním cílem tohoto rozhraní je přehlednost a možnost vykonávat rutiny pro větší množství neglobálních zón. Uživatel má možnost jednoduše specifikovat konkrétní zóny v uživatelském rozhraní a klientská aplikace se postará o vykonání daného příkazu pro všechny specifikované neglobální zóny.

Jak bylo uvedeno, nástroj musí umožňovat správu vzdálených zón. Tato funkcionalita je již implementována v modulu Solaris Zones. Klientská aplikace musí tedy implementovat pouze způsob, jakým způsobem jednoznačně identifikovat neglobální zónu v rámci infrastruktury. Dále klientská aplikace spravuje databázi virtualizačních hostů. Pomocí této databáze je možné vykonávat některé rutiny hromadně napříč všemi registrovanými hosty.

Poslední funkcionalitou klientské aplikace je udržování tzv. uživatelského žurnálu. Tento žurnál slouží uživateli pro uchovávání stavů jednotlivých neglobálních zón. V případě změny stavu registrované zóny jiným uživatelem, je aplikace schopna dohledat, že došlo ke změně.

5.4.1 Rozhraní na příkazové řádce

Hlavní ovládací prvek klientské aplikace, ale i celého nástroje, je rozhraní na příkazové řádce. Důvody pro výběr tohoto rozhraní byly popsány v kapi-

tole 4.3.1. Uživatel může toto rozhraní ovládat pomocí příkazů, které určují typ prováděné operace. Příkazy je možné rozdělit do dvou skupin. První skupina příkazů pouze mění vnitřní stav klientské aplikace. Druhá skupina příkazů slouží ke správě a manipulaci s neglobálními zónami. Téměř všechny tyto příkazy vyžadují jako argument jednu nebo více neglobálních zón, nad kterými se má provést požadovaná akce. Pro tento účel je nutné zavést unikátní identifikátor, který přesně specifikuje neglobální zónu v rámci několika hostů. Jelikož doménové jméno stroje musí být v rámci sítě unikátní, bude v identifikátoru figurovat. Druhou částí identifikátoru bude jméno neglobální zóny v rámci jednoho hosta. Toto jméno je v prostředí jedné globální zóny také unikátní. Kombinací těchto dvou identifikátorů je možné sestrojit název pro neglobální zónu, který bude unikátní v rámci celé infrastruktury. Pro identifikaci zóny na příkazové řádce bude uživatel používat název zóny a doménové jméno hosta spojené dvojtečkou. Globální identifikátor může vypadat následovně *z1:host1*.

V následujících kapitolách je popsána funkcionalita a parametry jednotlivých příkazů uživatelského rozhraní, které slouží pro správu virtualizačního kontejneru Solaris Zones.

5.4.1.1 Deploy

Prvním zástupcem příkazů pro práci s neglobálními zónami je `deploy`. Úkolem tohoto příkazu je tvorba zón ze zadaných parametrů. Jako zdroj pro vytvoření zóny umí tento příkaz využívat všechny způsoby, které poskytuje implementovaný modul Solaris Zones. Konkrétně se jedná o vytvoření pomocí souborů, šablony nebo existující zóny v rámci infrastruktury. Dále tento příkaz poskytuje interaktivní instalaci, která využívá grafického rozhraní. Tento způsob je podrobněji popsán v kapitole 5.5.2. Pokud uživatel nespecifikuje jinak, použije se pro vytvoření zóny právě interaktivní instalace.

Jako argumenty příkazu musí uživatel specifikovat minimálně jednu neglobální zónu pomocí výše zmíněného identifikátoru. Jméno zóny se použije při vytváření a doménové jméno se použije pro připojení ke vzdálenému hostu. Pro všechny takto specifikované zóny je použit stejný zdroj a příkaz je vykonán paralelně pro každou z nich.

Volitelnými parametry jsou *boot* a *force*, jejichž funkčnost byla popsána v kapitole 5.3.3.2.

5.4.1.2 Backup

Příkaz `backup` slouží pro zálohování zón. Tento příkaz umožňuje zálohovat zóny dvěma způsoby. Jedná o archivaci pomocí archivů ZFS nebo UAR. Struktura tohoto příkazu je velmi podobná předchozímu. Uživatel nejprve musí definovat seznam zón pomocí globálních identifikátorů. Aplikace se následně paralelně připojí ke každému specifikovanému hostu a začne vytvářet zálohy konkrétních zón. V případě zálohy pomocí ZFS archivu se záloha všech zón

na konkrétním serveru provádí paralelně. V druhém případě se záloha provádí v rámci hosta sériově, jelikož archivace pomocí UAR neumožňuje paralelní tvorbu archivů na jednom serveru.

Implicitně se záloha vytváří na serveru korespondujícím se zálohovanou zónou. Pomocí volitelných parametrů *destination* a *path* se toto chování dá změnit a zkopírovat zálohu na vzdálený server.

5.4.1.3 Recovery

Opačným příkazem k příkazu `backup` je `recovery`. Tento příkaz umožňuje obnovovat zóny z dříve vytvořených záloh. Podle typu dané zálohy nástroj rozhodne, jakou techniku obnovy použije. Uživatel tedy nemusí specifikovat, že používá zálohu typu UAR nebo ZFS. Syntaxe tohoto příkazu je stejná jako v případě zálohy. Uživatel specifikuje identifikátory zón, které chce obnovit a pomocí parametru *archives* specifikuje dané zálohy. Pořadí identifikátorů musí odpovídat pořadí archivů, jinak dojde k záměně diskových obrazů zón.

Volitelné parametry tohoto příkazu jsou *boot* a *force*, které umožňují obnovované zóny rovnou spustit nebo přepsat existující zóny.

5.4.1.4 Migrate

Příkaz `migrate` umožňuje přesun neglobálních zón mezi dvěma servery. Tento příkaz umožňuje přenášet větší množství zón libovolně rozložených v infrastruktuře na jeden konkrétní server. Cílový server může být buď lokální nebo vzdálený. Migrace je implementována v několika způsobech. Jedná se o migraci pomocí přímé metody nebo pomocí archivu ZFS a UAR.

Argumenty tohoto příkazu jsou identifikátory neglobálních zón, které chce uživatel přesunout. Pomocí parametru *destination* je možné určit na jaký server mají být specifikované zóny přesunuty. Pokud uživatel nespecifikuje jinak, jsou zóny migrovány na lokální server. Typ migrace se určuje parametrem *type*. Pokud není využíván způsob migrace pomocí UAR, jsou všechny migrace prováděny paralelně.

5.4.1.5 Template

Speciálním příkazem v rámci uživatelského rozhraní je `template`. Tento příkaz umožňuje vytvářet v infrastruktuře instance šablon neglobálních zón. V kombinaci s příkazem `deploy` a jeho parametrem *template* je možné vytvářet zóny dané konfigurace opravdu rychle.

Příkaz `template` obsahuje dva podpříkazy. Prvním z nich je `create`, který umožňuje vytvořit instanci šablony na jakémkoli vzdáleném serveru. Implicitní chování tohoto příkazu vytvoří instanci specifikované šablony na každém serveru, který je v aplikaci registrovaný. Chování tohoto příkazu je podobné jako vytváření zóny ze šablony. Následně může uživatel použít příkaz `deploy` s parametrem *template* a vytvořit specifikované zóny pomocí

funkce klonování. Tento způsob vytváření zóny je výrazně rychlejší než všechny ostatní a navíc šetří místo na disku.

Druhým podpříkazem je `destroy`, který implementuje opačnou funkcionalitu k příkazu `create`. Tento příkaz tedy smaže specifikovanou instanci šablony ze specifikovaných serverů. Implicitně je šablona smazána ze všech registrovaných serverů.

Pomocí parametrů *hosts* je možné specifikovat, na kterých serverech se má šablona vytvořit nebo smazat. Parametr *force* potom slouží k přepsání existujících instancí šablon se stejným jménem.

5.4.1.6 Manage

Poslední příkaz související se správou neglobálních zón je `manage`. Tento příkaz obsahuje několik podpříkazů, které jako argumenty vyžadují globální identifikátory zón. Názvy jednotlivých podpříkazů korespondují s akcí, která se má se specifikovanými zónami provést. Jedná se o příkazy spuštění, nenásilného vypnutí, zastavení, restartu a kompletní odinstalace zóny ze systému. Tyto podpříkazy nemají žádné parametry a pro každou specifikovanou zónu se provádí paralelně.

5.4.1.7 Logování

Uživatelské rozhraní vytváří ve složce *~/.szmgmt/logs*, která se nachází v domovském adresáři uživatele, soubory s průběhem jednotlivých transakcí. Pro každou akci s konkrétní zónou je vytvořený samostatný soubor, kde jsou uloženy informace o provádění konkrétní příkazů. V informačním výpisu uživatelského rozhraní se uživatel dozví, jaký soubor souvisí s konkrétní transakcí. V případě chyby transakce zde uživatel může dohledat co danou chybu způsobilo.

