TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

CYBER-SEARCH - REKONFIGUROVATELNÁ PLATFORMA SE SINGLE PAGE INTERFACE FRONTENDEM

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Ondřej Smola**
Vedoucí práce: Ing. Jiří Jeníček, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

CYBER-SEARCH - RECONFIGURABLE PLATFORM WITH SINGLE PAGE INTERFACE FRONTEND

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Ondřej Smola**
Supervisor: Ing. Jiří Jeníček, Ph.D.

Liberec 2014

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Ondřej Smola**
Osobní číslo:	**M12000224**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Cyber-Search - rekonfigurovatelná platforma se Single Page Interface frontendem**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s problematikou tvorby Single Page Interface aplikací a potřebnými technologiemi.
2. Navrhněte a realizujte serverovou a klientskou část aplikace společně s konfigurací virtuálního stroje.
3. Aplikace bude umožňovat jednoduše vytvářet nezávislé klony, automaticky se rekonfigurovat mezi produkčním, vývojovým a testovacím profilem, být přenositelná jak v rámci operačních systémů tak i různých poskytovatelů databázových systémů a bude umožňovat centralizovanou správu a monitoring vytvořených klonů a přidružených systémů.
4. Funkcionalita platformy bude přizpůsobena jako uživatelská brána pro interní systémy pro přepis a vyhledávání v řeči v hlasových nahrávkách vyvíjených na ústavu ITE.
5. Vytvořené řešení bude sjednoceno jako samostatná instance virtuálního stroje pro jeho další uplatnění.

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40 - 50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **O'REILLY MEDIA. JavaScript Patterns [online]. 2010. ISBN 978-0-596-80677-4.**

[2] **O'REILLY MEDIA / YAHOO PRESS. JavaScript: The Good Parts . 2008. ISBN 978-0-596-15873-6.**

[3] **JOHNSON, Rod. Expert One-on-One J2EE Design and Development. ISBN 0764543857.**

[4] **BAUER, Christian, Gavin KING a Gary GREGORY. Java Persistence with Hibernate. 2014. ISBN 1617290459.**

Vedoucí diplomové práce:	**Ing. Jiří Jeníček, Ph.D.** Ústav informačních technologií a elektroniky
Datum zadání diplomové práce:	**12. září 2013**
Termín odevzdání diplomové práce:	**16. května 2014**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 12. září 2013

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Předmluva

Tato práce byla realizována během dvou let autorova působení na Ústavu informačních technologií a elektroniky Technické univerzity v Liberci.

Tvorba této práce probíhala na základě grantů *Automatická transkripce a indexace přednášek* (TAČR - ALFA, TA01011142), *Zpřístupnění archivu Českého rozhlasu pro sofistikované vyhledávání (*Ministerstvo kultury, DF11P01OVV013) a *Živé Archivy* (TAČR – ALFA, TA01011204). Vývoj práce probíhal na základě předchozích výsledků kolektivu Laboratoře počítačového zpracování řeči a na základě spolupráce s firmou NEWTON Technologies a.s. .

Poděkování

Na tomto místě bych rád poděkoval především mé ženě a rodičům za podporu během celého mého studia a doby věnované tvorbě této práce. Dále bych chtěl poděkovat celému kolektivu Laboratoře počítačového zpracování řeči Ústavu ITE, jelikož jejich výsledky daly možnost vzniku této práce. V neposlední řadě bych chtěl poděkovat vedoucímu mé práce Ing. Jiřímu Jeníčkovi, Ph.D., za poskytnuté rady a čas věnovaný konzultacím.

Abstrakt

Práce se zabývá návrhem a realizací automatizovaného a soběstačného řešení pro systém umožňující offline rozpoznávání mluvené řeči v audiovizuálních materiálech, vyvíjeného v Ústavu informačních technologií a elektroniky Technické Univerzity v Liberci.

Celý systém se skládá z centrální jednotky spravující platformy, označované jako Cyber-Search. Všechny platformy Cyber-Search jsou realizovány na virtualizačním řešení KVM a v základní konfiguraci jsou tvořeny 5 virtuálními stroji. Platforma je rozdělena na databázový stroj, datový stroj se streaming serverem a síťovým úložištěm, stroj pro indexaci přepisů a úkolování pracovních stanic a stroj, na kterém běží hlavní Java EE aplikace s uživatelským rozhraním. Pracovní stanice jsou realizovány jako virtuální stroje, sloužící pro výpočetně náročné operace, zastoupené především převodem videa a procesem rozpoznání mluvené řeči.

Výsledkem práce je systém umožňující plně automatizovanou instalaci a konfiguraci platformy na vlastním hardwaru. Pro proces instalace je nutné pouze předání licencí a krátkého popisu konfigurace. Systém automaticky generuje a provádí veškeré konfigurace potřebné pro vytvoření platformy, spuštění aplikací, jejich monitoring a automatizovanou správu. Pro přístup a ovládání platformy je implementována Java EE aplikace. Ta je automaticky sestavena a umístěna na aplikační server s pomocí nástroje Maven a aplikace Jenkins. Prezentační vrstva řešení je realizována jako Single Page Interface webová aplikace, vytvořená na základě aplikačního rámce, vyvinutého během mého diplomového projektu.

Mezi hlavní výsledky této práce patří kromě samotné realizace výše popsané platformy především popis postupu automatizace komplexního řešení, vyžadujícího rozsáhlou integraci mnoha programů a služeb. Součástí práce je také popis realizace vybraných problematických částí, jako například automatizace konfigurace operačního systému Windows 7, klasifikace a autentizace klientů během automatizovaného instalačního procesu a diskuze škálovatelnosti platformy.

Klíčová slova: Automatizace řešení, KVM, Puppet, Single Page Interface, Spring, Java

Abstract

Main goal of this thesis is to describe a design and an implementation of an autonomous and self-contained solution for the spoken language transcription system developed at the Institute of information technology and electronics at Technical University of Liberec.

The whole system consists of a central unit managing platforms called Cyber-Search. Each Cyber-Search platform is implemented on KVM virtualized infrastructure and in it´s basic configuration contains 5 virtual machines. The platform is divided into detached virtual machines with: database systems; data storage with streaming server and network storage; full text transcript indexation and worker machines task manager; main Java EE application providing user interface. Workers are implemented as virtual machines and their main responsibility is to perform computationally intensive tasks represented mostly by video encoding and speech transcription.

Result of this thesis is the system that allows fully automated installation and configuration of the platform on user´s own hardware. Only licenses and short description of the target hardware is required for the installation process. Created system automatically generates and performs all steps required to install the Cyber-Search platform and all its services together with their automatic monitoring and maintenance. Access and management is controlled by the implemented Java EE application. The application is built and deployed by Maven and the system for continuous delivery – Jenkins. The presentation layer is implemented as a Single Page Interface web application implemented using framework developed during my master´s project.

Main benefit of this thesis is not only in the implementation of the Cyber-Search platform but in description of the process of automation of the complex solution that requires extensive integration of many programs and services. Part of this thesis is also the description of selected difficult tasks such as automation of Windows 7 operating system configuration, classification and authentication of clients during automated installation process and discussion of the system scalability.

Keywords: Solution automation, KVM, Puppet, Single Page Interface, Spring, Java

Obsah

Seznam obrázků

1 Úvod

Mezi aktuální výzkumné téma patří systémy pro automatické rozpoznávání mluvené řeči. Mezi jeden z hlavních cílů systému pro automatické rozpoznání řeči (dále jen rozpoznávač), je převod audio nahrávky do textového přepisu. Textový přepis je pro počítač a člověka nejen běžnou formou ukládání informace, ale především lze pro textový formát využít dostupných řešení pro indexaci a následné fulltextové vyhledávání.

Součástí rozpoznávače mohou být i další programy, umožňující například detekci pohlaví a nálady nebo rozpoznání mluvčích. V mnoha případech navíc nejsou dostupné pouze audio nebo video záznamy, ale i další podpůrné materiály, tvořené například slidy z přednášek. Z těchto důvodů kombinují některé rozpoznávače do celkového procesu zpracování i další programy, umožňující například zpracování slidů z přednášek nebo automatickou detekci známého mluvčího.

S rozšiřováním kvality rozpoznávače a přidáváním další rozšiřující funkcionality dochází k značnému nárůstu nároků na infrastrukturu, potřebnou pro zprovoznění a běh těchto řešení. Systémy pro rozpoznání řeči navíc musí poskytovat nejen co nejlepší úspěšnost rozpoznání, ale zároveň být co nejlépe integrovatelné do stávajících systémů uživatele. Potřeby integrace, rozšiřitelnosti a soběstačnosti celého systému pro rozpoznání řeči vytvořily základ pro vznik cílů této práce.

Hlavním úkolem této práce je tedy vytvořit soběstačný automatizovaný systém pro offline rozpoznávání v řečových nahrávkách. Systém musí obsahovat všechny části potřebné pro nahrávání dat, jejich zpracování a následnou prezentaci. Jelikož je s ohledem na množství technologií, potřebných pro běh systému, očekávána poměrně značná rozsáhlost řešení, je nutné realizovat systém co nejvíce automatizovaný a s možností centrální správy.

Z důvodu poměrně značně rozsáhlého teoretického úvodu, potřebného k pochopení této problematiky, bylo rozhodnuto do práce zahrnout pouze vybrané teoretické části. Tato práce předpokládá od čtenáře zkušenost s administrací linuxového serveru, pokročilou znalost vývoje uživatelského rozhraní v JavaScriptu (Douglas Crockford, 2008) a pokročilou zkušenost s vývojem a nasazením Java EE a Spring aplikací (Rod Johnson, 2003).

1.1 Dostupná řešení

V současnosti existuje celá řada řešení, zabývajících se problematikou rozpoznání mluvené řeči a prezentací záznamů z přednášek a konferencí. Dostupné jsou například portály superlectures.com, translectures.eu nebo systém Mediasite (sonicfoundry.com). První dvě jmenovaná řešení fungují na principu odeslání dat na server, kde jsou data zpracována, přepsána a později zobrazena uživateli. Poslední jmenované řešení je využíváno například pro zpracování přednášek na univerzitách. Mediasite ale neobsahuje systém pro rozpoznávání a tvorbu přepisů a slouží spíše jen pro prezentaci pořízených záznamů.

Žádné z dostupných řešení neumožňuje rozpoznání a vyhledávání v přepisech na hardwaru uživatele. Možnost zpracovávat data lokálně je ale naprosto kritická pro rozpoznávání citlivých dat, jakými jsou například interní firemní data, záznamy telefonních hovorů nebo placených konferencí a přednášek.

Uvedená řešení jsou dále dodávána jako hotový celek, který nelze, dle dostupných informací, jednoduše upravovat pro potřeby zákazníka. Problematická je i škálovatelnost takto centralizovaných řešení.

Na základě informací o dostupných řešeních bylo rozhodnuto práci zaměřit především na umožnění lokálního zpracování dat a vytvoření co nejvíce soběstačného systému.

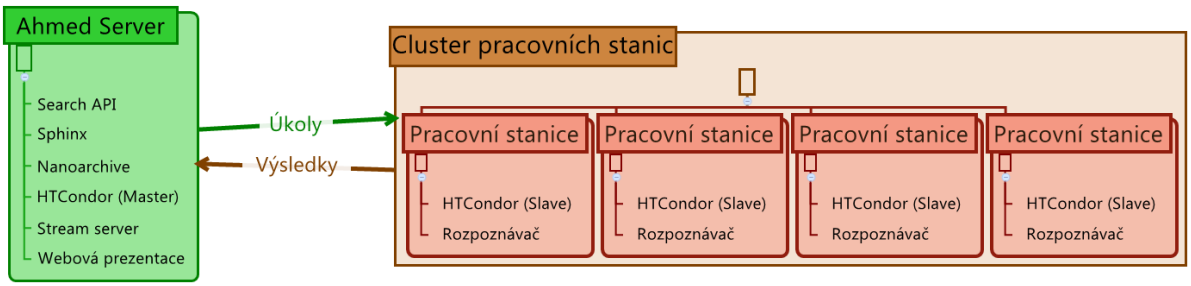
1.2 Cíl práce

Cílem práce je vytvořit plně automatizovaný a centralizovaně spravovaný systém, který využije stávající řešení, vyvíjená na Ústavu ITE a vytvoří z nich plně soběstačný a přenositelný celek, umožňující zpracování, rozpoznání a indexování nahrávek lidské řeči. Systém musí být schopen soběstačného běhu na infrastruktuře uživatele. Pro prezentaci dat bude vytvořena moderní Single Page Interface webová aplikace s administračním rozhraním. Systém bude umožňovat vzdálenou konfiguraci a dávkové zpracování většího množství dat. Systém musí být navržen tak, aby mohl běžet v libovolném množství instancí a umožňovat škálovat dle potřeb od využití pro jednotlivé uživatele až po služby s velkým počtem uživatelů. Dále musí být možné centralizovaně upravovat řešení tak, aby umožňovalo pružně reagovat na změny požadavků a v případě potřeby jednoduše přidávat další výpočetní kapacitu.

2 Analýza požadavků práce a stávajícího řešení Ahmed

Práce vychází ze stávajícího systému pojmenovaného Ahmed, a proto bylo nutné klást důraz na zachování původní funkcionality, a s tím souvisejícího začlenění do ostatních systémů a projektů. Výchozí návrh řešení je dostupný v diplomové práci autora Karla Blavky (2010). Součástí analýzy stávajícího řešení bylo tedy především vyhledat místa, která je možné využít bez nutnosti větších úprav na straně již dostupného řešení. Výsledek tedy musí být systém, obalující stávající řešení způsobem, umožnujícím plnohodnotnou funkcionalitu původního řešení.

2.1 Analýza funkcionality serveru Ahmed



Obr. 1: Hlavní komponenty původního řešení Ahmed

Hlavní částí stávajícího řešení je linuxový server označovaný jako Ahmed. Na serveru se nachází PHP aplikace pro prezentaci dat využívající interní rozhraní, označované jako Search API. Toto rozhraní umožňuje vyhledávání v přepisech řečových nahrávek za pomoci databázového serveru MySQL a aplikace Sphinx pro fulltextové vyhledávání. Součástí systému je dále hlavní aplikace Nanoarchive, napsaná v jazyce Python. Aplikace Nanoarchive slouží pro zpracování dat a úkolování pracovních stanic. Pro přehrávání záznamů je využit Red5 media streaming server. Běh aplikace a přístup k souborům je realizován pomocí HTTP serveru Apache. Poslední součástí je systém pro správu distribuovaných úloh HTCondor v režimu master. Clustery pracovních stanic jsou realizovány jako počítače v rámci učeben a dedikované výpočetní stroje. Na každé pracovní stanici je prováděn výpočet úloh, zadávaných systémem HTCondor v režimu slave. Mezi hlavní a nejnáročnější úlohu patří přepis řečové nahrávky prováděné rozpoznávačem. Rozpoznávač bylo možné zkompilovat pouze pro platformu Windows, a proto byl jeho běh podmíněn přítomností operačního systému Windows.

2.2 Diskuze výhod a nevýhod serveru Ahmed

Systém je již přes čtyři roky v provozu a na jeho vývoji autor stále pokračuje. Realizován je na základě rozšířených a běžně podporovaných technologiích a je logicky rozdělen na aplikaci pro prezentaci dat a jejich zpracování. Výhodou je také Search API, napsané pro původní webové rozhraní, které lze velice jednoduše využít jako přípojný bod pro vlastní vývoj.

Základní problém stávajícího řešení je manuální konfigurace, správa a monitoring prakticky všech částí systému. Stávající systém není navržen s ohledem na centralizovanou správu. Manuální konfigurace jednotlivých částí byla navíc bez jakékoliv dokumentace, a tím pádem byla plně závislá na znalostech a možnostech autora původního systému. Dalším problémem byla klonovatelnost samotného řešení. Systém neumožňuje jednoduše vytvářet další instance dle potřeb, a ani neumožňuje vytvořit oddělený přístup pro více uživatelů v rámci jedné instance. Stávající podoba klonování spočívá v ručním kopírování zdrojového kódu s následnou manuální rekonfigurací. Řešení dále chybí administrace. Dostupné grafické uživatelské rozhraní je stavěno především pro jednoduchou prezentaci výsledků a není v něm brán ohled na pokročilou interaktivitu a podporu mobilních zařízení.

Jako značný problém se od začátku ukázala nutnost operačního systému Windows pro běh rozpoznávače. Ačkoliv je systém Windows prakticky nejrozšířenějším operačním systémem mezi běžnými uživateli, z pohledu automatizovaného serverové stroje má oproti řešení s využitím linuxového operačního systému prakticky jen samé nevýhody. Mezi hlavní nevýhody patří vysoké nároky na hardware oproti serverové instalaci linuxového OS a nutnost aktivace každé nové instance. Většina nastavení je v rámci systému přizpůsobena především pro přístup z uživatelského rozhraní než z příkazové řádky. Například průběh aktivace licenčního klíče je zobrazen prostřednictvím grafických vyskakovacích oken, naprosto nepoužitelných pro automatizovanou instalaci.

2.2.1 Platforma Cyber-Search jako soustava virtuálních strojů

Na základě zjištěných požadavků a analýzy stávajícího řešení bylo rozhodnuto systém realizovat jako soustavu minimálně dvou strojů. Potřeba dvou strojů vychází z nutnosti alespoň jednoho stroje s operačním systémem Windows a jednoho s linuxovým operačním systémem. Možnost běhu celého systému na platformě Windows byla zamítnuta z důvodů

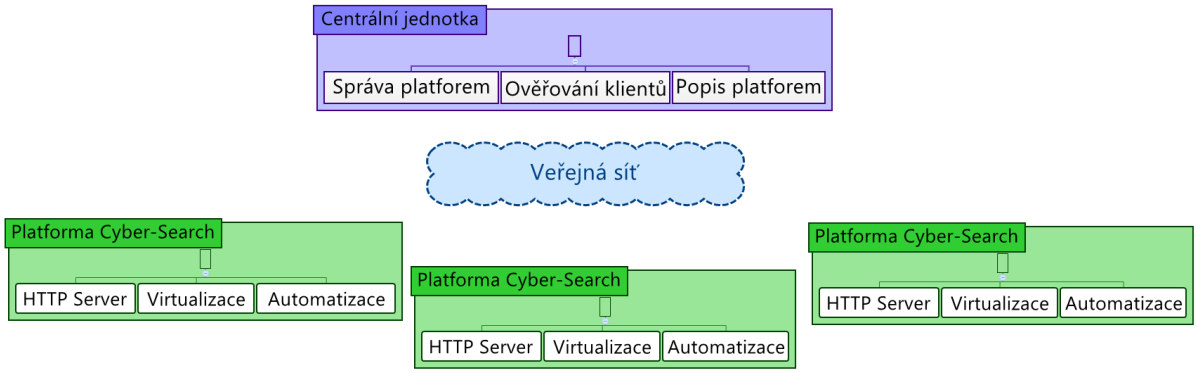
potřeby velkého množství licencí, větších nároků na potřebný hardware (bez klasického grafického uživatelského rozhraní je možné instalovat pouze serverové verze) a celkově horší podporu pro vzdálenou správu a automatizaci.

Realizace všech aplikací na jednom stroji byla také zamítnuta s ohledem na budoucí rozšiřitelnost. Sjednocením celé logiky uvnitř jednoho stroje by neumožňovalo běh více instancí a zároveň vyžadovalo úpravy stávajícího systému. Aby bylo možné vyhovět požadavkům na co nejmenší zásahy, bylo rozhodnuto aplikaci s administrací a uživatelským rozhraním realizovat jako třetí stroj, označovaný jako Cyber. Přidání dalšího virtuálního stroje sice přinese zvýšené nároky na realizaci, výsledkem však bude více flexibilní a rozšiřitelný systém. Rozdělením na více strojů navíc umožní mezi stroji pevně definovat a zdokumentovat způsob vzájemné komunikace. Logickým oddělením strojů je navíc získána zvýšená odolnost systému v případě selhání některého ze strojů. Z důvodu zajištění odolnosti systému vůči výpadku bylo rozhodnuto na stroj Cyber umístit systém pro správu a zálohování komunikace. Zálohování komunikace se strojem Ahmed bude zabráněno ztrátě důležitých informací v případě jeho nedostupnosti.

