Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Systém pro centrální zpracování logů

Plzeň 2014 Ladislav Hlom

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 25. června 2014

Ladislav Hlom

Abstract

Česky

Cílem práce bylo nastudovat problematiku pořizování a analýzy logových záznamů. Na základě nastudované problematiky vytvořit řešení pro centralizovaný sběr logových záznamů. Řešení je navrženo tak, aby využívalo již používané nástroje a bylo snadné ho nakonfigurovat. Výslednou aplikaci lze snadno rozšířit a umožnit zpracování logového záznamu libovolného formátu. Součástí řešení je uživatelské prostředí pro filtraci, zobrazování a vizualizaci záznamů, které je přístupné pomocí webového rozhraní a poskytuje statistiky nad získanými daty, umožňuje komplexní filtrování a fulltextové vyhledávání.

English

The goal of this bachelor thesis was to study issues of purchasing and analysing log records, and then to create a solution for a centralized collection of log records based on studied cases. The solution is designed to take advantage of already used tools and to be easy to configure. The final application can be easily extended to allow processing the log records of any format. The solution includes a user interface for filtering, displaying and visualization of records, which is accessible through a web interface and provides statistics for the data. It also allows complex filtering and full-text search.

Obsah

1 Úvod

Logový záznam je záznam obsahující informace, co se stalo a kdy. Správa a analýza logových záznámů je problém, který existuje již dlouhou dobu. V současné době ve větším měřítku než kdy předtím. Tomuto problému je v současné době věnováno více pozornosti, neboť společnosti se zaměřují na shromažďování údajů o využití aplikace a celkovém provozu. To má za následek velké množství přijatých logových záznamů, které je nutno zpracovat a získat z nich užitečné informace. Problémem při analyzování logových záznamů je neexistence jednotné struktury, kvůli které je třeba každý typ logového záznamů zpracovávat rozdílně. Často je také potřeba analyzovat záznamy z velkého množství různých serverů, což přináší další problémy.

V práci bude rozebrána problematika logování a centralizovaného řešení pro sběr logových záznamů. Budou probrány možnosti analyzování a filtrování logových záznamů. Z důvodu velkého množství logových záznamů budou rozebrány možnosti pro uložení dat a jejich následného zobrazení. V rámci bakalářské práce pak bude vytvořeno řešení pro centralizovaný sběr logových záznamů. To bude shromažďovat logové záznamy z mnoha různých serverů, získávat z nich důležité informace a zobrazovat je uživateli skrze webové rozhraní. Uživateli umožní vyhledávat nad libovolnými poli a zobrazovat data ve zvolených grafech. Toto řešení by mělo využívat již existující a používané nástroje pro sběr logových záznamů (syslog).

V rámci předvedení práce budou do aplikace posílány logové záznamy z aplikací vytvořených pomocí programovacích jazyků Java a Ruby. Na serveru budou data sbírána, analyzována, filtrována a ukládána. Logové záznamy budou přístupné pomocí webové aplikace, která bude vytvořena ve webovém frameworku Ruby on Rails. Aplikace bude rozšířitelná pro libovolný formát logového záznamu.

2 Logování

V následujících kapitolách bude popsána problematika pořizování, distribuce a ukládání logových záznamů. Bude popsáno co je logový záznam a k čemu slouží spolu se základním rozdělením logových záznamů. Bude uvedeno, jaké vlastností by měli aplikace pro logování splňovat. Na závěr bude probrána možnost centralizovaného logování a budou uvedeny předpokládané cíle výsledného řešení.

2.1 Logový záznam

Logový záznam je záznam, obsahující informace o událostech, které se staly při běhu softwaru. Logové záznamy mohou sloužit k rozpoznání místa a příčin vzniku chyby. Mohou obsahovat informace o tom, jak a kým byla daná aplikace či služba využívána (například z jaké IP adresy bylo přistupováno k serveru). Pořizování a udržování takových záznamu se nazývá logování. V nejjednodušším případě se jedná o ukládání záznamu do souboru. Existuje ovšem více možností, například ukládání záznamů do databáze, odesílání do lokálního syslogu, nebo vzdálené logování po síti, atd. Některé programy umožňují logování vypnout, zapnout nebo nastavit jeho úroveň, tzn. určit kolik a jak detailních informací se do logových záznamů má ukládat.

Formát logového záznamu závisí na typu aplikace/platformě a zvolené formě logování. Logové záznamy pak lze dělit na základní, které informuji o události nebo chybě, kterou chceme zaznamenat (například přihlášení uživatele). A na podrobné záznamy s detailními informacemi o události (například stacktrace). Stacktrace je logový záznam, který může být generován objektově orientovanými jazyky při výjimce. Stacktrace generuje detailní výpis, kde přesně k chybě došlo a je obvykle několik řádků dlouhý. Jeho délka závisí na zanoření volání v aplikaci (od několika až po stovky řádků). Logové záznamy lze dělit podle zařízení, které je vygenerovalo. Základní rozdělení záznamů podle zařízení je na systémové a aplikační, které bude rozebráno níže.

2.1.1 Systémové logové záznamy

Systémové logové záznamy obsahují události, které jsou pořizovány komponentami operačního systému. Systémové logové záznamy tak mohou obsahovat informace o systémových změnách, událostech, operacích, změnách komponent, ovladačů a dalších událostech. Typicky obsahují informace o spuštění a zastavení služby, pokusů o ověření, přístupů k souborům, změn účtů, změn oprávnění a privilegií nebo využití poskytovaných výsad. Systémové logování umožňuje mít přehled o všech komponentách a aplikacích v systému, poskytovat statistiky, umožnit monitorování stavu

celého systému a pomáhat při řešení problémů.

2.1.2 Aplikační logové záznamy

Aplikační logové záznamy jsou vytvářeny pomocí aplikací. Formát a obsah jednotlivých logových záznamů je určený vývojáři softwaru. Informace pořizované různými aplikacemi se proto mohou značně lišit. Mohou zahrnovat změny účtu, pokusů o ověření uživatele, detaily použití, činnosti klienta a činnosti serveru, změny konfigurace, selhání aplikace atd. Jednotlivé záznamy pak mají často vlastní specifický formát, což komplikuje jejich analýzu.

2.2 Důvody k logování

Z praxe je známo, že většina uživatelů aplikací nemá dostatečné znalosti k přesnému popsání problémového chování/stavu aplikace. Naprostá většina hlášení obsahuje pouze popis poslední akce (bylo kliknuto na tlačítko) nebo části aplikace, která uživateli nefunguje. Díky logovým záznamům je možné podle času a vyvolané akce dohledat více informací o akcích uživatele, případně o vložených datech nebo chybových hlášení aplikace. Z těchto dat je pak možné odvodit příčinu a opravit případnou chybu nebo uživateli popsat, co udělal s aplikací špatně. Někdy je také možné objevit problém dříve, než způsobí škodu.

Dalším důvodem pro logování je vytváření statistik. Ze statistik lze poznat, jak je aplikace používána, s jakou četností jsou jednotlivé části volány, jak volání dopadla a jak dlouho trvá jejich vyřízení. Logové záznamy jsou užitečné i pro zpětnou analýzu událostí, například pro aplikace manipulujícími s citlivými daty (peníze, důležitá data), které z nich mohou odhalit podezřelé transakce, pokusy o napadení aplikace nebo nadměrnou zátěž systému atd.

2.3 Užitečné funkce pro logování

Za dobu pořizování logových záznamů bylo definováno mnoho vlastností, které by kvalitní nástroje pro logování měli splňovat. Tyto definice vznikly pro snazší, přehlednější a efektivnější správu logových záznamů.

2.3.1 Rotování logových záznamů

Rotování[1] logových záznamů je funkce, která při dosažení zvoleného limitu provede rotaci souboru. Byla vytvořena z důvodu velkého množství logových záznamů. Jako limit pro použití funkce může být zvolena velikost souboru nebo datum. Rotování logových záznamů podle velikosti umožňuje nastavit maximální velikost souboru (kam se logové záznamy ukládají) na libovolnou hodnotu. Při dosažení nastavené hodnoty se souboru vytvoří kopie a z původního souboru jsou vymazána data nebo se přejmenuje (obvykle se přidá za jméno časová značka) a loguje se do nově vytvořeného souboru. Pracovat s několika soubory o velikosti například 10MB je snazší, než pracovat se souborem o velikosti 1GB. Rotování logových záznamů podle data umožňuje nastavit dobu, po které se má začít logovat do nového souboru. Pak lze mít logové záznamy roztříděné podle měsíců, dnů, hodin nebo libovolně v závislosti na jejich množství. To při znalosti přibližné doby události usnadňuje vyhledávání, neboť lze hledat v konkrétních souborech. Lze navíc jednoduše odstranit staré logové záznamy.

2.3.2 Synchronní logování

Synchroní logování znamená, že aplikace čeká, než se zapíše logový záznam. Aplikace pak nepokračuje do doby, dokud aplikace pro logování nezapíše logový záznam (například do databáze). Existují případy, kdy je nutné nejdříve provést uložení logového záznamu a až poté je možné provést další operace. Například při finančních transakcích chceme nejdříve uložit informaci, co zrovna provádíme a až po úspěšném zápisu logového záznamu transakci dokončit. Nevýhodou může být znatelné zpomalení aplikace, vzniklé v případě problému se zápisem logového záznamů.

2.3.3 Asynchronní logování

Asynchronní logování znamená, že se zapisování logových záznamů provádí nezávisle na aplikaci. Aplikace logový záznam předá k zapsání a již se nezajímá jak zápis dopadne. Díky tomu nemusí aplikace čekat na dokončení operací (IO operací, zápisu do databáze), které by zbytečně zpomalovali běh celé aplikace. O zápis se stará aplikace pro logování, zatímco vlákno hlavní aplikace běží dál. Díky tomu není celá aplikace zpomalena (pokud by například při zápisu do souboru nebylo místo na disku).

2.3.4 Formátování logových záznamů dle šablony

Formátování logových záznamů dle šablony umožňuje upravit logové záznamy do libovolného formátu. Například libovolně uspořádat pořadí jednotlivých informací, zvolit určitý formát zapsání informace (například datum ve formátu rok-měsíc-den) a nebo záznam uložit v určitém datovém formátu (JSON,XML). Šablona je předpis, podle kterého je formátování logových záznamů prováděno. Aplikace pro logování nabízí mnoho proměnných, které lze pro vytvoření šablony využít. Za pomoci těchto proměnných je možné vytvořit logové záznamy s velkým množstvím detailních informací nebo logové záznamy obsahující pouze základní informace.

2.4 Centralizované Logování

Při správě více serverů a aplikací (pro zjednodušení budou servery a aplikace dále označovány jako server) nechceme kontrolovat logové záznamy na každém zvlášť. Serverů může být velké množství a analyzovat záznamy na každém zvlášť je značně neefektivní. Proto se používá přístup, kdy se logové záznamy ze všech serverů odesílají na společné místo, odkud jsou pak přístupné. Na serveru jsou logové záznamy přijímány aplikací, která je filtruje a ukládá dle nastavení. K vytvoření kvalitního řešení pro centralizované logování musíme vyřešit následující fáze.

- Sběr
- Transport
- Zpracování
- Uložení
- Vyhledávání
- Vizualizace

2.5 Porovnání s existujícími nástroji a cíle práce

V současné době se pro centralizovaný sběr logových záznamů nejčastěji používá Logstash a Graylog 2. Ty jsou doplňovány dalšími nástroji, pro zobrazení na webu, indexování nebo ukládání do databáze. Tyto nástroje poskytují funkcionalitu jako jsou upozornění, zobrazení, filtrace a vizualizace dat. Dále umožňují zvolit větší množství vstupů a výstupů, odkud mohou být záznamy získávány a kam mohou být odesílány. Webové aplikace pro zobrazení jsou konfigurovatelné a dají se specifikovat zobrazované položky. Problémem může být konfigurace, která je u Logstash poměrně

jednoduchá oproti tomu konfigurace Graylogu je náročná a často při ní dochází k problémům.

Cílem vyvíjené aplikace je poskytnout podobnou funkcionalitu, jakou poskytují současné konkurenční nástroje (upozornění, zobrazení, filtrace a vizualizace dat). Aplikace si klade za cíl jednoduchou a přehlednou architektura zpracování zpráv. Umožnit snadné rozšíření aplikace pro libovolný formát logového záznamu, které může být vytvořeno libovolným uživatelem a například i sdíleno v rámci komunity. Umožnit dodávat grafy spolu s typy parserů, neboť lze na základě znalosti jednotlivých logových formátů vytvořit mnoho užitečných statistik. Poskytnout možnost jednoduchého přidání a nakonfigurování parseru (parser se bude starat o zpracování logových záznamů). Unifikovat vstupy a výstupy, neboť pro poskytnutí požadované funkcionality stačí jeden typ vstupu a výstupu. Zajistit jednoduchou instalaci a konfiguraci kompletního řešení pro centralizovaný sběr logových záznamů, které by mělo nejlépe používat existující nástroje. Při použití řešení pouze pro několik druhů logových záznamů zajistit minimální konfiguraci.

3 Sběr a transport logových záznamů

Sběr a transport jsou první dvě fáze, které je nutné vyřešit k vytvoření centralizovaného logování. Prvním krokem je pořízení logových záznamu na straně klienta (aplikace) a jejich odeslání do centrálního úložiště. Druhým krokem je sběr příchozích záznamů na straně centrálního úložiště. Oba kroky budou rozebrány v této kapitole.

3.1 Pořizování a odesílání logových záznamů

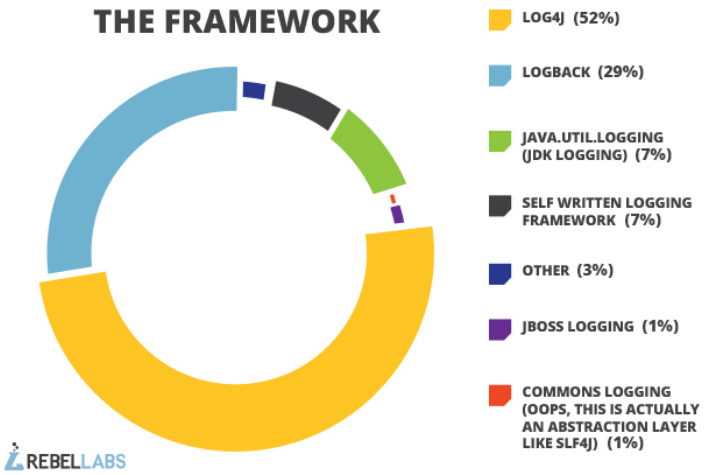
Hlavním požadavkem na framework pořizující logové záznamy je umožnit logování do syslogu, neboť syslog bude použit pro sběr logových záznamů viz sekce 3.2. Dalším požadavkem je poskytnout možnost centralizovaného logování co největšímu počtu klientů, a proto by se mělo jednat o rozšířený framework. Bylo by vhodné, aby obsahoval užitečné funkce pro logování zmíněné v sekci 2.3. Proto bude provedeno srovnání jednotlivých frameworků pro logování na základě výše zmíněného a budou vybrány vhodné. Bude se jednat o frameworky pro programovací jazyky Java a Ruby. Výběr těchto jazyků byl zvolen na základě zadání bakalářské práce.