5.4.2 Správa vzdálených hostů

Další komponentou klientské aplikace je správa vzdálených hostů. Tato komponenta umožňuje uživateli registrovat vzdálené hosty do vnitřního stavu aplikace. Hromadné akce nabízené uživatelským rozhraním jsou vykonávány právě pomocí této databáze hostů. Dalším důvodem, proč je nutné uchovávat hosty v aplikaci, je specifikace připojení. Jelikož se aplikace připojuje ke vzdáleným hostům pomocí SSH, je nutné specifikovat uživatelské údaje. Pro tyto účely je vytvořena databáze hostů uložená v domovském adresáři uživatele ve složce *~/.szmgmt/hosts/*.

Tato komponenta se ovládá pomocí uživatelského rozhraní a konkrétně pomocí příkazu `host`. Tento příkaz má následující tři podpříkazy:

- `add`,
- `delete`,

- `list`.

Příkaz `add` slouží pro registrování hostů do aplikace. Jako argument požaduje doménové jméno vzdáleného serveru, které je následně uloženo do databáze. Volitelné parametry tohoto příkazu umožňují specifikovat uživatele a jeho privátní klíče, které mají být použity pro připojování k tomuto serveru. Implicitní hodnoty těchto parametrů jsou demonstrovány v ukázce kódu 5.4.

Druhý příkaz `delete` slouží k odstranění daného hosta z databáze aplikace. Při této akci dojde i ke smazání specifikace SSH připojení k tomuto serveru. Poslední příkaz `list` vypíše na standardní výstup seznam všech registrovaných hostů.

Funkcionalita této komponenty značně souvisí se všemi příkazy uživatelského rozhraní, které se připojují ke vzdáleným hostům. Uživatel specifikuje neglobální zóny pomocí globálních identifikátorů, které obsahují doménová jména. V okamžiku připojování ke konkrétnímu vzdálenému serveru je použito právě nastavení uložené v databázi. Pokud pro daného hosta neexistuje záznam v databázi, jsou použity implicitní hodnoty.

Pro každého uživatele tato komponenta udržuje jeho vlastní databázi v domovském adresáři. Hlavním důvodem pro toto umístění je izolace pohledů jednotlivých uživatelů aplikace.

5.4.2.1 Struktura databáze

Databáze hostů je uložena v několika souborech, které se nacházejí v adresáři *~/.szmgmt/hosts* konkrétního uživatele. Hlavní soubor databáze se nazývá *hosts.json* a obsahuje seznam doménových jmen jednotlivých hostů. Pro každý záznam v tomto souboru je vytvořen další soubor, který obsahuje specifikaci SSH připojení. Název tohoto souboru odpovídá doménovému jménu konkrétního serveru. Všechny soubory databáze jsou uloženy v datovém formátu JSON. V ukázce kódu 5.4 je zobrazeno implicitní nastavení pro uživatele *zadmin* a server s doménovým jménem *shost1*.

Výpis kódu 5.4: Implicitní nastavení parametrů SSH připojení

```
{
  "host_name": "shost1",
  "user": "zadmin",
  "keys": ["~/.ssh/id_rsa"],
  "timeout": 1
}
```

5.4.3 Uživatelský žurnál

Poslední komponentou klientské aplikace je uživatelský žurnál. Solaris Zones neposkytují žádný nástroj pro přidělování zón jednotlivým uživatelům. Každý privilegovaný uživatel má možnost jakýmkoli způsobem ovlivnit kteroukoli zónu v rámci dané globální zóny. Z tohoto důvodu se může stát, že neglobální zóna vytvořená jedním uživatelem bude smazána nebo změněna druhým uživatelem. Standardně Solaris Zones na tuto skutečnost uživatele nijak neupozorní.

Uživatelský žurnál slouží pro sledování stavů jednotlivých neglobálních zón, se kterými uživatel nějakým způsobem manipuloval. Tyto stavy budou sloužit k informování uživatele o tom, zda se daná zóna od předchozí manipulace nějakým způsobem změnila. Konkrétně, zda se změnil její stav nebo diskový obraz. Žurnál umožňuje také detekci nových zón, které se v infrastruktuře od posledního spuštění aplikace objevily.

Práce této komponenty je z větší části automatická, ale také je umožněno její přímé ovládání pomocí uživatelského rozhraní. Automatická práce žurnálu spočívá v aktualizování stavů jednotlivých zón vždy, když uživatel vyvolá nějakou administrátorskou rutinu. Při vytvoření zóny dojde k přidání konkrétního stavu do uživatelského žurnálu. V případě odstranění zóny je odstraněn i stav z databáze. Pro manuální správu žurnálu poskytuje uživatelské rozhraní příkaz `journal`, který má následující podpříkazy:

- `track`,
- `detrack`,
- `update`,
- `clear`,
- `status`,
- `list`.

Příkaz `track` slouží pro zaregistrování dané zóny. Díky tomuto příkazu není nutné, aby uživatel s danou zónou manipuloval. Pokud uživatel spustí tento příkaz a předá mu jako argument globální identifikátor zóny, dojde k získání aktuálního stavu zóny a zaregistrování do žurnálu. Od této chvíle se konkrétní zóna zařadila do seznamu sledovaných. Opačnou akci provádí příkaz `detrack`, který ze žurnálu odstraní záznam o dané zóně.

Pokud chce uživatel znát stavy všech zón v rámci registrovaných hostů, může použít příkaz `update`. Tento příkaz se pomocí SSH přihlásí na všechny registrované hosty a stáhne informace o všech neglobálních zónách. Tyto informace jsou následně uloženy do žurnálu a všechny zóny se stávají sledovanými. Pro opačnou funkcionalitu slouží příkaz `clear`, který kompletně vymaže databázi sledovaných zón.

Poslední dvojicí podporovaných příkazů jsou `status` a `clear`. Oba tyto příkazy mají za úkol informovat uživatele o stavech sledovaných zón. Příkaz `list` jednoduše vypíše všechny zaregistrované zóny a jejich stavy. Tento příkaz se nikam nepřipojuje a vypisuje informace přímo ze žurnálu. Příkaz `status` slouží pro porovnání stavů uložených v žurnálu s aktuálním stavem všech zón v rámci registrovaných hostů. Pokud se některý uložený stav neshoduje se stavem aktuálním, je o tom uživatel prostřednictvím standardního výstupu informován. Program také označí zóny, které se nově objevily v infrastruktuře. Stejně jako v případě správy vzdálených hostů, je žurnál udržován pro každého uživatele zvlášť.

5.4.3.1 Struktura databáze

Uživatelský žurnál je uložen v souboru *~/.szmgmt/journal/tracked_zones.json* v domovském adresáři konkrétního uživatele. V tomto souboru jsou uložené atributy sledovaných zón, které slouží pro definici konkrétního stavu zóny. Kombinace hodnot těchto atributů by měla zajistit detekci změn, které nemusí být na první pohled patrné. Příkladem takové změny může být například přeinstalování zóny a navození stejného stavu. Při této akci zůstane jméno, konfigurace a stav zóny stejné, ale změní se diskový obraz. Pro tento účel udržuje žurnál u každé sledované zóny atribut *UUID*, který jednoznačně definuje diskový obraz zóny. Porovnáním uložených atributů s aktuálním stavem zóny je možné zjistit, zda bylo se zónou nějakým způsobem manipulováno. Ukládané atributy zóny a struktura záznamu je patrná z ukázky kódu 5.5. V uživatelském žurnálu je podobný záznam udržován pro každou sledovanou zónu.

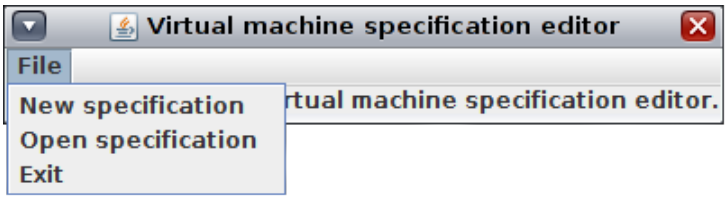
Výpis kódu 5.5: Ukázka záznamu v uživatelském žurnálu

```
"zweb1b-clone:localhost": {
  "zone_name": "zweb1b-clone:localhost",
  "zone_state": "running",
  "zone_path": "/system/zones/zweb1b-clone",
  "zone_uuid": "a000b847-7854-4939-8ac1-8ae5d9013072",
  "zone_brand": "solaris",
  "zone_ip": "excl"
}
```

5.5 Grafické rozhraní

Poslední součástí nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones je grafické uživatelské rozhraní. Jeho hlavní účel je

Obrázek 5.3: Ovládací menu editoru šablon



zvyšování uživatelského komfortu a odstínění uživatele od implementačních detailů. Implementované grafické rozhraní slouží jako nadstavba nad tvorbou šablon specifikovaných v kapitole 5.3.1 a umožňuje uživateli jejich tvorbu a editaci. Pro tyto účely byl vytvořen graficky orientovaný editor, které pomocí grafických elementů umožňuje vyplňování příslušných atributů neglobálních zón. Stejný princip je využitý i pro interaktivní instalaci zón.