Otázka možnosti aktualizací a zálohování strojů dala vzniku nápadu na oddělení dat od aplikační logiky. Proto bylo rozhodnuto přidat další dva stroje. První stroj bude sloužit jako centrální databázové úložiště. Na tomto stroji poběží všechny potřebné databázové systémy. Stroj díky čistě databázovému zaměření bude vhodné umístit na výkonné datové úložiště a zároveň bude svojí velikostí umožňovat jednoduchou přenositelnost a zálohovatelnost. Druhý stroj bude sloužit pro uživatelská data, která není vhodná ukládat do databáze. Oproti stroji s databází bude koncipován jako stroj s potřebou velké úložné kapacity, vhodný pro umístění například na pomalejší mechanické pevné disky. Stroj bude sloužit především pro ukládání mediálních souborů pro streaming server, uživatelských souborů a souborů pro rozpoznávač. Umístění všech uživatelských dat na datový stroj dále umožní přemístění streaming serveru ze stroje Ahmed na datový stroj.

Celkově bude systém tvořen minimálně 5 virtuálními stroji. Platforma tedy bude tvořena upraveným původním strojem Ahmed a dále strojem s uživatelským rozhraním (Cyber), pracovní stanicí s rozpoznávačem (Pracovní stanice), datovým (Data) a databázovým strojem (DB).

3 Návrh centrální jednotky a platformy Cyber-Search



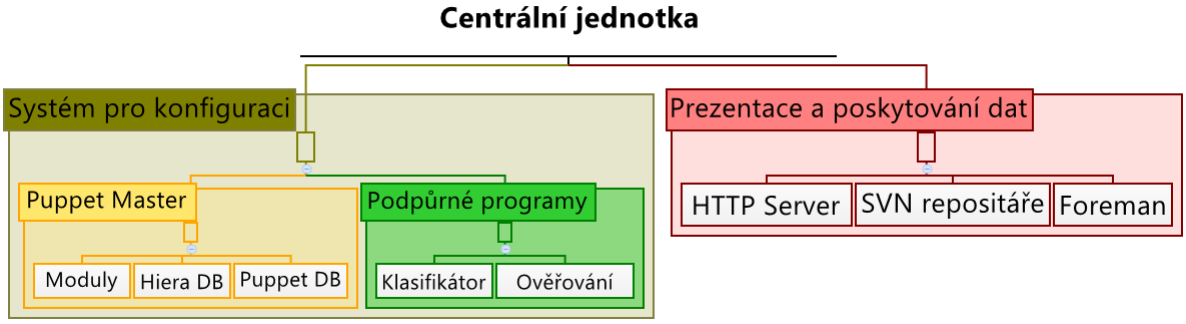
Obr. 2: Vztah centrální jednotky a platforem Cyber-Search

Návrh systému je rozdělen do částí zabývajících se návrhem centrální stanice, platformy Cyber-Search, vlastním aplikačním rámcem pro tvorbu uživatelského rozhraní a návrhem stroje s uživatelským rozhraním. Návrh pracovní stanice a stroje Ahmed byl dle analýzy již dostupný a bylo pouze rozhodnuto přesunout uživatelské rozhraní a správu dat do ostatních strojů. Proto se tato práce nebude zabývat jejich návrhem, ale pouze jejich automatickou konfigurací a správou v kapitole 6, která se zabývá se automatizací řešení.

3.1 Vztah centrální jednotky a platforem Cyber-Search

Na začátku návrhu bylo nutné definovat vztahy mezi jednotlivými částmi systému a základní omezující předpoklady. Prvním předpokladem je možnost funkcionality platformy i bez jediné dostupné pracovní stanice. Dále musí být umožněno provozovat velké množství platforem bez nárůstu komplexnosti správy a zároveň umožnit samostatnost platforem po dokončení instalace. Platforma musí být soběstačná i bez centrální jednotky a centrální jednotka nemusí mít neustálé spojení s platformami. V případě budoucího navázání spojení s centrální jednotkou ale musí být možnost platformu kdykoliv synchronizovat. Posledním omezením je nemožnost odebírat po instalaci důležité části systému bez asistence správce řešení. Platforma tedy bude moci například přidávat nové stroje, ale jejich odebrání bude vyžadovat akci správce. Celkově je při instalaci a správě platforem nutná opatrnost především při práci se zdroji obsahujícími uživatelská data. Důležité je, se vyvarovat kroků, které by například chybou správce řešení nebo špatně zapsanými údaji v centrální jednotce, mohly poškodit nebo smazat uživatelská data.

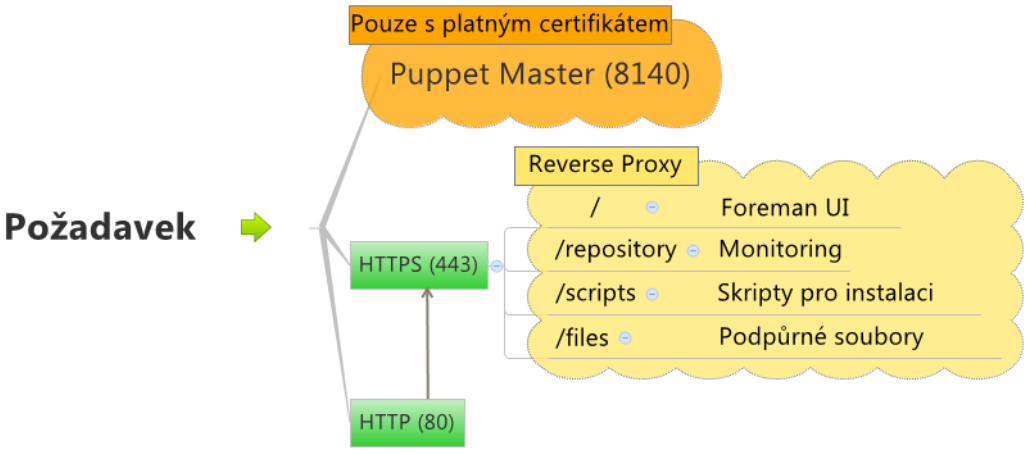
3.2 Návrh centrální jednotky



Obr. 3: Základní komponenty centrální jednotky

Centrální jednotka bude rozdělena na systém pro konfiguraci klientů pomocí nástroje Puppet a ostatní podpůrné aplikace (Obr. 3). Centrální jednotku bylo s ohledem na přenositelnost a vývoj rozhodnuto implementovat jako instanci virtuálního stroje. Během vývoje bude jednotka umístěna na výkonném dedikovaném stroji, zároveň sloužícím jako vývojová platforma. Umístění jednotky na nezálohované infrastruktuře sice nebude umožňovat nepřetržitou dostupnost, tu ale na základně hlavních bodů návrhu není třeba zaručit.

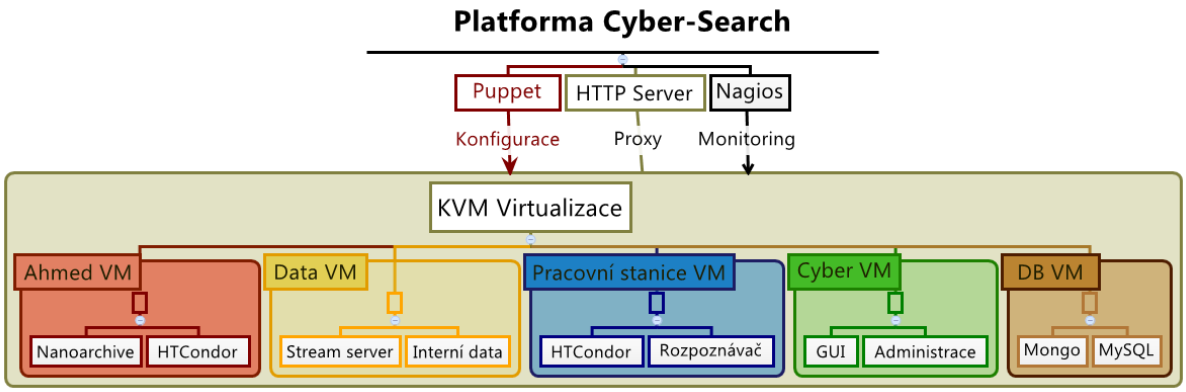
Systém pro konfiguraci se bude skládat ze systému pro automatizaci infrastruktury Puppet v režimu master (PuppetMaster). Pro uživatelské rozhraní bylo vybráno řešení Foreman jako náhrada za placené uživatelské rozhraní PuppetMaster. Dále zde bude nainstalován HTTP server Apache 2. Na centrální jednotce budou umístěny klony produkčních a testovacích repositářů všech vyvíjených aplikací (Cyber, Nanorachive) a zároveň zde budou umístěny další podpůrné programy a skripty pro automatizaci.



Obr. 4: Základní komponenty webového rozhraní centrální jednotky

Jelikož je třeba umožnit přístup k centrální jednotce z firemních sítí, které mohou mít omezení na povolené porty, je vhodné pro přístup využívat pouze porty HTTP a HTTPS. Proto bylo rozhodnuto povolit komunikaci pouze pomocí HTTPS s využitím reverse proxy a veškerou HTTP komunikaci směrovat na HTTPS port. Pro konfiguraci jednotlivých platforem je dále potřeba mít povolený port 8140, sloužící pro protokol Puppet (Obr. 4).

3.3 Návrh platformy Cyber-Search



Obr. 5: Schéma virtuálních strojů platformy Cyber-Search

Popis návrhu platformy Cyber-Search je rozdělen na výběr virtualizačního řešení, rozdělení funkcionality do virtuálních strojů, komunikaci mezi stroji a výběr systému pro monitoring a komunikaci s okolím.

3.3.1 Výběr virtualizačního řešení

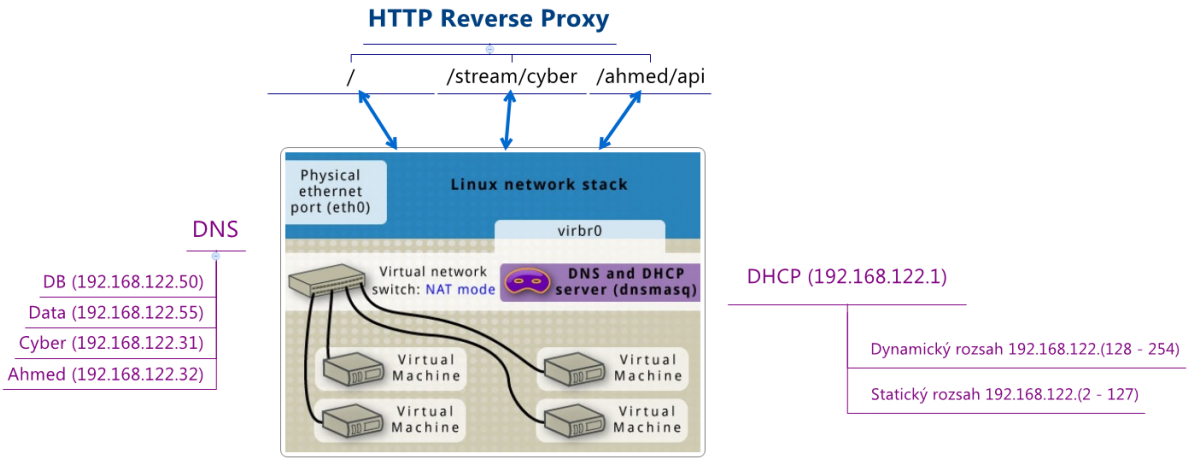
Pro virtualizaci platformy bylo rozhodnuto využít řešení KVM. Výběr virtualizačního řešení byl proveden na základě srovnání a informací o podpoře řešení v práci Martina Klepáče (2013). Dále na základě informací o používaných řešení u poskytovatelů virtualizačních infrastruktur a privátních cloudů (Gary Chen, 2013). Během návrhu bylo jako hlavní operační systém platformy rozhodnuto využít serverovou verzi operačního systému Ubuntu 12.04 LTS. Výběr operačního systému úzce souvisel s výběrem KVM jako virtualizačního řešení, z důvodu přímé integrace v jádře operačního systému. Pro práci s virtualizačním řešením bylo kromě základního nástroje pro správu *virsh* rozhodnuto využít nástroje *virt-top* a *virt-df* umožňujících přistupovat k informacím o využití procesoru a diskových kapacit virtuálních strojů. Během návrhu byly provedeny testy u výše uvedených nástrojů na schopnost provádět veškerou konfiguraci z příkazové řádky s ohledem na budoucí automatizaci.

3.3.2 Rozdělení funkcionality do virtuálních strojů

Návrh platformy vychází z rozdělení strojů během analýzy. Platforma je rozdělena na 5 strojů. Stroj Cyber bude zajišťovat uživatelské rozhraní a ovládat celou platformu. Součásti stroje bude aplikační server Tomcat, reverse proxy Nginx, systémy pro automatizované sestavení Maven, server pro integraci Jenkins a systém pro komunikaci pomocí zpráv RabbitMQ. Další částí je databázový stroj, na kterém poběží relační databáze MySQL a dokumentově orientovaná NoSQL databáze MongoDB. Na stroji Ahmed poběží upravená původní aplikace Nanoarchive naprogramovaná v jazyce Python, webové rozhraní v jazyce PHP a systém HTCondor v režimu master. Pracovní stanice bude pro svůj provoz potřebovat běhové prostředí pro .NET, Python a systém HTCondor v režimu slave. Dále je zde potřeba nainstalovat knihovny jazyka Python a to především knihovny NumPy, SciPy a PIL. Na datovém stroji bude umístěn streaming a reverse proxy server Nginx a systém pro síťové sdílení souborů NFS. Pro linuxové stroje bude použit operační systém Ubuntu 12.04 LTS a pro pracovní stanice operační systém Windows 7 Professional s českou lokalizací.

3.3.3 Návrh typů komunikace mezi virtuálními stroji

Pro komunikaci mezi virtuálními stroji bylo rozhodnuto využít privátní virtuální síť a pro sdílení většího množství dat využít sdílených síťových složek pomocí protokolu NFS.



Obr. 6: Konfigurace privátní sítě a nástroje dnsmasq

Během návrhu privátní sítě bylo rozhodnuto konfigurovat privátní síť virtualizačního řešení KVM. Součástí konfigurace sítě bude i konfigurace a správa DNS a DHCP služby *dnsmasq*. Konfigurace DHCP služby bylo rozhodnuto provést tak, aby došlo k rozdělení IP adres privátní sítě na dva logické bloky. Spodní blok bude sloužit pro staticky přidělované

adresy, využívané všemi stroji, kromě pracovních stanic. Vrchní blok je určen pro dynamické adresy, přidělované pracovním stanicím. Pro konfiguraci DNS serveru bylo rozhodnuto využít souboru hosts.txt na stroji provozujícím virtualizační řešení. Takto navržené řešení umožňuje pomocí reverse proxy serveru Nginx směrovat požadavky z veřejné sítě do jednotlivých strojů uvnitř privátní sítě podle jejich registrovaného jména.

Pro zabezpečení bude možné využít statického přidělování IP adres pro omezení přístupu pouze na jednotlivé stroje. Do takto navrženého řešení lze zároveň jednoduše zaintegrovat sdílené úložiště, které pro svoji konfiguraci bude využívat DNS jmen privátní sítě. Důležitým bodem, zjištěným z pokusů během návrhu, je nutnost nakonfigurovat privátní síť před instalací strojů, jelikož změna privátní sítě v potřebném rozsahu vyžaduje znovu připojení všech virtuálních síťových adaptérů.

3.3.4 Výběr nástroje pro monitoring řešení

Pro monitoring řešení bylo potřeba vybrat nástroj umožňující jednoduchou integraci vlastních skriptů, obsahující možnost automatické konfigurace a webové uživatelské rozhraní. Výše uvedené požadavky splňuje řešení Nagios (Obr. 7). Mezi hlavní důvody zvolení tohoto řešení byla možnost psát skripty v jazyce Python a Shell, popis konfigurace pomocí přehledného formátu a implementace v jazyce PHP. Implementace v jazyce PHP je výhodná kvůli možnosti využít automatizované nastavení infrastruktury pro PHP na stroji Ahmed.

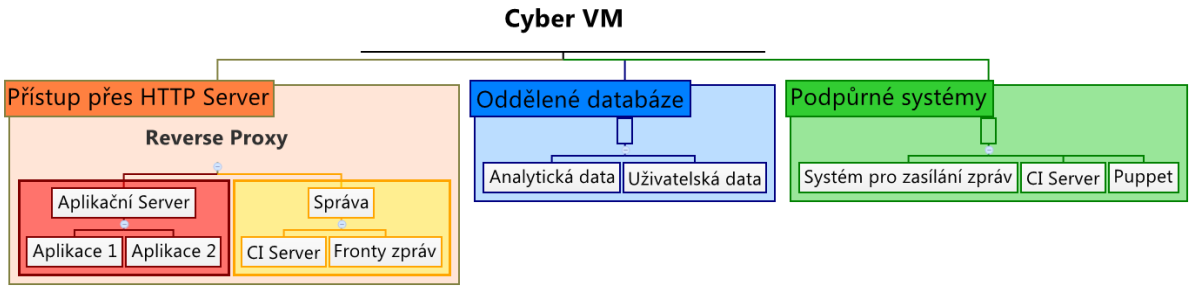
ahmed2	cpu	OK	04-06-2014 21:09:47	11d 6h 25m 58s	1/4	OK - usage 0.1%
	hdd	OK	04-06-2014 21:10:22	11d 6h 25m 10s	1/4	OK - total 37G, used 2.5G, free 33G (7%)
	mem	OK	04-06-2014 21:06:06	11d 6h 24m 22s	1/4	OK - MEMORY usage 12.0%
	nanoarchive	OK	04-06-2014 21:07:07	11d 6h 23m 35s	1/4	HTTP OK: Status line output matched "200,OK,302" - 838 bytes in 0.001 second response time
	nanoarchive queue	OK	04-06-2014 21:08:44	10d 10h 1m 47s	1/4	Nanoarchive task queue running (processing 0 tasks)
	nginx	OK	04-06-2014 21:10:39	10d 10h 3m 16s	1/4	OK - nginx is running. 1 requests per second, 1 connections per second (1.00 requests per connection)
	status	OK	04-06-2014 21:06:25	11d 6h 22m 32s	1/4	OK - VM ahmed2 is running
	time	OK	04-06-2014 21:09:48	11d 6h 25m 51s	1/4	NTP OK: Offset -0.0005620718002 secs
condor_1	cpu	OK	04-06-2014 21:10:34	12d 0h 11m 17s	1/4	OK - usage 1.0%
	hdd	OK	04-06-2014 21:06:12	12d 0h 12m 9s	1/4	OK - total 30G, used 14G, free 16G (47%)
	mem	OK	04-06-2014 21:07:18	10d 10h 4m 21s	1/4	OK - MEMORY usage 12.0%
	status	OK	04-06-2014 21:10:46	12d 0h 15m 3s	1/4	OK - VM condor_1 is running

Obr. 7: Uživatelské rozhraní pro monitoring Nagios

3.3.5 Přenositelnost platformy Cyber-Search

Platforma je díky zvoleným technologiím velice snadno přenositelná (kromě pracovních stanic). Všechny použité technologie patří mezi rozšířené a podporované na prakticky všech běžně používaných operačních systémech. Rozdělením na virtuální stroje navíc lze měnit interní systémy v rámci virtuálního stroje a pro kompatibilitu stačí zachovat rozhraní pro ostatní stroje. Takto lze například jednoduše vyměnit aplikační server pro Java EE aplikace, server pro spojitou integraci Jenkins, nebo reverse proxy Nginx za Apache.

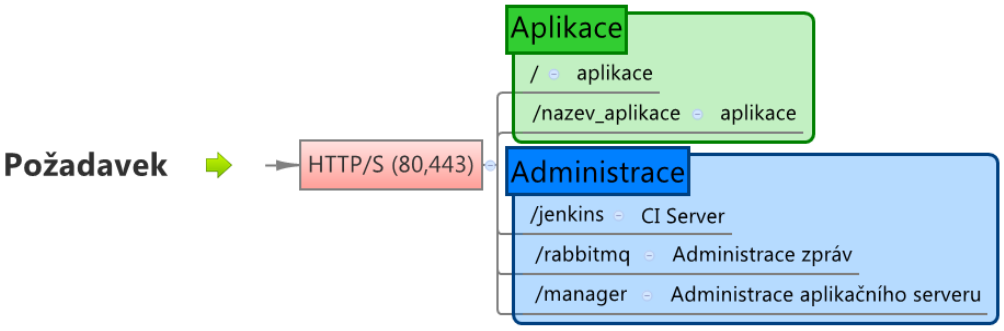
3.4 Návrh stroje s uživatelským rozhraním - Cyber



Obr. 8: Základní komponenty stroje Cyber

Základní komponentou virtuálního stroje Cyber je aplikační server Tomcat (Obr. 8). Aplikace budou na server umístěny lokálně s pomocí serveru pro spojitou integraci Jenkins. Hlavním úkolem nástroje Jenkins je vytvoření webového archivu ze sestavené aplikace a jeho následné umístění a spuštění přes administrační rozhraní aplikačního serveru (Obr. 9). Aplikace bude kromě zpracování požadavků uživatele spravovat celé řešení a komunikovat se strojem Ahmed. Jelikož nemusí být vždy virtuální stroj Ahmed dostupný, bude komunikace v rámci administrace zálohována přes systém pro zasílání správ RabbitMQ, který zaručí odesílání požadavků ve správném pořadí a v případě výpadku zprávy zálohuje a umožní jejich opětovné odeslání.