3.1.1 Framework pro Javu

Pro programovací jazyk Java existuje velké množství frameworků pro logování. Podrobnějšímu porovnání budou podrobeny pouze nejrozšířenější frameworky. O rozšířenosti byly získány přibližné údaje na základě nezávislé ankety viz obr.3.1 [2], kde zodpovídalo 110 lidí (převážně vývojářů a architektů systémů) na otázku, jaký framework pro logování v Javě používají. Na základě této ankety byly vybrány pro porovnání následující frameworky: Logback, Log4j, Java.Utill.Logging (J.U.L) viz tabulka 3.1.

Framework pro logování	J.U.L.	Log4j	Logback
Rotace logových záznamů podle data/velikosti	Ne/Ano	Ano/Ano	Ano/Ano
Logování do syslogu	Ne	Ano	Ano

Tabulka 3.1: Porovnání frameworků pro Javu



Obrázek 3.1: Nezávislý průzkum používaných frameworků pro Javu[2]

Log4j a Logback poskytují všechny požadované funkce. Výhodou Logbacku je použití SLF4J API, které lze do Log4j také přidat. SLF4J je nástroj, který umožňuje záměnu logovacího frameworku za jiný pouhou změnou jar souboru a bez nutnosti změn v kódu. Oproti tomu J.U.L neumožňuje logování do syslogu a rotování záznamů dle data. Proto byly jako vyhovující zvoleny Log4j a Logback. V přiložených souborech bude uvedena konfigurace pro oba frameworky.

3.1.2 Framework pro Ruby

Pro ruby byly nalezeny frameworky pro logování na základě výběru ze zdroje [3]. Na této stránce jsou k dispozici nejpoužívanější nástroje pro Ruby. Na základě počtu stažení byly k testu vybrány dva nejstahovanější: Log4r a Logging. K tomuto porovnání byl přidán výchozí logger, který je v Ruby implementován v základu. Dále Remote_syslog_logger, který umožňuje snadnou konfiguraci pro logování do syslogu. Frameworky podrobíme porovnání viz tabulka 3.2.

Framework pro logování	Ruby výchozí	Log4r	Logging	Remote syslog logger
Rotace logových záznamů podle data/velikosti	Ne/Ano	Ano/Ano	Ano/Ano	Ne/Ne
Logování do syslogu	Ne	Ano	Ano	Ano

Tabulka 3.2: Porovnání frameworků pro Ruby

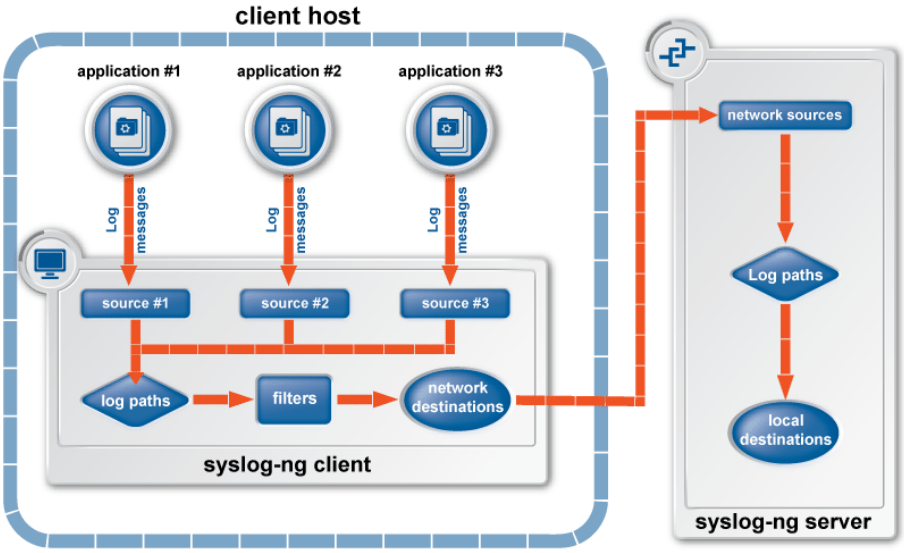
V testu výchozí logger pro ruby nevyhověl všem požadavkům, protože neumožňuje rotaci logových záznamů dle data, ale především neumožňuje logování do syslogu, což je hlavním požadavkem. Pro výchozí logger tedy nebude uvedena konfigurace. Log4r spolu s Logging splňují všechny požadavky. Log4r a Logging vycházejí ze stejného frameworku, kterým je Log4j a mají podobné šablony pro formátování záznamů. Proto bude uvedena konfigurace pouze pro jeden z nich. Remote syslog logger splnil hlavní požadavek logování do syslogu, což může být dostačující pro potřeby centralizovaného logování. Umožňuje především snadnou konfiguraci pro odesílání logových záznamů do syslogu, a proto pro něj bude předvedena konfigurace.

3.2 Sběr záznamů na serveru

Primárním požadavkem pro tvořenou aplikaci bylo pro přijímání příchozích logových záznamů využít existující řešení, které se již používá a umožnit jeho snadné propojení s vytvářenou aplikací. V rámci práce bude vytvořeno řešení pro Unixový systém. Syslog je označován za standard pro logování na Unixových systémech a splňuje výše zmíněný primární požadavek. Syslog dále splňuje požadavky jako je filtrování, upozorňování na předvolené události, zpracování velkého množství záznamů za sekundu a zápis dat do různých výstupů. Proto na základě výše zmíněných důvodů byl vybrán jako kolektor logových záznamů syslog.

3.3 Syslog

Syslog je standardizovaný mechanismus pro logování v počítačových systémech[4], jehož protokol je specifikován RFC5424[6]. V současné době je standardním řešením pro logování na Unixových systémech. Umožňuje oddělení softwaru, který logové záznamy generuje od softwaru, který je ukládá a zpracovává. Existuje velké množství implementací syslogu. Jednotlivé implementace mají rozdílné vlastnosti a každá se hodí pro určité použití. Od jednoduchých implementací, které ukládají data pouze do souborů až po takové, které umožňují ukládání do databáze a poskytují velké množství funkcí pro jejich zpracování. Syslog je realizován jako klient-server aplikace. Syslog daemon v roli serveru obstarává přijímací část. Zatímco aplikace odesílající logové záznamy je v roli klienta (viz obr. 3.2). Výhodou této architektury je, že jeden stroj může shromažďovat logové záznamy z mnoha různých počítačů. Pro komunikaci je použit syslog protokol.



Obrázek 3.2: Architektura Syslogu [5]

3.4 Syslog protokol

Dle syslog protokolu[6] musí být schopen přijímat zprávy o minimální délce 480-byte a měl by být schopen přijímat zprávy do délky 2048-byte. Delší zprávy jsou ořezávány do podporované délky, případně rovnou zahazovány. Délka zprávy 2048-byte může být pro logové záznamy nedostačující (zprávy mohou být podrobnější - například stacktrace záznam), a proto současné implementace syslogu dovolují nastavit maximální podporovanou délku zprávy. Pro odesílání je oficiálně podporován pouze UDP protokol a defaultní port pro komunikaci je 514. Současné implementace syslogu (Rsyslog, Syslog-ng) nabízejí podporu i pro TCP protokol.

3.5 Formát syslog zprávy

Formát syslog zprávy je popsán v definici syslog protokolu[6]. Syslogová zpráva se skládá ze 3 částí: hlavičky, strukturovaných dat a zprávy. Hlavička syslogové zprávy obsahuje PRI (Priority value - číselná hodnota vytvořená z kombinace hodnot zařízení a priority), verzi protokolu, zařízení, prioritu, datum, jméno počítače a aplikace (kde byla zpráva vytvořena), číslo procesu a zprávy. Strukturovaná data mohou obsahovat dodatečné informace o zprávě nebo specifické informace pro aplikaci. Zpráva obsahuje záznam, který poskytuje informace o události.

Pro přehlednost jsou v syslogu zprávy rozdělovány na základě zařízení a priority. Zařízení jsou rozdělena do 24 skupin a jednotlivé skupiny jsou použity pro specifikování typu aplikace, která záznamy pořizuje. Priorita udává význam události a existuje 7 různých skupin (od ladící zprávy po kritickou).

4 Zpracování logových záznamů

Logových záznamů existuje velké množství a každý má svůj specifický formát. Při přijmutí logového záznamu se obvykle jedná o zprávu bez struktury. Pro možnost analýzy a vyhledávání nad určitými poli je třeba získat ze zprávy zajímavé informace a ty uložit. Tím zpráva získá určitou strukturu, kterou lze pohodlně analyzovat a vyhledávat v ní konkrétní informace. Tento proces bude v textu dále označován jako parsování.

4.1 Problematika

Každý generátor logových záznamů zaznamenává informace, které sám považuje za důležité. Proto se obsah různých logových záznamů značně liší. Logové záznamy mohou být určeny pro člověka (čitelnost a přehlednost) a nebo pro počítače. Obvykle mají různou strukturu (prostý text, JSON, XML). Jako oddělovače jednotlivých polí se v rámci jedné zprávy (prostý text) používají čárky, mezery, různé symboly nebo jiné oddělovače znakových polí. I v případě zaznamenávání stejných hodnot můžou být reprezentované v různé podobě. Například datum může být uloženo ve formátech MMDDRRRR, MM-DD-YYYY, DD / MM / RRRR nebo jako časová značka (Unix timestamp - počet vteřin od 1.1.1970). V důsledku toho může být obtížné získat z různých logových záznamů potřebné informace. Pro možnost analýzy a vyhledávání dat ze zprávy (ve formátu prostý text) je potřeba dát datům určitou strukturu. Je tedy potřeba rozparsovat zprávu a získat z ní vhodné informace, čímž data získají určitý význam. Pro získání informací ze záznamu můžeme využít regulární výrazy.

4.2 Regulární výrazy

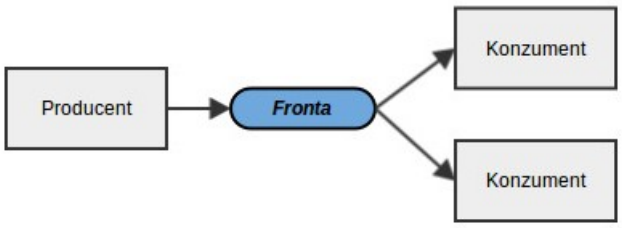
”Regulární výraz si můžeme představit jako speciální řetězec, který je šablonou vystihující určitý jazyk (tj. množinu textových řetězců). Každý textový řetězec takové šabloně buď vyhovuje nebo ne. S touto množinou vyhovujících řetězců můžeme manipulovat - hledat její prvky v textu nebo je nahrazovat jiným řetězcem”[7]. Díky regulárním výrazům[8] můžeme identifikovat typ logového záznamu a získat z něj informace. Doba provedení regulárních výrazů může být časově náročnější operací, v závislosti na délce textu a složitosti regulárních výrazů. To je třeba zohlednit v architektuře zpracování.

4.3 Architektura

Logových záznamů bude chodit velké množství a vyhodnocení regulárních výrazů může trvat delší dobu. Proto je nutné zvolit správnou architekturu zpracování záznamů. Logové záznamy jsou ze syslogu odesílány do úložiště, kde jsou připraveny k vyzvednutí a zpracování. Zpracování logových záznamů zabírá znatelně delší dobu než jejich vyzvednutí, a proto se nabízí použití známého návrhového vzoru producent/konzument.

4.3.1 Producent-konzument

Model producent-konzument[9] je vhodným modelem pro situace, kdy produkování dat je mnohem rychlejší než jejich zpracování. Tento model obsahuje dvě hlavní složky (produkování a zpracování), které jsou obvykle spojeny pomocí fronty viz obr.4.1. Producent vyprodukuje položku, která je umístěna do fronty (namísto okamžitého zpracování), ve které je připravena ke zpracování. Konzument vyzvedává položky z fronty a zpracovává je. Díky tomuto rozdělení se producent nemusí zajímat, jak bude která položka zpracována, kteří konzumenti jí budou zpracovávat nebo kolik existuje jiných producentů. Stejně tak konzumenti nemusí vědět, odkud položka pochází, kdo jí dal do fronty nebo kolik jiných producentů a konzumentů existuje. Jediné co musí udělat, je vzít položku z fronty a zpracovat ji. Toto rozdělení umožňuje mít více producentů obsluhovaných jedním konzumentem nebo více konzumentů zpracovávajících požadavky od jednoho producenta. Obecně to umožňuje měnit jednotlivé počty na základě potřeb výkonu nebo uživatelských požadavků.



Obrázek 4.1: Návrhový vzor producent konzument

4.3.2 Paralelismus

Paralelismus[10] je schopnost spouštět více programů nebo jeho částí souběžně. To znamená, že časově náročná část programu může být prováděna odděleně, nezávisle na ostatních a může být zpracovávána současně s dalšími. Díky tomu paralelismus umožňuje provést některé úlohy rychleji, například spuštěním náročných částí ve

více vláknech nebo procesech. Paralelní programování je známo delší dobu, ale až se současným rozvojem hardwaru se dostává do popředí. Hardware má dnes větší počet jader a využití těchto jader je prostředkem pro řešení úloh s náročným zpracováním a velkým objemem dat. Proto časově náročné parsování zpráv pomocí regulárních výrazů poběží paralelně.

Procesy a vlákna

Proces je instance běžícího programu. Probíhá nezávisle a izolovaně od jiných procesů. Nemůže proto přímo přistupovat ke sdíleným datům v jiných procesech. Vytvořit proces je obecně náročnější (na systémové zdroje) než vytvořit vlákno. Naproti tomu vlákno je tzv. lehký proces. Má svůj vlastní zásobník a také může přistupovat ke sdíleným datům jiných vláken v rámci stejného procesu.

