Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Odolnost virtuálních platforem vůči DDoS útokům

Diplomová práce

Autor: Bc. Martin Chaloupka
Studijní obor: AI2-p

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.
UHK FIM – Katedra informatiky a kvantitativních metod
Odborný konzultant: Ing. Jan Budina
UHK FIM – Institut moderních informačních technologií

Hradec Králové Duben 2021

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 29.4.2021 Martin Chaloupka

Poděkování:

Děkuji vedoucímu diplomové práce panu Ing. Filipu Malému, Ph.D. a panu Ing. Janu Budinovi za odborné vedení práce, ochotu a konzultace při tvorbě této odborné práce.

Anotace

Jedním z cílů této práce je teoretické prozkoumání nejvíce používaných virtualizačních technologií a jejich rozdílů proti kontejnerovým technologiím. V praktické části na těchto technologiích je vytvořena webová aplikace. Na tuto aplikaci je zde představena velká hrozba v podobě DDoS útoku. Jeden z takových útoků je v praktické části práce cíleně vytvořen a zaslán na testovací prostředí, provozující již zmíněnou Java Spring aplikaci. První testovací prostředí je založené na platformě VMware, druhé testovací prostředí je na Dockeru. Na to navazuje kapitola konfigurace haproxy load-balanceru. Poté jsou provedena testování virtualizačních i kontejnerových prostředí. Zkoumaná je hlavně zátěž webových serverů a dostupnost aplikace. V každém testu jsou upraveny hardwarové prostředky podle zatížení z minulého útoku a nakonec i vyvozen závěr.

Annotation

One of the goals of this paper is theoretical exploration of the most used virtualization technologies and its differences in comparison with container technologies. A web application is created on both of those technologies in the practical part of this paper. There is also introduced a threat for this application in a form of a DDoS attack. One of those types of attack is created and used against both test environments that are running Java Spring application, which was mentioned earlier. First test environment is based on VMware platform, the other is based on Docker. Follow-up on this chapter, there is a section about haproxy load-balancer configuration. After that there are multiple test executions on the previously created virtualization and container environments. The focus of those test results are mainly on web servers performances and application availability. There are modifications of hardware resources for each test dependent on server performance from the last executed test and at the end the conclusions are made.

Title: Resilience of virtual platforms to DDoS attacks

Klíčová slova:

Virtualizace, VMware, Docker, DDoS, Load-balancing, Java, Spring Framework

Key words:

Virtualization, VMware, Docker, DDoS, Load-balancing, Java, Spring Framework

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V moderní době informačních technologií se stále vytvářejí nové způsoby, jak by bylo možné využívané technologie vylepšit. Ve většině případů přechod na novější technologii znamená nejen dlouhodobou úsporu, ale také právě zefektivnění práce. Jednou z technologií přinášejících tyto benefity je právě virtualizace.

Různé typy virtualizace jsou využívány velkými i malými firmami. Díky virtualizaci není potřeba celý tým zaměstnanců starajících se o velké množství fyzických serverů. Proto jde o velmi účinnou a úspornou metodu tvorby firemní IT infrastruktury. Nejčastěji se využívá virtualizace serverová, protože šetří energii, výkon i potřebný fyzický prostor.

Někdy však ani dobře postavená infrastruktura nezaručí úplnou bezpečnost. S čím dál větší úspěšností firemního byznysu přichází i větší šance stát se obětí digitálního útoku, ať už od konkurence či nezávislé skupiny neetických hackerů. Jedním z jednodušších napadení systému je vytvoření tzv. DDoS útoku. Takový útok spočívá v tom, že obsahuje velké množství malých požadavků na server. Jednodušší DDoS útoky jsou vypuštěny z jedné nebo několika IP adres, takže je celkem snadné je na firewallu identifikovat a zablokovat. Složitější způsob DDoS útoku přichází z velkého množství zařízení, a tak je velmi těžké se proti takovému útoku efektivně bránit.

Obsahem teoretické části budou různé způsoby využití virtualizací a jejich platforem včetně příkladů využití v praxi. Dále bude provedeno porovnání virtualizace s Docker kontejnerizací. K této části také patří i rozebrání metody load-balancingu. Nakonec budou rozebrány hrozby ve formě digitálních útoků na webové aplikace, které se v dnešní době stávají velmi populární a často právě jsou provozovány na virtualizovaných zařízeních. V praktické části budou pak navržené a posléze vytvořené virtuální a kontejnerové testovací prostředí, na které bude poslán DDoS útok, jenž je již dlouhodobě jedním z častých digitálních útoků na webové aplikace. Po vyslání útoku se potom bude zkoumat efektivita a stabilita vytvořené aplikace provozované na virtualizovaném a kontejnerovém stroji.

2 Cíl práce

Jedním z cílů této práce je teoretické prozkoumání nejvíce používaných virtualizačních technologií. Další část této práce bude také rozbor hrozeb digitálních útoků na webové aplikace, jako je tzv. DDoS útok. K tomu patří i prozkoumání load-balancingu jako efektivní součásti softwarového návrhu infrastruktury. Praktická část se bude zabývat návrhem a konfigurací testovacího prostředí na různých platformách. V závěrečné části je příprava digitálního útoku a test odolnosti navržených prostředí, založených na různých platformách proti tomuto útoku, a jejich vzájemné porovnání.

3 Teoretická část

Teoretická část práce se věnuje metodám virtualizace, jejich využití převážně v podnikové, ale i soukromé sféře. Budou rozebrány typy virtualizace a výběr nejznámějších platforem. V další části pak bude rozebrána hrozba ve formě digitálních útoků konkrétně na webové aplikace, které se stávají čím dál populárnějšími. Nakonec teoretické části bude rozebrána služba load-balancingu v rámci zlepšení efektivity softwarové infrastruktury.

3.1 Virtualizace

Virtualizací je běžně nazýván proces, ve kterém je spuštěna virtuální instance počítačového systému. Takový systém funguje v abstraktní vrstvě oddělené od reálného hardwaru. Z toho důvodu je možné na jediném fyzickém stroji provozovat více různých operačních systémů, a tak ušetřit hardwarové prostředky. Hlavní výhodou je velká úspora energie a také technická správa daného stroje. Není totiž potřeba vymezit celé budovy jen pro umístění těchto strojů, když stačí parametrově vylepšit pár serverů. Tyto výkonné servery jsou schopné za pomoci virtualizačních platforem rozdělit své hardwarové prostředky virtuálním systémům. A samozřejmě je tu také výhoda správy těchto systémů, jelikož běží pod stejnou virtualizační platformou. V rámci několika kliknutí myší je možno přidat nebo odebrat hardwarové prostředky danému systému (například více nebo méně paměti, velikost disku a další).

Virtualizace však není záležitost pouze velkých firemních serverů. I běžné desktopy mají v dnešní době dostatek výkonu, aby za pomoci tohoto procesu mohly spouštět různé aplikace vytvořené pro jiné operační systémy, aniž by bylo zapotřebí daný počítač restartovat nebo dokonce reinstalovat.
Virtualizace může být rozdělena na několik typů:

- Takzvaná **úplná virtualizace** je případ využití, kde se kompletně oddělí fyzická vrstva hardwaru od poskytovaného operačního systému. Všechny požadavky na hardware jdou potom přes využívaný hypervisor (viz kapitola 3.1.1), který je zpracuje. Výhod je zde tedy mnoho. Virtualizovanému systému může být přiděleno více či méně paměti nebo

prostoru na disku, a tak lze daný virtuální stroj upravit podle momentální potřeby. Pokud to výkon umožňuje, může potom na daném fyzickém stroji běžet více takových systémů. To poskytne jak úsporu fyzického místa (nahrazením deseti fyzických strojů za jeden, na kterém bude provozována virtualizace), tak úsporu potřebného výkonu pro další virtuální stroje. Jedna z největších výhod je ovšem nezávislost virtualizovaného systému na poskytnutém hardwaru fyzického stroje. To nám dává bezproblémovou možnost zálohy a případné přenositelnosti na jiný fyzický stroj, třeba i s jiným fyzickým hardwarem, a to v rámci několika kliknutí myší.

- **Paravirtualizace** neboli virtualizace asistovaná operačním systémem. Tato virtualizace je založena na komunikaci mezi poskytovaným operačním systémem a hypervisorem. Paravirtualizace funguje na způsobu modifikace kernelu operačního systému. Instrukce, které nelze zvirtualizovat, jsou tímto nahrazeny hyper-voláními („hypercalls“), která komunikují přímo s virtualizační vrstvou hypervisoru. Tato hyper-volání jsou zde také využita za účelem kritických operací, jako například správy paměti nebo přerušení operací.
- **Hardwarově asistovaná virtualizace** je dalším druhem těchto metod virtualizace. Jde již o předpřipravenou podporu virtualizace daným procesorem. Taková podpora znamená, že se tyto instrukce zadané manažerem virtuálních strojů zpracovávají ve speciálně vytvořeném prostoru, a proto není zapotřebí existence prostředníka pro překlad těchto virtuálních instrukcí, nebo o vytvoření paravirtualizace.[1]
- **Virtualizace na úrovni operačního systému** je poslední metodou virtualizace, která se vytváří až na vrstvě operačního systému. Jde o případ, kde jádro operačního systému povolí více než jednu instanci izolovaného uživatelského prostoru. Takový prostor je pak nazván kontejnerem. Vznikne tedy jádro operačního systému, které spustí samostatný operační systém a umožní mu replikovat se do jednotlivých izolovaných částí neboli kontejnerů. [2]

Většina firemního podnikání je závislá na provozu svých aplikací. Pokud server nebo aplikace selže, je zapotřebí problém co nejdříve vyhledat a zprovoznit danou službu. Mezi již zmíněné nesporné výhody patří také zařazení funkce obnovy provozu po kritickém selhání (v anglickém jazyce „disaster recovery“). Jde o funkci, která zreplikuje každý virtuální stroj. Tuto repliku nebo její předchozí stav lze pak obnovit nebo nahrát na jiný virtualizační stroj v krátké době několika minut.[3]

3.1.1 Hypervisor

Pojem v angličtině zvaný jako „hypervisor“ je program, který obecně umožňuje paralelní běh více operačních systémů. Tyto operační systémy tedy sdílejí stejné hardwarové prostředky. Tyto jednotlivé systémy jsou pak nazývány virtuálním strojem (v anglickém jazyce „virtual machine” neboli „VM“). Virtuální stroj si lze představit jako softwarově vytvořený samostatný počítač. Hypervisor slouží jako rozhraní mezi virtuálními stroji a fyzickým hardwarem. Důležitou funkcí je kontrola přidělování paměťového prostoru, výpočetního výkonu a jiných hardwarových prostředků tak, aby nedocházelo k narušení chodu jednotlivých strojů. Jinými slovy lze říct, že hypervisor je monitor virtuálních strojů („virtual machine monitor“ neboli „VMM“). Obecně se hypervisory rozdělují na dva typy:

- **První typ** nahrazuje operační systém přímo na virtuálním serveru (např. VMware, HyperV a další). Tento typ je velmi efektivní právě kvůli přímému přístupu k fyzickému hardwaru.
- **Druhý typ** funguje jako aplikace na již existujícím operačním systému. Používá se například na koncových zařízeních, například na desktopech/laptopech, při potřebě spuštění jiného operačního systému bez nutnosti restartování počítače (např. Oracle VirtualBox, VMware Workstation…). Tento typ nám tedy nabízí rychlý a snadný přístup do alternativního operačního systému běžícího současně s primárním systémem. A ačkoliv to velmi zvyšuje produktivitu koncového uživatele, je zde nevýhoda menší bezpečnosti alternativního systému. Pokud by se útočníkovi úspěšně podařilo napadnout primární systém, mohl by snadno manipulovat s jakýmkoliv alternativním systémem běžícím na daném

hypervisoru. Tohle bezpečnostní riziko a odezva systému, způsobené veškerou procházející komunikací přes primární systém, je důvod, proč se tento typ hypervisoru většinou nepoužívá v serverovém řešení. [4]

3.1.2 Způsoby použití virtualizace

Serverová virtualizace

Ve firemním prostředí si síťoví administrátoři obvykle prosazují standard, že každá aplikace by měla mít svůj vlastní server. Je to z toho důvodu, aby k jednotlivému serveru byla možnost upřesnění bezpečnostní politiky firmy. Dále je také jednodušší vypátrat a opravit chyby, které se mohou vyskytnout. Proto se často stává, že výkon daných serverů je z větší části nevyužitý a je potřeba vyhradit více prostoru v serverových místnostech. Všechny tyto situace řeší serverová virtualizace. Jedná se o přeměnu jednoho fyzického serveru na více virtuálních strojů. Každý virtuální stroj se potom chová jako unikátní fyzické zařízení se schopností běhu vlastního operačního systému.[5]

Desktopová virtualizace

Desktopová virtualizace umožňuje spustit na jednom počítači více desktopových operačních systémů. Existují tři druhy desktopové virtualizace:

- **Virtual desktop infrastructure (VDI)** provozuje operační systém na virtuálním stroji v centrálním serveru, který je umístěn v datovém centru. Desktopový obraz operačního systému je poté poslán po síti do koncového zařízení uživatele. Uživatel poté může za pomoci tenkého klienta využívat desktop, jako by ho měl nainstalován lokálně. Tímto způsobem VDI poskytuje každému uživateli jejich vlastní vyhrazený virtuální stroj s vlastním operačním systémem.
- **Remote desktop services (RDS)** je metoda, kdy se uživatelé vzdáleně připojují k desktopům na Windows Server. Využívají k tomu speciálního protokolu RDP. Z pohledu koncového uživatele je VDI a RDS stejné, avšak

hlavním rozdílem je, že Windows Server podporuje mnoho zároveň přihlášených uživatelů limitovaných pouze výkonem serverového hardwaru. [6]

- **Desktop as a Service (DaaS)** je způsob, kde jsou virtuální stroje spouštěny na cloudovém řešení poskytovatele třetí strany. DaaS je nabízeno jako služba založená na předplatném. Funguje tedy na bázi webového prohlížeče nebo zabezpečené aplikace přes HTTP protokol. [7]

Síťová virtualizace

Síťová virtualizace se v dnešní době používá velmi často. Umožňuje totiž sdílené fyzické síťové infrastruktuře přidělit více funkcionalit. Například je požadováno, aby jeden fyzický server s jednou síťovou kartou provozoval více webových aplikací. Dále je tedy možné s minimální námahou z grafického uživatelského rozhraní rozdělovat rychlost pásma dle potřeby nebo i přidělit další virtuální síťový uzel.[8]

Síťové virtualizační funkce („Network functions virtualization“) neboli NFV zajišťují virtualizaci síťových služeb, jako jsou routery, firewally nebo také load-balancery, které slouží k rozdělení zátěže síťového provozu. V minulosti bylo celkem běžné, že tyto služby běžely na svém vlastním hardwaru. To se však změnilo s nástupem virtualizačních technologií. V dnešní době lze tyto služby vytvořit jako virtuální stroje na serveru. Tato NFV vlastnost zlepšuje schopnost poskytovatele přidávat nové síťové služby na požádání bez nutnosti instalace nových hardwarových prostředků. NFV architektura obsahuje:

- **Virtualizované síťové funkce (VNFs)**, což jsou softwarové aplikace zajišťující síťové funkce, jako je sdílení souborů, adresářové služby nebo konfiguraci IP protokolu.
- **Infrastrukturu virtualizačních síťových funkcí (NFVi)**, která se skládá z výpočetních, úložních nebo síťových komponent infrastruktury.
- **Správu, automatizaci a síťovou orchestraci (MANO)**, která poskytuje Framework pro správu NFV infrastruktury a obstarání nových VNFs.[9]

Aplikační virtualizace

Aplikační virtualizace spouští aplikační software bez nutnosti instalace na hostitelském operačním systému. Jedná se o zjednodušenou desktopovou virtualizaci. Tedy místo celého operačního systému je ve virtuálním prostoru spuštěna jen daná aplikace. Aplikační virtualizace lze rozdělit do tří typů:

- **Virtualizace lokální aplikace**, kde celá aplikace je spuštěna na koncovém zařízení v tzv. „runtime" prostředí. Tedy neběží přímo na nativním hardwaru.
- **Streamování aplikace**, kde aplikace běží na serveru, odkud se při požádání posílají malé komponenty daného softwaru koncovému zařízení.
- **Virtualizace serverové aplikace**, kde aplikace je spuštěna na serveru, odkud se koncovému zařízení posílá pouze uživatelské rozhraní. [10]

3.1.3 Virtualizační platformy

VMware

Společnost VMware byla založena v roce 1998 a je tedy jednou z největších a nejstarších společností na celém světě zabývajících se virtualizačními technologiemi. Jejich virtualizace se zaměřuje primárně na procesory založené na architektuře x86 a zahrnuje všechny způsoby využití virtualizací zmíněné v kapitole 3.1.2. Hypervisor společnosti VMware se označuje jako ESXi (dříve pouze ESX). Z hlediska firem zde stojí za zmínku nabídka produktů vSphere, vCenter nebo taky NSX Data Center.