Pro ukládání dat byly zvoleny oddělené databáze pro uživatelská data a pro analytická data. Obě databáze jsou fyzicky umístěny na databázovém stroji. Uživatelská data jsou uchovávána v relační databázi MySQL v rámci transakcí. Pro analytická data je použita dokumentově orientovaná databáze MongoDB. Databáze MongoDB byla zvolena z důvodu podpory analytických MapReduce dotazů a volnějšího způsobu ukládání dat jako dokumentů ve formátu JSON. MongoDB je v systému určeno především pro časté zápisy dat, která nejsou pro běh aplikace kritická a jejich ztráta by neomezila funkčnost aplikace.



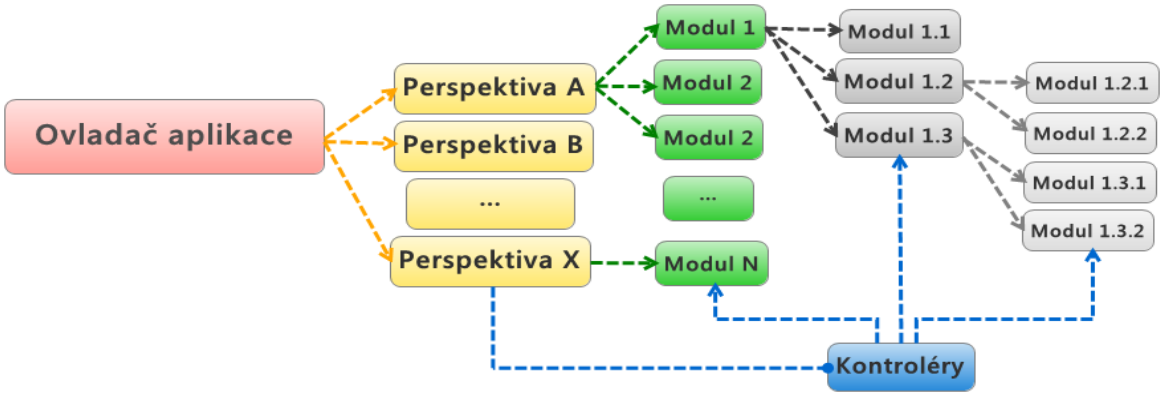
Obr. 9: Komponenty webového rozhraní stroje Cyber

3.5 Návrh aplikačního rámce pro uživatelské rozhraní

Jako aplikační rámec pro uživatelské rozhraní byl použit výsledek mého diplomového projektu (Příloha B). Jelikož podrobný popis jednotlivých částí rámce je součástí mé předchozí práce, zabývají se následující podkapitoly pouze krátkým přehledem použitého řešení a diskuzí hlavních výhod na nevýhod použití tohoto řešení.

3.5.1 Architektura aplikačního rámce

Architekturu rámce lze při pohledu shora rozdělit na ovladač aplikace, perspektivy, moduly a kontroléry (Obr. 10). Perspektiva tvoří pohled na celou stránku, jako v případě klasických stránek. Komunikace probíhá na základě zpráv, rozesílaných pomocí kontrolérů. Úkolem ovladače aplikace je správa přechodů mezi perspektivami.



Obr. 10: Celkový pohled na architekturu rámce z hlediska jednotlivých komponent

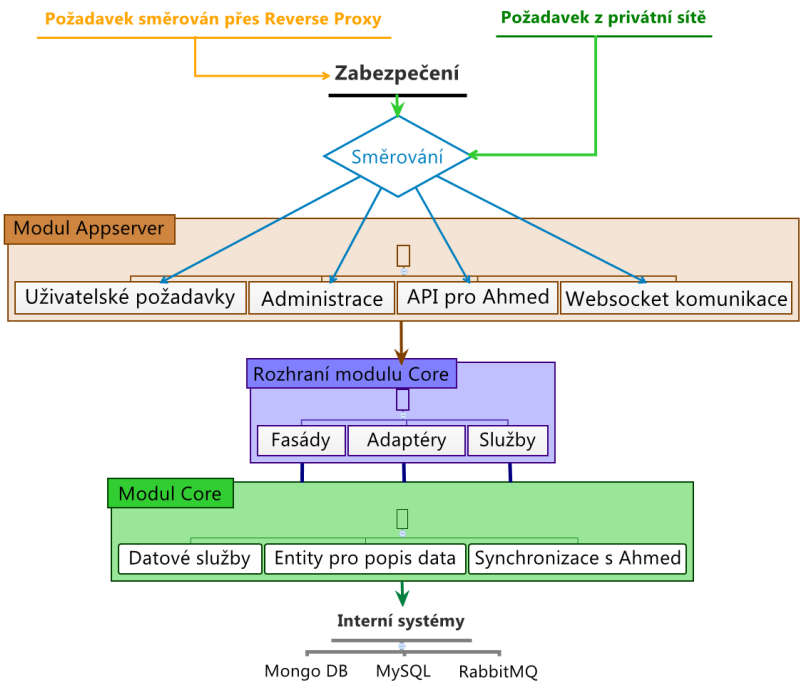
3.5.2 Využití vlastního aplikační rámce

Jelikož cílem práce má být vysoce interaktivní uživatelské rozhraní, bylo předem zřejmé, že nebude možné použít klasické řešení pomocí dynamických stránek. Pokud má být cílem práce umožnit uživateli interakci na úrovni klasické desktopové aplikace, bylo třeba zvolit úplně odlišný způsob návrhu a tvorby stránek, obecně známý pod pojmem Single Page Interface (dále jen SPI) aplikace. Principem SPI je tvorba webových stránek podobným způsobem, jako probíhá tvorba desktopových aplikací. Při zobrazení úvodní stránky je stažen ovladač aplikace a konfigurace modulů. Následně dojde ke spuštění SPI aplikace. Aplikace se poté dále stará o veškerou další komunikaci s uživatelem a serverem.

Veškerá komunikace se serverem je asynchronní a v případě nutnosti změnit uživatelský pohled, jsou vždy načítána pouze data, nezbytně nutná k aktualizaci pohledu. Oproti dostupným řešením pomocí technologií Flash, Silverlight a Java FX, je použité řešení realizováno v jazyce JavaScript, který má nativní podporu ve všech prohlížečích a pro jeho provoz není třeba žádných doplňků. Zároveň výkon interpretů/kompilátorů jazyka JavaScript se uvnitř prohlížečů neustále zlepšuje a tím dochází k nárůstu výkonu SPI aplikace. Přesun logiky na stranu klienta navíc umožňuje značné snížení zatížení na straně serveru, který ve většině případů slouží pouze pro zprostředkovávání dat přímo z databáze, bez nutnosti nad nimi dělat nějaké složitější úpravy a výpočty.

Pro úsporu přenášených dat a obsluhu množství požadavků obsahuje rámec vlastní systém vyrovnávacích pamětí, který dále přispívá k celkovému snížení zátěže na straně serveru. Nevýhodou využití vlastního rámce je nutnost jej udržovat zároveň s vývojem tohoto řešení a další problematikou, se kterou se potýká většina řešení na principu SPI. Mezi základní problémy patří problematika historie prohlížeče, přechodu stránky při stisku tlačítka zpět a lokalizace. Problém s historií prohlížeče je již v současné verzi aplikačního rámce vyřešen společně s řešením pro tlačítko zpět (Kapitola 7.5). Problém lokalizace zatím nebyl vyřešen.

3.6 Návrh hlavní aplikace pro virtuální stroj Cyber



Obr. 11: Návrh a rozdělení hlavní aplikace na stroji Cyber

Aplikace bude rozdělena na dva projekty Appserver a Core, reprezentované jako samostatné moduly (Obr. 11). Úkolem modulu Appserver bude zaručit ověření, směrování a následné zpracování požadavků. Komunikace s modulem Core bude probíhat přes vrstvu tvořící rozhraní mezi moduly. Modul Core bude odstíněn od přítomnosti webového rozhraní a v případě potřeby může být použit i samostatně. V modulu Core bude realizována většina logiky samotného systému. Budou zde přítomny služby pro práci s daty v relační a dokumentově orientované databázi, definice datových entit, řízení zpracování zpráv a obsluha synchronizace s virtuálním strojem Ahmed. Konfigurace jednotlivých modulů bude probíhat přes soubory generované nástrojem pro automatizované sestavení – Maven.

3.6.1 Návrh způsobu ukládání dat a jejich poskytování

Během analýzy bylo rozhodnuto rozlišovat mezi daty uloženými v rámci databáze a ostatními daty. Prvním důvodem byla zcela odlišná potřeba z hlediska zabezpečení těchto zdrojů. Databázový stroj musí být velice dobře zabezpečen na úrovni jednotlivých připojení a uživatelských práv. Datový stroj není naopak nutné zabezpečovat na úrovni uživatelských dotazů a stačí jednoduché rozdělení na data přístupná všem uživatelům, nepřístupná interní data a data vyžadující externí ověření strojem Cyber.

Dalším důvodem rozdělení jsou naprosto odlišné nároky z hlediska povahy ukládaných dat. Velikost dat uložených v databázovém stroji nebude při běžném použití přesahovat několik GB, naopak zde ale budou kladeny nároky na rychlé datové úložiště, dostatek operační paměti a časté zálohy. Oproti tomu datový stroj musí být navrhován i pro množství dat v řádu TB, data budou zpravidla zapsána jednou a poté jen čtena. S ohledem na množství ukládaných dat a běžné přenosové rychlosti pevných disků se může streaming server stát rychle úzkým hrdlem celého řešení. Škálovatelností řešení se zabývá kapitola 9.

Přenos dat mezi aplikačními virtuálními stroji (Ahmed, Cyber) a databázovým strojem bude probíhat běžným způsobem pomocí databázových spojení. Přenos dat do datového stroje bude nutné provést pomocí systému pro síťové sdílení souborů, jelikož je data nutné fyzicky ukládat na datový stroj ale mít k nim přístup i z ostatních strojů. Pro sdílení dat bylo rozhodnuto využít některé z řešení, umožňující vytvoření sdíleného úložiště nad protokolem IP. Zvoleno bylo řešení NFS 3, které má všechny potřebné vlastnosti, data jsou ukládána na straně serveru (datový stroj), umožňuje sdílení na základě zvolených IP adres z privátní sítě a poskytuje dostatečný výkon pro přenos větších objemů dat (Ju a kol., 2009).

4 Virtualizace infrastruktury platformy Cyber-Search

Jelikož se řešení skládá z řady virtuálních strojů, které pro svůj provoz potřebují virtualizační řešení, zabývá se tato kapitola popisem použitých typů virtualizace. Dále jsou zde popsány typy virtualizací, na kterých bylo řešení během vývoje otestováno a je i v současnosti nasazeno.

4.1 Virtualizace jako řešení více OS na společném hardware

Virtualizace platformy je řešení, umožňující souběžný a především vzájemně nezávislý provoz operačních systémů na sdíleném hardware. Aby bylo možné dosáhnout oddělení je třeba, aby řešení obsahovalo logiku pro řízení a přidělování zdrojů. Základní komponentou řízení je centrální jednotka zvaná hypervizor. Implementace a možnosti hypervizoru patří k základním ukazatelům celkového výkonu. Virtualizace byla zvolena jako řešení hlavního problému této práce a to umožnění provozu strojů s linuxovým operačním systémem a s operačním systémem Windows 7 na sdíleném hardware.

4.2 Použité metody virtualizace

Virtualizaci je možné rozlišit na základě úrovně vzdálenosti od hardware, na kterém je provozována. Plně softwarová virtualizace přináší výhody v přenositelnosti a menších nárocích na zprovoznění za cenu ztráty výkonu. Hardwarová virtualizace oproti tomu umožňuje využít plný výkon systému s minimální režií pro virtualizaci. Hlavním nevýhodou je potřeba specifického software a složitější zprovoznění a správa. Střední cestou jsou řešení, využívající upravené jádro operačního systému, zvané paravirtualizace. Během práce bylo využito zástupců všech tří kategorií, pro testování bylo využito softwarové virtualizační řešení VirtualBOX, pro samostatné produkční nasazení řešení využívající paravirtualizaci KVM a systém byl nasazen i na profesionálním virtualizačním řešení VMWare vSphere ESXI.

4.2.1 Softwarová virtualizace vývojového prostředí

Pro testování bylo využito virtualizační řešení VirtualBOX společně s programem Vagrant na operačním systému Windows 7. Hlavní výhodou zvoleného řešení je možnost rychlého nasazení nového stroje díky nástroji Vagrant. Program Vagrant umožňuje vytvářet nové instance virtuálních strojů uvnitř VirtualBOXu v řádu několika desítek sekund. Základem pro nový virtuální stroj jsou vzorové obrazy disku zvané Vagrant Boxy a inicializační skript

VagrantFile. Vytvořené vzorové obrazy disků obsahují nakonfigurované minimální instalace operačních systémů Debian 7 a Ubuntu 12.04. Mezi standardní kroky po instalaci je zařazen skript pro instalaci a konfiguraci aktuální verze nástroje Puppet a jeho automatické spuštění.

4.2.2 Paravirtualizované produkční prostředí

Pro produkční nasazení bylo využito paravirtulizační řešení KVM. KVM je instalováno na serverové instalaci operačního systému Ubuntu 12.04 provozovaného na hardware reprezentovaném výkonnými stolními počítači. V prostředí serverové instalace je oproti softwarové virtualizaci řízeno KVM především pomocí nástroje virsh využívající knihovnu *libvirt*. Aby bylo možné testovat pracovní stanici s operačním systémem Windows 7, byl na platformě pro vývoj vytvořen virtuální stroj s instalací grafického správce *virt-manager*, který pomocí VNC umožňuje vzdáleně spravovat plochu pracovních stanic.

O vytváření strojů se starají automaticky generované skripty společně se souborem odpovědí při instalaci. Skript obsahuje volání nástroje *virt-install* s požadovanými parametry a s cestou k souboru odpovědí. Soubor odpovědí se skládá z dvojic klíč a hodnota, které jsou poté automaticky předvyplněny během instalačního procesu. Během návrhu byl nalezen nástroj *ubuntu-vm-builder,* který umožňuje vytvářet nové stroje z předpřipraveného obrazu a je oproti klasické instalaci řádově rychlejší (prakticky jen dojde ke stažení obrazu). Přesto byl zvolen pomalejší způsob s kompletně čistou instalací a souborem odpovědí. Kompletní instalace totiž umožňuje libovolnou konfiguraci instalace a zaručuje plnou kontrolu nad výsledným stavem instalace.

4.2.3 Hardwarová virtualizace pro produkční prostředí

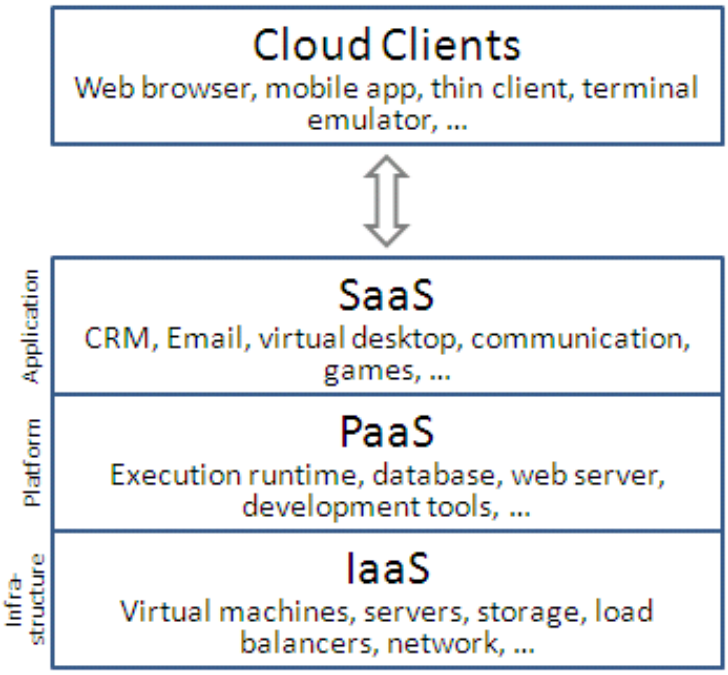
Platforma byla dále testována a zprovozněna na univerzitním virtualizačním řešení. Jedná se o řešení firmy Cisco skládající se z Blade serverů UCS B200 M3 s datovým úložištěm, realizovaným jako disková pole od firmy Dell. Na rozdíl od paravirtualizovaného řešení zde byl požadavek na dlouhodobý provoz pouze virtuálního stroje Cyber s přípojením na původní systém Ahmed. Realizace celé platformy na tomto virtualizačním řešení byla zamítnuta. Virtualizační řešení totiž dle stávajících podmínek není možné použít pro provoz náročných výpočetních úloh a ukládání větších objemů dat. Z těchto důvodů by zde nebylo možné umístit pracovní stanice ani datový virtuální stroj.

4.2.4 CyberSearch a cloudová řešení

Jelikož bylo řešení testováno na několika různých typech virtualizačních řešení, byla během práce zvažována myšlenka pokusit se zprovoznit systém na některém dostupném cloudovém řešení. Popis základních principů a typů cloudových řešení je popsán například v práci Jiřího Pětníka (2012). Pro potřeby této práce jsou zajímavé především služby IaaS a PaaS (Obr. 12).

IaaS, neboli infrastruktura jako služba, je možnost pronajmout si přímo hardware a s touto infrastrukturou lze přímo zacházet jako s vlastním virtualizačním řešení.

Poskytovatelé cloudových řešení obvykle nabízejí některé části systému jako služby PaaS (databáze, aplikačních server, fronty zpráv RabbitMQ), což velice dobře zapadá do celkového konceptu platformy jako sady oddělených virtuálních strojů, jelikož lze systém jednoduše pomocí automatizace překonfigurovat na použití služeb cloudového poskytovatele, který se dále stará například o pravidelné zálohování a celkovou dostupnost dat. Během práce bohužel nebylo nalezeno vyhovující řešení, které by umožňovalo odzkoušení této platformy bez nutnosti registrace se zadáním čísla platební karty a zároveň nebyl o cloudové řešení zatím projeven konkrétní zájem. Z těchto důvodů zůstala tato možnost realizace zatím pouze na teoretické úrovni.



Obr. 12: Rozdělení cloudových služeb na základě úrovní, které si již uživatel spravuje sám

5 Automatizace infrastruktury nástrojem Puppet

Automatizaci infrastruktury lze zjednodušeně popsat jako sadu kroků, které vybraný systém převedou do požadovaného stavu s ohledem na jeho omezení a odlišnosti. Mezi nejrozšířenější způsob automatizace patří sady skriptů psané ve skriptovacím jazyce, podporované na cílové platformě (Bash, Shell, Python, Ruby). Problém řešení pomocí sady skriptů spočívá ve svázanosti s cílovým operačním systémem a jeho prostředím. Pro automatizaci komplexních řešení se navíc stává vývoj pomocí skriptů neudržitelný a velmi obtížně testovatelný. Proto vznikla řada řešení, která poskytují ucelenou sadu nástrojů, které jednak umožňují udržitelný vývoj komplexních systémů a zároveň usilují o co největší přenositelnost a testovatelnost.

Mezi nejrozšířenější zástupce v této oblasti patří Puppet, Chef, Ansible a Salt. Popis rozdílů jednotlivých řešení je mimo rámec této práce a především při současném tempu vývoje jednotlivých zástupců by velmi rychle ztrácelo vypovídající hodnotu. Pro tuto práci bylo vybráno řešení Puppet od společnosti PuppetLabs. Puppet byl vybrán především díky tomu, že v době tvorby této práce měl nejpřehlednější dokumentaci a dostupné velké množství udržovaných modulů. Součástí systému Puppet je celá řada technologií rozdělených na stranu klienta a serveru, v rámci automatizace označovaného jako PuppetMaster, z nichž v této práci je krátce rozebrán samotný jazyk Puppet a systém modulů s tvorbou katalogů na straně centrální jednotky.