Výhody vláken

- Vlákna minimalizují dobu potřebnou pro přepnutí kontextu
- Použití vláken umožňuje paralelismus v jednom procesu
- Efektivní komunikace mezi vlákny
- Ekonomické - je snazší vytvořit vlákno

4.3.3 Fronta

Pro komunikaci mezi procesy konzumentů a producentů musí existovat fronta. Do této fronty konzument ukládá data ke zpracování, zatímco konzument z ní data vybírá. Ve vyvíjené aplikaci musí fronta zajistit persistenci (trvalost) záznamů. Aby nedošlo k situaci, kdy producent připraví záznamy ke zpracování a uloží je do fronty, během čehož dojde k nečekanému ukončení aplikace, která způsobí ztrátu všech záznamů. Nebo k obdobné situaci při vybírání dat ke zpracování konzumentem. Proto jako fronta musí být použit nástroj, který data ukládá (například na disk) a jsou tak přístupná i po nečekaném ukončení nebo selhání. Zároveň dalším požadavkem je zvládnutí velkého provozu a zajištění synchronizované komunikace mezi více procesy (jeden záznam nezpracuje více konzumentů).

Na základě výše zmíněného jsou hlavními požadavky na frontu rychlost, persistence dat a možnost synchronizace komunikace. Prozkoumáním dostupných nástrojů byly vybrány Redis a MongoDB, neboť splňují zmíněné požadavky.

Redis

Redis[11] je open source klíč-hodnota databází, což znamená, že pod jeden klíč se uloží jedna hodnota. Pod klíč je možné uložit složitější hodnotu v libovolném formátu, například strukturu JSON. Redis data uchovává primárně v operační paměti a díky tomu je rychlý. Obsah paměti je dle konfigurace ukládán na disk a pokud dojde k nečekanému selhání (například výpadek proudu), záznamy nezmizí. Redis nabízí efektivní metodu předávání zpráv, která může být použita jako fronta pro komunikaci mezi procesy. Jedná se o datový typ list. List jsou jednoduché seznamy řetězců, řazené dle pořadí vložení. Nabízejí operace jako vybrání krajního prvku, vložení prvku na kraj, smazání prvku obsahujícího hodnotu, vybrání a vložení zároveň (atomická operace - je zaručeno provedení obou operací, díky čemuž lze zajistit persistenci dat).

MongoDB

MongoDB[12] je open source dokument databází. Dokument je ukládán ve formátu JSON. MongoDB data uchovává primárně v operační paměti, a proto je přístup k nim rychlý. Velikost operační paměti, kterou používá, je proměnlivá v závislosti na jejím celkovém využití ostatními procesy. Pokud potřebuje jiný proces více paměti, Mongodb ji začne využívat méně. Pro zajištění persistence dat jsou průběžně ukládána na disk. V MongoDB je jako frontu možné použít array, které nabízí operace pop a push. Nenabízí v základu operace jako vybrání a vložení prvku zároveň.

Výběr

Pro frontu musí být zajištěna persistence dat spolu se zajištěním atomicity výběru a vložení prvku. Atomicita znamená, že operace se provede najednou a nemůže být přerušena. Atomicitu zaručují obě databáze. Do fronty budou data ukládána jako hodnota, a proto je vhodné schéma klíč-hodnota. Redis má navíc naimplementovánu funkci výběr a vložení zároveň, která je pro frontu důležitou operací z důvodu zajištění spolehlivosti zpracování dat. Proto na základě výše zmíněného bude jako fronta použita Redis databáze.

5 Ukládání záznamů

Pro centralizované logování je třeba pečlivě zvážit úložiště logových záznamů. Je očekáváno velké množství záznamů. V tomto množství záznamů je pak nutné umožnit vyhledávání (nejlépe v reálném čase) a vytvářet pro jednotlivá pole statistiky. Dále je třeba najít vhodné úložiště pro uživatele a jejich data.

5.1 Zápis logových záznamů

V mnoha případech jsou logové záznamy zapisovány do textového souboru, což má několik výhod. Hlavní výhodou je jednoduchost zápisu do souboru, neboť stačí jen otevřít soubor a na jeho konec zprávu zapsat. Obvykle je také rychlost zápisu do souboru vyšší než do databáze a to především díky jednoduchosti zápisu do souboru. Oproti zápisu dat do databáze, který může být z důvodu transakcí a počítání indexů pomalejší. K zobrazení a analýze logových záznamů v reálném čase není optimální mít záznamy uloženy v souboru, protože obvykle jsou potřeba konkrétní data a procházet celý soubor není efektivní. Za pomoci databáze je možné přistupovat ke konkrétním datům rychle a pohodlně využitím vyhledávání, které může využít indexu atp. Uložení dat do databáze ale může trvat delší dobu v závislosti na množství dat a způsobu jejich ukládání. Proto existuje několik přístupů, jak jsou přijaté logové záznamy (syslogem) zapisovány do databáze[22]. Možné přístupy zápisu do databáze:

1. Logové záznamy zapisovat do souboru, kde jsou pak později programem zpracovány a odeslány do databáze
2. Logové záznamy jsou zapisovány do roury (roura je mechanismus pro interní komunikaci mezi procesy). Program sleduje rouru a jakmile se objeví logový záznam, zpracuje ho a uloží do databáze.
3. Využít možností syslogu a data rovnou zapisovat do databáze

K vybrání přístupu, jakým budou logové záznamy ukládány záleží na očekávaném provozu (tedy množství logových záznamů). Využití přímého zápisů logových záznamů rovnou do databáze je optimální možností v případě malého množství záznamů. Přímé uložení ale není pro tvořenou aplikaci vhodné, neboť data je potřeba nejdříve zpracovat (vybrat z nich vhodné informace) a také jich je očekáváno velké množství. Proto je vhodné použití prvního nebo druhého přístupu.

5.2 SQL vs NoSQL

SQL databáze jsou tabulkově založené, zatímco databáze NoSQL[14] jsou založené dokumentově. SQL databáze reprezentují data v podobě tabulek. Zatímco databáze NoSQL jsou kolekce dokumentů, dvojicí klíč-hodnota nebo grafů, které nemají standardní definice schématu, jež by museli dodržovat. Databáze SQL mají předdefinované schéma, zatímco databáze NoSQL mají dynamické schéma pro nestrukturovaná data. Toto je velkou výhodou NoSQL databází, neboť příchozí logové záznamy nemusí mít pevnou strukturu.

Databáze NoSQL[13] jsou navrženy tak, aby vynikaly v rychlosti. K dosažení tohoto cíle používají techniky, které nemusí zaručovat konzistenci dat, neboť nemusí podporovat transakce. Výměnou za slabší modely konzistence dat je vyšší rychlost zpracování. Pro zpracovávání logových záznamů nejsou transakce potřebnou vlastností, oproti vyšší rychlosti. Vysoká rychlost ukládání záznamů je v případě logových záznamů důležitou vlastností, především z důvodu jejich očekávaného velkého množství.

SQL databáze[14] jsou vertikálně škálovatelné. Lze zvládnout rostoucí zatížení zvýšením CPU, RAM, SSD, atd, na jednom serveru. Databáze NoSQL jsou horizontálně škálovatelné (to mohou být i NoSQL, ale za určitých omezení). Stačí jen přidat pár dalších serverů a databáze zvládnou větší provoz. To je velká výhoda NoSQL databází, neboť pořídit opravdu výkonný server je velmi drahou záležitostí, naproti pořízení několika průměrně výkonných serverů.

Uživatelská data

Pro potřeby aplikace je plánováno použití uživatelských účtů. Každý uživatel bude mít k dispozici osobní nastavení pro zobrazení a vizualizaci dat. Mezi uživatelem a jeho jednotlivými nastaveními budou vazby. Pro data, která jsou provázána a mají mezi sebou závislosti jsou optimálním řešením relační databáze. Relačních databázových systémů existuje značné množství, například MySQL, Postgresql, Oracle, Sqlite. Pro potřeby vyvíjené aplikace nejsou na uživatelská data kladeny speciální požadavky a není očekáváno jejich velké množství. Proto může být použita libovolná relační databáze.

Logové záznamy

Příchozí zpráva obvykle nebude mít strukturu a formát záznamů se může změnit. Proto je vhodné umožnit uložení zprávy bez definování specifického schématu. Příchozích zpráv je očekáváno velké množství, proto je třeba zajistit rychlé uložení a umožnit zpracování většího množství zpráv. Na základě dynamických schémat,

vysoké rychlosti (NoSQL umožňují rychlejší ukládání z důvodu slabších modelů konzistence) a snadné škálovatelnosti bude jako primární úložiště logových záznamů NoSQL databáze. Nad logovými záznamy je ale ještě potřeba umožnit vyhledávání a to nejlépe fulltextové.

5.3 Full-textové vyhledávání

Fulltextové vyhledávání je obecné označení pro vyhledávače fungující na základě porovnávání fráze se všemi ostatními slovy v daném dokumentu. Fulltext je tedy souhrn algoritmů, které dokáží z daného dokumentu vytvořit podrobnou statistiku výskytu jednotlivých pojmů a tu zanést do databáze. Při vyhledávání algoritmy fulltextu porovnávají hledané klíčové slovo s touto databází.

Prozkoumáním nejúspěšnějších konkurenčních nástrojů pro centralizované logování (Loggly, Logstash a Graylog) bylo zjištěno, že jako úložiště logových záznamu používají nástroj Elasticsearch. Proto byl tento nástroj zvolen pro porovnání. Jako další vhodný nástroj pro porovnání byl zvolen Solr. Oba nástroje jsou si podobné, neboť pro vyhledávání používají Apache Lucene (populární open source nástroj používaný pro fulltextové vyhledávání).

Solr

Solr[15] je open source nástroj napsaný v Javě, který běží v Apache Tomcatu nebo Jetty jako fulltextový server. Jádro vyhledávání a indexování tvoří softwarový projekt Lucene. Pro komunikaci používá REST architekturu HTTP/(XML, JSON, CSV), což jej činí snadno použitelným v mnoha programovacích jazycích. Disponuje rychlým fulltextovým vyhledáváním, poskytuje statistiky nad daty, indexování blízko reálnému času a clustering. Nevýhodou je schéma, které není dynamické a musí být definováno před uložením záznamů.

Elasticsearch

Elasticsearch[16] je open source nástroj napsaný v Javě, který běží jako daemon. Všechno v Elasticsearch, co se týká algoritmů pro porovnávání odpovídajících textů a ukládání optimalizovaných indexů vyhledávacích podmínek je realizováno nástrojem Lucene. Elasticsearch sám poskytuje snadněji použitelné a stručné API, škálovatelnost a nástroje pro práci s Lucene. Pro komunikaci používá REST HTTP/JSON, což jej činí snadno použitelným. Elasticsearch umožňuje rychlé fulltextové vyhledávání, poskytuje statistiky nad daty, indexování blízko reálnému času, clustering a disponuje dynamickými schématy.

Výběr

Oba nástroje poskytují požadované funkce jako fulltextové vyhledávání, statistiky nad daty, snadnou komunikaci, škálovatelnost. Zásadním rozdílem jsou dynamická schémata při vytváření, která podporuje pouze Elasticsearch. Dynamické schéma může být vhodné, neboť nemusí být při vytváření záznamů definovány datové typy, ale jsou rozpoznávány na základě vkládané hodnoty. Z tohoto důvodu bude jako úložiště logových záznamů zvolen Elasticsearch.

6 Shlukování

Shlukování dat je jakýkoliv proces, ve kterém jsou informace shromažďovány a následně vyjádřeny v souhrnné podobě. Společný cíl agregace je získat více informací o jednotlivých skupinách na základě specifických proměnných, jako je třeba priorita, zařízení nebo zpráva.

Tento proces je velmi vhodný pro logování, neboť umožňuje získat přehledné výsledky k analyzování. Shlukování lze provést na základě zpráv, ty se totiž obvykle nemění (až na různé typy logových záznamů, jako například přístupový logový záznam, kde je vždy různá doba vyřízení požadavku a návratový kód). Jediné co je obvykle různé je zaznamenaný čas přijetí zprávy. Proto je vhodné ukládat totožné zprávy přijaté bezprostředně za sebou s určitým příznakem spolu s počtem výskytů a dobou poslední přijaté zprávy. Příznak pak určuje, zda se jedná o více zpráv v řadě. Tímto přístupem lze značně zpřehlednit zobrazení a snadněji tak získat přehled o datech. Pokud by totiž na serveru nastala chyba a ten by při každém požadavku generoval výjimku, kterou by následně odeslal ke zpracování do centrálního úložiště. Při vyhledávání v logových záznamech by se objevovala dokola jedna totožná zpráva a znepřehledňovala by vyhledávání. Shlukování umožňuje uvést u zprávy počet výskytů spolu s časem přijetí první a poslední zprávy, což podá uživateli důležité a přehledné informace, oproti například 1000 stejným zprávám zobrazených v řadě.

Shlukování je také vhodné pro rozdělení jednotlivých logových záznamů dle použitého parseru a druhu logového záznamu. To je vhodné převážně z důvodu, že každý typ parseru získává z logových záznamů různé informace a ukládá různá pole. Pro zobrazení dat je ovšem třeba znát jednotlivé položky, které data obsahují. Dává také smysl mít záznamy rozděleny kvůli vyhledávání. Obvykle je totiž vyhledáván konkrétní typ logového záznamu. Například pro získání přehledu o přístupech k jednotlivým stránkám pak stačí zobrazit pouze přístupové logové záznamy.

7 Webová aplikace

Pro efektivní práci s logovými záznamy je nutné mít možnost nad nimi vyhledávat a zobrazovat je. K tomu bude sloužit webová aplikace, která bude realizována ve webovém frameworku Ruby On Rails. Primární roli ve výběru hrály zkušenosti autora s vývojem aplikací v tomto frameworku. Webová aplikace tedy bude naprogramována v programovacím jazyce Ruby a jazyk Ruby bude použit pro programování všech částí aplikace.

