Mendelova univerzita v Brně
Provozně ekonomická fakulta

Možnosti provozu PHP aplikace

Diplomová práce

Vedoucí práce:

Ing. Jiří Lýsek, Ph.D.

Bc. Tomáš Kozák

Brno 2016

Rád bych poděkoval vedoucímu práce panu Ing. Jiřímu Lýskovi, Ph.D. za ochotu a vstřícnost, která přispěla k možnosti vypracování této diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Možnosti provozu PHP aplikace**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 30. prosince 2015 _____

Abstract

Kozák, T. Aspects of running a PHP application. Diploma thesis. Brno: Mendel University, 2016.
The thesis deals with various aspects of running a web application written in PHP (programming language). It considers capabilities of different interpreters and it also compares some cloud-based services. In the practical part the thesis describes the design and implementation of a test application which was stepwise deployed to various environments where it was subjected to benchmark test. The description of deployment also depicts all the problems encountered on each platform. Finally various performance-affecting resources are discussed for the purpose of improving the application efficiency.

Keywords

Web application, PHP, cloud, interpreter, load test

Abstrakt

Kozák, T. Možnosti provozu PHP aplikace. Diplomová práce. Brno: Mendelova univerzita v Brně, 2016.
Diplomová práce se zabývá tématikou provozu webové aplikace napsané v programovacím jazyce PHP. V úvahu bere možnosti interpretů a srovnává některé cloudové služby. V rámci testování byla navržena a následně implementována aplikace, která byla postupně nasazována a testována na jednotlivých prostředích. Práce obsahuje popis nasazení a s tím související problémy pro všechny testované platformy. Dále byl proveden soupis prostředků, které by mohly pomoci s výkonnostními parametry.

Klíčová slova

Webová aplikace, PHP, cloud, interpret, load test

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

1.1 Úvod

Dnešní informační společnost je založena na informačních a komunikačních zařízeních, které výrazně proměňují vnímání okolního světa. Poslat zprávu, fotografii nebo video na druhý konec světa během několika milisekund již není nepřekonatelný problém a někteří jedinci si život bez těchto technologií nedokáží ani představit. World wide web je neodmyslitelnou součástí této přeměny společnosti, se kterou je nutné počítat ještě po dlouhá léta. Webové rozhraní už neslouží jen jako zdroj informací, ale lze přes něj nakupovat, prohlížet média (fotky, filmy), chatovat nebo přes něj vést sociální život (rozvoj sociálních sítí). Počet uživatelů, kteří využívají tyto služby na internetu, dramaticky přibývá a provozovatelé těchto služeb se na tento nárůst musí důkladně připravit. Webová služba již není jen staticky uložená webová prezentace, ale rozsáhlé množství systémů, které spolu spolupracují ve snaze poskytnout svým klientům co možná nejlepší služby (informaci, nákup, zážitek nebo cokoliv jiného).

Zřejmě nejzranitelnějším druhem webové aplikace jsou takové aplikace, ve kterých jejich nedostupnost způsobuje finanční ztrátu svému majiteli. Typickým příkladem takového druhu aplikace je internetový obchod. Pokud obchod nefunguje, zákazník si nemůže objednat zboží, a tím vzniká poskytovateli služby ztráta. Z tohoto důvodu je důležité se touto problematikou zabývat již v průběhu návrhu systému a ne až v momentě, kdy se to stává kritické pro byznys. Systém jako takový by neměl mít žádné závažné bezpečnostní problémy a měl by být dostatečně výkonný k uspokojení všech požadavků. Tím není myšlena pouze strana IS/ICT, ale i systém z pohledu skladového hospodářství, expedice zboží, zákaznická podpora, atp. Tato diplomová práce se však zaměří pouze na IS/ICT část a to především na výkonnostní část systému.

1.2 Cíl práce

Cílem práce je zhodnotit možnosti provozu aplikace napsané v jazyce PHP z výkonnostního hlediska. Bude navržen a následně implementován e-commerce systém, na kterém budou provedeny testy, které si berou za cíl odhalit výhody a nevýhody jednotlivých řešení. Důraz bude kladen na alternativní interprety jazyka PHP a na možnost provozu PHP aplikace v cloudovém prostředí. Dílčím úkolem bude nalezení vhodných doplňujících systémů, které budou mít pozitivní vliv na výkonost aplikace.

2 Možnosti provozu PHP aplikace

K prosinci 2015 má PHP dle serveru builtwith.com 52% zastoupení mezi programovacími jazyky využité pro provoz webové aplikace. [1] V české republice je tento poměr ještě výraznější a dosahuje hodnoty 87 %. [2] PHP tedy zastává ve světe dominantního hráče. S touto popularitou také souvisí obrovská komunita programátorů, nespočet knihoven a frameworků, které mohou být k vývoji využity. Hlavně z těchto důvodů je třeba se věnovat možnostem provozu aplikace a prozkoumat alternativy, které trh nabízí.

2.1 Historie PHP a jeho verze

2.1.1 Prvopočátky PHP

PHP, které známe dnes, vzniklo v roce 1994 a jeho autorem se stal Rasmus Lerdorf jako tzv. PHP Tool. Postupem času Rasmus přepsal PHP Tool a začal produkovat větší a bohatší implementaci, která byla schopná např. komunikovat s databází a byla vhodná k vytváření jednoduchých dynamických webových aplikací. V roce 1995 Rasmus vydal zdrojové kódy PHP Tool, což dovolilo i ostatním vývojářům té doby používat a hlavně zlepšovat jeho produkt. V září roku 1994 Rasmus vydal kompletně přepsané PHP, ale upustil od jeho jména. V té době ho nazýval jako FI (zkratka pro "Forms Interpreter"), nová implementace zahrnovala některé základní funkce PHP, které přetrvaly až do dnešní doby. V říjnu byla vydána další verze, tentokrát už zase pod jménem PHP, která je brána jako vůbec první vydání PHP. Jazyk byl koncipován tak, aby byl podobný struktuře C a nedělal problémy s přechodem z C, Perlu a podobných jazyků. V roce 1996 vychází další vydání pod názvem PHP/FI, ve kterém přibyla podpora pro databáze (Postgres95), cookies nebo uživatelsky definované funkce. Téhož roku pak také vychází PHP 2.0. V dalších pár letech se kód hodně přepisoval a výsledkem bylo PHP 3.0, které se již velice podobalo dnešní verzi. Andi Gutmans and Zeev Suraski z Tel Avivu začali přepisovat parser a také doplnili funkcionalitu, aby bylo možné PHP použít pro elektronické obchodování. Také odstraňují z názvu FI a zůstává jen PHP. V zimě 1998 začalo opět přepisování, které si vzalo za cíle zvýšení výkonu a komplexnosti aplikace. Nový Zend Engine (kompromis ze jmen Zeev a Andi) dostál svých cílů a jeho první představení proběhlo v roce 1999. PHP 4.0 postavené na tomto enginu bylo vydáno o rok později a podporovalo mnoho nových funkcí jako HTTP sessions, outbut buffering, zvýšila se bezpečnost a hlavně do jazyka přibyly nové konstrukce. Po dlouhém vývoji se v roce 2004 dostává na světlo nová verze 5 s jádrem Zend Engine 2.0, s novým objektovým modelem a další funkcionalitou, která přetrvala až do dnešní doby. [3]

2.1.2 PHP 5

Ze začátku vývoje PHP5 se většinou opravovali bugy a doplňovala se pouze okrajová funkcionalita. Důležitější věci se začaly dít od roku 2010. Verze PHP 5: [4]

- PHP 5.0 červenec 2004 – Zend Engine II
- PHP 5.1 listopad 2005 – časové zóny
- PHP 5.2 listopad 2006 – rozšíření JSON a zip, výkonnostní změny
- PHP 5.3 červen 2009 – přelomová verze – jmenné prostory, closures
- PHP 5.4 březen 2012 – Traits, krátká syntaxe pro pole
- PHP 5.5 červen 2013 – operátor yield, blok finally pro ošetřování výjimek, označení extenze MySQL jako zastaralé
- PHP 5.6 srpen 2014 – konstantní skalární výrazy, variadické funkce, upload souborů větších než 2GB

2.1.3 PHP 6

Koncem roku 2005 vznikl poměrně dlouhý seznam věcí, které by měly být implementovány v nové majoritní verzi PHP. Jedna z největších změn měl být přechod na Unicode. Dále měla být přidána např. podpora pro velká čísla implementací nového datového typu a mnoho dalších změn. Také měli být odstraněny některé již zastaralé obraty jako např: *register_globals* nebo *magic_quotes*. Většina této požadované funkcionality byla do PHP přidána ve verzích PHP 5.3 a PHP 5.4. V té době již existovala celá řada publikací popisující verzi 6, proto bylo rozhodnuto tuto majoritní verzi přeskočit, a aby nedocházelo ke zbytečným střetům informačních zdrojů, vydat rovnou verzi 7.

2.1.4 PHP 7

Oficiální vydání verze 7 proběhlo 3. prosince 2015. Uvádí se, že díky rozsáhlému refaktorování kódu a kompletnímu přepracování datových struktur se výrazně zvýšila výkonnost PHP. Některé testy uvádějí až dvojnásobné zrychlení a 30% úsporu paměti. V této verzi se konečně objevuje typová kontrola skalárních parametrů a typová kontrola návratových hodnot funkcí, bezpečný generátor náhodných čísel, operátor "spaceship", kódování unicode, anonymní třídy a bylo odstraněno vyvolávání chyb programu po zavolaní funkce z nevhodnými parametry (nyní je vyvolána výjimka). Opět byly odstraněny funkce, které byly v minulých verzích označeny jako zastaralé. [5]

2.2 Interpreti pro PHP

Pod pojmem interpret se v tomto kontextu skrývá program, který dokáže parsovat, interpretovat a následně vykonat PHP kód. Existuje celá řada interpretů pro PHP, ale drtivá většina z nich jsou pokusné projekty, které se nijak masově nepoužívají a

jsou využívány v podstatě jen k experimentálním účelů. Kolem těchto interpretů neexistuje nijak velká komunita lidí a jejich nasazení na produkční servery rozhodně není dobrý nápad. Několik zástupců této skupiny interpretů:

- HappyJIT – převod PHP přes Zend a BC parser do HappyJIT Byte kódu [6]
- PHP.js – implementace PHP pomocí javascriptu [7]
- Phalanger – PHP kompilátor pro .NET framework, který je v současnosti podporován Microsoftem [8]
- PHP Compiler – převod PHP do nativního binárního kódu nebo do abstraktních syntaktických stromů sloužících k analýze [9]
- Rose – transformace a analýza kódu [10]
- Roadsend – převod do nativního binárního kódu [11]
- Pipp – převod PHP do Parrot bytekódu a spuštení na Parrot VM [12]
- Quercus – převod PHP do jazyka JAVA [13]
- Pie – implementace PHP v jazyce Python [14]

V podstatě ani jeden z výše uvedených možností se v praxi nijak neuplatňuje. Autoři jsou většinou jednotlivci nebo malá skupina lidí. Projekt je většinou v úpadku a vývoj vůbec nereflektuje vývoj jazyka PHP. To má bohužel za následek, že tyto možnosti v provozu upadají do zapomnění. Rozšíření zůstávají pouze dva interpreti, kterými jsou Zend engine a HHVM. Majoritním ovšem stále zůstává Zend engine, který je možné nalézt snad na všech službách poskytujících nejzákladnější formu webového hostingu.

2.2.1 Zend engine

Zend engine (aktuálně ve verzi 3) byl jádrem PHP již od verze 4. Jedná se o jedinou kompletní a zároveň nejrozšířenější implementaci PHP. Zend engine kompiluje PHP kód za chodu do svého vnitřního formátu, který umí spustit. Zend engine zároveň funguje jako referenční implementace, od které jsou ostatní implementace odvozeny. Je kompletně napsán v programovacím jazyku C. Aktuálně je vydána nová verze 3, které se také označuje jako phpng, jenž byla vytvořena pro PHP 7. Všechny zdrojové kódy jsou samozřejmě otevřené a kdokoliv může poslat svoje úpravy pomocí komunitního projektu na serveru GitHub. [15]

2.2.2 HHVM

HHVM (Hip Hop Virtual Machine) je virtuální stroj pro spouštění programů napsaných v jazycích Hack a PHP vytvořený společností Facebook, který HHVM používá pro běh jak ve vývojovém, tak produkčním prostředí. Omezením pro některé uživatele může být fakt, že jediný skutečně podporovaný operační systém je Linux. Kompletní zdrojové kódy jsou otevřené. HHVM by mělo být provozováno na serverech společně s FastCGI webserverem jako je např. Apache nebo ngnix.

HHVM používá just-in-time kompilaci a chlubí se především svým výkonem. [16] Na druhou stranu omezuje jazyk PHP o některé možnosti zápisu: [17]

- goto, if:...endif;
- AND, OR, XOR
- reference
- záměna statického volání metody (A::foo() vs. $a->foo())
- konstrukce break N and continue N
- dynamické spouštění kódu pomocí eval a $object->{$var}
- neumožňuje použít nedefinovanou proměnnou
- klíčové slovo global
- tzv. proměnná proměnných ($$a)
- zápis list(, $b), je nutné použít konstrukci list($_, $b)
- použití stringu jako volání funkce ($func = 'myFunc'; $func(1,2);)
- Rozhraní ArrayAccess
- Case-insensitive volání funkcí

Většinu těchto konstrukcí "slušný" programátor nepoužívá, ale pokud by byl starší kód migrován na HHVM, tyto restrikce mohou způsobit problémy. Většina moderních PHP frameworků se s těmito změnami vyrovnala a jsou plně kompatibilní.

2.3 PHP v cloudu

Možnosti interpretů umožní rychlejší zpracování, to však v případě některých aplikací nemusí být dostačující. Pro náročné aplikace je nutné rozdělit výpočetní výkon na více zdrojů, kde příkladem může být elektronický obchod s vysokou návštěvností. Jedna z možností je postavit si vlastní platformu, což má sice značné výhody jako např. absolutní přizpůsobení platformy aplikaci nebo vlastní správu serverů, ale z finančního hlediska se nejedná o úsporné řešení. Pokud přihlédneme k faktu, že na trhu jsou rozšířeny specializované služby, nemá cenu si vlastní platformu stavět, ale využít již nějakou stávající. Tyto služby přináší nejrůznější výhody: [18]

- není nutné znát hardware
- přednastavená konfigurace a možnost její změny přes administrační rozhraní
- garance bezpečnosti
- škálování výkonu
- platba pouze za využité výpočetní kapacity
- podpora provozovatele
- monitoring platformy

Cloud nám sice přináší tyto výhody, ale samozřejmě přináší také nevýhody:

- smluvní podmínky od poskytovatele
- nutnost přizpůsobit aplikaci platformě
- složité migrace
- data uložená na cizím hardwaru

Následující text detailně popíše vybrané cloudové služby, jejich parametry a další skutečnosti, které dodají základní představu pro následný návrh systému, na kterém budou všechny tyto platformy podrobeny výkonnostním testům.

2.3.1 Google App Engine

Jako první z cloudových řešení bude představen Google Cloud Platform, který umožňuje běh aplikací napsaných v jazycích Python, Java, PHP a Go. Základními vlastnosti, které Google nabízí, jsou:

- persistentní uložiště pro data s podporou dotazování, řazení a transakcemi
- automatické škálování výkonu a vyrovnávání zátěže
- asynchronní frontování požadavků
- možnost naplánovat úlohu ve stanovené době nebo ji spouštět v pravidelných intervalech
- podpora integrace s dalšími cloudovými službami společnosti Google

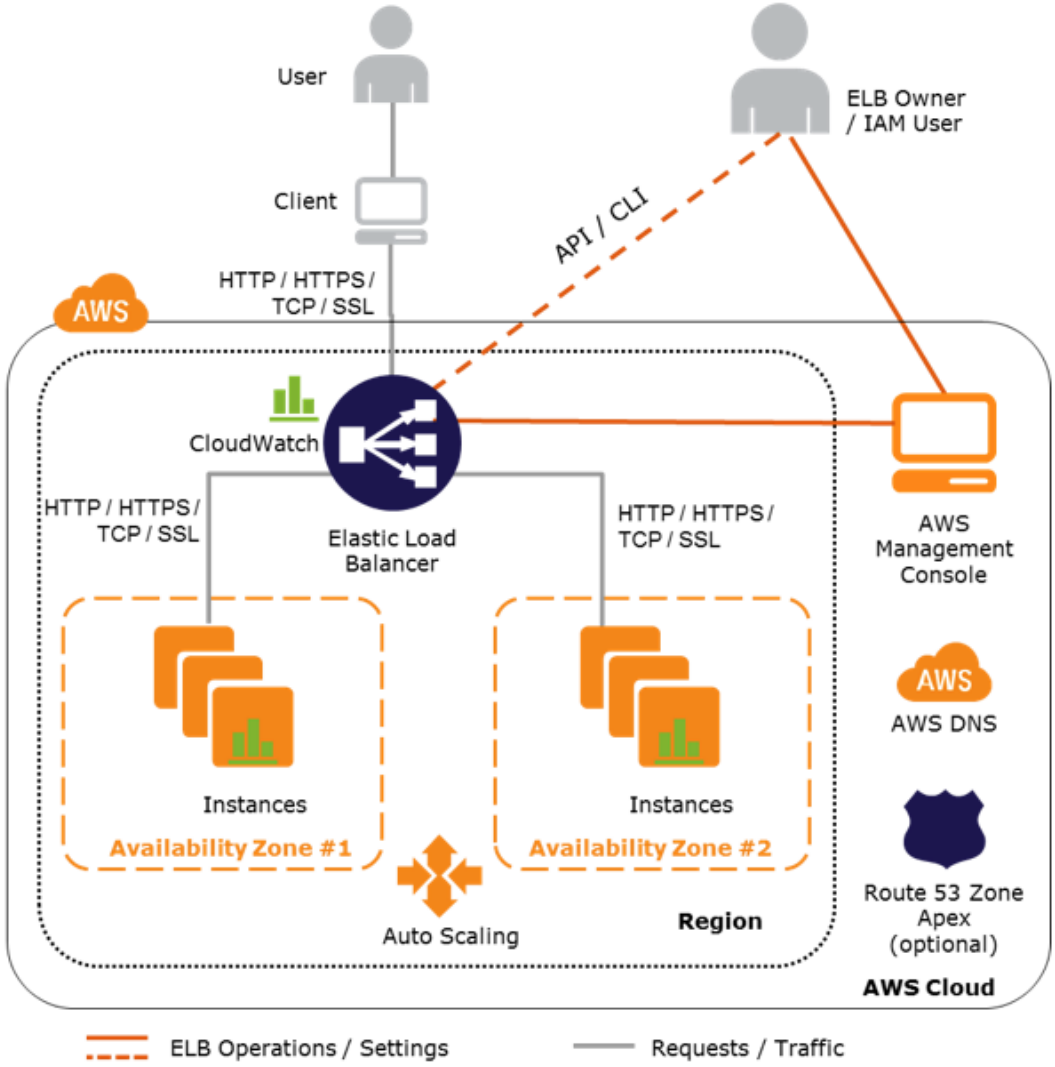
Platforma Googlu spouští aplikaci v prostředí interpretu PHP 5.5, kterému přidává pár výhod, ale také nevýhod. Mezi výhody lze zařadit např. automatické loadování tříd, takže programátor nemusí volat funkci *include* pokud chce třídu použít. Další výhodou je platba za využívání Google Enginu, kde se platí od prostředků, které servery potřebují pro svůj chod. Na druhou stranu má Google velké nedostatky v dostupnosti některých rozšíření do PHP. [19]