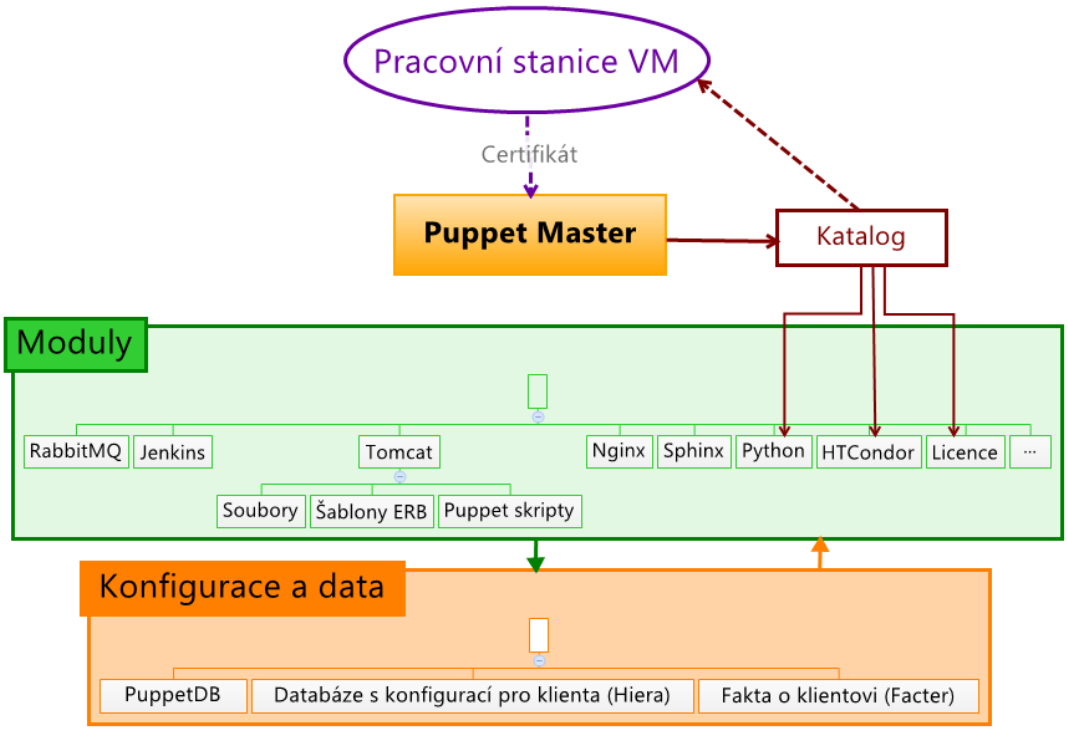
5.1 Deklarativní jazyk Puppet a Ruby DSL

Základem je vlastní deklarativní jazyk Puppet s možností využití doménově specifického jazyka Ruby. Hlavním principem je v obou případech zápis požadovaného stavu systému namísto způsobu, jakým se má tohoto stavu docílit. Díky tomu je možné dosáhnout snazší přenositelnosti a čitelnosti. Jazyk Puppet obsahuje celou řadu předdefinovaných datových typů pro práci se službami, instalacemi potřebných balíčků, vytváření uživatelů či konfiguraci systému pro monitoring řešení Nagios. Samotný běh plánu vykonávání jednotlivých deklarací je v základu čistě paralelní a pokud nejsou uvedeny žádné závislosti mezi deklaracemi, tak systém Puppet vytvoří plán vykonávání (katalog), ve kterém se mohou deklarace nacházet mimo pořadí.

5.2 Systém modulů a tvorba katalogů

Základem jsou soubory obsahující definice tříd a metod skládajících se z jednotlivých deklarací v jazyce Puppet. Jelikož při automatizaci složitějších infrastruktur je těchto částí velký počet, jsou dále seskupeny do modulů. Modul tvoří základní funkční celek při automatizaci větších systémů, obsahující skripty jazyka Puppet, šablony a další soubory nezbytné pro automatizaci (Obr. 13). Celkový popis infrastruktury systému je tedy složen ze seznamu modulů. Puppet obsahuje vlastní systém pro sdílení modulů PuppetForge. Bohužel celá řada dostupných modulů neobsahovala dostatečnou možnost úprav, a proto je nebylo možné použít přímo. Využity byly spíše jako součásti v rámci tvorby vlastních modulů.

Pro konfiguraci klienta je na základě vybraných modulů sestaven katalog. Katalog si lze zjednodušeně představit jako strom obsahující kroky, které je potřeba vykonat pro dosažení cílového stavu na klientovi. Realizaci pomocí stromu umožňuje paralelní vykonávání v rámci jednotlivých větví. Rozdělení do nezávislých větví v mnoha případech umožňuje značné zrychlení pro operace využívající různé zdroje, na druhou stranu představuje zvýšené nároky na návrh logiky. Nepřesnost v deklaraci závislostí často vede k složitě detekovatelným chybám, souvisejícími s náhodnou změnou pořadí v rámci vykonávání jednotlivých kroků.



Obr. 13: Princip sestavení katalogu na základě konfigurace a použitých modulů

5.3 Obnova z chyb během automatizované instalace

Systém musí být realizován tak, aby se dokázal automaticky obnovit z co největšího množství chyb. Mezi základní 4 typy chyb patří chyba při zadávání konfigurace, chyba v plánu pro instalaci platformy, výpadek internetového připojení během konfigurace a chyby vzniklé ze závislostí mezi stroji.

5.3.1 Chyba při zadávání konfigurace do centrální jednotky

Běžným příkladem této chyby je například překlep v cestě, špatné uvedení jména, hesla nebo licenčního klíče. Většinu těchto chyb lze opravit automaticky z centrální jednotky, případně vyžadují jen zásah z administračního rozhraní na platformě. Mezi chyby, které nelze opravit bez zásahu uvnitř platformy, patří uvedení špatných informací při instalaci virtuálních strojů, například hodnotami přesahujícími hardwarové možnosti nebo naopak neumožňující plynulý běh stroje. Během vývoje například došlo k chybnému nastavení operační paměti jednoho z virtuálních strojů na hodnotu 204 namísto 2048 MB, což je stále hodnota, při které se serverová edice úspěšně spustí. Přínos v této chvíli ukázal systém pro monitoring, který ihned po spuštění stroje odeslal email s kritickým stavem volné paměti. Na základě emailu poté bylo možné chybu jednoduše identifikovat a opravit. Aktuálně není implementován systém pro automatizovanou dynamickou změnu parametrů virtuálního stroje, a proto je tento zásah potřeba provést ruční změnou parametrů.

5.3.2 Chyba v plánu pro instalaci Puppet klienta

Chyba v plánu pro instalaci je v naprosté většině případů způsobena chybou programátora. Jelikož je plán značně rozsáhlý a navržen tak, aby systém Puppet měl co nejvíce možnosti pro optimální a efektivní zpracování co nejvíce kroků zároveň, může dojít k chybám ve skrytých závislostech. Jedním z mnoha odhalených případů bylo například nedůsledné popsání závislostí mezi vytvořením linuxové skupiny uživatelů nagios, instalací systému Nagios a vytvoření uživatele pro přihlašování do uživatelského rozhraní, který musí být součástí skupiny nagios. Ve výše uvedeném příkladu vzniklo místo, kde bylo možné prohodit instalaci systému Nagios s vytvořením skupiny nagios. Výsledkem bylo selhání instalace na základě chybějící uživatelské skupiny.

5.3.3 Výpadek internetového přípojení během konfigurace

Automatizační proces je realizován s ohledem na možnost výpadku internetového připojení. Kritická jsou s ohledem na výpadek připojení především dvě místa. Prvním je při stahování souborů z internetu pomoci externího programu *wget*. Pokud dojde k výpadku během stahování, soubor zůstane na disku a je potřeba další logiky k jeho odstranění. Druhé místo je během stahování balíčků aplikací instalačním procesem operačního systému na virtuálním stroji. Při výpadku spojení se instalace pozastaví a nabídne pokus o obnovu. Pokus o obnovu ale není možné automatizovaně provést a navíc v případě instalace doplňujících balíčků není možnost obnovy dostupná. Instalaci je po této chybě nutné ručně buď obnovit nebo virtuální stroj smazat a jeho instalace bude při další kontrole znovu spuštěna od začátku.

5.3.4 Závislosti mezi virtuálními stroji

Během automatizované instalace strojů není možné definovat pořadí, v jakém jsou virtuální stroje zprovozněny. Výsledkem instalace je tedy náhodná variace pořadí zprovoznění a spuštění strojů. S ohledem na pořadí může dojít k chybám během konfigurace strojů v závislosti na aktuální nedostupnosti funkcionality, poskytované jiným virtuálním strojem (databáze, sdílené úložiště). V případě nedostupnosti vyžadované funkcionality jsou závislé operace přerušeny a automaticky opakovány přibližně po půl hodině do doby, kdy je již potřebná funkcionalita k dispozici. Jelikož jsou závislosti mezi stroji pouze směrem k databázovému a datovému stroji, není zde možný vznik uváznutí. V praxi nedochází k více než dvěma odkladům a obvykle není potřeba odklad žádný.

5.4 Automatizace sestavení aplikace

Automatizací infrastruktury získáme možnost centralizovaně upravovat a vytvářet prostředí potřebné k provozu aplikací. Dalším krokem je vytvořit aplikaci tak, aby bylo možné změnou konfiguračního souboru provozovat různé klony aplikace, které navíc mohou obsahovat určité úpravy logiky pro jednotlivé uživatele. Následující podkapitoly se proto zabývají rozdělením aplikace do profilů, krátkým popisem použitých technologií a nakonec popisem způsobu sestavení.

5.4.1 Rozdělení aplikace do profilů rámce Spring

Jedním z hlavních problémů během vývoje složitějších aplikací je vytvoření způsobu, jakým aplikaci zprovoznit v testovacím, vývojovém a produkčním profilu. Každý profil má zpravidla velice rozdílné požadavky. Testovací profil musí být naprosto nezávislý na přítomnosti externích programů (databáze, fronta zpráv, webový server, virtuální stroj Ahmed). Otestování musí být dostatečně rychlé, aby jej bylo možné spouštět pravidelně. Otestování by zároveň mělo předcházet sestavení a spuštění aplikace na produkčním serveru. Vývojové prostředí na rozdíl od testovacího potřebuje pro svůj chod prostředí co nejvíce podobné prostředí, na kterém bude aplikace nasazena v produkčním prostředí. Od produkčního prostředí se liší především vypnutím nástrojů pro optimalizaci zdrojových kódů, systémů mezipaměti a znovupoužití databázových připojení. Pokud by byla jednotlivá prostředí provozována odděleně na úrovni různých vývojových větví, vznikl by značný problém s údržbou tohoto řešení. Proto bylo rozhodnuto vyvíjet aplikaci v produkčním prostředí a s pomocí podpůrných technologií umožnit transformaci aplikace do vývojového a dále do testovacího prostředí. Pro transformaci bylo využito možností aplikačního rámce Spring společně s nástrojem Maven.

5.4.2 Správa životního cyklu projektu s nástrojem Maven

Maven je nástroj pro správu životního cyklu projektu. Základním prvkem jsou soubory *pom.xml,* popisující jednotlivé moduly. V souboru jsou uvedeny všechny potřebné knihovny pro úspěšné sestavení včetně jejich repositářů, popis jednotlivých kroků sestavení, použitých zásuvných modulů, metadata o projektu a proměnných použitých při sestavení. Zároveň umožňuje transformovat projekt pro podoby, jakou očekává vývojové prostředí (např. Eclipse nebo Netbeans). Jelikož je hlavní aplikace rozdělena na dva projekty, jsou součástí řešení tři soubory *pom.xml* tvořící jednoduchý strom. Kořenem je hlavní soubor, obsahující všechny sdílené knihovny a jejich verze a definice profilů pro sestavení. Soubor dále deklaruje moduly Core a Appserver. Moduly poté deklarují pouze specifické knihovny a rozdíly oproti nadřazené konfiguraci.

5.4.3 Systém pro spojitou integraci Jenkins

Jenkins je aplikace, umožňující centralizovaně začleňovat software do prostředí. Jenkins sdružuje jednotlivé kroky do úkolů definovaných pomocí sady XML souborů. Jelikož je jejich struktura přehledná, dají se úkoly generovat automaticky pomocí nástroje pro automatizaci

Puppet. Jenkins je realizován jako velice flexibilní nástroj, jehož hlavní úkol lze zjednodušeně popsat jako vytvoření infrastruktury pro spouštění úloh v určitém pořadí se zaznamenáním výstupu. Spouštění úloh je možné realizovat z uživatelského rozhraní i periodicky. I když je možné umístit Jenkins přímo na použitý aplikační server Tomcat, bylo rozhodnuto využít soběstačné instalace z důvodu zaručení oddělení aplikačního serveru Tomcat od serveru pro sestavování. Pro potřeby platformy Cyber-Search je Jenkins doplněn o zásuvné moduly pro práci s SVN repositářem a pro možnost vzdáleného umístění výstupu kompilace na aplikační server.

S	W	Name ↓	Poslední úspěšný build	Poslední neúspěšný build	Délka posledního sestavení	
		archiv	4 hr 45 min - #8	žádný	51 sec	
		konference	4 hr 45 min - #15	žádný	59 sec	

Obr. 14: Výřez administrační rozhraní systému Jenkins s úkoly

5.4.4 Realizace způsobu sestavení a umístění aplikace

Proces automatizovaného sestavení je nejprve spuštěn z uživatelského rozhraní nebo požadavkem na API serveru Jenkins. Po aktivaci je úkol zařazen do fronty na zpracování, kde čeká na uvolnění pracovního slotu. Nejprve je stažena aktuální verze zdrojového kódu z SVN repositáře centrální jednotky. Z důvodu rychlosti a úspory přenášených dat je v případě dostupné některé předchozí verze repositáře pouze provedena aktualizace na nejnovější verzi. Po aktualizace zdrojových kódů je spuštěn nástroj Maven a jsou mu předány parametry generované nástrojem Puppet.

Nástroj Maven nejprve zkontroluje správnost konfigurace a následně stáhne všechny deklarované verze knihoven do lokálního repositáře. Po stažení všech knihoven dojde k vytvoření plánu pro kompilaci. Nejprve je potřeba zkompilovat modul Core, jelikož je deklarován v závislostech modulu Appserver. Po kompilaci a archivaci modulu Core je výsledný jar archiv umístěn do lokálního repositáře a tím jsou vyřešeny všechny závislosti modulu Appserver. Oproti modulu Core je ale výsledkem kompilace modulu Appserver webový archiv war. War je soběstačný archiv, obsahující všechny části, potřebné pro umístění do webového kontejneru aplikačního serveru Tomcat.

Pro umístění je využito zásuvného modulu v systému Jenkins, který přes předkonfigurovaného uživatele nástrojem Puppet provede umístění aplikace přes administrační rozhraní.

6 Implementace centrální jednotky

Implementace centrální jednotky je rozdělena do podkapitol, zabývajících se autentizací a zabezpečením klientů, přidělováním a tvorbou konfigurace a postupem pro vytvoření nové platformy.

6.1 Autentizace a zabezpečení Puppet klientů

Pro získání konfigurace musí Puppet klient získat certifikát podepsaný certifikační autoritou. Součástí instalace systému PuppetMaster je interní certifikační autorita Puppet CA. Certifikační autorita umožňuje 3 způsoby schvalování certifikátů. První způsob je realizován ručním schválením certifikátu z příkazové řádky. Druhým způsobem je využití souboru *autosign.conf,* který umožňuje automaticky schvalovat certifikáty z uvedených domén. Problém spočívá v ověření pouze na základě doménového jména, které není možné považovat za důvěryhodný způsob ověření. Proto bylo rozhodnuto využít třetího způsobu, a to ověření pomocí externího skriptu napsaného v jazyce Python *autosign.py*. Skript sice také ověřuje doménové jméno, ale využívá k tomu informací z databáze Hiera. Skript pracuje na principu kontroly konfiguračního souboru, vytvořeného při popisu nové platformy. Z informací v konfiguračním souboru lze odvodit povolená doménová jména pro nové stroje. I když i tento způsob ověření lze teoreticky obejít, útok by vyžadoval zjistit čas a plně kvalifikované doménové jméno jedním pokusem. V případě neúspěšného pokusu je vygenerována čekající žádost o certifikát a uloženy informace o požadavku, které lze poté použít k detekci útoku. Po získání certifikátu klient zažádá o svoji konfiguraci a veškerá komunikace dále probíhá přes protokol HTTPS na základě vzájemné autentizace certifikáty.

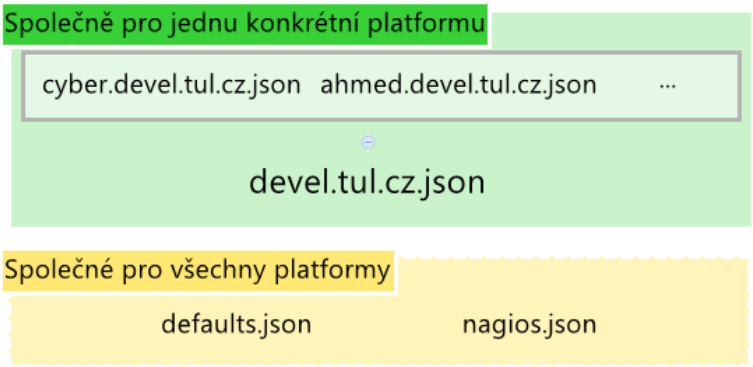
6.2 Způsob přidělení konfigurace pro Puppet klienty

V základním nastavení obsahuje konfigurace pouze seznam základních knihoven jazyka Puppet, které si klient musí stáhnout z centrální jednotky. Aby klient získal vlastní konfiguraci, musí být klasifikován externím klasifikátorem nebo mu musí být přidělena z rozhraní Foreman. Pro klasifikaci klientů byl rozšířen původní skript *node.rb* napsaný v jazyce Ruby, který je součástí instalace systému Foreman. Rozšíření skriptu umožňuje přidělovat konfiguraci na základě jména přiděleného certifikátu. Skript respektuje konfiguraci z uživatelského rozhraní Foreman a doplňuje konfiguraci pouze pokud není aktuálně uvedena.

Po přidělení konfigurace ji začne klient vykonávat, informace o postupu zaznamenává a nakonec odešle na centrální jednotku. Zde se vyskytl poměrně závažný problém v případě automatizace stroje s Windows 7, jelikož byla na stroji česká lokalizace využívající znakovou sadu Windows 1250. Klasifikátor *node.rb* ale interně očekává znakovou sadu Unicode s kódování UTF-8. Na základě této nekompatibility docházelo k chybám při přidělování konfigurace. Problém je způsoben systémem Facter, který poskytuje základní informace o klinetech. Facter při pokusu o uložení informací o klientovi ukončil volání chybou při práci se znakovým řetězcem během zápisu do databáze, a tím došlo k chybě celého procesu přidělování konfigurace. Chybu se podařilo obejít filtrací všech faktů, pro které volání končí chybou. Jediným omezením je nemožnost využívat filtrovaná fakta při vytváření konfigurace.

6.3 Konfigurace Puppet klientů

Puppet klientům jsou přidělovány konfigurační skupiny. Konfigurační skupiny obsahují seznam použitých modulů a konfiguraci. Konfigurace platforem probíhá na základě databáze Hiera. Hiera je nastavena tak, že konfiguraci vyhledává nejprve v konfiguračním souboru pojmenovaném podle jména certifikátu klienta, poté v souboru pojmenovaném podle domény a nakonec v souboru společném pro všechny platformy (Obr. 14). Takto realizovaná konfigurace umožňuje jednoduše vytvořit jeden společný soubor pro celou platformu a zároveň mít v případě potřeby možnost specifikovat konfiguraci pro konkrétní virtuální stroj uvnitř platformy.



Obr. 15: Způsob sdílení nastavení pro platformy

V době psaní této práce bylo v základních souborech více jak 550 konfiguračních možností. Konfigurační soubor pro platformu obsahuje pouze informace, které je třeba změnit oproti základnímu nastavení a povinné informace. Mezi povinné informace patří například

jméno aplikace, přihlašovací údaje do administračních rozhraní a licenční údaje. Po dokončení konfigurace klienta pomocí nástroje Puppet je již systém plně v provozu a je možné jej začít používat. Celková doba konfigurace je ovlivněna především rychlostí internetového připojení a disků. Podrobnější postup instalace je popsán v kapitole 8.

6.4 Vytvořené moduly pro automatizaci infrastruktury

Jelikož se platforma skládá z desítek programů se závislostmi v řádu stovek knihoven, bylo nutné vytvořit pro každou část platformy modul automatizující její instalaci a nastavení. V průběhu automatizace bylo vytvořeno přes 35 modulů a využito přibližně 15 modulů dostupných z centrálního repositáře PuppetForge. Moduly se značně liší svojí rozsáhlostí, některé jsou tvořeny pouze několika třídami jazyka Puppet a některé obsahují desítky tříd. Z důvodu velkého množství vytvořených modulů bylo rozhodnuto v této práci popsat pouze moduly Nginx, KVM, Cybersearch_Ahmed, Cybersearch a Nagios.