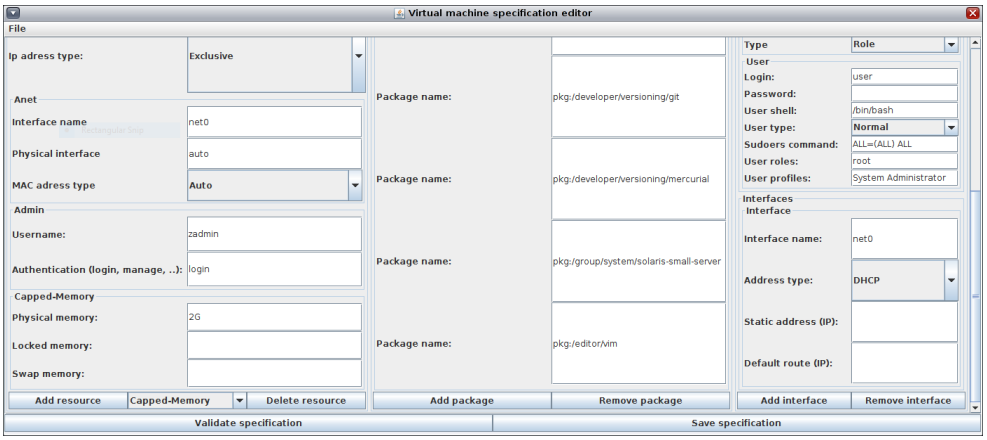
Implementované grafické rozhraní využívá grafické knihovny Swing, která je podporována pouze na platformě JVM. Z důvodů uvedených v kapitole 5.1.1 je použití grafického rozhraní podmíněno využitím interpretu JRuby. V případě, že uživatel spustí grafické rozhraní na jiné platformě, aplikace se ukončí a grafické rozhraní se nezobrazí. Ostatní funkce nástroje nejsou závislé na použitém interpretu.

5.5.1 Editor šablon

Prvním využitím grafického rozhraní je editor pro manipulaci se šablonami zón. Tento editor se spouští pomocí příkazu `editor`, který je součástí uživatelského rozhraní klientské aplikace. Po spuštění tohoto příkazu je uživateli zobrazeno grafické okno, které slouží pro editaci šablon.

Horní část editoru obsahuje ovládací panel, který umožňuje uživateli načítat šablony nebo vytvářet nové. Součástí tohoto panelu je také tlačítko pro ukončení editoru. Pokud uživatel zvolí možnost načtení šablony, je pomocí několika dialogových oken dotázán na cestu k dané šabloně. Po úspěšném zadání cesty se uživateli v prostřední části editoru zobrazí formulář vyplněný pomocí dat z načtené šablony. Pokud uživatel vybere druhou možnost, kterou je vytvoření šablony, je v prostřední části editoru zobrazen ten samý formulář, ale vyplněný implicitními hodnotami. Ovládací menu společně s úvodním oknem editoru je zobrazeno na obrázku 5.3. Formulář pro vyplňování atributu šablony je umístěn ve středu editoru a tvoří jeho nejpodstatnější část. Jelikož má struktura šablony neglobální zóny tři části, skládá se i hlavní okno editoru ze tří oddělených částí. Tyto části přímo odpovídají jednotlivým sekcím šablony a nazývají se konfigurace, manifest a nastavení. Jak je patrné z obrázku 5.4, každá sekce editoru má ve spodní části ovládací prvky. Tyto prvky slouží k přidávání a odebírání konfiguračních prvků ze šablony. Část s konfigurací umožňuje vybírat z několika různých typů zdrojů, které lze v editoru použít. V druhé části editoru lze přidávat a odebírat softwarové balíky, které mají

Obrázek 5.4: Formulář editoru šablon



být v instancích dané šablony nainstalované. Poslední část umožňuje přidávat a odebírat nastavení pro definované síťové adaptéry.

Poslední částí editoru je spodní ovládací panel, který obsahuje dvě funkční tlačítka. V případě použití prvního tlačítka pro validaci se pomocí knihovny vyvolá příslušná funkce a šablona se ověří. Uživatel je o výsledku informován pomocí dialogového okna. Druhé tlačítko slouží k uložení šablony do souboru a uživatel je vyzván k zadání jména šablony a adresáře, kam se šablona má uložit. V obou případech dojde k rekonstrukci datového formátu JSON popsaného v kapitole 5.3.1, který je vyplněn pomocí hodnot specifikovaných uživatelem v editoru. Tato konstrukce šablon je pro uživatele jednodušší, protože vždy vygeneruje validní šablonu. V případě manuální tvorby musí uživatel zajistit její validitu.

5.5.2 Interaktivní instalace

Grafické rozhraní je v nástroji využito ještě v případě, že uživatel spustí příkaz `deploy` s parametrem *interactive*. V tomto případě je spuštěna interaktivní instalace a uživateli je opět zobrazeno grafické okno. Toto okno je stejného charakteru jako výše popsaný editor a slouží ke specifikaci vlastností vytvářených zón. Uživatel nyní nebude mít na výběr z načtení šablon, ale bude muset zadat hodnoty manuálně. Některé atributy jsou vyplněny implicitními hodnotami a uživatel musí zadat minimálně počáteční heslo uživatele *root* a konfiguraci počátečního systémového uživatele.

Kapitola 6

Testování a měření

Poslední kapitola této diplomové práce popisuje testování funkcionality nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones. Zaměřuje se především na testování hlavních scénářů použití nástroje a zkoumá jeho chování. Na začátku této kapitoly je definováno prostředí, ve kterém byly testy prováděny. Následuje série testů, které zkoumají funkcionalitu nástroje v konkrétních případech použití. Kapitola je zakončena měřením, které zkoumá dobu trvání některých funkcí nástroje.

6.1 Definice testovacího prostředí

Pro účely testování výše zmíněného nástroje bylo nutné vytvořit prostředí odpovídající jeho cílové platformě. Toto prostředí obsahuje několik virtualizačních serverů s operačním systémem Solaris, který bude poskytovat své prostředky neglobálním zónám. Tyto servery jsou propojeny počítačovou sítí, pomocí které je lze ovládat. Tato infrastruktura byla virtualizovaně vytvořena na fyzickém systému s následujícími parametry:

- procesor Intel(R) Xeon(R) CPU E3-1230 v3 (3.30Ghz),
- RAM 16GB,
- operační systém Windows 10 (64-bit).

Virtualizace architektury byla docílena pomocí virtualizační technologie Virtualbox, která umožňuje spouštění virtuálních počítačů v rámci jiného operačního systému. Pomocí této technologie byly vytvořeny tři virtuální stroje s operačním systémem Solaris ve verzi 11.3. Tyto stroje byly propojeny pomocí virtuální počítačové sítě a nakonfigurovány tak, aby se na ně dalo připojovat pomocí SSH. Dále byly jednotlivým strojů přiřazena doménová jména *shost*, *shost1* a *shost2*. Tato doménová jména byla nakonfigurována pomocí souboru

/etc/hosts na všech zmíněných virtuálních počítačích. Toto nastavení umožňuje používat specifikovaná doménová jména místo IP adres a zjednoduší tak identifikaci strojů v testovacích ukázkách.

Jelikož provozování virtualizační technologie Solaris Zones vyžaduje nemalé množství výpočetních prostředků, bylo nutné dostupné prostředky fyzického systému rozdělit mezi virtuální stroje. Z tohoto důvodu byly každému virtuálnímu počítači přiřazeny následující výpočetní prostředky:

- jedno jádro fyzického procesoru,
- RAM 3 GB,
- virtuální disk 50 GB (HDD).

Na virtuální počítač s doménovým jménem *shost* byl nainstalován interpret programovacího jazyka Ruby ve verzi 2.4.2. Dále byla na stejný počítač nainstalována Java ve verzi 1.8.0_60 a následně druhý interpret programovacího jazyka Ruby tentokrát ve verzi 2.3.3 a implementaci JRuby. Pokud není uvedeno jinak, testovaný nástroj byl vždy spouštěn z virtuálního počítače s doménovým jménem *shost*.

Posledním učiněným krokem byla konfigurace uživatele *zadmin*, který má práva na vykonávání příkazů nutných ke správnému chodu implementovaného nástroje. Tyto nástroje byly vyjmenovány v kapitole 4.2.2. Uživatel byl pomocí nástroje RBAC vytvořen a nakonfigurován na všech vytvořených virtuálních počítačích.

6.2 Testování scénářů použití

V následujících kapitolách je popsáno akceptační testování hlavních scénářů použití nástroje pro podporu automatické správy Solaris Zones. Pro testování aplikace bylo vždy použito popsané prostředí, pokud není uvedeno jinak. Na začátku každého scénáře je stanoven cíl, který by uživatel mohl chtít pomocí implementovaného nástroje dosáhnout. Následně je popsán stav prostředí, ve kterém se systém nacházel před provedením konkrétní akce. Dále byl proveden korespondující příkaz v uživatelském rozhraní nástroje, který má splnit stanovený cíl. Výsledek tohoto kroku je ověřen pomocí systémových příkazů a v závěru scénáře je rozhodnuto, zda bylo dosaženo stanoveného cíle.

6.2.1 Vytvoření neglobálních zón ze šablony

Pro komplexní otestování funkcionality implementovaného nástroje byl zvolen scénář vytvoření několika neglobálních zón pomocí šablony. Hlavní důvod pro výběr tohoto scénáře je, že se do tohoto procesu zapojují téměř všechny části nástroje.