Tab. 1 Seznam podporovaných rozšíření v Google App Enginu

apc	bcmath	calendar	Core	ctype
date	dom	ereg	filter	FTP
gd	hash	iconv	json	libxml
mbstring	mcrypt	memcache	memcached	mysql
mysqli	mysqlnd	OAuth	openssl	pcre
PDO	pdo_mysql	Reflection	session	shmop
SimpleXML	soap	SPL	standard	tokenizer
xdebug	xml	xmlreader	xmlwriter	Zip

Mezi další nevýhody patří nedostupnost některých standardních funkcí a odlišné chování k sessions. Největším problémem se však zdá být manipulace se soubory. V App Enginu je lokální souborový systém nedostupný k zápisu z důvodů bezpečnosti a možnosti škálování. Google App Engine poskytuje k potřebě zápisu tzv. Google Cloud Storages stream wrappery, ke kterým však nelze přistupovat jako k normálním souborům na disku. Není možné soubor otevřít v některých režimech, není možné soubory zamykat a nejsou podporovány některé funkce pro práci se souboru jako např. *ftruncate*. [20] Google nabízí k práci se soubory také API, ale zdá se, že je možné se soubory pracovat relativně stejně jako na ostatních platformách, takže využívat API není zcela nutné.

2.3.2 Amazon EC2

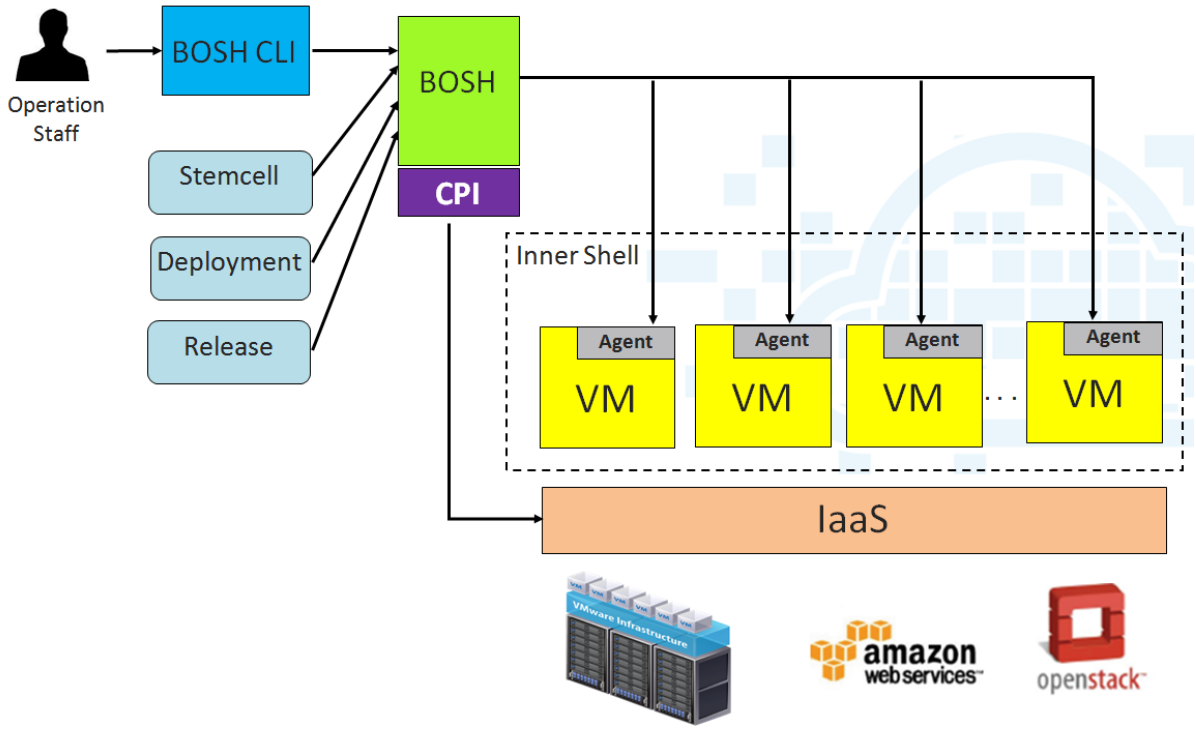


Obr. 1 Schéma architektury v prostředí Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) je služba, která poskytuje dynamickou výpočetní kapacitu v cloudu. Architektura této platformy pracuje na principu vyrovnávání zátěže mezi jednotlivými instancemi aplikace. V případě této práce o rozšiřování instancí pro webové servery. Jedná se o naprosto standardní linuxovou distribuci Ubuntu, ale Amazon umožňuje výběr i z jiných možností (Windows, RedHat, Amazon Linux). Tento server je potřeba nastavit, nainstalovat potřebný software (Apache2 + PHP) a zprovoznit na něm aplikaci. Tato instance se dále označí jako referenční a vytvoří se z ní tzv. image (obraz systému). Referenční in-

stance se bude při provozu aplikace replikovat a následně se přidá do skupiny serverů obsluhující danou aplikaci (servers pool). Počet instancí v poolu se může snižovat nebo zvyšovat a tím reflektovat celkovou současnou zátěž celého systému. Je však nutné nastavit pravidla, při kterých k tomu bude docházet. Před celou skupinu instancí aplikace je předřazen load balancer (vyrovnávač zátěže), který je nastaven jako vstupní brána pro uživatele přistupujícího k aplikaci. Konfigurovatelné jsou také instance v poolu - lze jim přidělit systémové parametry (počet jader procesoru, paměť RAM a velikost uložiště). Celý tento popsaný systém lze konfigurovat z webového rozhraní AWS management console nebo automaticky pomocí API. [21] Velkou výhodou tohoto řešení je možnost instalace jakékoliv dodatečné knihovny nebo softwaru, jelikož se jedná o standardní instalace operačního systému. Cenová politika Amazonu je nastavená tak, že se platí pouze za využité zdroje. Konečná cena se pak liší od druhu aplikace.

2.3.3 Cloud Foundry (IBM Bluemix)



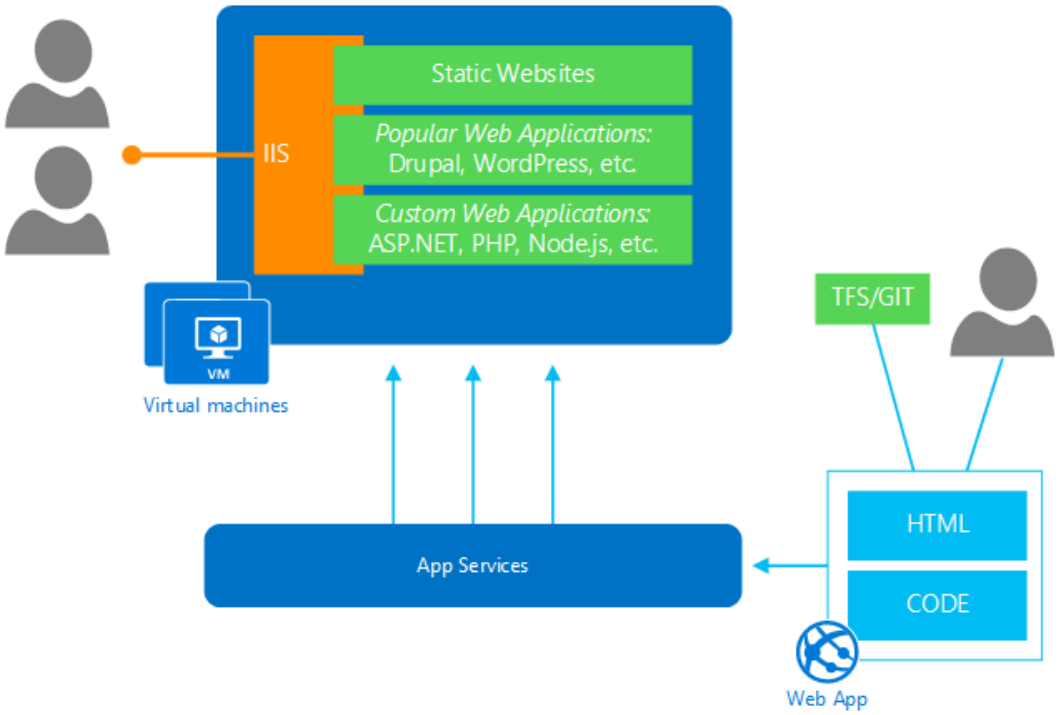
Obr. 2 Schéma architektury v prostředí Cloud Foundry

Cloud foundry je služba zajišťující platformu (PaaS) s otevřeným zdrojovým kódem (https://github.com/cloudfoundry). Cloud foudry umožňuje kompletní konfiguraci platformy včetně možností škálování systému. Součástí této platformy je projekt BOSH, který podporuje tvorbu releasů, deployment a řízení životního cyklu od malých aplikací až pro rozsáhlé škálované aplikace. [22] Nad tímto systémem

postavila společnost IBM svoji instanci Cloud Foundry a provozuje ji jako službu, kterou lze využívat k provozu vlastní aplikace. Na tomto prostředí lze spustit a provozovat takřka libovolnou aplikaci, je ale zapotřebí vytvořit sestavovací konfiguraci a následně v cloudovém prostředí aplikaci sestavit. Kompletní konfigurace není úplně jednoduchá, ale existuje velké množství předpřipravených řešení, které je možné použít. Příjemná je také podpora balíčkovacího systému composer. [23]

2.3.4 Microsoft Azure

Azure je cloudové řešení od společnosti Microsoft, na kterém lze provozovat aplikace napsané v jazycích Javascript, Python, .NET, PHP, Java a NodeJS. Azure je standardním cloudovým řešením, takže podporuje všechny výhody cloudu. Poskytuje i možnost globálního škálování nebo služby pro vývojáře jako vytváření buildů, nasazení kódu, roll-back nebo vlastní zátěžové testy, které ale nebudou použity, aby nebyla ovlivněna objektivita měření.



Obr. 3 Schéma architektury v prostření Microsoft Azure

Na rozdíl od Amazonu odstíní uživatele od správy samotných serverů a poskytne mu možnost konfigurace přes webové rozhraní. Toto sice může být považováno za výhodu, ale až do okamžiku, kdy bude některá z konfiguračních direktiv chybět a dostane správce aplikace do úzkých. Nespornou výhodou je kompatibilita s produkty společnosti Microsoft a to především s Visual Studiem. Azure není určeno jen pro webové aplikace, ale také pro další nejrůznější účely jako např. virtu-

ální počítače, databáze, datové sklady, nástroje pro strojové učení, počítačové sítě nebo zálohování. Ceny jsou vždy přizpůsobeny jednotlivým službám a platí se pouze za využitou výpočetní kapacitu. Jedná se o opravdu zajímavou platformu s kořeny v silné korporaci. [24]

2.4 Srovnání cen

Porovnání cen u cloudových služeb není jednoduché, protože každá z nich má svoji vlastní cenovou politiku. Jednu vlastnost však mají shodnou, a to že se platí pouze za využité zdroje. Ovšem odhadnout náročnost aplikace vyžaduje především zkušenosti s provozem a monitoring. Proto bylo rozhodnuto, že srovnání bude provedeno na fiktivní aplikace, která vyžaduje alespoň 2 instance serverů, minimálně 2 GB RAM na každé z nich a prostor pro ukládání souborů o velikosti minimálně 20 GB. Tyto parametry odpovídají menšímu elektronickému obchodu. 2 instance jsou vyžadovány, aby se již počítalo se škálováním na více serverů a nenastal problém při požadavku navýšit výkon. Tyto parametry byly zadány do cenových kalkulaček všech cloudových řešení a výsledky jsou k dispozici v následující tabulce.

Tab. 2 Srovnání cen cloudových služeb

	Azure	**App Engine**	**Amazon**	**IBM**
instance	2	2	2	2
RAM	3.5 GB	1.8 GB	3.7 GB	2 GB
CPU	2	2	2	-
uložiště	50 GB	375 GB	50 GB	20 GB
varianta	standard S2	n1-highcpu-2	c4.large	-
Cena	$280	$250	$165	$175

Bohužel IBM Bluemix neuvádí informace o CPU a nemá popsané detailní informace o vytvořených instancích. Jediné, co si uživatel může nastavit, je velikost paměti RAM a počet instancí. K dispozici má v základu 20 GB dat a více místa je možné obstarat pomocí placených balíčků. Na platformě Azure je možné si zvolit některý z předdefinovaných balíčků. Ostatní služby je možné dokoupit zvlášť (podpora, IP adresa, atp.). Google App Engine má svoji specifikovanou tabulku s cenami všech služeb po jednotkách a tím dává uživatelům lepší představu, kolik budou ve výsledku platit. Amazon používá podobnou politiku jako Google App Engine, ale vzhledem k počtu nabízených služeb je velice složité se v ní orientovat.

Byla vyzkoušena i konfigurace pro aplikace, která vyžaduje vyšší výkonnostní limity. Cena vzrostla, ale poměr cen u jednotlivých služeb byl zhruba zachován. Pokud by se aplikace rozrostla do obřích rozměrů, museli by správci kontaktovat přímo poskytovatele služby a ceny osobně dojednat.

V případě jednotek instancí cena převyšuje situaci, kdyby si správci aplikace postavili infrastrukturu sami pomocí VPS nebo dedikovaných serverů. To se ovšem

s přibývajícím počtem instancí změní a levněji začnou vycházet varianty cloudových služeb. Také zde odpadají náklady spojené se správou serverů, které mohou být nákladnější než provoz samotných serverů.

3 Návrh e-commerce aplikace

Základní myšlenkou aplikace je co možná největší podobnost s reálnými systémy. Tím by mělo být dosaženo co možná nejrealističtějších výsledků v následném testování. Pokud bychom pátrali po systému, který navštěvuje velké množství lidí, s největší pravděpodobností bychom došli ke zjištění, že se nejedná o jeden monolitický systém, ale o několik subsystémů, které spolu komunikují. V případě elektronického obchodu to může být skladové hospodářství, administraci objednávek, management stránek, atd. Základní dělení aplikací se rozlišuje na back office a front office systémy. Back office aplikace je typická tím, že je zákazníkovi skryta a on o její existenci většinou vůbec neví. Do této kategorie se řadí např. administrace produktů, kde dochází k doplňování informací, nastavování cen, přiřazování obrázků a jde vlastně o určitou formu CMS (content management systém). Na rozdíl od back office systémů jsou front office systémy viditelné pro zákazníka. Může se např. jednat o samotný obchod nebo pomocí systém pro sledování zásilky, popř. objednávky.

Aplikace, která má být vyvinuta pro účely této diplomové práce, tedy pro účely testování, se má co nejvíce podobat popsanému systému, aby bylo dosaženo co možná nejvěrohodnějších výsledků při testování. Jako první bude vyvinut jednoduchý skladový systém, který bude uchovávat pouze základní informace o produktech jako je název produktu, krátký popis, zařazení do kategorie a v neposlední řadě také cenu. Tento systém bude ve zbytku práce označován jako BRAIN.

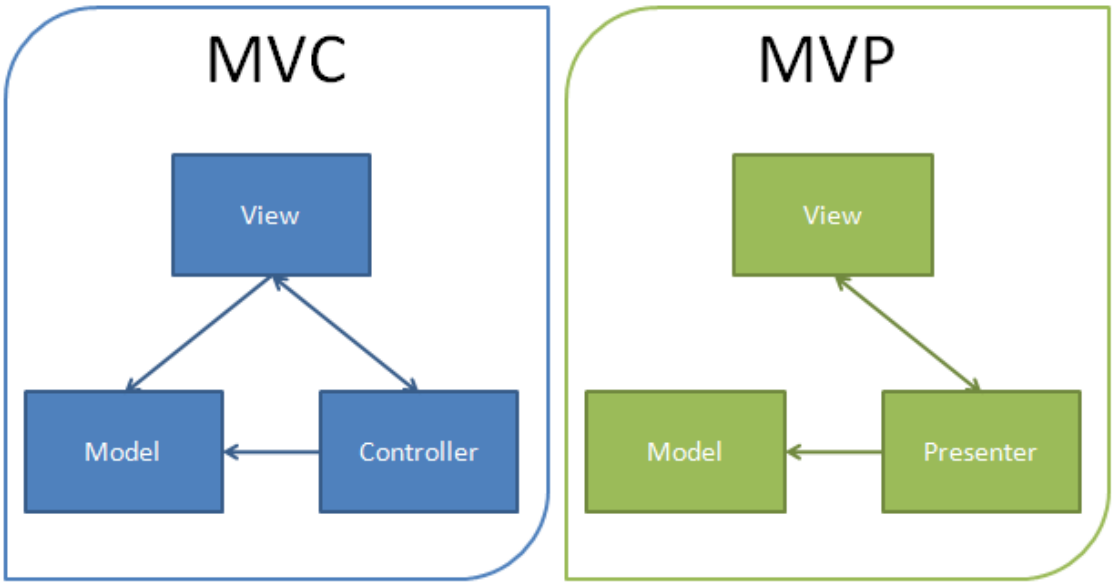
Dalším článkem pomyslného řetězu bude aplikace, která bude BRAIN rozšiřovat. V této práci využiji již hotového řešení v podobě systému OpenCart. Jedná se o e-commerce systém s otevřeným zdrojovým kódem, kde bude využito části administrace obchodu. Veškeré údaje o produktech budou brány z BRAINu a budou pravidelně synchronizovány do OpenCartu. Je důležité poznamenat, že databáze obou aplikací jsou striktně oddělené a oba systému mohou pracovat samostatně. Do systému OpenCart bude doprogramován submodul, jehož jediným úkolem bude synchronizace mezi BRAINem a OpenCartem. Nad těmito systémy pak vyroste samotný elektronický obchod.