7.1 Ruby on Rails

Ruby on Rails[17] (dále RoR) je framework pro vývoj webových aplikací napsaný v jazyce Ruby. Používá architekturu Model-View-Controller (MVC), která bude rozebrána v sekci 7.2. Je navržen tak, aby programování webových aplikací dělal jednodušším. Filozofie RoR zahrnuje dva hlavní principy:

- Neopakujte sami sebe: DRY je princip vývoje softwaru, v němž je uvedeno: *"Každý kus znalostí musí mít jedno, jednoznačné, autoritativní zastoupení v rámci systému."* Tím, že není psán stejný kód znovu a znovu, je kód lépe spravovatelný, více rozšířitelný a s menší chybovostí.
- Konvence má přednost před konfigurací: RoR přichází s konvencí, kterou se řídí webová aplikace a programátor konfiguruje pouze ty části aplikace, které se liší od běžného nastavení.

7.2 Návrhový vzor MVC

MVC[18] je ustálený návrhový vzor v softwarovém vývoji. Umožňuje transparentně oddělit různé části aplikace podle toho, jakou úlohu v ní hrají. Tuto architekturu používá webový framework RoR viz obr.7.1.

Model

Model je zodpovědný za práci s daty. Zprostředkovává především přístup k datům z datového úložiště, může také řešit autentizaci uživateli nebo provádět kontrolu vstupních dat. RoR pro snadnější práci s databází přichází s vlastním Object-Relational Mapping.

View

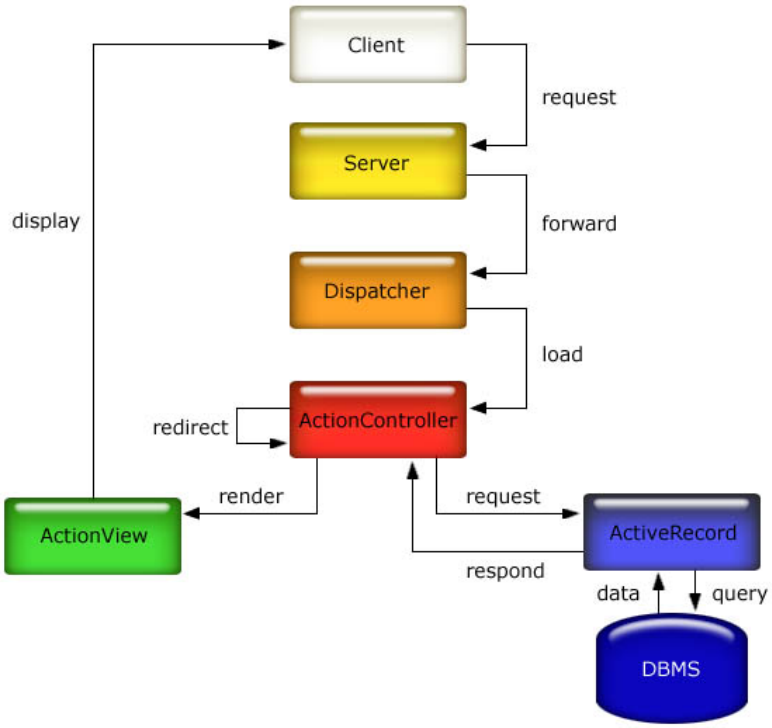
View je zodpovědný za generování zobrazení uživateli a to na základě údajů předaných controllerem. Podporuje zobrazení dat do různých formátů, jako HTML, XML, JSON.

Controller

Controller propojuje model s view. Pro příchozí požadavek zpracuje data s pomocí modelu a výsledky zobrazí ve view, které je následně zobrazeno uživateli.

Helper

Helper je rozšíření, s kterým přichází RoR oproti tradičnímu vzoru navíc. Slouží pro vytvoření přehlednější štábní kultury. Jedná se o část pro kód, který je často používaný ve view nebo je příliš dlouhý / špinavý, aby byl uvnitř view.



Obrázek 7.1: Architektura Ruby on Rails [19]

8 Vizualizace

Vizualizace dat je velmi důležitý obor, který je v poslední době na vzestupu. Je to způsobeno obrovským množstvím dat, které dnes máme k dispozici a které se neustále zvětšuje. Chceme-li se v takovém množství dat vyznat, je nutné tato data zobrazit v přehledné formě.

8.1 Vizualizace pomocí grafů

Zobrazením v tabulkách lze získat o datech přehled, ale obvykle je nutné detailnější zkoumání. Tabulky také mohou být velké a nepřehledné. Možným prostředkem pro přehledné zobrazení statistik je vizualizace dat grafem. Z grafů lze na první pohled získat přehled o datech. Při volbě vhodného grafu jsou na první pohled vidět odchylky, průměrné hodnoty anebo nejvyšší výskyt konkrétních údajů (například nejvíce logových záznamů od konkrétního serveru). Při vizualizaci je nutné zvolit vhodný typ grafu v závislosti na datech. Například při minimálních odchylkách dat je nevhodné zvolit koláčový graf, neboť z velikostí jednotlivých výsečí není na první pohled nic vidět a data lze porovnat pouze přímým porovnáním hodnot u popisků. Daleko vhodnější je použít sloupcový nebo spojnicový graf, kde na první pohled můžeme porovnat jednotlivé hodnoty, zjistit minimum, maximum atd.

8.2 Vykreslení grafů

Vykreslit grafy může být výpočetně složité a uživatel by měl mít možnost je upravovat (například skrýt zobrazení určitých dat). Z těchto důvodu by mělo být kreslení grafu realizováno na straně uživatele. To je možné použitím Javascriptu. Pro vykreslení grafů jsou v Javascriptu dostupné knihovny, kterým stačí vyplnit potřebné údaje a předat data k vykreslení. Těchto knihoven existuje velké množství. Pro porovnání byly vybrány knihovny Highcharts a jqPlot. Požadavkem na výsledné grafy je poskytnutí základních typů grafů (výsečové, sloupcové a spojnicové) pro možnost přehledné vizualizace rozdílných dat a umožnit interaktivitu s uživatelem. Dalším kritériem pro posouzení je výsledný vzhled.

Highcharts

Highcharts[20] je knihovna pro kreslení grafů napsaná v JavaScriptu. Nabízí snadný způsob přidání interaktivních grafů do webové aplikace. Stačí vytvořit kontejner pro graf, nastavit možnosti a předat data k vykreslení. Highcharts v současné době

podporuje mnoho typů grafů, například plošný, spojnicový, sloupcový, výsečový (koláčový), bodový a další. Jedná se o produkt, který věnuje značnou pozornost detailům. Produkt je placený, ale pro nekomerční použití je zdarma.

JqPlot

JqPlot[21] je knihovna pro kreslení grafů napsaná v JavaScriptu. Nabízí podobně jako Highcharts snadnou možnost přidání grafů. Je nutné vytvořit kontejner, do kterého bude graf vykreslen. Poté stačí vybrat typ grafu a předat data, přičemž lze specifikovat dodatečné možnosti. Podporuje základní skupiny grafů jako jsou výsečové, sloupcové, bublinové, spojnicové. Produkt je open source software.

Výběr

Více typů grafů poskytuje Highcharts, ale JqPlot poskytuje všechny standardní grafy, které jsou pro vizualizaci dat dostačující. Oba dva také umožňují interaktivitu uživatele s grafy. Hlavním posuzovacím kritériem je tedy vzhled grafů. Po grafické stránce vypadají lépe grafy vykreslené Highcharts. Z tohoto důvodu bude použita knihovna Highcharts.

9 Odesílání logových záznamů

Logové záznamy je nutné vygenerovat a odeslat do kolektoru logových záznamů. K dispozici jsou dvě varianty řešení. Aplikace obsahuje svůj vlastní logovací nástroj, který nakonfigurujeme pro odesílání do syslogu. Nebo do aplikace přidáme framework pro logování, který nakonfigurujeme stejným způsobem. Po nakonfigurování budeme schopni centralizovaně přijímat logové záznamy z různých aplikací. Níže bude předvedena konfigurace vloženého frameworku pro Javu a Ruby (pro ukázku bude předvedena pouze jednoho frameworku - další budou přiloženy v CD). Všechny ukázky v rámci práce jsou předváděny pro Linux a byly testovány na distribuci Ubuntu.

9.1 Konfigurace pro Javu

Níže je uvedena konfigurace pro framework Logback, který využívá SLF4j API. Díky použití SLF4j lze snadno framework vyměnit za jiný, stačí pouze zaměnit jar soubor. Do projektu je nutné přidat následující jar soubory: *slf4j-api-1.7.5.jar, logback-core-1.0.13.jar, logback-classic-1.0.13.jar*. Poté stačí importovat slf4j do projektu a vytvořit instanci loggeru. Následně již můžeme ukládat logové záznamy použitím *logger.error("error", e)*, viz ukázka zdr.9.1.

```
final static Logger logger = LoggerFactory.getLogger(Logback.class);

private void divide_by_zero() {
  try {
    int a = 5 / 0 ;
  } catch(Exception e) {
      logger.error("application",e);
  }
}
```

Zdr. 9.1: Použití Logback frameworku

Standardně je výstup zapisován pouze do konzole. Proto musíme provést nastavení v konfiguračním souboru. Konfigurační soubor je třeba uložit do složky *bin* příslušného projektu. Konfigurační soubor je ve formátu xml a musí být pojmenován *logback.xml*. V souboru je nutné provést následující konfiguraci:

```
<configuration>
  <appender name="SYSLOG" class="ch.qos.logback.
  classic.net.SyslogAppender">
    <Facility>LOCAL1</Facility>
    <syslogHost>127.0.0.1</syslogHost>
    <SuffixPattern>%msg %throwable</SuffixPattern>
    <ThrowableExcluded>true</ThrowableExcluded>
  </appender>
  <root level="INFO">
    <appender-ref ref="SYSLOG"/>
    <appender-ref ref="FILE"/>
```

```
  </root>
</configuration>
```

Zdr. 9.2: Konfigurace Logback frameworku

Konfigurace je uvedena pro lokální logování. V případě vzdáleného serveru je nutné syslogHost nahradit ip adresou nebo hostname serveru.

9.2 Konfigurace pro Ruby

Vytvářená webová aplikace v RoR generuje logové záznamy a proto bude předvedeno, jak ji nakonfigurovat pro odesílání záznamů do syslogu. Nejsnazší nastavení pro odesílání dat do syslogu poskytuje framework Remote syslog logger, a proto bude v rámci ukázky předvedena jeho konfigurace. Je nutné přidat následující řádek do *Gemfile* souboru, jenž se nachází v kořenovém adresáři webové aplikace:

```
gem 'remote_syslog_logger'
```

Dále je třeba spustit příkaz v kořenovém adresáři webové aplikace, který provede instalaci gemů ze souboru *Gemfile* (zároveň i logovacího frameworku):

```
bundle install
```

Poté ve složce *config/environments* přidat do *development.rb* nebo *production.rb* (v závislosti na spouštěném prostředí při startu webové aplikace) následující:

```
config.logger = RemoteSyslogLogger.new('localhost', 514, :program => "rails")
config.logger.level = Logger::INFO
config.log_formatter = ::Logger::Formatter.new
```

Zdr. 9.3: Konfigurace Remote syslog logger frameworku

Konfigurace je uvedena pro lokální logování. V případě vzdáleného serveru je nutné 'localhost' nahradit ip adresou nebo hostname serveru. Poté stačí restartovat server a logové záznamy jsou odesílány do syslogu.

10 Kolektor logových záznamů na serveru

Pro příjímání logových záznamů slouží syslog. Níže bude zvolena implementace syslogu, pro kterou bude předvedena konfigurace a budou rozebrány detaily sběru.

10.1 Porovnání syslog nástrojů

V současné době existuje mnoho implementací syslogu, ale za nejkvalitnější a nejčastěji používané jsou označovány dva. Prvním je Rsyslog[22] a druhým Syslos-ng[23]. Proto bude provedeno porovnání základních funkcí (důležité je především zapsání záznamů do roury) pouze pro tyto dva viz tabulka 10.1 a na základě výsledků bude vybrán ten, který bude lépe vyhovovat požadavkům.

Vlastnosti	**RSyslog**	**Syslog-ng**
Protokol UDP	Ano	Ano
Filtrování podle úrovně a zařízení	Ano	Ano
Filtrování podle IP adresy	Ano	Ano
Zápis do roury	Ano	Ano

Tabulka 10.1: Porovnání implementací syslogu

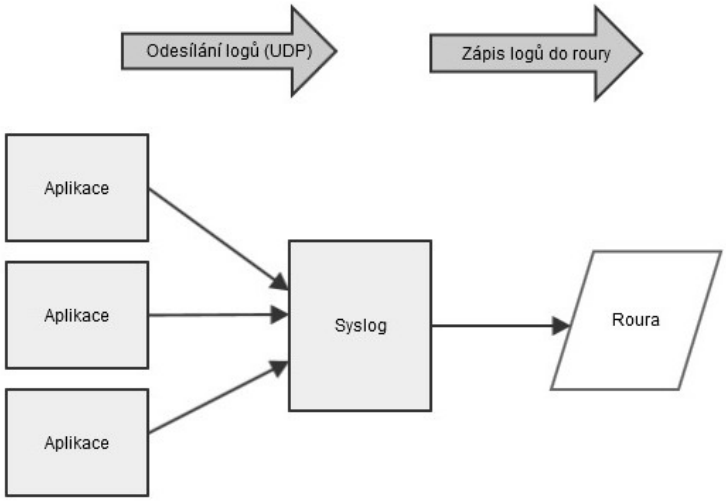
Zhodnocení výsledků

Oba nástroje jsou kvalitní a dostačující pro sběr logových záznamů. Z důvodů možného budoucího rozšíření v podobě zajištění doručení zpráv (které v současné době není z důvodu použití UDP protokolu) se jeví RSyslog jako vhodnější volba. Bylo tak učiněno s přihlédnutím k podpoře protokolu RELP. RELP zaručuje doručení zpráv, které nemusí být doručeny ani za pomoci TCP protokolu. Důvodem volby Rsyslogu je také jeho rozšířenost, neboť je v základu nainstalován na mnoha Linuxových distribucích. Proto bude aplikace testována na RSyslogu a bude pro něj předvedena konfigurace.

10.2 Architektura sběru

Logové záznamy jsou z jednotlivých serverů odesílány pomocí UDP protokolu do syslogu, který jednotlivé záznamy zpracuje a na základě předvolených filtrů je odešle na zvolené výstupy viz obr. 10.1. Pro následné zpracování jsou ve vyvíjené apli-

kaci ukládány do roury. Roura je zvolena proto, neboť poskytuje snadný a často používaný mechanismus výměny dat mezi procesy. Použití roury umožňuje změnu kolektoru logových záznamů za jiný a to bez nutnosti provedení změn v architektuře parsování zpráv, protože pro parsery jsou důležité pouze zprávy v rouře a už nezáleží kdo je tam uložil.