VMware vSphere je kompletní balíček pro virtualizaci datových center. Je v něm obsaženo mnoho virtualizačních komponent:

- **VMware ESXi** je hypervisor, který slouží jako virtualizační vrstva běžící na fyzickém serveru. Obecná funkce hypervisoru je popsána výše v kapitole 3.1.1.
- **VMware vCenter** server je centrální software balíčku VMware vSphere. Slouží pro celkovou správu virtualizovaného firemního prostředí a

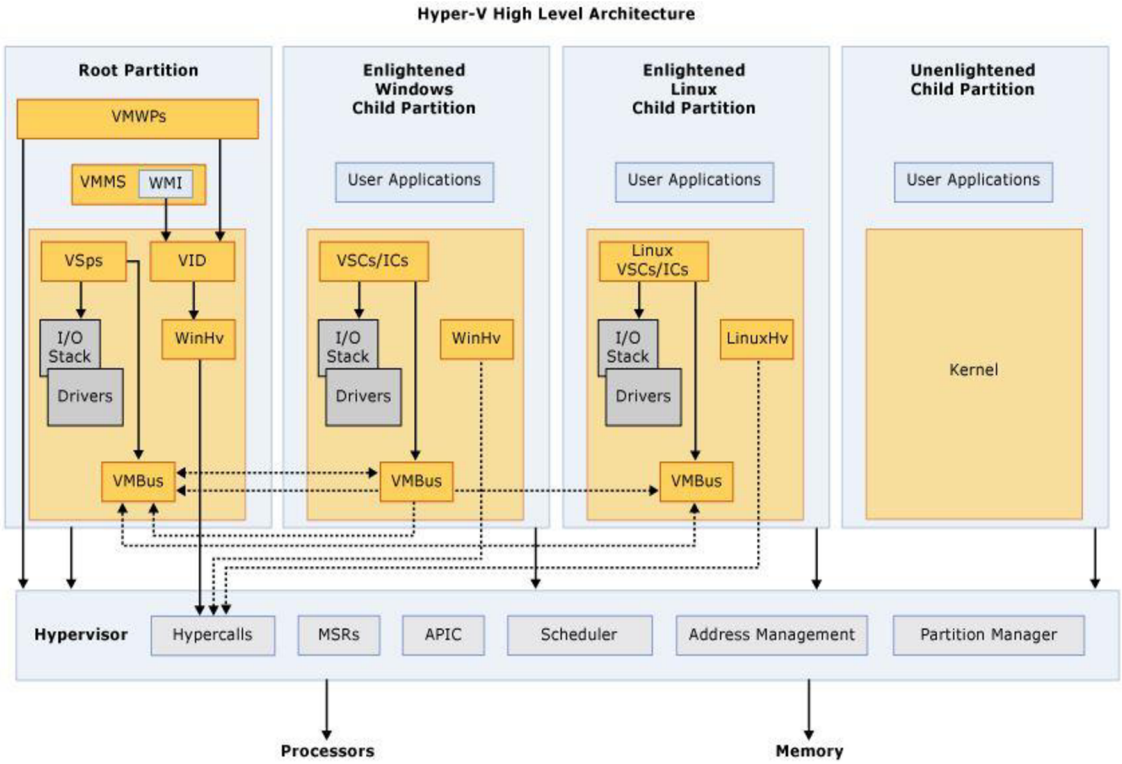
umožňuje tak zefektivnění práce a lepší přehled nad danou virtuální infrastrukturou. Vzdálený přístup do vCenter serveru je umožněn vSphere klientem, který je také obsahem vSphere balíčku.

- **vSphere High Availability (HA)** je vlastnost, která poskytuje vysokou dostupnost virtuálních strojů. Pokud tedy daný virtualizační server je z nějakého důvodu nedostupný, ovlivněné virtuální stroje se spustí na jiném dostupném serveru předem nastaveném v konfiguraci virtualizace.[11]

vSphere balíček ze základu obsahuje možnost vytvořit základní virtuální síťové uzly. Nicméně je zde možnost doinstalovat produkt **NSX Data Center**. Přes tento produkt lze nastavit a jednoduše spravovat logické switche, routery, load-balancery, firewally a další síťové funkcionality.[12]

Microsoft

Microsoft je jeden z největších konkurentů v oblasti virtualizace. Jejich produkt se nazývá Hyper-V. Hyper-V je hypervisor zabývající se virtualizací pro dané 64 bitové systémy Windows. Tento typ virtualizace lze spustit i na obyčejném desktopu bez nutnosti vlastnit server, avšak musí být podporován využívaným procesorem.



Obrázek 1 – Architektura Hyper-V. Zdroj: [13]

Hlavní vlastnosti Hyper-V dostupné pouze na Windows server oproti desktopu:

- Migrace virtuálních strojů za běhu systému z jednoho serveru na jiný.
- Možnost replikace virtuálních strojů vytvořených v Hyper-V.
- Technologii „Virtual Fiber Channel“, která umožňuje přímé připojení diskového pole s virtualizací přes Virtual Fiber Adapter. Komunikace přes Virtual Fiber Channel tedy obchází hostitelský server, čímž zvyšuje propustnost a snižuje čas odezvy.
- Sdílené virtuální disky (.VHDX) [14]

System center virtual machine manager neboli **SCVMM** je dodatečná a pouze serverová aplikace také od společnosti Microsoft. Hlavní funkcí SCVMM je poskytnutí snadnější správy fyzických a virtuálních strojů. I když je to tedy jen pro serverové systémy, logicky na desktopu neplánujeme vytvořit tak velkou virtuální strukturu, kterou bychom nezvládali spravovat.

Citrix

Citrix je společnost založená v roce 1989. Zabývá se virtualizací a cloudovými řešeními.

Produkt Citrix Hypervisor (dříve XenServer) je open-source platforma pro cloudové, serverové nebo i desktopové virtualizační infrastruktury. Jde o platformu podporující cloudové softwary, jako například CloudPlatform, Apache CloudStack nebo OpenStack, navržené pro lepší horizontální rozšíření.[15]

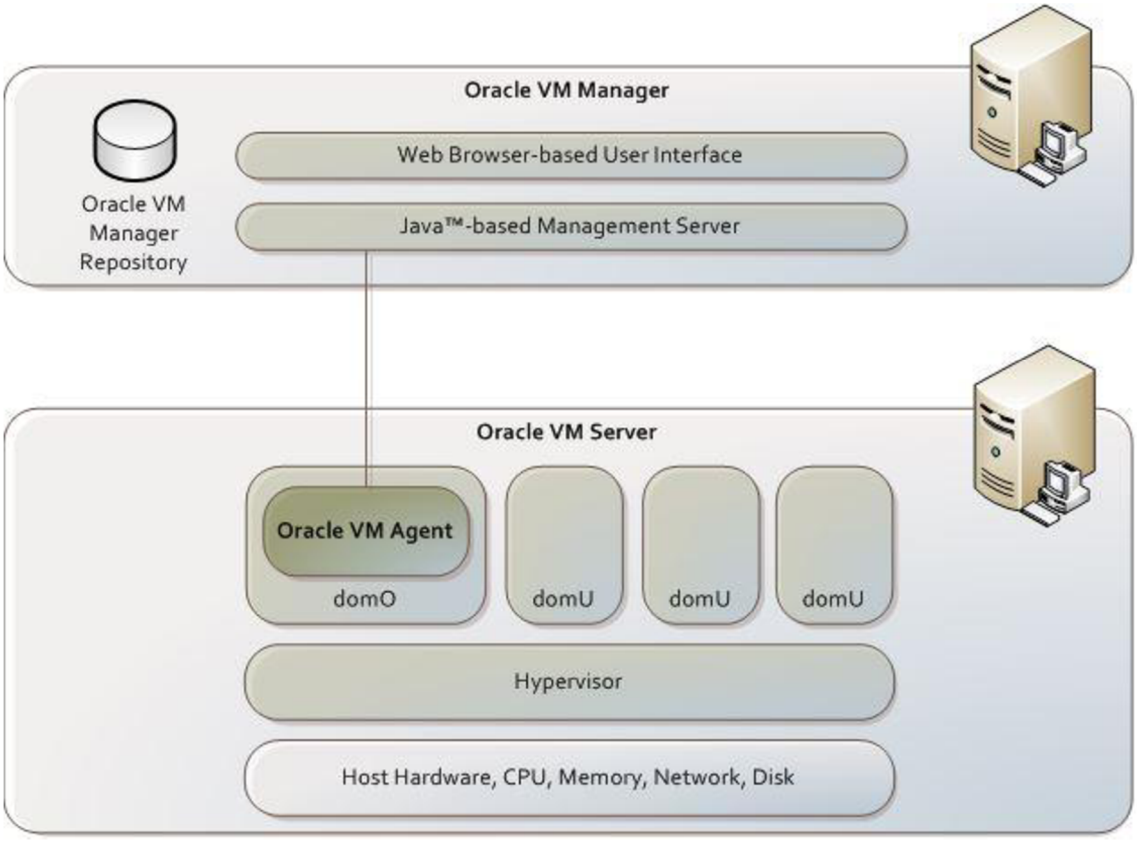
Jde o virtualizaci pro zařízení založených na x86 architektuře. Tato virtualizace je optimalizována jak pro Windows, tak i pro Linuxové virtuální servery.[16]

Oracle

Další velkou firmou, zabývající se nejenom virtualizacemi, je Oracle. Podobně jako společnost Microsoft má Oracle rozdělenou virtualizaci pro desktopy a servery.

Produkt **Oracle VM VirtualBox** je open-source multiplatformní software využívaný především k virtualizaci koncových zařízení. Jeho největší výhoda je podpora opravdu velkého množství operačních systémů. Protože však používá druhý typ hypervisoru, je potřeba mít na daném zařízení již předinstalovaný podporovaný operační systém.

Jak už z názvu vyplývá, produkt **Oracle VM Server** je serverové řešení virtualizace. Může být nainstalován pouze na zařízeních s hardwarem založeným na architektuře x86 nebo SPARC platformě. Výhodou tedy spíše je, že licence, podobně jako pro VirtualBox, je zcela zdarma.



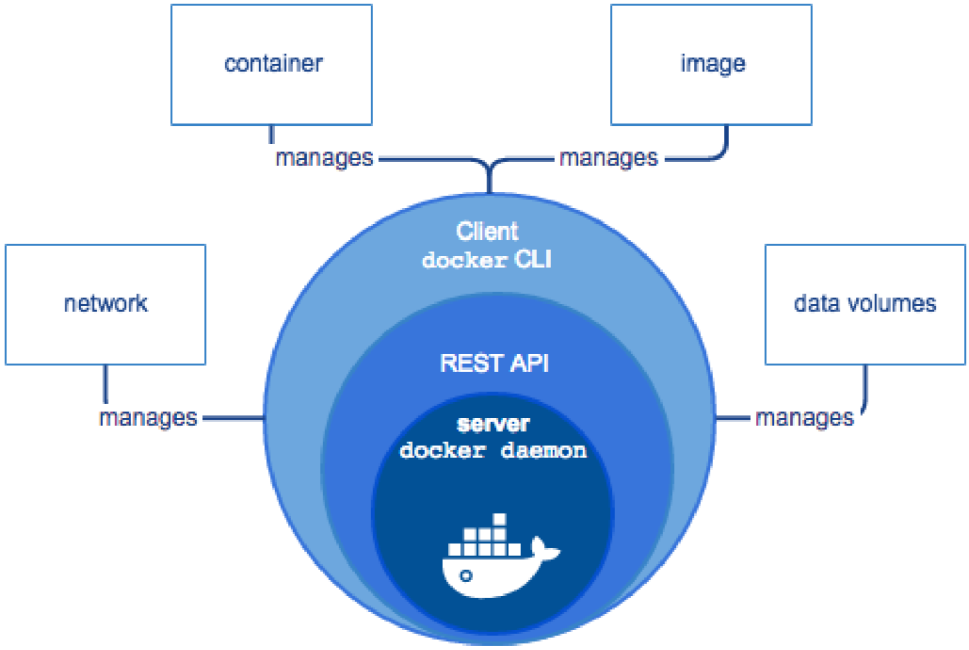
Obrázek 2 – Architektura Oracle virtualizace. Zdroj: [17]

3.2 Docker

Docker je open-source platforma pro vývoj a běh aplikací. Poskytuje možnost zapouzdření aplikace do izolovaného prostředí od ostatních programů, které se nazývá kontejner. Více těchto kontejnerů pak lze spustit na daném stroji. Kontejnerizace však bude podrobněji vysvětlená v další kapitole. Docker kontejnery lze spustit jak na lokálním laptopu, virtuálním stroji, datovém centru, tak i na cloudovém řešení. Díky této flexibilitě a nutnosti přibalení pouze knihoven potřebných pro daný program je velmi užitečnou alternativou k virtualizacím založeným na hypervisoru.

Docker Engine je klient-serverová aplikace obsahující tři komponenty. Server, který se nazývá daemon. Ten má za úkol vytvářet a spravovat Docker objekty. Docker objekty jsou například kontejnery, sítě nebo objemy dat. REST aplikační rozhraní, které může daný program použít ke komunikaci s daemonem.

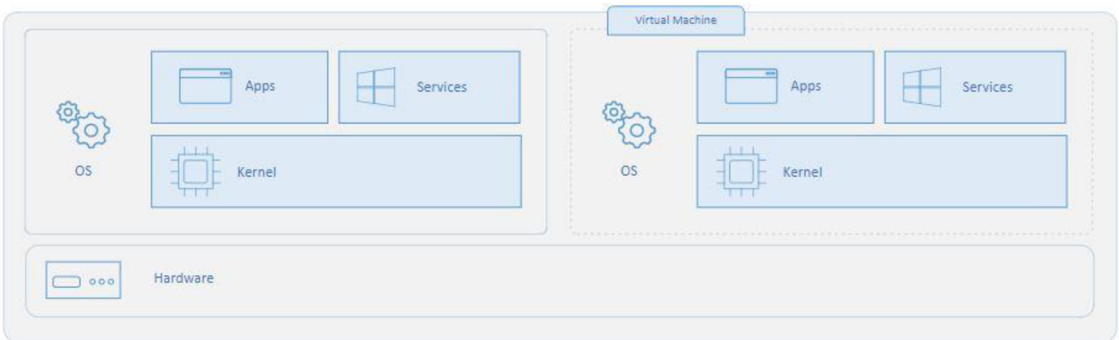
Poslední komponenta je rozhraní příkazového řádku („Command Line Interface“ neboli „CLI“).



Obrázek 3 – Docker engine. Zdroj: [18]

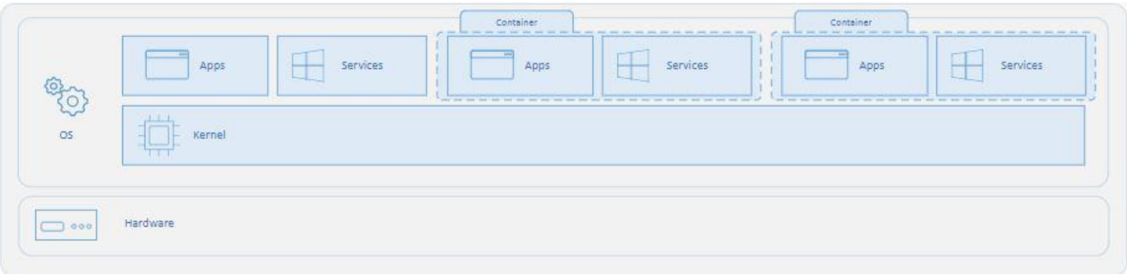
3.3 Virtualizace vs. kontejnerizace

Jak už víme z předešlých kapitol, virtuální stroje jsou spouštěny se svým vlastním operačním systémem. Na obrázku č. 4 je také vidět, že mají své vlastní systémové jádro neboli odborně kernel, které je zodpovědné za přidělování paměti a procesoru daným procesům.



Obrázek 4 – Architektura virtuálních strojů. Zdroj: [19]

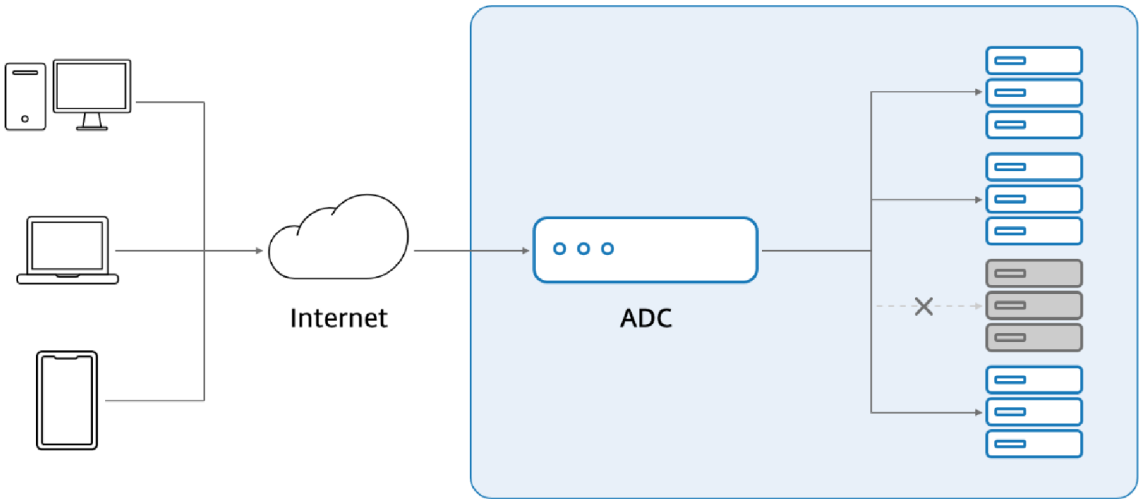
Naopak kontejner je izolovaný odlehčený prostor, ve kterém běží daná aplikace na stejném operačním systému, jako běží využívaný stroj. Dále s hostitelským strojem sdílí stejný kernel, na kterém jsou dané kontejnery spuštěny. Do těchto kontejnerů lze tedy nahrát jen jednoduchá aplikační rozhraní operačních systémů nebo služeb, které lze spustit v uživatelském modu. Velikost kontejnerů je tedy v řádech megabytů, zatímco virtuální stroje mají velikost v řádech gigabytů.