3.1 Společná architektura aplikací

Aplikace vytvořené pro účel této práce jsou napsaná ve frameworku Nette, který rozděluje aplikaci do modelu MVC. Podstatou MVC je logické rozdělení do tří vrstev, které spolu komunikují. Vrstva modelu obsahuje objety, které representují nějaké informace o problémové doméně (komunikace s databází, komunikace s API, různé výpočty, atp.). Vrstva pohledu (view) presentuje výsledky převzaté z modelu do nějakého uživatelského rozhraní. V zásadě by pak neměl být problém

celé aplikaci vyměnit tuto vrstvu a např. z webové aplikace udělat aplikaci mobilní. [25] V případě této práce bude využito součásti frameworku Nette a jeho knihovny Latte. Úlohou controlleru je především zpracovávat uživatelský vstup a manipulovat na jeho základě s modelem a pohledem, který se aktualizuje odpovídajícím způsobem. V tomto případě je uživatelské rozhraní složeno právě z vrstev controlleru a view.

Nette používá místo controllerů tzv. presentery. V zásadě se podobá controlleru z MVC, ale je striktnější ke komunikaci mezi modelem a view. Funguje jako jakási mezivrstva mezi modelem a view. Zpracování požadavků je stejné jako v případě controlleru. V Nette se obvykle pod pojmem presenter myslí potomek třídy Nette\Application\UI\Presenter. Podle příchozích požadavků spouští odpovídající akce a vykresluje šablony. [26]



Obr. 4 Rozdíl mezi MVP a MVC architekturou

Šablonovací systém Latte, který je součástí Nette frameworku, umožňuje poměrně hezký a přehledný zápis kódu, jehož výstupem má být odpověď ve formátu HTML. Obsahuje nejrůznější marka pro řízení obsahu (if, for, foreach), je možné definovat bloky, které se dají i dědit. Dále jsou to filtry na úpravu nebo přeformátování nějakého výstupu. Makra i filtry si lze do Nette doprogramovat a používat tak rozšířenou syntaxi a funkcionalitu. Hlavní výhodou Latte oproti konkurenčním šablonovacím systémům (Twig, Smarty) je bezpečnost. Latte se snaží automaticky vyhodnotit kontext, ve kterém mají být jednotlivé proměnné vykresleny, čímž šetří vývojáři podstatné množství práce a tím i času.

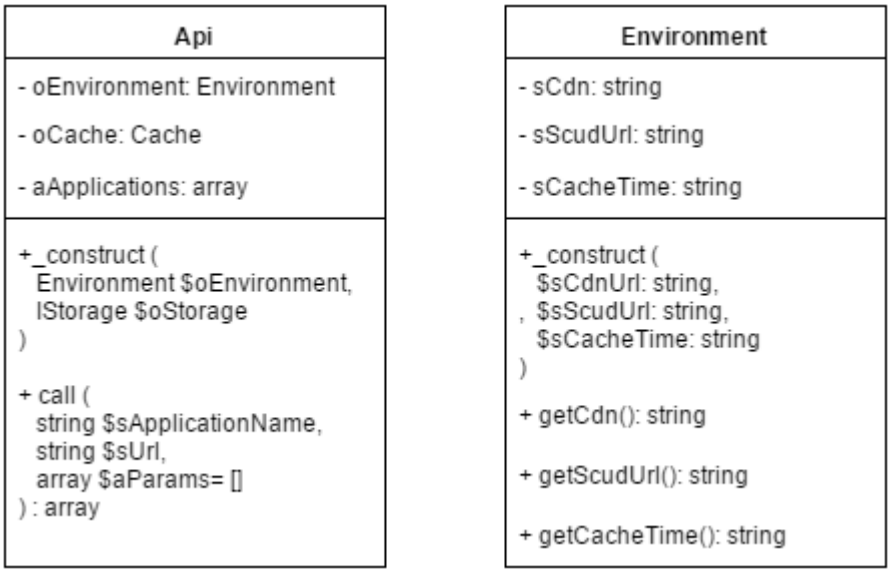
Na modelovou vrstvu má Nette framework balíček Nette database, která je prakticky obálkou nad standardní třídou PDO. Umožňuje efektivnější zápis SQL

dotazů a automatické vázání proměnných. Nette i tak modelovou stránku moc neřeší a nechává čistě na programátorovi, zda využije Nette database, použije některé z frameworků pro objektově relační mapování nebo zvolí jakoukoliv jinou možnost.

3.2 Balíček Ower

Balíček Ower byl vyvinut pro účely této platformy a pomocí **balíčkovacího nástroje composer** importován do všech ostatních aplikací. Ower obsahuje jednoduchou implementaci 2 tříd:

- Environment – třída na detekci prostředí, ve kterém se aplikace nachází
- Api – třída určená ke komunikaci s ostatními aplikacemi v platformě



Obr. 5 Model tříd balíčku Ower

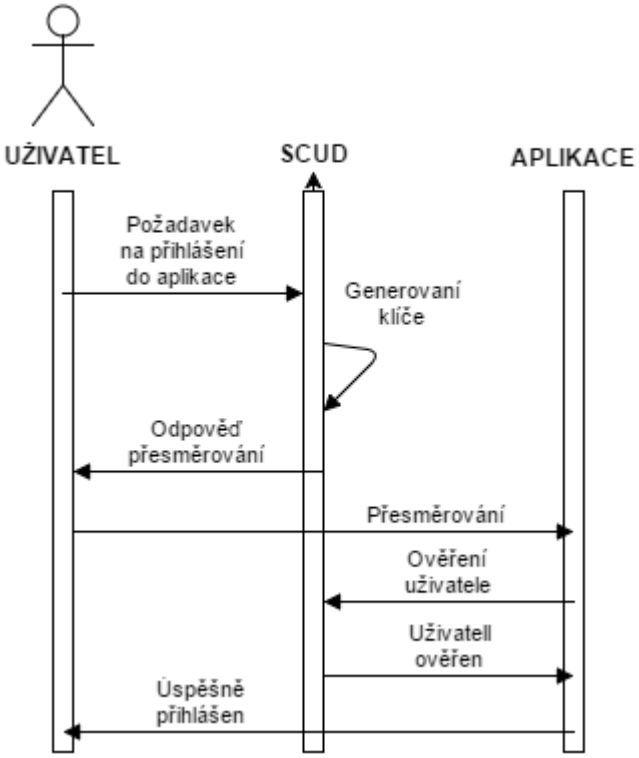
Celý balíček je pak závislí na balíčcích nette/utils, aby třídy v Oweru splňovali standardy Nette (reflexe) a nette/caching, díky které si je API schopno zacachovat adresy všech služeb v platformě. Poslední součástí balíku je výjimka BadApplicationException, která je vyvolána ve chvíli, kdy se API snaží zavolat neexistující službu.

Aby byl balíček přístupný přes composer, musí být umístěn v repositáři verzovacího systému Git a zaregistrován na serveru s balíčky. V případě soukromích repositářů by mohl být využit např. systém Satis, ale pro účely této práce bude stačit server Packagist – https://packagist.org/packages/pogodi/ower

V ostatních projektech je tento balíček stáhnut pomocí záznamu v souboru *composer.json*, kam se do sekce require přidá *"pogodi/ower": "~1.0"*.

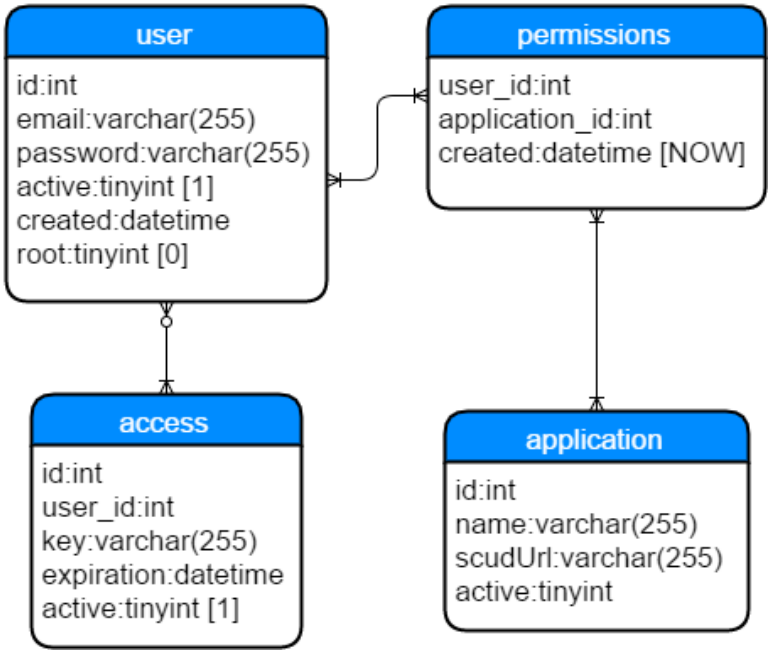
3.3 Aplikace SCUD

Aplikace SCUD slouží ke společnému přihlašování do ostatních aplikací. Vznikla z důvodů, aby si správce aplikace nemusel pamatovat velké množství hesel. Takhle se pouze jednou přihlásí přes SCUD, který ho je schopen přesměrovat na požadovanou aplikaci již v přihlášeném stavu. Pro jednotlivé uživatele je možné nastavit jednotlivá oprávnění, do jakých aplikací bude mít přístup. Na základě požadavku na přesměrování SCUD vygeneruje klíč (access token), se kterým uživatele přesměruje na aplikaci. Aplikace se pak následně ověří ve SCUDu, zda je klíč správný a na jeho základě uživatele přihlásí a vpustí do aplikace.



Obr. 6 Sekvenční model přihlášení administrátora do aplikace prostřednictvím SCUDu

Databáze obsahuje pouze 4 tabulky. V tabulce uživatel (*user*) jsou uloženy všechny potřebné údaje o uživatelích včetně zahashovaného hesla. Tabulka se seznamem aplikací (*application*) uchovává dostupné aplikace, které jsou přes tabulku práv (*permissions*) provázány s uživatelem. Na základě těchto 3 tabulek se v aplikaci rozhodne, zda má či nemá uživatel dostatečná oprávnění, a pokud má, klíč (*token*) je zapsán do tabulky přístupů (*access*), podle kterého se následně provede ověření.



Obr. 7 Databázový model aplikace SCUD

3.3.1 Struktura aplikace

Třídy modelu:

- Entity aplikace (*Application*) a přístupový klíč (*AccessToken*) - jednoduché entity na přepravu dat
- *AccessTokenManager* - generovaní, mazaní a kontrola klíče
- *ApplicationManager* - třída pro práci s aplikacemi
- *UserManager* - třída pro práci s uživateli včetně přihlašování

Presentery

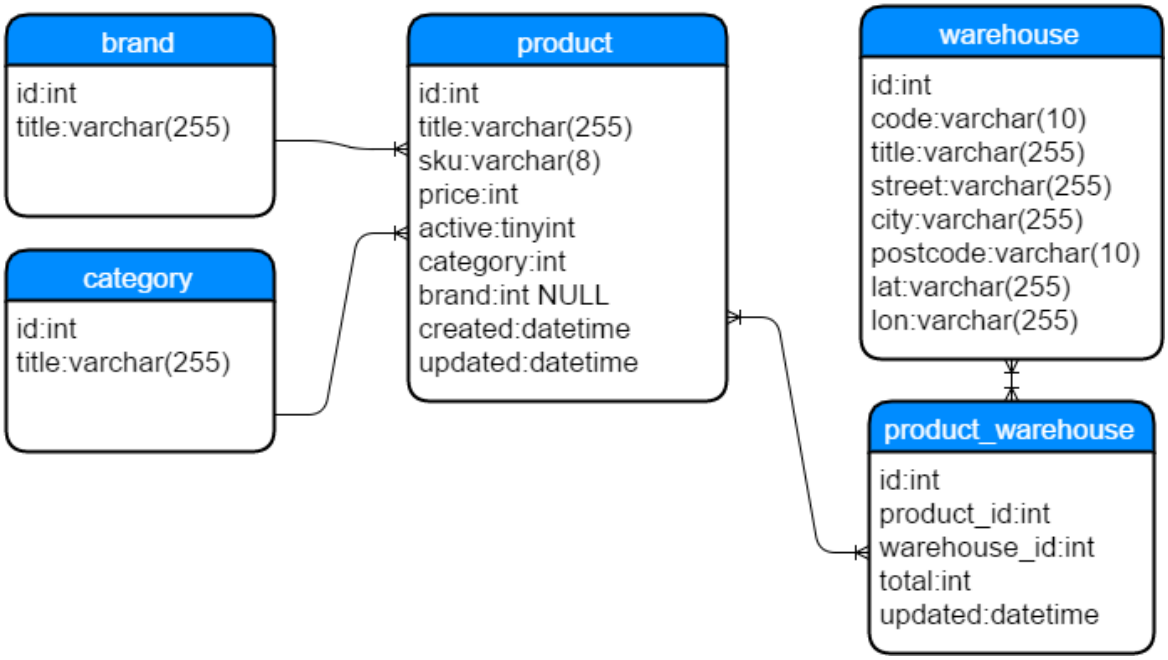
- *BasePresenter* - rodič pro ostatní presentery, zajišťuje, aby uživatel byl přihlášený a předává do view společné proměnné
- *SignPresenter* - přihlášení do aplikace SCUD
- *TransitionPresenter* - zvolení koncové aplikace s přesměrováním
- *CheckPresenter* - rozhraní sloužící ke kontrole klíče (access tokenu) ostatním aplikacím

3.4 Aplikace BRAIN

Jak již bylo zmíněno, BRAIN bude simulovat skladový systém, kde budou uchovány základní data o produktech. Konkrétně název produktu, jeho cena, stručný popis, zařazení do kategorie a hlavně jeho dostupnost v jednotlivých skladech. BRAIN nepočítá s žádnými dodatečnými rozšířeními jako je překlad do více jazyků nebo personalizace cen. Opravdu se nebude jednat o nic víc, než o jednoduchý skladova-

cí systém. Brain bude každý den dělat export svých souborů do formátu XML, které následně nahraje na FTP server, kde bude dostupný pro všechny ostatní aplikace, které budou tyto data vyžadovat.

3.4.1 Návrh schématu relační databáze



Obr. 8 Databázové schéma aplikace BRAIN

Ke každému produktu je uchováván jeho název (*title*), externí identifikátor (*sku*), cena (*price*), záznam o tom, zda je produkt aktivní (*active*), datum a čas vytvoření a poslední aktualizace (*created*, *updated*) a odkazy do tabulek značky (*brand*) a kategorie (*category*). V případě značky je tato vazba nepovinná, tedy produkt nemusí podléhat žádné značce. Seznam skladů je uložen v tabulce *warehouse*, která obsahuje kód skladu (*code*), jeho název (*title*), adresu včetně města a poštovního směrovacího čísla (*street*, *city*, *postcode*) a zeměpisné souřadnice (*lat*, *lon*). Vazba mezi skladem a produktem je uchována v tabulce *product_warehouse*, kde existují sloupce s aktuální informací o skladovosti v kusech (*total*) a datum poslední změny (*updated*). Pokud vazba mezi produktem a skladem neexistuje, počítá se s tím, že v daném skladu není. Aplikace zajistí, aby se neobjevoval záznam se skladem 0, ale aby byl záznam z tabulky rovnou smazán.

3.4.2 Struktura aplikace

Aplikace byla napsána od základů pomocí již zmíněné společné architektury.