Obrázek 10.1: Sběr logových záznamů Syslogem

10.3 Filtrování příchozích záznamů

Příchozí záznamy je možné filtrovat na základě nastavení. Níže budou probrány filtry, které poskytuje RSyslog.

Filtry RainerScript jsou hlavními prostředky vytváření konfigurací pro Rsyslog[22]. Umožňují filtrování nad libovolnými komplexními výrazy, které mohou zahrnovat pravdivostní, aritmetické a řetězcové operace. Mohou být použity k odeslání zprávy na výstup při splnění požadované podmínky.

Selektory jsou tradiční způsob filtrování syslogových zpráv. Tyto filtry jsou vhodnou volbou, pokud je třeba filtrovat na základě priority nebo zařízení (poskytují nejrychlejší zpracování).

Filtry na vybraná pole umožňují filtrovat nad poli, které Rsyslog získá ze zprávy při jejím přijetí. Jedná se například o jméno aplikace, syslogtag, zprávu, ip adresu atd. Jsou optimálním prostředkem, pokud je potřeba povolit zpracování zprávy pouze určitým aplikacím nebo serveru.

10.4 Problém s více řádkovými zprávami

Syslog považuje každou přijatou zprávu za nový logový záznam. To působí problémy při příjímání více řádkových zpráv (například stacktrace logových záznamů). Toto nelze vyřešit na straně syslogu. Problém je složitý a jediné přijatelné řešení lze provést na straně odesílatele logových záznamů. Je nutné nastavit, aby se výjimky odesílaly v jednom řádku. To je u většiny frameworků pro logování umožněno.

10.5 Konfigurace RSyslogu

Pro zpracování příchozích logových záznamů musí být uloženy do roury. Z roury jsou záznamy vyzvedávány k parsování a indexovány do databáze. Níže je uvedena konfigurace, která všechny příchozí záznamy zapisuje do roury.

V případě že není roura vytvořena, je nutné ji nejprve vytvořit pomocí následujícího příkazu(jmeno - jméno vytvářené roury):

```
mkfifo jmeno
```

Konfigurace pro Rsyslog se nachází v souboru */etc/rsyslog.conf*. Do tohoto souboru stačí přidat následující řádky.

Pro povolení příjímání zpráv pomocí UDP protokolu a nastavení maximální podporované délky zpráv (je zvolena maximální nastavitelná délka zprávy v RSyslogu):

```
$ModLoad imudp
$UDPServerRun 514
$MaxMessageSize 32k
```

Zdr. 10.1: Konfigurace pro UDP a maximální délku zprávy

Ukládání všech příchozích zpráv do roury dle definované šablony (CESTA musí být nahrazena umístěním, ve kterém se nachází roura, pipe - musí být nahrazena jménem, kterým je pojmenována vytvořená roura):

```
\$template MessageFormat,"%syslogtag% %syslogseverity% %syslogfacility% %timestamp
    :::date-rfc3339% %fromhost% %fromhost-ip% %msg% \n"

*.* |CESTA/pipe;MessageFormat
```

Zdr. 10.2: Konfigurace šablony

Pro načtení aktuální konfigurace je třeba v konzoli restartovat Rsyslog:

```
sudo service rsyslog restart
```

11 Databáze

Každý typ databáze se hodí pro různá data. Pro složitější aplikace je pak optimální zvolit kombinaci více databází, v závislosti na typu dat.

11.1 Elasticsearch

Níže bude popsáno filtrování záznamů v Elasticsearch a bude předvedena architektura ukládání záznamů. Pro práci v Ruby s Elasticsearch je k dispozici gem Tire, který bude popsán níže.

11.1.1 Filtrování zobrazovaných výsledků

Pro přehledné zobrazení a možnost efektivní práce s logovými záznamy je nutné umožnit komplexní filtrování záznamů. Elasticsearch[16] poskytuje mnoho druhů dotazů a filtrů, které umožní získat požadované záznamy. Níže jsou uvedeny ty, které jsou pro potřeby aplikace dostačující.

boolean query - umožňuje provést dotaz, pro který musí být splněny všechny (nebo některé v závislosti na nastavení) uvedené podmínky. Lze díky tomu vyfiltrovat velmi specifické záznamy. Například HTTP požadavky s návratovými kódy 200, jejichž celková doba zpracování byla vyšší než 1000ms a zároveň doba strávená databázovými operacemi byla menší než 300ms.

query - Při vyhledávání lze použít nad zvolenými poli regulární výrazy. Je umožněno i použití wildcard (tzv. divokých karet), které umožňují nalézt část textu s použitím * výrazu. To znamená, že například při použití výrazu *test** lze najít řetězec testování, nebo použitím *zk*ka* najít řetězec zkouška. Například pro analýzu zobrazovaných stránek je možné určit začátek cesty jako */log** a sledovat množství jednotlivých zobrazení stránek obsahujících *log*. Ve výrazech je umožněno používat i negace, tudíž je umožněno určit, které záznamy naopak vidět nechceme. Nad numerickými poli lze určovat rozsahy hodnot, které mají být zobrazeny. To může být užitečné například pro zobrazení požadavků, jejichž doba vyřízení je větší než 500ms.

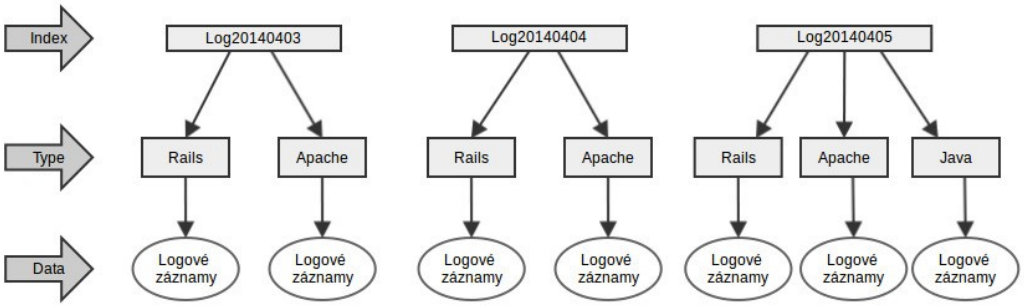
terms filter - Vrací dokumenty, které odpovídají jedné z povolených hodnot. Je definováno pole, nad kterým má být filtrování provedeno a jsou uvedeny všechny hodnoty, které jsou přípustné. Za pomoci těchto filtrů lze například získat všechny záznamy z určených aplikací a určených priorit.

range filter - Vrací dokumenty v závislosti na zvoleném rozsahu. Umožňuje fil-

trování dle datového pole a vrátit výsledky v rozmezí dvou dat. To je opět velmi užitečné, neboť nás často zajímají záznamy z konkrétního období.

11.1.2 Ukládání záznamů v Elasticsearch

Vzhledem k očekávanému velkému množství logových záznamů je nutné zvolit vhodné schéma ukládání. Ukládat logové záznamy dle určitého shlukování (například dle typu logového záznamu) a umožnit tak optimalizování vyhledávání. Dále umožnit snadné odstranění starých logových záznamů (v Elasticsearch lze smazat vybrané záznamy příkazem, viz přiložená dokumentace). Proto je při ukládání záznamů do Elasticsearch každý den vytvářen nový index (index je v abstrakci relačních databázích 'databáze') se zvoleným jménem (Log) a následovaný časovou značkou (ve formátu rok měsíc den a to bez mezer) viz obr.11.1. Type (type je v abstrakci relačních databázích 'tabulka') je pro index vytvořen dle typu parseru, kterým byl logový záznam zpracován. Je tak zvoleno v souladu s architekturou vyhledávání, kdy jsou data vyhledávána vždy pouze pro určitý typ parseru a logového záznamu.



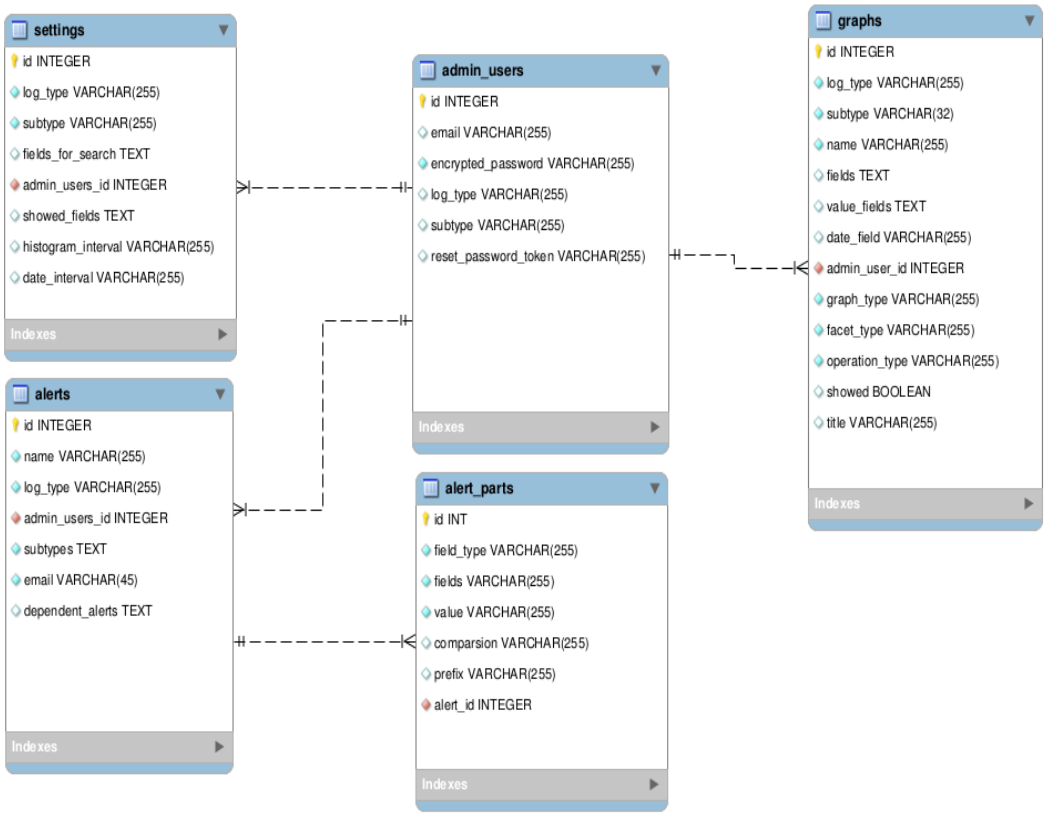
Obrázek 11.1: Schéma ukládání v Elasticsearch

Tire

Pro zápis logových záznamů do Elasticsearch existuje v Ruby gem Tire. Tire je Ruby klient umožňující pohodlnou práci s daty v Elasticsearch a jedná se o nejpoužívanější gem pro práci s Elasticsearch v Ruby. Byl použit především z důvodu velkého množství kvalitní dokumentace, kterou ostatní nástroje nedisponovali. Obsahuje většinu funkcí, které Elasticsearch implementuje a obsahuje všechny funkce, které jsou potřebné pro vyvíjenou aplikaci.

11.2 Relační databáze

Webová aplikace obsahuje uživatelská data, pro která je optimální využít tradiční úložiště v podobě relačních databází. Jsou to data o jednotlivých uživatelích, jejich nastavení pro jednotlivé logové záznamy, vytvořené grafy a přednastavená upozornění (viz schéma obr. 11.2).



Obrázek 11.2: Schéma relační databáze

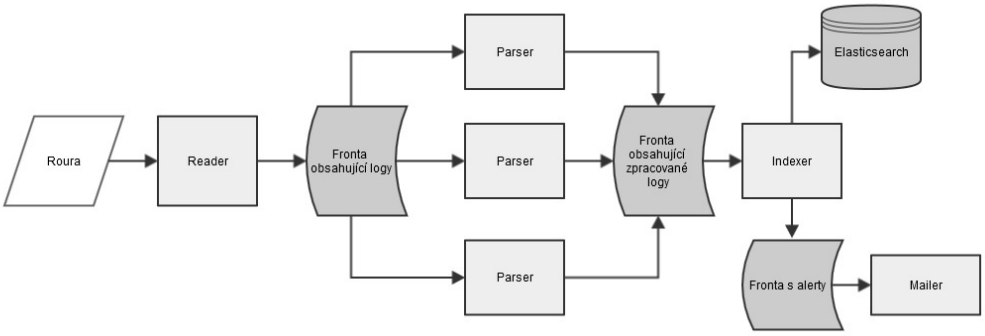
Tabulka Admin user slouží k uložení informací o uživateli a umožňuje vytvořit webovou aplikaci s přihlašováním. Uživatel má pro konkrétní parser a typ logového záznamu vytvořenu tabulku settings. Ta obsahuje seznam polí, které mají být zobrazeny pro vyhledávání a zobrazení v tabulce. Dále má nastaven interval pro zobrazení dat v čase a interval pro zobrazení rozsahu (tyto položky jsou využívány při zobrazování grafů). Uživatel může mít definován libovolný počet grafů pro konkrétní typ logového záznamu a může specifikovat pro který subtyp má být graf viditelný. Také může vytvořit veřejný graf, který je přístupný pro všechny uživatele. Nadefinované grafy slouží k vizualizaci logových záznamů. Tabulka Alert slouží k nastavení upozornění na události, které se definují pro konkrétní parser a typ logu. Každý alert může obsahovat několik alert parts. Alert parts určují podmínky, za kterých má být upozornění aktivováno a je mezi nimi závislost and.

12 Parsování logových záznamů

Příchozí zpráva je obvykle pouze text bez struktury. Pro pohodlné analyzování a vyhledávání je potřeba dát datům strukturu. V aplikaci jsou data strukturována za pomoci regulárních výrazů a to na základě znalosti jednotlivých formátů logových záznamů. Pro příchozí zprávu jsou na základě jména aplikace, priority a zdroje zprávy vybrány parsery.

12.1 Architektura

Reader sleduje rouru a jakmile se objeví zpráva, uloží ji do fronty. Z fronty si jednotlivá vlákna (Parser) vybírají zprávy, zpracovávají je a pokud proběhne shoda s regulárním výrazem, tak uloží strukturovanou zprávu do fronty pro následné zpracování indexerem. Indexer provede kontrolu pro nastavená upozornění a v případě pozitivního testu na upozornění uloží tuto zprávu do fronty, která je sledována procesem pro odesílání mailů. Indexer shromažďuje zprávy a při dosažení určitého počtu zpráv nebo po uplynutí časového limitu všechny zprávy hromadně odešle do Elasticsearch, přičemž provádí grupování stejných zpráv.



Obrázek 12.1: Architektura zpracování zprávy

Indexer je využit kvůli hromadnému ukládání zpráv, neboť odesílání každé zprávy k indexování zvlášť by bylo značně neefektivní. Jednak by zbytečně byla vytěžována síť a hromadné ukládání je obvykle optimalizované, čímž je znatelně rychlejší. Problémem je ale rychlost zpracování zprávy pomocí regulárních výrazů a doba odeslání upozornění uživateli emailem. Proto se parsování logových záznamů provádí ve více vláknech, díky čemuž mohou probíhat současně (rychleji na více jádrových procesorech). Upozornění emailem jsou odesílána asynchronně a to kvůli dlouhé době trvání odesílání emailu.

Vlákno reader je v roli producenta, zatímco několik dalších vláken (Parser) v roli konzumenta. Problémy spojené s paralelním zpracováním jsou vyřešeny za po-

moci Redis listu, který se stará o zajištění synchronizované komunikace mezi vlákny a o persistenci dat.

12.2 Paralelismus v Ruby

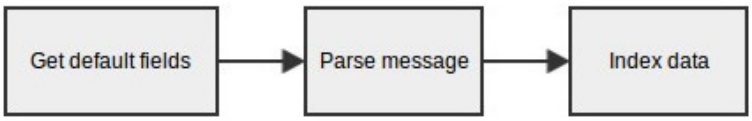
Na základě kapitoly 4.3.2 byly pro paralelismus zvoleny vlákna a to na základě uvedených výhod. Problémem je ovšem paralelismus[24] v Ruby, kdy v jeden čas může běžet pouze jedno vlákno a nejedná se tak o skutečné paralelní zpracování. Je to způsobeno instancí Global Interpreter Lock (GIL), která je vytvořena při spouštění procesu. Tato instance neumožňuje běh více než jednoho vlákna v jeden okamžik. Tento problém lze vyřešit použitím JRuby. JRuby je implementace Ruby, která umožňuje škálovat Ruby kód napříč více procesorovými jádry. Je to umožněno zkompilováním Ruby do bytecode a spuštěním na Java virtual machine. Vlákna jsou pak spouštěna bez GIL a umožňují skutečné paralelní zpracování.

12.3 Reader

Reader má na starosti spuštění všech potřebných částí pro zpracování logových záznamů. Při startu načítá konfigurační soubor, ve kterém jsou uvedeny všechny používané parsery. Pro každý parser je uvedeno jeho jméno, soubor ve kterém je k nalezení, podmínky za kterých má být použit a typ parseru. Na základě tohoto nastavení jsou data uložena do připravených struktur a vytvořeny instance jednotlivých parserů. V závislosti na nastavení je spuštěn počet zvolených vláken, která se starají o parsování jednotlivých zpráv, dále vlákno pro indexování a pro odesílání emailů. Vlákna v nekonečné smyčce čekají na příchozí zprávy, které po přijetí zpracovávají. Zprávy ke zpracování získávají pomocí fronty. Hlavní vlákno readeru dále otevře rouru a v nekonečné smyčce čte příchozí logové záznamy, které ukládá do fronty.

12.4 Parser

Parser má na starosti zpracování záznamu a získání strukturovaných dat. Zpracování zprávy je rozděleno do tří procedur viz obr.12.2. Jednotlivé procedury jsou detailně popsány níže.



Obrázek 12.2: Průběh zpracování zprávy

Get default fields

Na základě definované šablony v Rsyslogu je známa struktura zprávy:

```
[syslogtag] [syslogseverity] [syslogfacility] [timestamp:::date-rfc3339]
[fromhost] [fromhost-ip] [msg]
```

Zdr. 12.1: Šablona příchozí zprávy z Rsyslogu

Metoda získá hodnoty jednotlivých polí, které uloží do listu. Z jména aplikace jsou odstraněny koncové závorky (značící pid) a dvojtečky na konci, kvůli přehlednějšímu filtrování. Výsledný list vypadá vzhledem k výše zmíněné definici:

```
list = {
  "app_name" => [syslogtag],
  "facility" => [syslogfacility],
  "level" => [syslogseverity],
  "timestamp" => [timestamp:::date-rfc3339],
  "host" => [fromhost],
  "ip_address" => [fromhost-ip],
  "message" => [msg]
}
```

Zdr. 12.2: List se získanými hodnotami

Parse message

Tomuto zpracování je podrobeno pouze pole message. Na základě regulárních výrazů je nalezen začátek zprávy a je testována na shodu s daným formátem. Pro lepší pochopení si ukážeme ukázku v podobě zpracování přístupového logového záznamu pro RoR. Přístupový logový záznam pro RoR může vypadat následovně:

Started POST "/logs"for 194.55.44.15 at Wed Jan 22 15:24:28 +0200 2014

Pro RoR existuje více typů logových záznamů v závislosti na akci a každý začíná specifickým řetězcem. Aplikace proto provede následující regulární výraz:

```
(Started|Processing|Parameters|Completed|ActionController|ActionView|Rendered.*)
```

Zdr. 12.3: Regulární výraz pro identifikování RoR záznamu

Pomocí tohoto výrazu zjistí, zda se může jednat o RoR logový záznam. V případě neúspěchu je parsování ukončeno. V případě shody vrátí začátek záznamu (od shody s regulárním výrazem) spolu se zbytkem zprávy. Poté je testováno první slovo, které určí o jaký konkrétní typ se jedná (přístupový, výkonnostní, atd.). Na základě typu je proveden další regulární výraz, který zjistí zda se skutečně jedná o kompletní záznam:

```
/(?<method_type>\bStarted) (?<method>#{method}) "(?<path>[^"]+)" for(?<
    req_ip_adress>#{ip_address}) at (?<time>#{timestamp('%a %b %d %H:%M:%S %z %Y')
    }|#{timestamp('%Y-%m-%d %H:%M:%S %z')})/
```

Zdr. 12.4: Regulární výraz pro identifikování RoR záznamu typu přístupový

V regulárním výrazu jsou použity metody z napsaného modulu pro ulehčení psaní regulárních výrazů, který je součástí třídy (dědí se při implementaci parseru). Například metoda ip_address je tak nahrazena komplexním regulárním výrazem pro obsáhnutí všech typů ip adres. Pokud proběhne shoda s regulárním výrazem, jsou ze zprávy získány požadované informace:

```
"fields": {
  "method_type":"Started",
  "method":"POST",
  "path":"/logs",
  "req_ip_adress":"194.55.44.15",
  "time":"Wed Jan 22 15:24:28 +0100 2014"
}
```

Zdr. 12.5: List se získanými hodnotami z logového záznamu

Ke zprávě je uložen typ parseru, kterým byla zpracována a typ logového záznamu. V tomto případě je pro typ logového záznamu uloženo přístupový a pro typ parseru rails. Nad získanými daty je již možné provádět komplexní filtrování, zobrazit je v grafech, zobrazit konkrétní informace atd.

Index data

V případě úspěšného zpracování zprávy, kdy jsou získány ze zprávy informace se všechna získaná data uloží do listu. Tento list obsahuje základní pole získaná metodou Get_default _fields a pole získaná metodou Parse_message a je ukládán do fronty pro indexování. Může být určeno, která pole se nemají ukládat, například není vždy nutné ukládat pole message. Ve frontě zpracované záznamy vyčkávají na vyzvednutí indexerem a uložení do Elasticsearch.

12.4.1 Implementace parserů

Pro implementaci parserů je využito dědění. Je připravena třída v roli rodiče, která obsahuje všechny potřebné metody pro zpracování a modul pro ulehčení psaní re-

gulárních výrazů. Napsání parseru pro nový typ logového záznamu je tak relativně snadné. Děděním od třídy rodiče získá všechny potřebné metody a atributy. Pak již stačí jen překrýt metodu Parse message vlastní implementací regulárních výrazů pro daný formát logového záznamu. A také překrýt metodu na vytvoření mapování pro Elasticsearch. Je ovšem možné parser upravit zcela libovolně a tak například ani neprovádět parsování pole message. Níže jsou předvedeny a popsány části obecného parseru (obecný parser zpracováva RoR logové záznamy - pro lepší pochopení parsovacího procesu) od kterého jednotlivé implementace parserů dědí.

Pro zpracování zprávy je volána metoda process_message(message), která příchozí zprávu provede jednotlivými procedurami (Get default fields, Parse message, Index data) a v případě úspěšného parsování ji uloží do Elasticsearch:

```
# zpracování zprávy ze syslogu
def process_message(message)
  # získání standartních poli jako je datum, ip adresa, jméno aplikace ..
  hash = get_default_fields(message)

  # parsování zprávy, jako výsledek je vrácena struktura list
  ret_hash =  parse_message(hash[:message])

  # nastavení typu parseru
  hash[:type] = ret_hash[:type]
  # nastavení typu logového záznamu (například přístupový)
  hash[:subtype] = ret_hash[:subtype]
  # získaná zajímavá pole jsou uložena do listu pod klíč fields
  hash[:fields ] = ret_hash[:fields]

  # pokud se parsování povedlo (klíč fields v listu není prázdný) zaindexuje data
  index_data(hash) if  ret_hash[:fields].present?
end
```

Zdr. 12.6: Kompletní proces zpracování logového záznamu

Nejprve je zpráva podrobena získání základních informací, které do ní uložil syslog. To je provedeno na základě znalosti formátu, ve kterém je zpráva do roury ukládána. Ze zprávy jsou odstraněny koncové závorky a dvojtečka (odstranění pidu pro přehlednější filtraci). Získaná data jsou uložena do listu a vrácena.

```
ATTRIBUTES = [:app_name, :level, :facility, :timestamp, :host, :ip_address]

# získání polí ze zprávy a jejich uložení do listu
def get_default_fields(message)
  output = Hash.new
  # uložení získaných polí do listu
  ATTRIBUTES.each do |attribute|
    output[attribute] = message[/(\s*\S+){1}/]
    message = message.sub(/(\S+\s+){1}/, '')
  end

  # odstranění pidu z app name :
  output[:app_name] = output[:app_name].gsub(/(\[.*\]):?$|:$/, "")
  output[:message] = message
  output
end
end
```

Zdr. 12.7: Získání základních údajů ze zprávy ze Syslogu

Poté se zpracovává pouze záznam, který je obsažen v listu pod klíčem message. Při parsování je možné použít připravené regulární výrazy z modulu CommonRegularExpressions. Nejprve je na základě znalosti formátu logového záznamu nalezen začátek zprávy a data za ním. Poté je podle prvního slova určen typ logového záznamu (přístupový, atd) a je připraven regulární výraz pro zjištěný typu záznamu. Poté je regulární výraz proveden a pokud se data shodují, jsou získány zajímavé informace a ty jsou uloženy do listu. O provedení regulárního výrazu a uložení získaných dat do listu se stará metoda parse.

```
include CommonRegularExpressions
# zpracování logového záznamu pomocí regulárních výrazů
def parse_message(message)

  # nalezne začátek záznamu a data za ním
  regexp = /(Started|Processing|Parameters|Completed|ActionController|ActionView|
      Rendered).*/
  message = regexp.match(message).to_s
  subtype = ""

  # podle prvního slova určíme typ záznamu (přístupový, ..)
  case message[/\w+/]
    # pokud začíná started, může se jednat o přístupový
    # log: Started GET "/" for 127.0.0.1 at Wed Jul 07 09:13:27 -0700 2010
    when "Started"
      # regulární výraz, který zachytí přístupový RoR záznam
      regexp = /(?<method_type>\bStarted) (?<method>#{method}) "(?<path>[^"]+)"
          for (?<req_ip_adress>#{ip_address}) at (?<time>#{timestamp('%a %b %d %H
          :%M:%S %z %Y')}|#{timestamp('%Y-%m-%d %H:%M:%S %z')})/
      # nastavíme zjištěný typ
      subtype = "access"
    .. Kód pro další typy, podobný jako předešlý ..
   end
   hash = Hash.new
   # provedeme regulární výraz a výsledek uložíme do listu(hash)
   hash[:fields] = regexp.present? ? parse(message, regexp) : {}

   .. kód pro nastavení typu parseru a logového záznamu a vrácení listu ..
end

# parsuje zprávu pomocí regexp a vrátí výsledek jako list
def parse(message, regexp)
  row = message.to_s
  row.chomp!
  row.strip!
  return unless match = regexp.match(row)
  data = {}
  match.names.each_with_index { |field, index| data[field] = match[index + 1] }
  data
end
```

Zdr. 12.8: Parsování logového záznamu RoR

Poté je provedeno uložení získaných dat do fronty v Redisu. Ukládají se získaná data ze zprávy. Vytvoření mapování pro parsery není nutné, ale je doporučené. Elasticsearch dokáže sám rozpoznat struktury, které ale nemusí být vždy správné. Navíc chceme-li, aby byl řetězec s mezerami považován za celou zprávu (například při vytváření statistik), pak je nutné uložit ho jako not_analyzed.

```
# ukládání dat do elastic search (jsou nejprve uloženy do redis fronty)
def index_data(hash, doc_type)
  @@redis.LPUSH "indexing", hash.except("message").to_json#.to_s
end

def create_mapping(type)
  { type => {
  :properties => {
    :app_name => {:type => 'string', index: 'not_analyzed'},
    :level => { type: 'integer'},
    :fields => { :type => 'object',
      :properties => {
        path: {type: 'string', index: 'not_analyzed'},
}
end
```

Zdr. 12.9: Ukládání dat do Elasticsearch

12.5 Indexer

Indexer je vlákno, které se stará o kontrolování přednastavených upozornění a indexování dat do Elasticsearch. Je to hlavně z důvodů velkého množství příchozích logových záznamů za sekundu. Odesílat každý zpracovaný záznam samostatně by stálo spoustu času a značně by zpomalovalo indexování dat. Proto jsou záznamy shromažďovány a při dosažení nastaveného množství nebo uplynutí časového intervalu jsou hromadně odeslány. Vlákno čeká v nekonečné smyčce na záznamy z fronty pro indexování. Při přijetí záznamu z fronty je záznam nejprve zkontrolován na přednastavená upozornění a pokud je třeba uživatele upozornit, je záznam uložen do fronty pro emailová upozornění. Poté je záznam uložen do pole, které je při dosáhnutí určité velikosti nebo po uplynutí časového intervalu grupováno (stejným zprávám bezprostředně za sebou je nastaven příznak spolu s počtem a časem posledního výskytu) a uloženo do Elasticsearch.

Grupování probíhá za pomoci listu, kdy pro každý parser je uložena poslední zpráva s počtem výskytů a id s uloženým záznamem. Pokud je zpráva stejná jako předešlá, je inkrementován počet zpráv, zapsán čas výskytu a zpráva je buď uložena nebo upravena (upravena je pokud již byla uložena) v Elasticsearch.

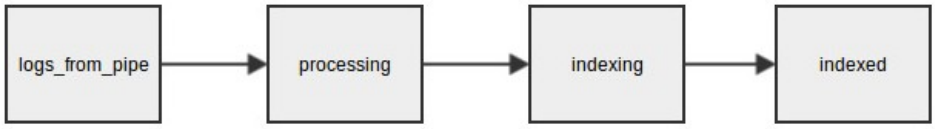
12.6 Mailer

Mailer je vlákno pro odesílání emailů uživatelům při splnění přednastavených upozornění. V nekonečné smyčce čeká na záznamy z fronty pro emailová upozornění. Při přijetí záznamu pak na jeho základě vytvoří zprávu informující uživatele o události, která je vygenerována na základě předdefinovaného upozornění. Vlákno je odděleno především kvůli dlouhé době trvání, které zabere odesílání emailu. Mailer si zazna-

menává odeslaná upozornění a stejné upozornění je znovu odesíláno až po uplynutí nastaveného časového limitu. Dále lze nastavit závislá upozornění, které se neaktivují, pokud již bylo jedno z upozornění v seznamu aktivováno. Nová upozornění jsou průběžně načítána dle nastaveného času.

12.7 Spolehlivost zpracování dat

Spolehlivost zpracování dat zaručuje, že přijatá data budou zpracována a zaindexována. Aby při nečekaném ukončení aplikace nebyla zpracovávaná data ztracena. To je umožněno kombinací 4 front v Redisu viz obr. 12.3.



Obrázek 12.3: Fronty zajišťující spolehlivost zpracování dat

Do první fronty (logs_from_pipe) je zpráva uložena okamžitě při vyzvednutí z roury. Z první fronty jsou data vybírána ke zpracování parsery (parsery využívají atomickou operaci pro výběr a uložení), které při výběru okamžitě zprávu uloží do další fronty (processing). To zaručuje, že nebudou data ztracena. Při úspěšném zpracování parsery jsou pak data uložena do fronty (indexing) a vyjmuta z fronty (processing). Poté jsou data připravována k zaindexování. Vlákno vybere zprávu z fronty (indexing) a okamžitě uloží do fronty (indexed). Pokud jsou data úspěšně uložena do Elasticsearch, tak vrátí seznam úspěšně uložených zpráv, které jsou pak z fronty (indexed) odstraněny. Při startu vlákno Reader vybírá záznamy z fronty (processing), které ukládá do fronty (logs_from_pipe) a z fronty (indexed) vybírá záznamy, které ukládá do fronty (indexing). Tímto postupem je zajištěna spolehlivost zpracování dat.

13 Webová aplikace

Požadavky na webovou aplikaci jsou následující: umožnit vyhledávání části textu nad logovým záznamem nebo nad konkrétním polem. Zobrazit logové záznamy v tabulce a umožnit nastavit zobrazovaná pole. Zobrazit zvolený typ statistik v grafu, který se nejvíce hodí pro daný typ dat. Umožnit filtrovat logové záznamy dle různých kritérií, například logové záznamy podle priority, zařízení, ip adres nebo času mezi dvěma období.

Pro webovou aplikaci byly zvoleny následující prvky, které budou obsaženy v menu a budou mít své stránky: log, dashboard, graph, admin user, alert. Tyto prvky rozebereme jeden po druhém a popíšeme k čemu slouží. Nejprve ale rozebereme technologie , které byly použity při vývoji webové aplikace.

13.1 Použité technologie

V rámci vývoje webové aplikace byl použit pro vytvoření efektivnější a uživatelsky přívětivější aplikace Javascript, Ajax a webový framework Ruby on Rails.

13.1.1 Javascript

Javascript[25] je kompaktní skriptovací jazyk, který je vkládán do webových stránek a do prohlížeče je načítán společně se stránkou. Kód se vykonává přímo v prohlížeči uživatele, čímž může přenést některé úkoly na stranu klienta a tím zmírnit zatížení serveru. Umožňuje oživení stránek, dynamické HTML, ověřování vstupních dat atd. V rámci webové aplikace bude použit pro vykreslení grafů na straně klienta, pro dynamické formuláře (pro každou hodnotu jsou dostupné různé údaje v závislosti na další hodnotě) a pro změnu umístění grafů na stránce.

13.1.2 Ajax

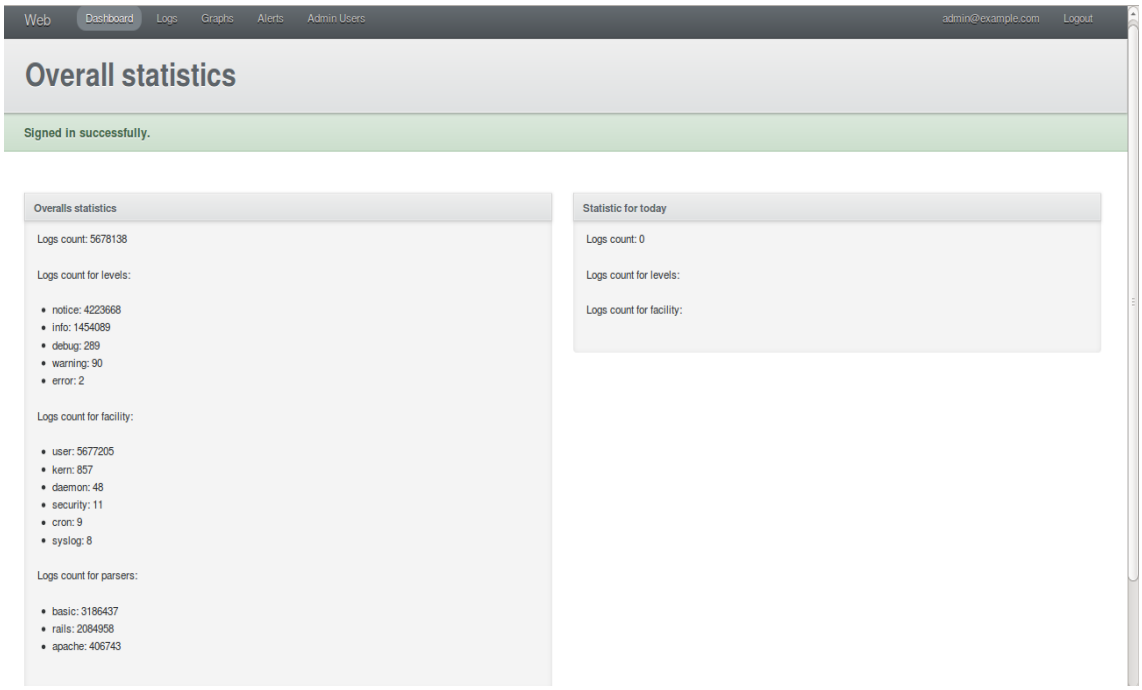
AJAX[26] (neboli Asynchronous JavaScript and XML) je obecné označení technologie, která využívá Javascript pro odesílání asynchronních HTTP/XML požadavků na server. To umožňuje měnit obsah stránek bez nutnosti znovu načítání stránky. Jedná se o kombinaci XML, JavaScript, HTTP a (X)HTML, která umožňuje, aby stránka pomocí Javascriptu kontaktovala server a obdržela od něj libovolná data. Ajax je ve webové aplikaci použit pro dynamické formuláře.

13.1.3 Ruby on Rails

Framework Ruby on Rails je použit pro vývoj webové aplikace. Detailně je popsán v kapitole 7.1.

13.2 Dashboard

Dashboard je uvítací stránka, která zobrazuje základní statistiky pro logové záznamy. Objeví se uživateli po přihlášení a obsahuje dva panely. V levém panelu jsou celkové statistiky logových záznamů. Statistiky obsahují záznamy o celkovém počtu logových záznamů, počtu pro jednotlivé priority, zařízení a pro jednotlivé typy parserů. V pravém panelu jsou obdobné statistiky za dnešní den.



Obrázek 13.1: Úvodní stránka se základními statistikami

Tato funkce je řešena pomocí Elasticsearch. Model odešle dotaz nad všemi indexy a definovanými typy v konfiguračním souboru. Dotaz nad všemi indexy je umožněn použitím tzv. wildcard (index je tvořen vždy názvem 'Log' a časovou značkou, proto použitím 'Log*' se provádí dotaz nad všemi indexy začínajícími Log). V dotazu jsou přítomné facety (statistiky, které poskytuje Elasticsearch) nad poli priorita, typ parseru a zařízení. Elasticsearch vrátí na dotaz výsledky, kde jsou ve facet přítomné statistiky pro požadovaná pole. Příklad vráceného facetu pro prioritu:

```
{"level"=> {"_type"=>"terms",
            "missing"=>0,
            "total"=>6931,
            "other"=>0,
            "terms"=>[
                {"term"=>6, "count"=>6831},
                {"term"=>5, "count"=>100}
            ]
}
```

Zdr. 13.1: Facet obsahující statistiky pro zařízení

Pole "level" je název facetu. Pole "terms" obsahuje pole s datovými strukturami list, v nichž se nacházejí výsledky. V jednotlivých listech pak "term" značí hodnotu a "count" počet nalezených hodnot. Pro výše uvedený příklad tedy existuje 6931 logových záznamu s prioritou 6 a 100 logových záznamu s prioritou 5. V dashboardu stačí už jen vypsat výsledky("terms") pro všechny facety do tabulek.

13.3 Admin user

Tato stránka slouží ke správě uživatelů. Je vygenerována automaticky gemem active admin (viz níže). Je v ní umožněno vytvořit uživatele, který bude mít přístup k webové aplikaci. Upravit nebo odstranit stávajícího uživatele a zobrazit údaje o jednotlivých uživatelích. Například kdy byl uživatel naposledy přihlášen, kolikrát byl přihlášen, z jaké ip adresy, jeho emailovou adresu. V současné době nejsou řešena oprávnění, neboť každý kdo dostane k aplikaci přístup, by měl mít možnost využívat všechny funkce, které aplikace nabízí.

Active admin

Active admin[27] je framework pro vytváření webových rozhraní v administrátorském stylu. Umožňuje vytvoření přehledného a elegantního webu s možností přihlašování. Byl vybrán kvůli zkušenostem autora s vývojem aplikací za pomoci tohoto frameworku a kvůli výslednému přehlednému vzhledu webové aplikace.

13.4 Alert

Stránka sloužící k nastavení emailových upozorněních. Upozornění se nastavuje pro konkrétní typ parseru a logového záznamu. Jsou k dispozici dva druhy upozornění. Prvním je numerické, pro které jsou k dispozici standardní porovnání (větší, menší, rovno). Druhým je regulární, kdy je možné napsat vlastní regulární výraz a porovnat ho s určeným polem. Tyto upozornění lze kombinovat za pomoci and. Jednotlivá

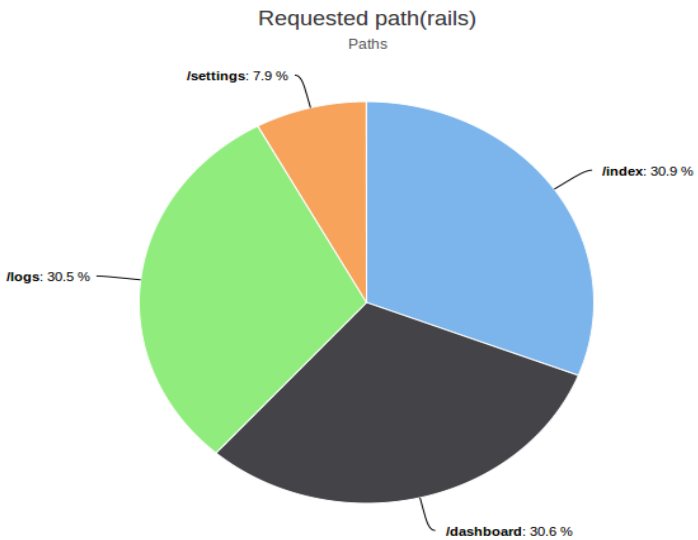
pole jsou k výběru určena v souladu s mapováním příslušného typu parseru. Tato upozornění jsou kontrolována a to ve třídě Mailer. Ten vybere všechna upozornění z databáze a uloží si je do struktury (průběžně jsou aktualizována dle nastavení). Tato pravidla jsou před zaindexováním aplikována pro zpracované logové záznamy .

13.5 Graph

Jednotlivé logové záznamy mohou obsahovat informace, které je možné následně zobrazit v grafu. Možné je grafem zobrazit i statistické údaje získané z jednotlivých zpráv. Uživatel si může vytvořit libovolný graf a definovat v něm jeho zobrazení. U každého grafu je k dispozici mnoho voleb, které definují výsledný graf a které budou probrány níže.