Cílem tohoto scénáře je vytvoření několika neglobálních zón na různých hostech v rámci dané infrastruktury. Jako zdroj byla použita šablona popsaná v kapitole 5.3.1. Ze šablony byly vybrány některé důležité vlastnosti, které mají vytvořené zóny mít. Typ zóny byl stanoven jako *solaris* s exkluzivní IP adresou. Dále má nainstalovaná zóna obsahovat softwarové balíky pro správu zdrojového kódu. Tyto balíky obsahují nástroje `hg` a `git`. Šablona také definuje počáteční heslo uživatele *root* a nastavuje typ tohoto uživatelského účtu na roli. Vedle uživatele *root* je v šabloně definován počáteční systémový uživatel, který má být zároveň systémovým administrátorem. Vytvářené neglobální zóny mají mít jedno síťové rozhraní se jménem *net0*, které bude konfigurováno automaticky pomocí DHCP. Takto definovaná šablona je uložena na severu s doménovým jménem *shost*.

Příkaz pro vytvoření čtyř zón pomocí uživatelského rozhraní nástroje je v ukázce kódu B.1 na první řádce. Tento příkaz říká, že mají být vytvořeny zóny *zdev* a *zdev1* na lokálním serveru a zóna *zdev* na vzdálených serverech *shost1* a *shost2*. Jako parametr *specification* je udána cesta k šabloně s výše popsanými vlastnostmi. Dále byl příkazu předán parametr *boot*, který má rovnou spustit vytvořené zóny.

Z konce standardního výstupu nástroje v ukázce výpisu programu B.1 je patrné, že vytvoření všech neglobálních zón proběhlo v pořádku. Jelikož se pro vytváření zón používá stejná rutina a stejná šablona, musí mít všechny stejné parametry. Pro otestování korektnosti práce nástroje byla použita neglobální zóna *zdev* na vzdáleném serveru *shost2*. Korektní vytvoření a spuštění zóny bylo ověřeno pomocí nástroje `zlogin(1)`, který umožňuje připojení k příkazové řádce dané zóny. Úspěšné přihlášení v ukázce výpisu programu B.4 signalizovalo hned několik věcí. Za prvé se zdárně podařilo vytvořit a spustit danou zónu a za druhé byly správně nakonfigurovány uživatelské systémové služby pomocí atributů ze šablony. Dále je z ukázky výpisu programu B.4 patrné, že došlo k vytvoření síťového adaptérů *net0* a jeho automatické konfigurace pomocí služby DHCP. Přítomnost softwarových balíků byla otestována pomocí jejich rozhraní na příkazové řádce, jak je vidět v ukázce výpisu programu B.4. Posledním kritériem úspěchu bylo správné nakonfigurování počátečního systémového uživatele. Jméno a heslo bylo ověřeno již při přihlašování do zóny. Zbývalo tedy ověřit, zda uživatel má práva systémového administrátora, což bylo provedeno pomocí příkazu `profile`. Výpis programu na ukázce B.4 je zkrácený, ale obsahuje profil *System Administrator*.

Pomocí výše zmíněných testů bylo ověřeno, že se daná zóna vytvořila, spustila a že měla vlastnosti specifikované v použité šabloně. Stejným způsobem byly ověřeny i ostatní vytvářené zóny. Jelikož tyto zóny vykazovaly stejné chování a vlastnosti, byl tento test uzavřen a považován za splněný.

6.2.2 Využití uživatelského žurnálu

Dalším využitím implementovaného nástroje může být využití uživatelského žurnálu. Cílem následujícího scénáře je kontrola funkcionality uživatelského žurnálu a jeho schopnosti informovat uživatele o změnách neglobálních zón v rámci infrastruktury. K tomuto účelu byl využitý stav, ve kterém se systém nacházel po testování předchozího scénáře. Součástí předchozího scénáře bylo vytvoření čtyř zón pomocí implementovaného nástroje. Před tímto vytvořením se v systému nenacházely žádné jiné neglobální zóny. V tomto stavu by měl uživatelský žurnál obsahovat čtyři zóny ve stavu *running*. Jak je vidět z výpisu programu B.2, součástí uživatelského žurnálu byly opravdu čtyři zóny ve stavu *running* a výpis neobsahoval žádné jiné nesledované neglobální zóny. Toto zjištění indikovalo, že nástroj opravdu aktualizuje uživatelský nástroj po provedení akcí.

Následně byla simulována situace, kdy jiný uživatel změní konkrétním způsobem stav sledované zóny. Konkrétně byla bez pomoci implementovaného nástroje přeinstalována zóna *zdev* na vzdáleném serveru *shost2* a její stav byl změněn z původního *running* na *installed*. Dále byla vytvořena nová zóna *zdev-clone* na stejném vzdáleném počítači. V tomto případě by měl uživatel při dalším vypsání uživatelského žurnálu zjistit, že se změnil stav a diskový obraz dané zóny *zdev*. Součástí výpisu by měla být i informace o nově vytvořené zóny v rámci infrastruktury. Z ukázky výpisu programu B.3 je vidět, že uživatelský žurnál opravdu informuje uživatele o změně sledované zóny a na konci výpisu je zobrazena informace o nově vytvořené zóně. Ostatní sledované zóny byly z výpisu vynechány.

Pomocí implementovaného nástroje může uživatel neglobální zóny vytvářet, mazat nebo měnit jejich stav. Jak bylo zjištěno na začátku této kapitoly, nástroj aktualizuje konkrétní záznam v uživatelském žurnálu, pokud danou zónu vytváří. Podobným způsobem bylo ověřeno, že uživatelský žurnál je aktualizován i při mazání a změně stavu zóny. Dále tento scénář ověřil funkcionalitu žurnálu, která má informovat uživatele v případě, kdy dojde ke změně stavu sledované zóny nebo vytvoření nové zóny v rámci infrastruktury. Z výše uvedených důvodů bylo testování využití uživatelského žurnálu označeno za úspěšné.

6.2.3 Migrace neglobálních zón

V rámci testování nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones byl otestován scénář, který zahrnuje migraci neglobálních zón mezi dvěma hosty. Jelikož nástroj umožňuje migraci zón jak z lokálního tak ze vzdálených serverů, bylo nutné vybrat komplexní scénář pokrývající tyto možnosti. Pro komplexní ověření této funkcionality nástroje bylo na hostech *shost* a *shost1* vytvořeno několik neglobálních zón, které byly migrovány na cílový vzdálený server *shost2*. Pro účely tohoto scénáře byla využita technika přímého přenosu souborového systému, kterou implementovaný ná-

stroj poskytuje.

Cílem tohoto scénáře je přesunutí neglobálních zón ze zdrojových serverů na cílový server. Na obou zdrojových serverech se na začátku testování nacházely následující neglobální zóny:

- *zmigr:localhost*,
- *zmigr1:localhost*,
- *zmigr2:shost1*,
- *zmigr3:shost1*.

Po provedení migrace by se tyto zóny měly nacházet na cílovém serveru *shost2* a na zdrojových serverech by se neměly nacházet žádné neglobální zóny. Uživatelský žurnál zobrazený ve výpisu programu B.5 potvrzoval stav před provedením migrace.

Pro migraci neglobálních zón slouží příkaz `migrate`, který je možné využívat skrze uživatelské rozhraní implementovaného nástroje. Právě tento příkaz byl použit pro testování tohoto scénáře a ve výpisu programu B.6 je možné pozorovat jeho výstup. Jak výstupu patrné, všechny zóny byly podle nástroje úspěšně přesunuty na cílový server *shost2*. Pro ověření funkcionality bylo nutné ověřit, zda se zóny opravdu nachází na cílovém serveru, zda jsou zóny smazány ze zdrojových serverů a také zda nástroj aktualizoval uživatelský žurnál.

Pomocí nástroje `zoneadm(1)` a příkazu `list` bylo ověřeno, že na vzdáleném serveru se opravdu nachází přesunuté zóny. Na zdrojových serverech byl spuštěn stejný příkaz pro ověření, že se na nich nenachází žádné neglobální zóny. Výstup těchto kroků je zobrazen ve výpisu programu B.7. Těmito kroky bylo ověřeno, že nástroj správně pracuje se zónami a umožňuje jejich přesun v rámci jednotlivých serverů infrastruktury. Dále bylo nutné ověřit, zda nástroj správně aktualizuje konkrétní záznamy v uživatelském žurnálu. Stav ve výpisu programu B.5 by se měl změnit tak, že přesouvané zóny budou zobrazeny pod hostem *shost2*. Tato skutečnost je potvrzena výpisem programu B.8, který ukazuje očekávaný výstup uživatelského žurnálu.

Výše popsaný testovací scénář dokazuje správné chování implementovaného nástroje v případě, kdy uživatel využívá administračních funkcí pro migraci zón. Test potvrzuje, že uživatel je schopný přesouvat neglobální zóny z lokálního i ze vzdáleného serveru na vzdálený cílový server. Migrace je provedena najednou a nástroj po jejím úspěšném provedení adekvátně aktualizuje uživatelský žurnál. Z výše popsaných důvodů bylo testování migračního scénáře označeno za úspěšné.

6.2.4 Záloha a obnova zón

Dalším testovaným scénářem bylo vytvoření zálohy a následná obnova. Jelikož se jedná o dvě samostatné funkce nástroje, byl tento test rozdělený do dvou

samostatných scénářů. V prvním scénáři použití bylo otestováno vytvoření zálohy několika zón. Následující scénář potom využil vytvořené zálohy k obnovení stejných zón.