Třídy modelu:

- *AbstractManager* - rodič pro všechny ostatní managery, zajišťuje připojení do databáze a spouští metodu *startup*
- *BrandManager* - manipulace se značkami
- *CategoryManager* - manipulace s kategoriemi
- *WarehouseManager* - manipulace se sklady
- Vrstva produkt, která je umístěna ve jmenném prostoru Pogodi\Product a dokáže se plně persistovat do databáze na základě současného stavu
- *FtpClient* - třída dokáže nahrát na FTP server požadovaný soubor
- *XmlResponse* - rozšiřuje Nette response o XML

Presentery

- *BasePresenter* - rodič pro ostatní presentery, zajišťuje, aby uživatel byl přihlášený, a předává do šablon společné proměnné
- *BrandPresenter* - administrace značek
- *CategoryPresenter* - administrace kategorií
- *DashboardPresenter* - hlavní stránka aplikace
- *ExporterPresenter* - CRON pro vytvoření exportu souborů do XML a následné nahrání na FTP server
- *ProductPresenter* - administrace produktů + řízení počtu produktů na skladě
- *WarehousePresenter* - administrace skladů
- *WelcomePresenter* - přivítání nepřihlášeného uživatele (presenter nedědí od *BasePresenteru*)
- *ScudPresenter* - přihlašování pomocí společné aplikace

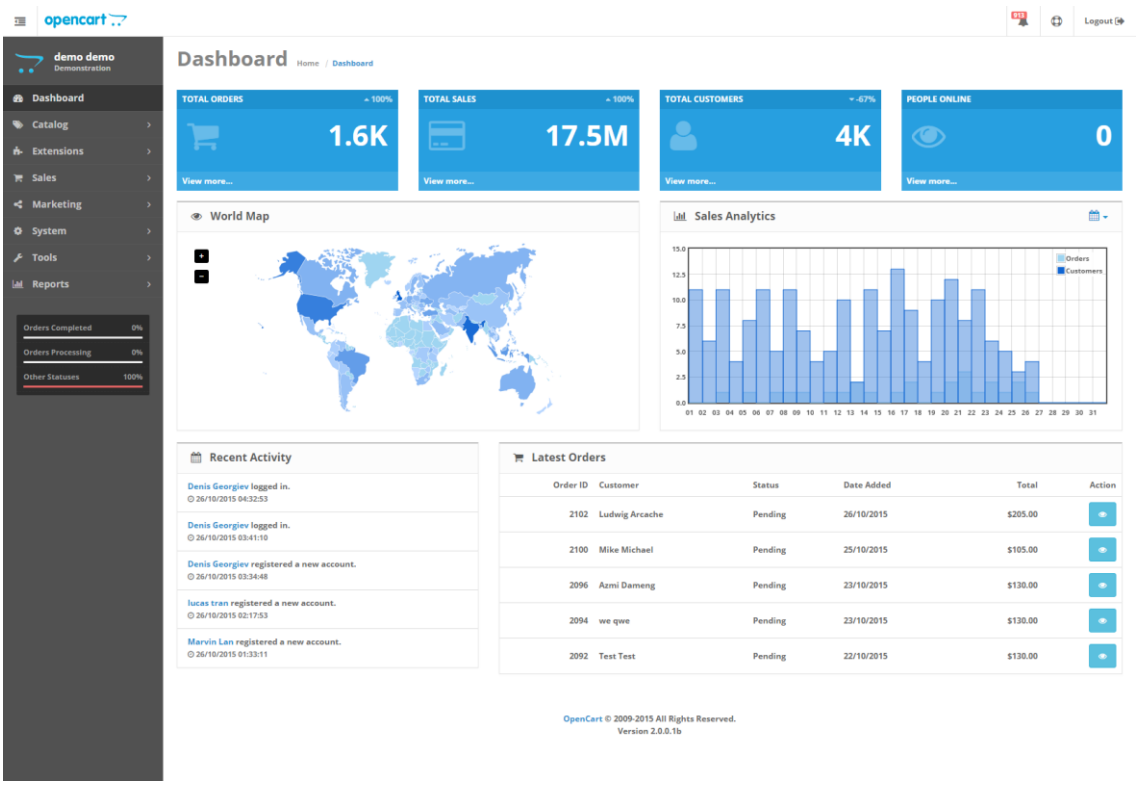
3.4.3 Vstupy a výstupy

Vstupem do aplikace je v této práci myšleno zadávání jednotlivých produktů do BRAINu ručně uživatelem. Je ovšem možné jeho rozšíření do formy virtuálního skladu, kde by stál jako mezivrstva mezi systémy reálného skladu (např. podnikovým informačním systémem) a administrací elektronického obchodu. Hlavním výstupem aplikace je XML soubor se všemi aktivními produkty, který je nahrán na FTP server pro účely ostatních aplikací.

3.5 OpenCart

Opencart rozšiřuje možnosti BRAINu, od kterého přijímá základní údaje a informace o stavu skladů. Na rozdíl od skladovacího systému bylo zvoleno využití již hotového řešení. Důvodem tohoto rozhodnutí byl fakt, že velké množství e-commerce aplikací nějaký podobný systém využívají a existuje velké množství firem, které administrační prostředí řeší právě využitím některého z volně dostupných systémů. Tato aplikace se svoji složitostí řadí už mezi větší. OpenCart se může pochlubit velkým množstvím funkcionality jako:

- Administrace kategorií - které můžou být mezi sebou provázané do stromové struktury
- Tagy – lze vytvořit štítky, které pak mohou být dále asociovány k produktům
- Ceny – k produktu jde vždy přiřadit pouze jedna cena + možnost nastavení daně s definovaného seznamu
- Měny – cenu lze přiřadit pouze jednu, ale existuje možnost mít uloženo více měn
- Administrace produktů – všechny popisky produktů, kategorie, atp. mohou být k produktu přiřazeny se závislostí na jazyku (v případě této práce se počítá pouze s jedním jazykem)
- Správa atributů produktu – lze si předpřipravit různé seznamy atributů, které poté jde přiřadit k jednotlivým produktům podle potřeby
- Správa objednávek – kompletní správa objednávek včetně asociace na objednané produkty
- Nástěnka – rychlý přehled o návštěvnosti a objednávkách včetně grafů



Obr. 9 Ukázka aplikace OpenCart

OpenCart obsahuje i další funkcionalitu a možnost uchování více informací, jako např. marketingové informace (affilates), slevové poukázky nebo dárkové poukazy. Pro tuto diplomovou práci ale bude výše uvedený výčet dostačující.

3.6 Propojení OPENCARTu a BRAINu

Od BRAINu se očekává, že to bude subsystém, který nebude závislý na ostatních subsystémech a bude fungovat zcela samostatně. Komunikaci tedy nebude aktivně vytvářet. Z tohoto důvodu bylo rozhodnuto, že BRAIN bude v pravidelných intervalech generovat XML feed, který si budou moci konzumenti (v tomto případě OpenCart) stáhnout a libovolně zpracovávat. Některé údaje o produktu jsou uchovány v obou systémech, proto je nutné určit priority jednotlivých atributů. Název, stručný popis a zařazení do kategorie bude prioritně bráno ze systému OpenCart, kde údaje z BRAINu budou použity pouze v případě nového produktu a naplnění výchozích údajů. Informace o stavu skladu a ceně však budou prioritně brány z BRAINu a při každém zpracování OpenCartem musí být aktualizovány, aby reflektovali data ve skladu. Předpokládaná granularita synchronizace je stanovena na hodnotu 1 – 2 × denně.

3.7 SITE

Site je subsystém, který se stará o interakci mezi uživatelem a administrací elektronického obchodu. V tomto konkrétním případě se bude jednat o jediný front office systém. Při návrhu se nepočítá s vlastní databází, ale s využitím přímé komunikace s administrací pomocí REST API, kde si budou brát jak informace o kategoriích a produktech, tak i posílat data o nových objednávkách. O kompletní vzhled elektronického obchodu se stará volně dostupná a plně responsivní šablona ze stránek w3layouts.com.

3.7.1 Struktura aplikace

Aplikace SITE taktéž dodržuje společné konvence. Je kompletně napsaná ve frameworku Nette a svoji strukturou reflektuje obsah elektronického obchodu.

Presentery

- *HomepagePresenter* – zobrazení produktů na hlavní straně
- *CategoryPresenter* – zobrazení výpisu kategorií včetně stránkování
- *ProductPresenter* – zobrazení produktu a vložení do košíku
- *BasketPresenter* – zobrazení a obsluha nákupního košíku

Všechny zmíněné presentery jsou potomky třídy *BasePresenter*, která má za úkol provést načtení tříd modelu a provést základní nastavení aplikace (např. nastavení adresy pro CDN server, nastavit výchozí jazyk).

Model

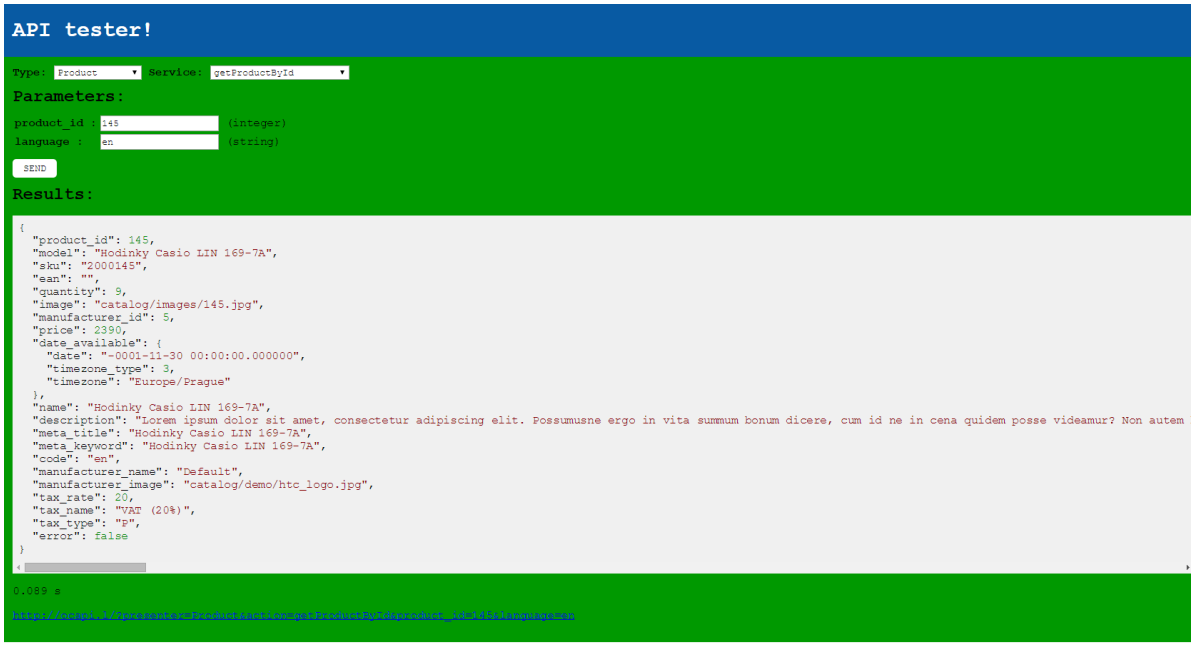
- *BasketItem* – třída odpovídající jednomu produktu vloženého do košíku. Je schopná se serializovat pro možnost uchování v sessions

- *Basket* – třída odpovídající celému nákupnímu košíku. Obsahuje informace o zákazníkovi, adrese a drží pole objektů type *BasketItem.* Taktéž je schopna se serializovat pro možnost uchování v sessions
- *BasketService* – služba pro lepší manipulaci s nákupním košíkem. Je schopná košík vytvořit a smazat, popř. z něj dostat všechny vložené produkty.

Model aplikace SITE také obsahuje třídy spojené s voláním OpenCart API, které budou popsány v části 3.9.

3.8 OpenCart API (OC Api)

Aplikace OC Api slouží ke komunikaci mezi administrací OPEN CARTu a aplikací SITE. OC Api nabízí rozhraní služeb, které může Site volat. Opačná inicializace spojení není možná a komunikace probíhá vždy tímto směrem. Aplikace je napsána tak, že služby se dělí do jednotlivých kategorií, které pak odpovídají Presenterům a jednotlivé akce pak metodám presenteru. Pro jednodušší testování a debugování bylo vyvinuto rozhraní API tester, které je dostupné v samotném kořenu webu. Tester po vybrání služby nabídne formulář s popisky a datovými typy parametrů pro jednotlivé služby. Výstup se pak snaží čitelně zformátovat, aby vývojáři co možná nejvíce zjednodušila práci. Aplikace je přímo připojená do stejné databáze jako OpenCart a provádí nad ní veškeré operace. Veškerá data jsou na klientskou stranu vracena ve formátu JSON.



Obr. 10 Ukázka volání Open Cart API

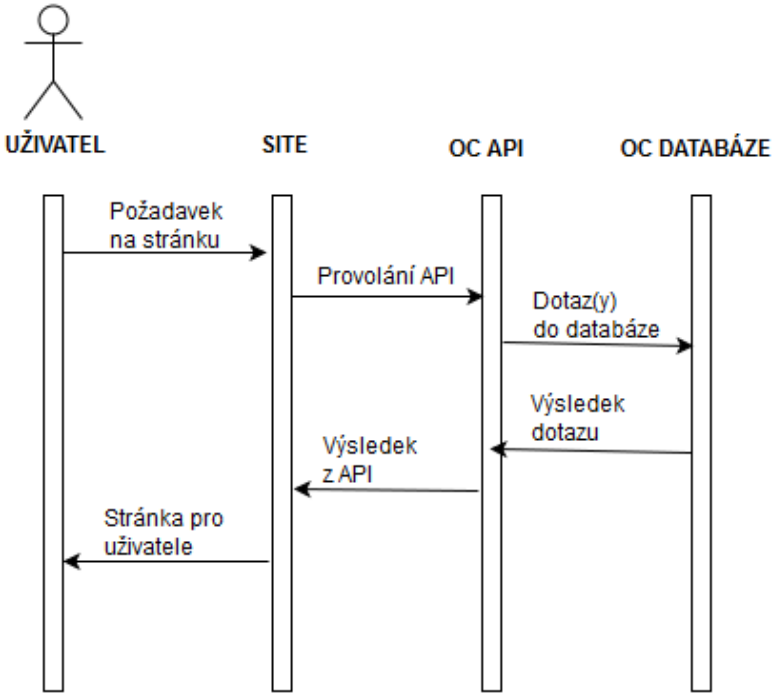
3.8.1 Nabízené služby

- *Product*
  - *getProductById* - vrací data o produktu na základě jazyka a unikátního identifikátoru (id)
  - *getProductImages* - vrací dostupné obrázky pro produkt na základě id
  - *getProductAttributes* - vrací doplňující atributy k produktu na základě id
  - *getProductsInCategory* - vrací seznam produktů v kategorii na základě jazyka a id kategorie
  - *getProductsInFilter* - vrací seznam produktů na základě jazyka a zvoleném filtru (např. hlavní strana)
- *Category*
  - *getAllCategories* - vrací všechny dostupné kategorie na základě jazyka
- *System*
  - *info* - vrací systémové údaje (např. kde se fyzicky nachází jednotlivé obrázky)
- *Fake*
  - *delay* - služba simuluje jakoukoliv úlohu, kterou by mělo API zpracovávat (např. žádost o zařazení do emailových rozesílek). V tomto důsledku není nutné implementovat pro testovací účely celé API, ale je

možné využít tuto metodu. V parametru sekundy se dá uvést, jak dlouho má operace trvat. Služba automaticky tento interval dodrží

3.9 Architektura komunikace v aplikaci SITE

V aplikaci SITE vznikla třída ***Caller***, která se stará o volání všech požadavků na API. Co se má volat pak zajišťují jednotlivé sužby implementované v aplikaci. Služby mají společného předka, který jim zajišťuje závislost na prostředí (třída *Environment* z OWERu), referenci na *Caller* a obecné uložiště pro cachování odpovědí (*ISTorage*). V jednotlivých službách (*Cateogry*, *Fake*, *Product*, *Order*, *System*) se pak provádí sestavení požadavku a předání *Calleru*, popř. operace spojení s cache. Všechny tyto služby jsou poté předány presenterům, které ho využívají k plnění šablon pro následné vykreslení obsahu uživateli.



Obr. 11 Sekvenční diagram - požadavek uživatele na stránku

3.10 Sdílená konfigurace

Jelikož celá aplikace bude nasazována na velké množství prostředí, bylo by dobré mít k dispozici centrální konfigurační soubor, kterým budou nastavovány všechny subsystémy v aplikaci. Všechny aplikace budou v souboru *bootstrap.php* očekávat konfigurační soubor *config.php*, který bude umístěn v adresářové struktuře o jednu

úroveň výš nad kořenem projektu. Výhodou tohoto řešení je umístění veškeré dostupné konfigurace v jednom místě a možnost tak rychle zasahovat do všech možností aplikace. Soubor *config.php* vrací jednoduché pole, kde klíče v poli odpovídají jednotlivým aplikacím. Hodnoty pak mohou obsahovat libovolnou další strukturu odpovídající požadavkům dané aplikace. Těchto hodnot pak následně využívají privátní konfigurace jednotlivých aplikací, většinou v souboru *config.neon,* kde jsou všechny dostupné jako parametry a jde k ním přistoupit přes speciální notaci. Ukázka vzorového konfiguračního souboru je součástí repositáře SITE.

3.11 Data

Po implementaci byla do elektronického obchodu nahrána data v podobě 10 000 testovacích produktů. Ke každému byl vygenerován název, cena, dlouhý a krátký popis a přiřazen ilustrační obrázek. Veškeré produkty byly rozděleny do 139 kategorií, které jsou uspořádány do stromové struktury. Na stránce jsou pak po vybrání kategorie zobrazeny nejen všechny produkty v aktuální kategorii, ale i produkty hierarchicky patřící pod tuto aktuální kategorii. Byl vytvořen jeden filtr sloužící k označení produktu jako „Homepage“. Pokud produkt spadá to tohoto filtru, je zobrazován na titulní stránce elektronického obchodu.

4 Testování aplikace

4.1 Metodika testování

Vlastní testování bude rozděleno do dvou základních skupin. První budou možnosti interpretu, kde si práce bere za cíl odhalit výkonností rozdíly mezi jednotlivými verzemi PHP běžícím na Zend Enginu a Hip Hop Virtual Machine. Bude vytvořen testovací skript pro testování výkonu, který bude spuštěn na serveru se stejnými podmínkami opakovaně pro různé interprety jazyka PHP. Druhá část testování se zaměří na možnosti platformy, tedy na možnost nasadit aplikaci do cloudu. Jelikož se jednotlivé platformy nedají zcela objektivně srovnat, je nutné použít alternativní ukazatele jako složitost nasazení aplikace, možnosti konfigurace prostředí a různé omezení platformy. Na každé prostředí bude spuštěn zátěžový test, kde cílem je odhalení chování platformy pod vysokou zátěží. Na základě těchto parametrů budou navrženy možnosti, kdy je danou platformu vhodné využít a upozornit uživatele na potenciální nedostatky.