Obrázek 5 – Architektura kontejnerů. Zdroj: [19]

3.4 Load-balancing

Ve firmách poskytujících online služby je častým požadavkem rozložení zátěže mezi více serverů, aby byly schopny zpracovat vysoké množství požadavků v co nejkratším čase a vykrýt jejich špičky. V takové době je zapotřebí zakomponovat load-balancing. Load-balancing je jedna z nejdůležitějších funkcionalit v oblastech, kde se vyskytuje více serverů. Load-balancer je tedy fyzické zařízení nebo virtualizovaná jednotka běžící na specializovaném stroji, která má za úkol rozdělování požadavků od klientů tak, aby rovnoměrně rozložil zátěž na poskytnuté servery. Obecně lze load-balancer rozdělit na dva typy. Jedná se o základní rozdělení buď na hardwarový nebo softwarový load-balancer.

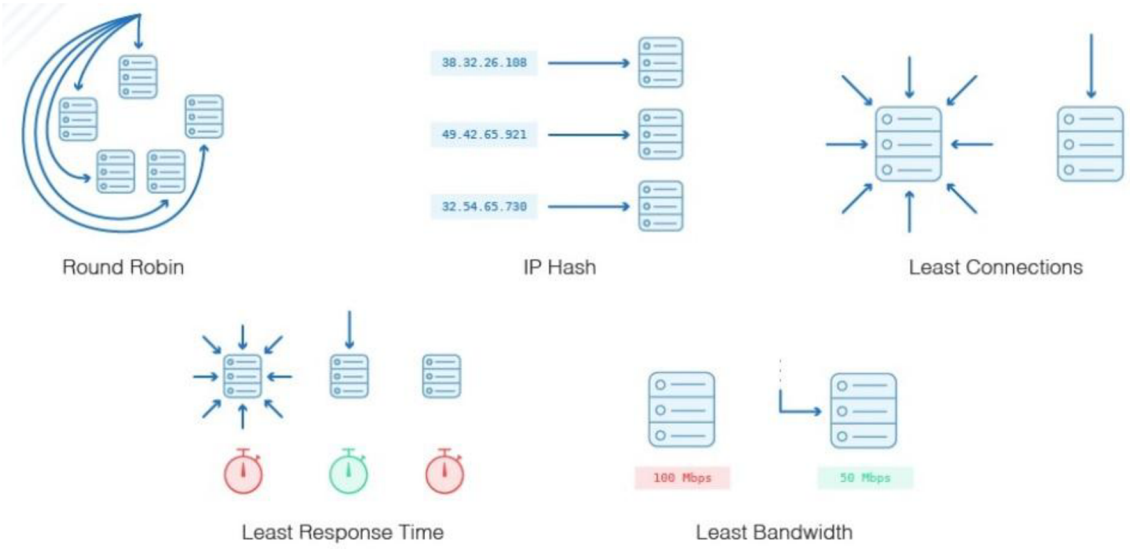


Obrázek 6 – Load-balancing. Zdroj: [20]

Na obrázku č. 6 je vidět komunikace ze strany klientů, která jde přes internet až do load-balanceru, jenž je zde zobrazen jako ADC („application delivery controller“). Ten má informace o zatížení poskytnutých serverů a na základě těchto informací rozpozná, zda příchozí požadavek daný server zvládne zpracovat. V jiném případě požadavek pošle na takový server, aby efektivně rozložil zátěž způsobenou požadavky klientů.

Existuje zde mnoho metod, jak daný load-balancer funguje:

- **Round Robin** je metoda, která se spoléhá na rotaci poskytnutých serverů. To znamená, že load-balancer zašle příchozí požadavek na první dostupný server a potom si daný server přesune na konec seznamu dalších dostupných serverů.
- **IP Hash** je jednoduchá metoda určování vhodného serveru na zpracování požadavku podle předem definované IP adresy serveru.
- **Least Connections** je metoda řadící seznam poskytnutých serverů od nejméně po nejvíce uživatelských připojení.
- **Least Response Time** je metoda, kde si load-balancer udržuje aktuální seznam podle nejmenší odezvy serverů.
- **Least Bandwidth** je metoda, kde load-balancer měří zahlcení sítě k serverům a potom pošle uživatelský požadavek na server s nejmenším síťovým zatížením. [21]



Obrázek 7 – Pět nejběžnějších load-balancingových metod. Zdroj: [21]

3.5 Webové aplikace a rozbor útoků

Webovou aplikací se nazývá taková aplikace, která je provozována na webovém serveru a umožňuje uživateli přístup přes webový prohlížeč. [22] Jde o velmi častou a rozrůstající se metodu tvorby aplikací. A jako na každou aplikaci i zde existuje mnoho útoků. V této podkapitole budou rozebrány záměrné útoky na webové aplikace, nikoliv útoky vytvořené chybou zařízení či jeho softwaru.

3.5.1 Účel digitálního útoku u webových aplikací

Obecně účel digitálního útoku lze rozdělit na tři druhy. Prvním druhem je získání přístupu k informacím, to znamená vniknutí do infrastruktury, ať už se jedná o osobní údaje klientů, či nějaké podklady pro analýzy dat. Dalším druhem je vyřazení konkurenční aplikace. Jedná se třeba o zatížení webových serverů natolik, že pro běžného klienta již aplikace není použitelná vzhledem k velké době odezvy požadavků. Posledním druhem je napadení samotného klienta. Většinou se jedná o prohlížení cookies souborů prohlížeče nebo přímo o zachytávání paketů v síti. Pokud se pak útočníkovi podaří tato data rozšifrovat, může sledovat kroky uživatele nebo zase sbírat osobní data klienta.

3.5.2 Konkrétní útoky na webové aplikace

SQL injection

SQL injection je velmi častý digitální útok, ve kterém hraje velkou roli nedostatečná znalost vývojáře aplikace. Jde o nesprávné zabezpečení vstupních požadavků ze strany klienta. Chybný je například generovaný SQL příkaz na zjištění informací přihlášeného uživatele do aplikace:

String param = getRequestString("UserId");
String textSQL = "SELECT * FROM Uzivatele WHERE Id_uzivatele = " + param;

Pokud tedy v aplikaci není ochrana, která by zabránila útočníkovi zadat jiný textový řetězec, stačí zaměnit nebo přidat určité praktiky k proměnné „param“ a útočník bude mít přístup ke všem informacím, které jsou uložené v tabulce uživatelů.[23]

Jde tedy o velmi jednoduchý a efektivní útok, kterému však lze jednoduše předejít při tvorbě dané webové aplikace.

Cross-Site scripting (XSS)

Jde o útok využívající nedostatečného zabezpečení webové aplikace či prohlížeče klienta. Tento útok využívá jazyka JavaScript, který vznikl už v roce 1995 a používá se do dnešní doby. Většinou se s tímto jazykem můžeme setkat při zobrazení webových stránek. [24]

Jedním ze způsobů je vložení skrytého JavaScriptového kódu do webového prohlížeče uživatele, který se tváří jako obyčejný text. Ten při překládání HTML jazyka rozezná tag *<Script>* a prohlížeč poté automaticky spustí předepsaný příkaz. Cross-Site scripting tedy vzniká na straně klienta webové aplikace a může poškodit jak uživatele, tak danou webovou aplikaci. JavaScriptový příkaz je spuštěn vždy na pozadí systému, a tak je těžké rozpoznat, zda je uživatel pod útokem. Tento útok může přesměrovat uživatele na předurčené webové stránky či dokonce ukrást tzv. „cookies“. Cookies jsou soubory webového prohlížeče, ve kterých se mimo jiné nachází i tzv. „session“ pro různé webové aplikace. Session primárně obsahuje

unikátní identifikátor webové aplikace, který je vytvořen a předán klientovi po úspěšné autentizaci a slouží k rozpoznání uživatele bez nutnosti pokaždé zadávat uživatelské jméno a heslo. S těmito údaji pak může útočník sledovat předešlé kroky klienta nebo vstupovat do webové aplikace v roli daného klienta, a tak narušit bezpečnost obou stran. V dnešních webových prohlížečích má JavaScript přístup k aplikačnímu rozhraní HTML5. To znamená, že zákeřný kód může získat přístup k poloze uživatele, webové kameře, mikrofonu a dalších příslušenství. [25]

Distributed denial of service (DDoS)

Jak už název v anglickém jazyce napovídá, jde o distribuovaný způsob odepření služby uživateli. Webové servery mají limitovaný počet požadavků, které mohou v daném čase splnit. Jedná se buď o limit výkonu serverů, nebo také o limit síťové infrastruktury. Pokud se jeden z těchto limitů překročí, dojde buď k velmi dlouhé odezvě požadavků, nebo dokonce k úplné nedostupnosti služby. [26]
Tento útok vychází obvykle z více již předem napadených zařízení, která jsou infikovaná malwarem. Každé z těchto zařízení většinou patří běžnému uživateli, a jelikož je tento útok spuštěn na pozadí infikovaného systému, uživatel ani netuší, že je součástí útoku. Takové uskupení napadených zařízení se nazývá „botnet“ nebo také „zombie armáda“. Obecně lze DDoS útoky rozdělit na 3 typy:

- **Množstevní** = Jde o vytvoření co největšího počtu požadavků za účelem zahlcení síťové infrastruktury. Využívá například UDP nebo ICMP („ping“) pakety.
- **Protokolový** = Vytváří tlak na serverové zdroje nebo na prostředky zpracovávající komunikaci, jako jsou firewally či load-balancery.
- **Aplikační vrstvy** = Obsahují požadavky s hlavičkou GET/POST, které se zaměřují na Apache server nebo na Windows systém. Účel tohoto typu útoku je zahltit samotný webový server. [27]

4 Praktická část

Praktická část této práce se nejdříve zabývá návrhem virtualizovaného testovacího prostředí za použití VMware technologie. Po návrhu bude dané testovací prostředí vytvořeno ve skutečném světě na fyzickém zařízení poskytnutém fakultou informatiky a managementu v Hradci Králové. Dále se obdobně vytvoří kontejnerové testovací prostředí s využitím Docker technologie. Na obě tyto prostředí bude navrhnuta a nasazena webová aplikace. Poté bude sestrojen DDoS útok (viz výše zmíněný v kapitole 3.5.2) na obě již vytvořené testovací prostředí a bude se provádět měření sledovaných veličin. Tento útok bude mít za úkol zjistit dostupnost webové aplikace při velké zátěži serverů. V provedených testech budou průběžně upravovány konfigurace prostředí podle vytížení hardwarových prostředků serverů a dostupnosti webové aplikace. Za úspěšný test se tedy bude považovat úplná dostupnost webové aplikace po celou dobu prováděného útoku. Na závěr této kapitoly budou shrnuty a porovnány výsledky testů obou technologií.

4.1 Návrh virtualizovaného testovacího prostředí

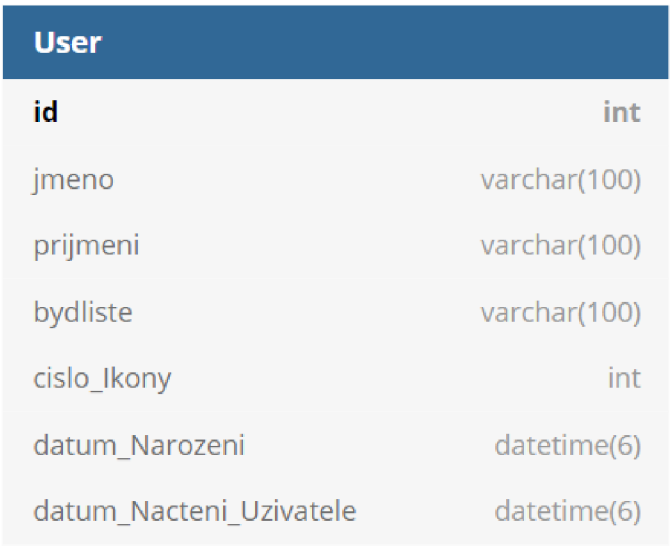
U návrhu testovacího prostředí je nutné nejprve určit funkcionalitu. V této práci bude testován DDoS útok na virtualizované servery poskytující webové aplikace. Je tedy zapotřebí nejdříve určit virtualizační technologii. Pro tento návrh virtualizovaného testovacího prostředí bude využita VMware technologie. Pro lepší rozdělení zátěže na server bude vhodné použít také load-balancer. Bude tedy zapotřebí jeden ESXi server, na kterém poběží více webových serverů, a jeden load-balancer, který bude distribuovat požadavky mezi zmíněné webové servery.

Bude tedy zapotřebí vytvořit čtyři další servery. Tři, které poskytnou přístup k webové aplikaci a jeden pro službu load-balanceru. Software na těchto serverech bude z hlediska bezplatného licencování založen na linuxové distribuci. Celá tato virtualizovaná infrastruktura bude mít svou vlastní VLAN 212. Z bezpečnostních důvodů bude celá síť odříznuta od internetu a jiných zařízení, která nejsou nutná pro toto testovací prostředí. Všechny zařízení budou mít staticky přidělenou IP adresu v dané VLAN.

Většina útoků nepřichází z lokální sítě, proto je vhodné vytvořit další ESXi server na zcela jiném fyzickém zařízení. Ten dostane IP adresu 192.168.212.11. Na něm budou alespoň dvě virtualizovaná zařízení, odkud bude vyslán cílený útok. Nakonec by tu mělo být vytvořeno zařízení, na kterém se budou monitorovat napadené virtuální stroje.

4.2 Návrh webové aplikace

K našemu testování bude stačit celkem jednoduchá webová aplikace. Hlavním požadavkem by mělo být načítání dat z jakékoliv databáze, která je snadno implementovaná jak ve virtualizovaném prostředí, tak i v tom kontejnerovém. Základní DDoS útok by v lepším případě měl pouze zpomalit odezvu daných webových serverů. Z toho důvodu bude vhodné navrhnout aplikaci tak, aby měla v běžném používání odezvu aspoň v řádech sekund. K těmto požadavkům bude dostačující lokální relační databáze s několika tisíci záznamů na jedné samotné tabulce, tudíž bez použití příkazu *„join“* nebo dokonce i *„union“*. Aplikace by měla být napsaná v jazyce, který je podporován linuxovou distribucí. Vzhledem k popularitě bude vhodnou volbou vytvoření aplikace v jazyce Java. Aplikace bude sloužit zcela základním zobrazením jednoduchých profilů nebo také vizitek registrovaných uživatelů.



Obrázek 8 – Návrh tabulky webové aplikace

Na obrázku č. 8 je zobrazen návrh tabulky pro ukládání dat uživatelů. Klasicky je zde nazván atribut id jako primární klíč. Jménu a příjmení daného uživatele včetně jeho bydliště byl přiřazen datový typ varchar s maximální délkou sta znaků. Číslo ikony je zde pro zjednodušení generovaných dat. Jde o náhodné číslo od jedné do pěti, podle kterého se načítá obrázek určující firemní oddělení uživatele. Tyto obrázky jsou zde přidané hlavně kvůli delšímu načítání celkové stránky a tím lepšímu odhadu odolnosti při plánovaném testování. Nakonec je tu také datum narození daného uživatele s datovým typem datetime s nejvyšší precizností na sekundy. Sloupec *„datum_Nacteni_Uzivatele“* je do tabulky přidán z důvodu možnosti do databáze také zapisovat. Tento sloupec bude přepsán pokaždé, když budou data z aplikace načtena. Nakonec aby se dala testovat funkčnost load-balanceru, bude vhodné i upravit webovou stránku pro každý webový server. Nejjednodušším způsobem je pouze přidat jeden řádek textu. Ten by mohl vypadat jako *„Hello World from server X!“,* kde X bude číslo webového serveru.

4.3 Vytvoření aplikace

Tato aplikace bude vytvořena v jazyce Java za použití Spring frameworku. Jde o moderní a populární nástroj pro tvorbu webových aplikací. Základní struktura aplikace je vygenerována pomocí služby Spring Initializr. Tuto službu lze najít na oficiálních stránkách Springu nebo i dokonce v nástrojích některých vývojových prostředí. Zjednodušeně to ušetří vývojáři čas s vytvořením potřebných konfiguračních souborů. Softwarová architektura bude rozdělena pomocí MVC (*„model view controller“*) modelu. Ten je běžným způsobem tvorby webových aplikací. Jde o architekturu rozdělující data od uživatelského pohledu za pomoci kontrolní komponenty.

V modelové části aplikace byl vytvořen objekt User, který bude pomocí JPA *(„Java persistence api“*) namapován na již navrhnutou tabulku uživatelů. JPA s využitím anotací umožňuje lehké mapování objektu. Tedy pro objektovou třídu user použijeme anotaci @Entity. Pro atribut id se použije @Id a v případě, že chceme vytvořit automatickou inkrementaci při vkládání objektů, použijeme také

@GeneratedValue. Ostatní atributy stačí označit @Column a v případně textových atributů přidat parametr length s požadovanou limitací délky slova.
Aby bylo možné přistupovat k databázi, je nutné si vytvořit repository, přes které bude k datům přistupováno. V této aplikaci je tedy nazváno UserRepository, které rozšiřuje JPArepository. JPArepository už obsahuje metody připojení datového zdroje a základní dotazovací funkce. Budou zde využity tři metody, které budou pracovat s databází. První, nazvaná *„findAll()“*, je už v základu definovaná v JPArepository. Slouží k načtení všech záznamů (v tomhle případě všech uživatelů). Druhou dotazovací metodu bude nutno vytvořit. Ta bude načítat maximálně tisíc uživatelských vizitek. Bude využita tato syntaxe s přidanou anotací:

```
@Query
(value = "SELECT * FROM user u WHERE u.id <= 1000", nativeQuery = true)
List<User> findThousandUsers();
```

A třetí metoda bude sloužit k zápisu do databázové tabulky *„User“*. Do sloupce *„datum_Načtení_Uživatele“* bude vložen datum načtení uživatelů. Tato metoda bude využita pouze při načtení všech uživatelů. Syntaxe pro úpravu dat databáze vypadá takto:

```
@Modifying
@Query(value = "UPDATE User u SET u.datum_Nacteni_Uzivatele =
:datumNacteni")
void addUserLoadDate(@Param("datumNacteni") Date
datum_Nacteni_Uzivatele);
```

Dále se naváže na výstupní část aplikace neboli zobrazování dat uživateli. V tomto typu aplikace bude nejprve nutno vytvořit složku s názvem webapp do kořenového adresáře. Potom do ní vytvořit adresář WEB-INF a podadresář s názvem JSP. JSP (*„JavaServer Pages“*) je souborový formát, který podporuje dynamický vývoj webových stránek. Jde o kombinaci HTML tagů s jazykem Java. Když jsou všechny složky vytvořeny, bude nutno zadat potřebné informace do

konfiguračního souboru aplikace. K tomu budou sloužit tyto příkazy v souboru application.properties:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Tímto suffixem je jasně určeno, že všechny webové výstupy budou zpracovány ve formátu JSP.