6.2.4.1 Záloha neglobálních zón

Cílem tohoto scénáře bylo ověření funkcionality nástroje ve vytváření záloh neglobálních zón. Při procesu vytváření zálohy má nástroj za úkol vytvořit archiv souborového systému zálohované zóny. Nástroj umožňuje vytvoření zálohy pomocí dvou typů archivů. V případě tohoto scénáře byl použit archiv typu UAR. Dále nástroj umožňuje specifikovat vzdálený server a cestu, kam má být záloha uložena. Pomocí tohoto scénáře bylo ověřeno, zda nástroj tuto funkcionalitu opravdu umožňuje. Pro kompletní ověření funkcionality bude sloužit následující scénář, který z vytvořených záloh bude zóny obnovovat. Pro účely tohoto scénáře byly v infrastruktuře vytvořeny následující zóny:

- *zback:shost2*,
- *zback1:shost2*,
- *zback2:shost1*,
- *zback3:shost1*.

Na lokálním serveru *shost* byl vytvořen zálohovací adresář */zonepool/backup*, ve kterém se před zahájením testu nenacházely žádné soubory.

Nástroj pro zálohování zón nabízí příkaz `backup`, který je možný využít v uživatelském rozhraní. Pomocí tohoto příkazu byla spuštěna záloha výše zmíněných zón. Výstup tohoto příkazu je zobrazen ve výpisu programu B.9 a ukazuje, že záloha zón proběhla úspěšně. Zálohy by se podle výstupu programu měly nacházet ve složce */zonepool/backup* na lokálním serveru.

Pomocí standardního nástroje `ls(1)` bylo ověřeno, že se zálohy opravdu vytvořily a že se nacházely v určeném adresáři na lokálním serveru. Tento testovací scénář potvrzuje, že nástroj je schopný najednou vytvářet zálohy více neglobálních zón různě umístěných v infrastruktuře serverů. Dále potvrzuje, že nástroj umí stáhnout zálohy do konkrétního adresáře na předem určeném serveru. Z výše zmíněných důvodů bylo testování vytváření zálohy neglobálních zón úspěšné.

6.2.4.2 Obnova neglobálních zón

Posledním testovaným scénářem použití nástroje byla obnova neglobálních zón z archivu. Cílem testování tohoto scénáře bylo ověření, zda nástroj umí obnovit (vytvořit) neglobální zóny pomocí dříve vytvořených záloh. Jako zdrojové archivy byly použity zálohy vytvořené během předchozího testování. Pro simulaci ztráty dat byly všechny zóny z minulého testování odstraněny a zachovány

byly pouze jejich archivy ve složce */zonepool/backup* na lokálním serveru. Výsledkem obnovy by mělo být vytvoření všech zón ze zálohy a jejich umístění na původní servery.

Ve složce */zonepool/backup* se na začátku testování nacházely čtyři archivy typu UAR, které obsahovaly diskové obrazy a konfigurace jednotlivých zón. Nástroj pro podporu automatické správy Solaris Zones poskytuje funkcionalitu pro obnovu zón skrze příkaz `recovery`. Pomocí tohoto příkazu byla spuštěna obnova z výše zmíněných archivů. Tento proces je vyznačen ve výpisu programu B.10, kde je vidět i výstup tohoto příkazu. Z příkazu je patrné, že pořadí specifikace identifikátorů zón musí odpovídat pořadí zadaných archivů. Pokud by toto pořadí nesouhlasilo, došlo by k výměně diskových obrazů daných zón. Podle výstupu nástroje proběhla obnova zón v pořádku a zóny by měly být nainstalovány na své původní hosty.

Stejně jako v ostatních případech vytváření zón bylo pomocí nástroje `zoneadm(1)` a příkazu `list` ověřeno, že se obnovené zóny opravdu vytvořily na daných vzdálených serverech. Testování tohoto scénáře potvrdilo, že nástroj umí obnovit neglobální zóny ze sady záloh. Tyto zálohy musí mít definovaný tvar. V případě tohoto scénáře se jednalo o zálohy typu UAR, které byly vytvořeny v rámci předchozího testování. Z tohoto důvodu je možné označit testování scénáře pro obnovu zón za úspěšné.

6.3 Měření

V rámci testování nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones bylo provedeno měření doby běhu dvou hlavních administrátorských rutin. Konkrétně se jednalo o měření doby běhu vytváření neglobálních zón a jejich migrace v rámci popsané infrastruktury. V obou případech byla sledována doba běhu dané rutiny v závislosti na počtu vytvářených nebo přenášených zón. Tato měření byla prováděna pro několik technik vytváření a migrace neglobálních zón. Podrobně jsou obě měření popsána v následujících kapitolách. Jelikož vytváření a migrace většího počtu zón na jednom fyzickém stroji potřebuje větší výpočetní nároky, byly pro tato měření použity pouze virtuální stroje *shost* a *shost1*. Oběma strojům byly přiřazeny čtyři procesorová jádra. Kromě výše zmíněné změny bylo použito prostředí popsané v kapitole 6.1.

6.3.1 Techniky vytváření zón

Prvním objektem měření bylo sledování doby běhu nástroje při vytváření zón v závislosti na jejich počtu a technice vytváření. Cílovým serverem instalace byl lokální server *shost*. Pro toto měření byly použity následující techniky vytváření neglobálních zón:

- klonování,

Tabulka 6.1: Doba běhu vytváření zón v závislosti na jejich počtu a použité technice

Počet zón	1	2	4	8	10
Klonování [s]	19.5	20.4	28.8	51.4	74.6
Šablona [s]	xxx	xxxx	xxxx	xxxx	xxxx
Vzdálená zóna [s]	136.5	189.7	320.1	668.6	819.5

- instalace ze šablony,
- instalace ze vzdálené zóny.

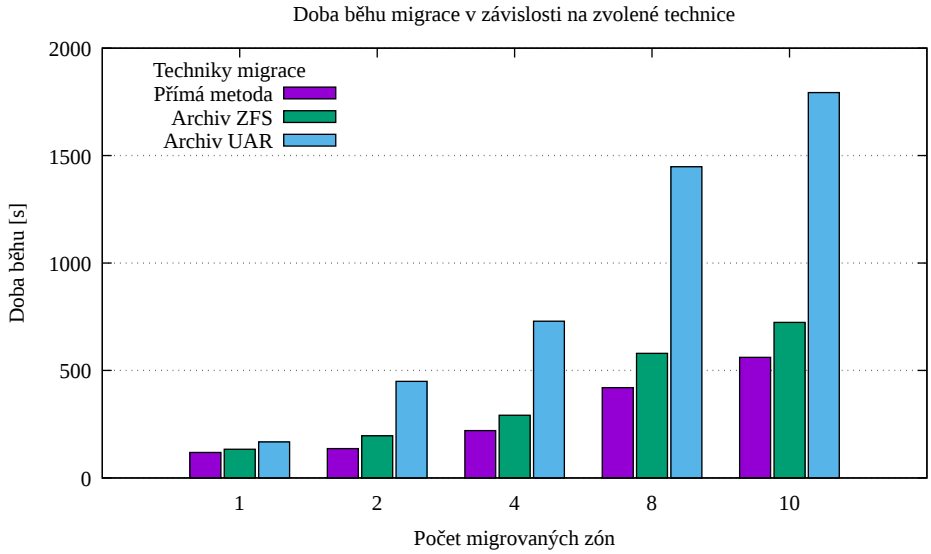
Pro účely klonování byla na lokálním serveru *shost* vytvořena instance šablony *template_z1*, která sloužila jako předloha pro klonování. Zdrojový souborový systém této zóny se v průběhu měření opakovaně klonoval a zkoumala se doba běhu v závislosti na počtu vytvářených klonů. V případě druhé techniky byla vytvořena šablona *zweb*, která specifikuje neglobální zónu. Její parametry nejsou pro měření podstatné. Tento typ instalace využívá online repozitář [21]. Z tohoto důvodu je výsledná doba instalace ovlivněná kvalitou internetového připojení. Pro účely posledního typu instalace byla na vzdáleném serveru *shost1* vytvořena neglobální zóna *z1*, která sloužila jako předloha pro instalaci zón. Ve výsledné době běhu je tedy zahrnutý i přenos souborového systému ze vzdáleného serveru na lokální server.

Jak je patrné z tabulce 6.3.1, měření bylo prováděno pro různé počty vytvářených zón. Maximální hodnota byla stanovena na deset zón, protože pro větší počet zón by se výrazně projevovala omezení testovacího prostředí. Pro každou kombinaci parametrů bylo měření několikrát opakováno a výsledek zprůměrován. Přesné hodnoty naměřených časů je možné pozorovat v tabulce 6.3.1. Výsledky jsou také graficky znázorněny v grafu 6.3.1.