4.2 Typy testování

Software (včetně webových aplikací) je možné testovat z nejrůznějších pohledů. Zcela automatizovaně je možné testovat funkcionalitu pomocí unit testů nebo integračních testů. Existují testy, které se zaměřují jen na cíl, ale vůbec netestují průběh procesů. Tato diplomová práce se bude zabývat jen výkonnostními testy i přes to, že některé služby aplikace jsou plně pokryty integračními testy. Práce se zaměří především na kategorii stress (load) testů, tedy na stav, kdy je aplikace pod velkým tlakem a je nucená zpracovávat abnormálně velké množství požadavků od svých uživatelů. [27] Je také nutné rozlišovat stress test a load test. Load test si bere za úkol zjistit, jaký maximální výkon daná aplikace má. Hlavní podmínkou je relevantní výstup pro uživatele a žádné nestandardní chování aplikace. Stress test ovšem zkoumá jaké je chování aplikace, pokud dojde k překročení výkonnostních limitů. Práce se bude zajímat výkonem, při kterém je chování standardní a doba odezvy je přijatelná – proto se bude ve zbytku práce hovořit o load testech. [28]
Hned na začátku výběru vhodného softwaru pro load testy byly nastaveny podmínky, které musí aplikace splňovat:

1. škálovatelnost – je nutné, aby bylo možné test spustit z více vláken.
2. multiplatformnost – je možné, že v rámci zvýšení výkonu bude nutné přejít na jiné prostředí a pokud by došlo k výměně operačního systému, nesmí dojít k nekompatibilitě.
3. jednoduché vytvoření scénáře – ideálně v nějakém GUI

4. jednoduché spuštění a zastavení testu – pro případy nouze (např. hroutícího se prostředí)

Pomyslným vítězem se stala aplikace Jmeter, která se v oblastí load testování stala téměř standardem. Není určená jen pro testování webových aplikací, ale poradí si z celou řadou dalších jako např. FTP, databáze, LDAP, SMTP, TCP, atd. Aplikace je napsaná v programovacím jazyku JAVA a má kompletně otevřené zdrojové kódy. Test jde spustit z více strojů pomocí ne zcela intuitivního GUI. Vytváření testů v Jmetru je opravdu jednoduché – program je schopen vytvořit proxy server, který si uživatel poté nastaví do prohlížeče a program zachytává všechny požadavky jdoucí přes něj a kompletuje ho do testu. Tímto způsobem jde "nahrát" chování uživatele a vytvořit tak testovací scénář. Výsledkem je soubor, který obsahuje testovací scénáře a může být spuštěn na jakémkoliv prostředí. [29]

4.3 Testování interpretů

Všechny testy budou provedeny na virtualizovaném stroji, kterému bude poskytnuto 4 GB paměti RAM a jedno jádro procesoru. Hostitelský počítač pak bude obsahovat procesor Intel Core i5-4670K s taktem 3.4 GHz.

4.3.1 Testovací skript

Byl použit již existují a v komunitě osvědčený testovací skript ze serveru www.php-benchmark-script.com. Tento kód testuje interpret v čtyřech skupinách funkcionality. V prví části jsou otestovány matematické funkce od absolutní hodnoty, přes goniometrické funkce až po odmocniny, ve druhé manipulace s řetězci jako např. escapovaní, výpočet md5 hashe nebo vrácení délky řetězce, ve třetí cykly a v poslední čtvrté části podmínka if-else. Testovací skript bude vždy spuštěn v CLI režimu (command line interpret). [30]

4.3.2 Testovaní interpreti

Bylo rozhodnuto, že otestovány budou všechny minoritní verze PHP od verze 5.3 tak, že u každé minoritní verze se otestuje vždy nejvyšší build. K 29. 11. 2015 stále neexistuje stabilní verze PHP 7, proto v této práci bude PHP 7 otestováno ve verzi release candidate 8. Dále bude otestována nejvyšší verze HHVM. Instalace jednotlivých vydání PHP bylo zprovozněno následujícím skriptem.

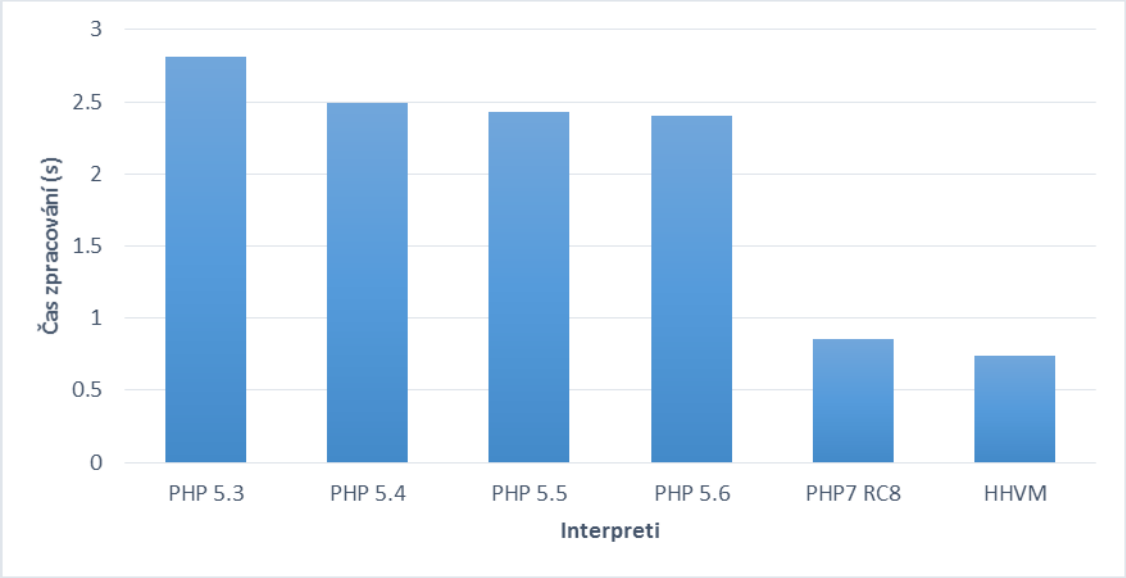
```
apt-get install build-essential libcurl3
wget ODKAZ NA ZDROJOVÉ KÓDY
cd php-x.xx.xxx
./configure
make
make install
```

4.3.3 Výsledky testovaní výkonu interpretů

Tab. 3 Výsledky porovnání interpretů

Test	PHP 5.3.29 Zend Engine v2.3.0	PHP 5.4.45 Zend Engine v2.4.0	PHP 5.5.30 Zend Engine v2.5.0	PHP 5.6.16 Zend Engine v2.6.0	PHP 7.0.0RC8– Zend Engine v3.0.0	HipHop VM 3.10.1
Matematické funkce	0,734 s	0,729 s	0,743 s	0,725 s	0,217 s	0,271 s
Manipulace s řetězci	0,769 s	0,848 s	0,754 s	0,752 s	0,319 s	0,367 s
Cykly	0,802 s	0,553 s	0,563 s	0,559 s	0,189 s	0,035 s
Podmínky	0,503 s	0,362 s	0,363 s	0,367 s	0,128 s	0,067 s
Celkem	**2,808 s**	**2,492 s**	**2,423 s**	**2,403 s**	**0,853 s**	**0,740 s**

Zdroj: Vlastní práce



Obr. 12 Srovnání jednotlivých interpretů

4.3.4 Vyhodnocení výsledků testování

Jak lze vidět z výsledků, největší výkonnostní pokrok nativního interpretu PHP je při skoku z PHP 5 na PHP 7. Jak již bylo zmíněno, v této verzi bylo PHP výrazně přepsáno s vysokým důrazem na výkon. HHVM v tomto testu bylo o malý kousek lepší, ale rozdíl oproti PHP 7 je do jisté míry zanedbatelný a dá se konstatovat, že nejlépe z uvedených testovaným možností se jeví PHP 7, tedy za předpokladu, že v stabilní verze bude dosahovat stejné výkonnosti. Standardní implementace nám oproti HHVM přináší veškerou funkcionalitu PHP, což představuje větší výhodu, než zanedbatelný rozdíl ve výkonu.

4.4 Testování cloudových řešení

4.4.1 Metodika vytvoření Load testu a segmentace uživatelů

V práci *Segmentace uživatelů internetu* autor dle své metodiky rozdělil uživatele do 4 základních skupin, které specifikují jejich dosaženou úroveň práce s počítačem, což bylo převzato do prostředí internetu a schopnosti nakupovat v elektronickém obchodě. Tyto skupiny jsou: [31]

1. uživatelé, kteří nemají s internetem ani s počítačem žádnou zkušenost (důchodci, lidé s nízkými příjmy)
2. uživatelé, kteří se k internetu nepřipojují častěji než několikrát za měsíc (nízké příjmy, hledání praktických informací)
3. uživatelé, kteří se k internetu připojují denně anebo alespoň několikrát do týdne (pracující lidé, školáci)
4. uživatelé, kteří na internetu tráví více než 3 hodiny denně (studenti, počítačový experti)

Tab. 4 Zastoupení jednotlivých skupin uživatelů ve společnosti

Skupina	% zastoupení
1.	9 %
2.	17 %
3.	63 %
4.	11 %

Zdroj: Segmentace uživatelů internetu [31]

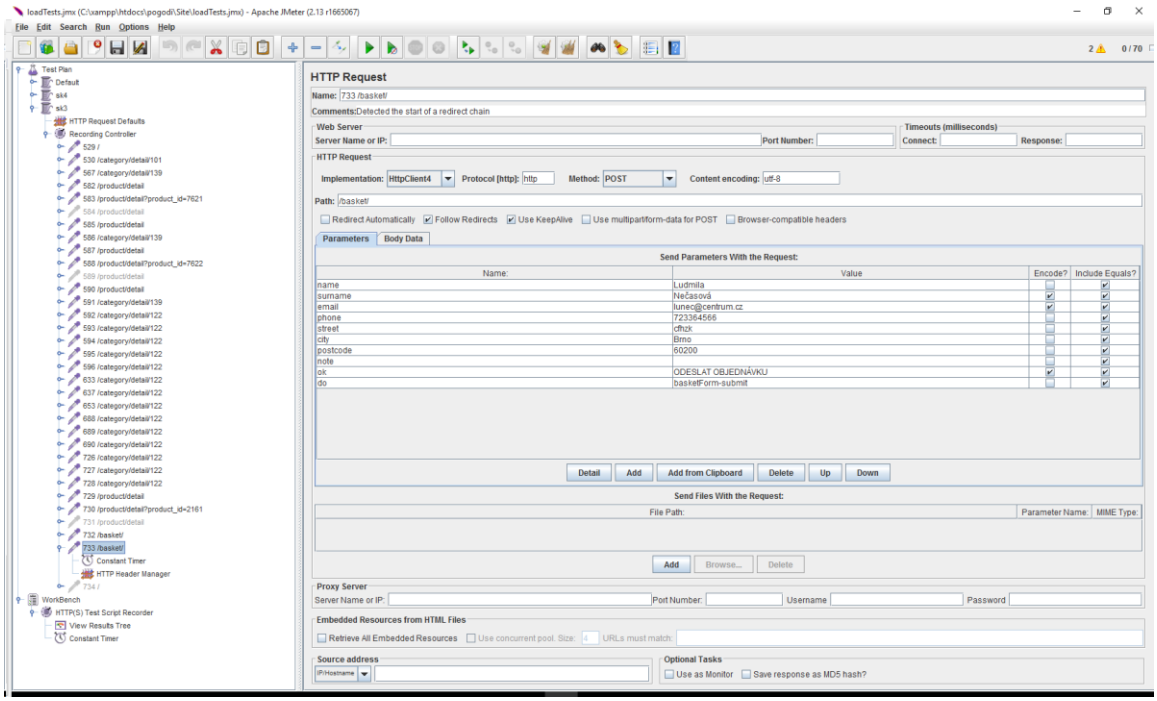
Budou vytvořeny 4 scénáře procházení webu tak, aby bylo zachyceno co možná nejvěrohodnější chování jednotlivých skupin uživatelů. Pro 1. skupinu lidí, kteří neumí s internetem a obecně s počítačem příliš dobře pracovat je plán vytvořit scénář, ve kterém se budou jednotlivé stránky obchodu procházet pomaleji, a důraz bude kladen také na možnost vracení se zpět, jelikož se dá očekávat špatná ori-

entace v hierarchii webu. U 2. skupiny bude procházení rychlejší a stránku budou používat jiným způsobem (např. si nechají posílat informace o slevách do emailu). Pro 3. skupinu tvořenou z již zkušenějších uživatelů bude typické "nebát se na něco kliknout" (např. využití košíku ke zboží, které si chci potencionálně koupit) a obecně bude interakce s aplikací rychlejší. 4. skupina se od 3. v zásadě moc neliší, až na pár drobností, které jsou většinou dány vyšší úrovní znalostí práce na počítači (např. si otevřou velké množství nových panelů s produkty, které se jim líbí, a jsou schopni pracovat třeba i s 50 aktivními panely). U scénářů nebude dodrženo pouze chování jednotlivých skupin, ale také procentuální zastoupení ve společnosti, čímž bude dosažena možnost **měřit výsledky v lidech**. Měřítko v lidech je důležité, protože všechna odhadovaná návštěvnost poskytnutá z ostatních zdrojů jako je marketing nebo počet historických objednávek je vždy udávána v počtu lidí a ne v počtu požadavků na stránku. Celé měření tím získá objektivnější přístup, které jde synchronizovat s dalšími odděleními ve firmě.

V případě testování reálné webové aplikace by byly použity analytické údaje, které by byly využity ke tvorbě scénářů. Bylo by i možné sledovat chování některých reálných návštěvníků a scénáře potom postavit na základě jejich chování. Tyto testy se pak velice přibližují reálnému prostředí a výsledky testu jsou vysoce spolehlivé.

4.4.2 Tvorba Load testu

Jak již bylo zmíněno, Jmeter umí zachytávat průchod webovou aplikací pomocí proxy serveru. Této funkcionality bylo v tomto případě využito. Před tím, než začne samotné zaznamenávání, je nutné Jmeter nastavit. V Jmeteru existuje nepřeberné množství nejrůznějších nastavení, ze kterých bylo využito pouze portu pro proxy server a seznamu vzorů pro adresy, které se mají nebo nemají zaznamenávat a konstantního časovače (Constant Timer) na zaznamenání časové mezery mezi jednotlivými požadavky na aplikaci. Jmeter pak z jednotlivých požadavků dělá záznamy do zvolené testovací skupiny (target controller). Na následujícím obrázku můžete vidět požadavek při odeslání nákupního košíku. Jedná se o požadavek typu POST, který směřuje na adresu */basket/* a obsahuje parametry zadané uživatelem při vyplnění košíku. Jmeter pak zachovává i HTTP hlavičky, kódovaní znaků i prodlevu mezi jednotlivými požadavky pomocí konstantního časovače. Tím je sestaven celý testovací scénář, který je připraven k opětovnému spuštění. Na konci bylo ještě upraveno chování tak, aby navštívený produkt byl vybrán zcela náhodně a nedocházelo tak k požadavkům jen na úzký vzorek stránek, které by se stihly rychle uložit do cache.

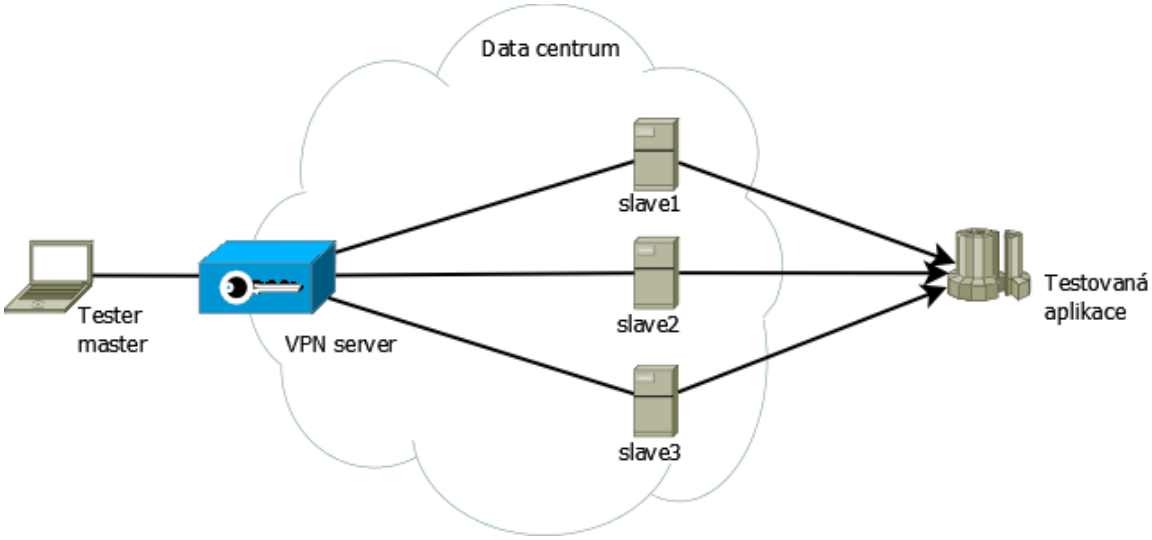


Obr. 13 Ukázka vytvořeného testovacího scénáře v programu jmeter

4.4.3 Distribuované testování

Jmeter je schopen spouštět testy v distribuovaném prostředí tak, že existuje vždy jeden *master server*, který ovládá ostatní *slave servery*. Na master serveru jsou uloženy informace o load testu a je jediný, který fyzicky neodesílá žádné HTTP požadavky na target (testovaná aplikace). Odesílat požadavky je úkolem slave serverů, které pak odesílají výsledky zpět na master. Podmínkou tohoto prostředí je, aby všechny servery byly dostupné na jedné síti a mohli spolu komunikovat. [32] V případě této práce bylo vytvořeno prostředí pěti serverů, které byly umístěny v datacentru, ale požadavkem bylo, aby tester mohl celý průběh testu ovládat ze svého osobního počítače. Z tohoto důvodu byl v datacentru vytvořen ještě jeden server, který bude sloužit jako VPN brána. Tester se pak pomocí VPN připojí přímo do LAN sítě v datacentru a může ovládat všechny slave servery. Aby master věděl o jednotlivých slave serverech, je nutné, aby byla nastavena konfigurační direktiva *remote_host* v souboru jmeter.properties, kde se nachází IP adresy jednotlivých slave serverů oddělené čárkou. Ukázka:

```
remote_hosts=192.168.0.10,192.168.0.11,192.168.0.12,192.168.0.13
```



Obr. 14 Schéma testovací prostředí pro jmeter

Na všech serverech musí být naprosto shodná konfigurace a instalována stejná rozšíření. Poté se spustí Jmeter v režimu slave pomocí příkazu:

```
sh jmeter -Dserver_port=${SERVER_PORT:-1099} -s -j log.txt
```

Spuštění na straně master severu se děje pomocí grafického rozhraní programu Jmeter. Pro každý scénář je nastaven počet testovacích instancí (počet simulovaných uživatelů), rozjezdová doba (zátěž se může postupně zvyšovat) a počet opakování po rozjezdové době. Počet instancí byl vždy nastaven podle kvótního výběru na testovaný počet uživatelů, rozjezdová doba většinou na 10 minut, počet opakování nebyl nastaven a zastavení testu probíhalo vždy ručně (v GUI aplikace). Po zastavení testu bylo nutné vždy všechny slave servery restartovat, protože vždy zůstávají navázaná nějaká spojení mezi master a slave server, který je nutné vyresetovat.