6.4.1 Modul Nginx

Modul Nginx je celkově nejvyužívanějším modulem a podílí se kromě stroje s databází na konfiguraci všech virtuálních strojů a stroje s virtualizačním řešením. Modul umožňuje instalaci serveru Nginx pomocí správce balíčků apt. Druhou možností instalace je automatizovaná kompilace a instalace ze zdrojového kódu s možností uvedení Nginx modulů, které mají být přikompilovány. Instalace ze zdrojových kódů je využita na strojích Data a Ahmed. Před instalací pomocí správce balíčků je součástí nastavení repositáře tak, aby byla vždy instalována aktuální verze.

Po instalaci je provedena konfigurace cílového virtuálního stroje. Nejprve jsou odstraněny všechny nepotřebné soubory základní instalace, vygenerován inicializační skript a vytvořen základní soubor s konfigurací *nginx.conf*. Základní soubor s konfigurací je generován ze 2 šablon. První šablona slouží pro stroje s virtualizací, obsahuje kromě směrování také definici systému mezipamětí a direktivy pro zabezpečení na základě souboru s přihlašovacími údaji a volitelně i povolenými IP adresami. Druhá šablona slouží pro virtuální stroje a obsahuje direktivy pro směrování požadavků. Obě šablony deklarují dále základní parametry ovlivňující výkon serveru, cesty k logům a chybové stránky. Po vygenerování souboru ze šablony dojde k jeho umístění do předdefinované složky k automatickému načtení.

6.4.2 Modul KVM

Hlavním úkolem modulu je zprovoznění virtualizačního řešení na linuxovém operačním systému, zastoupeného Ubuntu 12.04 LTS. Nejprve modul nainstaluje všechny potřebné závislosti a provede základní instalaci virtualizačního řešení KVM. Poté je odstraněna privátní síť vytvořená během instalace a nakonfigurována nová privátní síť včetně DHCP a DNS serveru. Po konfiguraci dojde k postupné instalaci virtuálních strojů. Instalace linuxového stroje začíná připravením základní složky pro umístění. Poté je vygenerován a spuštěn zaváděcí skript s konfigurací virtuálního stroje a souborem pro automatickou instalaci *preseed*. Před instalací je nejprve stažen minimální obraz pro serverovou instalaci Ubuntu a poté dojde k samotné instalaci operačního systému. Po úspěšné instalaci je virtuální stroj restartován a výsledek instalace je odeslán na centrální jednotku. Poté instalace pokračuje dalším strojem. Instalace virtuálního stroje se systémem Windows je realizována odlišným způsobem. Namísto instalace je stažen předpřipravený obraz disku, který je rozbalen a pomocí knihovny libvirt naimportován a nakonfigurován.

6.4.3 Automatizace virtuálního stroje Windows 7

Automatizace instalace linuxového operačního systému se skládá ze souboru odpovědí. Bohužel systém Windows 7 neumožňuje plně automatizovanou instalaci pomocí souborů odpovědí a ani nebylo nalezeno žádné plnohodnotné alternativní řešení. Proto byla automatizace vyřešena pomocí vlastního obrazu disku s upraveným operačním systémem Windows 7. Upravený obraz obsahuje ovladače *virtio* (Gal Motika, 2012) pro síťové a diskové operace pro efektivní běh na cílovém virtualizačním řešení. Součástí je dále instalace systému Puppet a umístění inicializačního skriptu. Takto připravený systém je zabalen a přenášen jako celek.

Při prvním spuštění na cílové platformě dojde ke spuštění inicializačního skriptu. Úkolem skriptu je změna jména počítače tak, aby mu byl vygenerován správný certifikát na centrální jednotce, a na nastavení spuštění služby Puppet po restartu systému. Po spuštění systému Puppet dojde k instalaci všech potřebných služeb a spuštění aplikace *WindowsActivation* napsané v jazyce C#, která se stará o automatizovanou obsluhu licence operačního systému.

6.4.4 Modul Cybersearch_ahmed

Modul slouží pro nastavení virtuálního stroje Ahmed. Jeho úkolem je automatizovat a upravit původní řešení popisované v analýze. Nejprve dojde k instalaci všech potřebných knihoven a nastavení aktuálních repositářů zdrojových kódů na centrální jednotce. Modul zajistí instalaci jazyka Python včetně všech vyžadovaných knihoven, systému pro fulltextové vyhledávání SphinxSearch a konfiguraci systému HTCondor v režimu master. Poté je řízení předáno modulu Nanoarchive, který se postará o instalaci hlavní aplikace pro obsluhu požadavků na vyhledávání a zpracování dokumentů. Modul Nanoarchive vytvoří adresářovou strukturu, do které je z repositáře umístěna aktuální verze zdrojových kódu a následně nakonfiguruje další aplikace, potřebné pro provoz aplikace Nanoarchive. Modul Cybersearch_ahmed následně nainstaluje a nastaví služby *php5-fpm*, *fastcgi* a server Nginx. Posledním krokem je vytvoření inicializačního skriptu, který obalí aplikaci Nanoarchive jako službu a poté ji spustí. Celý systém je v konfiguraci provázán a nakonfigurován tak, aby například při detekci nové verze zdrojových kódů došlo k jejich aktualizaci a následnému restartu všech závislých služeb. Na závěr dojde k připojení sdílených složek z NFS serveru umístěného na stroji se streaming serverem.

6.4.5 Modul Cybersearch

Úkolem modulu je konfigurace stroje Cyber. Konfigurace začne instalací a nastavením běhového prostředí pro jazyk Java a aplikačního serveru Tomcat 7. Na základě seznamu aktivních instancí jsou vygenerovány všechny konfigurační soubory aplikačního serveru a nastaveno administrátorské rozhraní. Součástí instalace je vygenerování konfigurace správce databázových připojení k databázovému virtuálnímu stroji, tak aby jej mohly aplikace uvnitř webového kontejneru vyhledat pomocí jména na základě rozhraní JNDI. Dále je provedena instalace systému pro odesílání zpráv RabbitMQ s instalací grafického administračního rozhraní. Následuje instalace systému pro integraci řešení Jenkins. Součástí konfigurace Jenkins je vygenerování adresářové struktury jednotlivých úloh reprezentující aktivní instance. Následně dojde k instalaci serveru Nginx a konfiguraci reverse proxy k jednotlivým administračním rozhraním služeb a aplikačního serveru. Součástí konfigurace je i připojení sdílených složek z NFS serveru, umístěného na stroji se streaming serverem.

6.4.6 Modul Nagios

Modul Nagios se stará o instalaci a konfiguraci služby Nagios, sloužící pro monitoring platformy. Základem je automatická kompilace ze zdrojových kódů s instalací základních zásuvných modulů. Po instalaci je smazána veškerá základní konfigurace a vygenerována nová sada konfiguračních souborů pro každý virtuální stroj a konfigurační soubor pro monitoring platformy jako celku. Dále jsou nakonfigurovány a vytvořeny skripty, sloužící jako další zásuvné moduly pro Nagios. Vytvořené skripty rozšiřují funkcionalitu o možnost monitorovat aktivitu virtuálních strojů, využití procesorového času, volného místa datového úložiště a kontrolovat frontu zpracovávaných úloh. Po dokončení konfigurace je s pomocí modulu Nginx realizována konfigurace reverse proxy a instalace služeb *php5-fpm* a *fcgi* pro zpřístupnění uživatelského rozhraní.

U každého virtuálního stroje je poté automaticky monitorováno využití procesoru, operační paměti, volné diskové místo, zda-li je aktivní a zátěž webového serveru. Na stroji Cyber jsou dále monitorovány aktivní instance. Na stroji Ahmed je monitorován běh aplikace Nanoarchive a množství úloh, čekajících na zpracování systémem HTCondor.

6.5 Postup pro vytvoření nové platformy

Před samotnou konfigurací je potřeba poskytnout licenční klíče pro stroje s operačním systémem Windows 7 a vyřešit licenční podmínky pro použití rozpoznávače NanoDictatet. Poté je možné upravit některé části konfigurace, týkající se hardwarových dispozic cílové platformy a další specifické požadavky, jako například počty strojů, využití vlastního databázového stroje či umístění monitorovacího řešení. Na základě těchto informací je vytvořena konfigurace na centrální jednotce. Poté je již nutné na cílovém stroji ručně nebo s pomocí poskytnutého skriptu nainstalovat požadovanou verzi aplikace Puppet. Následuje plně automatizovaný proces a není třeba do jeho průběhu nijak zasahovat. Pokud by během zavedení došlo k chybám, je chyba odeslána na centrální jednotku a analyzována.

Chyby týkající se krátkodobých problémů s připojením, jsou automaticky vyřešeny při dalším pokusu o konfiguraci v základu nastaveného po 30 minutách. Většinu konfiguračních chyb (neexistující složka, špatně připojený disk, špatné údaje o volné paměti či počtu jader procesoru) je možné řešit z centrální jednotky a není potřebný zásah, vyžadující přístup do stroje. Pouze v případech vzniklých obvykle kombinací několika chyb může dojít k takové

chybě, kterou není možné vyřešit automatizovaně a je nutný správce řešení. Zásah správce řešení probíhá zpravidla zrušením stroje a odstraněním jeho certifikátu z centrální jednotky. Nástroj Puppet při další kontrole konfigurace zjistí nepřítomnost stroje a opakuje instalaci znovu.

Po dokončení konfigurace je systém připraven k provozu. Posledním krokem je nutnost pustit automatické sestavení a umístění aplikace na aplikační server z uživatelského rozhraní nástroje Jenkins. I když by tento krok mohl být jednoduše automatizován také, bylo rozhodnuto jej nechat jako samostatný krok k zaručení určitého oddělení od automatizace po fyzické sestavení aplikace a spuštění. Zároveň je tento mezikrok zvolen i jako možnost provést ruční testování především prvků zabezpečení, které není možné provést automatizovaně či jednoduše monitorovat pomocí nástroje Nagios. Mezi základní kroky otestování zabezpečení patří především kontrola povolených IP adres, správných přihlašovací údajů do administračních rozhraní a aktivací licenčních klíčů.

6.6 Umístění a zálohování centrální jednotky

V době psaní této práce byla centrální jednotka v provozu uvnitř virtuálního stroje, umístěného ve Virtual Boxu běžícím na stroji, určeném pro vývoj všech částí, týkajících se automatizace a konfigurace platforem. I když toto umístění je výhodné s ohledem na vývoj a především testování, bylo dohodnuto jej po dokončení testování přesunout na zálohované virtualizační řešení.

Přestože je centrální jednotka správcem celého řešení, není nezbytně nutné zálohovat celou instanci virtuálního stroje. Z hlediska zálohování jsou důležité zdrojové kódy modulů, které jsou ale uloženy v repositáři společně s konfigurací a daty databáze Hiera. Zcela kritickým prvkem z hlediska zálohování jsou ale certifikáty pro Puppet CA. Ztráta certifikátu certifikační autority Puppet by znamenala nutnost znovu ručně vydat nové certifikáty pro všechny klienty. Přesto ztráta nebo zneplatnění certifikátu neomezí funkcionalitu platforem, pouze by do doby opravy nebylo možné přidělovat klientům nové konfigurace.

7 Implementace aplikace Cyber

Během návrhu bylo rozhodnuto rozdělit vývoj aplikace na dva samostatné moduly Core a Appserver. Hlavním důvodem rozdělení bylo oddělení logiky samotné aplikace od způsobu, jak budou požadavky přijímány a odpovědi reprezentovány. Tento způsob umožňuje například v případě potřeby využít modul Core jako základ a pro vstup a výstup použít například namísto webové prezentace příkazovou řádku. Uživatelské rozhraní je fyzicky součástí modulu Appserver, ale popisováno je zvlášť jako oddělená součást, realizovaná jako Single Page webová aplikace.

7.1 Spojení funkcionality modulů pomoc IoC kontejneru

V kapitole o automatizaci sestavení bylo popsáno, jakým způsobem jsou moduly zkompilovány do výsledného archivu. Samotné zkompilování automaticky neumožní modulům spolu komunikovat a ani nezaručí například možnost vyměnit modul Appserver. O spojení s ohledem na funkcionalitu se stará aplikační rámec Spring. S využitím rámce Spring, způsobu návrhu zvaném Inverze kontroly (IoC) a jeho součásti zvané Dependency Injection (DI), je možné funkcionalitu pospojovat až za běhu.

Při spuštění aplikace dojde s pomocí rámce Spring k vytvoření kontejnerů, zajišťujících inverzi kontroly na základě konfiguračních tříd. Inverzi kontroly lze zjednodušeně popsat jako způsob, díky kterému je kontrola nad závislostmi mezi třídami umístěna mimo třídy do oddělené logiky. Každý modul obsahuje jednu hlavní konfigurační třídu, vytvořenou spojením interních konfiguračních tříd modulu. Konfigurační třídy se starají o pospojování funkcionality vytvořením instancí tříd, které jsou poté přidělovány za běhu komponentám uvnitř aplikace. Přidělení instancí jednotlivým komponentám je realizováno na základě návrhového vzoru Dependency Injection, způsobem umožňujícím předávat závislosti do konstruktoru třídy nebo pomocí anotací. Každá konfigurační třída je zároveň anotována výčtem profilů, pro které má být aktivní.

Pro příklad lze uvést logiku pro práci s dokumenty, definovanou rozhraním *DocumentFacade,* poskytovaného modulem Core. Třída, implementující toto rozhraní, deklaruje v konstruktoru závislosti na další rozhraní, dostupném uvnitř modulu Core, především rozhraní pro synchronizaci s aplikací Nanoarchive, umístěnou na virtuálním stroji

Ahmed a službu pro práci s databázovými záznamy o entitě *Document*. Výsledkem je tedy strom závislostí, který je potřeba doplnit, aby bylo možné vytvořit instanci třídy implementující rozhraní *DocumentFacade*. O doplnění stromu závislostí se stará aplikační rámec Spring s využitím kontejneru pro inverzi kontroly. Výsledkem je automatické doplnění závislostí za běhu dle aktuálního aktivního profilu. Takto vytvořená konfigurace umožňuje například automaticky pro testovací účely nahradit implementaci rozhraní pro komunikaci se strojem Ahmed jednoduchou logikou, která namísto odesílání zpráv je zapisuje do bufferu, který je poté možné jednoduše ověřit.

7.2 Modul Core

Uvnitř modulu se nachází veškerá logika pro interní provoz aplikace. V následujících podkapitolách je rozebrána vybraná funkcionalita modulu Core. Především je zde detailněji popsán způsob práce s databázemi, frontou zpráv, komunikace se strojem Ahmed, zpracovávání analytických dat, systému mezipamětí a na závěr je rozebrán způsob hierarchického ukládání kategorií a dokumentů a možnosti přidělování přístupových práv.

7.2.1 Reprezentace dat a použité databáze

Modul pro svou činnost využívá relační databázový systém MySQL pro ukládání aplikačních dat a dokumentově orientovanou databázi MongoDB pro analytická data. Pro mapování mezi relační a objektově orientovanou reprezentací dat byl použit rámec Hibernate konfigurovaný s pomocí rámce Spring, který rámci Hibernate při startu přiřadí databázový zdroj. Vyhledání databázové zdroje je vyřešeno pomocí dotazu využívající rozhraní JNDI, které zdroj vyhledá pomocí adresářové služby s využitím konfigurace aplikačního serveru.

Vyhledáním databázového zdroje je odstraněna nutnost specifikovat autentizační údaje pro databázový server uvnitř aplikace a citlivá data byla přesunuta do konfigurace aplikačního serveru. Jako databázový zdroj byla použita standardní implementace, dostupná v rámci projektu Apache Commons DBCP, která zároveň řeší i pooling databázových spojení. Konfigurace spojení s databázovým serverem MongoDB je řešena s pomocí knihoven rámce Spring, které také zároveň řeší interně pooling spojení. Mapování objektů do relační podoby tedy probíhá s pomocí rámce Hibernate a pro dotazování jsou využity jazyky SQL, HQL a Criteria API. Pro mapování do podoby dokumentů ve formátu JSON pro databázi MongoDB bylo využito mapování dostupné v rámci knihovny Spring Data - MongoDB.

7.2.2 Zálohování synchronizačních zpráv pro stroj Ahmed

Jelikož je především u zpráv pro synchronizaci se strojem Ahmed nutné zaručit odolnost systému proti výpadku internetového spojení nebo krátkodobé nedostupnosti, je komunikace se strojem Ahmed řešena pomocí perzistentní fronty zpráv. Jako server pro implementaci fronty zpráv bylo zvoleno řešení RabbitMQ na protokolu AMPQ (Videla and Williams, 2012). Při požadavku je nejprve vygenerována zpráva, která je poté uložena do databáze MongoDB a zároveň do perzistentní fronty zpráv. Uvnitř aplikace je aktivní jedna instance konzumenta pro každou frontu. Konzument sériově odebírá zprávy z fronty a odesílá je na stroj Ahmed. Logika odesílání je realizována pomocí transakcí a v případě chyby spojení vrátí zprávu na vrchol fronty a krátkodobě se odmlčí. Konzument je schopen zároveň detekovat chyby, které není možné klasifikovat jako chyby spojení a v jejich případě je zpráva přeložena do fronty odložených zpráv, upozorněn administrátor a do jeho zásahu je zastaveno další odesílání zpráv z fronty. Zastavení fronty zpráv umožňuje přijímat nové zprávy, ale neodesílá je.

7.2.3 Komunikace se strojem Ahmed pomocí zpráv

Pro obsluhu požadavků, jako například fulltextové vyhledávání nebo získání přepisu dokumentu, bylo vytvořeno rozhraní pro komunikaci se strojem Ahmed. Rozhraní překládá požadavky na nezálohované zprávy. Před přeložením dotazu na zprávu je nejprve provedena kontrola přístupu, jelikož stroj Ahmed nemá uloženy žádné informace o uživatelích a jejich právech v rámci aplikace. Zprávy jsou poté paralelně odesílány pomocí jedné instance konzumenta, která vnitřně využívání pooling spojení se strojem Ahmed. Výsledek dotazu na stroj Ahmed je zpracován do interních struktur, zkontrolován a použit jako výsledek celého dotazu na rozhraní.

7.2.4 Práce s analytickými daty uvnitř databáze MongoDB

Prezentační vrstva sbírá anonymní data o činnosti uživatelů. Shromažďovány jsou informace především o sledovanosti jednotlivých dokumentů. Při přehrávání dokumentů je zaznamenáváno, které minuty audiovizuálního záznamu si uživatel přehrál. Dále je monitorován počet spuštění audiovizuálního záznamu za časová období (den, týden, měsíc). Analytická data jsou ukládána do databáze MongoDB, nad kterou jsou poté prováděny analytické dotazy.

V době psaní této práce byly vytvořeny Map-Reduce dotazy, které umožňují vyhodnocovat a vykreslovat průběh sledovanosti jednotlivých minut nahrávky, její průměrnou sledovanost a kolikrát byla nahrávka přehrána alespoň z *X* desítek procent.

Princip fungování těchto dotazů lze názorně předvést na posledním uvedeném dotazu. Nejprve je provedena první fáze a to mapování záznamů obsahující informace o procentech sledovanosti dokumentu do skupin. Výsledkem rozdělení do skupin je vytvoření uspořádaných dvojic obsahujících na první pozici skupinu a na druhé jedničku. Poté následuje druhá část a tou je redukce. Výsledky z první fáze jsou redukovány do skupin podle první pozice uspořádané dvojice a následně jsou sečteny všechny prvky na druhém místě. Výsledkem dotazu jsou dvojice, kde na první pozici je reprezentant počtu procent a na druhé počet takovýchto dvojic. Výsledek je odeslán klientovi, který před vykreslením provede seřazení výsledků. Výhodou tohoto analytického dotazu je jednoduchost realizace a údržby díky rozdělení algoritmu na logické celky, ale na druhou stranu není tyto výpočty vhodné provádět pro každý dotaz. Proto bylo rozhodnuto, umístit před volání metod, provádějících analytické dotazy, mezipaměť. Záznam mezipaměti je platný následující hodinu pro provedení výpočtu a systém mezipamětí uchovává pouze posledních 1000 hodnot s nahrazovací strategií, odstraňující vždy nejméně žádaný záznam (LFU).