Z provedeného měření je možné vypozorovat, že technika klonování zón je jednoznačně nejrychlejším způsobem vytváření zón. Časy pro ostatní techniky nejsou tak malé, protože tyto způsoby instalace zahrnují vytváření archivů a přenosy velkých objemů dat po síti. Velikost jedné neglobální zóny v průběhu testování byla přibližně 1,5GB. Pokud se má po síti přenášet několik takových zón najednou, je nutné přenést opravdu velké množství dat. Paralelní instalace zón pomocí těchto technik klade velké nároky na propustnost sítě i pevných disků. Z výše uvedeného měření plyne, že technika klonování je v rámci měřených způsobů nejrychlejší. Tato technika nešetří pouze čas instalace, ale také diskové místo.

6.3.2 Techniky migrace zón

Druhé měření se zabývá různými technikami způsobu migrace zón a měří dobu jejich trvání v závislosti na počtu migrovaných zón. Implementovaný nástroj umožňuje přenášet zóny mezi servery pomocí následujících technik:



Obrázek 6.1: Doba běhu vytváření zón v závislosti na jejich počtu a použité technice

- přímá migrace,
- migrace pomocí ZFS archivu,
- migrace pomocí UAR archivu.

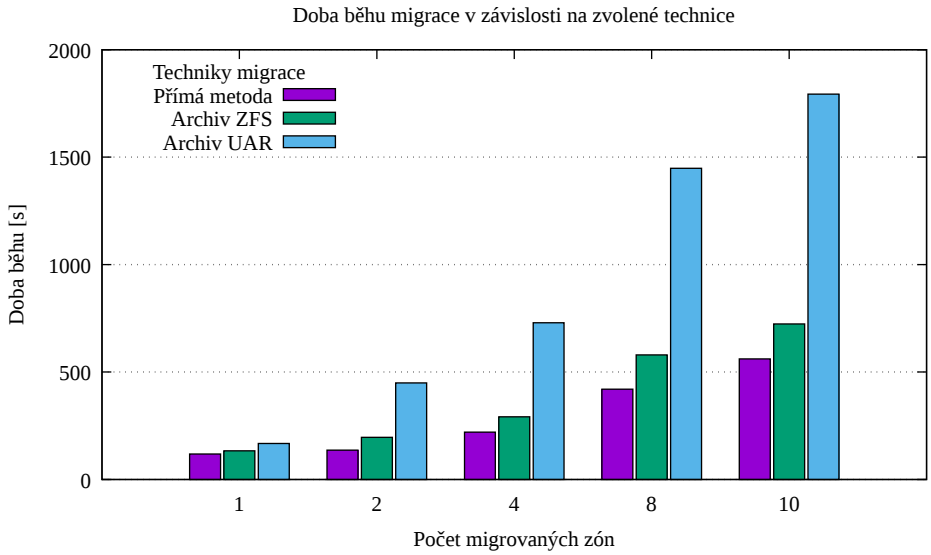
První technika využívá k migraci přímo příkazy souborového systému ZFS a umožňuje přenášet souborový systém bez nutnosti jeho dočasného uložení. Migrace pomocí ZFS archivu je podobná přímé technice, ale v průběhu migrace se zdrojový souborový systém uloží do archivu, který je následně přenesen na cílový server. V případě poslední techniky je použitý archiv typu UAR, který je standardním způsobem zálohování v operačním systému Solaris. I tato technika vyžaduje dočasné uložení vytvořeného archivu. Provedené měření bylo zaměřeno na porovnání doby trvání migrace v závislosti na počtu migrovaných zón a zvolené technice migrace.

Pro účel tohoto měření byly na lokálním serveru *shost* vytvořeny neglobální zóny, které byly vždy přesouvány na vzdálený server *shost1*. Migrace byla několikrát opakována pro každou zmíněnou techniku a počet migrovaných zón. Výsledná doba běhu byla v rámci opakování se stejnými hodnotami parametrů zprůměrována. Stejně jako v minulém měření bylo přenášeno maximálně deset neglobálních zón najednou. Naměřené hodnoty doby běhu pro různé počty migrovaných zón a různé techniky migrace je možné pozorovat v tabulce 6.3.2. Graficky jsou tyto hodnoty znázorněny v grafu 6.3.2.

Z měření je možné vypozorovat, že přímá technika migrace přenáší zóny nejrychleji. Tento výsledek se byl očekáván, jelikož tato technika nevyžaduje

Tabulka 6.2: Doba běhu migrace zón v závislosti na použité metodě

Počet zón	1	2	4	8	10
Přímá technika [s]	118.3	134.6	220.2	420.1	561.2
Technika ZFS [s]	133.5	196.1	291.5	580	723.7
Technika UAR [s]	167.8	449.7	729.5	1448.3	1793.6



Obrázek 6.2: Doba běhu migrace zón v závislosti na použité metodě

dočasné ukládání přenášeného souborového systému. Technika využívající ZFS archivy je pomalejší. Důvodem je hlavně nutnost dočasného uložení vytvořeného archivu na vzdáleném serveru. Výše zmíněné techniky jsou vykonávány paralelně pro všechny přenášené zóny. V případě techniky využívající archivy typu UAR tomu tak není. Jak je patrné z tabulky 6.3.2, naměřené časy této techniky odpovídají sériovému provádění migrací. Tento fakt je způsobem nástrojem `archiveadm(1)`, který nemůže být současně používán více procesy v rámci jednoho systému. Z měření vyplynulo, že nejrychlejší technikou pro migraci zón je přímá technika.

6.4 Závěr testování

V rámci testování nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones byly testovány hlavní scénáře použití a změřena doba běhu některých funkcí nástroje. Popsaná měření dokazují, že implementovaný nástroj je schopen provádět administrátorské rutiny pro větší množství neglobálních zón. S rostoucím počtem těchto zón však roste i celková doba běhu

programu. Z měření je také patrné, že některé funkce nástroje jsou v určitých situacích výhodnější než jiné.

Všechny testované scénáře byly prováděny v rámci definovaného prostředí. Vybrané scénáře použití by měly pokrývat hlavní funkcionalitu nástroje a jejich splnění by mělo vypovídat funkčnosti celého nástroje. Jelikož testování těchto scénářů proběhlo bez chyb a s očekávanými výsledky, je možné konstatovat, že implementovaný nástroj splňuje všechny stanovené požadavky a obsahuje požadovanou funkcionalitu.

Závěr

Virtualizace se stala běžnou a možná i nezbytnou součástí dnešního počítačového světa. Teto technika se využívá v mnoha oblastech informačních technologií, kde přináší různé benefity. Virtualizace umožňuje společnostem efektivně využívat dostupné fyzické prostředky a tím výrazně ušetřit náklady na provoz fyzických zařízení. Tématem této diplomové práce byla virtualizace serverů, která umožňuje současný běh několika virtuálních počítačů v rámci jednoho fyzického systému.

Jednou z technik virtualizace serverů je technologie Solaris Zones, která je součástí operačního systému Solaris vyvíjeného firmou Oracle. Tato virtualizační technika umožňuje běh mnoha virtualizačních kontejnerů v rámci jedné instance operačního systému Solaris. Tyto kontejnery se nazývají zóny a jsou izolovány na úrovni počítačové sítě, souborového systému a spuštěných procesů. Standardně hlavní operační systém sdílí svoje jádro s ostatními zónami, ale tato technika umožňuje vytvářet i zóny s vlastním jádrem.

Cílem teoretické části této diplomové práce bylo tuto virtualizační techniku popsat a porovnat s ostatními běžně využívanými technikami virtualizace. Dále se práce zaměřila na popis konfigurace, instalace a základních administrátorských rutin pro správu zón. Předmětem byly také pokročilé rutiny vyžadují provedení několika kroků, jejichž postupným vykonáváním lze dosáhnout požadovaného výsledku. Z tohoto důvodu bylo hlavním cílem práce vytvořit nástroj, který by tyto procesy automatizoval a umožnil je provádět i pro vzdálené zóny. Hlavním důvodem pro vytvoření tohoto nástroje je fakt, že standardní nástroje pro správu Solaris Zones tuto funkcionalitu neposkytují. Jelikož Solaris Zones poskytují uživateli pouze rozhraní na příkazové řádce, bylo nutné implementovaný nástroj postavit právě nad tímto rozhraním a využívat tak standardní nástroje pro správu zón.

Výsledná implementace nástroje se skládá z několika částí. Jednou z nich je modul Solaris Zones, který poskytuje funkcionalitu ostatním částem nástroje. Jeho architektura se skládá z několika hierarchicky uspořádaných vrstev. Nejníže v hierarchii se nachází vrstva, která umožňuje lokální i vzdálené

spouštění nástrojů *zonecfg*, *zoneadm*, *archiveadm* a v neposlední řadě také *zfs*. Ostatní vrstvy modulu využívají těchto příkazů a vytvářejí z nich sekvence, které reprezentují jednotlivé administrátorské rutiny pro vytváření, mazání, zálohu, obnovu a migraci zón. Jednotlivé rutiny jsou implementované jako transakce. V rámci transakce je zajištěno, že všechny dočasně vytvořené soubory budou smazány a že všechny provedené změny budou navráceny do původního stavu v případě neúspěchu transakce. Takto implementované rutiny jsou poskytovány ostatním částem nástroje pomocí předem definovaného rozhraní.