Jako poslední z modelu MVC zbývá vytvoření kontroleru. V této aplikaci je nazván jako HomeController. Opět za použití anotace @Controller je vytvořena Spring bean. Dále za pomoci @Autowired propojíme kontroler s potřebnou UserRepository, která už byla vytvořena. Nyní lze z kontroleru přistupovat k datům aplikace. Je také nutné nastavit mapování webové adresy na danou metodu kontroleru. K tomu slouží anotace @RequestMapping s prázdným parametrem nebo případným názvem zadaným za IP adresou webového serveru v prohlížeči. Příkaz pro vyhledání tisíce uživatelů bude vypadat takto:

```
@RequestMapping("/")
public String defaultPage(Model model){

   users = userRepository.findThousandUsers();

   if (users.size() == 0){
      initDatabaseService.initDB();
      users = userRepository.findThousandUsers();
   }
   uzivatel.sortUsers(users);
   model.addAttribute("users", users);

   return "/home";
```

V této metodě je podmínkou, že pokud v databázi není nalezen ani jediný uživatel, bude provedeno opětovné načtení základního testovacího souboru. Tento soubor vloží do databáze deset tisíc uživatelských záznamů. List uživatelů je poté předán modelu aplikace, který zajišťuje přenos těchto dat do JSP stránky s názvem home.jsp. Obdobně je vytvořena druhá metoda, namapována na název *„loadmoredata“*, která na stejné stránce zobrazí všechny uživatelské záznamy (v

tomto testovacím případě je jich tedy deset tisíc). Tato metoda pro načtení všech uživatelů se liší od výše zmíněné jiným mapováním odkazu, nahrazením metody *„findThousandUsers()“* metodou *„findAll()“* a nakonec přidáním jediného řádku, který přes proměnnou userRepository zavolá metodu *„addUserLoadDate(new Date())“*. Tím aktualizuje datum načtení všech uživatelů v databázi.

4.4 Konfigurace virtualizovaného testovacího prostředí

Virtualizace bude tedy provedena za pomoci VMware technologie verze 6.7.0. Všechny vytvořené virtuální stroje v této virtualizaci budou obsahovat operační systém Ubuntu server 20.04 LTS, který je jednou z mnoha otevřených linuxových distribucí. Základem každé VMware virtualizace je již zmíněný ESXi hypervisor.

4.4.1 ESXi

Podle předchozího návrhu bude nastavena IP adresa prvního ESXi serveru na 192.168.212.10. Jakmile je nastavena IP adresa, stačí zadat přihlašovací údaje. Teď už lze přejít na nastavení serverů přes VMware grafické rozhraní ve webovém prohlížeči.

Při vytváření virtuálních webových serverů budou stanoveny hardwarové limity. Každý stroj bude mít v základu čtyři procesory, 8 GB paměti a 40GB HDD. Avšak tyto hodnoty budou v průběhu testování upraveny. Poté už stačí zadat adresářovou cestu k instalačnímu souboru operačního systému. V tomto případě Ubuntu Server 20.04 LTS. Webové servery budou mít nastaveny doménové jméno *„ubuntu-webappXX“*. Za XX je dosazeno číslo od 01 do 03 daného serveru.

4.4.2 Webové servery

V daném ESXi hypervisoru jsou již tedy nainstalovány tři webové servery. Všechny mají operační systém Ubuntu Server 20.04 LTS. Nejdříve je zapotřebí nainstalovat žádoucí aplikace. Za pomoci DHCP serveru ještě není nutné nastavovat žádnou statickou IP adresu, jelikož server automaticky obdrží adresu z VLAN 101. Tato VLAN připojuje server k internetu. Jakmile se tak stane, použijeme příkaz *„sudo apt update“*. To aktualizuje seznam nejnovějších verzí

aplikací. K této praktické části práce je využita databázová aplikace MySQL a webový server Tomcat.

Jakmile bude dokončena instalace těchto programů, bude vhodné tyto servery opět odpojit od internetu. Poté bude zadán příkaz *„ip a“*. Tak budou v terminálu zobrazeny všechny síťové karty serveru a jejich IP adresy. Když je lokalizováno číslo síťové karty, která je přirazena v ESXi VLAN 212, je přepsán tzv. „netplan“ konfigurační soubor:

```
network:
 version: 2
 renderer: networkd
 ethernets:
  ens192:
   dhcp4: no
   addresses:
    - 192.168.212.15/24
```

Ens192 je číslo síťové karty, které je pro každý webový server stejné. Jde o připojení přes ESXi k VLAN 212. IP adresa je samozřejmě na každém serveru jiná.

Webový server	Operační systém	IP adresa
ubuntu-webapp01	Ubuntu server 20.04	192.168.212.15
ubuntu-webapp02	Ubuntu server 20.04	192.168.212.16
ubuntu-webapp03	Ubuntu server 20.04	192.168.212.17

Tabulka 1 – Nastavení webových serverů pro virtualizované prostředí.

U webových serverů je důležité nastavit službu s názvem Tomcat. Jde o webový kontejner, na kterém bude spuštěna vytvořená *„userlistapp“* aplikace. Jsou zde dva způsoby, jak do daného Tomcat serveru umístit WAR soubor aplikace. První způsob je jednoduše zkopírovat tento soubor do složky *„webapps“,* umístěné v Tomcat adresáři. Druhý způsob, který bude využit v této práci, je otevření souboru *„tomcat-users.xml“* a dopsání konfigurace do oddílu *„<tomcat-users>“*.

<role rolename="admin-gui"/>

<role rolename="manager-gui"/>

<user username="XXXX" password="XXXX" roles="admin-gui,manager-gui"/>

Za „XXXX" bude dosazeno libovolné uživatelské jméno a heslo. Tím je docílen přístup k webovému GUI Tomcatu. Pokud je Tomcat spuštěn, zadáme IP adresu webového serveru do internetového prohlížeče v IMIT-Manage zařízení. Adresa na server ubuntu-webapp01 bude vypadat následovně: „*192.168.212.15:8080/manager/html*". Po úspěšném přihlášení zde lze najít přehledný grafický výstup, v němž je vidět, které aplikace jsou na daném Tomcatu spuštěny. Je tu také možnost nahrání daného WAR souboru z průzkumníka Windows a poté je možno vidět status nahrané aplikace.

4.4.3 Load-balancer

V této části bude nastaven haproxy load-balancer verze 2.0.13-2. V tomto případě jde o ochranu na aplikační sedmé vrstvě OSI modelu. Ten bude zachytávat pouze HTML požadavky. Znamená to tedy, že bude hledat packety na portu 80. Potom je podle vybraného algoritmu přepošle na již vytvořené webové servery z předešlé kapitoly. Pro tuto část byl vybrán HAproxy load-balancer. Ten však potřebuje operační systém, na kterém bude spuštěn. Linuxové platformy jsou tímto load-balancerem podporovány. Znovu tedy může být využit operační systém Ubuntu server, který jsme použili také na webové servery. Proto je na ESXi serveru vytvořen další virtuální stroj s názvem „*ubuntu-loadbalancer*". Jedinou změnou je, že tomuto load-balanceru budou přiděleny dvě síťové karty. Jedna bude propojovat webové servery na VLAN 212 s IP adresou 192.168.212.20, druhá bude vstupní branou pro útok vyslaný z virtuálních strojů na VLAN 213 a IP adresou 192.168.213.20.

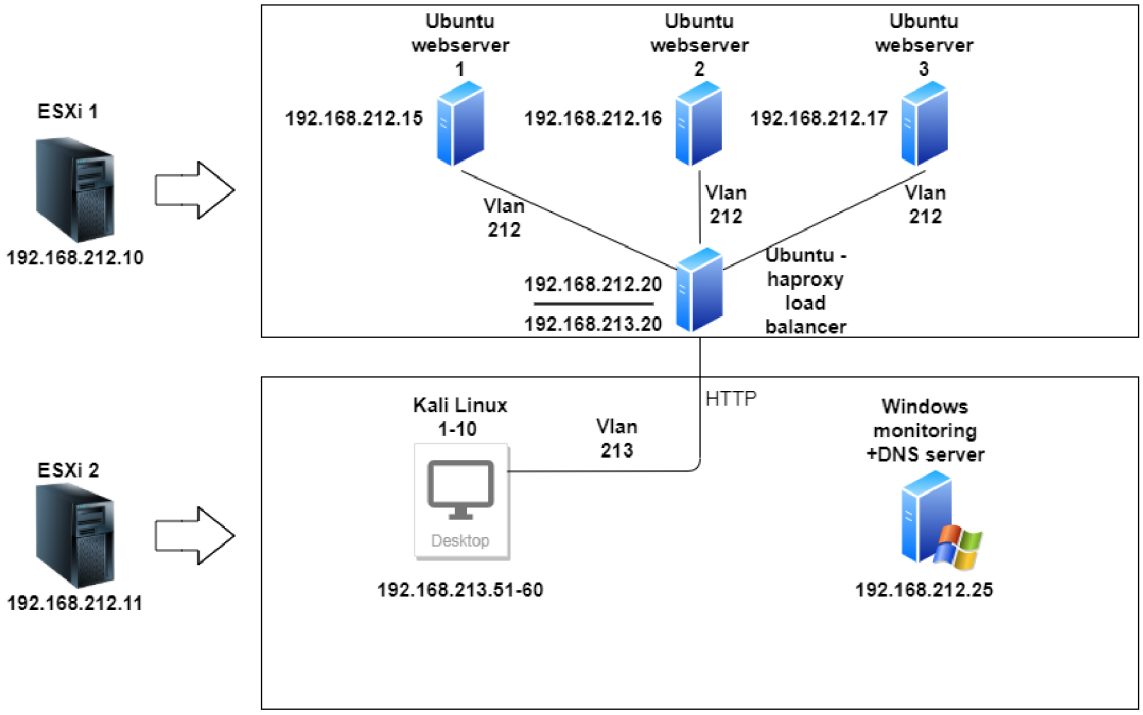
Po nainstalování HAproxy verze 2.0.13 je otevřen konfigurační soubor aplikace v adresáři *„/etc/haproxy/haproxy.cfg"*. Zde je vidět obecné a defaultní nastavení, kde je možné vytvořit a upřesnit třeba logování nebo maximální dobu připojení klienta. Při přípravě na DDOS útok je vhodné v tomto souboru přidat atribut „*maxconn*" a nenechávat ho na defaultní hodnotě. V našem testovacím

prostředí bude stačit nastavení hodnoty na 500 000. Poté už jsou v této práci přidány pouze dva odstavce konfigurace:

```
frontend Local_Server
        bind *:80
        option forwardfor
        mode http
        default_backend web_servers
backend web_servers
        mode http
        balance roundrobin
        server ubuntu-webapp01 192.168.212.15:8080
        server ubuntu-webapp02 192.168.212.16:8080
        server ubuntu-webapp03 192.168.212.17:8080
        option httpchk
```

Oddíl *„frontend Local_Server“* určuje situace příchozích požadavků, kterým bude load-balancer naslouchat. Řádek *„bind *:80“* je nastavením IP adresy, na které bude load-balancer odchytávat požadavky. Hvězdička definuje jakoukoliv IP adresu momentálně přidělenou serveru, v roli load-balanceru, za předpokladu použití protokolu 80. Metoda *„option forwardfor“* je způsob zachycení reálné klientské IP adresy, která vyslala HTML požadavek. Nakonec je definován oddíl *„backend web_servers“*. Jelikož jsou všechny webové servery vytvořeny s úplně stejně velkými hardwarovými prostředky, je vhodné použít load-balancingovou metodu Round Robin, která rovnoměrně rozloží zátěž mezi poskytnuté servery. Jiná situace by byla v případě nutnosti udržet v prohlížeči sessionu nebo databázové připojení na delší dobu. Potom by stálo za to použít jiný směrovací algoritmus. Toto prostředí však není pro složitější požadavky tohoto typu navržena. Nakonec je vidět zásobník serverů, mezi kterými může load-balancer vybírat. Zajímavé je číslo 8080, které je za každou adresou přidáno. Označuje port, na kterém funguje Tomcat u každého z těchto webových serverů. *„Option httpchk“* zaručuje, že load-balancer nepošle HTML požadavek na server, který nemá status

HTTP odezvy mezi 2xx-3xx. Příkazem „*systemctl status haproxy*“ lze opět ověřit, zda je tato služba spuštěna.



Obrázek 9 – Návrh virtualizovaného prostředí.

Finální virtualizované testovací prostředí je nakresleno na obrázku č. 9. Nachází se zde dva ESXi servery. První obsahuje tři webové servery a load-balancer. Ve druhém ESXi serveru je deset útočících desktopů a monitorovací Windows Server.

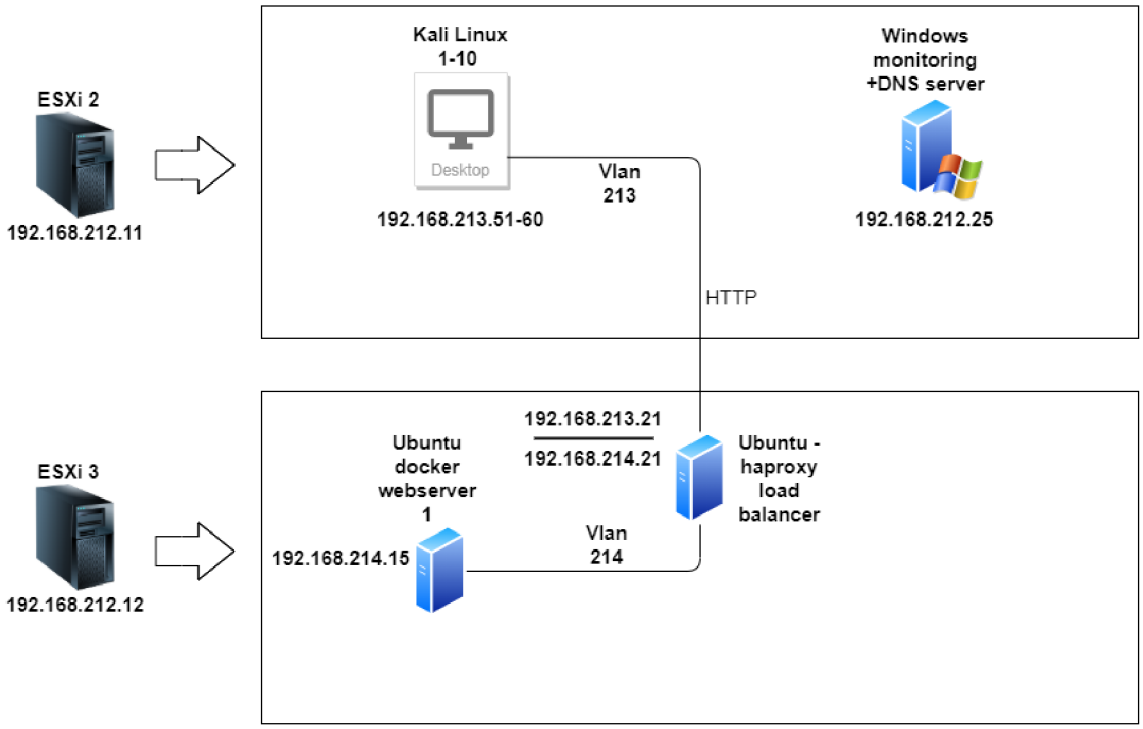
4.5 Návrh a konfigurace kontejnerového testovacího prostředí

Aby testování v této práci bylo objektivní, je nutné, aby se obě tato prostředí shodovala jak poskytnutými prostředky, tak strukturou. Budou zde jen logické změny a změny v souladu s vytvořením struktury v Dockeru.

Webový server	Operační systém	IP adresa
docker-webapp01	Ubuntu server 20.04	192.168.214.15

Tabulka 2 – Nastavení webového serveru pro kontejnerové prostředí.

Na rozdíl od virtualizovaného prostředí zde mohou být všechny tři stroje zastoupeny pouze v jednom serveru, který bude obsahovat více kontejnerů. Tento stroj bude mít opět operační systém Ubuntu Server 20.04 tak jako virtualizované prostředí.



Obrázek 10 – Návrh kontejnerového prostředí.

Na obrázku č. 10 je znázorněno finální kontejnerové testovací prostředí s využitím Dockeru. ESXi 2 server, odkud bude vyslán DDoS útok je stejný jako ve virtualizovaném prostřední. ESXi 3 naopak obsahuje pouze jeden webový server, na kterém bude spuštěn Docker s aplikacemi umístěnými do jednotlivých kontejnerů. Tedy jmenovitě Tomcat, Mysql a samotná webová aplikace poběží přímo v Dockeru, každý ve svém vlastním kontejneru. Přístup k nim bude umožněn upravením vstupního protokolu, na kterém bude daný kontejner naslouchat. Tomcat bude konkrétně umístěn ve třech kontejnerech, kterým budou přiděleny protokoly 8080, 8081 a 8082. Vzhledem k úspoře zdrojů bude tento stroj vytvořený opět na ESXi serveru. Ten bude fungovat na zcela jiném ESXi zařízení než virtualizované prostředí, avšak se stejně velkými hardwarovými prostředky. ESXi server pro kontejnerové testovací prostředí bude mít přidělenou IP adresu

192.168.212.12. Docker stroj bude mít svou vlastní VLAN s číslem 214. Do stejného ESXi bude také zkopírován load-balancer z virtualizovaného prostředí. Tomu se pouze musí změnit seznam adres serverů v haproxy.cfg:

server docker-webapp01 192.168.214.15:8080
server docker-webapp02 192.168.214.15:8081
server docker-webapp03 192.168.214.15:8082

V konfiguraci je vidět, že je poskytnuta stejná adresa odlišená pouze jiným číslem na konci adresy. Na těchto třech protokolech jsou nastaveny tři různé Tomcat servery, na kterých bude spuštěna testovací webová aplikace. Tu je pro kontejnerovou Docker strukturu zapotřebí upravit. Do kořenové složky je tedy přidán Dockerfile:

```
From tomcat:8.0.51-jre8-alpine
RUN rm -rf /usr/local/tomcat/webapps/*
COPY ./target/userlistapp.war /usr/local/tomcat/webapps/ROOT.war
CMD ["catalina.sh","run"]
```

Protože jsou všechny tři servery spuštěny na jediném stroji, je vhodné každé kontejnerové aplikaci přidělit vlastní databázové schéma (db_example1-3). Dále je nutné zjistit IP adresu kontejneru databázové mysql. Jelikož kontejner funguje na své vlastní virtuální síťovce, doména localhostu již není funkční. Za využití portaineru je zjištěna adresa databáze a v aplikaci upraven soubor application.properties:

```
spring.datasource.url=jdbc:mysql://172.17.0.6:3306/db_example1?createDatabaseIfNotExist=true&serverTimezone=UTC
```

4.6 Návrh a implementace testovacího scénáře

Úkolem této kapitoly bude navrhnout a připravit testovací scénář. Ten bude využit jak na virtualizované, tak i na kontejnerové prostředí. Půjde o vytvoření virtuálních strojů, z nichž bude vyslán DDoS útok na již předpřipravené prostředí. Mimo to je také nutné připravit stroj, na kterém se budou dané webové servery

monitorovat. Z tohoto monitorování bude poté vytvořen souhrn výsledků a vyvozen závěr. Na obrázku č. 11 testovacího prostředí níže byl přidán Windows Server. Odtud je možný přístup k webovému rozhraní vCenter serveru, ležícího mimo testovací infrastrukturu, který sleduje zátěž všech ESXi serverů a virtuálních strojů fungujících pod nimi.

Vzhledem k předvídané zátěži serverů, které budou pod útokem, je vhodné tyto stroje oddělit od těch, ze kterých bude vycházet útok. Stejně tak je nutné oddělit i stroj k monitorování webových serverů. Z toho důvodu bude vytvořen další ESXi server, fungující na jiném fyzickém stroji s přidělenou IP adresou 192.168.212.11.

Na tomto ESXi vznikne deset strojů, ze kterých budou vyslány útoky. Na takovéto testy je vhodné použít operační systém Kali Linux. Z něho bude využit program SlowHttpTest, který má několik útočných metod. K této práci lze využít tzv. slowloris útok. Jeho funkcionalita spočívá v zasílání pomalých paketových hlaviček. Tváří se tedy jako klient s velmi pomalým připojením. Po navázání kontaktu se pak snaží toto připojení udržet co nejdelší dobu za účelem vyčerpání limitu možných připojení napadených serverů.

Ve VMwaru je vytvořeno deset virtuálních Kali Linux strojů. Každý z nich má přidělenou síťovou kartu na všechny VLAN a IP adresu v rozsahu 192.168.XXX.51-60, kde XXX je číslo VLAN.

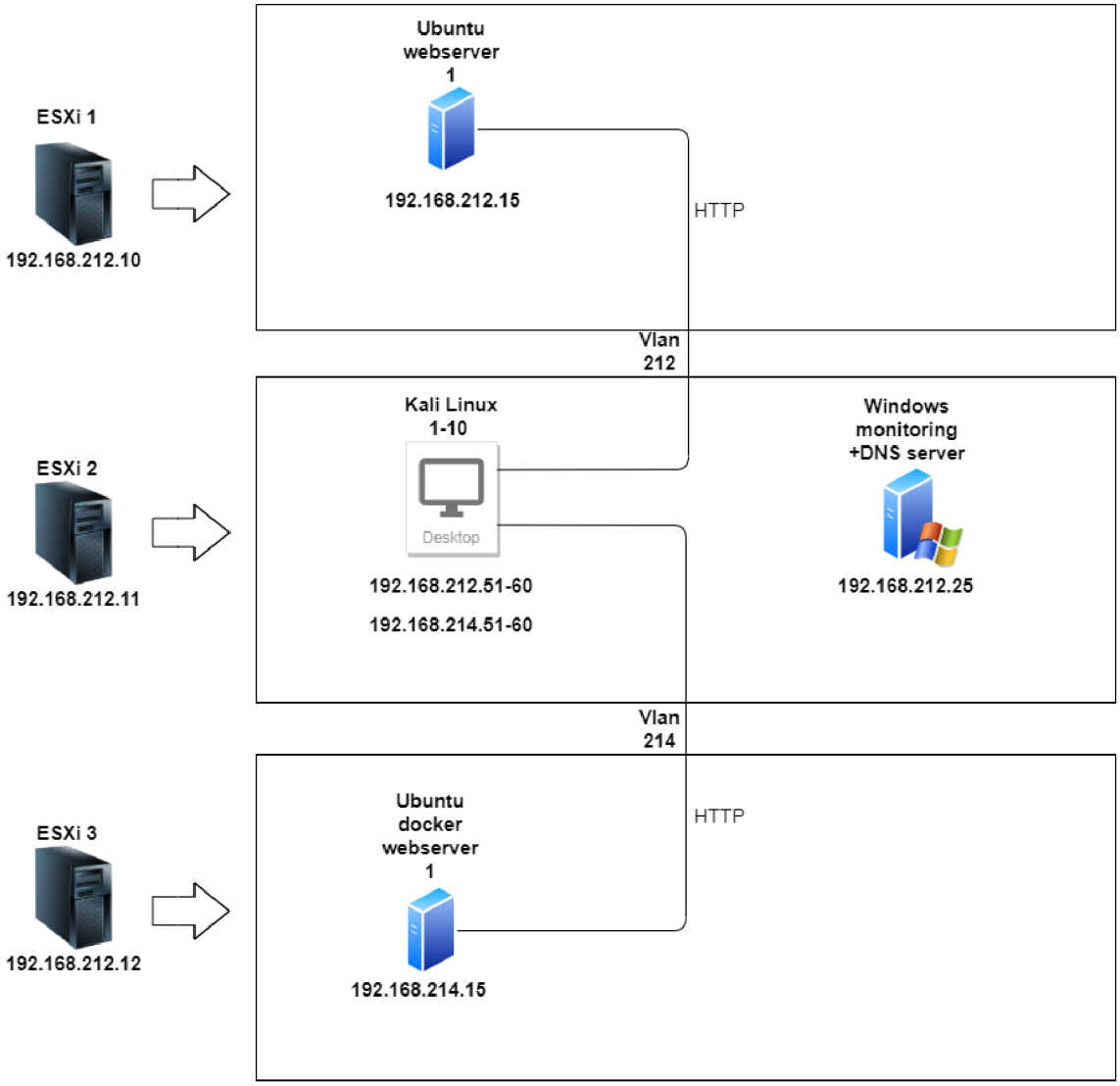
Vždy před zahájením testovacího útoku bude vhodné cílový virtuální webový server restartovat, aby byl test objektivní a aby nebyl webový server ovlivněn z průběhu předchozího testu.

Test je považován za úspěšný, pokud bude webová aplikace dostupná po celou dobu průběhu testování. V případě výpadků se bude upravovat konfigurace testovacího prostředí.

4.6.1 Test 1

V prvním testu bude útok proveden přímo na jednotlivý webový server. Tomu budou zprvu sníženy hardwarové prostředky. V kontejnerovém prostředí bude proveden útok se stejnými parametry přímo na jeden z kontejnerů. V obou prostředích bude nastaven virtuálnímu stroji pouze 1 CPU a 8GB RAM. Útok bude

nastaven na dobu 30 minut (1800 sekund) s maximálním počtem 28 000 připojení za každý Kali Linux desktop.



Obrázek 11 – Dočasné testovací prostředí pro test 1-3.

Na obrázku č. 11 lze vidět konfiguraci testovacího prostředí pro vyzkoušení jednoho virtualizovaného serveru, nastaveného v rámci ESXi 1, na kterém je spuštěna služba MySQL a Tomcat. Na ESXi 3 je potom nastaven druhý webový Docker server, na kterém se nachází Tomcat v rámci jednoho kontejneru a MySQL databáze v rámci kontejneru druhého. Na obě zařízení je nainstalován Ubuntu Server 20.04 LTS a na oba servery bude zaútočeno z ESXi 2, kde se kromě serveru pro monitorování zátěže nachází deset desktopů s Kali Linux operačním systémem

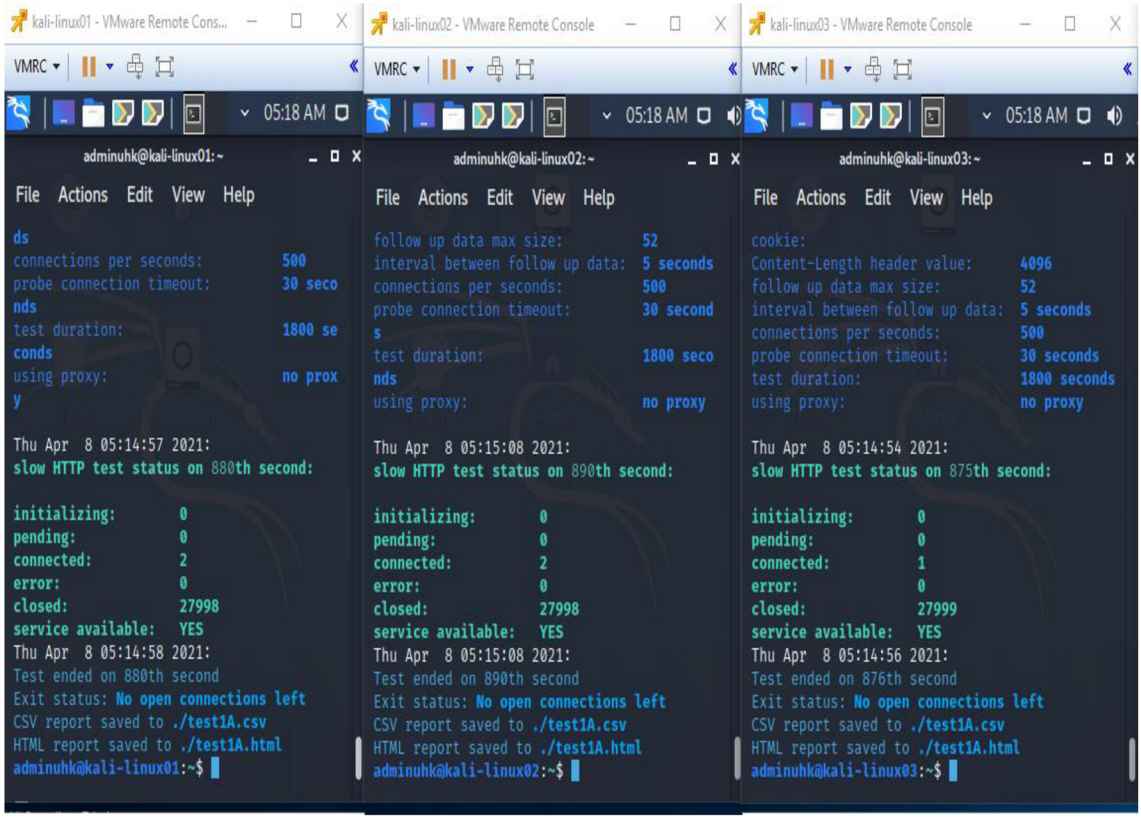
a aplikací SlowHttpTest. Terminálové příkazy k zahájení DDoS útoku budou vypadat takto:

slowhttptest -c 28000 -H -g -o ./test1A -i 5 -l 1800 -r 500 -t GET -u http://192.168.212.15:8080/userlistapp/loadmoredata -x 24 -p 30

slowhttptest -c 28000 -H -g -o ./test1B -i 5 -l 1800 -r 500 -t GET -u http://192.168.214.15:8080/userlistapp/loadmoredata -x 24 -p 30

První příkaz slouží k napadení virtualizovaného prostředí a druhý pro napadení kontejnerového prostředí. Důležité parametry stojící za zmínku jsou tyto:

- **-c** Celkový počet provedených připojení.
- **-H** Zvolení testovacího modu. V tomto případě útok s pomalými paketovými hlavičkami, tedy tzv. Slowloris útok.
- **-g** Zapnutí možnosti generování statistik.
- **-i** Interval v sekundách mezi zasíláním dat.
- **-l** Délka testu v sekundách.
- **-r** Počet možných připojení za sekundu
- **-t** Nastavení HTTP požadavků na GET.
- **-u** Cílová adresa útoku.
- **-p** Doba v sekundách, kdy se čeká na odpověď. Po vypršení limitu se server označí jako nedostupný.



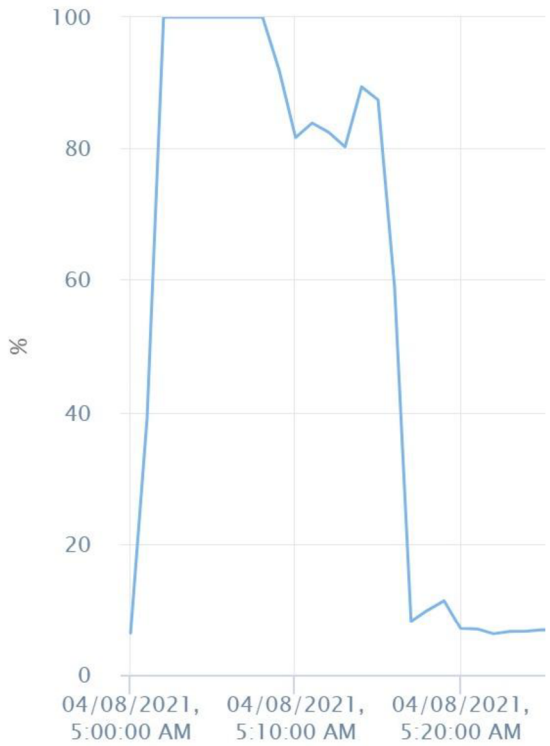
Obrázek 12 – Konzole několika Kali Linux desktopů po prvním testu na Ubuntu webserver 1.

Obrázek 13 – Grafický výstup prvního testu SlowhttpTest aplikace z Kali Linux 01 desktopu.

Z výstupu konzole tří Kali Linux desktopů lze vidět, že test skončil dříve, než bylo plánováno (zhruba po čtrnácti minutách), a všechna připojení byla mezitím ukončena. Stejně tak dopadlo i kontejnerové prostředí.

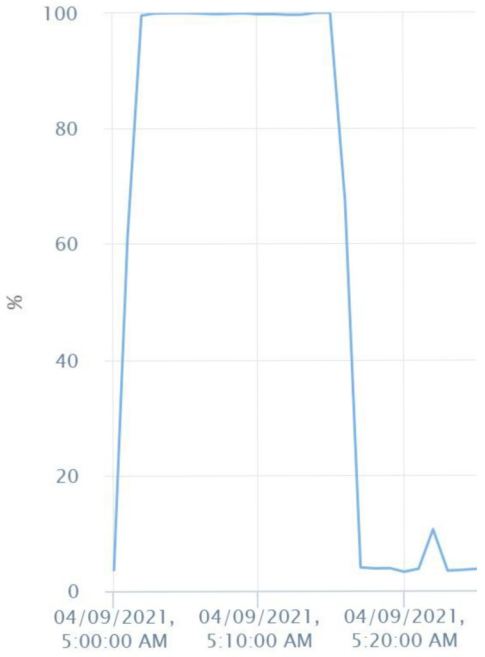
Grafický výstup útoku z Kali Linux 01 desktopu se dělí na levou a pravou část. V levé části je útok na Ubuntu webserver 1. V pravé části je útok na webový Docker server. Z grafů lze vyčíst, že server měl potíže již po první minutě útoku, kdy dosáhl zhruba 40 000 připojení (pro zlehčení je číslo vynásobeno počtem vytvořených Kali Linuxových desktopů, tedy deseti). Poté byl aplikací označen za nedostupný, jelikož zřejmě některé požadavky nedostávaly odpověď déle než 30 sekund. Tomu nasvědčuje i rostoucí modrá křivka označující počet zavřených připojení způsobených webovým serverem.

V průběhu testu je vidět, že se webový server, po zavření velkého počtu připojení, stal několikrát opět dostupným, ale nakonec už neměl vůbec žádné navázané spojení s útočícími desktopy. Za zmínku také stojí, že po dokončení testu byla testovací webová aplikace přístupná z internetového prohlížeče.

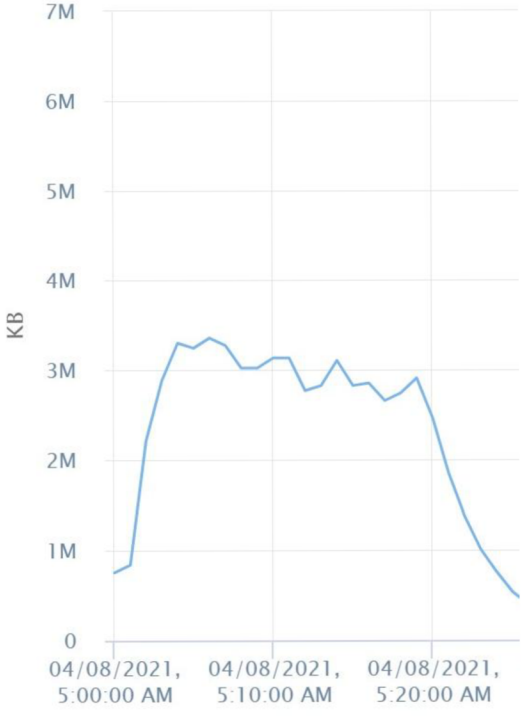


Obrázek 14 – Využití CPU Ubuntu webserver 1 při prvním testu.

Z obrázku č. 14 využití CPU na serveru lze vidět obrovskou zátěž v době útoku. Útok probíhal od 5:00 hodiny ranní do zhruba 5:15. Lze zde jasně vyčíst, že zátěž na daný server byla až příliš velká (je vidět souvislé až několikaminutové 100% vytížení procesoru), a zřejmě proto server nestíhal zpracovat všechny příchozí HTTP požadavky.



Obrázek 15 – využití CPU Docker webserveru při prvním testu.



Obrázek 16 – Využití paměti Ubuntu webserver 1 při prvním testu.

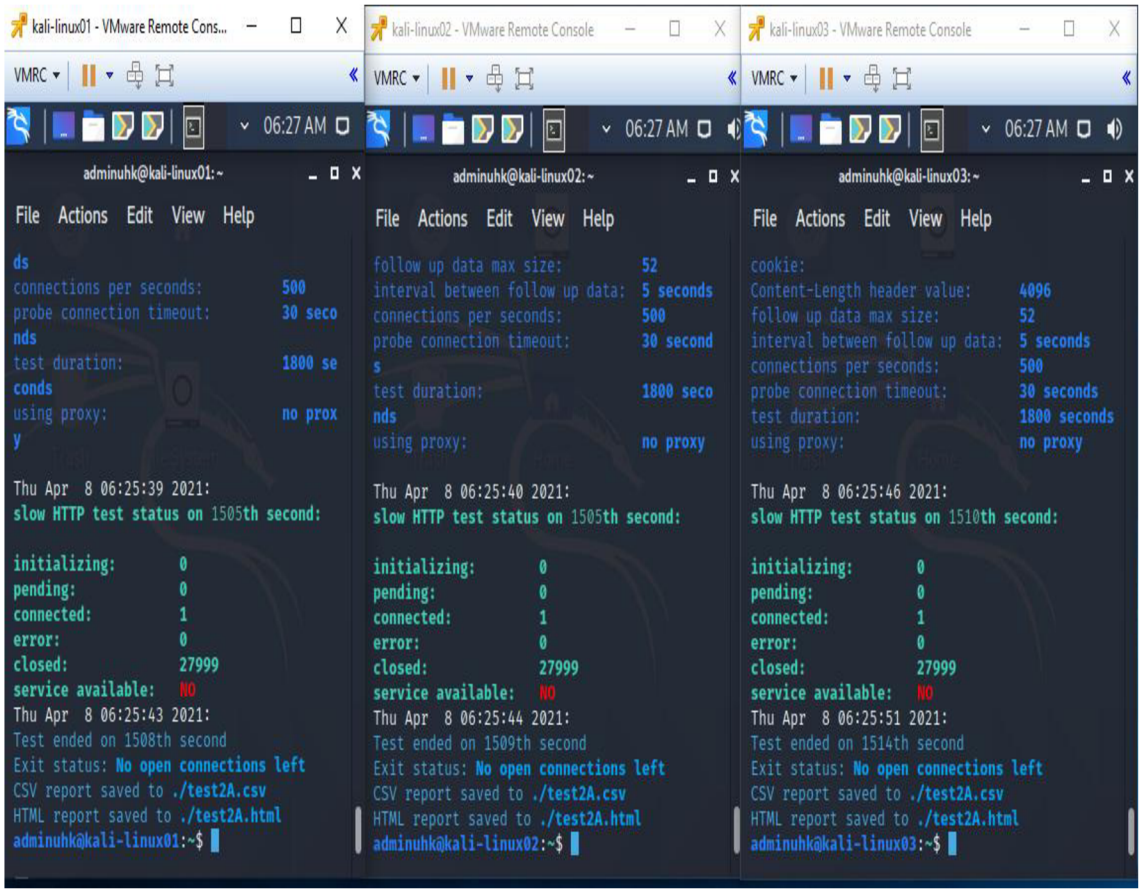
Křivka Docker serveru využití CPU z obrázku č. 15 vypadá velmi podobně jako využití u Ubuntu webserveru 1 na obrázku č. 14. A jelikož využití paměti nedosáhlo limitu (tedy 8GB), bude vhodné zkusit navýšit pouze množství CPU přidělených oběma webovým serverům.

TEST 1	Délka testu	Počet připojení na konci testu	Úspěch testu
VMware	890s	0 z 280 000	Neúspěšný
Docker	895s	0 z 280 000	Neúspěšný

Tabulka 3 – Shrnutí prvního testu.

4.6.2 Test 2

V tomto testu tedy bude proveden stejný útok. Budou zde však navýšeny hardwarové prostředky na 4 CPU. Všechno ostatní zůstane stejné.

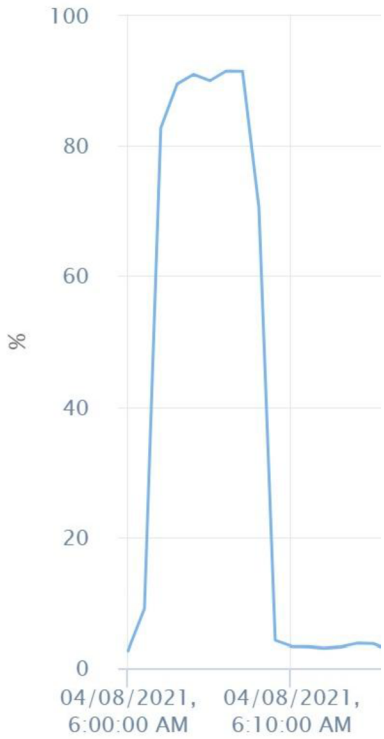


Obrázek 17 – Konzole několika Kali Linux desktopů při druhém testu na Ubuntu webserver 1.

Obrázek 18 – Grafický výstup druhého testu SlowHttpTest aplikace z Kali Linux 01 desktopu.

Výsledek druhého testu ukazuje, že zvýšení hardwarových zdrojů způsobilo delší výdrž dostupnosti webové aplikace. V prvním testu spadla aplikace poprvé již zhruba po minutě od započetí útoku. V tomto druhém testu se první výpadek vyskytl až po asi 100 sekundách. Na rozdíl od prvního testu je už zřejmé, že se webová aplikace nebo spíše samotný Tomcat po prvním selhání již nevzpamatoval. Po ukončení tohoto testu je totiž nepřístupná nejen testovací aplikace, ale i webový manažer Tomcatu.

Útok provedený na Docker sice vydržel celých 1800 sekund (o necelých 300 sekund déle než na Ubuntu webserver 1) a test tedy dosáhl nastaveného času, ale vzhledem k tomu, že se aplikace podle grafu nejevila jako dostupná a ani nepřibývala nově otevřená připojení mezi serverem a Kali Linux desktopem, je tento test při konfiguraci druhého testu považován za neúspěšný.



Obrázek 19 – Využití CPU Ubuntu webserver 1 při druhém testu.

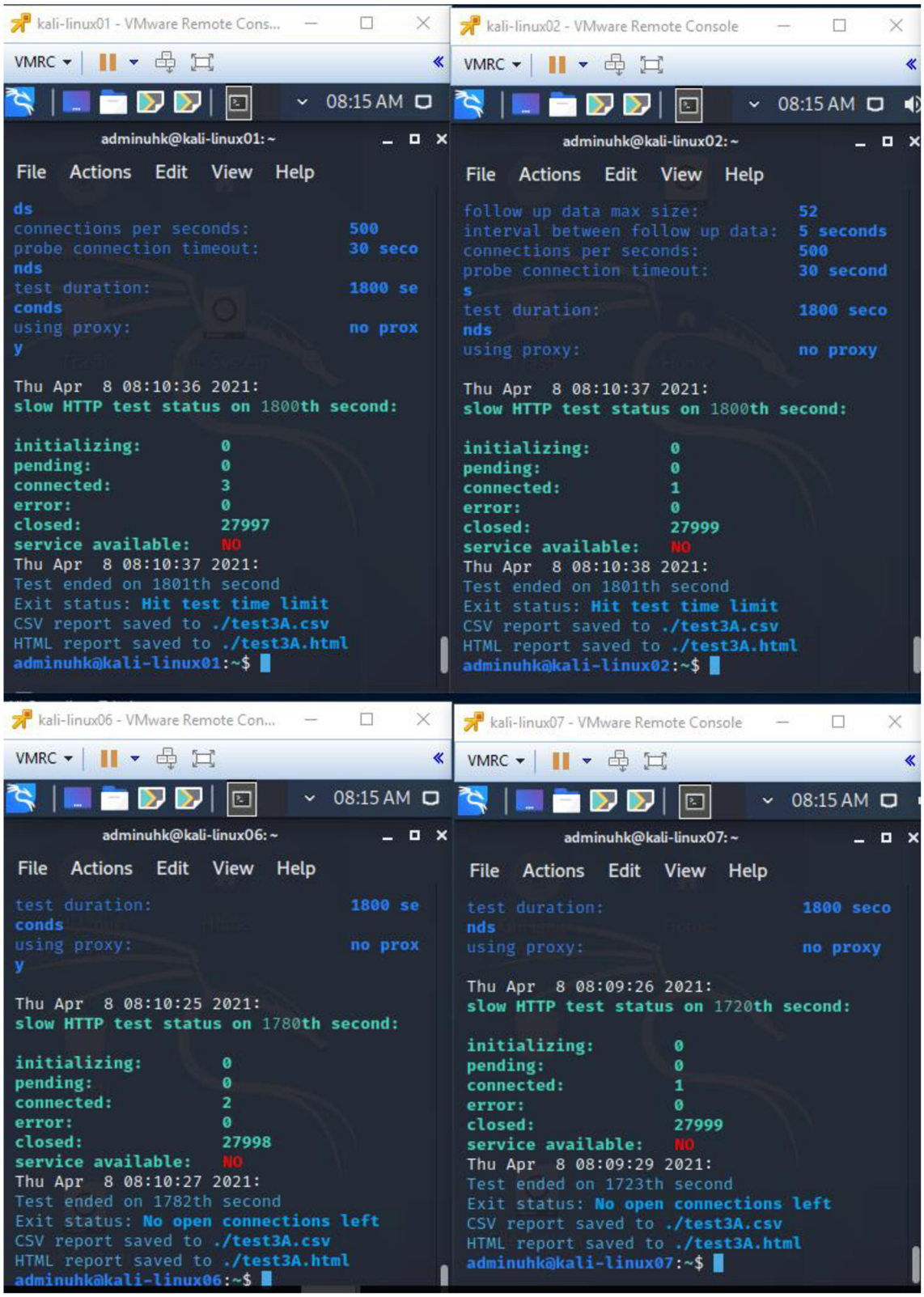
Opět se využití CPU zdá celkem významné, a to i na Docker serveru. Dosahuje, až 90% využití zvýšeného množství CPU. Využití paměti se zdá stále v normě. To znamená, že po celou dobu testu se využití paměti nepřiblížilo limitu nastaveného pro tento test. Proto bude vhodné do dalšího testu opět pouze navýšit množství CPU.

TEST 1	Délka testu	Počet připojení na konci testu	Úspěch testu
VMware	1510s	0 z 280 000	Neúspěšný
Docker	1800s	114 z 280 000	Neúspěšný

Tabulka 4 – Shrnutí druhého testu.

4.6.3 Test 3

Znovu bude proveden stejný útok na obě testovací prostředí. Každému prostředí bude tentokrát zvýšen počet CPU na 24. Ostatní hardwarové prostředky zůstávají stejné.



Obrázek 20 – Konzole několika Kali Linux desktopů při třetím testu na Ubuntu webserver 1.

Na obrázku č. 20 terminálových výstupů lze zpozorovat, že na některých desktopech byl test ukončen předčasně z důvodu zavření všech připojení k serveru. Oproti minulému testu se naopak v tomto případě útok na kontejnerové prostředí zastavil na každém desktopu jinak, a to na 1400 až 1600 sekundách. Jako podstatný údaj bych tedy zase označil počet zavřených připojení.

Obrázek 21 – Grafický výstup třetího testu SlowHttpTest aplikace z Kali Linux 01 desktopu.

Obrázek 22 – Využití CPU Ubuntu webserver 1 při třetím testu.

Obrázek 23 – Využití CPU Docker webserveru při třetím testu.

Při zkoumání grafického výstupu SlowHttpTest aplikace se může zdát, že kontejnerové prostředí dopadlo o něco lépe, jelikož první výpadek mělo o chvilku později než Ubuntu webserver 1. Avšak v testu stejně selhalo. Tudíž je zřejmé, že zvýšení množství CPU v odolnosti proti danému DDoS útoku nepomůže. Tohle tvrzení podporuje i graf využití CPU na Ubuntu webserveru i Docker webserveru. Objevil se zde však jeden vedlejší efekt, a to u Ubuntu webserveru 1. Útok byl

ukončen v čase 8:15 ráno, přitom využití CPU bylo zacyklené až do chvíle restartování virtuálního stroje v čase 8:32.

Do dalšího testu tedy budou upravena obě testovací prostředí a vráceny hardwarové zdroje na původní hodnoty.

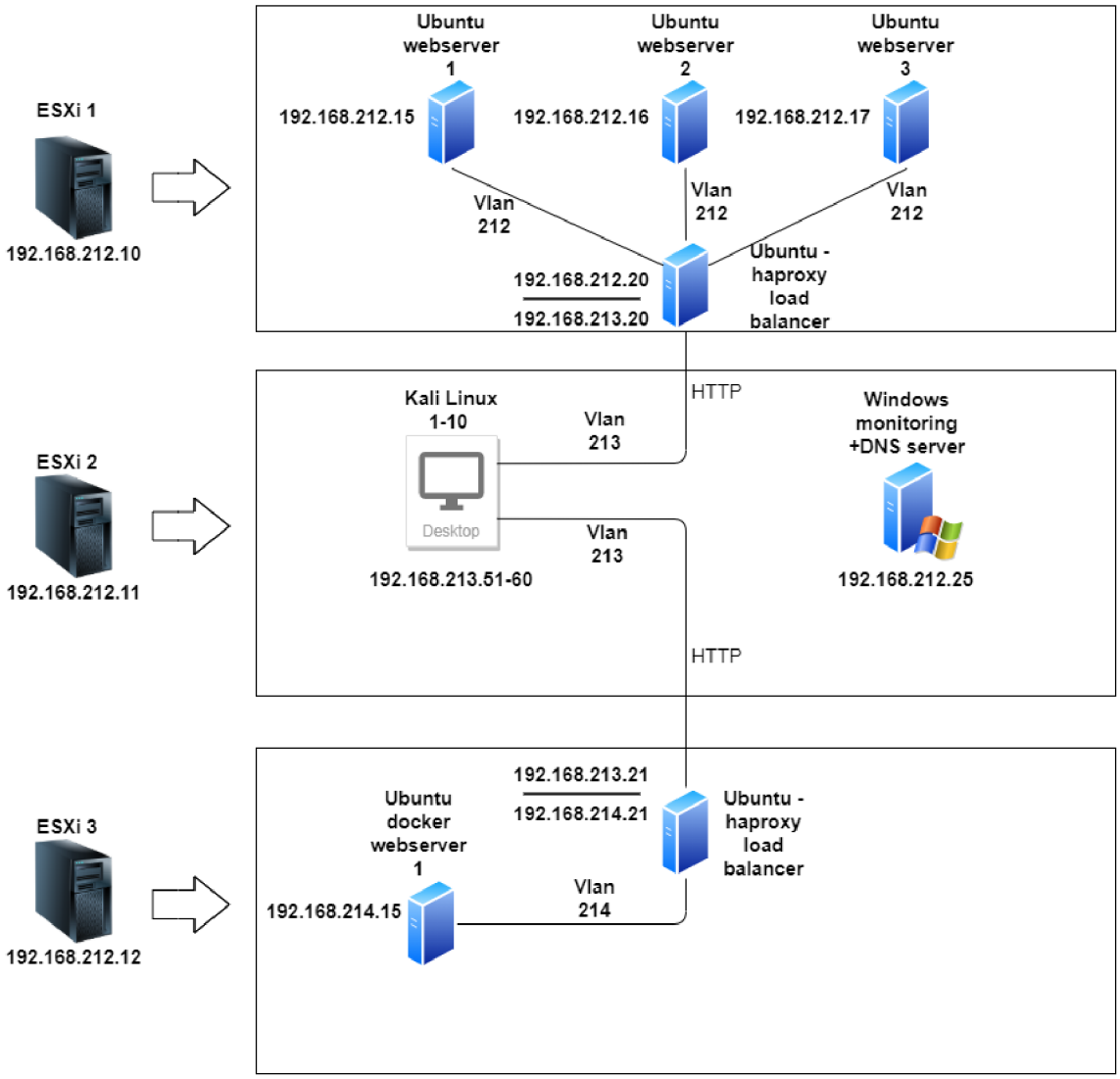
TEST 1	Délka testu	Počet připojení na konci testu	Úspěch testu
VMware	1800s	23 z 280 000	Neúspěšný
Docker	1635s	0 z 280 000	Neúspěšný

Tabulka 5 – Shrnutí třetího testu.

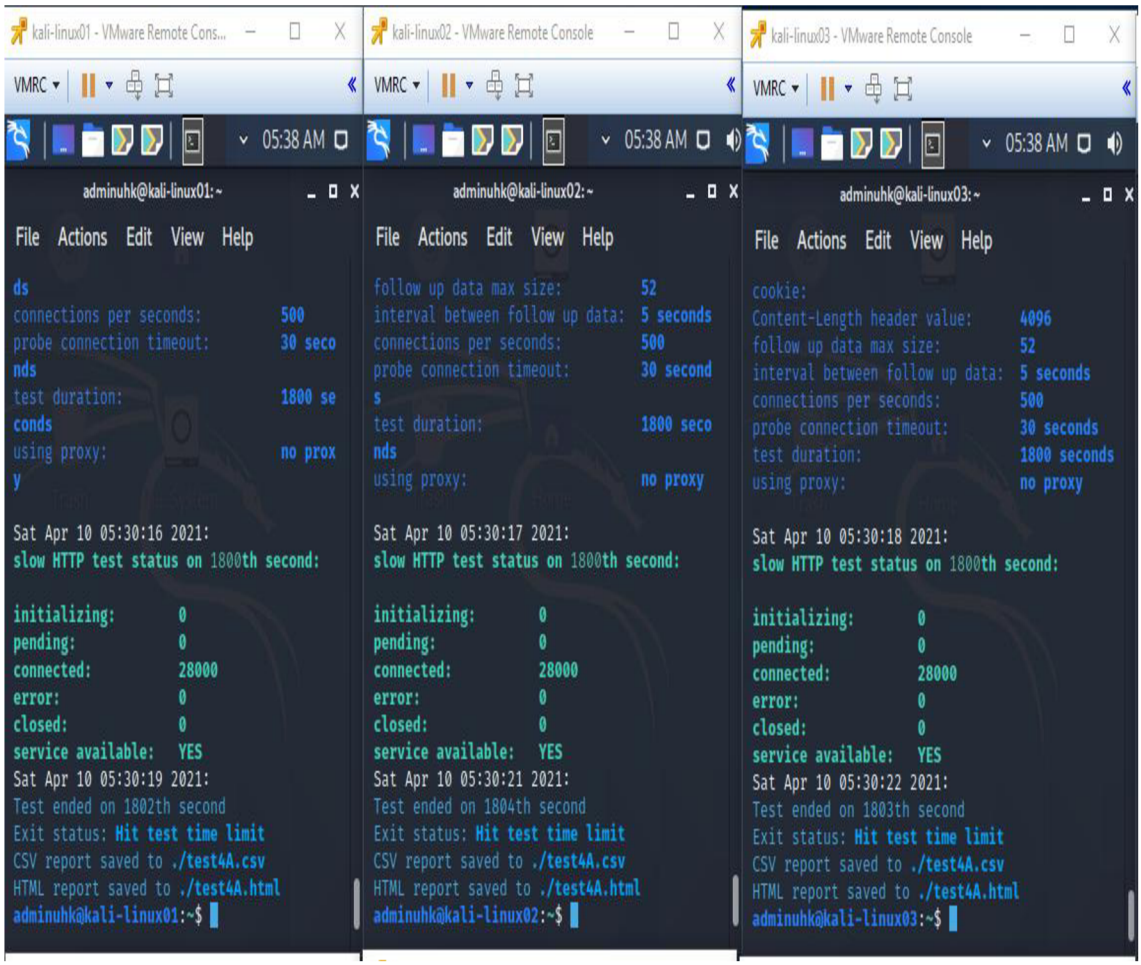
4.6.4 Test 4

Útok bude proveden se sníženými hardwarovými prostředky a bude veden přes load-balancer.

Útok číslo 4 bude stejný jako v předchozích případech. Tentokrát se však rozšíří virtualizované prostředí o další dva servery a load-balancer. Každému webovému serveru v tomto prostředí bude přiděleno 1 CPU a 8GB RAM. Load-balanceru budou přiděleny 4 CPU a 8GB RAM. Kontejnerové prostředí bude rozšířeno o další dva Tomcat kontejnery a samostatný load-balancer se stejnou konfigurací jako load-balancer ve virtualizovaném prostředí na ESXi 1. Změna oproti virtualizovanému prostředí bude v přidělení Docker webserveru tří CPU místo jednoho. Tím budou hardwarové prostředky vyrovnány v obou prostředích.

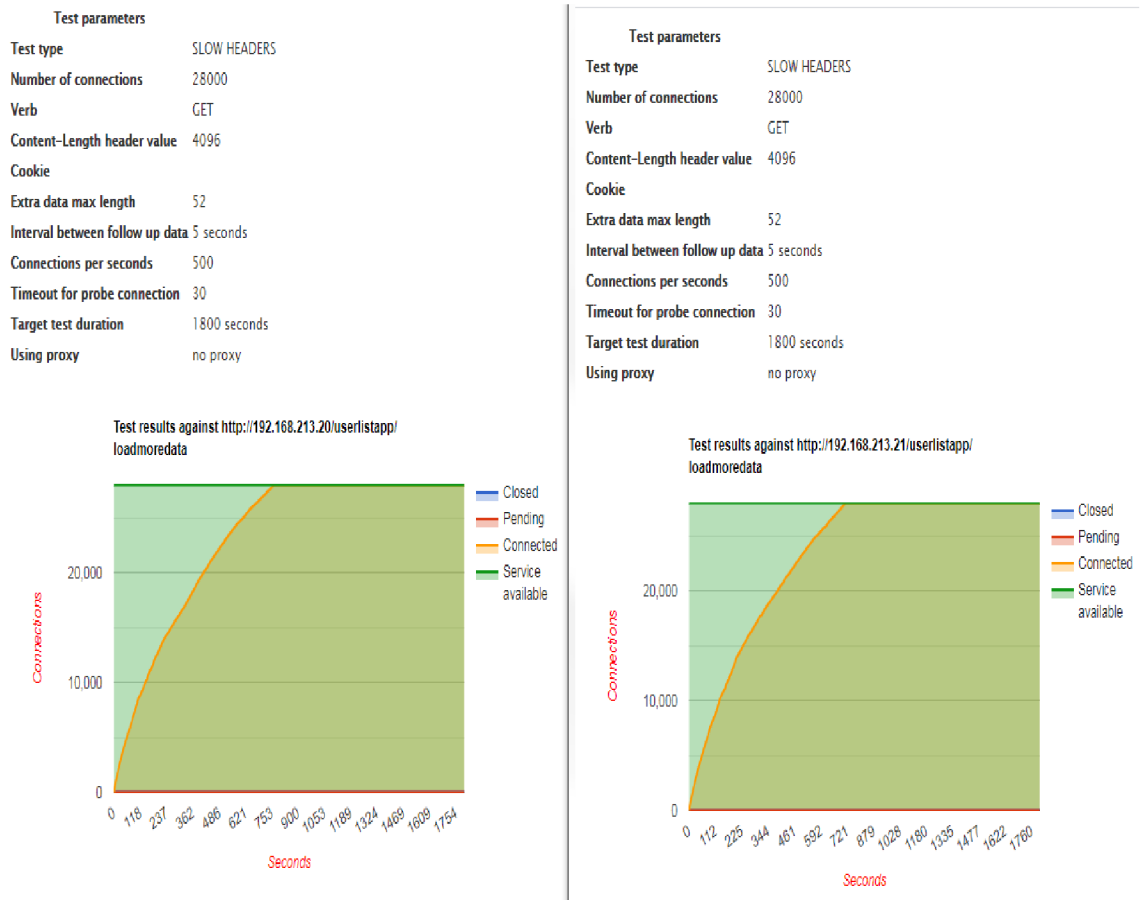


Obrázek 24 – Finální testovací prostředí pro test 4.



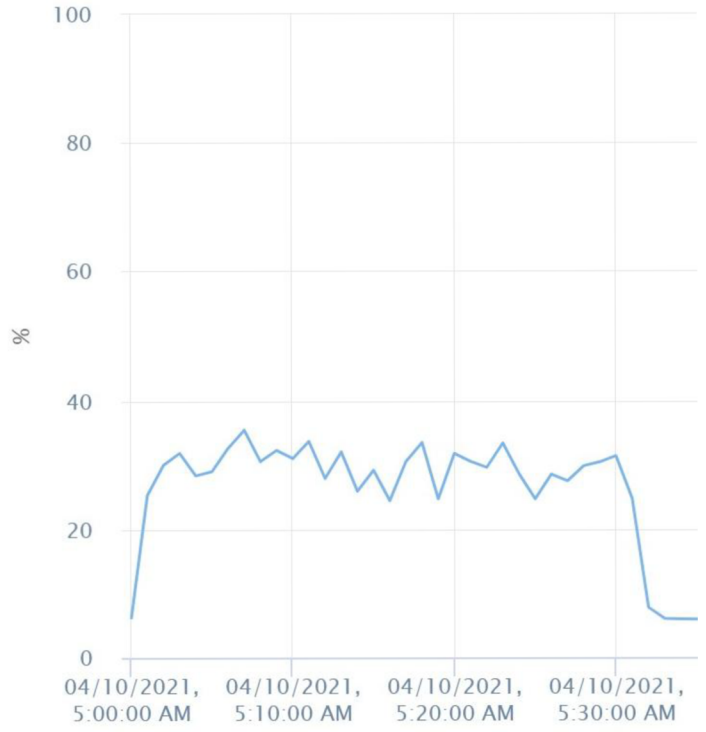
Obrázek 25 – Konzole několika Kali Linux desktopů při čtvrtém testu na Ubuntu webserver 1

Z konzole lze zpozorovat, že při dokončení testu byl připojen maximální počet vytvořených připojení k serveru (280 000 připojení). Dobré také je, že test byl spuštěn celých naplánovaných 30 minut. To, že nebylo ani jedno připojení zavřeno a aplikace byla na konci testu dostupná, znamená, že test byl konečně úspěšný. Konzole z útoku na kontejnerové prostředí dopadla úplně stejně.

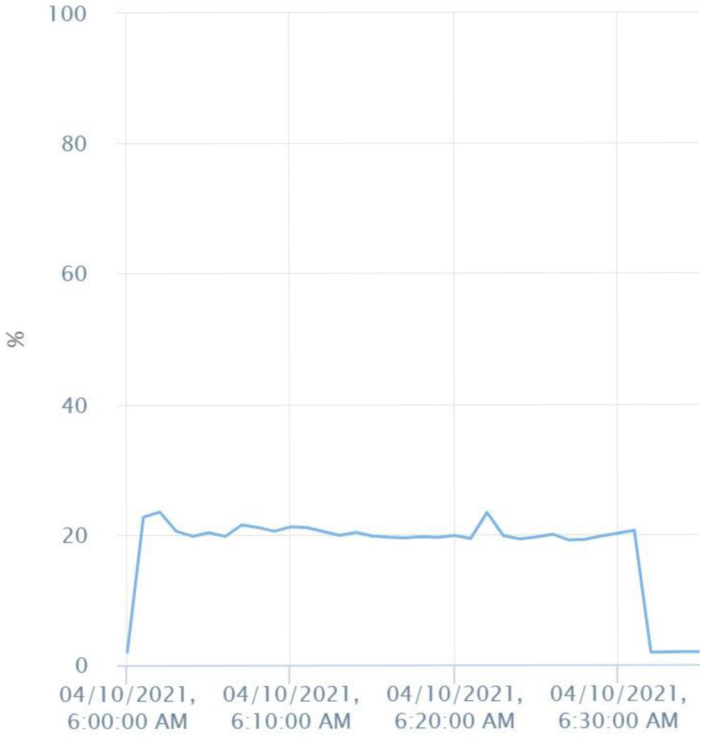


Obrázek 26 – Grafický výstup čtvrtého testu SlowHttpTest aplikace z Kali Linux 01 desktopu.

Zelené pozadí potvrzuje, že aplikace byla dostupná po celou dobu testu. K maximálnímu navázání připojení došlo již dříve než v polovině trvání útoku, a to v obou testovacích prostředích. Ani jedno připojení nebylo zavřené a ani jedno nebylo zařazeno do fronty z důvodu, že by server nestíhal daný požadavek zpracovat.



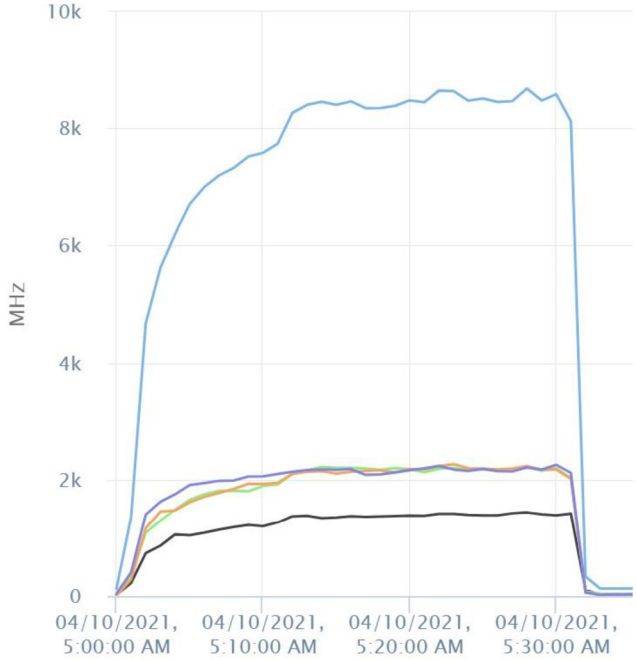
Obrázek 27 – Vytížení CPU z virtualizovaného prostředí Ubuntu webserver 1 při čtvrtém testu.



Obrázek 28 – Vytížení CPU webového serveru z kontejnerového prostředí při čtvrtém testu.

Po celou dobu třicetiminutového testu lze z obrázků č. 27 a č. 28 vytížení CPU na obou webových serverech poznat, že s jedním CPU měly zatížení mezi 20 až 40 procenty. Obdobně vypadá i vytížení virtualizovaného webserveru 2 a 3. Lze tedy předpokládat, že většinu práce vyřešil přidaný load-balancer.

Obrázek 29 – CPU zatížení load-balanceru virtualizovaného prostředí při čtvrtém testu.

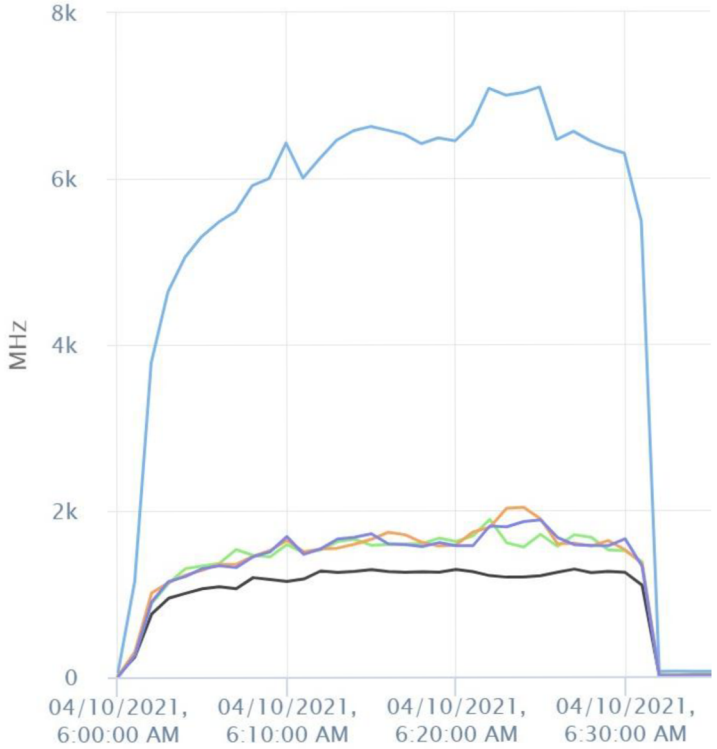


Obrázek 30 – Výkon všech CPU v MHz load-balanceru ve virtualizovaném prostředí při čtvrtém testu.

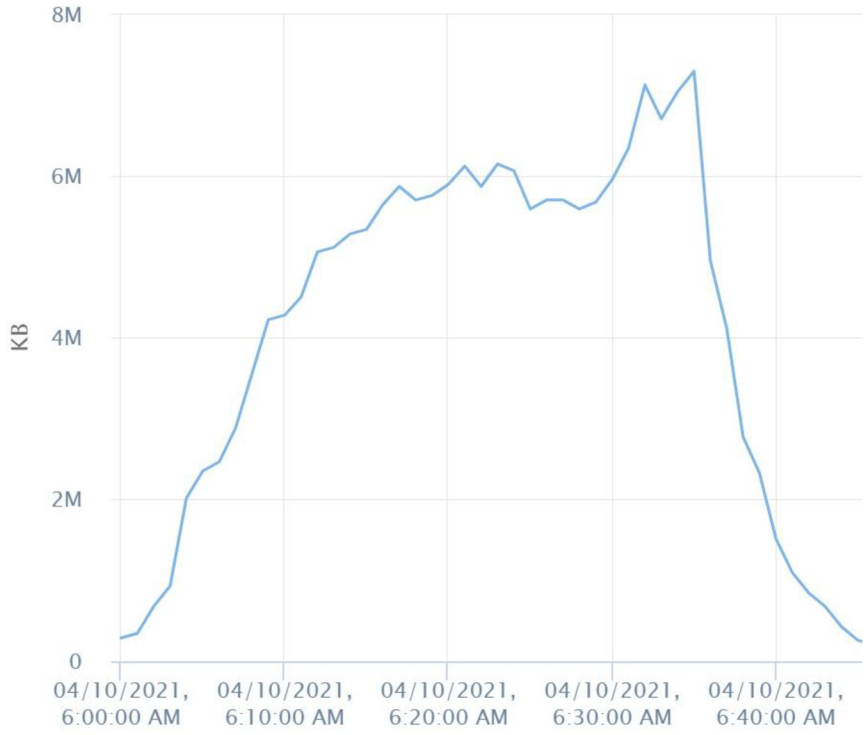
Obrázek 31 – Využití paměti load-balanceru ve virtualizovaném prostředí při čtvrtém testu.