7.2.5 Systém mezipamětí v modulu Core

Jelikož je aplikace realizována jako dva oddělené moduly, jsou mezipaměti umístěny na úrovni vstupu do modulu Core a Appserver. Jako implementace mezipamětí bylo zvoleno řešení EhCache (Daniel Wind, 2013), které bylo možné plně zaintegrovat do aplikačního rámce Spring. Tvorba mezipamětí je realizovaná uvnitř konfigurační třídy rámce Spring, který poté mezipaměti poskytuje celému systému. Ukládání do mezipamětí je prováděno na úrovni vstupu a výstupu z volání metod. Pro tento způsob práce s mezipamětí je možné využít podpory rámce Spring, kdy metodu stačí anotovat *@Cacheable(„nazev_cache“)* a rámec se postará o vyhledání a obalení volání metody pojmenovanou mezipamětí. Další místo pro využití mezipamětí je při práci s relační databází. Použitý rámec Hibernate v aktuální konfiguraci obsahuje systém mezipamětí o dvou úrovních. První úrovní je mezipaměť v rámci jedné *session* rámce Hibernate, v aplikaci realizované zpravidla jako volání metody obalené transakcí relační databáze. Druhá úroveň je globálního charakteru a umožňuje sdílení výsledků z dotazů i v rámci více transakcí. Při využití druhé úrovně mezipaměti je pro

dodržení konzistence dat nutné provádět dotazy do databáze výhradně přes rámec Hibernate. Pro zvýšení efektivity jsou zároveň anotovány metody, které během transakce data neupravují, anotací *@ReadOnly*, která umožňuje rámci Hibernate využít dalších optimalizací s ohledem na zámky a plánování během dotazů do databáze.

S ohledem na uspořádání platformy do virtuálních strojů s centrálním přístupem přes reverse proxy Nginx umístěné nad virtualizačním řešením, bylo rozhodnuto nevyužívat mezipaměti pro ukládání výsledků pro uživatelské rozhraní. Správa a rozhodnutí o tom, které zdroje budou ukládány do mezipaměti, byla přenechána na reverse proxy stroje, provozující virtualizační řešení. Prvním důvodem je s ohledem na výkon a efektivitu nutné umístit mezipaměti co nejblíže ke klientovi. Druhým důvodem bylo pevně specifikovat, kde bude obsah ukládán do mezipamětí. Vstupní reverse proxy se ukázala jako efektivní řešení, které navíc zaručuje, že mezipaměti budou dostupné pouze pro dotazy z okolní sítě a ne pro komunikaci uvnitř virtuální sítě.

7.2.6 Dokument jako reprezentace multimediálních dat

Celý systém pracuje s pojmem dokument. Dokument obsahuje především audiovizuální materiál a volitelně soubor s přepisem. Každý dokument je explicitně reprezentován v rámci systému unikátním identifikátorem a implicitně jménem instance, ve které byl vytvořen. Jelikož nemusí na celé platformě běžet pouze jedna instance, je nutné vždy při přístupu k dokumentu uvést jméno instance jako součást URL. Při dotazu je podle jména instance provedeno směrování do interní sítě na cílový virtuální stroj a uvnitř stroje na cílovou aplikaci do kontejneru serveru Tomcat. Například při pokusu o zahájení přehrávání dokumentu je nejprve dotaz ověřen na příslušné instanci aplikace na stroji Cyber a poté teprve směrován do virtuálního stroje se streaming serverem.

7.2.7 Postup při zpracování dokumentu

Prvním krokem při zpracování dokumentu je nahrání audiovizuálního materiálu s volitelným přepisem. Pro zpracování jsou aktuálně podporované například formáty mp3, flv, wav, mp4 a avi. Pro nahrávání přepisů jsou podporované interní formáty trsx a res. Nahrání dokumentu je možné provést z administračního rozhraní nebo z programu pro dávkové zpracování. Pro přenos dat souboru je využita binární verze protokolu Websocket. Po nahrání je jméno souboru anonymizováno pomocí hashovací funkce a z dat vytvořen databázový záznam.

Následně je s pomocí fronty zpráv na stroji Cyber zkontaktován přiřazený program Nanoarchive na spárovaném stroji Ahmed a je mu odeslána zpráva o vytvoření dokumentu společně s hashem dokumentu. S pomocí hashe si poté Nanoarchive vyžádá dokument ze stroje Cyber. Po úspěšném stažení souboru je soubor ze stroje Cyber vymazán. Stroj Ahmed následně zařadí konverzi dokumentu do fronty pro zpracování systému HTCondor.

HTCondor při detekci volného výpočetního slotu přenese dávku souborů nutných pro konverzi na pracovní stanici. Hlavní součástí dávky je soubor dokumentu s programem pro konverzi mediálních souborů. Během vývoje byl nejprve využíván nástroj FFmpeg, který je ale v aktuální verzi nahrazen za nástroj Avconv. Na pracovní stanici je poté proveden převod do podoby použitelné pro rozpoznávač (formát wav, 16 kHz) a do formátu pro streaming server (mp4). Po konverzi stroj Ahmed zařadí výsledek znovu do systému HTCondor jako další dávku. Tato dávka již obsahuje instanci rozpoznávače a po zpracování je jejím výstupem přepis. Výsledný přepis je pak začleněn do systému a za indexován pomocí nástroje Sphinx Search.

Součástí zpracování dokumentu jsou i další volitelné úkoly. Například v případě použití formátu získaného ze záznamů přednášek systému MediaSite jsou součástí i sekundové snímky prezentací, které jsou zpracovány, prořezány a vybrané slidy prezentace označeny jako klíčové. V případě použití formátu obsahující video, jsou vytvořeny náhledy z videa pomocí nástroje Avconv. Po dokončení konverze do formátu pro streaming server na pracovní stanici je soubor nástrojem MP4Box převeden do podoby vhodné pro streaming. Úprava pro streaming spočívá v přesunutí údajů hlavičky na začátek souboru, tak aby nebylo nutné před započetím přehrávání načíst celý soubor v přehrávači webového prohlížeče, ale stačilo načíst pouze hlavičku souboru. Po konverzi je soubor umístěn do složky uvedené ve vygenerované konfiguraci aplikace Nanoarchive.

Stroj Ahmed je na konfigurován tak, aby cílová složka byla sdíleným síťovým úložištěm a při ukládání je soubor přenesen do centrálního úložiště umístěného na virtuálním stroji Data, odkud je možné již mediální soubor přehrávat. Stejným způsobem jsou přeneseny i další soubory vzniklé během zpracování. Po dokončení zpracování jsou vymazány všechny dočasné soubory a zpracování dokumentu je dokončeno. Během výše uvedeného procesu je ve vybraných momentech (konverze, přepis, indexace) informována pomocí zpráv aplikace na stroji Cyber. Na základě zpráv je možné v administraci sledovat celý průběh zpracování.

Celý proces je realizován tak, aby v případě úspěšného zpracování dokumentu byly všechny výsledky uloženy na datovém a databázovém stroji a na ostatních strojích nezůstala žádná data vytvářená během zpracování. Tento způsob umožňuje provozovat stroje Cyber, Ahmed a pracovní stanice s velmi nízkými požadavky na potřebnou diskovou kapacitu a volné diskové kapacity využívat pouze jako mezipaměť. Jelikož je mezi kroky zpracování dokumentu umístěna logika umožňující odklad zpracování (fronta zpráv RabbitMQ, HTCondor), je tento systém schopný pružně reagovat na zvýšenou zátěž a s ohledem na dobu přepisu, která je zpravidla delší než doba záznamu, jej prakticky není možné přetížit. Zároveň je systém schopný udržovat ve frontách na zpracování množství dokumentů omezené pouze diskovou kapacitou jednotlivých členů.

7.2.8 Dávkové zpracování dat

Jelikož systém začal být vyvíjen v době, kdy již bylo k dispozici velké množství dat, která bylo potřeba nahrát do systému, bylo rozhodnuto implementovat program pro automatické dávkové zpracování velkého množství dat. Program je ovládán z příkazové řádky a je napsán v jazyce Python. Po přihlášení program nejprve přečte vstupní soubor a vytvoří strom kategorií s dokumenty. Následně je strom kategorií sesynchronizován se stromem na serverové části a případně jsou vytvořeny nové kategorie. Pomocí protokolu Websocket jsou poté sériově odeslány všechny dokumenty a volitelně přepisy. Jelikož může odeslání v závislosti na množství dat a rychlosti internetového připojením trvat i v řádu desítek hodin, je součástí programu mechanizmus, který v případě chyby pokračuje dalším dokumentem a na konec vytvoří opravný soubor. Opravný soubor je poté možné znovu pustit jako vstup programu. Jelikož je možné odesílat i značně velké dokumenty je průběh odesílání každého dokumentu zobrazován na výstup konzole.

7.2.9 Systém oprávnění pro přístup k dokumentům

Každý dokument je možné přiřadit do libovolného počtu kategorií. V případě nepřiřazení dokumentu, nebo odebrání ze všech kategorií, je stav dokumentu označen jako volný. Volný dokument není dostupný uživatelům a pouze volný dokument je možné kompletně odstranit ze systému. Každá kategorie patří do libovolného počtu skupin, které zároveň nesou informace o oprávněních a uživatelích přiřazených do skupiny. Kategorie tvoří stromovou strukturu s virtuálním kořenem.

Jelikož jsou ale kategorie ukládány do relační databáze, bylo nutné vyřešit způsob, jak tuto stromovou strukturu efektivně uložit do relační podoby, tak aby bylo možné nad ní provádět potřebné dotazy bez využití rekurzivních dotazů, které jednak použitá databáze MySQL nepodporuje a zároveň jejich syntaxe není podporována v žádném z jazyků přímo podporovaných rámcem Hibernate. Základním požadavkem byla možnost efektivně, pokud možno nejhůře lineárním průchodem všemi záznamy, vybrat všechny potomky libovolné kategorie, vybrat všechny přímé potomky kategorie a nalézt cestu mezi dvěma členy.

Pro relační reprezentaci byla vybrána kombinace klasického hierarchického modelu společně s modelem označovaným jako vnořené množiny (Celko, 2012). Každá kategorie obsahuje odkaz na předka a pravý a levý index, určující polohu uvnitř stromu. S využitím těchto vlastností lze jednoduše nalézt všechny potomky kategorie, jelikož mají pravý index větší a levý index menší než kořen hledaného podstromu. Pro nalezení přímých potomků je využit odkaz na předka. Nalezení cesty z předka do potomka je komplikovanější záležitostí a nelze ji zjistit přímo. Přesto lze vytvořit efektivní dotaz pomocí následující úvahy. Hledáme- i cestu C z A do B, tak všechny prvky této cesty musí mít levý index mezi pravým a levým indexem A a zároveň levý index B musí ležet mezi pravým a levým indexem každého prvku cesty. Seřazením výsledků vzestupně podle levého indexu získáme hledanou cestu. Při použití databázového indexu na levém indexu kategorie je výsledný dotaz řádově efektivnější než neefektivní řešení pomocí několika rekurzivních dotazů. Využití indexů navíc umožňuje využít systém mezipamětí a zároveň snadno detekovat, kdy je nutné záznam obnovit. Hlavním problémem tohoto řešení je mazání a přidávání nových kategorií. Jelikož musí v obou případech dojít k aktualizaci indexů stromové struktury a následnému přestavění databázových indexů.

7.3 Modul Appserver

Úkolem modulu Appserver je zprostředkovávat komunikace mezi modulem Core a uživatelským rozhraním. Součástí modulu Appserver jsou především soubory potřebné pro webovou prezentaci. Jelikož je uživatelské rozhraní realizováno jako SPI aplikace, není součástí modulu prakticky žádná logika pro generování souborů jazyka HTML a hlavní část tvoří logika pro směrování požadavků, jejich zabezpečení a správa Websocket komunikace.

7.3.1 Směrování požadavků rámcem Struts2

Pro směrování požadavků byl vybrán aplikační rámec Struts2 (Brown a kol., 2008) zaintegrovaný do kontejneru pro inverzi kontroly rámce Spring. Během vývoje bylo rozhodnuto pro poskytování rozhraní definovat hierarchickou strukturu podle entity, které se požadavek týká. Například všechny dotazy, pracující s dokumentem, obsahují jako součást URL část *document*. Na základě hierarchie dotazu se rámec Struts2 stará o směrování do příslušně metody Akce. Akce je interní označení v rámci Struts2 pro instanci třídy, zpracovávající uživatelský požadavek. Samotné přidělení instance Akce je řízeno rámcem Spring. Celková hierarchie všech dotazů je složena při startu z několika souborů s definicemi. Jednotlivé Akce zpravidla neobsahují žádnou komplexní logiku a jsou realizovány tak, aby jejich úkolem byla pouze validace parametrů, zavolání metod fasád poskytovaných modulem Core, případné ošetření vzniklých výjimek a serializace výstupu. Akce nekontroluje uživatelské oprávnění a jako celý modul Core očekává, že dotaz je již ověřen. Příjem požadavků je možný třemi způsoby, jako parametry metody GET, uvnitř těla metody POST nebo ve formátu JSON.

7.3.2 Zabezpečení požadavků pomocí Spring Security

Pro zabezpečení požadavků byl vybrán modul rámce Spring s názvem Spring Security (Winch a Mularien, 2012). Modul byl zvolen především díky prakticky identickému způsobu zápisu oprávnění, jaké je použito pro směrování požadavků rámcem Struts2. Díky tomu lze jednoduše udržovat a porovnávat obě hierarchie a přidání nového záznamu je v obou případech podobné. Pro ověření slouží správci oprávnění. V aplikaci jsou implementováni aktuálně dva správci, jeden pro klasické použití v běžné HTTP komunikaci a druhý pro kontrolu otevření komunikace s pomocí protokolu Websocket. Zabezpečení je nastaveno tak, aby zamítalo jakékoliv požadavky, které nejsou deklarovány v konfiguračním souboru.

Požadavky jsou z hlediska zabezpečení rozděleny podle rolí do skupin anonymní uživatelé, přihlášení uživatele a administrátor. Přihlášení a anonymní uživatelé mají aktuálně stejné oprávnění v rámci povolených dotazů, jelikož obě role smí vykonávat pouze vybrané dotazy pro čtení dat. Pro anonymní roli se ale v závislosti na nastavení veřejných skupin nemusí zobrazit žádné dostupné dokumenty a kategorie. Pro dotazy modifikující stav na serverové straně musí mít uživatel roli administrátor. Ověření a přidělení role je provedeno

před směrováním požadavku rámcem Struts2 uvnitř webového kontejneru. Role se aplikují pro všechny příchozí požadavky s výjimkou spojení pomocí protokolu Websocket. Jelikož otevření spojení pomocí Websocket předchází požadavek pomocí HTTP protokolu, je možné dotaz validovat před změnou protokolu na Websocket. K tomuto účelu slouží druhý správce oprávnění, který provede jednorázové ověření a na jeho základě povolí spojení.

7.3.3 Obsluha Websocket dotazů

Přímá podpora pro protokol Websocket byla přidána do použitého aplikačního serveru Apache Tomcat 7 backportováním z verze 8, která ještě v době zahájení práce nebyla dostupná vůbec a aktuálně je dostupná pouze beta verze 8.05. Proto bylo rozhodnuto použít interní implementaci specifikace Websocket dostupnou uvnitř aplikačního serveru Tomcat rozšířením servletu WebsocketServlet. Při srovnání s aktuálně již dostupnou specifikací JSR 365 se jedná o více nízkoúrovňovou komunikaci a použité knihovny jsou již ve verzi Tomcat 8 označené jako zastaralé. I přesto, že bude tuto část logiky nutné v budoucnu nahradit, bylo rozhodnuto stávající implementaci ponechat do doby, kdy bude rozhodnuto přejít na novou verzi aplikačního serveru Tomcat.

Vytvořená implementace umožňuje přenos souboru z klienta na server pomocí binárních zpráv. Na každou binární zprávu, aktuálně nastavenou na velikost 512 kB, server odpovídá pomocí textové zprávy obsahující identifikátor souboru a procenty stažení. V případě ukončení spojení je kromě kódu chyby kontrolováno očekávané množství přenesených dat, jelikož vzhledem ke způsobu přenášení dat pomocí zpráv je možné komunikaci korektně ukončit i bez nutnosti přenést všechna potřebná data. Dále je kontrolováno maximální množství přenesených dat, aby nebylo možné využít komunikaci přes Websocket k nekontrolovanému zaplnění diskové kapacity stroje Cyber.

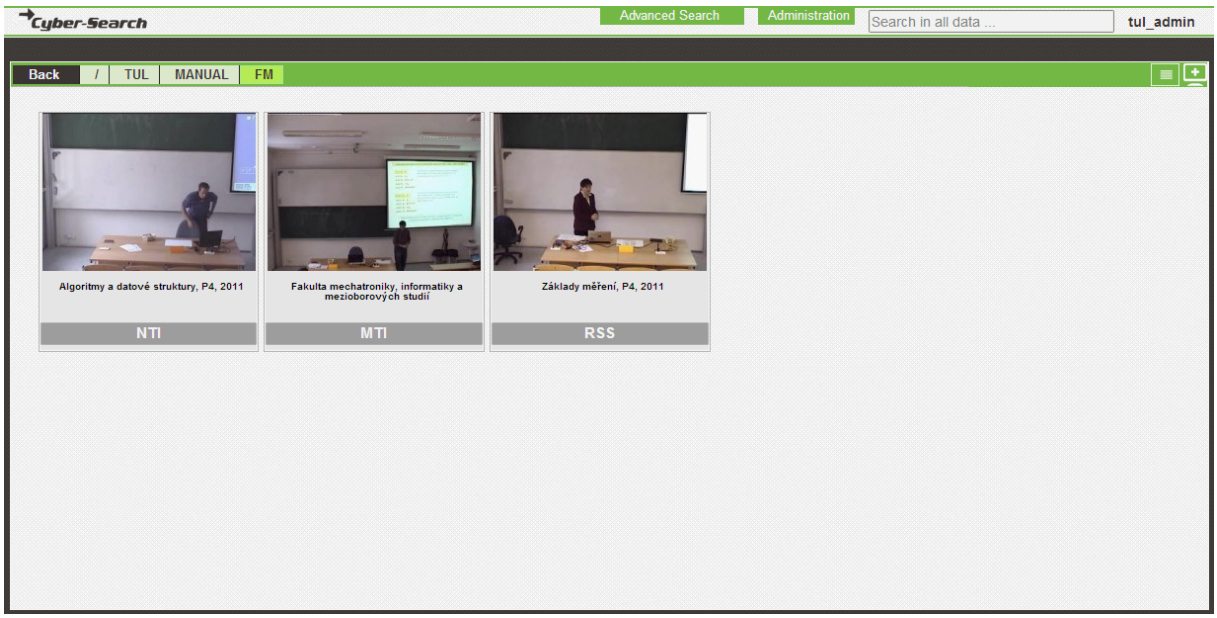
Součástí stažení souboru aktuálně není rezervace diskové kapacity pro uložení souboru a proto může dojít k „povolenému“ zaplnění disku. Jelikož soubory aktuálně může odesílat pouze administrátor, bylo rozhodnuto místo složitějších řešení, která by navíc vyžadovala rezervaci diskové kapacity nejen na stroji Cyber, ale i v rámci celé platformy, spoléhat na monitorovací systém Nagios. Systém Nagios je nakonfigurován tak, aby při zaplnění disku jakéhokoliv stroje z více jak 80% upozornil administrátora a při 90% již hlásil kritickou chybu. Jelikož se tento problém týká především dávkového zpracování, je před započetím dávky možné zkontrolovat volné diskové kapacity z uživatelského rozhraní Nagios.

7.4 Uživatelské rozhraní

Uživatelské rozhraní je rozděleno do 4 hlavních perspektiv a jedné interní perspektivy. Úvodní perspektivou je katalog. Pokud uživatel začne vyhledávat, je automaticky přepnut do perspektivy pro vyhledávání. S možností vyhledávání se pojí i interní perspektiva, která slouží pro pokročilé vyhledávání. Při vybrání dokumentu uživatelem, ať už z katalogu nebo z výsledků vyhledávání, dojde k přechodu do perspektivy pro přehrávání. Poslední perspektivou, dostupnou pouze přihlášenému uživateli, je perspektiva pro administraci řešení.

Přechod mezi perspektivami je řešen pomocí postranních záložek. Aktivace záložky uspí aktuální pohled a ukončí všechny aktivity, které spotřebovávají zdroje. Během uspání ale nedochází ke ztrátě jakékoliv informace a uspanou perspektivu je možné kdykoliv znovu aktivovat. I když logika uživatelského rozhraní obsahuje nejvíce kódu z celé aplikace Cyber, bylo rozhodnuto pouze krátce popsat vybrané části z uživatelského rozhraní, jelikož cílem práce není poskytnou podrobný popis, jak implementovat a navrhovat uživatelské rozhraní.

7.4.1 Perspektiva katalog



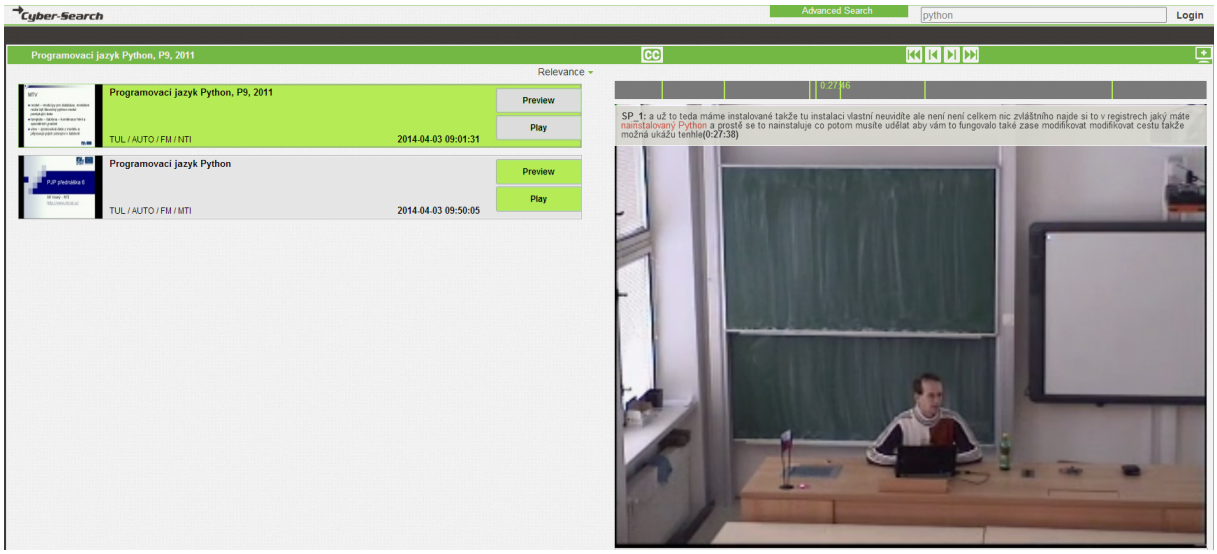
Obr. 16: Grafická podoba perspektivy katalog (tabulkové zobrazení)

Perspektiva umožňuje rychle najít libovolnou kategorii a k ní přiřazené dokumenty. Během procházení stromové struktury kategorií je možné přepínat mezi tabulkovým a řádkovým zobrazením. Pro každou kategorii je vygenerován náhled z dokumentů (Obr. 16), které obsahuje, společně s náhodně vybranými dokumenty všech podkategorií. Z důvodu výkonu je

ale výběr omezen na maximálně pět podkategorií a z každé vybrány maximálně tři dokumenty. Náhled obsahuje vybrané klíčové snímky pořízené z videonahrávky nebo v případě dostupnosti náhodně zvolené slidy. Náhledy kategorií není možné ovládat a jsou generovány automaticky.

Podobný náhled je dostupný i pro dokumenty. Náhled pro dokumenty obsahuje snímky videa zároveň se slidy společně s přepisem aktuálně mluveného textu. V případě dokumentů je možné procházet časovou osou, zobrazenou pod náhledem a v případě potřeby začít přehrávání od aktuálně navolené časové hodnoty. Po vybraní dokumentu dojde k uspání perspektivy katalog a je aktivována perspektiva pro přehrávání.

7.4.2 Perspektiva pro vyhledávání



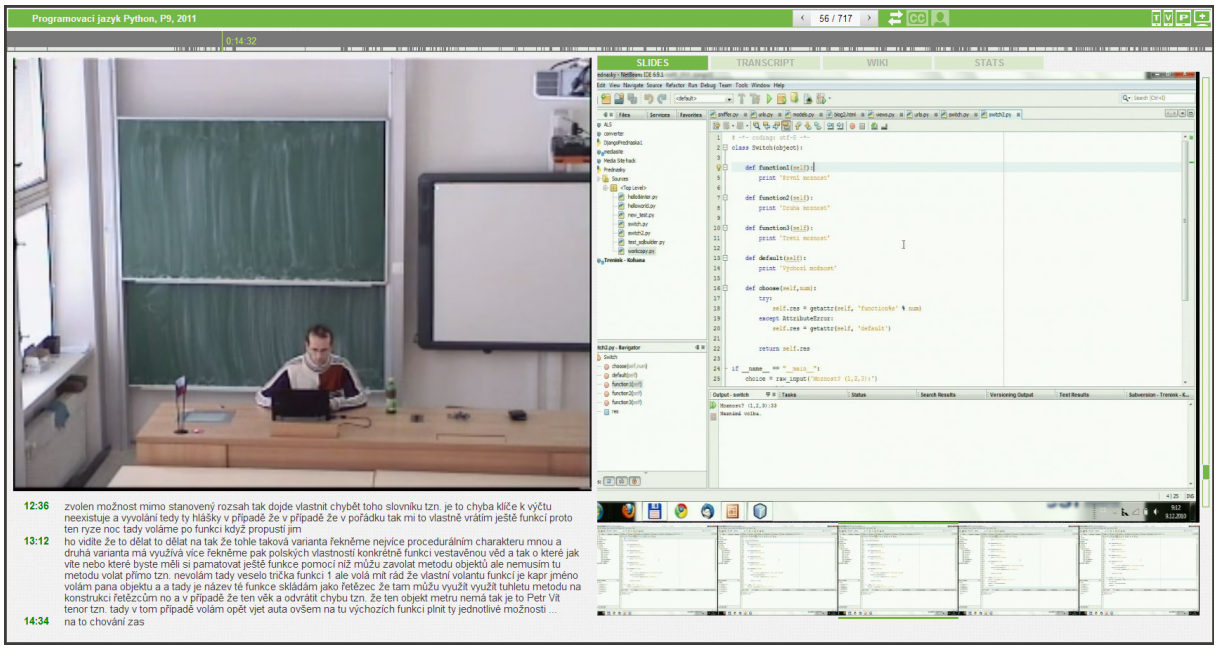
Obr. 17: Grafická podoba perspektivy pro vyhledávání

Perspektiva pro vyhledávání slouží k prezentaci výsledků fulltextového hledání v přepisech řečových nahrávek. K aktivaci perspektivy dojde krátce po zaznamenání uživatelského vstupu v liště pro vyhledávání. Výsledky vyhledávání jsou automaticky obnovovány na základě uživatelského vstupu a další omezujících podmínek, zvolených v perspektivě pro pokročilé vyhledávání. Po aktivaci nálezu je zahájeno přehrávání a uživateli se zobrazí časová osa s vyznačenými časovými body nálezu hledané fráze (Obr. 17).

U každého výsledku vyhledávání je možné aktivací zobrazit cestu ve stromu kategorií a jejím vybráním se přepnout zpět do perspektivy kategorií přímo do vybrané úrovně. Při přepnutí nedochází ke ztrátě výsledků vyhledávání, pouze dojde k pozastavení přehrávání

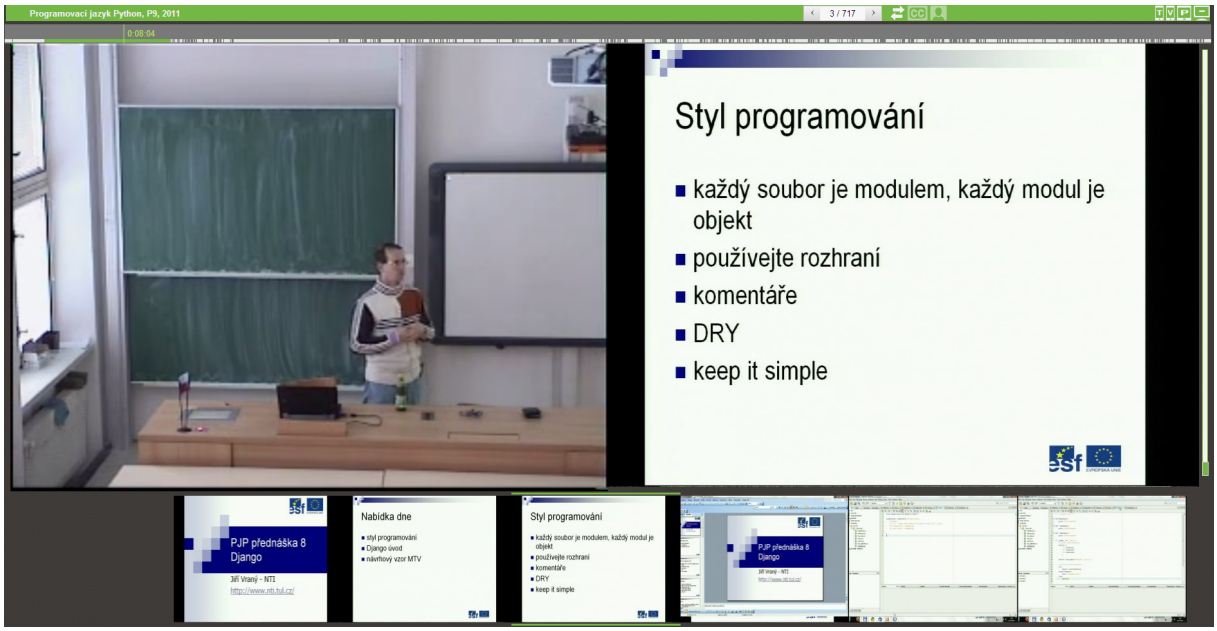
videa. K dispozici jsou dále ovládací prvky, umožňující rychlé přepínání mezi nálezy oběma směry. Přepínání mezi nálezy funguje jednak mezi dokumenty a zároveň i mezi stránkami nálezů. Pro každý dokument je pod videem zobrazena lišta, poskytující základní informace o sledovanosti zvoleného dokumentu. Součástí funkcionality je stejně jako v případě perspektivy kategorií možnost zobrazit rychlý náhled na dokument zobrazující přepis a náhledy na video a slidy. Hlavním důvodem pro rychlý náhled je možnost procházet video bez nutnosti využít webový přehrávač, který při potřebě rychle projít náhledem může mít poměrně značné datové nároky na internetové připojení a u pomalých mobilních připojení je procházení, oproti načtení náhledů o velikost v řádu kB, výrazně pomalejší.

7.4.3 Perspektiva pro přehrávání



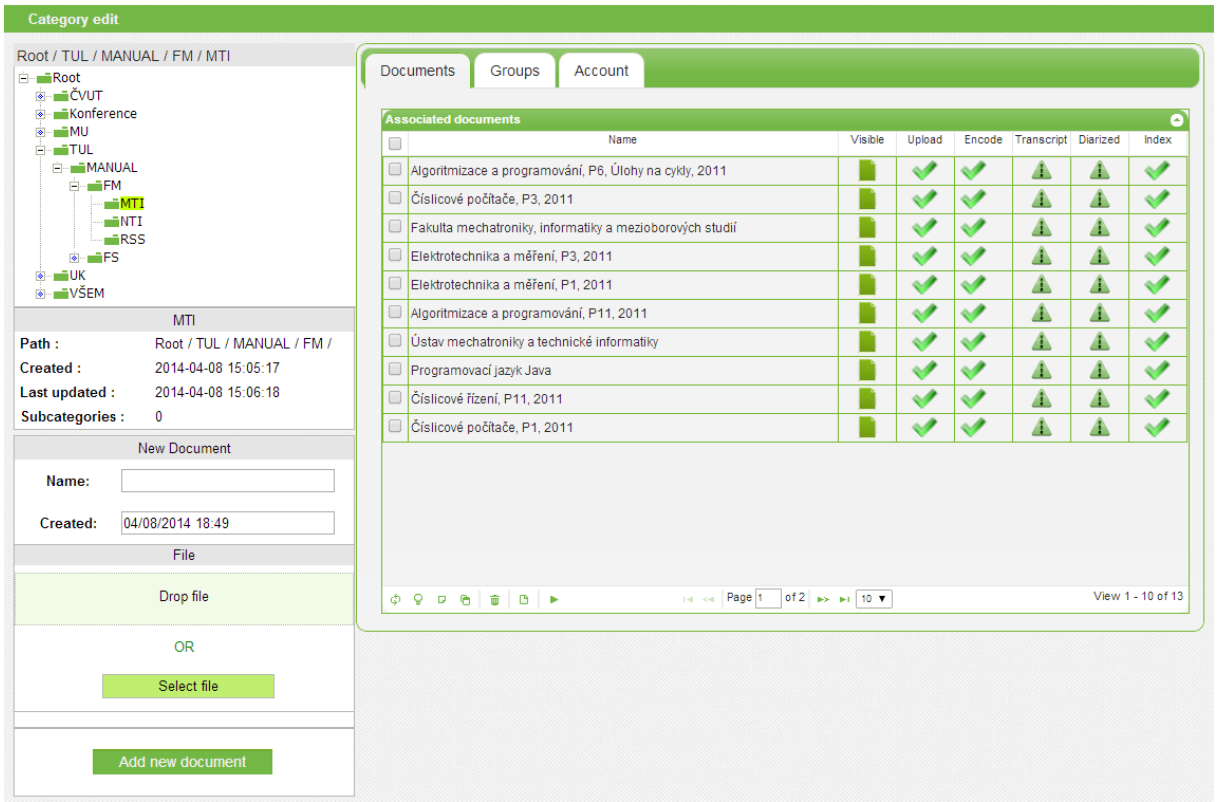
Obr. 18: Grafická podoba perspektivy pro přehrávání

Perspektiva pro přehrávání je co do funkcionality zdaleka nejrozsáhlejší. Hlavní úkolem je přehrávání videa společně se synchronizovaným přepisem, titulky a slidy s náhledy (Obr. 18). Dále perspektiva umožňuje vyhledávat fráze z přepisu v dokumentu nebo na české wikipedii. Součástí je také časová osa, která má v sobě integrovaný náhled na přepis, video a slide v čase. Perspektiva obsahuje záložku zobrazující pomocí grafů analytické informace. Zobrazení lze přepínat mezi čtyřmi režimy pro celou obrazovku a klasickým režimem.



Obr. 19: Přechod do režimu zobrazení na celou obrazovku

7.4.4 Administrační rozhraní



Obr. 20: Grafická podoba administračního rozhraní

Součástí řešení je i administrační rozhraní pro správu celého řešení (Obr. 20). Součástí administračního řešení je implementace asynchronní fonty pro přidávání dokumentů. Fronta

umožňuje zpracovávat nové požadavky na přidání dokumentu nezávisle na činnosti administrátora a tím umožnit například vytvoření větší dávky, která se poté bude postupně zpracovávat. Jelikož je administrace součástí ostatních perspektiv, je možné například kliknutím na dokument přejít k jeho přehrávání. Zde se navíc ukazuje výhoda asynchronní fronty, která i při přechodu k přehrávání na pozadí dále pokračuje ve zpracování.

7.4.5 Pokročilé vyhledávání v dokumentech

Perspektiva pro pokročilé vyhledávání na rozdíl od ostatních perspektiv netvoří samostatný pohled, ale je pouze součást ostatních perspektiv. Proto je možné aktivovat pokročilé vyhledávání v jakékoliv jiné perspektivě (kromě administrace). Pokročilé vyhledávání umožňuje upřesnit vyhledávání pomocí omezení. Aktuálně je možné omezit výsledky vyhledávání na základě času, dne v týdnu a mluvčího. Vybraná omezení jsou řazena do fronty omezení, která jsou aktivní až do odstranění, případně opuštění webové stránky.

7.5 Práce s historií v SPI aplikaci

Během práce se s použitím vlastního aplikačního rámce podařilo vyřešit jeden z největších problémů SPI aplikací, a to práce s historií. Problém vychází ze způsobu, jakým stránka pracuje, jelikož vždy dochází pouze k asynchronní aktualizaci obsahu. Asynchronní aktualizaci obsahu prohlížeč nedetekuje jako změnu historie a ani ji tak z praktického hlediska nelze považovat. Pokud poté uživatel aktivuje tlačítko zpět, dojde ke kompletnímu opuštění stránky. Pro vyřešení této problematiky bylo využito rozhraní pro historii, poskytovanou prohlížečem k implementaci vlastního správce historie. Díky způsobu návrhu aplikace do perspektiv a definováním, kdy může dojít k přechodu, lze jednoduše odposlouchávat zprávy které si mezi sebou moduly přeposílají a ty následně serializovat do adresy. Řešení lze demonstrovat na adrese */catalog/path/25*. V případě adresy je zřejmé, že je nutné aplikaci přepnout do perspektivy pro katalog a rozeslat modulům zprávu o změně aktuální cesty v hierarchii kategorií s identifikátorem 25. Jelikož jsou moduly nezávislé a nemají sdílený stav, pokusí se samy získat potřebná data o cestě s identifikátorem 25 a zobrazit ji. Hlavní výhodou vytvořeného řešení je především jednoduchost a tím pádem i robustnost při použití. Řešení zároveň nevyžaduje prakticky žádnou údržbu při změně logiky, jelikož pouze přijímá a odesílá zprávy a tím pádem není svázáno s implementací ostatních částí.

8 Reálné nasazení platformy

Následující kapitola popisuje příklad reálného nasazení. Zprovoznění proběhlo na univerzitní síti s využitím předkonfigurovaného routeru. Následný popis byl vytvořen anonymizováním reálného případu nasazení. Během popisu jsou uváděny doby trvání jednotlivých částí a slouží pouze pro orientační představu. Případné přesné měření by s ohledem na značné množství faktorů ovlivňující výsledek mělo prakticky zanedbatelnou vypovídající hodnotu.

8.1 Testovací scénář a použitý hardware

Uživatel má zájem o zprovoznění dvou instancí pojmenované *archiv* a *konference*. Instance budou dostupné pod doménovým jménem *app.tul.cz*. Po zprovoznění je do instance *archiv* potřeba nahrát přibližně 15 GB dat. Uživatel platformy vyžaduje mít nad počítačem fyzickou kontrolu. K dispozici je jeden výkonný stolní počítač s následující hardwarovou konfigurací:

CPU	Intel Xeon CPU E3-1245 V2 @ 3.40GHz (4 jádra, 8 vláken)
RAM	16GB DDR3
HDD #1	128GB SSD ADATA SX900
HDD #2	2TB Seagate ST2000VX000-1CU1

Před započetím instalace je nutné získat licenční klíč pro operační systém Windows a pro rozpoznávač. Uživatel má již doménu zaregistrovanou a má pro ni vygenerovaný SSL certifikát. Nejprve dojde k předání licenčních informací a nakopírování SSL certifikátu do centrální jednotky. Posledním krokem před instalací je popis platformy pomocí souboru ve formátu JSON.

8.2 Popis platformy pro automatickou instalaci

Konfigurační soubor uvedený v příloze postačuje ke kompletnímu nastavení platformy. Při popisu konfigurace je díky databázi Hiera využito hierarchického hledání, a tak je zapotřebí uvést pouze údaje, lišící se od základního nastavení. Základní nastavení obsahuje přes 550 základních konfiguračních údajů a také definuje základních 5 strojů. Z těchto důvodů je součástí konfigurace uvedené v příloze A pouze definice stroje ahmed2, který není součástí základní instalace. Deklarace *nginx::kvm::ssl* nastavená na hodnotu *true* zaručí nainstalování SSL certifikátů na klienta. Definice *cybersearch::repo_root* umožňuje vybrat, jestli bude použita vývojová nebo stabilní větev v repositářích pro aplikace Nanoarchive a Cyber-Search.

Definice *nginx::username* a *nginx::password* obsahují uživatelská hesla pro přístup do aplikace, využívaná pro většinu služeb a aplikací. Součástí nastavení je pojmenování instancí konference a archiv a jejich asociace se stroji Ahmed a Ahmed2.

8.3 Základní příprava operačního systému

Následuje instalace operačního systému. Jako operační systém byl zvolen doporučená serverová instalace Ubuntu 12.04 LTS. Při instalaci obou disků je zvolen formát souborového systému *ext4* a druhý disk je připojen jako */media/virtuals*. Z volitelných aplikací byl zvolen pouze SSH server. Po instalaci je stažen skript pro zavedení instalace z centrální jednotky a pomocí skriptu spuštěno zavedení platformy. Poté již není třeba mít ke stroji fyzický přístup.

8.4 Postup během automatické instalace

Zavedení platformy začíná spuštěním skriptu na připraveném operačním systému. Skript nainstaluje systém Puppet a nastaví DNS záznam pro centrální jednotku do souboru hosts.txt. Poté je již kontaktována centrální jednotka, automaticky vygenerován a odsouhlasen certifikát a zahájena instalace. Ta nainstaluje a nakonfiguruje všechny potřebné součásti a následně připraví soubor pro automatickou instalaci virtuálních strojů (soubory preseed, instalační skripty, obraz disku s Windows 7). Následně je provedena instalace všech virtuálních strojů. Součástí instalace stroje je automatické spuštění systému Puppet, který provede konfiguraci virtuálního stroje.