Jelikož v dnešní době existuje mnoho virtualizačních technik s podobnými vlastnostmi jako Solaris Zones, byla architektura nástroje navržena tak, aby se dala v budoucnu lehce rozšířit o další moduly. Tuto funkcionalitu zajišťuje v rámci nástroje knihovna, která jasně definuje rozhraní jednotlivých modulů. Pomocí tohoto rozhraní knihovna zprostředkovává funkcionalitu modulů klientským aplikacím. Ve výsledném nástroji je implementován pouze výše zmíněný modul pro podporu správy Solaris Zones. Tato knihovna také definuje generickou šablonu, která slouží pro popis konkrétních virtuálních strojů. Výše popsaný modul implementuje rozšíření této generické šablony, které umožňuje uživateli specifikovat konfiguraci, softwarové vybavení a systémové nastavení zón. Tyto šablony je možné využít pro vytváření většího množství zón s danými vlastnostmi.

Ovládání nástroje je zajištěno pomocí uživatelského rozhraní, které je uživateli prezentováno pomocí příkazové řádky. Jednotlivé příkazy umožňují vyvolat odpovídající funkce modulu pro vytváření, mazání, zálohu, obnovu nebo migraci zón. Uživatel může jednoduše specifikovat pro jaké zóny chce danou akci provést. Tyto zóny se mohou nacházet na lokálním i vzdáleném serveru a nástroj pro každou specifikovanou zónu vykoná konkrétní akci pokud možno paralelně. Uživatelské rozhraní dále implementuje systém pro správu vzdálených hostů, který umožňuje dané hosty registrovat a specifikovat parametry připojení. Hromadné akce poskytované uživatelským rozhraním jsou prováděny právě pro všechny registrované hosty. Jelikož se zónami může manipulovat každý privilegovaný uživatel, implementuje tato část nástroje také uživatelský žurnál. V žurnálu jsou udržovány jednotlivé stavy zón, aby bylo možné poznat, zda v době nepřítomnosti uživatele nedošlo ke změně stavu zón. Další součástí uživatelského rozhraní je grafický editor šablon, který uživateli poskytuje možnost interaktivní tvorby a editace šablon. Hlavním smyslem tohoto editoru je odstínění uživatele od implementačních detailů šablon. Tohoto grafického rozhraní je využito i v případě interaktivní instalace zón, která uživateli nabízí možnost specifikace vlastností v průběhu vytváření zón.

Implementovaný nástroj umožňuje uživateli jednoduchou správu většího množství zón pomocí automatizovaných rutin pro vytváření, mazání, zálohu, obnovu a migraci těchto zón. Součástí nástroje jsou šablony, které umožňují přesně specifikovat vlastnosti vytvářených zón. Tato funkcionalita se dá dobře využít v procesu vývoje software pro definici produkčních, testovacích a vývo-

jářských prostředí. Implementované uživatelské rozhraní prezentuje uživateli funkce nástroje v jednoduché a přehledné formě. Grafické součásti nástroje slouží především pro komfortní vytváření a editaci šablon.

Implementovaný nástroj pro podporu automatické správy virtualizačního kontejneru Solaris Zones poskytuje očekávanou funkcionalitu a splňuje stanovené požadavky. Tato skutečnost byla ověřena v rámci testování tohoto nástroje, které je součástí této diplomové práce. Z těchto důvodů je možné konstatovat, že všechny stanovené cíle této diplomové práce byly naplněny. Instalační balík a zdrojové kódy implementovaného nástroje i zdrojové kódy celé diplomové práce jsou dostupné na přiloženém médiu.

Literatura

[1] University, O.: *Definition of virtual in English* [online]. [cit. 2018-03-26]. Dostupné z: https://en.oxforddictionaries.com/definition/virtual

[2] VMware: *The Virtues of Network Virtualization* [online]. Publikováno 4.11.2015, [cit. 2018-03-26]. Dostupné z: https://www.vmware.com/ciovantage/article/the-virtues-of-network-virtualization

[3] Smith, J. E.; Nair, R.: The architecture of virtual machines. *Computer*, ročník 38, č. 5, Květen 2005: s. 32–38, ISSN 0018-9162, doi: 10.1109/MC.2005.173.

[4] Armstrong, E.: *HotSpot: A new breed of virtual machine* [online]. Publikováno 1.3.1998, [cit. 2018-04-30]. Dostupné z: https://www.javaworld.com/article/2076604/hotspot--a-new-breed-of-virtual-machine.html

[5] Oracle: *Reasons to Use Virtualization* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.oracle.com/cd/E27300_01/E27309/html/vmusg-virtualization-reasons.html

[6] Kašpar, J.; Tvrdík, P.: *Techniky virtualizace II.* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2018-03-12]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-POA/_media/lectures/mi-poa09.pdf

[7] Kašpar, J.; Tvrdík, P.: *Techniky virtualizace I.* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2018-03-12]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-POA/_media/lectures/mi-poa08.pdf

[8] Lynn, S.: *Oracle Solaris 11.4 Open Beta Released!* [online]. Publikováno 30.1.2018, [cit. 2018-05-01]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54760/solosvirt.html

[9] Muzikář, Z.; Ždárek, J.: *Start systému, proces init, Solaris Service Management Facility* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2018-03-23]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-ADU/_media/lectures/07/biadu_p07_start.pdf

[10] Šimáček, T.: *Nástroj pro správu a monitorování systému souborů ZettaByte*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií.

[11] Oracle: *Introduction to Oracle® Solaris 11 Virtual Environment* [online]. [cit. 2018-03-23]. Dostupné z: https://docs.oracle.com/cd/E53394_01/pdf/E54760.pdf

[12] Oracle: *Oracle Solaris 11.3 Virtualization Technologies* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54760/solosvirt.html

[13] Oracle: *Introduction to Oracle® Solaris 11. Zone Brand Overview* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54762/gitrc.html

[14] Oracle: *Hardware and Software Requirements for Oracle Solaris Kernel Zones* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54751/gnwoi.html

[15] Oracle: *Zone Administration Overview* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54762/gqhar.html

[16] Oracle: *Man pages section 1M: System Administration Commands. zonecfg(1M)* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/cd/E23824_01/html/E21798/gklep.html

[17] Oracle: *Man pages section 7: Standards, Environments, Macros, Character Sets and Miscellany. zones_solaris-kz(7)* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E88353_01/html/E37853/zones-solaris-kz-7.html

[18] Oracle: *How to Modify a Resource Type in a Zone Configuration* [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54752/z.conf.start-63.html

[19] Oracle: *Creating and Using Oracle® Solaris Zones How Zones Are Installed.* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E88353_01/html/E37853/zones-solaris-kz-7.html

[20] Oracle: *Man pages section 1M: System Administration Commands. zoneadm(1M)* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E88353_01/html/E37853/zones-solaris-kz-7.html

[21] Oracle: *Oracle Solaris 11 Package repository* [online]. [cit. 2018-04-26]. Dostupné z: http://pkg.oracle.com/solaris/release/en/index.shtml

[22] Hardie, D.: *How to Get Started Creating Oracle Solaris Kernel Zones in Oracle Solaris 11* [online]. [cit. 2018-04-25]. Dostupné z: http://www.oracle.com/technetwork/articles/servers-storage-admin/howto-create-kernal-zones-s11-2251331.html

[23] Oracle: *About Adding Packages in Systems With Zones Installed* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54752/z.pkginst.ov-14.html

[24] Oracle: *About Cloning Non-Global Zones* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54752/gcrsy.html

[25] Oracle: *Specifying Configuration in a System Configuration Profile* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E23824_01/html/E21798/gklea.html

[26] Oracle: *Man pages section 1M: System Administration Commands. sysconfig(1M)* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E36784_01/html/E36871/sysconfig-1m.html

[27] Oracle: *Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.2* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E36784_01/html/E38524/gmrlo.html

[28] Oracle: *Migrating a Non-Global Zone to a Different System* [online]. [cit. 2018-04-25]. Dostupné z: https://docs.oracle.com/cd/E53394_01/html/E54752/gpolc.html

[29] Oracle: *Oracle Solaris Administration: Security Services. Role-Based Access Control* [online]. [cit. 2018-04-26]. Dostupné z: https://docs.oracle.com/cd/E23824_01/html/821-1456/rbac-1.html

[30] Zahradnický, T.: *Security and Secure programming* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2018-04-26]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-BPR/_media/lectures/mi-bpr-2014-lecture-1.pdf

[31] Ruby, C.: *Ruby programming language* [online]. [cit. 2018-04-27]. Dostupné z: https://www.ruby-lang.org/en/

[32] Barton, T.; Vlnas, J.; Szolár, T.; aj.: *Úvod do jazyka Ruby* [online]. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2018-04-27]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-RUB/_media/lectures/01/01a.pdf

[33] Koichi, S.: *YARV: Yet Another Ruby VM* [online]. [cit. 2018-04-27]. Dostupné z: http://www.atdot.net/yarv/

[34] JRuby: *The Ruby Programming Language on the JVM* [online]. [cit. 2018-04-27]. Dostupné z: http://jruby.org/

[35] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, Březen 2014, [cit. 2018-04-27]. Dostupné z: http://www.rfc-editor.org/rfc/rfc7159.txt

[36] Wright, A.; Andrews, H.: *JSON Schema: A Media Type for Describing JSON Documents* [online]. Publikováno 19.3.2018, [cit. 2018-04-27]. Dostupné z: http://json-schema.org/latest/json-schema-core.html