4.4.4 Nasazení aplikace do jednotlivých prostředí

Google App Engine

Hned na začátku je třeba v konzoli Google App Enginu udělat dvě věci. První z nich je vytvoření samotné aplikace a druhou věcí je vytvoření cloud storage (uložiště) pro všechny soubory, které jsou při provozu aplikace ukládány na disk. Poté je již možné přistoupit k samotnému nahrání aplikace do cloudového prostoru. V případě App Enginu je tato operace provedena pomocí softwaru Google App Engine Launcher. Samotný program pak bude v kořenu nahrávaného projektu očekávat konfigurační soubor *app.yaml*, který obsahuje potřebná nastavení pro běh služby. Ukázka konfiguračního souboru:

```
application: pogodioc
version: 1
runtime: php55
api_version: 1
threadsafe: yes

handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: /(.+\.(css|js|gif|png|jpg))$
  static_files: ../www/\1
  upload: www/.*\.(gif|png|jpg)$

- url: (.+).php
  script: \1.php

- url: .*
  script: www/index.php
```

Poté už je možné aplikaci pomocí Google Launchru nahrát a ta se rozjede na subdoméně *nazev_aplikace.appspot.com*

Amazon EC2

Na začátku je nutné vytvořit jednu instanci serveru. Tímto procesem uživatele provede AWS konzole a po cestě si může vybrat z nejrůznějších možností od operačního systému až po zabezpečení. Pro tuto diplomovou práci byl vybrán operační systém Ubuntu 14.04. Po vytvoření instance je možné se k ní připojit a nainstalovat všechny služby, které bude v budoucnu obsluhovat. V tomto případě to byl Apache2 + PHP 5.6. Nahrání zdrojových kódů si uživatel zajistí libovolnou možností. Lze využít např. FTP, SFTP, Git, atp. Amazon tyto možnosti, vzhledem k tomu že se jedná o klasickou instalaci operačního systému, nijak neomezuje. Poté se z instance udělá tzv. obraz (image), je to referenční instance, která se bude v případě potřeby replikovat. To jak a kdy bude k replikaci docházet je nastavitelné v sekci *Auto Scaling Groups,* kde je možné vybrat obraz, maximální a minimální počet instancí, atp. Právě v nastavení Scaling groups je možné konfigurovat, kdy se spustí další instance nebo kdy je možné instanci z poolu odebrat. Poslední sekce nutná pro provoz je vytvoření *Load Balanceru*. Jedná se o místo, na které budou následně všichni návštěvníci nasměrováni a load balancer přesměruje jejich požadavek na konkrétní instanci. Také zde najdeme URL adresu s koncovým bodem, kde je aplikace spuštěna.

Cloud Foundry (IBM Bluemix)

Úvodní část je podobná jako u Googlu, ale není zde nutné vytvářet speciální uložiště pro soubory. Nahrání zdrojových kódu je ale u Cloud Foundry mnohem složitější. K samotnému nahrání slouží konzolová aplikace, kterou je nutné stáhnout a cestu k ní přidat do systémové proměnné *Path*. Poté je již možné využívat programu *cf.exe* v příkazové řádce Windows. Tento program k nahrání všech zdrojových souborů vyžaduje ještě několik dalších konfiguračních souborů. V případě této práce jsou to:

.bp-config/options.json

```
{     "PHP_VERSION": "{PHP_55_LATEST}",
      "COMPOSER_VENDOR_DIR": "vendor",
      "WEBDIR": "public",
      "PHP_EXTENSIONS": ["openssl","fpm","curl","pdo"]
}
```

.cfignore

```
composer.phar
vendor
```

manifest.yml

```
applications:
- name: pogodisite
  memory: 128M
  buildpack: https://github.com/cloudfoundry/php-buildpack.git
  env:
    CF_STAGING_TIMEOUT: 15
    CF_STARTUP_TIMEOUT: 15
```

V souboru *options.json* je definováno spuštění instalace závislostí pomocí composeru, verze PHP a využité rozšíření pro jazyk PHP. Soubor *.cfignore* obsahuje soubory a složky, které budou při nasazování aplikace přeskočeny. Podobnost k tomuto souboru lze nají např. v .gitignore ve verzovacím systému GIT. Posledním konfiguračním souborem je *manifest.yml*, který obsahuje další nastavení služby a také tzv. buildpack. Je to další možnost konfigurace, ale pro potřeby této diplomové práce byl využit již hotový a dokumentací doporučený buildpack. Pokud jsou všechny konfigurační direktivy v pořádku, je nutné zapnout příkazovou řádku, přesunout se do složky s projektem a vyvolat příkaz:

```
cf.exe push
```

Po úspěšném nahraní na server, které trvá i několik minut, je aplikace opět dostupná na subdoméně *nazev_aplikace.mybluemix.net*.

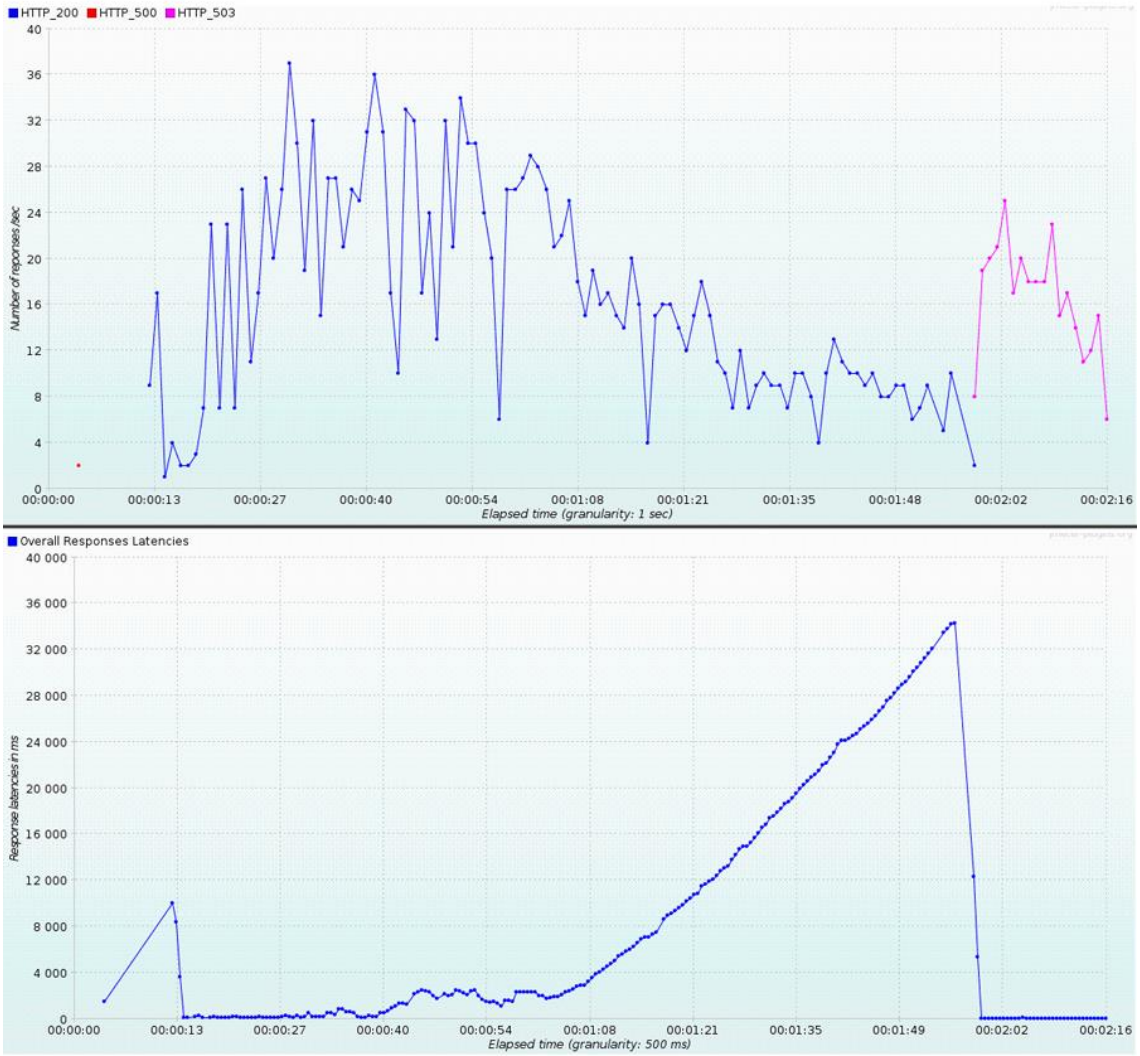
Microsoft Azure
Po vytvoření webové aplikace na hlavní stránce administrace následuje nahrání zdrojových souborů na server. Azure umožňuje hned několik možností jak toho dosáhnout. Mezi nejpříjemnější lze zřejmě zařadit verzovací systémy Git a Mercurial, ale mezi ostatními je možné najít i „exotické" metody jako např. Dropbox. Možností jsou desítky a všechny jsou přehledně popsány v dokumentaci. [33] Opět je nutné provést některá nastavení, aby aplikace fungovala. Nejdůležitější je položka virtuální aplikace a adresáře, které je velice podobné nastavení virtuálního hosta v nastavení serveru Apache. Jde o namapování relativní adresy URL na relativní umístění složky se spouštěcím programem (index.php). Důležité je nastavení používaného balíčku – logika je jednoduchá, čím dražší balíček, tím vyšší výkon serverů a pokročilá nastavení aplikace. Automatické škálování je v prostředí Azure možné až od balíčku Standard, kde se ceny pohybují od 60 EUR za kalendářní měsíc. U nižších balíčků se počítá s manuálním škálováním, kde si uživatel nastaví počet instancí sám. Aplikace se následně rozběhne na interní doméně *nazev_aplikace.azurewebsites.net*.

4.4.5 Vyhodnocení výsledků testování

Distribuované zátěžové testy v programu Jmeter bylo nutné ještě po nějakou dobu ladit, než bylo dosaženo možnosti simulovat alespoň 1000 aktivních uživatelů. I přes to, že v datacentru bylo umístěno hned 5 slave serverů, problémy se nevyhnuly ani této části práce. Největší komplikace způsobilo velké množství paralelně běžících procesů a počet navázaných síťových spojení. Po zdlouhavé konfiguraci od správců hostingové služby bylo dosaženo alespoň zmíněné úrovně 1000 uživatelů, která se stala limitem pro další průběh testování. Průběhy jednotlivých testů jsou detailně popsány v následujícím textu.

Google App Engine
Krátce po začátku testování se začaly projevovat nedostatky této platformy, protože načtení jedné stránky zabralo v průměru něco kolem 4 sekund. Poté byly provedeny dodatečné testy, které měly za cíl odhalit, čím bylo toto zpoždění způsobeno. Ukázalo se, že ze 4 sekund zpracování požadavku 3,5 sekundy zabralo čtení a zápis do souborů, kde se zapisovala nebo četla zacachovaná konfigurace aplikace. I přes to, že v dokumentaci je uvedeno, že standardní funkcionalita pro práci se soubory je na této platformě možná (až na podporu některých funkcí), není možné ji na plno využit na produkčním prostředí. Existuje však alternativa v podobě API pro práci se soubory, ale ta ovšem v této práci nebyla vyzkoušena. Tato změna by si vyžádala obrovské refaktorování nejen kódu aplikace, ale také využitých kniho-

ven a frameworku. Google App Engine by byl použitelný pouze za předpokladu, kdy by aplikace byla vyvíjena přímo na tuto platformu a veškerá funkcionalita by jí byla plně přizpůsobena. Byl proveden i zátěžový test, ze kterého však lze konstatovat, že Google se jako prostředí pro provoz hodí pouze v případě, že je na něj aplikace přímo přizpůsobena nebo v aplikacích, kde není nutné komunikovat se souborovým systémem. Ovšem všechny dnešní moderní frameworky tuto interakci vyžadují, takže by se zřejmě muselo jednat o aplikaci napsanou v čistém PHP s abstraktní vrstvou komunikace se souborovým systémem.



Obr. 15 Měření provedené na Google App Enginu

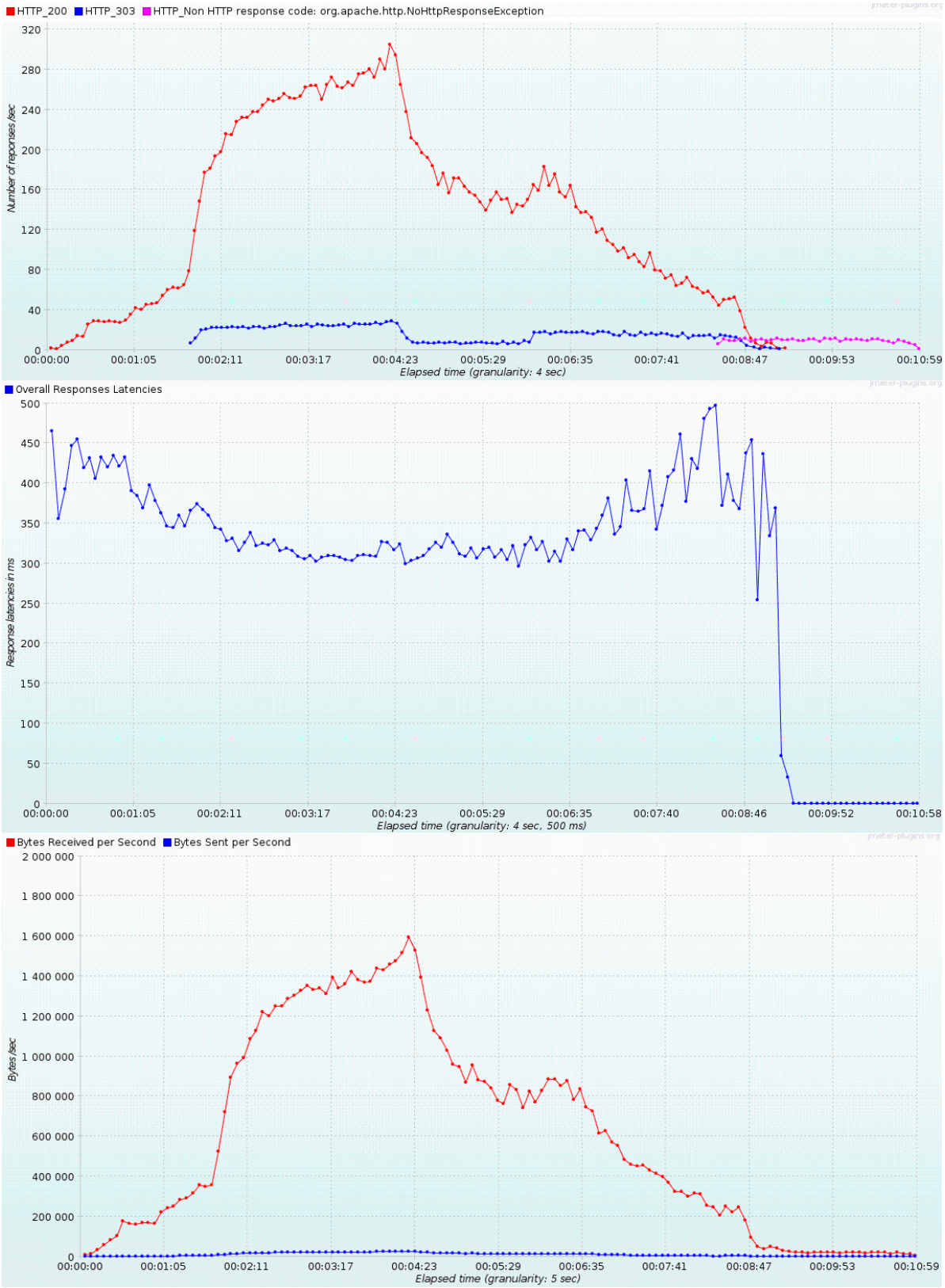
Při zátěži cca 200 uživatelů dosahovala průměrná odezva hranice 5 sekund a chybovost dosahovala až 30 %, což je pro provoz absolutně nepřípustné. Se zvyšující se zátěží tyto čísla ještě rostla, takže nemělo cenu zkoušet větší zátěž a test byl pře-

rušen. Jak je z grafu patrné, při stále rostoucí zátěži začala celá platforma postupně kolabovat.

Amazon EC2

Z testovaných možností dopadl pro tuto konkrétní aplikaci Amazon zřejmě nejlépe. Největším problémem bylo správné nastavení pro škálování. Amazon umožňuje tyto parametry libovolně nastavovat a trvá delší dobu, než majitel aplikace nalezne vhodnou konfiguraci pro svojí aplikaci. Zde bylo nastaveno spuštění další instance za předpokladu vytížení 70 % procesoru nebo 70 % paměti RAM v průměru za poslední uplynulou minutu. Minimální počet instancí byl stanoven na 10. V případě větších projektů by ovšem žádné problémy nastat neměly, protože se dá předpokládat, že v delším období bude nárůst zátěže postupný a to dává správcům čas na přípravu a vhodnou konfiguraci.

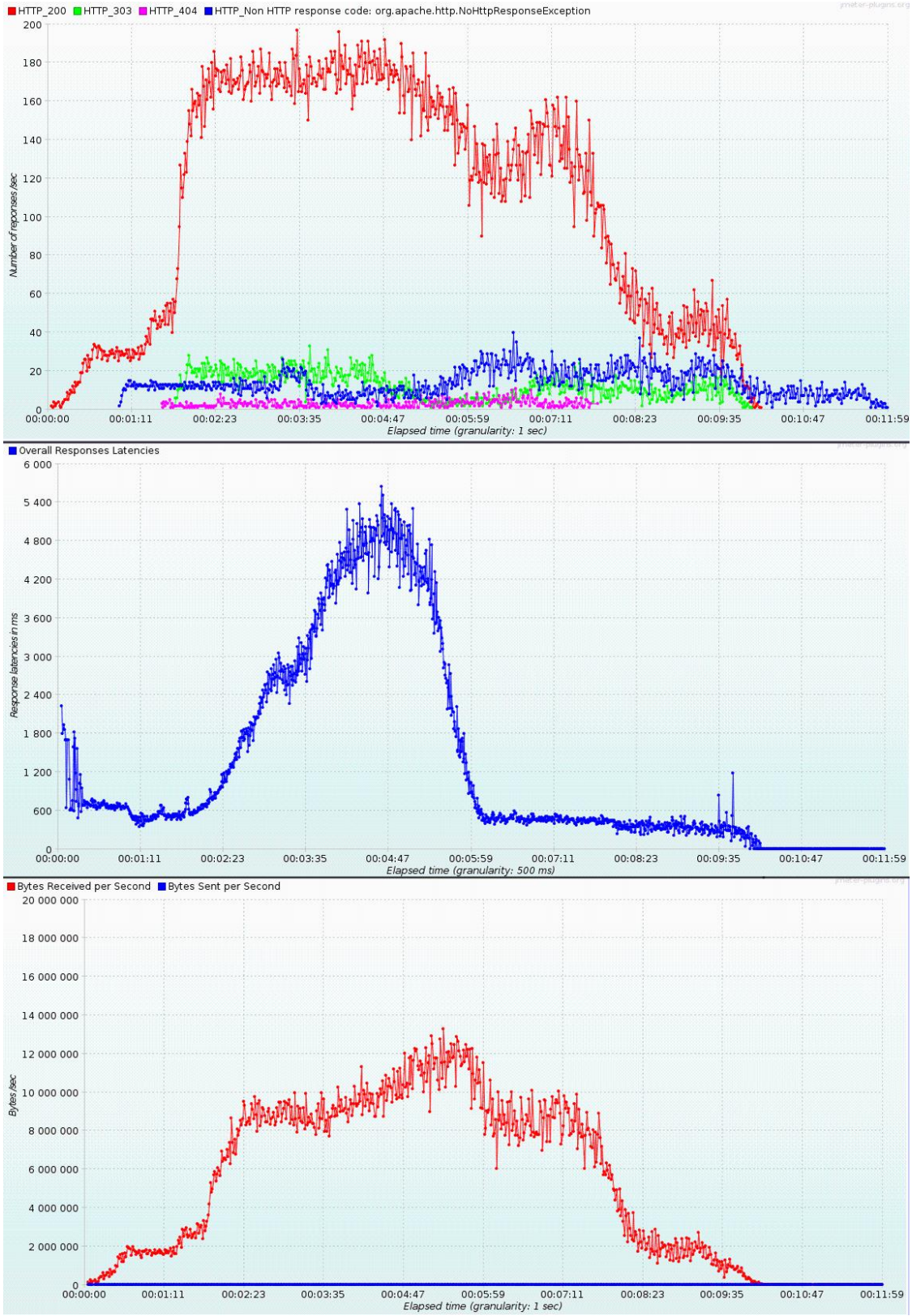
Jelikož již byly provedeny nějaké předběžné testy, při který platforma vykazovala vysokou odolnost proti zátěži, byla nastavena kratší rozjezdová doba. Z obrázku jde vidět, že absolutního vrcholu, tedy nejvyšší zátěže, bylo dosaženo již po necelých pěti minutách. V tu chvíli na aplikaci chodilo zhruba 300 požadavků každou sekundu a platforma na ně stíhala odpovídat při průměrné době odezvy okolo 400 ms. Tzn., že v by bez problému dokázala obsluhovat i 1000 uživatelů v jeden okamžik a zřejmě by bylo možné se dostat ještě mnohem dál. Vykazované hodnoty jsou více než velice dobré a platforma by bylo schopna tuto konkrétní aplikaci zvládat i v produkčním prostředí. Poté byla zátěž snižována a nakonec byl test ukončen bohužel ne zcela korektním způsobem, což způsobilo na grafu určitou chybovost (růžové tečky na prvním grafu), které ovšem nelze do výsledků zahrnout. Před začátkem testu bylo nastartováno 10 serverových instancí a během testu si Amazon automaticky ještě 4 přidal. Tzn., že v absolutním maximu celou platformu obsluhovalo 14 instancí v neplacené variantě t2.micro s jedním jádrem procesoru a 1 GiB RAM.



Obr. 16 Měření provedená na Amazon EC2

Cloud Foundry (IBM Bluemix)

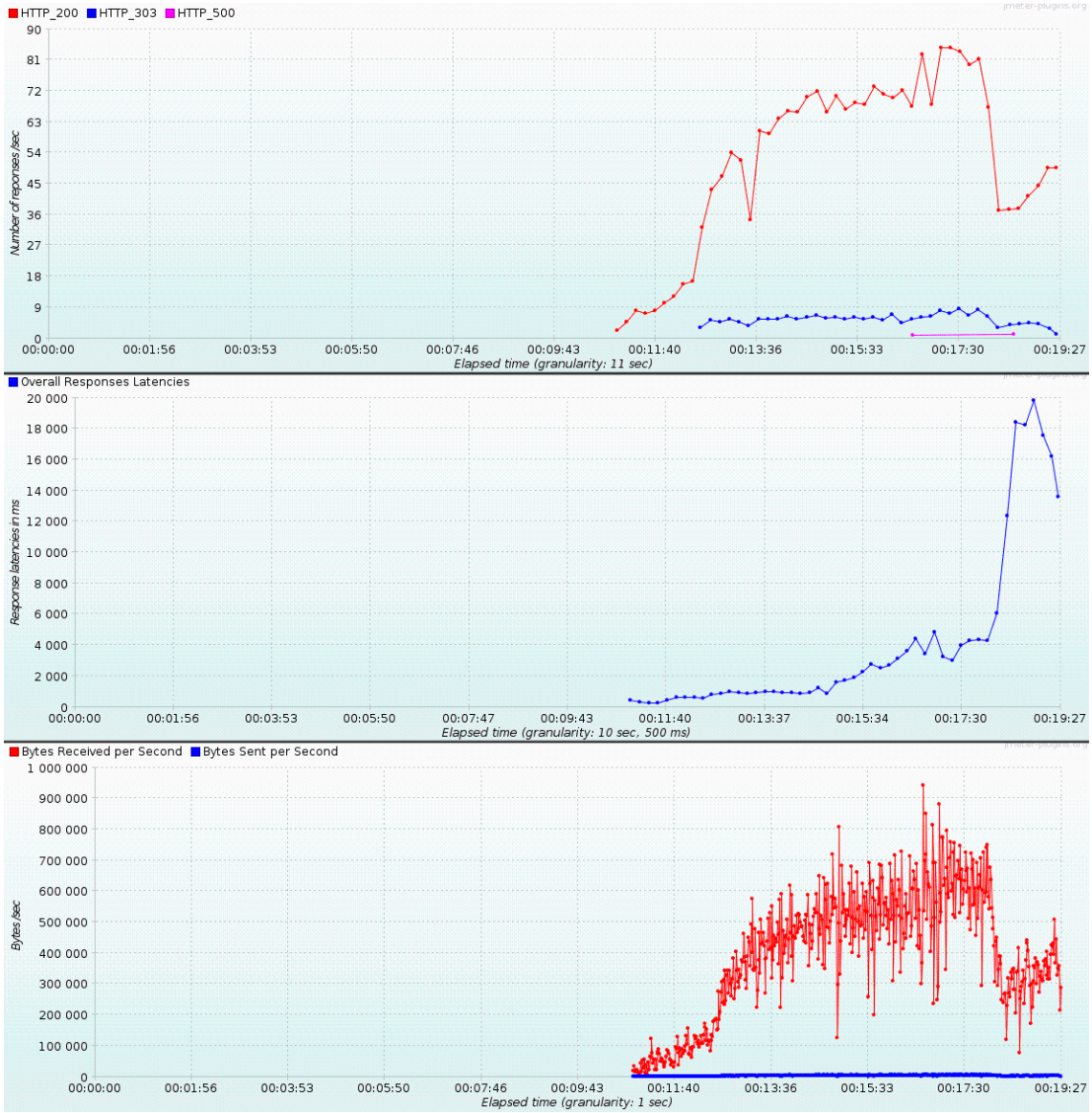
Pro platformu Cloud Foundry od IBM Bluemix bylo k dispozici 10 instancí s 2 GB RAM pro každou z nich. Informace o jádrech procesoru bohužel nedává IBM k dispozici. Na této platformě se také ukázalo, že pro provoz PHP aplikace není zcela vhodná. Z testovaných řešení se umístila na pomyslném druhém místě, kdy i přes nějaký podíl chybových odpovědí dokázala snášet zátěž přes 700 připojených uživatelů. Při této zátěži se pohybovala chybovost okolo 3 % a odezva se držela pod 2,5 sekundy. S nárůstem ještě vyšší zátěže však chybovost vzrostla na úroveň přesahující 10 % a zároveň s tím rostla i odezva. Po přerušení testu lze na grafech vidět stabilizaci platformy, které je doprovázeno snižující se odezvou.



Obr. 17 Měření provedená na IBM Bluemix

Microsoft Azure

I přes velké očekávání nedopadlo testování platformy Microsoft Azure úplně nejlépe. Pro testování byla pořízena varianta S3 standard, kde bylo k dispozici 10 instancí se 4 jádry a 7 GB paměti RAM. Jedná se o druhý nejlepší balíček nabízený touto platformou. Po spuštění testu se platforma chovala stabilně, ale na úrovni 60 požadavků za sekundu začala být nestabilní. Výrazně začala růst odezva a objevily se i chyby. Při zátěži 80 požadavků za sekundu odezva začala dosahovat až 20 sekund a celý test byl v tomto bodě ukončen. I přes pokusy odhalit příčinu tohoto podivného chování nebylo bohužel nic nalezeno a platforma musela být označena jako nedostatečná.



Obr. 18 Měření provedená na Microsoft Azure

5 Možnosti vylepšení výkonnosti

Výkonnost aplikace lze zvyšovat nejrůznějšími způsoby od vylepšování algoritmického řešení jako takového (časová a prostorová složitost, paralelizace) po využití nejrůznějších nástrojů sloužící pro tyto účely (indexy v databázi, cache). Kolem komunity PHP se točí celá řada nástrojů, které si svoje místo ve světe PHP vydobyli především podporou pro tento jazyk nebo vstřícným chováním jejich autorů. Většina z nich je uvolněna pod volnou licencí a některým z jich bude věnován kousek v následujícím textu o dalších možnostech zlepšení výkonnosti.

5.1 Časová a prostorová složitost

Aplikací je možné rozložit do velkého množství procedur a funkcí, které vždy řeší danou úlohu. Na každou takovou úlohu pak lze pohlížet jako na samostatný algoritmus, ke kterému lze vždy vypočítat jeho časovou a prostorovou složitost. V ideálním případě by měl programátor při vytváření jednotlivých algoritmů využít nejlepší možná řešení, která by mu zajišťovala co možná nejnižší požadavky na čas a paměť, ale skutečnost bývá taková, že je vždy co vylepšovat. Nemusí se nutně vždy jednat o přepsání celého algoritmu do třídy s nižší asymptotickou složitostí, ale například o odstranění nepotřebné alokované paměti (nevyužité proměnné) nebo o odstranění nepotřebného výpočtu. Nedá se však od této možnosti čekat zrychlení ve stovkách procent. V drtivé většině případů se bude jednat o kosmetické úpravy a po výkonností stránce se bude jednat spíše zanedbatelný pokrok. Příkladem, kdy bude možné využít této možnosti optimalizace, je zpracování velkého XML souboru, kdy přepis do podoby využívající knihovny XMLReader, která XML zpracovává jako stream, ze zpracování pomocí DOMDocument může přinést snížení prostorové složitosti i o stovky procent. [34]

5.2 Optimalizace báze dat

Možností, jak ukládat data v aplikaci, je v dnešní době nespočetné množství. Důležité je si uvědomit, jaké data aplikace obsahuje a přizpůsobit jim i optimální způsob jejich ukládání.

V případě této diplomové práce je to relační databáze, která se výborně hodí pro většinu elektronických obchodů a informačních systémů.

5.2.1 Relační databáze

Data jsou zde uloženy v jednotlivých tabulkách a pro manipulaci s daty se stal standardem jazyk SQL. Hodí se pro většinu ekonomicky zaměřených aplikací, jako jsou obchody nebo nejrůznější informační systémy. Jsou to zřejmě nejrozšířenější možnost uchovávání dat ve webových aplikacích. [34] U většiny hostingových spo-

lečností je relační databáze prodávána přímo u webhostingové služby a směruje tím programátora ji využívat i v případech, kdy to není úplně vhodné. Nejrozšířenějšími relačními databázovými systémy jsou MySQL, Postgres. Oba tyto systémy jsou zdarma a pro drtivou většinu aplikací jsou dostatečné. Pokud již aplikaci nestačí, je možné nasadit placené systémy jako je např. Oracle.

5.2.2 Objektové databáze

Myšlenka tohoto typu uložiště dat je přiblížení aplikace a databáze, kdy by se ukládaly přímo objekty (instance tříd) z aplikace. V PHP není tento typ příliš podporován, ale komunita se snaží tento nedostatek odstranit různými implementacemi ORM (objektově relační mapování) knihoven jako je např. Doctrine. Tyto knihovny jsou pak schopné převádět objekty do různých databázových systémů a tím vytvořit abstraktní vrstvu mezi aplikací a databázovým systémem. [35] Tato abstrakce není vždy úplně vhodná a u některých typů aplikací to může způsobit nemožnost naprogramovat určitou funkcionalitu. [36] Tento přístup je však momentálně velice populární.

5.2.3 Dokumentové databáze

Dokumentové databáze umožňují uložení strukturovaných dat (dokumentů), typicky ve formátu JSON nebo XML. Mezi dokumenty je pak následně možno vyhledávat, mazat nebo aktualizovat záznamy pomocí těchto formátů. [37] Výhodou těchto databází oproti relačním je jejich rychlost, která ovšem výrazně klesá s množstvím uložených dokumentů. Mezi podporovanými typy pro jazyk PHP jsou zejména MongoDB a Couchbase.

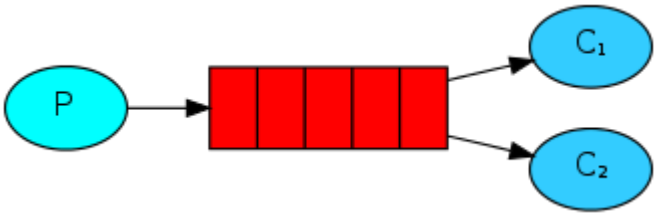
5.2.4 Ostatní

Existuje i mnoho dalších typů systémů pro ukládání specifických dat jako grafy, kde je vhodné využít např. neo4j. Naprosto specifickým formátem jsou geoprostorová data, kde je možné použít např. Postgis. Existuje i nespočet dalších, ale jako ukázka možností je pro tuto práci dostačující těchto pár. U specifických typů si lze povšimnout nedostatku ORM, kdy je nutné kombinovat dva nebo více databázových systémů. Je vždy nutné vědět co a kam budeme ukládat a podle těchto zjištění navrhnout vhodnou aplikační vrstvu, která bude s databázovými systémy komunikovat.

5.3 Zařazení požadavků do fronty

Čas zpracování nějaké akce lze zrychlit tak, že některé kroky procesu přeskočíme a tyto vynechané segmenty zařadíme do fronty pro následné zpracování. Typickým příkladem použití teto metody je odeslání emailu po dokončení objednávkového

procesu v elektronickém obchodu. Email nemusí být odeslán neprodleně při odeslání objednávkového formuláře, ale může několik minut počkat. Stačí si do fronty přidat záznam, že tento email má být odeslán. Výhodou je možnost paralelního zpracování fronty – může existovat více konzumentů. Implementace fronty je možná pomocí tabulky v relační databázi, kde ovšem vzniká problém v případě více konzumentů, protože vždy když si bude konzument brát další práci, bude nutné tabulku zamknout pro čtení i zápis. Další výhodou je možnost naprogramovat konzumenta v zcela jiném programovacím jazyce optimalizovaném pro danou úlohu. Další možnosti vytvoření fronty je využití specializované služby jako je např. RabbitMQ, který má oficiální knihovnu pro jazyk PHP. [38]



Obr. 19 Schéma RabbitMQ pro více konzumentů

5.4 Optimalizace vyhledávání

Vyhledávání jako takové je důležité v každé aplikaci. Otázkou je, do jaké míry je v dané konkrétní aplikaci vyhledávání využito. Pokud se jedná o filtraci do výpisu kategorie, popř. seřazení podle klíčových atributů, bude zřejmě stačit využít relační databázi. Pokud ovšem bude třeba dělat pokročilejší selekci nad daty včetně fulltextového vyhledávání, jako lepší varianta se nabízí využít službu, která slouží přesně pro tyto účely. Lze využít např. Elasticseatch, což je vyhledávácí služba postavená na Apache Lucene. Lucene je fulltextový vyhledávací engine pro textová data, jehož největší výhody jsou vysoká výkonost a možnost škálovatelnosti. S nadstavbou Elasticsearch pak aplikace komunikuje pomocí RESTfull API a data jsou vracena ve formátu JSON. Pro PHP existuje oficiální knihovna pro komunikaci s Elasticsearch. Zřejmě z tohoto důvodu je mezi PHP komunitou tak rozšířený a stal se standardem v oblasti vyhledávání ve webových aplikacích naspaných v jazyce PHP. [39]

5.5 Cachovaní

Cachováním se rozumí uložení výsledku nějaké spouštěné úlohy. Pokud je tato úloha vyvolávána opakovaně, je možné přeskočit celý výpočet a rovnou vrátit výsledek, který je uložený v cache. V aplikaci je nutné odhalit dlouho trvající bloky, kde by cache mohla pomoci. Nemusí se jednat ani o výpočetně složité úlohy, ale např. i o čtení nějakých dat ze souboru, volání API nebo složitější dotaz

do databáze. Cache bývá několikanásobně rychlejší, ale bohužel to s sebou přináší i určité nevýhody, kde největším problémem je neaktuálnost dat. Např. pokud by byla zacachována informace o ceně produktu a někdo ji v administraci elektronického obchodu změnil, k její změně na stránce může dojít až po expiraci cache. Proto je nutné najít vhodnou dobu, po kterou mohou být data uloženy nebo programově najít způsob, jak odstranit takto změněný záznam z cache (nemusí vždy být úplně jednoduché). Cachovat je možné hned na několika vrstvách aplikace - jednotlivé možnosti a technologie budou představeny v následujícím textu.

