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.

29. dubna 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 29. dubna 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 ukázek kódů

Úvod

Virtualizace je technika, se kterou se dnes v IT můžeme setkat v mnoha podobách. Jednou z hlavních oblastí využití virtualizace je virtualizace serverů a mimo jiné se objevuje i v oblasti komunikačních sítí a desktopů. Tato technologie umožňuje vytvářet virtuální prostředí nebo prostředky na fyzickým hardware. Speciální softwarová vrstva zvaná virtualizační monitor (VMM) zajišťuje efektivní rozdělování prostředků fyzického systému mezi virtualizované subjekty.

Hlavním tématem této práce je virtualizace serverů, která umožňuje rozdělit jeden fyzický systém na několik nezávislých virtuálních prostředí zvaných virtuální počítač (VM). Možnost vytváření VM značně snižuje náklady na pořizování a provoz fyzických strojů, jelikož už není třeba dedikovaný server pro každou instanci OS. A konečně správným rozdělením VMs na fyzické servery můžeme docílit ideálního rozdělení zátěže a tím efektivně využít dostupné fyzické prostředky.

Rostoucí počet virtualizovaných serverů může mít za následek obtížnější správu. Automatizované nasazování, instalace nebo zálohování VMs může být značným ulehčením vývoje software, testování nebo nasazování aplikací do produkčního prostředí. Tomuto tématu se tato práce věnuje v souvislosti s virtualizačním kontejnerem Solaris Zones.

Cíle práce

Prvním cílem této práce je seznámení se s operačním systémem Solaris a jeho funkcemi. Především jde o popis virtualizační techniky Solaris Zones. Důraz je kladen na popis základních principů, které umožňují běh více zón v rámci jednoho sdíleného jádra OS.

Dalším cílem je detailní popis možností konfigurace zón, jejich instalace, zálohování a v neposlední řadě také integrace Solaris Zones s ostatními službami operačního systému Solaris.

Třetí cíl této práce je porovnat Solaris Zones s ostatními virtualizačními technologemi.

Posledním cílem této práce je implementace nástroje, který umí spravovat větší množství Solaris Zones. Tento nástroj bude umožňovat (dávkovou a interaktivní) instalaci zón na lokální i vzdálené servery, náhradu existujících zón a jejich zálohování. Dále bude umožňovat automatické přidání předem definovaných softwarových balíčků po instalaci zóny.

Struktura práce

TODO

Kapitola 1

Virtualizace

Jak je z názvu kapitoly patrné, hlavním obsahem následující části práce bude virtualizace a to především odvětví, které se věnuje výpočetní technice. Virtualizace je velice komplexní téma, a proto je nutné řádně specifikovat z jakého úhlu pohledu se na toto téma koukáme.

Po představení obecného konceptu virtualizace se proto podíváme na několik oblastí využité této technologie v informačních technologiích. Detailní popis všech oblastí virtualizace není předmětem této práce, a proto se ve zbytku diplomové práce budeme věnovat pouze tématu virtualizace serverů. I takto specifikované téma ale obsahuje mnoho virtualizačních principů a technik, které si u jednotlivých typů virtuálních strojů představíme. Jelikož virtualizace zažívá v dnešní době velký rozvoj, podíváme se také na jednotlivé scénáře nasazení serverů využívající virtualizaci. V poslední části této kapitoly jsou blíže představeny vybrané principy virtualizace a základy virtualizace CPU, paměti a I/O zařízení. TODO (uvidíme jestli bude třeba popisovat vše)

1.1 Obecná definice virtualizace

Než se pustíme do popisu jednotlivých typů virtualizace detailněji, je vhodné definovat tento pojem v obecném slova smyslu. Slovo virtuální je dle [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 tedy můžeme definovat jako vytváření virtuálních prostředků, které skrývají nebo upravují podstatu fyzických prostředků před uživatelem. Tento proces zahrnuje vytváření více virtuálních prostředků z jednoho fyzického. Jako příklad této virtualizace můžeme použít virtuální paměť, kdy se virtuální paměť více procesů mapuje do hlavní (fyzické) paměti počítače. Na druhou stranu 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 si zcela jistě objevuje i v jiných oblastech, ale nás bude zajímat, 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 ale také programovacích jazyků.

Pokud se podíváme na architekturu dnešních počítačových systémů z pohledu struktury a typů zařízení, které se v ní vyskytují, můžeme najít hned několik oblastí ve kterých se virtualizace využívá.

1.2.1 Virtualizace serverů

V dnešní době je pro většinu společností téměř nutností nějakým způsobem využívat výpočetních prostředků. Důvod pro jejich použití může být potřeba ukládání a zálohy obchodních záznamů, poskytování interních nebo externích služeb či běh 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 klasicky myslí operační systém. Druhy serverů můžeme rozdělit dle typu běžících uživatelských programů v operačním systému. Konkrétně pak hovoříme o aplikačních, souborových nebo výpočetních serverech, 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 toho virtuálního prostředí je zajištěno speciální softwarovou vrstvou zvanou virtualizační monitor, hraje roli řídícího programu a pracuje mezi HW a jednotlivými

instancemi OS. 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 OS a jejich uživatelské programy. Tyto servery se vyznačují především velkým množstvím operační paměti a vysokým výpočetním výkonem, který je díky VMM rozdělován mezi hostované operační systémy. Výhody nasazení virtuální infrastruktury jsou dále 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 of firmy Oracle, která slouží pro vyváření virtuálních strojů (zón), které sdílejí jedno jádro OS.

1.2.2 Využití virtualizace v sítích

Oblast komunikačních sítí je další neméně významnou oblastí pro využití virtualizačních technik. Bez síťové infrastruktury by mezi sebou počítače nemohli 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 a to i její topologie. To vše lze uskutečnit z jednoho místa a bez nutnosti zasahovat do fyzických zařízení sítě.

Virtualizace sítí je koncept, který se v mnoha ohledech podobná 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 a 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 zmínku. Pod pojmem desktop si představme 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 další podpůrných systémů jsou všechny informace o uživatelském nastavení uloženy na desktopu a uživatel se k nim dostane pouze z toho samého stroje. Virtualizací desktopu je rozuměno oddělení uživatelského prostředí a nastavení od fyzického stroje. Jednou z možností je přesunutí tohoto prostředí do virtuálního stroje, který je centrálně spravován a spouštěn, když uživatel potřebuje. Tento koncept umožňuje uživateli pří-

stup 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 části 1.1 jsme si definovali virtualizaci obecně jako virtualizaci fyzických (HW) prostředků. Virtualizace systému nebo komponenty jako je procesor, paměť nebo I/O zařízení na určité vrstvě architektury počítače znamená mapování jeho rozhraní na rozhraní nižší vrstvy způsobem, který může reprezentovat v jiném smyslu než fyzicky existuje.

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

Než budeme pokračovat s klasifikací virtuálních strojů, představíme si architekturu klasického počítačového systému.

1.3.1 Architektura počítačového systému

Jelikož implementace virtuální strojů operují na rozhraních jednotlivých vrstev architektury počítačového systému, je nutné se řádně s těmito vrstvami seznámit. Tyto vrstvy reprezentují několik úrovní abstrakce v počítačovém systému, které mají za úkol odstínit složité implementační detaily některých rozhraní. Čím výše se v této hierarchii nacházíme, tím abstraktnější a jednodušší funkce máme k dispozici.

Každá z vrstev má dobře definované rozhraní, což umožňuje vývoj vyšších vrstev nezávisle na implementaci nižších vrstev, pokud tyto vrstvy budou dodržovat toto rozhraní. Jako příklad si můžeme vzít výrobce procesorů Intel a AMD vyrábějící mikroprocesory, které implementují instrukční sadu IA-32 (x86) [3]. Zatímco nezávisle na vývoji těchto procesorů můžou 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

Obrázek 1.1: Architektura počítačového systému [3]

SPARC. Nicméně díky některým technikám virtualizace se tohoto dá dosáhnout.

Obrázek 1.2 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.

Instruction set architecture

Instrukční sada neboli ISA definuje rozhraní mezi HW a SW. Rozdělit jí můžeme na dvě části a to na systémovou a uživatelskou instrukční sadu. Rozhraní s číslem 4 na obrázku 1.2 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.

Application binary interface

Fyzické prostředky a zařízení dostupné v fyzickém systému spravuje operační systém, který k nim poskytuje přístup ostatním programům skrze svoje rozhraní. Toto rozhraní (číslo 2) se nazývá systémové a společně s uživatelskou částí (číslo 4) 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

Application programming interface

Důležitou vrstvou softwarového vybavení počítače jsou uživatelské knihovny. Tyto knihovny skrývají implementační detaily systémových volání a poskytují rozhraní (číslo 1) pro vyšší programovací jazyky jako je C nebo C++. Společně s uživatelskou částí instrukční sady (číslo 4) tvoří rozhraní nazývané *application programming interface* neboli API. Uživatelské programy pak mohou využívat tohoto rozhraní, což přináší výhody v přenositelnosti na systémy, které nabízejí stejné API .

1.4 Klasifikace virtuálních strojů

Abychom mohli rozlišit jednotlivé typy virtuálních strojů, musíme se podívat v jaké části počítačové architektury operují a tedy jakou vrstvu virtualizují. V části 1.3.1 jsme definovali tři dobře definované rozhraní počítačového systému a je logické, že virtualizační software bude virtualizovat nějakou z nich. Dle rozdělení v [3] můžeme virtuální stroje obecně rozdělit 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 systému. Systémový virtuální stroj tedy musí poskytovat operačnímu systému stejné rozhraní jako OS očekává. Nabízí tak operačnímu systému přístup k HW prostředkům fyzického stroje, které mohou být virtualizovány.

Na druhou stranu procesy využívají ke svému běhu mimo uživatelské části ISA také ABI nebo v případě vyšších programovacích jazyků API. Virtuální stroje, které se specializují na virtualizaci těchto dvou systémových rozhraní, budeme nazývat *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 podkapitolách jsou popsané jednotlivé typy virtuálních strojů. Tato klasifikace byla převzata z [3] a upravena podle požadavků práce.

Terminologie

Pro účely dalších částí diplomové práce si definujeme některé pojmy, které se v architektuře počítačového systému s virtuálním strojem vyskytují.

Po přidání virtualizačního SW mezi nějaké vrstvy počítačového systému nám vzniknou tři části. Tuto skutečnost popisuje obrázek 1.2, který zároveň ukazuje na jaké místo zaujímá virtualizační software v případě systémových virtuální strojů (b) a virtuálních strojů v procesech (a).

Softwarová 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.

Na druhou stranu vrstvy poskytující rozhraní virtualizačnímu SW neboli se nacházejí níže v hierarchii, nazýváme *host*.

Poslední vrstva, která zbývá popsat je samotný virtualizační SW. V případě systémových virtuálních strojů se typicky nazývá virtualizační monitor

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

neboli VMM. Pro virtualizační SW v druhém typu virtuálních strojů se používá název *runtime*.

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

1.4.1 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 na úrovni OS

Pro představení prvního zástupce z této rodiny virtuálních strojů nemusíme chodit daleko. Jako virtuální stroj můžeme totiž považovat operační systémy, které umožňují současný běh více procesů najednou. 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 se sdílí 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. Konečně můžeme říct, že OS poskytuje virtuální prostředí pro každý proces v systému.

TODO partišnování zdrojů operačního systému.

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 vykonává systém hosta a poté je dynamicky překládají do instrukční sady hosta. Konkrétní implementace by mohla například umožňovat vykonávat programy zkompilované pro architekturu IA-32 na systému s architekturou

SPARC. Těmto virtuálním strojům říkáme překladače nebo emulátory, jelikož emulují prostředí dostupné na jiných architekturách a mapují ho do architektury hosta.

Techniky překladu instrukcí jsou detailněji popsaný v kapitole 1.7.1.4. Jen pro ukázku si zmíníme techniku interpretace, která jednotlivé instrukce nejprve načte, dekóduje a poté vykoná ekvivalentní instrukci nebo sadu instrukcí pomocí instrukční sady hosta.

Virtuální stroje v HLL

Virtuální stroje napsané ve vyšších programovacích jazycích přímo navazují na výše popsané téma přenositelnosti mezi platformami. Nevýhoda emulátorů spočívá ve faktu, že se specializují na překlad jedné instrukční sady do druhé. Pokud bychom tedy chtěli docílit přenositelnosti mezi všemi platformami, museli bychom vytvořit emulátory pro každou kombinaci instrukčních sad. Jelikož je toto řešení poněkud složité a náročné na implementaci, můžeme k vytvoření virtuálního stroje použít právě vyšší programovací jazyky. Virtuální stroj napsaný v nějakém HLL se tedy 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, nakterých 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, která je napsaná v jazyce C++ a kterou v dnešní době vyvíjí a udržuje společnost Oracle, je pravděpodobně neznámější a nejvíce využívanou implementací JVM.

1.4.2 Systémové virtuální stroje

Druhým typem virtuálních strojů jsou tzv. *systémové virtuální stroje*, které kompletně virtuálizují celou HW platformu. Virtualizační software je většinou umístěn hned na HW 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.

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 systému na jednom fyzickém počítači.

Všechny operační systémy musí využívat instrukce stejné instrukční sady, kterou vykonává host. Jinými slovy tento typ virtuálního stroje žádným způsobem nepřekládá instrukce do jiné instrukční sady.

Dle umístnění virtualizačního monitoru v architektuře počítače, můžeme rozdělit tento typ virtuálního stroje do dvou hlavních kategorií.

- Nativní VMM
- Hosted VMM

Hlavní 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.

Virtualizace celého systému

V případě, kdy operační systém a jeho aplikace používají stejnou instrukční sadu jako nižší HW vrstvy, má virtuálizační monitor za úkol spravovat přístup k HW a sdílet ho mezi virtuální počítače. Pokud bychom chtěli ve virtuální stroji spouštět operační systém používající odlišnou instrukční sadu, musíme zajistit emulaci prostředí. Jinými slovy VMM musí vytvořit virtuální prostředí pro konkrétní OS tak, aby si OS myslel, že běží na odpovídající HW platformě. Tento typ virtuálního stroje nazýváme *virtualizace celého systému* a jak je z názvu patrné, mimo instrukční sady můžou být virtualizovány i některá I/O zařízení.

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 1.4.1.

Resource partitioning

Frekvence dnešních procesorů již není hlavním měřítkem jejich výkonosti, jak tomu kdysi bylo. Moderní procesory škálují svůj výkon s počtem výpočetních jader a díky tomu můžeme nezávisle spouštět více výpočetních úloh najednou. Tyto jádra jsou propojeny sdílenou pamětí, která zajišťuje prostor pro ukládání dat a komunikaci. Tohoto faktu využívá technika zvaná *resource partitioning*, která prostředky vícejádrového systému spojuje do 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

Obrázek 1.3: Konsolidace serverů

k síti. Jednotlivé fyzické komponenty nebo jejich části jsou tedy přímo přiřazeny konkrétní části (partition) systému. V takto rozděleném systému pak můžeme současně provozovat 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 nastane SW nebo HW chyby 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 sdílení procesoru a jeho jader se například může používat časový multiplexu, 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 dobrých 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 podkapitolách, 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 běží různé aplikace. Vstup procesu je naznačen na obrázku 1.3 vlevo. Výstupem konsolidace je jeden fyzický server s dostatečnými prostředky, na kterém konsolidované systémy běží jako virtuální počítače. Výstup můžeme vidět na obrázku 1.3 vpravo.

Obrázek 1.4: Izolace aplikací

Využití scénáře

Dnes je zcela běžnou praktikou 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 takovýchto 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 [**?**].

Rychlý vývoj technologií v oblasti hadware zapříčiňuje rychlé stárnutí některých systémů a přechod ze staršího na nový může být složitý. Obzvláště v případě, kdy systém potřebuje ke svému běhu speciální hadware. Aby bylo možné provozovat služby poskytované těmito zastaralými systémy, můžeme je spustit jako virtuální počítač na modernějším hadware. Systém se bude chovat stejně jako kdyby běžel na zastaralém hadware, zatímco výkonost služby může těžit z novější a výkonnější hadwarové vrstvy [**?**].

1.5.2 Izolace

Dalším ze scénářů využití virtualizované infrastruktury je izolace aplikací. Proces izolace aplikací spočívá v oddělení dvou a více kritických aplikací běžících na jednom systému do nezávislých virtuálních prostředí. Vstupem je jeden systém s aplikacemi, které se mohou negativně ovlivňovat. Vstup izolačního scénáře je naznačen na obrázku 1.4 vlevo. Výstupem je několik nezávislých virtuálních počítačů, ve kterých běží jednotlivé aplikace. Výstup izolace aplikací je ukázán na obrázku 1.4 vpravo.

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 ohroženy všechny data a aplikace, které na daném systému běží. Vhodným krokem v tomto případě je proto využití virtualizace a rozdělení aplikací 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

Obrázek 1.5: Migrace virtuálního stroje

jednoho VM sice dojde k jeho vyřazení, ale ostatní VM a jejich služby mohou dále pokračovat v běhu.

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 se budeme bavit o přesouvání 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 na konci procesu běží ve virtuálním prostředí nějakého VMM. Dle typu migrovaného systému můžeme rozdělit scénář na následující typy.

Migrace VM

Migrací virtuálního stroje se rozumí přesun VM mezi dvěma různými fyzickými stroji s VMM. Tento přesun byl dříve možný pouze v případě když oba stroje měli stejný HW, operační systém a procesor [**?**]. 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 VM uvnitř VMM. Pokud dojde k selhání některé VM, jiná VM převezme obsluhu požadavků a v minimálním čase dojde k obnovení služby. Druhým typem je selhání celého VMM nebo hosta. V tomto případě je nutné provozovat více redundantních hostů pro VM, které v případě HW chyby převezmou obsluhu služby.

Proces migrace VM je představen na obrázku 1.5, kde můžeme vidět konfiguraci před migrací (vlevo) a po provedení migrace VM (vpravo).

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 tedy systém běžící na stroji bez VMM, jak je naznačeno na obrázku 1.6 vlevo. 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 tak společnost může 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.

Obrázek 1.6: Migrace fyzického na virtuální stroj

1.6 Virtualizační monitor

Jak již bylo zmíněno v definici z kapitoly **??**, virtualizační monitor je softwarová starající se o virtualizaci fyzických prostředků hosta. Vytváří tedy virtuální prostředí pro běh virtuální počítačů. Jinak řečeno VMM vytváří iluzi pro každý virtuální počítač, který si myslí že přímo ovládá HW fyzického systému.

V klasickém nevirtualizovaném prostředí se OS stává po zavedení do hlavní paměti hlavním řídícím programem, který spravuje běh aplikací a vyřizuje požadavky aplikací na HW zařízení pomocí ovladačů. Operační systém tedy pracuje v nejvíce privilegovaném režimu procesoru známého jako *ring 0* a může využívat všechny instrukce instrukční sady.

Druhým případem je prostředí s VMM nebo také virtualizované prostředí. V tomto prostředí přebírá roli hlavního řídícího programu VMM, který je po startu systému zaveden do hlavní paměti. Od té chvíle pracuje VMM v nejvíce privilegovaném režimu CPU a spravuje všechny hostované OS (guest OS). Ovladače k jednotlivým HW zařízení nyní nejsou součástí těchto OS, ale jsou součástí samotného VMM. Každý OS si sám spravuje vlastní aplikace a chová se jako kdyby nebyl izolován virtualizačním monitorem ve virtuálním prostředí.[**?**]

1.6.1 Požadavky na VMM

Z úlohy virtualizačního monitoru při virtualizaci systémů plyne několik základních požadavků, které by měl VMM splňovat. Specifikace těchto požadavků je převzata z [4].

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. Některé základní techniky virtualizace jsou popsány v kapitole 1.7.

Izolace

Virtualizační monitor vytváří virtuální prostředí, které by mělo izolovat jednotlivé instance OS od sebe. Každý virtuální stroj má svůj kontext procesoru

Obrázek 1.7: Nativní (Bare-Metal) VMM

i svoji virtuální paměť mapovanou do jiných oblastí fyzické paměti. Jinak řečeno jednotlivé VM se nemohou vzájemně ovlivňovat svoji činnost a pád jedné VM by neměl ovlivnit činnost ostatních.

Ochrana

Virtualizační monitor by měl být chráněn proti všem vyšším vrstvám virtuální počítačů. Operační systém virtuálního stroje běží v privilegovaném režimu úrovně 1 (ring 1) a VMM běží v nejvíce privilegovaném režimu [1] úrovně 0. Pro operační systém virtuálního počítače to znamená, že nemůže přistupovat do paměti mapované pro VMM a privilegované instrukce vyvolají volání do virtualizačního monitoru, který emuluje jejich chování.

1.6.2 Typy VMM

V následujících třech podkapitolách jsou představeny tři typické architektury VMM, které se liší ve způsobu přístupu k fyzickým prostředkům systému. Obecně lze tyto způsoby rozdělit na přímý a nepřímý přístup.

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 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. Architektura nativního VMM je ukázána na obrázku 1.7

Pokud hypervisor nepodporuje určité typy procesorů nebo periferních zařízení některých HW plaforem, může softwarové vybavení VMM limitovat přenositelnost na tyto platformy.[**?**]

Hostovaný VMM

Druhým typem architektury VMM se nazývá hostovaný, protože využívá ke svému běhu existující a běžící operační systém, který 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 (kernel) v privilegovaném režimu procesoru společně s operačním systémem hosta. Požadavky na HW zařízení jsou

[1] Současné procesory podporují plnou virtualizaci v HW. Mají speciální režim ochrany úrovně -1 speciálně pro VMM [4]

Obrázek 1.8: Hostovaný VMM

Obrázek 1.9: VMM se servisním OS

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 [**?**]. Jednotlivé komponenty hostovaného VMM jsou naznačeny na obrázku 1.8.

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

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 hypervisory, 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 [**?**]. Tento typ architektury je naznačen na obrázku 1.9.

1.7 Techniky virtualizace

O pár odstavců výše jsme si popsali podstatu virtualizačního monitoru a také jsme si představili jeho základní typy. Nyní se pokusíme popsat techniky a principy, které VMM používá pro práci s HW, aby docílil vytvoření virtuálního prostředí.

Na chvilku se oprostíme od VMM a podíváme se na HW počítače z pohledu OS. Přesněji chceme vědět, co potřebuje OS ke svému běhu, abychom byli schopni vytvořit virtuální prostředí pro jeho běh. Důležité je zmínit, že chceme na jednom fyzickém systému provozovat více VM, tedy více instancí OS.

Operační systém je v konečném součtu jenom soubor instrukcí, kterým procesor rozumí a umí je vykonávat. Abychom mohli na fyzickém systému provozovat operační systém nebo jakýkoli jiný program, je nutné aby využíval jen instrukce z dostupné instrukční sady (ISA) procesoru. Pokud je tento požadavek splněn, může být OS na tomto systému provozován bez dalších opatření. Na druhou stranu pokud požadavek splněn není a instrukce programu jsou

napsané v jiné instrukční sadě, je nutné tuto sadu emulovat. Jinými slovy musíme zajistit překlad z jedné instrukční sady do druhé. Tomuto tématu se více věnuje kapitola **??**. Vedle překladu instrukcí je nutné zajistit sdílení procesoru mezi jednotlivými virtuálními stroji.

Vedle vykonávání instrukcí potřebuje OS někde uchovávat svoje datové struktury a svůj kód. To samozřejmě vede k hlavní paměti počítače a nutnosti sdílení této paměti mezi virtuálními počítači.

Nemusíme chodit daleko a podobný princip sdílení prostředků mezi více entit můžeme najít v dnešních operačních systémech. V roli virtuálních počítačů si můžeme představit klasické procesy a roli VMM převezme OS, který má za úkol přidělovat prostředky procesům [5]. Stejně jako v případě virtuálních strojů spolu procesy musí sdílet procesorový čas a hlavní paměť, a proto jsou techniky virtualizace počítačů podobné těm, které se používají v OS pro souběžný běh více procesů.

V neposlední řadě OS potřebuje ovládat zařízení a periferie připojené k fyzickému systému. K čemu by nám jinak byl počítač, kdybychom nevěděli co nám vlastně říká nebo ho dokonce nemohli ovládat.

Všechny výše zmíněné problémy musí VMM řešit, pokud chce vytvořit prostředí pro běh více OS na jednom fyzickém stroji. Ve zkratce můžeme říct, že virtualizační monitor musí implementovat multiplexování CPU, virtualizaci paměti, virtualizaci I/O zařízení a v některých případech i emulaci ISA.

Následujících podkapitoly představují základní techniky, které by měl VMM v nějaké podobě implementovat. Pro lepší pochopení této problematiky si definujeme několik pojmů.

Definice 3 *Kernel mód*

Definice 4 *User mod*

Definice 5 *PC*

1.7.1 Virtualizace CPU

Nyní se opět podíváme na fyzický stroj z pohledu virtualizačního monitoru a na jeho práci s procesorem. Spustit operační systém ve virtuální prostředí nutně musí znamenat, že se nějakým způsobem začnou vykonávat jeho instrukce. Jelikož OS virtuálního počítače neví o existenci VMM, předpokládá, že je v systému sám a má přímý přístup k HW. Na rozdíl od klasické nevirtualizované architektury není OS hlavním řídícím programem, a proto je nutné, aby VMM obsloužil volání privilegovaných instrukcí. Dále je třeba zajistit ochranu paměti alokované VMM před neprivilegovaným přístupem. Neprivilegovaný přístup znamená, že se OS virtuálního počítače nebo procesy v něm běžící pokusí přistoupit k paměti, která je alokovaná virtualizačním monitorem.

1.7.1.1 Režimy ochrany CPU

Moderní procesory mají 4 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, které se někdy přezdívá 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 [4].

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.

1.7.1.2 Multiplexování CPU

Zjednodušeně můžeme říct, že spuštění virtuálního počítače v prostředí virtualizačního monitoru zařídíme skokem na adresu první instrukce OS.

Pro jednoduchost předpokládejme, že máme k dispozici jedno výpočetní jádro CPU a chceme ho sdílet dvěma virtuálními stroji. Oba virtuální stroje mají svojí instanci OS a běží na nich jejich aplikace. Princip střídání virtuálních strojů na CPU je velmi podobný s principem střídání běžících procesů v OS. Operační systém provádí tzv. změnu kontextu (context switch), kdy je přerušen běh aktuálního procesu a procesor je přidělen jinému. V případě VMM je nutné provést tzv. změnu stroje (machine switch). Při tomto procesu dojde nejprve k uložení celého stavu virtuálního stroje, což zahrnuje uložení registrů, PC a na rozdíl od změny kontextu je nutné uložit i aktuální stav HW. Dále je obnoven stav virtuálního stroje, kterému plánovač přidělil procesor a poté je proveden skok na adresu instrukce uvedené v PC [6]. Tímto krokem je proces změny virtuálního stroje na CPU dokončen.

V případě systému s více procesorovými jádry je situace velmi podobná. Jelikož je k dispozici více jader CPU, může v jednom okamžiku běžet více VM. Z tohoto důvodu je nutné synchronizovat přístup těchto virtuálních strojů k HW systému.

1.7.1.3 Virtualizace privilegovaných instrukcí

Instrukční sadu procesoru můžeme rozdělit na dvě části. V první části jsou neprivilegované instrukce, které mohou být prováděny v každém režimu ochrany CPU. Tyto instrukce jsou využívány především uživatelskými programy a ve virtualizovaném prostředí je můžeme přímo provádět na CPU bez nutnosti nějakých opatření. Pokud aplikace nebo OS virtuálního počítače potřebují vykonávat tento typ instrukcí, dojde k jejich provedení na CPU bez nutnosti zásahu VMM.

Druhou částí ISA, jsou tzv. privilegované instrukce, které mohou být prováděny pouze v privilegovaných režimech CPU. Některé privilegované instrukce mohou přímo skryté paměti cache, TLB, vyvolávat přerušení nebo jiným způsobem ovlivňovat běh systému. Operační systém ve virtualizovaném prostředí nemůže mít možnost vykonávat privilegované instrukce, protože by pak ovládal fyzický systém a od toho je na systému přítomen VMM. Z tohoto důvodu musí virtualizační monitor nějakým způsobem zachytávat pokusy o vykonávání privilegovaných operací a zajistit jejich správné provedení.

Jedním z případů, kdy VMM musí reagovat na volání privilegované instrukce, je situace, kdy se proces operačního systému snaží provést systémové volání. Příkladem systémového volání může být volání funkce `open()` pro otevření souboru na disku, volání funkce `write()` pro zápis dat do souboru nebo třeba volání funkce `exec()` pro vytvoření dalšího procesu v systému.

Nejprve se podíváme na to, jak funguje systémové volání v klasických systémech bez virtualizace. Konkrétně se podíváme na architekturu x86, kde se pro dosažení systémového volání používá speciální instrukce přerušení `int` s argumentem `0x80`. V ukázce 1.1 je zobrazen kód assembleru ze systému FreeBSD [**?**], který spouští systémové volání funkce `open()` pro otevření souboru. Z kódu 1.1 je vidět, že volání funkce `open()` potřebuje tři parametry a to konkrétně mód otevření souboru, příznaky a cestu k souboru na disku. Typ systémového volání je určen díky poslednímu parametru na zásobníku, kterým je v tomto číslo 5 [6]. A konečně na posledním řádku ukázky se volá zmíněná instrukce `int`, která vyvolá přerušení.

Kód 1.1: Systémové volání na FreeBSD

```
open:
        push    dword mode
        push    dword flags
        push    dword path
        mov     eax, 5
        push    eax
        int     80h
```

Z pohledu virtualizačního monitoru je důležité, co se děje po vyvolání přerušení. Jelikož se jedná o privilegovanou instrukci, dojde k přepnutí procesoru z uživatelského režimu (user) do privilegovaného režimu (kernel) a kontrolu

Proces	Hardware	Operační systém
1. *Systémové volání:* Vyvolání přerušení;		
	2. *Kernel mód:* Skok na obsluhu přerušení;	
		3. *Obsluha přerušení:* Dekódování přerušení a spuštění odpovídající rutiny OS; Návrat z OS;
	4. *User mód:* Návrat z obsluhy;	
5. *Obnova běhu;*		

Tabulka 1.1: Sytémové volání bez virtualizace [6]

dostane operační systém. Konkrétně je řízení předáno rutině starající se o obsluhu přerušení, která byla zaregistrována v HW operačním systémem při jeho prvním startu [6]. Po obsloužení systémového požadavku je řízení vráceno zpět volajícímu procesu. Posloupnost volání je naznačena v tabulce 1.1 společně s přepínáním režimu ochrany procesoru.

Je logické, že případě virtualizace bude nutné přidat několik kroků do posloupnosti volání. Rozdílem oproti předchozímu případu je fakt, že OS nemá nad systémem plnou kontrolu a neběží v plně privilegovaném režimu. Tuto funkci nyní plní VMM, který má v HW zaregistrovanou rutinu pro obsluhu přerušení. Pokud tedy nějaký uživatelský proces chce provádět systémové volání, bude postupovat stejně jako v předchozím případě. Nicméně zpráva o přerušení se nedostane k OS ale k VMM, jak je naznačeno v druhém kroku v tabulce 1.2. Virtualizační monitor nemůže přímo vyřídit systémové volání, protože mu nerozumí a neví co má dělat (není OS). Tuto funkci plnil operační systém a jeho rutina pro obsluhu přerušení. Jak již bylo zmíněno, při prvním startu OS se systém pokusí registrovat v HW rutinu pro obsluhu přerušení, což je ale privilegovaná operace [6]. Tento pokus VMM odchytí a ke každému běžícímu OS si zaznamená adresu těchto rutin. Virtualizačnímu monitoru tedy stačí spustit rutinu korespondující OS v méně privilegovaném režimu a počkat na její dokončení. V okamžiku ukončení obsluhy systémového OS opět provede privilegovanou instrukci a řízení je předá zpět VMM, jak je ukázáno ve čtvrtém kroku tabluky 1.2. Konečně VMM přepne procesor do uživatelského režimu a vrátí řízení zpět uživatelskému programu, čímž je systémové volání ukončeno. V tabulce 1.2 jsou z důvodu úspory místa vynechány operace HW, které přepínají režimy procesoru.

Z obou tabulek je patrné, že při virtualizaci se musí vykonat více kroků,

Proces	Operační systém	Virtualizační monitor
1. *Systémové volání:* Vyvolání přerušení;		
		2. *Obsluha přerušení:* Skok na obsluhu přerušení odpovídajícího OS (snížení privilegií)
	3. *Obsluha přerušení:* Dekódování přerušení a obsloužení systémového volání; Návrat z OS;	
		4. *Obsluha ukončena:* Provedení reálného návratu;
5. *Obnova běhu;*		

Tabulka 1.2: Sytémové volání s virtualizací [6]

čímž se zvyšuje doba vykonávání privilegovaných instrukcí a to může mít za následek zhoršení výkonu systému [6].

1.7.1.4 Emulace instrukční sady

Doposud jsme se bavili o situaci, kdy stroje běžící ve virtualizovaném prostředí zpracovávaly instrukce stejné architektury jako host. Nyní se krátce podíváme na situaci, kdy virtuální stroje zpracovávají instrukce jiné architektury než fyzický stroj.

V tomto případě procesor nerozumí instrukcím VM a je nutné zajistit překlad z jedné ISA do druhé. Tato technika se nazývá emulace instrukční sady a spočívá v navození iluze, že virtuální stroj běží na HW podporující jeho instrukční sadu.

- Přímá interpretace
- Binární překlad
- Inkrementální binární překlad za běhu

[7]

1.7.2 Virtualizace Paměti

1.7.3 Virtualizace I/O zařízení

Kapitola 2

Solaris

Solaris nebo dříve SunOS je operační systém původně vytvořený firmou Sun Microsystems, ale v současné době 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

V době psaní této diplomové práce je nejaktuálnější stabilní verze operačního systému Solaris verze s označením 11.3, avšak ke dni 30. ledna 2018 byla do světa vypuštěna beta verze 11.4 [**?**]. 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 pořízení architektury SPARC by bylo složitě, bude pro účely této diplomové práce použita architektura x86 přesněji její 64 bitová verze.

2.2.1 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 SPARC

Scalable Processor Architecture neboli SPARC je z pohledu Solarisu domovská architektura. Architektura SPARC byla stejně jako Soalris 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 škálovatelnost řešení.

2.3 Služby

Hlavní předností operačního systému je kvalita ale i kvantita jeho služeb. Tyto služby umožňují nasazení toho 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 [8]. Pro účel startu jsou v systému definovány speciální služby tzv. *milestone*, které ve skutečnosti nic nedělají. 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 [**?**]. 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 nezabírají téměř žádné 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. Jako příklad můžeme uvést virtualizační techniku Solaris Zones, která je hlavním tématem této diplomové práce.

2.3.3 Virtualizace

Dle specifikace [9] 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 vytvořené prostředí jsou izolovány 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 diku, paměti nebo CPU na rozdíl od zbylých dvou modelů.

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

2.3.3.2 Virtuální počítače

Model *virtuálních počítačů* popsaný v kapitole **??** umožňuje souběžný běh více instancí operačního systému na jednom fyzickém stroji. Konkrétně 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, že jsou na fyzickém stroji samy. 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. 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 ne přímo 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 hypervisoru, a tudíž tato technika poskytuje reálný výkon systému. Hardware partitions je technika poskytující běžícím operačním systémům největší izolaci, ale na druhou stranu není tak flexibilní v konfiguraci prostředků jako výše zmíněné modely.

Tato technika není z logických důvodů poskytována OS 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.

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 strojů na jednom fyzickém stroji. Jak již bylo zmíněno v kapitole 2, tato technika je standardní součástí operačního systému Oracle Solaris od verze systému Solaris 10. Aktuální verze je Solaris 11.3 a přináší novou funkcionalitu v podobě podpory starších verzí operačního systému Solaris.

Pokud bychom chtěli zařadit Solaris Zones do klasifikace virtuálních strojů popsané v kapitole 1.4, pak bychom jí zařadili do sekce *virtualizace na úrovni OS*. Jinak řečeno, tato technika 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 pak můžeme 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. Jinými slovy, je to zóna, která běží přímo na hardwaru počítače nebo ve virtualizovaném prostředí. Tato zóna má dvě hlavní funkce. Za prvé tato zóna plní funkci hlavního operačního systému a přebírá kontrolu nad fyzickými prostředky po startu systému. Za druhé je 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 ostatních zón. Z tohoto důvodu je doporučené používat globální zónu pouze pro účely administrace systému a managementu ostatních zón.

Zóny, které jsou spouštěny v rámci globální zóny nazýváme **neglobální** 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 nějakým 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.

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 někde 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 [5]. 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 z jiné zóny, nemůže k tomu využít mezi procesovou komunikaci, ale musí použít počítačové sítě. Naopak procesy, které běží v rámci jedné zóny spolu mohou komunikovat pomocí signálů, sdílené paměti a tak podobně. 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 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 typ neglobální zóny. Tato vlastnost se specifikuje při konfiguraci zóny v globální zóny a dle přehledu [10] může typ zóny mít následující hodnoty:

- *solaris*
- *solaris-kz*
- *solaris10*

Brand neboli typ zóny určuje jakým způsobem se zóna bude po spuštění chovat. Implicitním typem zóny v Solaris Zones je *solaris*, kterému se také jinak přezdívá nativní zóna nebo také tenká zóna. Dalším typem zóny je *solaris-kz*, kde zkratka za pomlčkou v názvu odpovídá slovnímu spojením kernel zone. Jak název napovídá, tato zóna má vlastní jádro operačního systému a někdy se jí také přezdívá 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 a umožňuje uvnitř této zóny spouštět aplikace určené pro tento systém.

V následujících podkapitolách jsou podrobněji popsány výše zmíněné typy zón.

3.1.1.1 Nativní zóna

Nativní neboli tenká zóna umožňuje administrátorovi vytvořit zónu, která má sdílené jádro operačního systému s globální zónou. Jinými slovy verze jádra operačního systému musí být stejná jako v globální zóně. Tento typ zóny je izolovaný pouze nad svým souborovým systémem a nemá 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. Pokud chceme této zóně delegovat nějaký typ zařízení, musíme tak učinit při konfiguraci zóny v globální zóně. Tímto způsobem můžeme nativní zóně zpřístupnit souborové systémy, ZFS pool nebo ZFS dataset. Takto definované prostředky pak po instalaci zóny můžeme z této zóny využívat.

Dále tento typ zóny má svoji vlastní databázi produktů, která obsahuje informace o všech nainstalovaných softwarových komponentech v konkrétní globální zóně. Opět platí, že konkrétní neglobální zóna vidí pouze své balíčky. Díky tomu je možné instalovat dodatečné softwarové balíčky do neglobálních zón, které nemusí být nainstalované v globální zóně [10]. Některé softwarové

balíčky jsou ale společné s globální zónou (kernel) a nelze tedy provádět 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 nějakým 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 můžeme pozorovat podobnost s technikou NAT v sítích. Jako druhý typ rozhraní můžeme 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 jinou ip adresu než zóna globální.

Tento typ zóny nepodporuje vytváření další neglobálních zón. Jinými slovy se nativní neglobální zóna nemůže chovat jako globální zóna a vytvářet nové zóny uvnitř sebe. Stejně tak z nativní zóny nemůžeme vytvářet ani spravovat jiné neglobální zóny.

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 [10].

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 balíčků. Jelikož i kernel zóna je neglobální, nelze z ní žádným způsobem vidět balíčky ostatních zón. Na rozdíl od nativní zóny administrátor může provádět aktualizaci všech balíčků, protože kernel zóna nesdílí nic s globální zónou. Žádné balíčky tedy nejsou závislé na balíčkách globální zóny.

V případě síťových rozhraní kernel zóny podporují pouze rozhraní typu *exclusive-ip* a neumožňuje sdílet síťovou adresu s globální zónou.

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

Požadavky

Jelikož provoz kernel zóny se liší od provozu standardní nativní zóny, liší se i požadavky na hostitelský systéme. Požadavky se liší v závislosti na platformě. Pro jednoduchost si uvedeme pouze požadavky pro systémy s architekturou x86 a procesorem intel. Podle specifikace [11] se na hostitelský systém kladou následující požadavky.

- Procesor musí být 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 balíček *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 na virtualizované platformě VMvare, kde jsou splněné výše zmíněné požadavky může vypadat následovně.

```
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* [10] 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 podkapitole 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.

Tabulka 3.1: Porovnání typů zón a jejich vlastností

Neglobální zóna může být několika druhů, které poskytují různé vlastnosti. Tabulka 3.1 poskytuje stručný přehled základních vlastností jednotlivých druhů zón.

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 a správných chod neglobálních zón vyžaduje bezproblémový chod globální zóny. Administrátor takového systému by tento fakt 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)`. Více o tomto nástroji a možnostech konfigurace zón je popsáno v kapitole 3.3.

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

Čerstvě 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 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 se bude moc 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. Zónu můžeme převést z jednoho stavu do druhého pouze nějakou kombinací nástrojů `zonecfg(1)` a `zoneadm(1)`. Podle specifikace [12] 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 nějakou podmnožinu příkazů, které zónu ovládají. Pro příklad můžeme uvést, že nelze zónu spustit pokud se nachází například ve stavu *configured*. Pro spuštění se zóna musí nacházet ve stavu *installed*. V následujících odstavcích je stručně shrnut význam jednotlivých stavů.

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 asociován žádný diskový obraz a tedy nemá připojený kořenový souborový systém. Tento stav je v pořadí první stav, ve kterém se zóna může od svého vzniku nacházet. Nachází se v něm buď bezprostředně po vytvoření konfigurace pomocí `zonecfg(1)` nebo když 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 `zonecfg(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 vrací 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 migrujeme neglobální nativní zónu 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. Jinými slovy ještě nemůže být 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 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 [12]. Tento stav je tranzitní a nastává v okamžiku kdy zahajujeme boot zóny pomocí příkazu `zoneadm(1)`.

3.2.2.6 Running

Ve stavu *running* se zóna nachází když 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, inicializované všechny 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 [12].

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, které zmíníme jenom v rychlosti, jelikož nejsou tolik podstatné. Jedná se o stavy *suspended*, *debugging*, *panicked*, *migrating-out* a *migrating-in*. Jména stavů jsou samovysvětlující, a proto nemá smysl je podrobněji popisovat.

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 vytváře, měnit nebo mazat konfigurace jednotlivých neglobálních zón. Dle manuálových stránek [13] 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 zón. 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. Tím jsou 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 si syntaxi ukážeme na následující výpisu 3.1 konfiguračního souboru zóny. V ukázce 3.1 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 nějaké globální atributy zóny. Více o nich v následující podkapitole 3.3.1. Dále můžeme v konfiguračním souboru vidět příkaz `add`, který reprezentuje přidání zdroje k zóně. V tomto případě se jedná o přidání síťového rozhraní s automatickou konfigurací. Základní typy jednotlivých prostředků popisuje podkapitola 3.3.1. Následuje opět sekvence příkazů `set`, která se ale nyní váže k předchozímu příkazu `add` a nastavuje tak lokální atributy daného

Kód 3.1: Ukázka konfigurace 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
```

síťového rozhraní. Celý konfigurační soubor je ukončený příkazem `end`, který reprezentuje výstup z konfigurace prostředku.

Nástroj `zonecfg(1)` umožňuje dva módy editace konfigurace zóny. Prvním mód je editace konfigurace uložené v souborovém systému. Změna konfigurace v tomto módu, žádným způsobem neovlivní běžící zónu. Druhý způsobem je editace konfigurace v takzvaném živém módu, která 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 [13] umožňuje příkaz `zonecfg(1)` konfigurovat třináct globálních atributů zóny. V této kapitole nebudeme popisovat všechny, ale zaměříme se na nejdůležitější z nich. Důraz bude kladen především na atributy, které jsou nezbytné pro vytvoření zóny.

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 balíčky 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 v kapitole 3.1.1, typ neglobální zóny určuje jak 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

Stejně jako předchozí atribut byl i atribut určující typ síťové adresy již zmíněn v kapitole 3.1.1. V krátkosti pouze naznačíme, že tento atribut určuje jestli bude síťová adresa sdílená s adresou globální zóny či nikoli. Tento atribut se nastavuje 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 zmíníme dva atributy, které souvisí se spouštěním a vypínáním neglobálních zón. Atribut *autoboot* vyjadřuje jestli se má daná neglobální zóna spustit při startu zóny globální. Tento atribut může nabývat dvou hodnot. Jestliže chceme aby se zóna spustila při startu globální zóny, musíme nastavit hodnotu tohoto atributu na *true*. V opačném případě nastavíme hodnotu *false*, která je implicitní hodnotou tohoto atributu. O automatické spouštění neglobálních zón se stará systémová služba `svc:/system/zones:default`, a proto je nutné, aby byla spuštěna [13].

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

3.3.2 Zdroje

Mimo generických atributů umožňuje nástroj `zonecfg(1)` přidávat do konfigurace zóny takzvané zdroje. Zdroj je objekt nějakého typu, který má svoje lokální atributy a do konfigurace zóny se přidává pomocí příkazu `add`. Zdroj reprezentuje většinou nějaké zařízení, souborový systém, prostředky pro přidělování zdrojů zónám nebo například specifikuje uživatele, který může zónu administrovat. Některé zdroje mohou být do konfigurace zóny přidány vícekrát. V takovém případě jim `zonecfg(1)` automaticky přidělí číselný identifikátor, který daný zdroj jednoznačně určuje. Podle manuálových stránek [13] umožňuje konfigurace zón přidávat až jednadvacet různých typů zdrojů. Stejně jako v případě globálních atributu si zde popíšeme jenom zdroje, které jsou nejdůležitější pro správný běh neglobální 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 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, jestli zóna bude moci 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 [14] 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 3.2 Podle doporučení v [13] 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.

Kód 3.2: Konfigurace zařízení kernel zóny

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

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 sítových 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 další 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 síťového zdroje, který může být neglobální zóně přiřazen. 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 [13] opravdu velké množství a opět si popíšeme jenom některé z nich. 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. Jinými slovy přes toto zařízení bude proudit provoz generovaný vytvořeným síťovým rozhraním. Další atributy již nebudeme podrobně zmiňovat. Ostatní parametry slouží například k nastavování MAC adresy, ochrany linkové vrstvy, VLAN a mnohého dalšího. V ukázce 3.3 je naznačeno jak by mohla vypadat konfigurace automatického síťového rozhraní pomocí nástroje `zonecfg(1)`.

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 na při-

Kód 3.3: Konfigurace síťového rozhraní

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

dělováním prostředků jednotlivým neglobálním zónám, umožňuje konfigurace specifikovat některé omezení na využívání zdrojů. Tyto omezení se do konfigurace přidávají jako kterékoli jiné zdroje. Podle manuálových stránek [13] 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. Hodnotou 1 se tedy myslí, že daná zóna může využít 100% procesorového času.

Posledním zdrojem, který si zmíníme 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).

Kód 3.4: Delegace administrace jinému uživateli

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

3.3.2.4 ZFS Dataset

Standardně mají neglobální zóny ponětí pouze o svém kořenovém souborovém systému a nevidí žádný jiný. Pokud chce zóna využívat nějaké další úložiště, musíme jí delegovat buď celý disk nebo můžeme 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 v kapitole 3.2.1, 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 různé privilegia, které mu umožňují nějakým způsobem spravovat zónu. Podle manuálových stránek [13] jsou možná privilegia následující.

- *login*
- *manage*
- *copyfrom*
- *config*
- *liveconfig*

Názvy jednotlivých privilegií odpovídají akcím, které může daný uživatel se zónou provádět. Za zmínění snad stojí jenom privilegium *copyfrom*, které umožňuje administrátorovi vytvářet nové zóny jako klony dané neglobální zóny. V kódu 3.4 je ukázána konfigurace která umožňuje uživateli *zadmin* přihlašování a základní správu dané zóny.

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 není nic jiného než konfigurace zóny, která je již zaregistrovaná v systému. Pokud tedy administrátor již vytvořil předem nějaké neglobální zóny a chce jejich konfiguraci znovu využít, stačí specifikovat jejich jméno jako argument příkazu `create`. Tím vznikne úplně nová konfigurace zóny, která má ale 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 ostatní zón v adresáři `/etc/zones`. Zvídavý uživatel může zjistit, že adresář obsahuje například šablony *SYSdefault.xml*, *SYSsolaris10.xml* nebo *SYSsolaris-kz.xml*, které odpovídají jednotlivým typů neglobálních zón. V ukázce kódu 3.5 je viditelné rychlé vytvoření konfigurace zóny se jménem *z1*, které využívá systémovou šablonu *SYSdefault*.

Kód 3.5: Vytvorení 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
```

Z ukázky je patrné, že šablona *SYSdefault* nastavuje základní atributy zón s použitím implicitních hodnot a přidává jedeno 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. Více o tomto archivu je popsáno v kapitole 3.6, která se zabývá zálohováním. Druhou možností archivu je složka se 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 *configured* tak i ve stavu *running*. Pokud zóna není spuštěná administrátor může editovat pouze konfiguraci, která je uložená v adresáři `/etc/zones`. V případě, že konfigurovaná zóna běží může si administrátor vybrat jestli 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 totiž 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 nějakého zdroje, například síťového rozhraní, musí použít příkaz `set` pro jeho výběr jak je naznačeno v [15] 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 operace, která se dá s konfigurací zóny provádět, je mazání. K tomuto účelu slouží příkaz `delete` nástroje `zonecfg(1)`. Tato akce nemůže být vrácena, a proto by si administrátor měl tento úkon řádně promyslet. 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 umožňuje několik poskytuje administrátorovi několik způsobů pro vytvoření kořenového souborového systému zóny. Dle specifikace [16] a manuálových stránek [17] 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 zdrojové souborového systému zón. Kritériem pro výběr může být například doba trvání instalace, protože mezi jednotlivými typy můžeme pozorovat zásadní rozdíl. 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 balíčků, které jsou pro operační systém Solaris poskytovány. Standardní poskytovatel této databáze je společnost Oracle, která hlavní databázi balíčku poskytuje jako webovou službu dostupnou z *pkg.oracle.com/solaris/release*. 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 musí nacházet ve stavu *configured*. Po puštění příkazu začne instalátor stahovat potřebné balíčky 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*, 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, balíček s jádrem se do nativní zóny neinstaluje. V případě kernel zóny zde nastává rozdíl a balíček s jádrem se nainstaluje. Podle článku [18] je možné tímto typem instalace nainstalovat 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 balíčky má nainstalovat do kořenového souborového systému. Podrobněji je manifest popsán v kapitole 3.4.1.1. Druhý parametr odpovídá cestě k tzv. systémovému profilu, který specifikuje systémové nastavení. Tento parametr je společný pro všechny typy instalace a je podrobněji popsán v kapitole 3.4.4.

Po úspěšném dokončení instalace jsou všechny softwarové balíčky 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 zapříčiněno tím, že se všechny balíčky musí stáhnou pomocí počítačové sítě.

3.4.1.1 Manifest

Manifest je soubor obsahující definici všech softwarových balíčků, které má instalátor nainstalovat do souborového systému zóny. Tento soubor je ve formátu XML a je povinnou součástí instalace. V minulé kapitole je řečeno, že tento parametr je volitelný. V případě, kdy uživatel nespecifikuje cestu k souboru explicitně, je použit předdefinovaný soubor. Podle specifikace [19] 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 tedy možnost si vytvořit vlastní XML soubory s definicemi softwarových balíčků. Tyto soubory pak může používat při instalaci nových zón. V ukázce kódu 3.6 je naznačeno, jak by mohl takový uživatelský manifest vypadat. Pro přehlednost je zobrazena pouze část s definicí balíků a zbytek XML dokumentu je vynechán.

Kód 3.6: Ukázkový 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 nechceme instalovat zónu přímo z repozitáře můžeme využít možnosti instalace z archivu. Archiv je soubor, který obsahuje zdrojový souborový systém zóny neboli diskový obraz. V tomto případě se nemusí balíčky stahovat přes počítačovou síť z repozitáře, ale jsou zkopírovány ze zdrojového archivu. V předchozím případě instalátor zajistí, že se z repozitáře stáhnou správné verze balíčků a administrátor se o to nemusel starat. V tomto případě může

nastat problém s nekompatibilitou balíčků 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 balíčků. 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 opět pomocí příkazu `install` nástroje `zoneadm(1)`, 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 nainstalované zóny. Instalace z archivu je rychlejší než instalace z repozitáře, ale nekompatibilita globální zóny a archivu může vyústit v neúspěch.

3.4.3 Klonování

Posledním podporovaným druhem instalace je klonování zóny. Vstupem klonování je již existující nakonfigurovaná a nainstalovaná 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 změní. Jak říká specifikace [20], vytvoření klonu trvá zlomek času klasické instalace.

Klonování zón se spouští pomocí příkazu `clone` nástroje `zoneadm(1)`, který bere zdrojovou zónu jako parametr. Pro efektivní využívání tohoto typu instalace je třeba dodrže následující požadavek. Aby bylo opravdu použito funkce klonování souborového systému ZFS musí se zdrojová i cílová zóna nacházet ve stejném ZFS poolu. V opačném případě je zdrojový souborový systém opravu zkopírovaná a není využito této techniky. Dále si administrátor musí dát pozor při konfigurování cílové zóny a musí 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 [20] se všechny aktualizace balíčků, 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 [21] umožňuje konfigurovat kteroukoli systémovou službu v rámci SMF. Tento soubor se tedy 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 kódu 3.7. Pro přehlednost jsou vynechány ostatní nepodstatné části souboru.

Kód 3.7: Konfigurace uživatele root

```
<service name="system/config-user" version="1" type="service">
  <instance name="default" enabled="true">
    <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>>
  </instance>
</service>
```

V ukázce je vidět, 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 zmíněno v manuálových stránkách [22], pomocí parametru *grouping* lze stanovit typ služby, pro kterou chcete vygenerovat systémový profil. Pokud administrátor 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 můžou měnit jejich stav. K účelu správy zón se používá opět nástroj `zoneadm(1)` a jeho příkazy. Podle manuálových stránek [17] může administrátor pro základní správu neglobálních zón používat následující příkazy.

- `list`
- `ready`
- `boot`
- `shutdown`
- `halt`

První příkaz v pořadí 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, na které chceme danou operaci provést. V ukázce kódu 3.8 je předveden výpis příkazu `list`. Pro úsporu místa byl z výpisu vynechán atribut *zonepath*

Kód 3.8: 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
```

Příkaz `boot` slouží pro start zón zatímco příkazy `shutdown` a `halt` slouží k jejímu zastavení. Podrobnější informace o těchto příkazech poskytují následující kapitoly.

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. Jinými slovy 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 tehdy kdy se nachází ve 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 ke konzoly. Pokud se zóna před spuštěním příkazu `boot` nenacházela ve stavu *ready*, je automaticky spuštěn stejnojmenný příkaz v rámci příkazu `boot`.

Podle manuálových stránek [17] 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 [17] ekvivalentem volání příkazu `/usr/sbin/init 0` v dané zóně. Tento příkaz počká až se validně ukončí všechny služby systému. Pomocí

dalšího přepínače je možné specifikovat, že chceme aby se zóna vzápětí opět nastartovala.

Pokud volání příkazu `shutdown` trvá moc dlouho nebo chceme zónu rychle zastavit, můžeme 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 ke konzoly dané neglobální zóny. Pomocí této konzole pak 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 tyto dvě věci je nutné 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*.

3.6.1 Záloha a obnova pomocí ZFS

Prvním 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 nezabírá téměř žádné 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 zařídí připojení souborového systému ZFS 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 přesměrování 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 [23] 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 zóna.

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 obnovíme konfiguraci zóny přímo z archivu pomocí nástroje `zonecfg(1)` a následovně spustíme příkaz `zoneadm install`, kde jako parametr specifikujeme cestu k archivu.

Výhodou zálohy pomocí UAR je její jednoduchost. Celý proces je otázkou několika málo příkazů. Nevýhodou naopak je, že uživatel potřebuje speciální práva na práci s nástrojem `archiveadm(1)`. Další nevýhodou je fakt, že se nedá vytvářet více archivů najednou ale musíme archivovat více zón do jednoho archivu.

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 využívající virtualizační techniky Solaris Zones mohou být případem takové infrastruktury a i Solaris Zones poskytují několik technik pro migraci neglobálních zón.

Migrace v kontextu Solaris Zones je přesun neglobální zón z jedné globální zón do globální zóny na vzdáleném počítači. Jak je zmíněno v manuálových stránkách [17] a administrátorské příručce [24] 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

Následující kapitola 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 kapitoly jsou 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 označen za splněný.

Jak bylo zmíněno v kapitole , 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 platformě *x86*. Hlavní důraz musí být kladen na kompatibilitu programovacího jazyka a jeho knihoven s operačním systémem Solaris. Z výše zmíněných důvodů je možné vyvodit první požadavek na administrační nástroj, který 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 . Mezi tyto postupy patří především vytváření neglobálních zón, ale také podpora jejich správy, zálohování nebo migrace. Automatickou správou se myslí hlavně automatizace procesů vytváření zóny, zálohy nebo migrace, které se skládají z několika kroků. Aplikace by tedy měla administrátorovi poskytovat funkce, které umožní provedení výše zmíněných procesů pomocí jednoho příkazu. Požadavky na apli-

kaci vyplývající z účelu nástroje je možné pojmenovat 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 ale chybí podpora pro správu zón na vzdálených serverech. V dnešních infrastrukturách počítačových systémů využívající virtualizaci serverů se nachází mnoho serverů, které poskytují své výpočetní prostředky virtuálním strojů. Z tohoto pohledu je tedy žádoucí, aby implementovaná aplikace umožňovala správu neglobálních zón, které se nacházejí na **vzdálených** serverech.

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

Uživatel musí mít možnost jak ovládat nástroj pro podporu automatické 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 toho 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**

Obrázek 4.1: Funkční bloky aplikace

Splněním těchto požadavků by mělo vést k značnému zjednodušení správy virtualizačního kontejneru Solaris Zones. Dále by výsledná aplikace měla zajistit přehled o neglobálních zónách, které se nacházejí na vzdálených serverech a také umožňovat jejich správu.

4.2 Architektura aplikace

Prvním krokem v návrhu aplikace je její architektura. Architektura aplikace popisuje její strukturu a určuje jak 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 nějakou akci se zónou, aplikace sestaví z těchto nástrojů potřebný příkaz nebo jejich sekvenci a vykoná je. Mimo výše zmíněných 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 ty bude simulovat práci administrátora při vykonávání základních administračních rutin tím, že bude vykonávat výše zmíněné příkazy na příkazové řádce.

Pro usnadnění vývoje rozdělíme aplikaci 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. Jinými slovy tato vrstva bude poskytovat základní funkce pro vytvoření, správu, zálohu a migraci neglobálních zón. Poslední částí aplikace bude klientská aplikace, které bude skrze knihovnu využívat funkce modulu. Na obrázku 4.1 jsou vidět jednotlivé funkční bloky aplikace a jejich vzájemná interakce. V následujících kapitolách je detailněji představena funkcionalita jednotlivých funkčních bloků aplikace.

4.2.1 Knihovna

Knihovna je jakýsi kontejner, který se bude start 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 chceme 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 pak budou implementovat základní administrátorské rutiny, které budou složené z funkcí nižších vrstev. Mimo jiné tento modul bude zajišťovat smazání všech dočasných souborů, které byli 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 případe, 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 implementovat 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 prostředek, jak přímo spravovat konkrétní zónu. Z tohoto důvodu bude na klientské aplikace, aby zařídila možnost správy velké 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 pak 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 nějakým způsobem interagovat s danou aplikací. Tento prvek je možné implementovat pomocí mnoha technologií. Ne všechny typy uživatelského rozhraní jsou ale vhodné pro konkrétní typy aplikací.

Hlavní podstatou navrhovaného nástroje je podpora automatizace správy. Jinými slovy 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, ale musí 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. Hlavní 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 v následujících podkapitolách.

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 případe 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 popsaný v kapitole 3.3. Po vyplnění formuláře bude aplikace pokračovat klasickým 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 4.2.1 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 muset být 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 bylo popsáno v kapitole 4.1, 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 [13], ale jejich podmnožinu, která je využitelná ve většině případech.

4.4.1.2 Softwarové balíčky

Další podstatnou částí šablony bude sekce se softwarovými balíčky. Balíčky, které zde uživatel definuje, budou při instalaci zóny z této šablony nainstalovány do kořenového souborového systému zóny. Ihned po první startu zóny je uživatel bude moci využívat.

Přehled softwarové balíčků dostupných pro konkrétní verzi operačního systému Solaris může uživatel získat pomocí nástroje pro správu softwarových balíčků `pkg(1)` nebo z oficiálního repozitáře [25].

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 k správné a funkční konfiguraci systému.

Minimální konfigurace obsahuje definici hesla uživatele *root* a 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 z 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 nějakou 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 zadat větší počet neglobálních zón, pro které se má daný příkaz vykonat. Aplikace potom paralelně vykoná daný příkaz nad danou zónou.

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 těchto 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 tedy 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 virtualizačních 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 rozhraní 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 dávat zabezpečení aplikace velkou váhu. 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 zón v systému 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 [26], která administrátorovi umožňuje jemněji rozdělit práva mezi uživatele. S 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**
- Vytvořený uživatel s právy (**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 ale několik důvodů proč není vhodné uživatele *root* používat pro spouštění navrhovaného nástroje pro automatickou správu Solaris Zones. Prvním důvodem je ctění principu nejnižších privilegií [27]. 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 [26], 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 ale nemá v systému žádnou funkci a nedá se na ni dokonce ani přihlásit. Z tohoto důvodu by nebylo možné přihlašovat se vzdálených systémech jako uživatel *root* a aplikace by nesplňovala požadavek vzdálené správy.

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

V důsledku používání uživatele *root* ke spouštění aplikace by došlo k porušení principu nejnižších privilegií [27] a navíc by muselo dojí k vypnutí některých 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 potřebné oprávnění pro používání potřebných nástrojů. Za prvé bude aplikace spouštěna v souladu s principem nejnižších oprávnění a za druhé nebude nutné vypínat bezpečnostní opatření nástroje `ssh(1)` ani jiné systémové nastavení.

Kapitola 5

Implementace

V následující kapitole je představena implementace nástroje pro automatickou správu Solaris Zones. Hlavní částí je popis knihovny, modulu Solaris Zones a klientské aplikace. Důraz je kladen na popis funkcí jednotlivých funkčních bloků 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 bude stavět hlavně nad využíváním nástrojů na příkazové řádce a zpracovávání jejich výstupu. Pro tento účel bude vhodné zvolit interpretovaný programovací jazyk, který umožní jednoduše nástroje spustit a následně snadno zpracovat jejich výstup. Na základě výstupu se pak nástroj rozhodne o dalším průběhu zpracování uživatelského příkazu. První podmínka není ve volbě tolik omezující. Pro operační systém Solaris totiž existuje implementace standardního kompilátoru `gcc(1)` pro jazyk C a stejně tak i 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 je nutné při volbě jazyka dbát hlavně na dostupnost grafických knihoven pro operační systém Solaris. Standardním skriptovacím jazykem pro většinu operačních systému typu UNIX je `shell`. 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 zvolíme programovací jazyk Ruby [28], který umožní splnit obě dvě stanovené podmínky.

5.1.1 Ruby

Ruby je objektově orientovaný programovací jazyk, který má mnoho možností pro využití. Jedním z 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 [29] existuje několik implementací interpretu jazyka Ruby, z nichž nejpoužívanější jsou YARV [30] a JRuby [31]. 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 ale nejsou na na operačním systému Solaris podporované. Z toho důvodu bude 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 bude tedy využívat programovacího jazyka Ruby. Pokud bude chtít uživatel nástroje využívat grafického rozhraní bude muset nástroj spouštět pomocí interpretu JRuby. Zbytek nástroje bude nezávislý na použitém interpretu programovacího jazyka Ruby.

5.2 Knihovna

Hlavním centrálním prvkem implementace bude knihovna, která bude zprostředkovávat 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 Virtualbox. Výsledná implementace bude obsahovat pouze modul pro podporu automatické správy virtualizačního kontejneru Solaris Zones, který bude dále podrobně popsán v kapitole 5.3.

Knihovna bude poskytovat hlavní rozhraní, pomocí kterého bude moci klient využívat funkcí jednotlivých modulů. Jednotlivé moduly tedy budou sloužit jako hlavní zdroj funkcionality pro knihovnu.

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

Jelikož moduly budou 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 mohli jednotně komunikovat s knihovnou a také aby knihovna mohla zprostředkovávat jejich funkce klientovi. Definice rozhraní modulu je podrobně popsána v kapitole 5.2.1.

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

5.2.1 Rozhraní modulu

Povinné rozhraní modulu bude sloužit především ke komunikaci mezi knihovnou a samotným modulem. Funkcionalitu, kterou modul musí poskytovat můžeme shrnout do následujících tří 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 volitelné parametry. Hlavním smyslem této rutiny je inicializace daného modulu. Hlavní knihovna v rámci inicializační smyčky spustí tuto rutinu pro každý registrovaný modul. V rámci této rutiny modul může 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.

Dalším 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. Jinými slovy modul, který podporuje správu virtualizačního kontejneru, 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 jak 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 v kapitole 5.2.1, knihovna si udržuje odkazy na hlavní třídy modulů, které reprezentují jejich funkční rozhraní.

V okamžiku, kdy klientská aplikace vznese požadavek na zavolání nějaké funkce, knihovna v době 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 vyvolala 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 názorně ukázáno na obrázku 5.1.

5.2.3 Generická šablona

Poslední funkcionalitou knihovny je definice generické šablony, která má za úkol specifikovat virtuální stroj. Hlavním úkolem generické šablony bude specifikace typu virtuálního stroje. Tento atribut bude určovat, 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 bude tedy zajišťovat validaci těchto dvou atributů a v případě úspěchu předá šablonu zodpovědnému modulu.

Knihovna tedy bude klientským aplikacím poskytovat funkce pro načítání a validaci šablon. V případě úspěšného načtení šablony bude knihovna vracet objekt, který bude moci být použit v rámci konkrétního modulu. Validace šablony bude provedena ve dvou krocích. Knihovna nejprve zjistí jestli daná

Kód 5.1: Generická šablona

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

šablona obsahuje atribut jména a typu. Podle typu šablony se knihovna rozhodne jakému modulu ji předá na druhý krok validace. Způsob validace je popsaný v kapitole 5.2.3.2.

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 bude použit datový formát JSON [32], který bude sloužit jako reprezentace 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 bude součástí uživatelského rozhraní klientské aplikace.

Struktura šablony se bude skládat ze dvou částí. První částí bude název a definice typu šablony. Tato hlavička bude určovat způsob zacházení s danou šablonou. V ukázce kódu 5.1 je naznačeno, jak 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. Z důvodu úspory místa byly tyto atributy v ukázce vynechány.

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 použitý datový formát JSON, může být pro validaci šablony použité tvz. JSON schéma definované v [33]. Tento dokument je opět ve formátu JSON, ale neslouží pro ukládání dat. Jeho funkcí je definovat formát jiného dokumentu JSON. Jinými slovy pomocí tohoto schématu je možné specifikovat atributy a typy jejich hodnot, které má konkrétní typ dokumentu obsahovat.

Pomocí tohoto nástroje je tedy možné 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 bude implementovat základní schéma, které

Kód 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
}
```

bude sloužit pro základní validaci šablon. Jak je vidět v ukázce 5.2, toto schéma vyžaduje, aby v dokumentu byly přítomné atributy *name* a *type*. Podle atributu *type* je rozhodnuto, kterému modulu bude konkrétní šablona předána. Moduly aplikace musí implementovat podrobnější schéma, které bude definovat konkrétní typ virtuálního stroje.

Aby bylo možné v programovacím jazyku Ruby validovat JSON dokumenty, je nutné použít použít knihovnu, která bude implementovat JSON schéma. Pro tento účel bude aplikace využívat volně dostupné řešení json-schema [34], které implementuje funkce validace dokumentů typu JSON pomocí schémat.

5.3 Modul Solaris Zones

Modul Solaris Zones bude hlavním stavebním kamenem celé implementace výsledného nástroje. Tento modul bude zařazen do knihovny popsané v kapitole 5.2 a mimo jiné bude poskytovat 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. V rámci tohoto modulu se bude jednat o šablony neglobálních zón.

Obrázek 5.2: Modul Solaris Zones

Pro jednoduchou orientaci a nezávislost bude modul rozdělen do následujících vrstev, které budou poskytovat různou funkcionalitu a budou se vzájemně využívat.

- Management šablon
- Nástroje pro správu Solaris Zones
- Administrátorské rutiny
- Funkční rozhraní modulu

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

5.3.1 Šablona Solaris Zones

Nástroje pro správu Solaris Zones neposkytují možnost vytvářet zóny pomocí jednoho předpisu. Jak bylo popsáno v kapitole 4.1, 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 šablona kombinovat vlastnosti výše zmíněných souborů.

Kostra šablony pro neglobální zónu je naznačena v ukázce kódu 5.3. Z ukázky je patrné, že šablona obsahuje tři sekce, které korespondují s jednotlivými soubory vyžadovanými při instalaci. Pro úsporu místa jsou z ukázky vynechány konkrétní atributy zón, které jsou nahrazeny třemi symboly tečky. Modul tedy musí implementovat JSON schéma, které bude využívat pro validaci šablony a mechanizmy pro její zpracování. Jinými slovy modul musí být ze šablony schopný reprodukovat soubor s konfigurací, manifestem a systémovým profilem. Tyto soubory pak nástroj bude využívat pro instalaci neglobální zóny s parametry, které jsou určené v dané šabloně. Pokud bude modul zpracovávat rutinu využívající šablonu, v prvním kroku dojde k rozdělení šablony do třech zmíněných částí. V následujícím kroku budou tyto části převedeny do vnitřní reprezentace a následně zpracovány.

5.3.1.1 Zpracování konfigurace

Konfigurační sekce šablony se bude skládat z definice globálních atributů zóny popsaných v kapitole 3.3.1 a z definice zdrojů zóny, které jsou popsané v

Kód 5.3: Kostra šablony neglobální zóny

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

kapitole 3.3.2. Jednoduché globální atributy budou v šabloně specifikovány přímo pomocí jejich jména a hodnoty. Pro globální atribut typu zóny by mohla definice vypadat takto následovně `"brand" :"solaris"`.

Dále bude tato sekce šablony obsahovat definici zdrojů zóny, které mají složitější strukturu a obsahují několik atributů. Z tohoto důvodu bude šablona obsahovat speciální atribut `resources`, který bude typu pole a bude obsahovat definici všech zdrojů zóny. V rámci jednotlivého zdroje bude použita stejná technika definice atributu jako ve výše zmíněném případě.

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 `manifest` šablony 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 balíčku. V tomto atributu může uživatel specifikovat libovolné množství balíčků.

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 tedy jednotlivé balíčky načteny ze šablony a ve správném formátu vloženy do kopie tohoto souboru.

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 bude mít stejnou strukturu jako sekce s konfigurací zóny. Na rozdíl od způsobu zpracování konfigurace 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 bude existovat korespondující soubor obsahující potřebnou část výsledného XML souboru. Tyto soubory budou uloženy v kořenovém adresáři aplikace ve složce *szones/profile*. V průběhu zpracování šablony pak budou jednotlivé soubory načítány a vyplňovány hodnotami ze šablony. Tímto způsobem se zkonstruuje celý soubor, který může být předán instalátoru.

5.3.2 Nástroje

Základní základním stavebním kamenem modulu bude vrstva, která bude zajišťovat vykonávání potřebných příkazů na příkazové řádce. Tato vrstva bude poskytovat vyšším vrstvám modulu možnost vykonávání základní administračních příkazů pro správu Solaris Zones a souborového systému ZFS. Konkrétní příkazy jsou vyjmenovány v kapitole 4.2.2.

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

Pomocí výše zmíněné třídy bude tato vrstva modulu umožňovat volání jednotlivých nástrojů s požadovanými parametry a argumenty. Například bude umožňovat spouštět nástroj `zonecfg(1)` se jménem konfigurované zóny a cestou k souboru s konfigurací. Vyšším vrstvám pak bude poskytnutý standardní výstup a standardní chybový výstup daného nástroje.

5.3.3 Administrátorské rutiny

Hlavní částí modulu bude vrstva tříd, které budou implementovat požadované rutiny pro správu Solaris Zones. Rutiny budou využívat nástrojů implementovaných v nižší vrstvě a budou pomocí nich vytvářet sekvence příkazů nutné k vykonání požadované činnosti. Tato vrstva bude poskytovat pokročilejší rutiny pro vytváření neglobálních zón, zálohu, obnovu a migraci. Mimo jiné bude také poskytovat základní funkce pro správu zón.

5.3.3.1 Transakce

Jednotlivé rutiny budou implementované jako transakce. Transakce se bude skládat z jednotlivých příkazů, které budou tvořit celek. V případě úspěchu všech příkazů v transakci bude celá transakce označena za úspěšnou. Jestliže jakýkoliv příkaz v průběhu transakce selže, selže i celá transakce.

V průběhu některých rutin dochází k vytváření dočasných souborů. Tyto soubory souží 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 nemám na vykonání tohoto procesu vliv. V případě Dále některé transakce vytvářejí nové konfigurace zón, snapshoty konkrétních souborových systémů nebo celé diskové obrazy zón. Některé z těchto entit mají být například výsledkem 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 nějakým 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 bude v rámci knihovního modulu zajišťovat třída `Rollbacker`.

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

5.3.3.2 Vytváření zón

Hlavní součástí administrátorských rutin budou funkce pro vytváření zón. Tyto rutiny budou umožňovat 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 vytvoření zóny je pomocí standardních tří 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 poskytnuta podpora pro vytváření zón specifikovaných v šablonách popsaných v kapitole 5.3.1. V průběhu této rutiny nejprve dojde k transformaci šablony, popsané ve stejné kapitole, na standardní soubory. Tato transformace proběhne na vždy na lokálním serveru. Dále rutina pokračuje stejně jako v případě instalace ze standardních souborů popsané výše.

Výše zmíněné rutiny pro vytváření zóny neměli k dispozici diskový obraz zón a instalace zóny musela vždy probíhat z repozitáře. Následující dva způsoby tvorby neglobálních zóny využívají jako zdroj již existující diskový obraz. První z těchto dvou postupů využívá diskový obraz již existující zóny. Kroky této rutiny můžeme rozdělit na části získání diskového obrazu a samotné instalace zóny. Obě tyto části mohou být prováděny buď na lokální serveru nebo na vzdáleném. Sever odkud je zóna získávána se označuje jako **zdrojový** a server kde je vytvářená nová zóna se nazývá **cílový**. Před získáváním diskového obrazu se nástroj nejprve musí ujistit jestli je zóna v konzistentním stavu a tedy jestli není spuštěná. Pokud je, 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 popsaná v kapitole 3.4.3 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. Tvorba obou těchto archivů je popsána v kapitole 3.6. 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 nabírá na váze v případě, kdy již existuje zóna se jménem zóny, kterou chce rutina vytvořit. V případě, že je tento parametr zapnutý, rutina danou existující zóny smaže a na 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 jestli 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 archivu a druhý způsob používá přímo nástroje souborového systému ZFS. Rutiny pro zálohu budou podporovat oba tyto způsoby a budou se lišit pouze v technice vytvoření daného archivu.

Oba typy zálohovacích rutin bude možné provádět na lokálních i vzdálených serverech. V prvním kroku je nutné uvést danou neglobální zónu do konzistentního stavu a v 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 v podstatě 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 již existující zóny. Podstatou je vytvoření archivu zóny na zdrojovém serveru a přesun tohoto archivu na server cílový. Jak již bylo popsané v následující kapitole, vytvoření archivu je 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 server se musí opravdu smazat až v případě, kdy je zóna kompletně nakonfigurována a zdárně nainstalována na cílovém serveru. Dřívější smazání zdrojové zóny by mohlo vést ke ztrátě dat.

Další rozdíl je v typu 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 server 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 budou poskytovat 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` poté již nemá skoro žádnou práci protože nemusí souborový systém extrahovat z archivu.

Všechny výše zmíněné funkce budou umožňovat migraci mezi všemi hosty dané infrastruktury. Jinými slovy bude umožněno migrovat lokální zónu na vzdálený server a naopak a také 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 bude mimo výše zmíněných pokročilejších rutin poskytovat i základní rutiny pro manipulaci a správu neglobálních zón. V těchto rutinách bude zahrnuto spouštění, násilné i nenásilné vypnutí, restart nebo úplné odstranění zóny ze systému. Všechny tyto akce bude 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 bude rozhraní, které bude nabízet klientským aplikacím prostřednictvím knihovny. Pro tyto účely bude vytvořena speciální třída, která bude zprostředkovávat administrátorské rutiny popsané v minulé kapitole. Princip rozhraní bude fungovat stejně jako v případě knihovny. Toto rozhraní bude mít zaregistrované všechny třídy jejichž metody chce veřejně poskytovat knihovně a klientským aplikacím. Pokud knihovna obdrží požadavek na volání nějaké rutiny, přesměruje tento požadavek právě na toto rozhraní. Rozhraní vyhledá v rámci zaregistrovaných tříd zadali 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 toto rozhraní reprezentováno třídou `SZONESAPI`, která má zaregistrované třídy modulu 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 bude tato klientská aplikace zprostředkovávat uživateli pomocí následujících komponent.

- Uživatelské rozhraní (CLI)
- Správa vzdálených hostů
- Uživatelský žurnál

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

Jak bylo zmíněno v požadavcích 4.1, nástroj musí umožňovat správu vzdálených zón. Tato funkcionalita je již implementována v modulu Solaris Zones. Klientská aplikace musí implementovat pouze způsob, jak jednoznačně identifikovat neglobální zónu v rámci infrastruktury. Dále bude klientská aplikace spravovat databázi virtualizačních hostů. Pomocí této databáze bude možné vykonávat některé rutiny hromadně, napříč všemi registrovanými hosty.

Poslední funkcionalitou klientské aplikace udržování tzv. uživatelského žurnálu. Tento žurnál bude sloužit uživateli pro uchovávání stavů jednotlivých neglobálních zón. V případě změny stavu registrované zóny jiným uživatelem, bude 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 kapitole 4.3.1. Uživatel může toto rozhraní ovládat pomocí příkazů, které určují typ prováděné operace. Příkazy můžeme rozdělit do dvou skupin. První skupina příkazů mění vnitřní stav klientské aplikace a budou popsány v kapitolách 5.4.2 a 5.4.3.

Druhá skupina příkazů slouží k 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 bude přesně specifikovat 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 taky 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 pak může vypadat následovně *z1:host1*.

V následujících kapitolách bude 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. Mimo tyto způsoby poskytuje příkaz `deploy` interaktivní instalaci, která využívá grafického rozhraní. Tento způsob je podrobněji popsán v kapitole 5.5.2. Implicitní způsob vytvoření zóny je právě pomocí interaktivní instalace.

Jako argumenty uživatel musí specifikovat minimálně jednu neglobální zónu pomocí výše zmíněného identifikátoru. Jméno zóny se použije pro vytvoření a doménové jméno se použije pro připojení ke vzdálenému hostu. Pro všechny takto specifikované zóny bude použit stejný zdroj a příkaz bude 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

V pořadí druhý příkaz `backup` slouží pro zálohování zón. Tento příkaz umožňuje zálohovat zóny dvěma způsoby, které již byly popsány v kapitole 5.3.3.3. Jmenovitě se jedná o archivaci pomocí 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 se záloha všech zón na tomto serveru provádí paralelně. V druhém případě tomu tak není, 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 Migrate

Příkaz `migrate` zajišťuje 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, které jsou popsány v kapitole 5.3.3.4. Jedná se o migraci 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.

5.4.1.4 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 jakýkoli vzdáleny server. Implicitní chování tohoto příkazu vytvoří instanci specifikované šablony na každém serveru, který je v aplikaci registrovaný. Funkcionalita registrace serverů bude teprve vysvětlena v kapitole **??**. Chování tohoto příkazu je tedy 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.

5.4.1.5 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 na specifikovaných zónách 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.

5.4.2 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 pak 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`

První příkaz v pořadí `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 je možné vidět v ukázce kódu 5.4.

Příkaz `delete` slouží k odstranění daného hosta z databáze aplikace. Při této akci dojde i k smazání specifikace projení SSH k tomuto hostu. 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 globální identifikátoru, který obsahuje doménové jméno. V okamžiku připojování ke konkrétnímu vzdálenému server je použito právě nastavení uložené v databázi. Pokud pro daného hosta neexistuje záznam v databázi, jsou použiti implicitní hodnoty.

Pro každého uživatele takto komponenta udržuje jeho vlastní databázi v jeho 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. Hlavním soubor databáze se nazývá *hosts.json* a obsahuje seznam doménových jmen jednotlivých hostů. Pro

Kód 5.4: Implicitní nastavení parametrů SSH připojení

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

každý záznam v tomto souboru je vytvořen další souboru, 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*.

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 kteroukoliv 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, jestli se daná zóna od předchozí manipulace nějakým způsobem změnila. Konkrétně jestli se změnil její stav nebo diskový obraz. Mimo jiné žurnál umožňuje detekci nových zón, které se v infrastruktuře od posledního spuštění aplikace objevily.

Práce této komponenty je z větší části automatická, ale 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. Jinými slovy odstraní zónu ze seznamu sledovaných.

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 uložený 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, jestli bylo se zónou nějakým způsobem manipulováno. Ukládané atributy zóny a struktura záznamu je patrná z ukázky kódu 5.5. V uživatelském žurnálu je takovýto záznam pro každou sledovanou zónu.

Kód 5.5: Záznam stavu zóny v 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 vritualizačního kontejneru Solaris Zones bude grafické uživatelské rozhraní. Jeho hlavní účel je 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.

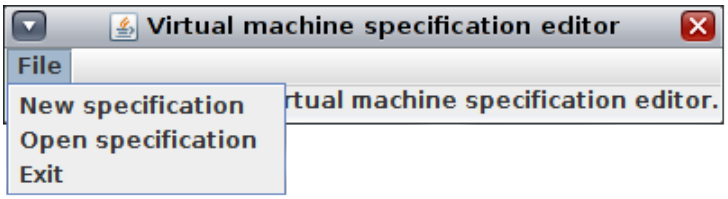
Implementované grafické rozhraní využívá grafické knihovny Swing, která je podporována pouze na platformě JVM. Z důvodů uvedených v kapitole 5.1.1 je použití grafického rozhraní podmíněno využitím interpretu JRuby jako platformu pro programovací jazyk Ruby. V případě, že uživatel spustí grafické rozhraní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

5.5.1 Editor šablon

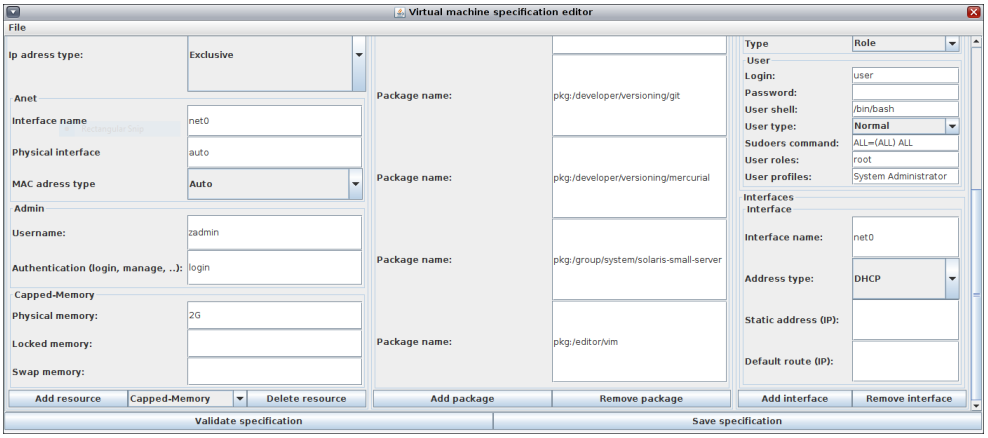
První využití grafického rozhraní je editor pro manipulaci se šablonami zón. Tento editor se spouští pomocí příkazu `editor`, který je součástí uživatelského rozhraní klientské aplikace. Po spuštění tohoto příkazu je uživateli zobrazeno grafické okno, které slouží pro editaci šablon.

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 a to vytvoření šablony, je v prostřední části editoru zobrazen ten samý formulář, ale vyplněný implicitními hodnotami. Ovládací menu společně s úvodní obrazovkou editoru je zobrazeno na obrázku 5.3. Formulář pro vyplňování atributu šab-

Obrázek 5.3: Ovládací menu editoru šablon



Obrázek 5.4: Formulář editoru šablon



lony 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 vidět na obrázku 5.4, každá sekce editoru má ve spodní části ovládací prvky. Tyto prvky slouží k přidávání a odebírání 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 balíčky, které mají být v instancí dané šablony nainstalované. Poslední část umožňuje přidávat a odebírat nastavení pro definované síťové adaptéry.

Poslední částí editoru je hlavní spodní ovládací panel, který obsahuje dvě funkční tlačítka. V případě použití první 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 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 k specifikaci vytvářených zón. Uživatel nyní nebude mít na výběr z načtení šablon, bude muset hodnoty zadat ručně. Některé atributy jsou vyplněny implicitními hodnotami a uživatel musí zadat minimálně počáteční heslo uživatele *root* a 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, jehož implementace je popsána v kapitole 5. Zaměřuje se především testování jednotlivých scénářů použití nástroje a zkoumá jeho chování. Na začátku této kapitoly je definováno prostředí, ve kterém byly testy prováděny. Následuje série testů, které zkoumají funkčnost 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 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 přiřazeny doménové jména *shost*, *shost1* a *shost2*, a přidány korespondující řádky do souboru */etc/hosts*. Toto

nastavení umožňuje používat specifikované doménové jména místo IP adres a zjednoduší tak identifikaci strojů v testovacích ukázkách.

Jelikož provozování virtualizační technologie Solaris Zones vyžaduje nemalé množství výpočetní 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 nebude uvedeno jinak, testovaný nástroj bude vždy spouštěn z virtuálního počítače s doménovým jménem *shost*.

Posledním učiněným krokem byla konfigurace uživatele *zadmin*, který má práva na vykonávání příkazů nutných k 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 některých 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 se by uživatel chtěl pomocí implementovaného nástroje dosáhnout. Následně je popsán stav prostředí, ve kterém se sytém nachází před provedením konkrétní akce. Dále proveden korespondující příkaz v uživatelském rozhraní nástroje, který má splnit stanovený cíl. Výsledek tohoto kroku je ověřen pomocí systémových příkazů a v závěru je rozhodnuto, zda bylo dosaženo stanoveného cíle.

6.2.1 Vytvoření neglobálních 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 se 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. Z 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 balíčky pro správu zdrojového kódu. Tyto balíčky 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 systémový administrátor. 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 C.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 rovnou spustí vytvořené zóny.

Z konce standardního výstupu nástroje v ukázce kódu C.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 ke konzoly dané zóny. Úspěšné přihlášení v ukázce kódu C.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 ukázky C.4 patrné, že došlo k vytvoření síťového adaptérů *net0* a jeho automatické konfigurace pomocí služby DHCP. Daná neglobální zóna tak byla okamžitě po vytvoření dostupná ze sítě. Přítomnost softwarových balíčků byla otestována pomocí jejich rozhraní na příkazové řádce jak je vidět v ukázce C.4. Posledním kritériem úspěchu bylo správné nakonfigurování počátečního systémového uživatele. Jméno a heslo bylo ověřeno již při přihlašování do zóny. Zbývalo tedy ověřit jestli uživatel má práva systémového administrátora, což bylo provedeno pomocí příkazu `profile`. Výpis na ukázce C.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 konstatován jako 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

Obrázek 6.1: Migrační scénář

rámci infrastruktury. K tomuto účelu byl využitý stav, ve kterém se systém nacházel po testování předchozího scénáře popsaného v kapitole 6.2.1. 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 ukážky kódu C.3, 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 provedenou akcí.

Následně byla simulována situace, kdy jiný uživatel změní nějakým 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* změnil. Součástí výpisu by měla být i informace o nově vytvořené zóny v rámci infrastruktury. Z ukázky kódu C.2 je vidět, že uživatelský žurnál opravdu informuje uživatele o změně sledované zóny a na konci výpisu je zobrazena informace o nově vytvořené zóně. Pro úsporu místa byly ostatní zóny 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 uživatelský žurnál a jeho konkrétní záznam pokud danou zónu vytváří. Podobným způsobem bylo ověřeno, že uživatelský žurnál je aktualizován i při mazání a změně stavu. 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 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 posílání souborového systému, kterou implementovaný nástroj poskytuje. Scénář migrace je graficky znázorněn na obrázku 6.1.

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 měli nacházet na cílovém serveru *shost2* a na zdrojových serverech by se neměly nacházet žádné neglobální zóny. Uživatelský žurnál zobrazený v ukázce C.5 tuto skutečnost potvrzuje.

Pro migraci neglobálních zón slouží příkaz `migrate`, který je možné využívat skrze uživatelské rozhraní implementovaného nástroje. Právě tento příkaz byl použitý pro testování tohoto scénáře a v ukázce C.6 je možné vidět jeho výstup. Jak je z konce výstupu patrné, všechny zóny byly podle nástroje úspěšně přesunuty na cílový server *shost2*. Pro ověření funkcionality bylo nutné ověřit jestli se zóny opravdu nachází na cílovém serveru, jestli jsou zóny smazány ze zdrojových serverů a také jestli 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 globální zóny. Výstup těchto kroků je zobrazen v ukázce C.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 v ukázce C.5 by se tedy měl změnit tak, že přesouvané zóny budou zobrazeny pod hostem *shost2*. Tato skutečnost je potvrzena ukázkou C.8, která ukazuje očekávaný výstup z 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 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

Posledním testovaným scénářem bylo vytvoření zálohy a následná obnova. Jelikož se jedná o dvě samostatné funkce nástroje, byl tento test rozdělený do dvou scénářů. V prvním scénáři použití bylo otestováno vytvoření zálohy

několika zón. Následující scénář potom využil vytvořené zálohy k obnovení 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 zón. 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 chceme zálohu uložit. 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*

Dále byl na lokálním serveru *shost* 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 bylo spuštěna záloha výše zmíněných zón. Výstup tohoto příkazu je zobrazen v ukázce C.9 a ukazuje, že záloha zón proběhla úspěšně. Zálohy by se podle výstupu programu měly nacházet v složce */zonepool/backup* na lokálním serveru.

Pomocí standardního nástroje `ls(1)` bylo ověřeno, že se zálohy opravdu vytvořili a že se nachází v určeném adresáři na lokálním serveru. Tento testovací scénář potvrzuje, že nástroj je schopný najednou vytvářet zálohy více neglobálních zón runě umístěných v infrastruktuře serverů. Dále potvrzuje, že nástroj umí stáhnout zálohy to 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 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 nástroje byla spuštěna obnova z výše zmíněných archivů. Tento proces je vyznačen v ukázce C.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 prohození diskových obrazů daných zón. Podle výstupu nástroje proběhla obnova zón v pořádku a zóny by měly být nainstalované 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 tvar definovaný v kapitole 5.3.3.3. V případě tohoto scénáře se jednalo o zálohy typu UAR, které byly vytvořeny v rámci předchozího testování. Závěrem je možné říct, že testování tohoto scénáře využití bylo úspěšné.

6.3 Měření

V rámci testování nástroje pro podporu automatické správy virtualizačního kontejneru Solaris Zones bylo provedeno měření doby běhu některých funkčních prvků. Konkrétně se jednalo o měření doby běhu vytváření a migrace neglobálních zón v rámci popsané infrastruktury. V obou případech se přihlíželo k jiným parametrům měření, které jsou detailněji popsány v následujících dvou kapitolách. K měření bylo opět použito prostředí definované v kapitole 6.1.

6.3.1 Vytváření zón

Prvním objektem měření bylo sledování doby běhu nástroje při vytváření zón v závislosti na počtu vytvářených zón.

6.3.2

6.3.3 Závěr testování

Všechny testované scénáře byly prováděny v rámci prostředí definovaném v kapitole 6.1. Vybrané scénáře použití by měly pokrývat hlavní funkcionalitu nástroje a jejich splnění by mělo vypovídat funkčnosti celého nástroje. Jeli-

kož jejich testování proběhlo bez chyb a s očekávanými výsledky, je možné konstatovat, že funkcionalita splňuje požadavky stanovené v kapitole 4.1.

Závěr

Virtualizace se stala běžnou a možná i nezbytnou součástí dnešního počítačového světa.

Literatura

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

[2] staff, V.: *The Virtues of Network Virtualization* [online]. [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] 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

[5] 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

[6] Arpaci-Dusseau, R. H.; Arpaci-Dusseau, A. C.: *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 0 vydání, Květen 2015. Dostupné z: http://www.ostep.org

[7] Kašpar, J.; Tvrdík, P.: *Techniky virtualizace III.* [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-poa10.pdf

[8] Muzikář, Z.; Žďárek, J.: *Start systému, proces init, Solaris Service Management Facility* [online]. Praha, České vysoké učení tech-

nické 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

[9] 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

[10] 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

[11] 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

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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

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

[21] 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

[22] 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

[23] 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

[24] 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

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

[26] 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

[27] 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

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

[29] 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

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

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

[32] 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

[33] TODO: *The Ruby Programming Language on the JVM* [online]. [cit. 2018-04-27]. Dostupné z: http://json-schema.org/latest/json-schema-core.html

[34] TODO: *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

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

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**

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
```

Příloha C

Testování

Kód C.1: Vytvoření neglobálních zón ze šablony

```
# szmgmt_cli deploy -b zdev zdev zdev:shost1 zdev:shost2 -s ~/zdev.json
Solaris zones deployment from virtual machine specification initialized.
  ------------------------------------------------------------
  Options:
               Boot zones: enable
    Rewrite existing zones: disable
                   Source: specification </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
```

Kód C.2: Uživatelské žurnál po změně

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

Kód C.3: Uživatelské žurnál 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
```

Kód C.4: Ověření správného vytvoření zóny

```
zadmin@shost2:~$ zlogin -C  zdev
[Connected to zone 'zweb' console]
Hostname: solaris
solaris console login: admin
Password:
admin@solaris:~$ ifconfig net0
net0: flags=100001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,PHYSRUNNING>
    inet 10.164.85.13 netmask ff000000 broadcast 10.255.255.255
admin@solaris:~$ git --version
git version 1.7.9.2
admin@solaris:~$ hg --version
Mercurial Distributed SCM (version 3.4)
simactom@solaris:~$ profiles
System Administrator
...
```

Kód C.5: Uživatelský žurnál před migrací

```
zadmin@shost:~/repositories/szmgmt$ bundle exec bin/szmgm_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
```

Kód C.6: Migrace zón

```
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' to perform migration (direct).
  Processing migration of zone 'zmigr1:localhost' to host shost2.
    See log '/export/home/zadmin/.szmgmt/log/zmigr1_migration_ph5o7o.log'.
  Processing migration of zone 'zmigr2:shost1' to host shost2.
    See log '/export/home/zadmin/.szmgmt/log/zmigr2_migration_uy1frm.log'.
  Processing migration of zone 'zmigr3:shost1' to host shost2.
    See log '/export/home/zadmin/.szmgmt/log/zmigr3_migration_lhc24y.log'.
  Processing migration of zone 'zmigr:localhost' to host shost2.
    See log '/export/home/zadmin/.szmgmt/log/zmigr_migration_h2rdat.log'.
  ------------------------------------------------------------
  Migration finished.
    Status:
      localhost:
        zmigr:  success
        zmigr1: success
      shost1:
        zmigr2: success
        zmigr3: success
```

Kód C.7: Stav zóny na serverech po migraci

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

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

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

Kód C.8: Uživatelský žurnál po migraci

```
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
```

Kód C.9: Vytvoření zálohy zón pomocí 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.
  ------------------------------------------------------------
  Options:
          Backup directory: /zonepool/backup
          Destination host: shost
  ------------------------------------------------------------
  Connecting concurrently to hosts 'shost2, shost1' to perform backup (UAR).
  Processing zone backup of 'zback2:shost1'.
    See log '/export/home/zadmin/.szmgmt/log/zback2_backup_syikav.log'.
  Processing zone backup of 'zback:shost2'.
    See log '/export/home/zadmin/.szmgmt/log/zback_backup_71blh4.log'.
  Processing zone backup of 'zback1:shost2'.
    See log '/export/home/zadmin/.szmgmt/log/zback1_backup_5rks8b.log'.
  Processing zone backup of 'zback3:shost1'.
    See log '/export/home/zadmin/.szmgmt/log/zback3_backup_shmtgw.log'.
  ------------------------------------------------------------
  Backup finished.
    Status:
      shost2:
        zback: success
        zback1: success
      shost1:
        zback2: success
        zback3: success
```

Kód C.10: 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' to recovery.
  Processing zone 'zback1' recovery on host 'shost2'.
    Source archive: /zonepool/backup/zback1_backup_1525020859.uar.
    See log '/export/home/zadmin/.szmgmt/log/zback1_recovery_vai7x.log'.
  Processing zone 'zback2' recovery on host 'shost1'.
    Source archive: /zonepool/backup/zback2_backup_1525020859.uar.
    See log '/export/home/zadmin/.szmgmt/log/zback2_recovery_lpl0c1.log'.
  Processing zone 'zback' recovery on host 'shost2'.
    Source archive: /zonepool/backup/zback_backup_1525020859.uar.
    See log '/export/home/zadmin/.szmgmt/log/zback_recovery_7ecl60.log'.
  Processing zone 'zback3' recovery on host 'shost1'.
    Source archive: /zonepool/backup/zback3_backup_1525020859.uar.
    See log '/export/home/zadmin/.szmgmt/log/zback3_recovery_7ta1fr.log'.
tes  ------------------------------------------------------
  Recovery finished.
    Status:
      shost2:
        zback: success
        zback1: success
      shost1:
        zback2: success
        zback3: success
```