Dle předchozích grafů lze zpozorovat, že využití load-balanceru bylo téměř na limitu. Využití procesoru dosahovalo až 80 procent po většinu doby útoku. Z grafu využití CPU v jednotkách MHz je vidět, že přidané 4 procesory dosahovaly výkonu nad 8 GHz. Využití RAM na konci testu bylo chvilkově blízko limitu 8GB RAM, který byl původně load-balanceru nastaven. Ten to však vydržel a útok přestál. Pro porovnání je zde přiloženo i vytížení load-balanceru z kontejnerového testovacího prostředí.

Obrázek 32 – Využití CPU load-balanceru v kontejnerovém prostředí při čtvrtém testu.



Obrázek 33 – Využití CPU v MHz load-balanceru v kontejnerovém prostředí při čtvrtém testu.



Obrázek 34 – Využití paměti load-balanceru v kontejnerovém prostředí při čtvrtém testu.

Podle grafů využití CPU load-balanceru v kontejnerovém prostředí je v některých časových úsecích vidět až o 20 procent menší zátěž než u load-balanceru virtualizovaného prostředí. Je možné, že v tomto případě se zde odráží o něco menší zátěž CPU Docker webserveru oproti virtualizovanému webserveru. Avšak využití paměti RAM obou load-balancerů má velmi podobnou křivku. V rámci těchto testů lze tento jev považovat jako zanedbatelný rozdíl, jelikož neměl žádný vliv na výsledek testu.

TEST 1	Délka testu	Počet připojení na konci testu	Úspěch testu
VMware	1800s	280 000 z 280 000	Úspěšný
Docker	1800s	280 000 z 280 000	Úspěšný

Tabulka 6 – Shrnutí čtvrtého testu.

4.6.5 Shrnutí testů

Celkem byly provedeny čtyři testy. Test 1, 2 a 3 má své vlastní upravené prostředí, ve kterém se útočí pouze na samostatný webový server. V těchto testech

lze zpozorovat nebezpečnost DDoS útoku. Ani v jednom z těchto testů nebyla dosažena stabilita webové aplikace. Všechny tyto testy ukázaly, že rozšíření hardwarových zdrojů nemá proti DDoS útoku očekávaný pozitivní účinek. Obě prostředí založené na VMware nebo Docker platformě měly ve všech testech stejnou úspěšnost. Jediné, co pomohlo až ve čtvrtém testu, bylo přidání load-balanceru s dalšími webovými servery (v případě kontejnerového prostředí přidání kontejnerů), mezi které byla rozdělena zátěž. S takovou konfigurací již webová aplikace vydržela nápor DDoS útoku a dokonce bylo připojeno všech 280 000 připojení po celou dobu průběhu testu.

5 Závěry a doporučení

V teoretické části byla představena podstata virtualizace a jejího různorodého využití. Dále zde byla rozebrána kontejnerizace, tedy její hlavní rozdíly oproti virtualizaci. Na tuto kapitolu poté navazují základy load-balancingu a jeho funkcionality. V poslední kapitole teoretické části je představeno nebezpečí pro webové aplikace. Takovou hrozbou je myšlen DDoS útok.

V praktické části této práce byla vytvořena dvě testovací prostředí. Obě prostředí obsahovala svůj vlastní ESXi server. V těch byly nastaveny virtuální stroje odpovídající virtualizační a kontejnerové technologii. Poté byla za pomoci služby Tomcat implementována webová aplikace v jazyce Java, která byla napsána speciálně pro tato testovací prostředí. Vytvořená webová aplikace využila MySQL databáze a načetla z ní 10 000 záznamů, které poté seřadila podle atributu data narození uživatele. Nakonec byl také nakonfigurován haproxy load-balancer. Ten však byl využit až v posledním testu.

Pro testování odolnosti těchto serverů a aplikace samotné byla využita aplikace SlowHttpTest, která vytvářela tzv. „slowloris“ DDoS útok. Tento útok byl rozdělen do čtyř kategorií. Test 1-3 útočil na samotné webové servery a poukázal na slabinu webových aplikací běžících ve službě Tomcat. Tuto slabinu se nepodařilo vyřešit ani výrazným zvýšením hardwarových prostředků. Pro test č. 4 byl tedy do obou testovacích prostředí přidán a nakonfigurován haproxy load-balancer spolu s dalšími dvěma webovými servery (v případě Dockeru kontejnery), které pomohly snížit nápor útoku.

V testech bylo zjištěno, že odolnost vůči DDoS útokům není zlepšena zvýšením hardwarových prostředků. Větší přínos zde přineslo obohacení infrastruktury o load-balancer se kterým dokázala obě prostředí útoku odolat. Dalším faktorem, který zde byl předveden, byla jednoduchost vytvoření DDoS útoku. Bez potřebné ochrany dokázal zablokovat webový server z pouhých deseti desktopů. Většinou má útočník k dispozici daleko více strojů. Z tohoto důvodu je vhodné hledat a testovat další možné ochrany proti různým digitálním útokům. Například doporučení do budoucna, když už je nastaveno testovací prostředí, schopno odolat DDoS útoku, bylo by vhodné zkusit přidat server, který by měl na starost firewall.

6 Seznam použité literatury

[1] „VMware_paravirtualization.pdf", *VMware*. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf (viděno čer. 29, 2020).

[2] „Operating System (OS) Virtualization", *w3schools*. https://www.w3schools.in/cloud-virtualization/os-virtualization/ (viděno čvc. 29, 2020).

[3] „vmw-5-essential-characteristics-ebook.pdf", *VMware*. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/solutions/vmw-5-essential-characteristics-ebook.pdf (viděno čer. 29, 2020).

[4] „us-en_cloud_learnhub_hypervisors_a_complete_guide", *IBM*, kvě. 08, 2020. https://www.ibm.com/cloud/learn/hypervisors (viděno čer. 29, 2020).

[5] J. Strickland, „How Server Virtualization Works", *HowStuffWorks*, čer. 02, 2008. https://computer.howstuffworks.com/server-virtualization.htm (viděno čer. 29, 2020).

[6] „Desktop-Virtualization", *IBM*, pro. 13, 2019. https://www.ibm.com/cloud/learn/desktop-virtualization (viděno čvc. 30, 2020).

[7] „What is Desktop as a Service (DAAS)? - Citrix", *Citrix*. https://www.citrix.com/glossary/what-is-desktop-as-a-service-daas.html (viděno čvc. 30, 2020).

[8] „Network Virtualization", *VMware*. https://www.vmware.com/topics/glossary/content/network-virtualization (viděno čer. 29, 2020).

[9] „What is NFV?", *Red Hat*. https://www.redhat.com/en/topics/virtualization/what-is-nfv (viděno čer. 29, 2020).

[10] „us-en_cloud_learnhub_virtualization", *IBM*, čvc. 16, 2020. https://www.ibm.com/cloud/learn/virtualization-a-complete-guide (viděno čvc. 30, 2020).

[11] „vSphere Documentation Center", *VMware*. https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.introduction.doc_50%2FGUID-A8B8E6DC-A881-4F14-AEC1-D17365731E68.html (viděno čer. 29, 2020).

[12] „VMware NSX Data Center for vSphere Documentation", *VMware*. https://docs.vmware.com/en/VMware-NSX-Data-Center-for-vSphere/index.html (viděno čer. 29, 2020).

[13] „Hyper-V Architecture", *Microsoft*, led. 11, 2018. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture (viděno čer. 29, 2020).

[14] „Introduction to Hyper-V on Windows 10", *Microsoft*, čer. 25, 2018. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/ (viděno čer. 29, 2020).

[15] „citrix-xenserver-industry-leading-open-source-platform-for-cost-effective-cloud-server-and-desktop-virtualization.pdf", *Citrix*. https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/citrix-xenserver-industry-leading-open-source-platform-for-cost-effective-cloud-server-and-desktop-virtualization.pdf (viděno čer. 29, 2020).

[16] „Citrix Hypervisor 8.2", *Citrix*. https://docs.citrix.com/en-us/citrix-hypervisor.html (viděno dub. 29, 2021).

[17] „Introduction to Virtualization", *Oracle*. https://docs.oracle.com/cd/E15458_01/doc.22/e15444/intro.htm (viděno čer. 29, 2020).
[18] „Docker overview", *Docker Documentation*, čvc. 28, 2020. https://docs.docker.com/get-started/overview/ (viděno čvc. 31, 2020).
[19] „Containers vs. virtual machines", *Microsoft*, říj. 21, 2019. https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm (viděno čer. 29, 2020).
[20] „What is a Load Balancer? - Load Balancing Definition - Citrix", *Citrix*. https://www.citrix.com/glossary/load-balancing.html (viděno čer. 29, 2020).
[21] „What Is Load Balancing? Types, Configurations, and Best Tools - DNSstuff", *Software Reviews, Opinions, and Tips - DNSstuff*, led. 06, 2020. https://www.dnsstuff.com/what-is-server-load-balancing (viděno srp. 31, 2020).
[22] „Web Application Attack: What Is It and How to Defend Against It?", *Acunetix*. https://www.acunetix.com/websitesecurity/web-application-attack/ (viděno čvc. 15, 2020).
[23] „SQL Injection", *w3schools*. https://www.w3schools.com/sql/sql_injection.asp (viděno čvc. 21, 2020).
[24] „JavaScript Tutorial", *w3schools*. https://www.w3schools.com/js/ (viděno čvc. 22, 2020).
[25] „What is Cross-site Scripting and How Can You Fix it?", *Acunetix*. https://www.acunetix.com/websitesecurity/cross-site-scripting/ (viděno čvc. 22, 2020).
[26] „What is a DDoS Attack? - DDoS Meaning", *kaspersky*, led. 31, 2019. https://www.kaspersky.com/resource-center/threats/ddos-attacks (viděno čvc. 24, 2020).
[27] „DDoS Attack Types & Mitigation Methods | Imperva", *Learning Center*. https://www.imperva.com/learn/application-security/ddos-attacks/ (viděno čvc. 24, 2020).

7 Přílohy

Vytvoření třídy User v jazyce Java

```
@Entity
public class User {

   @Id
   @GeneratedValue
   private int id;
   @Column(length = 100)
   private String jmeno;
   @Column(length = 100)
   private String prijmeni;
   @Column(length = 100)
   private String bydliste;
   @Column
   private int cislo_Ikony;
   @Column
   private Date datum_Narozeni;
   @Column
   @Nullable
   private Date datum_Nacteni_Uzivatele;

   public User(){}

   public User(String jmeno, String prijmeni, String bydliste, int cislo_Ikony, Date
datum_Narozeni) {
      this.jmeno = jmeno;
      this.prijmeni = prijmeni;
      this.bydliste = bydliste;
      this.cislo_Ikony = cislo_Ikony;
      this.datum_Narozeni = datum_Narozeni;
      this.datum_Nacteni_Uzivatele = null;
   }

   public void sortUsers(List<User> uzivatele){
      Collections.sort(uzivatele, new Comparator<User>() {
         @Override
         public int compare(User o1, User o2) {
            return o1.getDatum_Narozeni().compareTo(o2.getDatum_Narozeni());
         }
      });
   }

   @Nullable
   public Date getDatumNacteniUzivatele() {
      return datum_Nacteni_Uzivatele;
```

```
    }

    public void setDatumNacteniUzivatele(@Nullable Date
datum_Nacteni_Uzivatele) {
        this.datum_Nacteni_Uzivatele = datum_Nacteni_Uzivatele;
    }

    public String toString(){
        return jmeno + prijmeni;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getJmeno() {
        return jmeno;
    }

    public void setJmeno(String jmeno) {
        this.jmeno = jmeno;
    }

    public String getPrijmeni() {
        return prijmeni;
    }

    public void setPrijmeni(String prijmeni) {
        this.prijmeni = prijmeni;
    }

    public String getBydliste() {
        return bydliste;
    }

    public void setBydliste(String bydliste) {
        this.bydliste = bydliste;
    }

    public int getCislo_Ikony() {
        return cislo_Ikony;
    }
```

```
    public void setCislo_Ikony(int cislo_Ikony) {
        this.cislo_Ikony = cislo_Ikony;
    }

    public Date getDatum_Narozeni() { return datum_Narozeni;
    }

    public void setDatum_Narozeni(Date datum_Narozeni) {
        this.datum_Narozeni = datum_Narozeni;
    }
}
```

Vytvoření UserRepository v jazyce Java

```
@Transactional
@Repository
public interface UserRepository extends JpaRepository<User,Integer> {

    @Query(value = "SELECT * FROM user u WHERE u.id <= 1000", nativeQuery =
true)
    List<User> findThousandUsers();

    @Query(value = "SELECT * FROM user u WHERE u.id <= :amount", nativeQuery
= true)
    List<User> findRandomAmountOfUsers(@Param("amount") int amount);

    @Modifying
    @Query(value = "UPDATE User u SET u.datum_Nacteni_Uzivatele =
:datumNacteni")
    void addUserLoadDate(@Param("datumNacteni") Date
datum_Nacteni_Uzivatele);
}
```

Inicializační služba databáze v jazyce Java

```
@Service
@Transactional
public class InitDatabaseService {

private UserRepository userRepository;

@Autowired
DataSource dataSource;

public InitDatabaseService(@Autowired UserRepository userRepository){

    this.userRepository = userRepository;
```

```
}

public void initDB(){

   userRepository.deleteAll();
   userRepository.flush();
   System.out.println("data smazana");
   loadData();
   System.out.println("data nactena");

}


public void loadData(){

   try {

      Resource resource = new ClassPathResource("testData.sql");
      ResourceDatabasePopulator resourceDatabasePopulator = new
ResourceDatabasePopulator(resource);
      resourceDatabasePopulator.execute(dataSource);

   }catch (Exception e){
      System.out.println("Nepodarilo se nahrat data / precist soubor");
   }
 }
}
```

Vytvoření HomeController kontroleru v jazyce Java

```
@Controller
public class HomeController {

   @Autowired
   UserRepository userRepository;

   @Autowired
   InitDatabaseService initDatabaseService;

   List<User> users;
   User uzivatel = new User();

   @RequestMapping("/")
   public String defaultPage(Model model){
```

```
        users = userRepository.findThousandUsers();
        if (users.size() == 0){
            initDatabaseService.initDB();
            users = userRepository.findThousandUsers();
        }
        System.out.println(users.size() + "pocet useru");
        uzivatel.sortUsers(users);
        model.addAttribute("users", users);
        return "/home";
    }
    @RequestMapping("/loadmoredata")
    public String loadMoreData(Model model){

        users = userRepository.findAll();
        if (users.size() == 0){
            initDatabaseService.initDB();
            users = userRepository.findAll();
        }
        System.out.println(users.size() + "pocet useru");
        uzivatel.sortUsers(users);
        userRepository.addUserLoadDate(new Date());
        System.out.println("datum pridano k uzivatelum");
        model.addAttribute("users", users);
        return "/home";
    }

    @RequestMapping("/init")
    public String init(){

        initDatabaseService.initDB();
        return "redirect:/";
    }
}
```

Vytvoření home.jsp s využitím HTML, CSS a JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page pageEncoding="UTF-8" %>
<!doctype html>
<html lang="cs">
<head>
<style type="text/css">
    .vizitka {float:left;margin-left: 5%;display: inline-block;border-
style:solid;border-width: 1px;text-align: center;}
</style>
```

```
</head>
<body class="bg-light">

<h1>HELLO WORLD from server 1!</h1>

<c:forEach var="u" items = "${users}">
  <div class = "vizitka">
    <p><h2>${u.getJmeno()}
      ${u.getPrijmeni()}</h2> </br> </p>
    <p>Bydliště: ${u.getBydliste()} </br></p>
    <p>Datum narození: <fmt:formatDate dateStyle="medium"
value="${u.getDatum_Narozeni()}" /> </p>
  </br>

    <img src="img/${u.getCislo_Ikony()}.jpg" width="300" height="300" />


  </div>
</c:forEach>

</body>
</html>
```

Ukázka vkládání testovacích dat do databáze zakomponovaných ve webové aplikaci v jazyce SQL

```
insert into user (id, jmeno, prijmeni, bydliste, cislo_Ikony, datum_Narozeni)
values (1, 'Babbette', 'Hagwood', 'Néa Plágia', 1, '1969-04-01');
insert into user (id, jmeno, prijmeni, bydliste, cislo_Ikony, datum_Narozeni)
values (2, 'Crichton', 'Nisbet', 'Checun', 2, '1972-04-04');
insert into user (id, jmeno, prijmeni, bydliste, cislo_Ikony, datum_Narozeni)
values (3, 'Jacqueline', 'Martinson', 'Pivovarikha', 1, '1990-07-25');
insert into user (id, jmeno, prijmeni, bydliste, cislo_Ikony, datum_Narozeni)
values (4, 'Urbano', 'Flight', 'Parung', 1, '1963-06-10');
insert into user (id, jmeno, prijmeni, bydliste, cislo_Ikony, datum_Narozeni)
values (5, 'Lucilia', 'McDiarmid', 'Delmas', 3, '1968-05-15');
```

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Martin Chaloupka**
Studium:	I1700701
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Odolnost virtuálních platforem vůči DDoS útokům**
Název diplomové práce AJ:	Resilience of virtual platforms to DDoS attacks

Cíl, metody, literatura, předpoklady:

Jedním z cílů této práce je teoretické prozkoumání nejvíce používaných virtualizačních technologií. Další část této práce bude také rozbor hrozeb digitálních útoků na webové aplikace, jako je tzv. DDoS útok. K tomu patří i prozkoumání load-balancingu jako efektivní součástí softwarového návrhu infrastruktury. Praktická část se bude zabývat návrhem a konfigurací testovacího prostředí na různých platformách.

- Úvod
- Virtualizace
- Docker
- Load-balancing
- Rozbor útoků na webové aplikace
- Návrh testovacího prostředí
- Implementace testovacého prostředí
- Návrh testovacího scénáře
- Testování
- Závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Ing. Filip Malý, Ph.D.
Datum zadání závěrečné práce:	17.9.2019