[37] Hoxworth, K.; Beeston, I.; Hargraves, K.: *Ruby JSON Schema Validator* [online]. [cit. 2018-04-27]. Dostupné z: https://github.com/ruby-json-schema/json-schema

Příloha A

Seznam použitých zkratek

BIOS Basic Input Output System

CLI Command Line Interface

DNS Domain Name System

EPT Extended Page Tables

HLL High Level Langugage

HW Hardware

I/O Input/Output

ISA Instruction Set Architecture

IT Information Technology

JSON JavaScript Object Notation

NAT Network Address Transaltion

OS Operating System

PC Program Counter/Personal Computer

RVI Rapid Virtualization Indexing

SW Software

SMF Solaris Management Facility

SSH Secure Shell

TLB Translation Lookaside Buffer

YARV Yet Another Ruby VM

VM Virtual Machine

VMM Virtual Machine Monitor

XML Extesible Markup Language

ZFS Zettabyte File System

Příloha B

Testování

Výpis kódu B.1: Výstup příkazu pro vytvoření neglobálních zón ze šablony

```
zadmin@shost:~$ szmgmt_cli deploy -b zdev zdev \
                                       zdev:shost1 \
                                       zdev:shost2 \
                                       -s ~/zdev.json
Solaris zones deployment initialized.
  ------------------------------------------------------------
  Options:
      Boot zones: enable
   Rewrite zones: disable
         Source: </export/home/zadmin/zdev.json>
  ------------------------------------------------------------
  Loading virtual machine specification.
  Virtual machine specification loaded.
  ------------------------------------------------------------
  Connecting concurrently to hosts 'localhost,shost1,shost2'.
  Processing zone 'zdev' deployment on host 'localhost'.
  Processing zone 'zdev1' deployment on host 'localhost'.
  Processing zone 'zdev' deployment on host 'shost1'.
  Processing zone 'zdev' deployment on host 'shost2'.
  ------------------------------------------------------------
  Deployment finished.
    Status:
      localhost:
        zdev:  success
        zdev1: success
      shost1:
        zdev:  success
      shost2:
        zdev:  success
```

Výpis kódu B.2: Výpis uživatelského žurnálu po vytvoření zón

```
zadmin@shost:~$ szmgm_cli journal status
Tracked zones:
  Host localhost
      zdev1:localhost
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zdev1
      zdev:localhost
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zdev
  Host shost2
      zdev:shost2
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zdev
  Host shost1
      zdev:shost1
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zdev
```

Výpis kódu B.3: Výpis uživatelského žurnálu po změně původní zóny

```
zadmin@shost:~$ szmgm_cli journal status
Tracked zones:
  ...
  Host shost2
      zweb:shost2
           Zone type: solaris
          Zone state: running
                      MISMATCH - Current state installed.
           Zone path: /system/zones/zweb
                      MISMATCH - UUID mismatch.
Untracked zones:
  Host shost2
      zdev-colne:shost2
          Zone type: solaris
         Zone state: installed
          Zone path: /system/zones/zdev-colne
```

Výpis kódu B.4: Sekvence příkazů pro ověření správnosti vytvoření zóny

Výpis kódu B.5: Výpis uživatelského žurnálu před migrací zón

```
zadmin@shost:~$ szmgmt_cli journal status
Geting fresh information about zones on all registered hosts...
Tracked zones:
  Host localhost
      zmigr1:localhost
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zmigr1
      zmigr:localhost
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zmigr
  Host shost1
      zmigr3:shost1
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zmigr3
      zmigr2:shost1
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zmigr2
```

Výpis kódu B.6: Výpis příkazu pro migraci neglobálních zón

```
zadmin@shost:~$ szmgm_cli migrate -t d zmigr \
                                      zmigr1 \
                                      zmigr1:shost1 \
                                      zmigr1:shost1 \
                                      -d shost2
Solaris zones migration initialized.
  ------------------------------------------------------------
  Options:
                Boot zones: disable
    Rewrite existing zones: disable
  ------------------------------------------------------------
  Connecting concurrently to hosts 'localhost shost1'.
  Processing migration of zone 'zmigr1:localhost'.
    See log '~/.szmgmt/log/zmigr1_migration_ph5o7o.log'.
  Processing migration of zone 'zmigr2:shost1'.
    See log '~/.szmgmt/log/zmigr2_migration_uy1frm.log'.
  Processing migration of zone 'zmigr3:shost1'.
    See log '~/.szmgmt/log/zmigr3_migration_lhc24y.log'.
  Processing migration of zone 'zmigr:localhost'.
    See log '~/.szmgmt/log/zmigr_migration_h2rdat.log'.
  ------------------------------------------------------------
  Migration finished.
    Status:
      localhost:
        zmigr:  success
        zmigr1: success
      shost1:
        zmigr2: success
        zmigr3: success
```

Výpis kódu B.7: Výpis zón na jednotlivých serverech po migraci

```
zadmin@shost:~$ zoneadm list -vic
  ID NAME     STATUS            BRAND      IP
   0 global   running           solaris    shared

zadmin@shost1:~$ zoneadm list -vic
  ID NAME     STATUS            BRAND      IP
   0 global   running           solaris    shared

zadmin@shost2:~$ zoneadm list -vic
  ID NAME     STATUS            BRAND      IP
   0 global   running           solaris    shared
   - zmigr1   installed         solaris    excl
   - zmigr2   installed         solaris    excl
   - zmigr3   installed         solaris    excl
   - zmigr    installed         solaris    excl
```

Výpis kódu B.8: Výpis uživatelského žurnálu po migraci zón

```
zadmin@shost:~$ szmgmt_cli journal status
Geting fresh information about zones on all registered hosts...
Tracked zones:
  Host shost2
      zmigr1:shost2
           Zone type: solaris
          Zone state: installed
           Zone path: /system/zones/zmigr1
      zmigr2:shost2
           Zone type: solaris
          Zone state: installed
           Zone path: /system/zones/zmigr2
      zmigr:shost2
           Zone type: solaris
          Zone state: installed
           Zone path: /system/zones/zmigr
      zmigr3:shost1
           Zone type: solaris
          Zone state: running
           Zone path: /system/zones/zmigr3
```

Výpis kódu B.9: Výpis příkazu pro vytvoření zálohy zón pomocí archivu UAR

```
zadmin@shost:~$ szmgmt_cli backup zback:shost2 \
                                  zback1:shost2 \
                                  zback2:shost1 \
                                  zback3:shost1
                -d shost -p /zonepool/backup -t uar
Solaris zones backup initialized (UAR).
  ------------------------------------------------------------
  Options:
          Backup directory: /zonepool/backup
          Destination host: shost
  ------------------------------------------------------------
  Connecting concurrently to hosts 'shost2, shost1'.
  Processing zone backup of 'zback2:shost1'.
    See log '~/.szmgmt/log/zback2_backup_syikav.log'.
  Processing zone backup of 'zback:shost2'.
    See log '~/.szmgmt/log/zback_backup_71blh4.log'.
  Processing zone backup of 'zback1:shost2'.
    See log '~/.szmgmt/log/zback1_backup_5rks8b.log'.
  Processing zone backup of 'zback3:shost1'.
    See log '~/.szmgmt/log/zback3_backup_shmtgw.log'.
  ------------------------------------------------------------
  Backup finished.
    Status:
      shost2:
        zback: success
        zback1: success
      shost1:
        zback2: success
        zback3: success
```

Výpis kódu B.10: Výpis příkazu pro obnovení zón ze zálohy typu UAR

```
zadmin@shost:~$ szmgmt_cli recover zback:shost2 \
                                   zback1:shost2 \
                                   zback2:shost1 \
                                   zback3:shost1
                -a \
                /zonepool/backup/zback_backup_1525020859.uar \
                /zonepool/backup/zback1_backup_1525020859.uar \
                /zonepool/backup/zback2_backup_1525020859.uar \
                /zonepool/backup/zback3_backup_1525020859.uar
Solaris zones recovery initialized.
  ------------------------------------------------------------
  Options:
                 Boot zones: disable
    Rewrite existing zones: disable
  ------------------------------------------------------------
  Connecting concurrently to hosts 'shost2, shost1' .
  Processing zone 'zback1' recovery on host 'shost2'.
    Aarchive: /zonepool/backup/zback1_backup_1525020859.uar.
    See log '~/.szmgmt/log/zback1_recovery_vai7x.log'.
  Processing zone 'zback2' recovery on host 'shost1'.
    Archive: /zonepool/backup/zback2_backup_1525020859.uar.
    See log '~/.szmgmt/log/zback2_recovery_lpl0c1.log'.
  Processing zone 'zback' recovery on host 'shost2'.
    Archive: /zonepool/backup/zback_backup_1525020859.uar.
    See log '~/.szmgmt/log/zback_recovery_7ecl60.log'.
  Processing zone 'zback3' recovery on host 'shost1'.
    Archive: /zonepool/backup/zback3_backup_1525020859.uar.
    See log '~/.szmgmt/log/zback3_recovery_7ta1fr.log'.
tes  ------------------------------------------------------------
  Recovery finished.
    Status:
      shost2:
        zback: success
        zback1: success
      shost1:
        zback2: success
        zback3: success
```

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS