VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VÍCEÚROVŇOVÁ VIRTUALIZACE PRO SIMULACI KOMPLEXNÍHO PROSTŘEDÍ

MULTI-LEVEL VIRTUALIZATION FOR SIMULATION OF COMPLEX ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP ZELINKA**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Zelinka Filip**
Program: Informační technologie
Název: **Víceúrovňová virtualizace pro simulaci komplexního prostředí**
Multi-Level Virtualization for Simulation of Complex Environment
Kategorie: Softwarové inženýrství

Zadání:

1. Seznamte se s procesem virtualizace a vnořené virtualizace. Seznamte se s existujícími VMM (Virtual Machine Monitor).
2. Analyzujte možné vlastnosti víceúrovňové virtualizace pro simulaci komplexního prostředí.
3. Na základě provedené analýzy a po konzultaci s konzultantem firmy SolarWinds Czech s.r.o. specifikujte požadavky a navrhněte architekturu aplikace pro simulaci komplexního prostředí.
4. Zvolte vhodné vývojové prostředí a implementujte prototyp navržené aplikace.
5. Použitelnost aplikace otestujte v reálném prostředí a demonstrujte na vzorku dat vybraném po dohodě s vedoucí.
6. Zhodnoťte dosažené výsledky a navrhněte možné rozšíření projektu.

Literatura:

- LOWNDS, Patrick, Charbel NEMNOM a Leandro CARVALHO. *Windows Server 2016 Hyper-V Cookbook*. Second Edition. Birmingham: Packt Publishing, 2017. ISBN 978-1-78588-431-3.
- MARSHALL, Nick. *Mastering VMware vSphere 6*. Indianapolis, IN: Wiley, 2015. ISBN 978-1-118-92515-7.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kreslíková Jitka, doc. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Tato bakalářská práce se zabývá problematikou neefektivního vytváření virtuálního prostředí. Zvolený problém jsem vyřešil pomocí sady skriptů, která využívá víceúrovňovou virtualizaci. Skript se dá modifikovat pomocí jednoduchého uživatelského rozhraní. Vytvořené řešení poskytuje nástroj pro generování komplexních virtualizovaných prostředí s předinstalovaným operačním systémem. Přínosem této práce je zvýšení efektivity při vytváření komplexních virtualizovaných prostředí.

Abstract

The Bachelor thesis deals with a problematics of inefficient creation of virtual environment. I have solved this problem by creating a set of scripts that uses multilevel virtualization. The script can be modified by simple user interface. The solution provides an ability to create complex virtualized environments with preinstalled operating system. The benefit of this work is to increase efficiency in creating complex virtualized enviroments.

Klíčová slova

virtualizace, server, Hyper-V, hypervizor, virtuální stroj, správce virtuálních strojů, Windows Server 2016, powershell

Keywords

Virtualization, server, Hyper-V, hypervizor, virtual machine, virtual machine manager, Windows Server 2016, powershell

Citace

ZELINKA, Filip. *Víceúrovňová virtualizace pro simulaci komplexního prostředí.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

Víceúrovňová virtualizace pro simulaci komplexního prostředí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc. Další informace mi poskytl Ing. Jakub Dražka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Filip Zelinka
16. května 2019

Poděkování

Tímto bych chtěl poděkovat své vedoucí bakalářské práce doc. RNDr. Jitce Kreslíkové, CSc. za pomoc a za rady při zpracování této práce. Mé poděkování také patří Ing. Jakubovi Dražkovi za odborné konzultace a rady při zpracování této práce.

Obsah

Kapitola 1

Úvod

1.1 Motivace

Ve firemní sféře nastává často situace, kdy je potřeba efektivně vytvářet nové, čisté prostředí pro testování a vývoj. Tento problém se často řeší nakupováním drahého hardwaru a opakujícím se procesem přípravy prostředí. Toto řešení však není vždy únosné a už vůbec ne efektivní.

Představme si, že máme klient-server produkt, u kterého podporujeme zpětný servis několika verzím a tento produkt spravuje a monitoruje klientovo serverové prostředí. Klientovo serverové prostředí může být propojeno různou síťovou strukturou a každý prvek v síti může mít rozdílný operační systém. V takovém případě bychom museli pořídit spoustu hardwaru pro testování a strávili bychom spoustu času nastavováním testovaného prostředí.

Uvedený problém nám částečně řeší virtualizace. Dovoluje nám na jednom fyzickém počítači vytvořit další virtuální počítače a tak efektivněji využít daný hardware. Co nám samotná virtualizace neřeší, je testování komplexních prostředí. K vyřešení tohoto problému lze využít vnořené virtualizace. Ta nám dovoluje vytvářet virtuální počítače na virtuálních počítačích, a tudíž modelovat komplexní prostředí.

1.2 Obsah kapitol

Kapitola 2 se zabývá vysvětlením některých základních pojmů z oboru virtualizace a serverové hierarchie, pro lepší orientaci v problematice. Část 3 představuje několik nejrozšířenějších virtualizačních řešení. Nejpodrobněji je představeno řešení od společnosti Microsoft - Hyper-V, které je použito pro řešení této bakalářské práce. Následující kapitola 4 se zabývá analýzou a specifikací požadavků pro správnou funkčnost vytvořené aplikace a jí vytvořených prostředí. V kapitole 5 si podrobněji rozebereme návrh architektury aplikace. Kapitola 6 nás seznámí s konkrétním řešením problémů této práce z pohledu kódu. Další kapitola 7 pojednává o průběhu testování práce. V závěru práce 8 zhodnotím dosažené výsledky mé práce a proberu možné vylepšení, či rozšíření vzniklé aplikace.

1.3 Cíl práce

Výsledkem práce bude aplikace, která umožní vytváření komplexních prostředí pomocí vnořené virtualizace. Dále bude tato aplikace umožňovat importování a exportování vytvořených konfigurací prostředí, pro jejich jednoduchou znovupoužitelnost a přenositelnost.

Kapitola 2

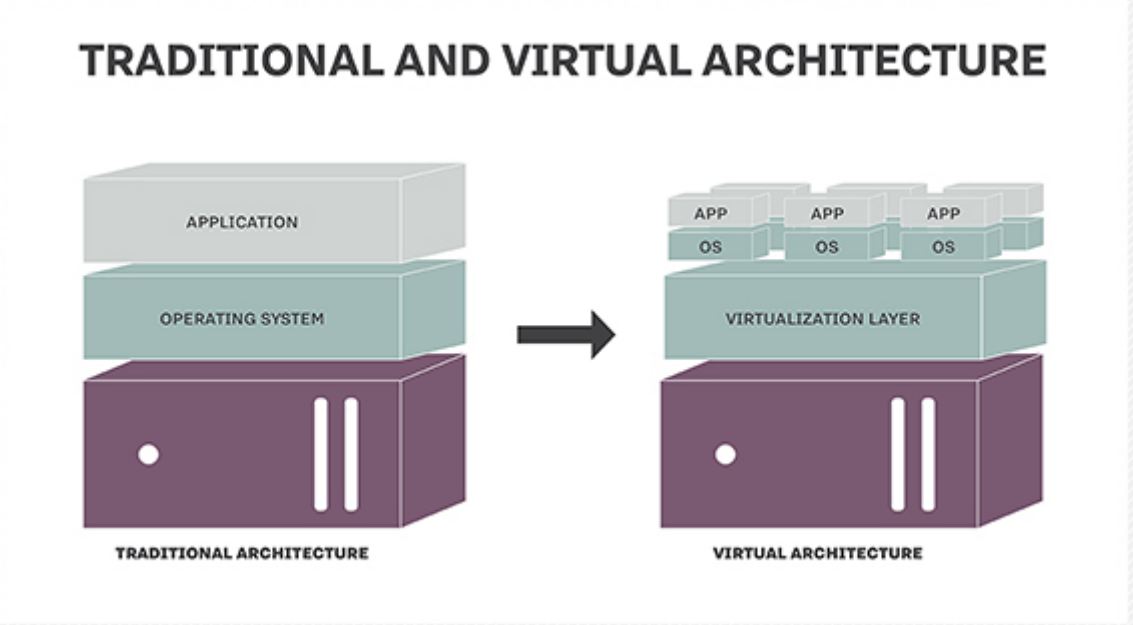
Základní pojmy

V této kapitole si pro usnadnění orientace v problematice vysvětlíme několik základních pojmů spojených s virtualizací a virtualizační hierarchií.

2.1 Virtualizace

Původní význam termínu virtualizace, pocházející z 60. let 20. století, je vytváření virtuálních strojů za pomoci kombinace hardwaru a softwaru.

Virtualizace je prováděna na dané hardwarové platformě pomocí softwaru hostitele (řídící program), který vytváří simulované prostředí počítače (virtuální stroj) pro hostovaný software. Software hosta, což často bývá celý operační systém, běží, jako by byl nainstalován na samostatné hardwarové platformě. Typicky je simulováno více takových virtuálních strojů na jednom fyzickém stroji. Pro správnou funkci hosta je třeba, aby simulace byla dostatečně robustní, aby podporovala všechna vnější rozhraní hostovaného systému, což (vzhledem k druhu virtualizace) může zahrnovat ovladače hardwaru. [32]



Obrázek 2.1: Porovnání tradiční a virtuální architektury (převzato z [23])

2.1.1 Druhy virtualizace

Plná (nativní) virtualizace

Jak napovídá název, jedná se o virtualizaci, kde virtualizujeme veškeré části počítače. Občas se tento druh virtualizace označuje jako nativní. Při tomto druhu virtualizace běží nemodifikovaný hostovaný operační systém ve virtuálním prostředí a není si vědom, že běží pouze ve virtualizovaném prostředí. Operační systém nemá přístup k fyzickému hardwaru. Veškeré aplikace v takto vytvořeném virtuálním stroji jsou klasické, bez úprav. Přístup operačního systému, či aplikací k fyzickým prostředkům je vždy zprostředkován. [15]

Pokud se hostovaný systém pokusí přistoupit k hardwaru, jeho požadavek odchytí hypervizor, upraví ho (např. čtení z virtuálního disku je třeba převést na čtení určitého místa na disku fyzickém) a následně předá ke zpracování hostitelskému systému. To zapříčiňuje, že veškeré vstupně-výstupní operace sebou přináší značné režijní náklady. [14]

Paravirtualizace

Tento druh se velmi podobá plné virtualizaci s tím rozdílem, že se nevytváří kompletní virtuální hardware, ale je povoleno jádru virtualizovaného operačního systému komunikovat přímo s hardwarem fyzickým. Tato komunikace probíhá přes speciální programové rozhraní upraveného jádra hostitelského stroje. Aby komunikace přes programové rozhraní byla úspěšná, je nutno modifikovat jádro hostovaného stroje. Jednotlivá systémová volání jsou uskutečňována pomocí hypervizoru. [1]

Emulace (Simulace)

Tento druh virtualizace vznikl z potřeby provozovat na jednom stroji, stroj rozdílných parametrů. Využívá se například v situacích, kdy se vyvíjí software pro procesory, které nejsou v dnešní době běžné. Pomocí emulace lze vytvořit víceprocesorový stroj na jednoprocesorovém a podobně.

Při emulaci běží aplikace a hostovaný operační systém bez dalších modifikací. Emulace je vytvořena pouze pomocí softwaru. Z toho vyplývá, že nemá hardwarovou podporu. Kvůli tomu má emulace velkou provozní režii. [15]

Virtualizace OS

Tento druh virtualizace patří spolu s paravirtualizací k nejefektivnějším formám virtualizace. Virtualizovaná prostředí běží nad společným jádrem, které má přímý přístup k fyzickému hardwaru. Režijní ztráty způsobuje pouze oddělení procesů, diskových prostorů a síťového provozu serverů. V rámci této technologie lze provozovat více instancí identického operačního systému, který je určen pro stejnou architekturu jako hostitelský stroj.

Virtualizační vrstva se nachází mezi operačním systémem hostitelského stroje a jednotlivými virtuálními servery. Tato virtualizační vrstva obsahuje jednotlivé virtuální stroje. Virtuální servery vlastně sdílejí jeden operační systém s hostitelským strojem. Tento hostitelský operační systém má přímý přístup k fyzickému hardwaru, tudíž není potřeba vytvářet virtuální zařízení ani žádné další programové rozhraní. [1]

2.1.2 Typy virtualizace

Serverová virtualizace

Serverovou virtualizací se označuje technika, kdy se před uživatelem serveru zamaskují serverové zdroje jako například počet a identita individuálních fyzických serverů a procesorů. Serverový administrátor používá software k rozdělení jednoho fyzického serveru na několik izolovaných virtuálních prostředí. Tato virtuální prostředí jsou někdy nazývána virtuální privátní servery, ale také se můžeme setkat s označeními jako host, instance, kontejner a emulace. [21]

Virtualizace datových úložišť

Virtualizace datových úložišť je proces seskupování několika fyzických úložišť tak, aby toto seskupení vypadalo jako jedno datové úložiště. Pomáhá administrátorovi, snadněji a za kratší dobu, provádět úkony jako jsou zálohování, archivace a obnovování tím, že maskuje skutečnou komplexitu SAN[1]. [20]

Síťová virtualizace

Síťová virtualizace odkazuje na správu a monitorování celé počítačové sítě, jako by se jednalo o jedinou administrativní entitu, z jedné softwarové administrátorské konzole. Síťová virtualizace může také zahrnovat virtualizaci úložišť. Je navrhnuta tak, aby umožňovala síťovou optimalizaci přenosové rychlosti dat, flexibility, škálovatelnosti, spolehlivosti a bezpečnosti. Automatizuje mnoho síťových administračních úkolů, což má za následek možné maskování skutečné komplexity sítě. V podstatě je možné sloučit všechny servery a služby do jediného zdroje prostředků, který může být dále použit bez ohledu na fyzické části. [29]

2.2 Virtuální stroj

Virtuální stroj (dále jen VM) je počítačový program nebo operační systém, který nejenže vykazuje chování jako samostatný počítač, ale je také schopen plnit úkoly, jako je spouštění aplikací a programů, jako samostatný počítač.

VM, obvykle známý také jako host (guest VM), je vytvořen v rámci jiného počítačového prostředí, takzvaného hostitele. V jeden okamžik může existovat několik hostů na jednom hostiteli. [30]

2.3 Hostitel (host VM)

Hostitelský virtuální stroj je serverová část VM, která poskytuje výpočetní zdroje konkrétnímu hostu. Hostitelský VM může existovat jako součást zdrojů jednoho fyzického stroje, nebo jako součást zdrojů několika fyzických strojů.

2.4 Hypervizor

Termín hypervizor byl poprvé použit v roce 1956 firmou IBM, která tak označovala program, který umožňoval sdílet paměť počítače, na němž byl hypervizor nainstalován. [28]

[1]SAN (Storage-area network) je dedikovaná vysokorychlostní síť, která propojuje a představuje sdílené zdroje úložných zařízení několika serverům. [22]

Hypervizor, někdy také nazývaný správce virtuálních strojů, je tedy název jedné z metod virtualizace hardwaru počítače, která umožňuje běh několika hostovaných VM na jediném hostiteli zároveň. Hostované VM sdílejí hardware hostitelského počítače tak, jakoby každý VM vlastnil vlastní procesor, paměť a další hardwarové prostředky. [13]

Hypervizory se tradičně rozdělují do dvou typů [2].

2.4.1 Type-1, nativní nebo bare-metal hypervizor

Tento druh hypervizoru se instaluje přímo na počítač uživatele. Nepoužívá se žádný další hostitelský operační systém a tak má hypervizor přímý přístup ke všem hardwarovým zdrojům a vlastnostem.

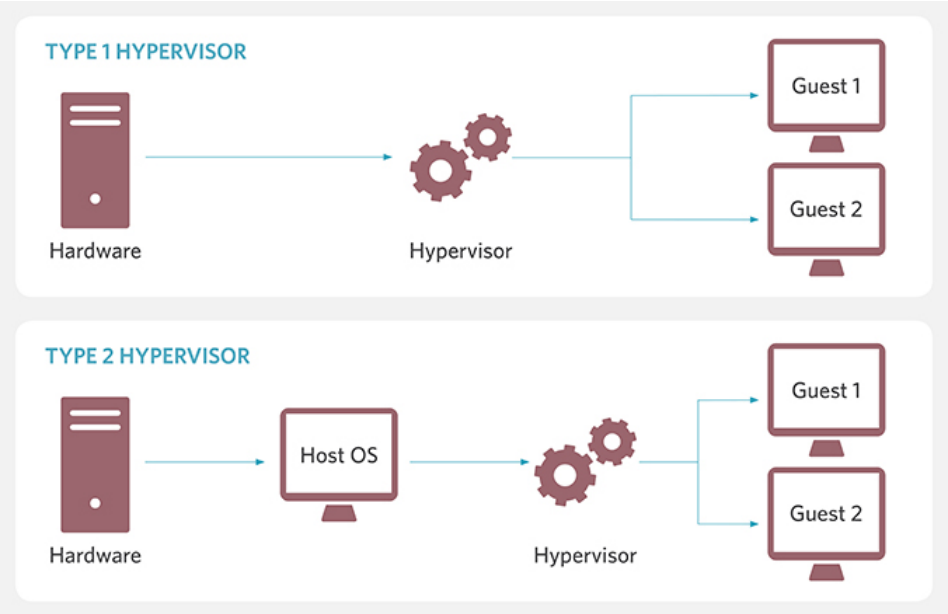
Hlavní důvod pro použití prvního typu je to, že můžeme provozovat na jednom počítači několik operačních systémů bez provozních nákladů na hostitelský operační systém. Další výhodou tohoto řešení je přenositelnost a abstrakce hardwaru.

Bare-metal hypervizor je nejčastěji používán jako serverové řešení, kvůli jeho bezpečnosti a přenositelnosti z jednoho hardwaru na druhý v případě poruchy. [13]

2.4.2 Type-2, hostovaný hypervizor

Hostovaný hypervizor je to, co si většina lidí představí pod výrazem virtualizace operačního systému. Tento hypervizor potřebuje hostitelský operační systém a je často chápán jako nainstalovaný software uvnitř hostitelského operačního systému.

Druhý typ stále dokáže provozovat více operačních systémů najednou, ale nemá přímý přístup k hardwarovým zdrojům a tak při běhu hosta vznikají určité provozní náklady. To znamená, že hostovaný operační systém nevyužije svůj plný potenciál a v případě nehody ztratíme přístup k hostovanému operačnímu systému. [13]



Obrázek 2.2: Hypervizor typu 1 a typu 2 (převzato z [24])

[2] V [17] se autoři zmiňují ještě o třetím typu - hybridním. V tomto typu funguje hypervizor na stejné úrovni jako operační systém. Tuto architekturu používal například předchůdce Hyper-V, Microsoft Virtual Server 2005.

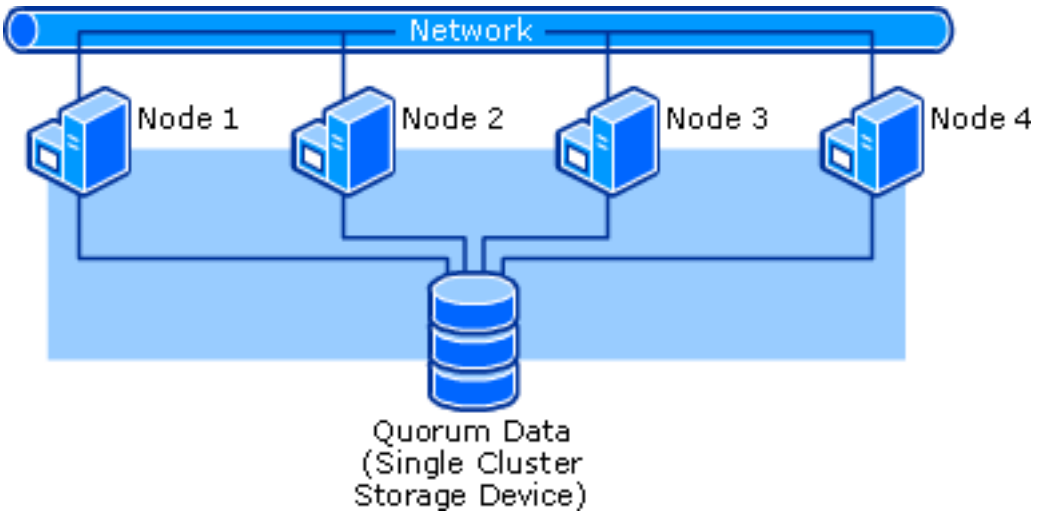
2.5 Vnořená virtualizace

Vnořená virtualizace je pojem, kterým se označuje virtualizace, která již běží ve virtualizovaném prostředí. Jinými slovy to je vlastnost, která umožňuje běh hypervizoru uvnitř VM, který sám běží pod hypervizorem.

S vnořenou virtualizací do sebe lze účinně vnořovat hypervizory, kteří se podle jejich vnoření postupně označují úrovněmi. Hypervizor, který spravuje hlavní VM se považuje za úroveň 0 nebo se označuje L0 hypervizor. Hypervizor, který běží ve VM L0 hypervizoru, se označuje jako úroveň 1 nebo L1 hypervizor. Další vnořený hypervizor by byl úroveň 2 a tak dále. [25]

2.6 Server cluster

Serverový cluster je skupina nezávislých serverů, také nazývaných uzlů (node), které spolu pracují jako osamocený systém, poskytující vysokou dostupnost (high availability) služeb pro klienty. Pokud se objeví selhání jednoho uzlu v cluster, jeho zdroje jsou přesměrovány a pracovní zátěž je přerozdělena na jiný uzel v cluster. Proto se serverové clusters používají pokud je potřeba zajistit uživatelům neustálý přístup k důležitým službám. [26]



Obrázek 2.3: Jednoprůchodové zařízení cluster (převzato z [26])

2.7 Datastore

Datastore je úložiště pro ukládání, spravování a rozdělování datových sad na podnikové úrovni. Může zahrnovat data od databázových aplikací koncových uživatelů, složky, nebo dokumenty. Může být strukturovaný, nestrukturovaný, v jiném elektronickém formátu.
V závislosti na organizaci může být datastore klasifikované jako specifické pro aplikaci (application-specific), provozní (operational), nebo centralizované (centralized). Dále může být také navrhnuto a vytvořeno pomocí účelově vytvořeného softwaru, nebo pomocí typických databázových aplikací. [27]

Kapitola 3

Přehled nejpoužívanějších hypervizorů

V této kapitole si představíme tři hypervizory. Zvláštní pozornost pak věnujeme hypervizoru Hyper-V, který budu používat při řešení této bakalářské práce.

3.1 VMware ESXi

VMware ESXi je počítačový virtualizační software vyvíjený společností VMware Inc. Jedná se o hypervizor 1. typu. Je navržen jako tzv. bare metal embedded hypervizor, což znamená, že běží přímo na hardwaru počítače a nepotřebuje žádný další základní operační systém. ESXi místo toho obsahuje a integruje životně důležité části operačního systému, jako je například jádro operačního systému. [9]

3.2 VMware Workstation Player

VMware Workstation Player, dříve známý jako VMware Player, je bezplatný virtualizační softwarový balíček pro platformu x64, na které běží Windows, nebo Linux. Jedná se tudíž o hypervizor typu-2. VMware Player dokáže jak spustit existující virtuální zařízení, tak vytvářet nové virtuální stroje. [5]

3.3 KVM

KVM (Kernel-based Virtual Machine) je open source virtualizační architektura pro Linuxové distribuce, která mění Linuxové jádro v hypervizor 1. typu. Byl původně vyvíjen firmou Qumranet, později pak odkoupen roku 2008 firmou Red Hat.

Tento software patří do skupiny plné virtualizace. Díky podpoře hardwarové virtualizace Intel VT-x a AMD-V je možné spouštět nemodifikované virtuální operační systémy napříč platformami. [16]

3.4 Oracle VM VirtualBox

VirtualBox byl původně nabízen německou společností Innotek GmbH. V roce 2008 tuto firmu odkoupila společnost Sun Microsystems, která byla v roce 2010 odkoupena firmou Oracle Corporation. Tato firma přejmenovala původní název produktu na její nynější tvar - Oracle VM VirtualBox.

Oracle VM VirtualBox je bezplatný open-source hypervizor, který může být nainstalovaný na několik operačních systémů: Linux, macOS, Windows, Solaris a OpenSolaris. Existují také modifikace pro FreeBSD a Genode. Jedná se tak o hypervizor typu-2. [8]

3.5 Red Hat Virtualization (RHV)

Red Hat Virtualization (RHV) je podnikový virtualizační produkt vyvíjený společností Red Hat. Vychází z KVM hypervizoru a používá SPICE[1] protokol a VDSM (Virtual Desktop Server Manager) s centralizovaným serverem pro správu, založeném na RHEL[2] Dále také dokáže obdržet informace o uživateli nebo skupině z Active Directory nebo FreeIPA Active Directory emulátoru. [3]

3.6 Xen

Xen vznikl jako výzkumný projekt na univerzitě Cambridge pod vedením odborného asistenta Iana Pratta. Později Ian Pratt založil společnost XenSources, Inc., která se v dnešní době stará o vývoj a podporu tohoto volně šiřitelného projektu. Xen byl původně vyvíjen jako paravirtualizační technika, ale díky uvedení hardwarové podpory virtualizace (VT-x, AMD-V) lze Xen od verze 3.0 provozovat v plně virtualizovaném režimu. To znamená, že odpadá nutnost provozovat speciálně upravené virtualizované operační systémy. [33]

3.7 Hyper-V

Hyper-V je virtualizační řešení od firmy Microsoft. Aktuální verze Hyper-V se vyskytuje v Windows 10 a Windows Server 2016.

3.7.1 Historie

Hyper-V původně vychází z produktů Virtual PC firmy Connectix a Virtual Server. Virtual PC bylo navrženo v 90. letech a následně pak vydáno v únoru 2003 firmou Microsoft za účelem vytváření VM na hardwarové platformě x86.

Virtual PC pro Windows poskytoval desktop uživatelům nástroje pro migraci na Windows XP, nebo Windows 2000 Professional, podporu pro legacy aplikace a umožnil řadu dalších využití ve sférách vývoje aplikací, call centrech, technické podpory, vzdělávání a školení.

Virtual Server odpovídal na požadavek zákazníků, kteří chtěli řešení migrace aplikací, založené na virtualizaci a podporované firmou Microsoft. Dále poskytoval významnou efek-

[1] SPICE (Simple Protocol for Independent Computing Environments) je systém vzdáleného zobrazení vytvořený pro virtuální prostředí. Umožňuje uživateli vidět své výpočetní počítačové prostředí odkudkoli v internetu.[4]

[2] RHEL (Red Hat Enterprise Linux) je distribuce Linuxu vyvíjena společností Red Hat, která cílí na komerční použití [7].

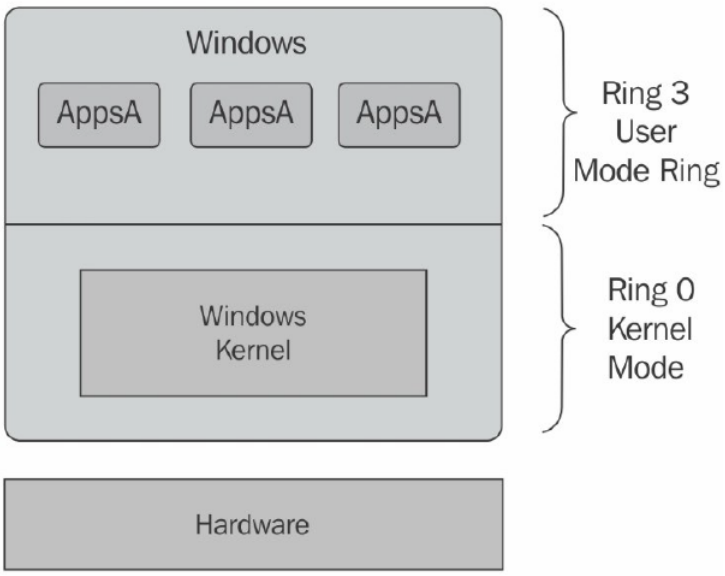
tivnost pomocí konsolidace několika Windows NT 4.0 serverů a jejich aplikací do jediného Windows Server systému.

Virtual PC i Virtual Server jsou hypervizor typu 2. Tyto virtualizační platformy obsahovaly několik limitací, nakonec zastaraly a byly nahrazeny Hyper-V. [37]

3.7.2 Architektura

Windows bez Hyper-V

Předtím, než se ponoříme do architektury Hyper-V si na obrázku 3.1 pro snazší pochopení ukážeme, jak vypadá operační systém Windows bez Hyper-V. V normální instalaci Windows, jsou instrukce rozděleny do čtyř přístupových úrovní, takzvaných Kruhů (Rings). Kruh s nejvyššími přístupovými právy se nazývá Kruh 0. Může přímo přistupovat k hardwaru a také tam, kde sídlí Windows Kernel. Kruh 3 je zodpovědný za hostování uživatelské úrovně, kde běží většina běžných aplikací a má nejmenší přístupové oprávnění. [17]

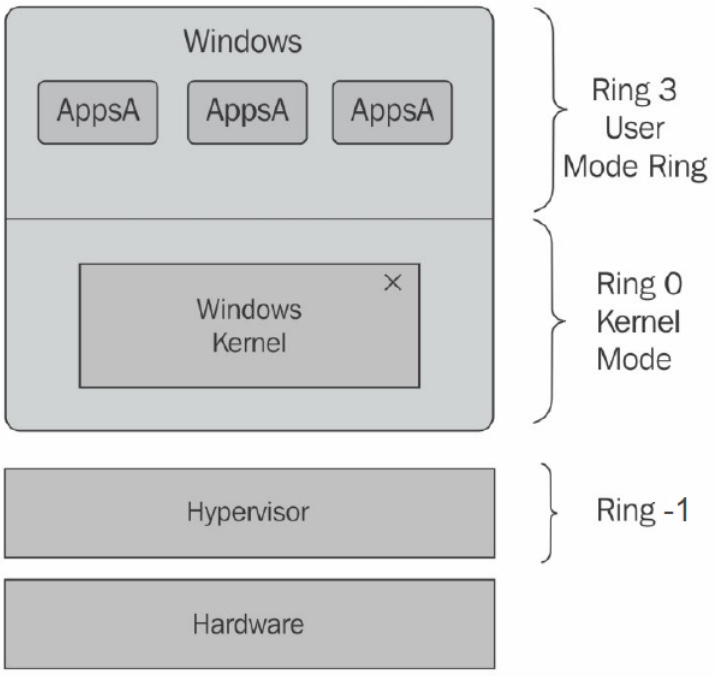


Obrázek 3.1: Windows bez Hyper-V (převzato z [17])

Windows s Hyper-V

Poté co je Hyper-V nainstalováno, potřebuje vyšší oprávnění než Kruh 0 a také vyhrazený přístup k hardwaru. Toto je umožněno díky vlastnostem nových procesorů od firmy Intel a AMD, zvaných Intel-VT a AMD-V. Tyto technologie umožňují vytvoření pátého Kruhu, zvaného Kruh -1. Hyper-V využívá tohoto Kruhu k umístění jeho hypervizoru. Tímto způsobem dostane vyšší přístupová práva, běží pod Kruhem 0 a kontroluje všechen přístup k fyzickým komponentám tak, jak je ukázáno na obrázku 3.2.

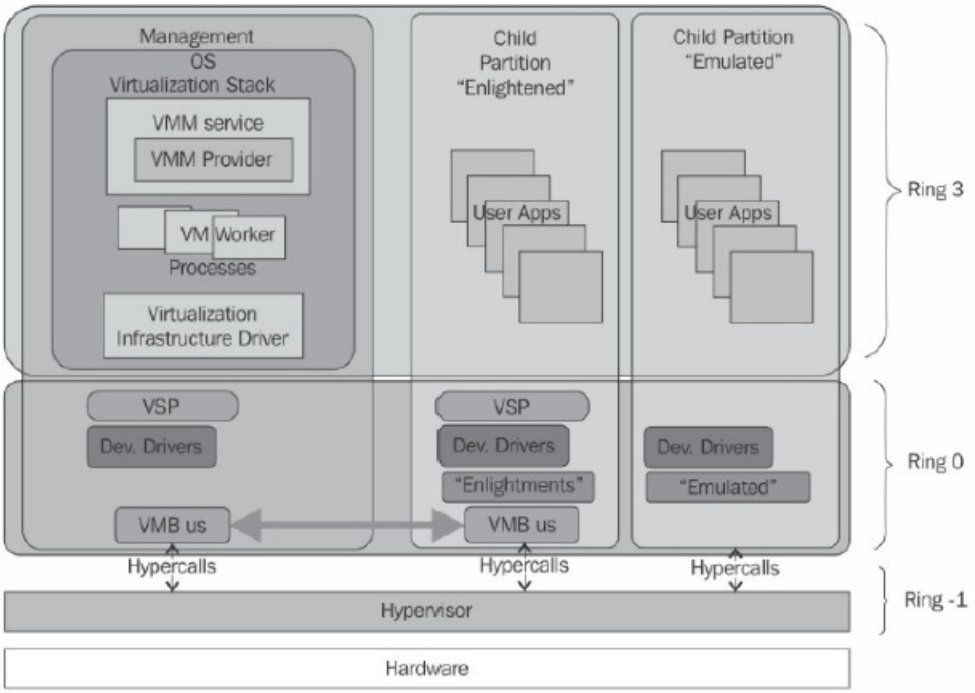
Po nainstalování Hyper-V zaznamená architektura operačního systému několik změn. Hned po prvním načtení, `Boot Loader Operačního Systému` (soubor winload.exe) zkontroluje použitý procesor a načte obraz hypervizoru na Kruh -1. Poté je spuštěn Windows Server, který běží nad hypervizorem. Následně běží každé další VM vedle Windows Serveru a mají stejná přístupová práva. Windows Server je zodpovědný za správu VMs, k čemuž využívá různé komponenty. [17]



Obrázek 3.2: Windows s nainstalovaným Hyper-V (převzato z [17])

Architektura komponent

Hyper-V má mnoho komponent, které jsou zodpovědné za poskytování řešení end-to-end správy pro VMs a správu operačních systémů. Na obrázku 3.3 vidíme nejdůležitější komponenty Hyper-V. [17]



Obrázek 3.3: Nejdůležitější komponenty Hyper-V (převzato z [17])

Hypervizor

Hypervizor jsme si už popsali v předchozí kapitole 2.4. Malý Hyper-V hypervizor (asi jen 20 MB) je zodpovědný za správu, oddělení a kontrolování přístupu oddílů (partition). Také dohlíží nad izolací jednotlivých oddílů s vysokou bezpečností a spolehlivostí. [17]

Oddíly (Partitions)

Když je Hyper-V přítomno v počítači, hostitelský operační systém a VMs běží se stejnými přístupovými právy a oboje nazýváme oddíly (partitions). Je ale nutno dodat, že hostitelský operační systém ovládá několik komponent pro správu VMs a proto se mu říká nadřazený oddíl (parent partition), nebo správce operačních systémů (management OS) a VMs jsou označovány jako podřízené oddíly (child partitions), nebo hostující operační systémy (guests OS). [17]

Virtualizační zásobník (Virtualization stack)

Vytváření a správa VMs je realizováno několika virtuálními zařízeními a softwarovými komponentami, které nazýváme virtualizační zásobník. Nachází se v nadřazeném oddílu a pracuje ve spojení s hypervizorem.

Virtualizační poskytovatel služeb (Virtualization Service Provider (VSP)) je softwarová komponenta, která kontroluje I/O požadavky jménem VMs v nadřazeném oddílu. **Sběrnice Virtuálního stroje** (Virtual Machine Bus (VMBus)) je zodpovědná za přenos dat, služby a dodání zařízení mezi nadřazeným a podřízenými oddíly přes vyhrazený kanál, dostupný mezi VSPs a **Klienty virtualizačních služeb** (Virtualization Service Clients (VCSs)). VSP používá VMBus ke komunikaci s podřízenými oddíly a používá VSCs k poskytování umělých ovladačů, které běží v podřízených oddílech.

Pro každé běžící VM je vytvořen pracovní proces v nadřazeném oddílu. Pracovní proces a **Správní služba virtuálního stroje** (Virtual Machine Management Service (VMMS)) jsou komponenty uživatelského režimu, které nadřazenému oddílu poskytují schopnost vytvořit, spustit, zastavit, uložit a vymazat VMs. Všechny tyto úkoly jsou řízeny **Ovladačem virtuální infrastruktury** (Virtual Infrastructure Driver (VID)), který spravuje komunikaci mezi nadřazeným a podřízenými oddíly. [17]

Enlightened (vysoký výkon) proti emulated (nízký výkon)

Přístup mezi oddíly a hypervizorem je realizován přes speciální rozhraní zvané **Hypervolání** (Hypercalls). Zaručují, že VMs můžou přistupovat k hardwaru pomocí komponent jako jsou - VID, VMBus, VSCs a VSPs. Tyto mechanismy jsou dostupné během instalace **Integrace komponent** (Integration Components (ICs)). Některé operační systémy mají balíčky s integračními komponentami již nainstalovány v jejich jádru. VMs, které mají tyto komponenty se nazývají Enlightened VMs. Pro staré, nebo nepodporované operační systémy přerušuje nadřazený oddíl VM komunikaci, aby mohl emulovat Hypervolání. Správce operačního systému tak musí pracovat jako můstek mezi VM a hardwarem. To má za následek nízkou výkonnost a omezení při přístupu k hardwaru. [17]

Kapitola 4

Analýza a specifikace požadavků

V této kapitole analyzujeme a specifikujeme softwarové a hardwarové požadavky pro správné fungování vyvíjené aplikace a jí vytvořených prostředí.

4.1 Analýza požadavků

Požadavky můžeme specifikovat do dvou kategorií.

4.1.1 Hardwarové požadavky

Hardwarové požadavky, jsou to požadavky na fyzické zdroje počítače, na kterém naše aplikace běží. Jedná se hlavně o velikost operační paměti (RAM) a počet jader procesorů. Neméně důležitá je však podpora technologií, které jsou potřeba k zprovoznění jak samotné virtualizace, tak virtualizace vnořené. V neposlední řadě je také důležitá velikost úložiště. Nároky na hardwarové zdroje se mění s počtem vytvořených virtuálních strojů. Při nedostatečné kapacitě zdrojů riskujeme pomalý, nebo dokonce nefunkční stav vytvořeného prostředí.

4.1.2 Softwarové požadavky

Softwarové požadavky se zabývají hlavně podporovanými operačními systémy. Také se zabývají rozdíly v podporovaných funkcích mezi Hyper-V na Windows 10 a Windows Server 2016.

4.2 Specifikace požadavků

Hardwarové požadavky

Seznam nezbytných hardwarových požadavků hostitelského počítače [10]

- 64-bitový procesor s Second Level Address Translation (SLAT)
- Intel procesor s podporou technologií Extended Page Tables (EPT) a VT-x
- Minimálně 4 GB operační paměti
- V BIOSu povolit Hardware-enforced Data Execution Prevention (DEP)

Seznam nezbytných hardwarových požadavků pro VM [31]

- Minimálně 512 MB operační paměti
- Velikost přiděleného úložiště 32 GB

Softwarové požadavky

Seznam softwarových požadavků pro vnořenou virtualizaci [11]

- Jak host tak hostitel musí být Hyper-V Windows Server 2016, nebo Windows 10 Anniversary Update či pozdější
- VM musí být nakonfigurováno na verzi 8.0 nebo vyšší
- Powershell 3.0+

Kapitola 5

Architektura aplikace

Tato kapitola se bude zabývat návrhem architektury aplikace, která je výsledkem této bakalářské práce. Při návrhu aplikace se kladl důraz na použití moderních softwarových architektur. Na základě těchto architektur byly poté vytvořeny vizualizace, které usnadňují orientaci v architektuře a určují jakým způsobem bude probíhat implementace softwaru.

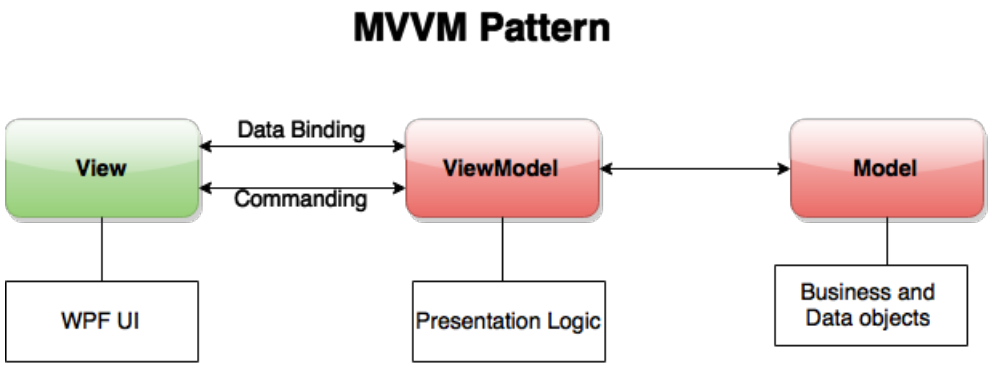
Pro vizualizaci návrhu architektur se v dnešní době nejhojněji používá univerzální jazyk pro vizuální modelování systémů, UML[1].

Při návrhu architektury byly použity principy a zásady moderního architektonického stylu Model-View-ViewModel, zkráceně MVVM. Cílem tohoto stylu je umožnit čisté oddělení zodpovědností mezi uživatelským rozhraním a logikou systému. Toto oddělení má několik výhod: Jednotlivé komponenty lze snadno nahrazovat, při změně vnitřní implementace komponent nehrozí změna funkčnosti jiných komponent, komponenty pracují nezávisle na sobě a komponenty lze snadněji testovat. [18] MVVM styl používá tři hlavní komponenty: model, view a view model. Jak vidíme na obrázku 5.1 Každá z těchto komponent odpovídá za jinou část systému.

- View - Tato komponenta zodpovídá za definování struktury, rozvržení a vzhledu toho, co uživatel vidí na obrazovce. V ideálním případě by view nemělo obsahovat žádnou obchodní logiku.
- Model - V MVVM architektuře je model implementací aplikačního doménového modelu, který zahrnuje datový model s obchodní a validační logikou.
- View model - Komponenta view model se chová jako prostředník mezi view a modelem a je odpovědná za obsluhu zobrazovací logiky. Tradičně interaguje s modelem pomocí volání funkcí v třídách modelu. Následně pak prezentuje data z modelu tak, aby je mohl view snadno používat.

V následujících podkapitolách popisuji jednotlivé UML diagramy.

[1] UML (Unified Modeling Language) [34]



Obrázek 5.1: MVVM pattern (převzato z [12])

5.1 Diagram případů užití

Tento diagram se při vývoji softwaru používá pro vyobrazení chování systému z hlediska uživatele. Zobrazuje možnosti uživatele jak se systémem interagovat. Jak vidíme na obrázku 5.2, tato aplikace nerozlišuje uživatele z hlediska rozdílnosti práv, kterými uživatel disponuje. Tudíž existuje pouze jeden typ uživatelů. Tento uživatel může provádět následující úkony:

- Ukončit aplikaci
- Vytvořit nový model prostředí - Prázdné prostředí s kořenovým prvkem
- Uložit model prostředí do XML souboru - Při ukládání modelu prostředí můžeme vytvořit nový, nebo vybrat již existující XML soubor
- Nahrát model prostředí z XML souboru - Při nahrávání modelu prostředí můžeme procházet souborový adresář uživatele
- Modelovat prostředí - Při tomto úkonu uživatel může přidávat nové virtuální stroje, mazat již vytvořené virtuální stroje a upravovat atributy vytvořeným virtuálním strojům
- Vytvořit prostředí - Uživatel vytvoří namodelované prostředí

5.2 Doménový model

Tento diagram se při vývoji softwaru používá v počáteční fázi vývoje softwaru. Jedná se o formu diagramu tříd a znázorňuje základní entity systému a vztahy mezi nimi. [19]

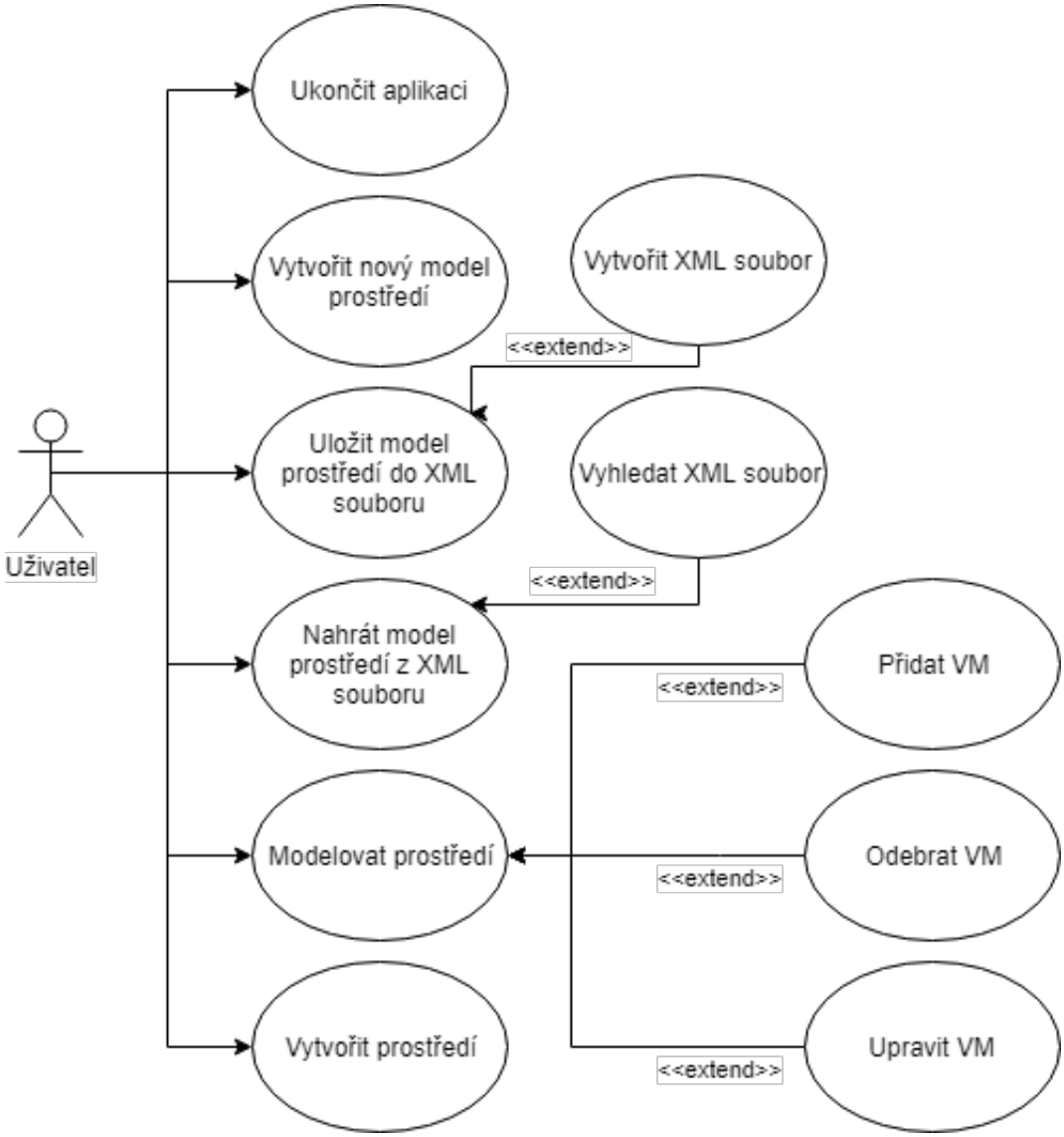
Na obrázku 5.3 vidíme, že tento systém obsahuje jedno view - MainView, které zodpovídá za zobrazení struktury navrženého prostředí a za poskytnutí informací o jednotlivých vlastnostech vybraného virtuálního stroje. Data která zobrazuje mu jsou dodávána skrze atribut DataContext. Tento atribut je provázaný s třídou VmStructureViewModel, která obsahuje stromovou strukturu virtuálních strojů v atributu VMs. Také obsahuje právě vybraný virtuální stroj, který zpřístupňuje své jednotlivé atributy. Dále obsahuje několik příkazů (commands), které manipulují buď s vybraným virtuálním strojem, nebo provádí řídící akce aplikace jako jsou například načítání a ukládání souborů. Pro vykonání těchto

příkazů volá tato třída třídy PsScripts a EnvironmentSerializer. Třída PsScripts obsahuje powershellové skripty, které je nutné vykonat pro zdárné vytvoření navrženého prostředí. Kontroluje, zda má aktuální uživatel práva správce a zda se na použitém počítači nachází nainstalované Hyper-V. Dále vytváří samotný skript pro vygenerování prostředí. Třída EnvironmentSerializer se využívá pro serializaci stromové struktury VMs pro její uložení do XML souboru. Také dokáže zpětně přeložit XML soubor na stromovou strukturu VMs. Na ukládáni informací o jednotlivých virtuálních strojích se používá třída VmModel, která obsahuje v atributu Children stromovou strukturu vnořených virtuálních strojů. Dále do svých atributů ukládá informace o názvu virtuálního stroje, velikosti přidělené operační paměti, virtuálním úložišti, verzi a generaci virtuálního stroje, zařízení z něhož se má načíst operační systém, názvu virtuálního switche. Informace o virtuálním úložišti je uložena v další třídě - VirtualHd, která ve svých atributech obsahuje informace o cestě k úložišti, velikosti úložiště a stavu úložiště, který může nabývat několika stavů, které jsou dány výčtovým typem VHdState. Informace o zařízení ze kterého se bude načítat operační systém se ukládají do třídy BootDevice, která nese údaje o cestě k danému zařízení a typu zařízení, který může nabývat hodnot obsažených ve výčtovém typu BootDeviceType.

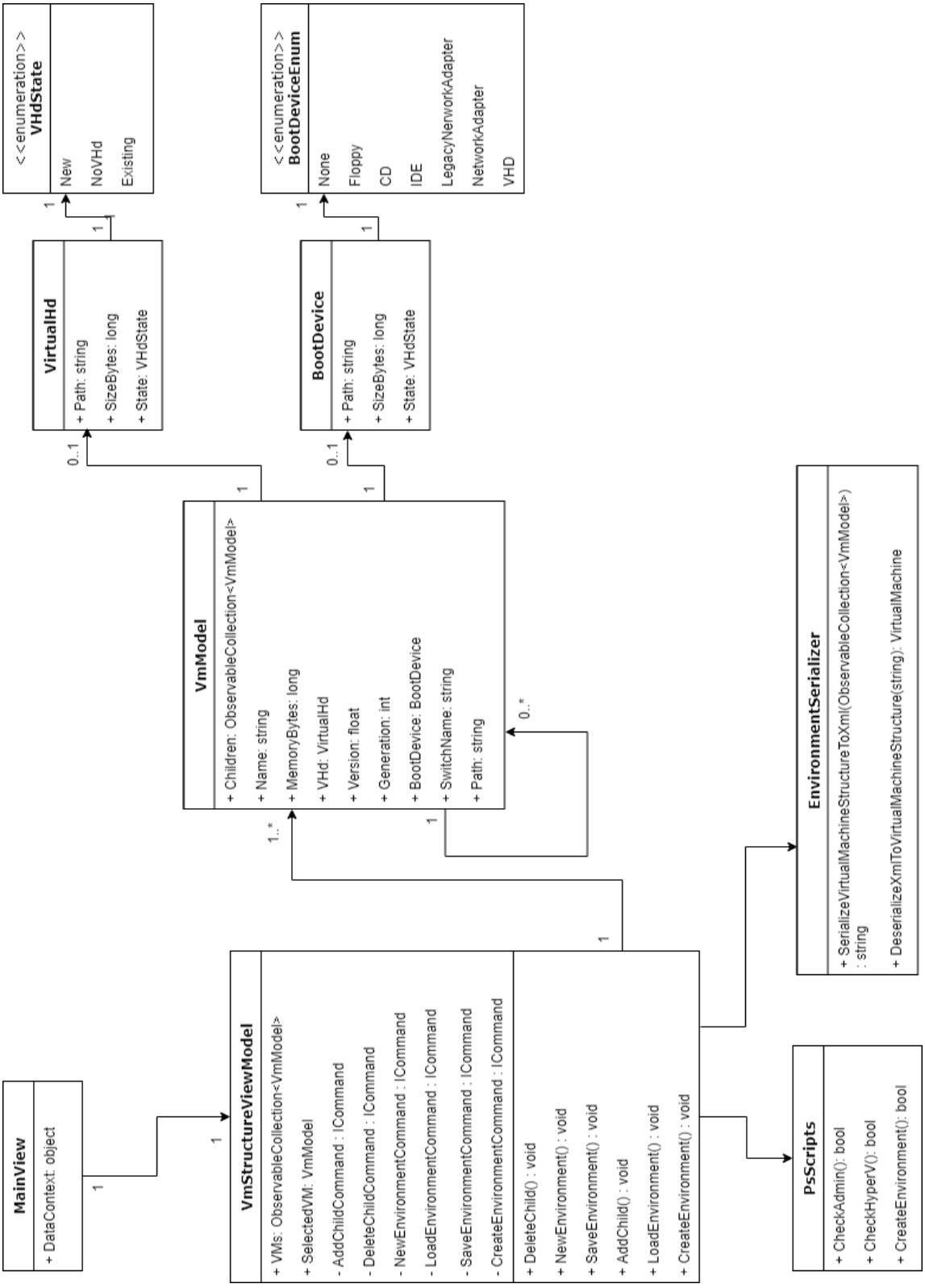
5.3 Mockup aplikace

Mockup se v softwarovým inženýrství používá hlavně pro navržení designu. Je to prototyp systému, který neobsahuje žádnou, nebo obsahuje minimální funkcionalitu. [6]

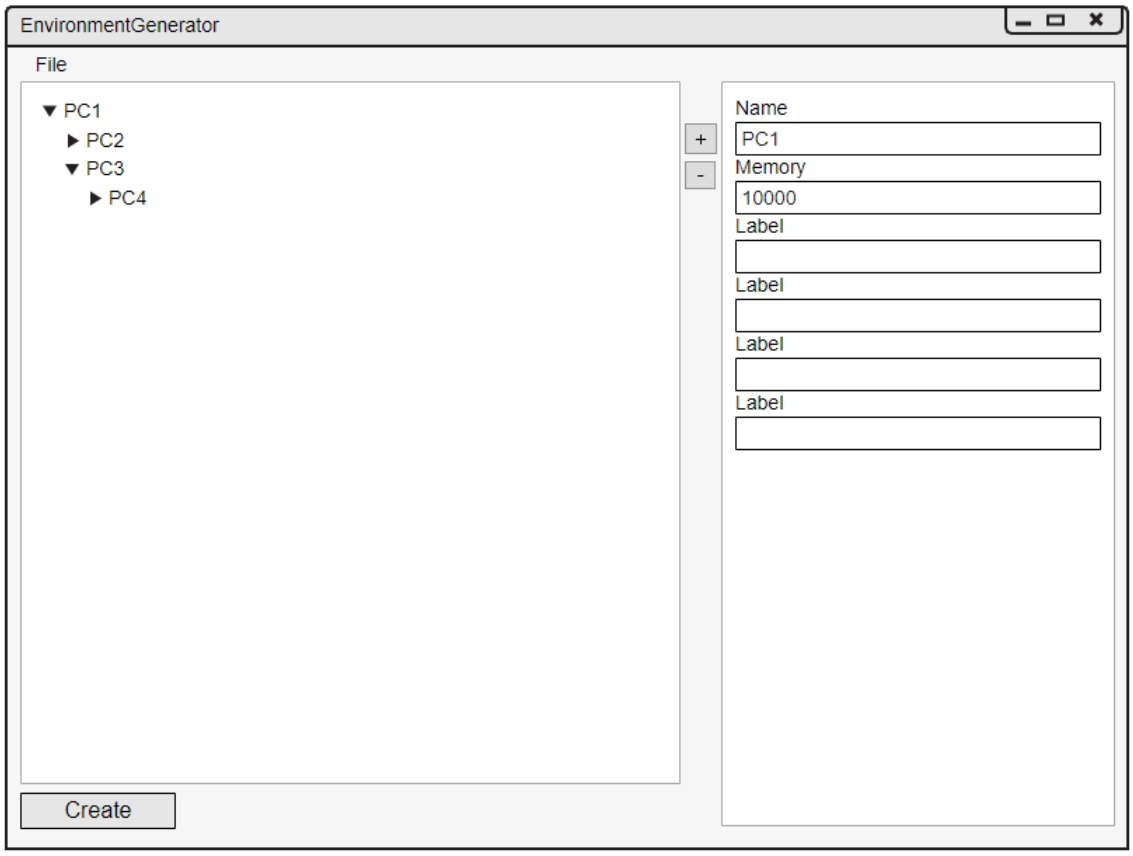
Na obrázku 5.4 vidíme okno aplikace, které se skládá z pěti komponent. První komponentou je kontextové menu v levém horním rohu. Toto menu obsahuje řídící akce aplikace. Pod kontextovým menu se nachází hlavní komponenta aplikace - okno se stromovou strukturou virtuálních strojů, která znázorňuje vytvářené prostředí. Každá položka znázorňuje jeden virtuální stroj a po označení jednoho z nich se v pravém okně zobrazí jeho atributy, které následně můžeme upravovat. Mezi těmito okny se nachází třetí komponenta, která se skládá ze dvou tlačítek, které se aktivují při výběru některého z virtuálních strojů. Tyto tlačítka slouží k přidávání a odebírání virtuálních strojů. Poslední komponentou je tlačítko v levém dolním rohu, které slouží ke spuštění skriptů pro vytvoření navrženého prostředí.



Obrázek 5.2: Diagram případů užití



Obrázek 5.3: Diagram doménového modelu



Obrázek 5.4: Mockup aplikace

Kapitola 6

Implementace

Tato kapitola pojednává o řešení implementace navržené aplikace. Pro vývoj této desktopové aplikace jsem využil platformy .NET Framework ve verzi 4.6.1 a programovací jazyk C#. Pro tento typ aplikací se velmi často využívá návrhového vzoru MVVM, který jsem se také snažil následovat.

6.1 Souborová struktura aplikace a jmenná konvence

Souborová struktura aplikace je důležitá pro dobrou orientaci v projektu na úrovni komponent a neměla by se podceňovat. Problémy se špatnou orientací v projektu se většinou dostavují až při práci na větších projektech, ale je dobrý návyk tato pravidla dodržovat i u projektů menších. Další důležitá věc, která se musí zvážit před začátkem implementace je jmenná konvence. Tato konvence nám pomáhá porozumět a orientovat se v projektu na kódové úrovni.

6.1.1 Souborová struktura

- `/Properties/` Obsahem této složky jsou lokalizační soubory, uživatelské a aplikační nastavení a třída `AssemblyInfo,cs`, která poskytuje vlastnosti pro získání informací o aplikaci.
- `/References/` Složka, ve které jsou uloženy závislosti na použité zdroje.
- `/Helpers/` Zde jsou uloženy pomocné třídy, které jsou používány v rámci volaných příkazů.
- `/Logs/` Tato složka obsahuje textový soubor, který ukládá stavové výpisy z aplikace.
- `/Models/` Obsahem této složky jsou datové struktury a výčtové typy, které jsou použity pro uložení informací o vytvářeném virtualizovaném prostředí.
- `/Scripts/` Tento adresář obsahuje powershell skripty, které vytváří virtuální stroje a nastavují jejich parametry.
- `/ViewModels/` - V této složce se nalézá podadresář `/ViewModels/Base`, kde můžeme naleznout třídu `BaseViewModel.cs`, ze které dědí ostatní třídy, které využívají vlastnosti `INotifyPropertyChanged`. V tomto podadresáři dále nalezneme třídu `RelayCommand.cs`, která implementuje interface `ICommand` a je použita pro všechny

příkazy, které v této aplikaci nalezneme. Vrátíme-li se do složky `/ViewModels/`, nalezneme tam dále třídu `VirtualMachineStructureViewModel.cs`, která poskytuje data našemu uživatelskému prostředí a obsahuje všechny příkazy, které přes toto uživatelské rozhraní ovládáme.

- `/Views/` - Tato složka obsahuje soubor `MainWindowView.xaml`, který definuje vzhled uživatelského rozhraní.
- `/App.config/` - Tento soubor obsahuje konfigurační hodnoty aplikace, které se používají pro některé vlastnosti vytvářených VM. Uživatel má přístup k modifikaci těchto hodnot.

6.1.2 Jmenná konvence

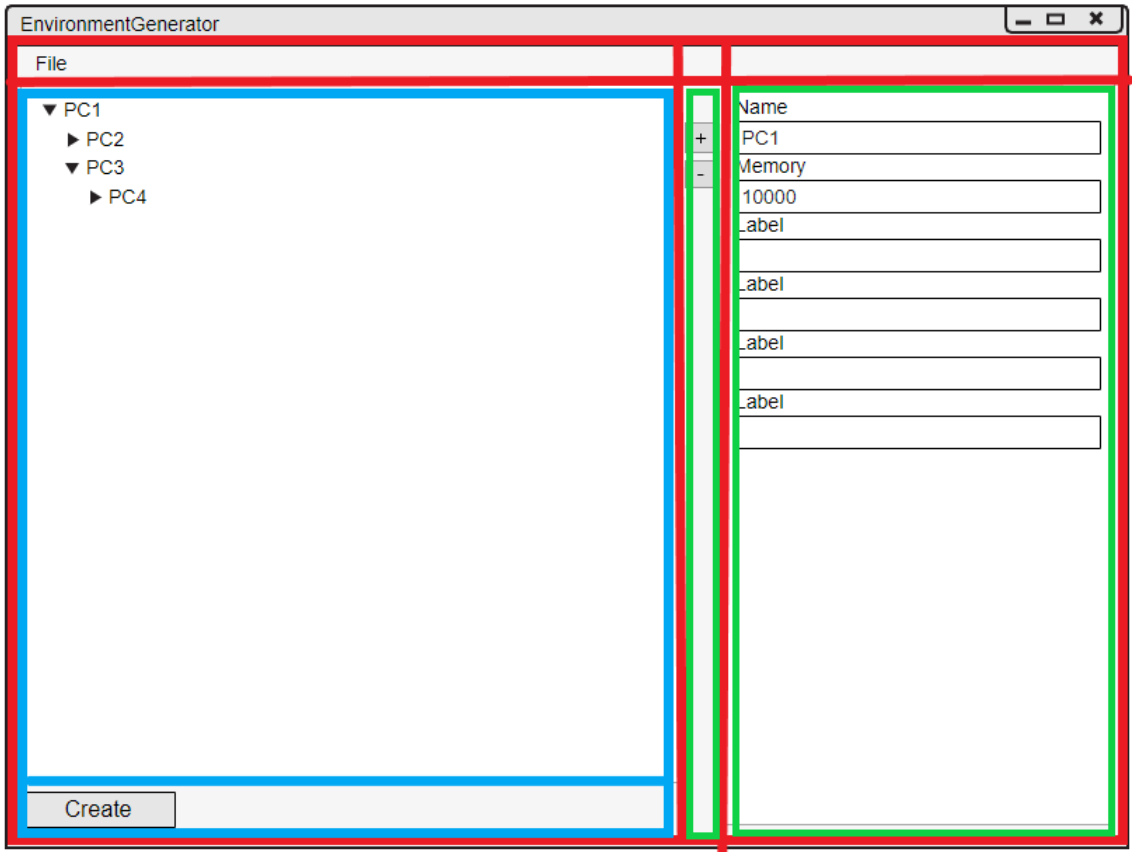
- Třídy - Název třídy začíná velkým písmenem a každé následující jednotlivé slovo začíná také velkým písmenem. Např. `NewClass`
- Pole (Fields) - privátní pole začínají podtržítkem, které je následované slovem začínajícím malým písmenem a každé další slovo začíná písmenem velkým.
  Např. `_privateField`. Veřejné pole začíná bez podtržítka, jinak je konvence stejná.
  Např. `publicField`
- Vlastnosti (Properties) - Veřejné i privátní vlastnosti dodržují stejnou konvenci jako třídy. Např. `PublicProperty`
- Proměnné (Variables) - Proměnné ctí stejné zásady pojmenování jako veřejné pole.
  Např. `publicVariable`
- Metody - Metody používají stejnou konvenci jako třídy a vlastnosti.
  Např. `PublicOrPrivateMethod()`

6.2 Grafické uživatelské rozhraní

Definice uživatelského rozhraní je uložena ve formátu XAML [36]. XAML je XML varianta od Microsoftu pro popisování uživatelských rozhraní.

Jak vidíme na obrázku 6.1, hlavní okno aplikace je rozděleno na několik částí. Pro snadné rozlišení těchto částí jsou barevně odděleny. Toto rozvržení bylo dosaženo elementem `Grid`, který nám umožňuje rozdělit okno na několik sloupců a řádků, do kterých dále můžeme vkládat další zanořené elementy. Základní rozdělení okna je znázorněno červenou barvou. Jak vidíme dělí okno na tři sloupce a dva řádky. První řádek obsahuje kontextové menu, které je vytvořeno pomocí elementu `Menu`. Druhý řádek obsahuje v levém sloupci další element `Grid` označený modrou barvou. Tento grid má jeden sloupec a dva řádky. V prvním řádku nalezneme strukturu našeho vytvářeného prostředí, která je vytvořena elementem `TreeView`. Tento element je obalen elementem `ScrollView`, který nám při přetečení obsahu zobrazí posuvník. Okolo tohoto elementu je dále `Border`, který nám vytváří rámeček. Ve druhém řádku pak nalezneme `DockPanel`, který obsahuje na levé straně tlačítko, pomocí kterého spustíme skripty pro vytvoření prostředí. Toto tlačítko je vytvořeno pomocí elementu `Button`. Vedle tohoto tlačítka se nalézá ukazatel průběhu `ProgressBar`, který nám signalizuje průběh běhu skriptu. Vedle levého sloupce nalezneme prostřední sloupec, který je označen zelenou barvou. Je vytvořený pomocí elementu `StackPanel`, který se používá

pro skládání prvků na sebe. Tento panel obsahuje pod sebe umístěná tlačítka, která jsou vytvořena pomocí elementu `Button` a slouží k přidávání a odebírání virtuálních strojů. Dále na obrázku vidíme zeleně označený pravý sloupec, který se skládá z elementu `Stackpanel`, který je obalený elementem `ScrollViewer`, okolo kterého je rámeček vytvořený pomocí elementu `Border`. Obsah tohoto sloupce je většinou tvořen dvojicí elementů `TextBlock` a `TextBox`, kde se první element chová jako označení vstupního pole, které uchovává informace o virtuálním stroji. V tomto sloupci dále můžeme najít list hodnot, který je vytvořen elementem `ComboBox` a vstupní pole pro zadání cesty k souboru nebo složce, které je vytvořeno pomocí vnořeného elementu `Grid`, který má dva sloupce a jeden řádek. V prvním sloupci se nachází textová část a druhý sloupec obsahuje tlačítko, díky kterému můžeme otevřít prohlížeč souborů.



Obrázek 6.1: Mockup aplikace s vyznačenými sekcemi

6.3 Data binding

Data binding je obecná technika, která k sobě svazuje dva datové/informační zdroje a udržuje jejich data synchronizovaná. [2] Data binding můžeme provádět několika způsoby [35]

- One time - Jednorázově stáhne data a ignoruje aktualizace na zdroji
- One way - Komunikace probíhá pouze jedním směrem (read-only přístup k datům)
- Two way - Zdroj i klient spolu komunikují, aktualizace se promítají do obou

Binding se v této aplikaci provádí v souboru `MainWindowView.xaml`, kde k požadovanému elementu uživatelského rozhraní přiřazujeme veřejně přístupnou vlastnost ze souboru `VirtualMachineStructureViewModel.cs`. Provázání mezi těmito dvěma soubory provádíme v souboru `MainWindowView.xaml.cs`, kde přiřazujeme vlastnosti `DataContext` třídu, kterou chceme s uživatelským rozhraním svázat. Na příkladu dole vidíme, že svazujeme atribut `Command` elementu `MenuItem` s vlastností `NewCommand`.

```
<MenuItem Header="_New" Command="{Binding NewCommand}"/>
```

6.4 Volání příkazů

Příkazy umožňují definovat akce na jednom místě a odkazovat na ně z kteréhokoliv místa v uživatelském rozhraní. Příkazy se v tomto projektu vytváří přes třídu `RelayCommand`, která implementuje rozhraní `ICommand`. Tato třída je pomocí veřejné vlastnosti přístupná z uživatelského rozhraní a po jejím zavolání s parametrem obsahujícím název metody, tuto metodu provede. Nutné je dodat, že aby tato třída věděla, kterou metodu provést, musíme v konstruktoru třídy `VirtualMachineStructureViewModel` přiřadit správné příkazy veřejným vlastnostem. Na příkladu dole vidíme přiřazení příkazu veřejné vlastnosti a metodu, kterou tento příkaz volá.

```
public VirtualMachineStructureViewModel()
{
   ExitCommand = new RelayCommand(Exit);
}

public ICommand ExitCommand { get; set; }

private void Exit()
{
   System.Windows.Application.Current.Shutdown();
}
```

6.5 Ukládání záznamu stavů

Pro záznam průběhu vytváření prostředí jsem vytvořil statickou třídu `Logger`, která má na starosti zaznamenávat výstupy skriptů do souboru `Log1.log`, který se nachází ve složce `Logs`. Tato třída obsahuje metodu `Log(string msg)`, která ukládá předanou zprávu do záznamového souboru. Metoda také volá kontrolu existence záznamového souboru a v případě jeho nenalezení jej vytváří.

6.6 Volání skriptů

Powershell skripty hrají v této aplikaci ústřední roli. Používají se jak pro komunikaci s hypervizorem, tak pro spojení s VM. Pro správnou funkčnost potřebují Powershell skripty naimportovat knihovnu `System.Management.Automation`. Po stisknutí tlačítka pro vytvoření virtuálního prostředí se volá asynchronní metoda, která dále volá powershell skripty s parametry získanými z uživatelského rozhraní. Na příkladu níže můžeme vidět konstrukci,

která se používá k volání skriptů. Nejprve se vytvoří nová instance powershell skriptu. Do této instance přidáme skript následovaný jeho parametry. Pokud požadujeme zřetězit dva a více souborů, zařadíme mezi ně metodu `AddStatement()`, která umožní přidání následujícího skriptu. Pro ukládání výstupů skriptů je nutné vytvořit kolekci objektů `PSObject`, kterou předáme metodě `BeginInvoke()` jako parametr. Tato metoda volá asynchronně skripty a jejich výstupy ukládá do zmíněné kolekce. Následně testujeme zda skripty proběhly v pořádku a vypisujeme do záznamového souboru stav. Po dokončení skriptů vypíšeme do záznamového souboru jejich výstup.

```
using (PowerShell pSInst = PowerShell.Create())
{
   pSInst.AddCommand(@"..\..\Scripts\.\Build-Images.ps1");
   pSInst.AddParameter("WindowsKey",ConfigurationManager.AppSettings["winKey"]);
   pSInst.AddStatement();

   pSInst.AddCommand(@"..\..\Scripts\.\NewSwitch.ps1");

   PSDataCollection<PSObject> outputColl = new PSDataCollection<PSObject>();

   IAsyncResult result = pSInst.BeginInvoke<PSObject, PSObject>(null, outputColl);

   while (result.IsCompleted == false)
   {
      Logger.Log("Generating environment");
      Thread.Sleep(10000);
   }

   foreach (PSObject outputItem in outputColl)
   {
      Logger.Log(outputItem.BaseObject.ToString());
   }
}
```

6.7 Popis skriptů

- `Build-Images.ps1` - Tento skript slouží k přípravě bázového disku, který obsahuje předinstalovaný operační systém Windows Server 2016. Tento operační systém se nainstaluje automaticky, díky souboru `unattend.xml`, který můžeme konfigurovat pomocí hodnot v souboru `App.config`. Zvláště důležité je nastavení hodnoty `winKey`, která v sobě ukládá instalační klíč pro operační systém. Vytvořený bázový disk se používá jako předloha pro rozdílové disky, které ukládají pouze změny vůči bázovému disku a tak jsou daleko menší.
- `/NewSwitch.ps1/` - Skript sloužící k vytváření nových virtuálních switchů.
- `/Copy-VMFile.ps1/` - Hlavní úloha tohoto skriptu je přenos bázového disku do vytvořených virtuálních strojů. Přenos je zprostředkován skrz hypervizor Hyper-V. Aby bylo možné tento přenos uskutečnit, je nutné povolit na obou strojích `Rozhraní služby hosta`, což má tento skript také na starosti.

- `/Prepare-VM.ps1/` - Tento skript se stará o přípravu virtuálního stroje. Pokud existuje VM se jménem stejným jako má právě vytvářené VM, skript starší VM smaže. Dále vytvoří nový rozdílový disk z disku bázového a vytvoří nové VM.
- `/Install-HyperV.ps1/` - Úkolem tohoto skriptu je připojit se na VM a nainstalovat Hyper-V s nástroji pro správu.
- `/GuestVM.ps1/` - Tento skript má za úkol se připojit na VM a v něm vytvořit vnořené VMs. K vytváření nových VM opět používá již vytvořeného bázového disku.

Kapitola 7

Testování

Tato kapitola se věnuje ověření správné funkcionality implementované aplikace. Nejprve popisuje jednotlivé typy testů a následně popisuje konkrétní testování aplikace. Aplikace byla také testována v průběhu implementace pomocí ladícího režimu (debug), díky čemuž byli odstraněny některé zásadní chyby aplikace.

7.1 Jednotkové testy

Za jednotkové testy (tzv. Unit testy) se označují testy, které testují specifické kusy kódu, zejména na úrovni metod. Používají se pro ověření zamýšlené funkcionality metod. Tyto testy se nejčastěji píší už při vývoji aplikace a také slouží pro zpětnou kontrolu při změně staršího kódu. Jedná se o tzv. White-box testy, což znamená, že programátor který testy píše zná vnitřní strukturu testované metody a tak je schopen nastavit vstupy tak, aby otestoval správné výstupy. Při implementaci aplikace bylo využito jednotkových testů zvláště pro testování třídy `VirtualMachineStructureViewModel`.

7.2 Systémové testování

Systémové testování se používá ke kontrole buď finálního produktu, nebo jeho funkčních částí. Většinou se jedná o po sobě navazující úkony systému. Např. Přihlášení do systému, následné vytvoření objednávky, vytisknutí objednávky a konečně odhlášení ze systému. Jedná se tzv. Black-box testy. Tyto testy se vytváří s ohledem na systémové požadavky. Při těchto testech se zadávají určité vstupy, které mají vést k očekávanému výstupu a tak se určí úspěch či neúspěch testu.

7.3 Testovací případy

Pro otestování aplikace jsem vytvořil několik testovacích případů. Tyto případy testují funkcionalitu systému. V prvním testovacím případu 7.1 testujeme funkcionalitu ověření oprávnění, kde se nám stav vypíše do záznamového souboru. V dalších dvou 7.2 7.3 případech testujeme import a export xml souborů vytvořeného prostředí. Poslední testy se zabývají samotnou tvorbou virtualizovaného prostředí. V testovacím případu 7.4 testujeme generování dvou virtuálních strojů na stejné úrovni. V případu 7.5 test ověřuje schopnost tvorby nového bázového disku. Tento test končí chybovou hláškou v záznamovém souboru.

V posledním uvedeném testu 7.6 prověřuji tvorbu vnořeného VM. Vnořené VM se úspěšně vytvoří, avšak bázový disk hlásí při pokusu o načtení operačního systému chybu.

č.1	Administrátorské práva
Popis	Ověření nutnosti administrátorských práv
Vstupní podmínky	1)Aplikace běží bez administrátorského oprávnění
Postup	1)Spustíme aplikaci 2)Vytvoříme jednoho VM potomka hostitelského počítače 3)Spustíme generování prostředí
Očekávaný výsledek	V logovacím souboru Log1.log nalezneme stavovou hlášku "Please run this application as Administrator."
Skutečný výsledek	Podle očekávání

Tabulka 7.1: Administrátorské práva.

č.2	Export prostředí
Popis	Ověření funkčnosti exportu vymodelovaného prostředí
Vstupní podmínky	1)Spuštěná aplikace 2)Vytvořená struktura prostředí
Postup	1)V horním menu klikneme na File 2)Vybereme Save... 3)V otevřeném okně zadáme název souboru 4)Klikneme na Uložit
Očekávaný výsledek	Ve vybraným adresáři nalezneme xml soubor s vytvořeným prostředí
Skutečný výsledek	Podle očekávání

Tabulka 7.2: Export prostředí.

č.3	Import prostředí
Popis	Ověření funkčnosti importu vymodelovaného prostředí
Vstupní podmínky	1)Spuštěná aplikace 2)Vytvořená struktura prostředí v xml souboru
Postup	1)V horním menu klikneme na File 2)Vybereme Open... 3)V otevřeném okně vybereme souboru 4)Klikneme na Otevřít
Očekávaný výsledek	V aplikaci se objeví vytvořené prostředí podle xml souboru
Skutečný výsledek	Podle očekávání

Tabulka 7.3: Import prostředí.

č.4	Tvorba prostředí
Popis	Ověření funkčnosti tvorby prostředí dvou VMs
Vstupní podmínky	1)Spuštěná aplikace
Postup	1)Vytvoříme dvě VMs na stejné úrovni 2)Zvolíme dostatečný počet RAM (1000MB) 3)Vybereme existující VHd 4)Klikneme na tlačítko Create
Očekávaný výsledek	Aplikace vygeneruje dva VMs podle zadání
Skutečný výsledek	Podle očekávání

Tabulka 7.4: Tvorba prostředí dvou VMs na první úrovni.

č.5	Tvorba prostředí
Popis	Ověření funkčnosti tvorby prostředí VM
Vstupní podmínky	1)Spuštěná aplikace
Postup	1)Vytvoříme VM 2)Zvolíme dostatečný počet RAM (1000MB) 3)Vybereme nové VHd 4)Klikneme na tlačítko Create
Očekávaný výsledek	Aplikace vytvoří bázový disk a vygeneruje VM podle zadání
Skutečný výsledek	Aplikace končí chybou

Tabulka 7.5: Tvorba prostředí dvou VMs na první úrovni.

č.6	Tvorba prostředí
Popis	Ověření funkčnosti tvorby prostředí vnořeného VM
Vstupní podmínky	1)Spuštěná aplikace
Postup	1)Vytvoříme vnořené VM 2)Zvolíme dostatečný počet RAM (1)2500MB 2)1200MB) 3)Vybereme existující VHd 4)Klikneme na tlačítko Create
Očekávaný výsledek	Aplikace vygeneruje VMs podle zadání
Skutečný výsledek	Aplikace končí chybou

Tabulka 7.6: Tvorba prostředí dvou VMs na první úrovni.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo analyzovat a specifikovat požadavky pro vytvoření systému, který automatizuje vytváření komplexních prostředí pomocí vnořené virtualizace. Dále bylo požadováno tento systém navrhnout, implementovat a otestovat nad dostatečným obsahem testovacích případů. První část práce byla věnována stěžejním informacím pro pochopení problematiky virtualizace. Práce se dále zmiňuje o nejčastěji se vyskytujících virtualizačních řešení s hlavním důrazem na Hyper-V od Microsoft, které je použito v této práci. Dále se práce zabývá analýzou a specifikací požadavků pro vytvořený systém a jeho správné fungování. Na základě analýzy byla navržena architektura aplikace. Následující část je věnována popisu implementace systému, která se uskutečnila v jazyce C# a powershell skriptech. V poslední části je věnována pozornost testování aplikace, kde je ověřována správnost funkcionality vytvořené aplikace. Při testování se objevily některé chyby, které se nepodařily vyřešit. První chyba znemožňuje vytvořit nový bázový disk pomocí aplikace. Tato chyba lze obejít manuálním spuštěním skriptu `Build-Images.ps1` z konzole a poté zvolením existujícího bázového disku v aplikaci. Druhá objevená chyba se vyskytuje při použití bázového disku u vnořené virtualizace. Vytváření VM probíhá v pořádku, dojde však k poškození disku. Prototyp aplikace pro vytváření komplexních prostředí pomocí vnořené virtualizace se až na zmíněné chyby podařilo úspěšně implementovat. Tento prototyp může být po odstranění chyb využíván pro generování prostředí v rámci Windows Server 2016. Dále může být použít jako základ pro vytvoření robustnějšího řešení podporující rozšířené možnosti virtualizace a jejího nastavení.

8.1 Budoucí rozšíření systému

Následující úpravy systému by se měli zaměřit na opravu uvedených chyb. Další rozšíření systému by se mohlo týkat rozšíření možností vytvářených prostředí. Existuje velké množství rozšiřujících možností VMs a VMs struktur. Mohla by se například implementovat podpora pro linuxové operační systémy, nebo podpora pro rozšířené možnosti specifikace úložných prostorů ve vytvářených VM.

Literatura

[1] Bureš, M.: *Virtualizace - porovnání dvou daných platforem*. Diplomová práce, Bankovní institut vysoká škola Praha, 2009.

[2] contributors, W.: Data binding — Wikipedia, The Free Encyclopedia. 2017, [Online; citováno 28.09.2018].
URL
https://en.wikipedia.org/w/index.php?title=Data_binding&oldid=817164232

[3] contributors, W.: Red Hat Virtualization — Wikipedia, The Free Encyclopedia. 2017, [Online; citováno 28.09.2018].
URL https://en.wikipedia.org/w/index.php?title=Red_Hat_Virtualization&oldid=786896213

[4] contributors, W.: Simple Protocol for Independent Computing Environments — Wikipedia, The Free Encyclopedia. 2017, [Online; citováno 28.09.2018].
URL https://en.wikipedia.org/w/index.php?title=Simple_Protocol_for_Independent_Computing_Environments&oldid=814756324

[5] contributors, W.: VMware Workstation Player — Wikipedia, The Free Encyclopedia. 2017, [Online; citováno 25.09.2018].
URL https://en.wikipedia.org/w/index.php?title=VMware_Workstation_Player&oldid=808217164

[6] contributors, W.: Mockup — Wikipedia, The Free Encyclopedia. 2018, [Online; citováno 28.09.2018].
URL https://en.wikipedia.org/w/index.php?title=Mockup&oldid=832867290

[7] contributors, W.: Red Hat Enterprise Linux — Wikipedia, The Free Encyclopedia. 2018, [Online; citováno 28.09.2018].
URL https://en.wikipedia.org/w/index.php?title=Red_Hat_Enterprise_Linux&oldid=818308306

[8] contributors, W.: VirtualBox — Wikipedia, The Free Encyclopedia. 2018, [Online; citováno 25.09.2018].
URL
https://en.wikipedia.org/w/index.php?title=VirtualBox&oldid=819207996

[9] contributors, W.: VMware ESXi — Wikipedia, The Free Encyclopedia. 2018, [Online; citováno 25.09.2018].
URL
https://en.wikipedia.org/w/index.php?title=VMware_ESXi&oldid=818138110

[10] Cooley, S.: Windows 10 Hyper-V System Requirements. 2016, [Online; citováno 13.10.2018].
URL https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-requirements

[11] Docs, M.: Run Hyper-V in a Virtual Machine with Nested Virtualization. 2016, [Online; citováno 13.10.2018].
URL https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/nested-virtualization

[12] Dotnetpattern: WPF MVVM Introduction. [Online; citováno 02.10.2018].
URL http://dotnetpattern.com/wpf-mvvm-introduction

[13] Garrison, J.: What Is a Virtual Machine Hypervisor? 2016, [Online; citováno 03.10.2018].
URL https://www.howtogeek.com/66734/htg-explains-what-is-a-hypervisor/

[14] Halamíček, B. P.: *Techniky a možnosti virtualizace výpočetního prostředí.* Diplomová práce, Mendelova zemědělská a lesnická univerzita v Brně, 2009.

[15] Šindelář, B. P.: *Virtualizace serverů v podnikovém prostředí.* Diplomová práce, Bankovní institut vysoká škola Praha, 2014.

[16] Komínek, J.: *Porovnání virtualizačních technologií.* bakalářská práce, Vysoká škola polytechnická Jihlava, 2013.

[17] Lownds, P.; Nemnom, C.; Carvalho, L.: *Windows Server 2016 Hyper-V cookbook: save time and resources by getting to know the best practices and intelligence from industry experts; 2nd ed.* Birmingham: Packt Publishing, 2017.
URL http://cds.cern.ch/record/2254224

[18] MSDN, M.: The MVVM Pattern. [Online; citováno 02.10.2018].
URL https://msdn.microsoft.com/en-us/library/hh848246.aspx

[19] Čápka, D.: 4. díl - UML - Doménový model. [Online; citováno 13.10.2018].
URL https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram

[20] Posey, B.: Storage virtualization. 2006, [Online; citováno 10.10.2018].
URL
http://searchstorage.techtarget.com/definition/storage-virtualization

[21] Rouse, M.: Server virtualization. 2009, [Online; citováno 10.10.2018].
URL http://searchservervirtualization.techtarget.com/definition/server-virtualization

[22] Rouse, M.: Storage area network (SAN). 2015, [Online; citováno 13.10.2018].
URL
http://searchstorage.techtarget.com/definition/storage-area-network-SAN

[23] Rouse, M.: Virtualization. 2016, [Online; citováno 10.10.2018].
URL http:
//searchservervirtualization.techtarget.com/definition/virtualization

[24] Rouse, M.: Virtualization. 2016, [Online; citováno 10.10.2018].
URL
http://searchservervirtualization.techtarget.com/definition/hypervisor

[25] Stroud, F.: Nested virtualization. [Online; citováno 13.10.2018].
URL https://www.webopedia.com/TERM/N/nested-virtualization.html

[26] TechNet, M.: What Is a Server Cluster? [Online; citováno 03.10.2018].
URL https://technet.microsoft.com/en-us/library/cc785197(v=ws.10).aspx

[27] Techopedia: Datastore. [Online; citováno 03.10.2018].
URL https://www.techopedia.com/definition/23343/datastore

[28] Techopedia: Hypervisor. [Online; citováno 02.10.2018].
URL https://www.techopedia.com/definition/4790/hypervisor

[29] Techopedia: Network virtualization. [Online; citováno 03.10.2018].
URL https://www.techopedia.com/definition/655/network-virtualization

[30] Techopedia: Virtual Machine (VM). [Online; citováno 02.10.2018].
URL https://www.techopedia.com/definition/4805/virtual-machine-vm

[31] Vigo, J.: Windows Server 2016: A cheat sheet. 2017, [Online; citováno 13.10.2018].
URL https://www.techrepublic.com/article/windows-server-2016-the-smart-persons-guide/

[32] Wikipedie: Virtualizace — Wikipedie: Otevřená encyklopedie. 2017, [Online; citováno 25.09.2018].
URL https://cs.wikipedia.org/w/index.php?title=Virtualizace

[33] Wikipedie: Xen — Wikipedie: Otevřená encyklopedie. 2017, [Online; citováno 25.09.2018].
URL https://cs.wikipedia.org/w/index.php?title=Xen&oldid=14732023

[34] Wikipedie: Unified Modeling Language — Wikipedie: Otevřená encyklopedie. 2018, [Online; citováno 28.09.2018].
URL https://cs.wikipedia.org/w/index.php?title=
Unified_Modeling_Language&oldid=16031471

[35] Wikipedie: Windows Presentation Foundation — Wikipedie: Otevřená encyklopedie. 2018, [Online; citováno 02.10.2018].
URL https://cs.wikipedia.org/w/index.php?title=
Windows_Presentation_Foundation&oldid=16023112

[36] WPF-tutorial: What is XAML? [Online; citováno 02.10.2018].
URL http://www.wpf-tutorial.com/xaml/what-is-xaml/

[37] Wyckoff, C.: *All You Need to Know About Microsoft Windows Server 2016 Virtualization (Updated for Windows Server 2016 GA)*. VEEAM, 2017.

Příloha A

Obsah přiloženého CD

- **dp.pdf** - Tato bakalářská práce ve formátu PDF
- **source/** - Zdrojové soubory aplikace
- **source/Readme.txt** - Návod ke spuštění aplikace
- **textSource/** - Zdrojové soubory textové části bakalářské práce