5.5.1 Souborová cache

Všechny moderní PHP frameworky umožňují cachování do souborů. Většinou je tato možnost implementována jako uložení serializovaného objektu (funkce *serialize*). Toto řešení není v zásadě špatně – je to relativně rychlé a jednoduché na implementaci. Komplikace přicházejí s použitím více aplikačních serverů, kde by si každý musel držet svoji cache nebo by bylo nutné vytvořit napříč všemi servery jeden souborový systém. Tuto možnost lze doporučit menším stránkám, kde se počítá pouze s jedním aplikačním serverem.

5.5.2 Memcache

Memcache je cachovací služba s otevřeným zdrojovým kódem. Jedná se o distribuovaný systém, kde jsou všechny data uloženy v paměti systémem key-value (klíč-hodnota). Memcache je používána do oblastí webů ke cachování výsledků API dotazů, segmentů stránky, atp. Velkou výhodou Memcache je přímá podpora ze strany PHP, kde je možné využít rozšíření (extension) Memcache. [40] V praxi však nastává problém s replikací klíčů po jednotlivých serverech. Memcache od určitého počtu požadavků na zápis dat přestane odpovídat a data ukládat, což může vést až k úplnému zhroucení celé platformy. Proto se toto řešení doporučuje pro středně velké webové aplikace, kde se nepočítá s více instancemi pro cachovací servery.

5.5.3 Couchbase

Couchbase je NoSQL databáze. Jedná se také o distribuovaný systém, který podporuje všechny standardní operace nad daty. Lze také využít jako key-value (klíč-hodnota) uložiště - cache. Pokud se Couchbase využívá tímto způsobem, je velice podobná Memcache. [41] Couchbase ovšem odstraňuje její hlavní nevýhodu, tedy replikaci dat na více serverů. V Couchbase je tato funkcionalita implementována lépe a není problém mít více instancí cache serveru s podporou vyrovnání zátěže. Jedná se ovšem o robustnější řešení a proto se dá doporučit až pro rozsáhlejší systémy.

5.5.4 Redis

Redis je také open source služba pro ukládání strukturovaných dat, kde je možné ho použít jako samotnou databázi nebo jako cache. Redis nabízí podporu pro standardní typy dat jako jsou řetězce, množiny, pole a navíc podporuje ukládání a operace s prostorovými daty. [42] Redis si v PHP komunitě vysloužil pozici jako vhodný nástroj pro ukládání sessions, k čemuž se náramně hodí.

5.5.5 Varnish

Varnish je reverzní HTTP proxy server, který se někdy označuje také jako HTTP akcelerátor. Reverzní proxy je server, který se klientům zobrazuje jako primární server. Cílem je, aby již na tomto proxy serveru byla připravená (nacachovaná) odpověď a požadavek nebyl propuštěn na aplikační servery. Samozřejmostí je možnost nastavit politiku cachování, protože mohou existovat stránky, které není žádoucí mít na Varnish serveru (např. proces nákupního košíku). U e-commerce aplikací je možné Varnish využít na drtivou většinu stránek, kde míra zachycení požadavku na reverzní proxy může dosahovat i 90 % všech požadavků, které směřují na aplikaci (společnost Dixons Carphone). Je jasné, že tato technologie má na výkon aplikace obrovský vliv, ale je důležité zmínit, že možnost využití reverzní proxy je pro každou aplikaci indivituální a není možné udělat nějaké generalizované prohlášení. Varnish má i další možné využití jako je firewall pro webovou aplikaci, obrana proti DDoS útokům, vyrovnání zátěže (load balancer), atd. Schéma komunikace s reverzní proxy. [43]

6 Zhodnocení práce a závěr

Existuje nespočet způsobů, jak lze provozovat webovou aplikaci. Vždy je ale třeba dbát na skutečnosti, že každá aplikace je unikátní, a nejde dělat generalizované závěry. Každá aplikace má svoje specifické vlastnosti, které vyžadují specifické požadavky na platformu. I přes tento fakt byl proveden průzkum možností ze dvou pohledů. První průzkum byl proveden na poli interpretů jazyka PHP, kde výsledky ukázaly, že nemá cenu s interprety nijak experimentovat, protože výkonnostní parametry nové verze PHP 7 běžící na Zend enginu 3 jsou srovnatelné s dalším porovnávaným interpretem HHVM od společnosti Facebook. I přes drobnou výkonnostní převahu HHVM nemá význam využívat toto řešení, ale použít nativní a lety prověřené řešení ve formě Zend Enginu. Druhou částí byl průzkum cloudových řešení, které současný trh nabízí. Mezi testované platformy byly zahrnuty tyto 4 služby: Google App Engine, Amazon EC2, IBM Bluemix a Microsoft Azure. Na každé z těchto prostředí byla nasazena e-commerce aplikace, která taktéž vznikla na základě této práce. Aplikace byla navržena s hlavním požadavkem na co možná největší podobnost s rozsáhlým e-commerce řešením, a tím dostat z testování co možná nejvíce vypovídající výsledky. S nasazením aplikace napříč platformami se nesly nejrůznější komplikace, se kterými se bylo nutné vypořádat. Každá z platforem má svoje specifické vlastnosti, které omezují možnosti aplikace. Jednalo se např. o práci se soubory nebo manipulací s cache. Ukázalo se, že z testovaných řešení vykazují největší odolnost IBM Bluemix a Amazon EC2. Google App Engine ztroskotal na práci se souborovým uložištěm, kdy nedokázal zapisovat a číst soubory pomocí nativních funkcí implementovaných v PHP. U platfomy Microsoft Azure se aplikace pod nárůstem zatížení začala hroutit. Nebylo však možné identifikovat zdroj problémů a testování této platformy bylo ukončeno. IBM Blumix vykazoval poměrně slušné výsledky i pod zvyšující se zátěží, ale výkonům Amazonu se ani nepřiblížil. Amazon dokázal bez jakýchkoliv komplikací unést zátěž přes 1000 uživatelů a jeho limity nebyly zdaleka vyčerpány. Vyrovnávání zátěže fungovalo na výbornou a v okamžiku nejvyššího zatížení celý systém obsluhovalo 14 aplikačních serverů. Pokud by bylo zátěž ještě zvýšena, Amazon by zřejmě nastartoval další instance a zátěž by ustál. Je důležité zmínit, že za celou dobu zátěžového testu nepřekročila odezva platformy hodnotu 500 ms. Poslední části práce bylo zhodnoceny možností, které mohou pomoci s vysokou zátěží, kde byly představeny nejrůznější nástroje, které jsou v PHP komunitě využívané a disponují přímou podporou pro tento jazyk.

PHP jako takové se postupem času zbavuje nálepky méněcenného jazyka, a dokazuje, že se řadí mezi skupinu technologií, která je ve světě velice populární, s čímž narůstá i počet služeb zajišťující jeho provoz. V dnešní době již není problém provozovat webové stránky s miliony návštěvníky denně na pronajaté platformě v cenové hladině tisíců korun měsíčně. Tyto služby zajišťují velkou část nut-

ných administračních činností a snaží se možnosti konfigurace zabalit do přehledných webových administrací, na které již není třeba špičkových specialistů v oboru, ale postačí uživatelé, kteří si bezpečně a spolehlivě zajistí správu celé webové aplikace sami. To s sebou samozřejmě přináší i celou řadu nevýhod způsobenou především nemožností konfigurovat veškeré možnosti platformy nebo vynucení souhlasu s právním ujednáním od poskytovatele. I přes tyto nevýhody jsou tyto služby vhodné především s cenových důvodu, kde se cena pohybuje výrazně níže, než kdyby si majitel aplikace stavěl platformu svépomocí.

7 Literatura

[1] *Programming Language technologies Web Usage Statistics* [online]. [cit. 2015-12-19]. Dostupné z: http://trends.builtwith.com/framework/programming-language

[2] *Programming Language usage in Czech Republic. Builtwith* [online]. [cit. 2015-12-19]. Dostupné z: http://trends.builtwith.com/framework/programming-language/country/Czech-Republic

[3] *PHP: History of PHP - Manual* [online]. [cit. 2015-12-19]. Dostupné z: http://php.net/manual/en/history.php.php

[4] PHP: Releases. *Php.net* [online]. [cit. 2015-12-19]. Dostupné z: https://secure.php.net/releases/

[5] HUJER, Martin. Jaké novinky přinese PHP 7. *Zdroják.cz* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://www.zdrojak.cz/clanky/jake-novinky-prinese-php-7/

[6] HOMESCU, Andrei a Alex SUHAN. HappyJIT: A Tracing JIT Compiler for PHP. *Donald Bren School of Information and Computer Sciences @ University of California, Irvine* [online]. 2011 [cit. 2015-12-19]. Dostupné z: http://www.ics.uci.edu/~ahomescu/happyjit_paper.pdf

[7] Niklasvh/php.js. *Github* [online]. 2014 [cit. 2015-12-19]. Dostupné z: https://github.com/niklasvh/php.js

[8] PHP compiler for .NET | The PHP language compiler for .NET Framework. *Phalanger* [online]. [cit. 2015-12-19]. Dostupné z: http://www.php-compiler.net/

[9] *Phc -- the open source PHP compiler* [online]. 2011 [cit. 2015-12-19]. Dostupné z: http://phpcompiler.org/

[10] *ROSE compiler infrastructure* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://rosecompiler.org/

[11] *Roadsend* [online]. 2014 [cit. 2015-12-19]. Dostupné z: http://www.roadsend.com

[12] Home · bschmalhofer/pipp Wiki. *Github* [online]. 2010 [cit. 2015-12-19]. Dostupné z: https://github.com/bschmalhofer/pipp/wiki

[13] *Caucho Resin : Reliable, Open-Source Application Server* [online]. [cit. 2015-9-3]. Dostupné z: http://quercus.caucho.com

[14] Pie-interpreter/pie. *Github* [online]. 2013 [cit. 2015-12-19]. Dostupné z: https://github.com/pie-interpreter/pie

[15] Zend Engine. *Wikipedia, The Free Encyclopedia* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Zend_Engine&oldid=693632790

[16] SIMMERS, Brett. Faster and Cheaper: The Evolution of the hhvm JIT. *Hhvm.com* [online]. 2013 [cit. 2015-12-19]. Dostupné z: http://hhvm.com/blog/2027/faster-and-cheaper-the-evolution-of-the-hhvm-jit

[17] *Unsupported: Others* [online]. [cit. 2015-12-19]. Dostupné z: https://docs.hhvm.com/hack/unsupported/others

[18] ANTHONY T. VELTE, Anthony T.Toby J. *Cloud computing a practical approach*. New York: McGraw-Hill, 2010. ISBN 978-007-1626-958.

[19] Google App Engine for PHP. *Google Cloud Platform* [online]. 2015 [cit. 2015-10-28]. Dostupné z: https://cloud.google.com/appengine/docs/php/

[20] Reading and Writing Files — Google Cloud Platform. *Google Cloud Platform* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://cloud.google.com/appengine/docs/php/googlestorage/

[21] Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS. *Amazon Web Services* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://aws.amazon.com/ec2/

[22] Bosh. *Bosh* [online]. [cit. 2015-12-19]. Dostupné z: https://bosh.io/docs/about.html

[23] Bluemix overview. *IBM Bluemix* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://www.ng.bluemix.net/docs/overview/index.html#ov_arch

[24] *Microsoft Azure: Cloud Computing Platform and Services* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://azure.microsoft.com/en-gb/

[25] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003, xxiv, 533 p. ISBN 03-211-2742-0.

[26] MVC aplikace & presentery | Nette Framework. *Nette* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://doc.nette.org/cs/2.3/presenters

[27] MYERS, Glenford J, Corey SANDLER a Tom BADGETT. *The art of software testing*. 3rd ed. Hoboken, N.J.: John Wiley & Sons, 2012, xi, 240 p.

[28] *Load vs. Stress testing - Stack Overflow* [online]. 2012 [cit. 2015-12-19]. Dostupné z: http://stackoverflow.com/questions/9750509/load-vs-stress-testing

[29] *Apache JMeter - Apache JMeter&trade* [online]. [cit. 2015-12-19]. Dostupné z: http://jmeter.apache.org/

[30] *Free PHP Benchmark Performance Script* [online]. [cit. 2015-12-19]. Dostupné z: http://www.php-benchmark-script.com/

[31] *Segmentace uživatelů internetu - Seminarky.cz* [online]. [cit. 2015-12-19]. Dostupné z: http://www.seminarky.cz/Segmentace-uzivatelu-internetu-diplomova-prace-6121

[32] *JMeter Distributed Testing Step-by-step* [online]. [cit. 2015-12-19]. Dostupné z: http://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.pdf

[33] *Azure App Service deployment documentation* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://azure.microsoft.com/cs-cz/documentation/articles/web-sites-deploy/

[34] VRÁNA, Jakub. *1001 tipů a triků pro PHP*. Vyd. 1. Brno: Computer Press, 2010, 456 s. ISBN 978-80-251-2940-1.

[35] KOFLER, Michael. *Mistrovství v MySQL 5: [kompletní průvodce webového vývojáře]*. Vyd. 1. Brno: Computer Press, 2007, 805 s. Mistrovství. ISBN 978-80-251-1502-2.

[36] *Getting Started with Doctrine — Doctrine 2 ORM 2 documentation* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/tutorials/getting-started.html

[37] *The MongoDB 3.2 Manual — MongoDB Manual 3.2* [online]. 2015 [cit. 2015-12-19]. Dostupné z: https://docs.mongodb.org/manual/?_ga=1.32690686.1551448386.1450561838

[38] *RabbitMQ - Getting started with RabbitMQ* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://previous.rabbitmq.com/v3_4_x/getstarted.html

[39] VLČEK, Lukáš. Elasticsearch: Vyhledáváme hezky česky. *Zdroják.cz* [online]. 2013 [cit. 2015-12-19]. Dostupné z: https://www.zdrojak.cz/clanky/elasticsearch-vyhledavame-cesky/

[40] *PHP: Memcache - Manual* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://php.net/manual/en/book.memcache.php

[41] *Document Database* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://www.couchbase.com/nosql-databases/couchbase-server

[42] *Redis* [online]. [cit. 2015-12-19]. Dostupné z: http://redis.io/

[43] VELÁZQUEZ, Francisco, Kristian LYNGSTØL, Tollef FOG HEEN a Jérôme RENARD. *The Varrnish Book* [online]. 2015 [cit. 2015-12-19]. Dostupné z: http://info.varnishsoftware.com/the-varnish-book

Přílohy

A Benchmark skript na testování interpretů

```
<?php

function test_Math($count = 140000)
{
      $time_start = microtime(true);
      $mathFunctions = array("abs", "acos", "asin", "atan",
            "bindec", "floor", "exp", "sin", "tan", "pi", "is_finite",
            "is_nan", "sqrt");
      foreach ($mathFunctions as $key => $function) {
            if (!function_exists($function))
                  uset($mathFunctions[$key]);
            }
      for ($i = 0; $i < $count; $i++) {
        foreach ($mathFunctions as $function) {
            $r = call_user_func_array($function, array($i));
        }
    }
    return number_format(microtime(true) - $time_start, 3);
}


function test_StringManipulation($count = 130000)
{
    $time_start = microtime(true);
    $stringFunctions = array("addslashes", "chunk_split", "metaphone",
      "strip_tags", "md5", "sha1", "strtoupper", "strtolower",
      "strrev", "strlen", "soundex", "ord");
    foreach ($stringFunctions as $key => $function) {
            if (!function_exists($function))
                  unset($stringFunctions[$key]);
            }
    $string = "the quick brown fox jumps over the lazy dog";
    for ($i = 0; $i < $count; $i++) {
        foreach ($stringFunctions as $function) {
            $r = call_user_func_array($function, array($string));
        }
    }
    return number_format(microtime(true) - $time_start, 3);
}
```

```
function test_Loops($count = 19000000)
{
    $time_start = microtime(true);
    for ($i = 0; $i < $count; ++$i);
    $i = 0;
    while ($i < $count) ++$i;
    return number_format(microtime(true) - $time_start, 3);
}


function test_IfElse($count = 9000000)
{
    $time_start = microtime(true);
    for ($i = 0; $i < $count; $i++) {
        if ($i == -1) {
        } elseif ($i == -2) {
        } else if ($i == -3) {
        }
    }
    return number_format(microtime(true) - $time_start, 3);
}


$total = 0;
$functions = get_defined_functions();
$line = str_pad("-", 38, "-");

echo "<pre>$line\n|" . str_pad("PHP BENCHMARK SCRIPT", 36, " ",
STR_PAD_BOTH) . "|\n$line\nStart : " . date("Y-m-d H:i:s") . "\nServer
: {$_SERVER['SERVER_NAME']}@{$_SERVER['SERVER_ADDR']}\nPHP version : "
. PHP_VERSION . "\nPlatform : " . PHP_OS . "\n$line\n";

foreach ($functions['user'] as $user) {
    if (preg_match('/^test_/', $user)) {
        $total += $result = $user();
        echo str_pad($user, 25) . " : " . $result . " sec.\n";
    }
}

echo str_pad("-", 38, "-") . "\n" . str_pad("Total time:", 25) . " : "
. $total . " sec.</pre>";
```

B CD

Obsah

1. Zdrojové kódy všech aplikace + databáze
2. Zdrojové soubory load testů
3. Všechny využité grafy z load testů
4. Samotná práce ve formátu PDF