Name	Operating system	Environment	Model	Host group	Last report
ahmed.app.tul.cz	Ubuntu 12....	production	Bochs	CyberSearch_Ahmed	15 minutes ago
app.tul.cz	Ubuntu 12....	production	To be fille...	CyberSearch_KVM	15 minutes ago
condor-25032014-152654-90.a...	windows 6.1	production	Bochs	CyberSearch_Windows	12 minutes ago
cyber.app.tul.cz	Ubuntu 12....	production	Bochs	CyberSearch_Application	15 minutes ago
data.app.tul.cz	Ubuntu 12....	production	Bochs	CyberSearch_Data	about 11 hours ago
db.app.tul.cz	Ubuntu 12....	production	Bochs	CyberSearch_Database	15 minutes ago

Obr. 21: Kontrola platformy z rozhraní Foreman

Celkové zprovoznění platformy trvalo necelé dvě hodiny. S ohledem na množství operací, které bylo nutno provést, se jedná o velmi dobrý čas. K rychlosti značnou měrou přispělo použití SSD disku a především rychlost internetového připojení. V případě použití pouze mechanických pevných disků a pomalejšího internetového připojení, lze očekávat čas

přesahující tři hodiny. Během celého procesu od spuštění prvotního skriptu nebylo potřeba žádného lidského zásahu a informace o průběhu instalace byly automaticky ukládány do centrální jednotky pro případnou detekci chyby.

8.5 Spuštění aplikace po automatizované instalaci

Po zprovoznění je celý systém funkční a připraven na spuštění aplikace. Součástí zprovoznění je nastavení serveru pro automatizované sestavení aplikace. Sestavení a spuštění aplikace z grafického rozhraní serveru je tedy posledním krokem. Po kliknutí na sestavení je stažen zdrojový kód z repositáře a nakonfigurován pro potřeby konkrétní aplikace. Následuje stažení závislostí, sestavení a následné umístění na aplikační server. V závislosti na rychlosti internetového připojení je tento poslední krok proveden během několika minut. Pomocí serveru pro sestavení je kdykoliv možné aplikaci aktualizovat na nejnovější verzi. Aktualizace aplikace trvá na uvedeném hardware přibližně minutu. Z této doby je ale aplikace nedostupná pouze několik sekund v době, kdy dochází k fyzické výměně webového archivu.

8.6 Výsledné dávkové nahrání dat

Posledním krokem je nahrání dat pomocí programu pro dávkové zpracování. Během nahrávání dat je možné z administračního rozhraní sledovat průběh zpracování. Podle typu dokumentů a jejich velikosti se následně odvíjí doba zpracování dávky. Zpravidla však zpracování dokumentu trvá minimálně 1 až 2 krát déle než je délka záznamu. Testovací dávka obsahovala 113 dokumentů o celkové velikosti 13GB. Dávka se skládala z video souborů se slidy s délkou v rozmezí 1 hodiny až 1.5 hodiny, videozáznamů bez slidů s délkou v rozmezí 0.5 hodiny až 1.5 hodiny a mp3 souborů o podobné délce. Celková doba zpracování trvala přibližně den a půl při využití 4 front pro zpracování na pracovní stanici. Po přenesení dat programem pro dávkové zpracování, trvající přibližně 2 hodiny, je již celý proces řízen vnitřně a není třeba jej nijak kontrolovat. Průběh zpracování je v případě potřeby možné kontrolovat v administračním rozhraní.

9 Škálovatelnost platformy Cyber-Search

Rozdělení funkcionality do virtuálních strojů, umístěných uvnitř virtualizačního řešení, umožňuje škálovat řešení jek vertikálně, tak také zároveň horizontálně. Horizontální škálovatelnost umožňuje přidávat virtuální stroje na platformu. Navržené řešení umožňuje přidávat další stroje pouze změnou konfigurace na centrální jednotce, která následně provede všechny potřebné konfigurace na cílové platformě. Horizontální škálovatelnost je důležitá především pro stroje Ahmed, které jsou přidávány pro nové instance aplikací na stroji Cyber. I když je možné horizontálně škálovat i pracovní stanice, není to s ohledem na potřebu licencí a použití Windows 7 efektivní způsob. Pro pracovní stanice je mnohem efektivnějším způsobem využít možností vertikálního škálování. Vertikální škálování umožňuje upravovat výkon virtuálních strojů, v případě pracovních stanic především počtem přidělených procesorových jader. Během testování bylo pro pracovní stanici odzkoušeny konfigurace s 2, 4 a 8 procesory. Při přidávání procesorů pro Windows 7 je nutné zajistit, aby byly přidělovány jako procesorová jádra v rámci jednoho procesorového soketu (Windows 7 podporuje maximálně 2 procesorové sokety). Toho lze jednoduše docílit použitím upřesňujících parametrů *sockets* a *cores* při konfiguraci.

9.1 Výkonové omezení platformy

Následující podkapitoly popisují dva hlavní limitující faktory při škálovatelnosti řešení. Prvním problémem je maximální teoretická propustnost pro streaming server a druhým výkon s ohledem na zpracování dokumentů. Poslední kapitola se zabývá popisem řešení pro sdílení pracovní stanic.

9.1.1 Maximální teoretická propustnost pro streaming

Pro výpočet budeme uvažovat video ve formátu H.264/MPEG-4 AVC s HD rozlišením (1366x768 px), přehrávané 20 fps. Použitím výpočtu pro hrubý odhad uvedeného Derekem Stanlym (2014), získáme pro výše uvedené parametry požadovanou propustnost přibližně 180 kB/s. Při použití pevných disků s 7200 ot/s a kapacitou kolem 3 TB lze očekávat podle dostupných měření přenosovou rychlost přibližně 140 MB/s (Manuel Masiero, 2013). S použitím systému mezipamětí na streaming serveru lze tedy teoreticky přesáhnout hranici 800 současně přehrávajících uživatelů.

Problém nastává ale s propustností internetového připojení. Propustnost 140 MB/s (1,12 Gb/s) vyžaduje prakticky privátní připojení k minimálně 2Gb Ethernetové lince. V běžném nasazení lze ale očekávat maximálně 100 Mb propustnost. S použitím 100 Mb linky lze dosáhnout maximálně 90 současně přehrávajících uživatelů. Po uvážení dalších vlivů (ostatní uživatelé, HTTP režie) lze pro běžné nasazení počítat s maximální zátěží kolem 50-60 uživatelů.

Zvýšení propustnosti je možné dosáhnout zvýšením datové propustnosti internetového připojení. Zvýšení propustnosti je sice spolehlivý způsob, ale obvykle velice nákladný a někdy i velice těžko realizovatelný. Mnohem jednodušší a rychlejší alternativou je pronájem služeb poskytovaných v rámci sítí CDN a nechat streaming data poskytovat jejich infrastrukturu. Problémem je v tomto případě nemožnost provádět kontrolu práv k mediálnímu souboru a v závislosti na množství dat i cena.

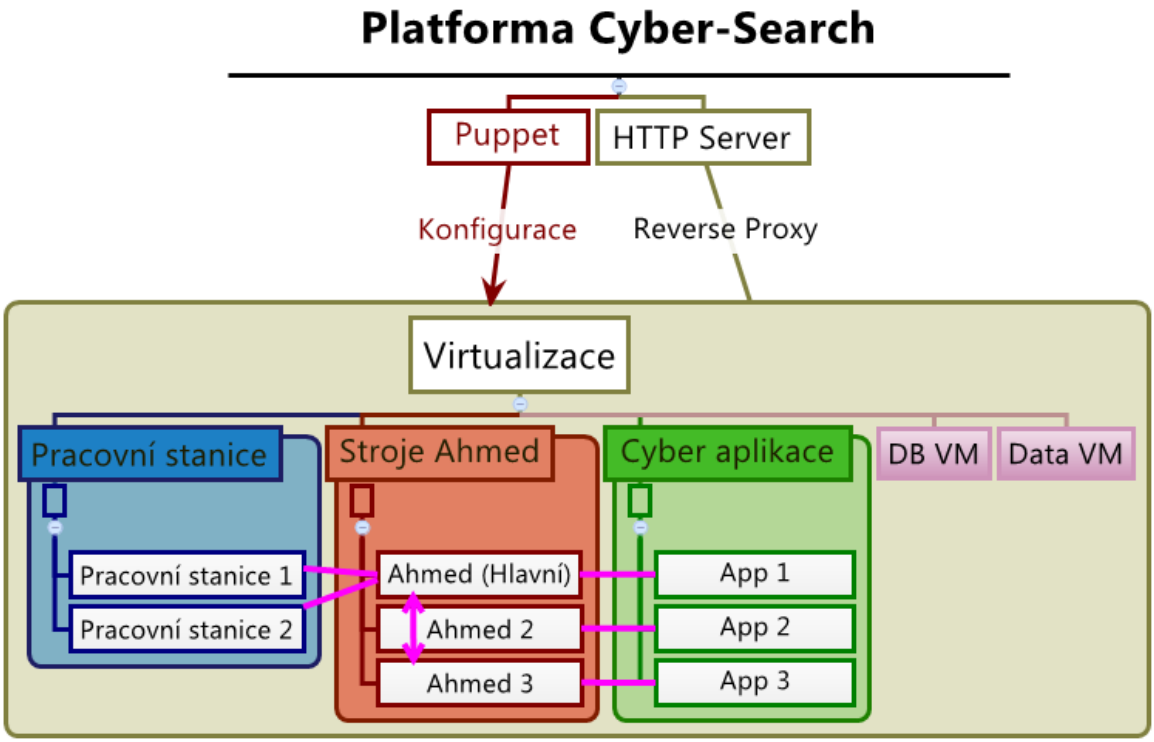
9.2 Výkon při zpracování dokumentů

Stávající implementace rozpoznávače neumožňuje paralelní zpracování a je možné jej urychlit pouze zvýšením frekvence procesoru nebo zpracováváním více dokumentů najednou. Pro počítač, použitý během testovacího nasazení v kapitole 8.2 (8 jader s HT, 4 fyzická, 3.4 GHz), je možné využít při běžném provozu 4 jádra pro pracovní stanici. V případě potřeby většího výkonu je nejvhodnějším řešením využít dalších stolních počítačů pro zrychlení výpočtu. Stávající implementace neumožňuje automatizovaně přidávat další pracovní stanice umístěné mimo platformu. I když tento postup nebyl zatím otestován, jako jedno z jednoduše realizovatelných řešení se jeví možnost připojit stroje přes VPN síť do privátní sítě virtualizačního řešení.

Další možností je realizovat řešení na serverové platformě, podobné například univerzitnímu řešení, popsaném v kapitole 4.2.3 nebo využít možností cloudových IaaS řešení. Nevýhodou obou výše uvedených možností je řádově vyšší cena a především u IaaS řešení navíc problém s nižšími frekvencemi běžně používaných procesorů. Například Google Compute Engine na virtuálních strojích v době psaní této práce garantuje pouze 2.7 GHz Intel Xeon procesory.

9.3 Konfigurace pro více strojů Ahmed

Jelikož je pro každou pracovní stanici potřeba licence operačního systému Windows 7 společně s licencí rozpoznávače, bylo rozhodnuto implementovat logiku umožňující sdílení pracovních stanic libovolným množstvím strojů Ahmed. Pokud systém pro automatickou konfiguraci detekuje více strojů Ahmed (Ahmed1, Ahmed2), zvolí centrální strojem stroj Ahmed a na ostatních strojích překonfiguruje HTCondor mastera, tak aby úkoly přenášely na centrální stroj Ahmed (Obr. 22). Tento mód je označovaný v konfiguraci systému jako *flocking* a jeho součástí je nutnost nakonfigurovat pracovní stanici tak, aby přijímala úkoly od všech strojů Ahmed, nacházejících se v privátní virtuální síti. Zvolené řešení kromě finanční úspory umožňuje také mnohem snadnější správu výpočetní kapacity přidělováním nebo odebíráním front na pracovní stanici. Při použití stejných priorit dochází ke spravedlivému a efektivnímu přiřazování výpočetních kapacit jednotlivým strojům způsobem z FIFO fronty. Nevýhodou je závislost na hlavním stroji Ahmed, který musí být v provozu. V případě potřeby je možné z centrální jednotky platformu překonfigurovat s určením jiného centrálního stroje Ahmed.



Obr. 22: Princip škálovatelnosti a provozu více instancí aplikace na jedné platformě

10 Závěr

Cílem této práce bylo vytvoření ucelené soběstačné platformy pro rozpoznávač a podpůrné systémy vyvíjené na ústavu ITE. Dále bylo úkolem pro systém realizovat moderní Single Page Interface webové rozhraní.

Během práce byl navržen a realizován systém, tvořený centrální jednotkou a jednotlivými platformami. Každá platforma je v minimální konfiguraci tvořena 5 virtuálními stroji na virtualizačním řešení KVM. Instalace platformy je realizována jako automatizovaný proces trvající několik hodin a pro jeho provedení je nutné pouze předat licence a na nainstalovaném operačním systému pustit skript. Výsledkem instalace je plně funkční platforma, připravená zpracovávat data. Zpracování dat je plně automatizováno a je potřeba pouze nahrát zvolený soubor. Systém umožňuje dávkové a ruční nahrání dat a poskytuje pomocí grafického rozhraní informace a stavu zpracování. Pro webovou prezentaci byla realizována moderní a vysoce interaktivní webová aplikace s využitím aplikačního rámce pro tvorbu Single Page web aplikací. Platforma je automaticky udržována systémem Puppet v požadováném stavu na základě konfigurace, uložené v centrální jednotce. Součástí platformy jsou dále automaticky konfigurované systémy pro monitoring, umístění aplikace a její správu.

Vytvořením uceleného systému vznikla platforma, která značnou měrou snížila komplexnost správy rozpoznávače a přidružených systémů a zároveň jim umožnila mnohem větší možnosti vzájemné spolupráce. Tím bylo splněno a zároveň značně rozšířeno zadání práce.

10.1 Aktuální nasazení

Systém aktuálně spravuje 3 plně aktivní platformy s 6 instancemi. Kromě testovací a vývojové platformy je k dispozici platforma, sloužící pro záznamy z jednání Akademického senátu TUL. Aktivní je dále platforma pro prezentaci záznamů přednášek a prezentaci dat potencionálním zájemcům. Pro testovací účely byly na platformě prezentovány přednášky z vybraných českých univerzit a pro prezentační účely firmy Newton Technologies a.s. zpracovány díly pořadu „Peklo na Talíři“.

10.2 Prezentace výsledků

Aplikační rámce pro tvorbu SPI rozhraní byl v roce 2013 prezentován na studentské konferenci FM TUL (SKFM). V době psaní práce je již výsledek této práce zapsán na letošní ročník konference. Mimo konference byl výsledek této práce prezentován vedení a vývojářům firmy Newton Technologies.

10.3 Plány pro budoucí vývoj

Budoucí vývoj je nutné směřovat především na dosažení větší samostatnosti systému. Systém aktuálně dokáže automaticky přidávat a konfigurovat virtuální stroje, ale poté již nedokáže měnit jejich hardwarovou konfiguraci bez asistence správce. Mezi hlavní cíle patří možnost realizovat změnu pouze úpravou konfiguračního souboru na centrální jednotce. Dalším krokem je poté s využitím dat z monitoringu umožnit platformě dynamicky reagovat na změny zátěže.

Z hlediska aplikace a uživatelského rozhraní je aktuálně nutné především odladit co nejvíce chyb spojených s různými verzemi webových prohlížečů. Během psaní této práce je do systému implementována integrace s programem NanoTrans, umožňujícím úpravu přepisů a řešena možnost výběru a správy jazykových modelů pro rozpoznávač. Možnost správy modelů a další části rozpoznávače by měla umožnit uživateli například zvolit jaký jazyk a slovník chce použít, případně zkusit přepis znovu s jiným slovníkem.

Seznam použité literatury

BLAVKA, Karel. Indexace a prohledávání multimédií. Liberec, 2010. Dostupné z: http://knihovna-opac.tul.cz/authorities/431209. Diplomová práce. Technická Univerzita v Liberci. Vedoucí práce Ing. Jindřich Žďánský, Ph.D.

BONZENE, Paolo, 2013. Virtualization with KVM [video]. Youtube.Dostupné z: https://www.youtube.com/watch?v=6n-ANRWqll8.

BROWN, Don, Chad Michael DAVIS a Scott STANLICK. Struts 2 in Action. Greenwich, CT: Manning Publications, 2008. ISBN 193398807X.

CELKO, Joe. Joe Celko's Trees and Hierarchies in SQL for Smarties. Second Edition. Amsterdam; Boston: Morgan Kaufman, 2012. ISBN 0123877334.

CHEN, Gary. KVM: Open Virtualization Becomes Enterprise Grade. In: IBM [online]. 2013 [cit. 2014-04-25]. Dostupné z: http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=XSW03128USEN.

CROCKFORD, Douglas. JavaScript: The Good Parts. Farnham: O'Reilly Media, 2008. ISBN 0596517742.

JOHNSON, Rod. Expert One-on-One J2EE Design and Development. Indianapolis: Wiley, 2003. ISBN 978-0764543852.

JU, Dapeng, Chuanyi LIU, Dongsheng WANG, Hong LIU a Zhizhond TANG, 2009. Performance Comparison of IP-Networked Storage. Tsinghua Science & Technology 14, 29–40. doi:10.1016/S1007-0214(09)70004-5.

KLEPÁČ, Martin. Private IaaS cloud comparison. Praha, 2013. Dostupné z: http://www.diplomovaprace.cz/33/thesis.pdf. Bakalářské práce. ČVUT. Vedoucí práce Ing. Tomáš Vondra.

KVM – KERNEL BASED VIRTUAL MACHINE, Red Hat [online]. [cit. 2014-04-05]. Dostupné z: http://www.redhat.com/resourcelibrary/whitepapers/doc-kvm.

MASIERO, Manuel. Tom´sHARDWARE. Seagate Desktop HDD 4TB Review: Big Capacity At 5900 RPM [online]. 2013 [cit. 2014-04-21]. Dostupné z: http://www.tomshardware.com/reviews/desktop-hdd.15-st4000dm000-4tb,3494-3.html

MOTIKA, Gal. 2012. Virtio network paravirtualization driver: Implementation and performance of a de-facto standard. Computer Standards & Interfaces 34, 36–47. doi:10.1016/j.csi.2011.05.002.

PĚTNÍK, Jiří. Automatizace úloh v cloudovém prostředí. Praha, 2012. Diplomová práce. ČVUT. Vedoucí práce Ing. Miroslav Čepek.

STANLEY, Derek. EZS3. [online]. [cit. 2014-04-21]. Dostupné z: http://www.ezs3.com/public/What_bitrate_should_I_use_when_encoding_my_video_How_do_I_optimize_my_video_for_the_web.cfm.STEFANOV, Stoyan, 2010. JavaScript Patterns. ISBN 978-0596806750.

STEFANOV, Stoyan. JavaScript Patterns. Sebastopol, CA: O'Reilly Media, 2010. ISBN 9780596806750.

VIDEAL, Alvaro a Json J. W. WILLIAMS. RabbitMQ in Action: Distributed Messaging for Everyone. Shelter Island, NY: Manning Publications, 2012. ISBN 9781935182979.

VOMOČIL, Břetislav, 2009. Linuxový virtualizační nástroj KVM a jeho využití na výpočetních clusterech [online]. [cit. 2014-04-05]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Miroslav Ruda. Dostupné z: http://theses.cz/id/zh48jz/.

WIND, Daniel. Instant Effective Caching with Ehcache. Birmingham, UK: Packt Publishing, 2013. ISBN 1782160388.

WINCH, Robert a Peter MULARIEN. Spring Security 3.1. Birmingham, UK: Packt Publishing, 2012. ISBN 9781849518260.

Příloha A

```
{
    "platform_dns" : "app.tul.cz",
    "cybersearch::repo_root" : "stable",
    "nginx::kvm::ssl" : true,
    "kvm::vms" : {
        "ahmed2": {
            "ip": "%{hiera('kvm::setup::network::subnet')}.33",
            "vcpus" : 2,
            "hdd_size" : 40,
            "ram" : 2048,
            "img_url" : "%{hiera('kvm::ubuntu_precise_url')}"
        }
    },
    "nanoarchive::setup::config::vm_mapping" : {
        "ahmed" : "konference",
        "ahmed2" : "archiv"
    },
    "nginx::username" : "changeme",
    "nginx::password": "changeme",
    "cybersearch::install::instances" : {
        "konference" : {
            "ahmed_vm_name" :"ahmed"
        },
        "archiv" : {
            "ahmed_vm_name" :"ahmed2"
        }

    }
}
```

Obr. 23: Konfigurace platformy pro testovací scénář

Příloha B

Přiložené CD obsahuje:

- Text práce ve formátu PDF
- Text diplomového projektu autora ve formátu PDF