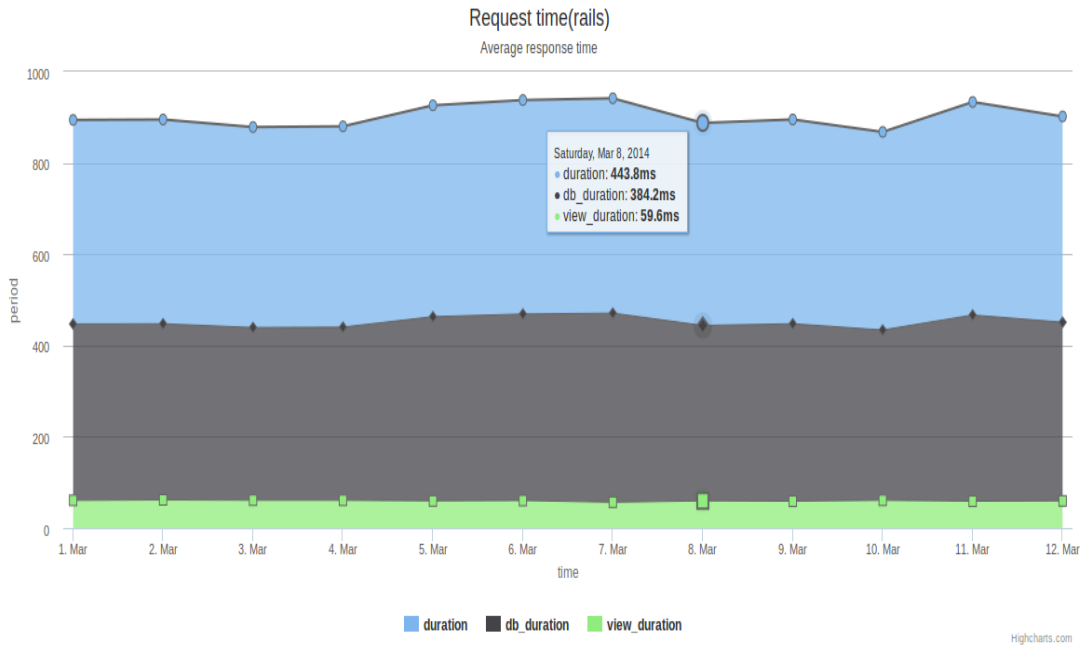
Typy facetů: Aplikace v současné chvíli podporuje 4 druhy facetů (statistik). Byly vybrány facety z Elasticsearch, které z dat dokáží získat užitečné informace.

terms: Statistika počtu výskytů různých hodnot v určeném poli. Výhodou tohoto facetu je, že může být použit na libovolný typ pole (například i pro řetězec). Lze tak například získat statistiky zobrazování jednotlivých stránek viz obr.13.2



Obrázek 13.2: Ukázka použití terms facetu (statistika návštěvnosti stránek)

date: Statistika za časový interval pro určené pole. Lze nastavit, podle kterého pole má být čas určován a časový interval (minuta, hodina , den , atd). Pole podle kterého je čas určován musí být typu date a určené pole (jehož hodnoty se zobrazují) musí být numerické. Statistika může být vypočtena z počtu hodnot, maximální nebo minimální hodnoty, celkové sumy nebo průměru. Lze pak například zjistit průměrnou dobu odpovědí serveru v intervalu dnů viz obr.13.3.

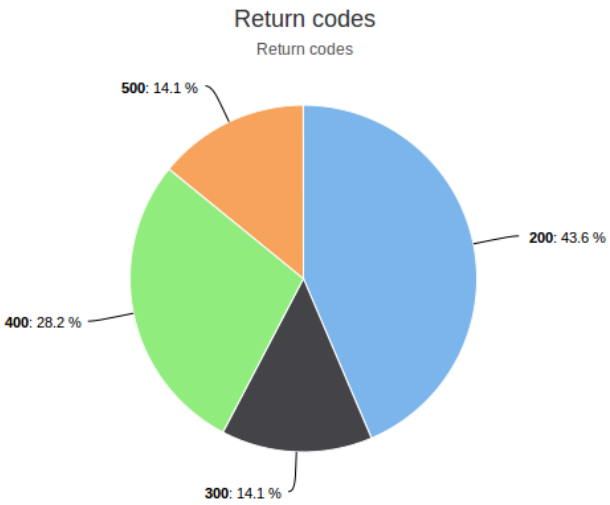


Obrázek 13.3: Ukázka použití date facetu (průměrná doba vyřízení požadavků za období)

histogram: Statistika pro určené pole dle zvoleného rozsahu. Určené pole musí být numerické. Dále vyžaduje nastavit rozsah, podle kterého jsou hodnoty rozdělovány do intervalů. Statistika je vhodná, pokud potřebujeme vědět kolik hodnot je v určitém intervalu. To může být užitečné pro návratové kódy HTTP požadavků. Při volbě intervalu 100 jsou rozděleny do intervalů, z nichž lze vidět počet informačních (1xx), úspěšných (2xx), přesměrovaných (3xx), chybných (4xx), na serveru chybných (5xx) požadavků viz obr.13.4.

statistical: Statistika pro určené pole, která umožňuje provést speciální statistické funkce. Určené pole musí být numerické. Poskytuje statistické funkce počet, maximum, minimum, sumu, průměr, sumu čtverců, standardní deviaci a variaci. Může být vhodné využít pro určení průměrné doby vyřízení všech požadavků.

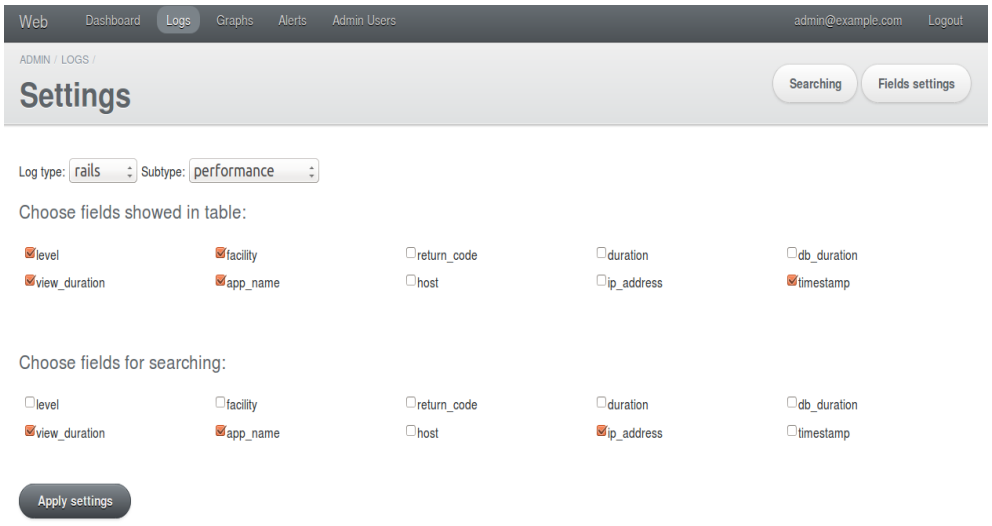
Typy grafů: Jak již bylo řečeno, pro každý typ dat se hodí specifický typ grafu. Proto je uživateli umožněno vybrat libovolný typ grafu z nabídky a určit jeho velikost (dle přednastavených možností). V aplikaci je k dispozici 5 typů grafů (výsečové, plošné, bodové, sloupcové, spojnicové).



Obrázek 13.4: Ukázka použití histogram facetu (zastoupení návratových kódů v rozsahu 100)

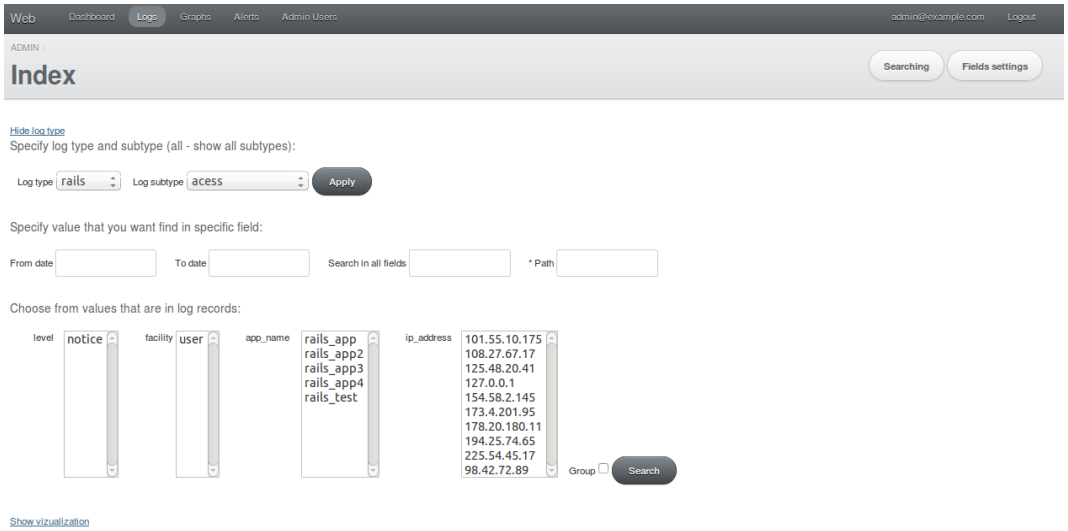
13.6 Log

Stránka je určena pro zobrazování, vyhledávání a vizualizaci logových záznamů. Obsahuje dvě možnosti přepnutí: Searching a Fields settings. Fields settings přepne okno na stránku, kde je umožněno pro typ parseru a logového záznamu určit, která pole budou zobrazována v tabulce a která budou k dispozici pro hledání viz obr.13.5. Toto nastavení se ukládá uživateli do databáze podle typu parseru a logového záznamu.



Obrázek 13.5: Stránka pro nastavení vyhledávacích a filtrovacích polí

Searching je stránka pro vyhledávání logových záznamů nad zvolenými poli, jejich filtrování, zobrazení v tabulce a vizualizaci v grafech. Skládá se z několika částí. Vrchní část obsahuje volbu typu parseru a logového záznamu. Po zvolení typu se zobrazí definovaná pole pro vyhledávání viz obr. 13.6. Pod nimi se nachází vizualizace logových záznamu, kde lze nastavit, které grafy mají být zobrazeny. Ve vizualizaci se nechají nastavit intervaly pro zobrazení (například počet logových záznamů za jednotlivé dny, měsíce, atd). Pod vizualizací se nachází tabulka zobrazující data pro zvolená pole.



Obrázek 13.6: Webová aplikace - filtrování

Vyhledávání a zobrazování logových záznamů probíhá následovně. Do Elasticsearch je nejprve proveden dotaz na získání všech hodnot pro pole, z nichž lze vybírat (nabídka z existujících hodnot - například ip adresy serverů). Jedná se o pole jako jméno aplikace, ip adresy, jméno hosta, priorita a zařízení. Děje se tak pro usnadnění práce uživateli se záznamy. Následně je proveden druhý dotaz se zvolenými parametry pro vyhledávání. Některé parametry (například priorita, zařízení) se musí převést na číselnou hodnotu, ve které jsou uloženy. Jako čísla jsou uloženy kvůli možnostem podrobnějších statistik (zobrazení grafů s jednotlivými prioritami v intervalu hodin). Poté jsou načteny zobrazované grafy z databáze, které jsou převedeny do požadovaného tvaru a připojeny k dotazu do Elasticsearch. Elasticsearch vrátí výsledky s nalezenými záznamy a statistikami pro zvolené grafy.

Poté aplikace na základě uživatelova nastavení zobrazí logové záznamy s určenými poli v tabulce. Vytvoří grafy pro získané statistiky a zobrazí pole pro vyhledávání s hodnotami. Grafy jsou vytvořeny v podobě Javascriptového kódu, který je na stránce uložen. Následně jsou vykresleny pomocí Javascriptové knihovny Highcharts na straně uživatele do předpřipravených kontejnerů (divů).

14 Testování aplikace

Testování aplikace probíhalo za pomoci dvou aplikací, kterým bylo nastaveno logování do syslogu. Jednalo se o webovou aplikaci v Ruby on Rails a šlo o vytvořenou aplikaci pro zobrazování logových záznamů. Druhá byla aplikace v Javě, která byla napsána pro generování stacktrace záznamů (například při dělení nulou), jenž byly posílány do syslogu. Pomocí tohoto postupu byla pouze ověřena funkčnost celého řešení a to že jsou záznamy skutečně odesílány a následně jsou přístupné ve webové aplikaci. Nebylo ovšem možné tímto způsobem nasimulovat reálný provoz.

Pro možnost nasimulování reálného provozu byly napsány generátory fiktivních logových záznamů, pro RoR a Javu (Tomcat). Tyto generátory náhodně vytvářely údaje s různými atributy. Jednalo se o přístupové logové záznamy pro RoR a Tomcat. Pro RoR byly ještě generovány výkonnostní logové záznamy. Generátory ukládaly záznamy do roury, odkud byly vyzvedávány ke zpracování. Ukládání záznamů rovnou do roury bylo provedeno z důvodu zanedbatelné režie syslogu a protože cílem testů bylo především otestovat vytvořené řešení pro parsování záznamů. Pro testování byly nakonfigurováno 4 typy parserů, jednalo se o parsery pro Apache, RoR, Tomcat a basic. Basic je parser, který pouze získává základní údaje o zprávě jako je jméno aplikace, ip adresa, zpráva, atd. Basic neprovádí funkci parse message, ale pouze uloží příchozí zprávu ve strukturované podobě. Tento parser uloží zprávu vždy a obsahuje pouze údaje získané syslogem. Proto je obvykle záznam uložen dvakrát, pokaždé ovšem v jiné podobě, neboť je rozparsován ještě jiným parserem než je Basic.

Pro zvolení správných typů parserů pro příchozí záznam byla zvolena v souboru *config.yml* následující konfigurace (obdobně byly nakonfigurovány i parsery pro Tomcat a Apache, které nejsou součástí ukázky):

```
rails:
  class_name: Parser
  file_location: parser.rb
  log_type: rails
  app_name:
    - "rails_test"
    - "rails_app"
    - "rails_app2"
    - "rails_app3"
    - "rails_app4"

syslog:
  class_name: Syslog
  log_type: basic
  file_location: syslog.rb
```

Zdr. 14.1: Konfigurace parserů při testování

Tato konfigurace na základě aplikace vybere vhodný parser, kterým je pak zpráva zpracována. Vybrání vhodného parseru může být na základě priority, zařízení nebo jména aplikace. Konfigurace byla provedena v souladu s vytvořenými generátory

a jejich možnými hodnotami.

Parametry počítače, na kterém byla aplikace testována:

procesor: Intel® Core™ i5-2450M CPU @ 2.50GHz, 4 jádra
operační paměť: 4 GB
operační systém: Ubuntu 13.04 (64 bitový)

Pro nasimulování bylo generováno 100 000 zpráv jednoho typu logového záznamu, s různým množstvím zpráv za sekundu. Pro test byly zvoleny dva rozsahy: 500 zpráv za sekundu a 1000 zpráv za sekundu. Tento test byl uskutečněn pro přístupové logové záznamy typu RoR a Tomcat. Každý test byl proveden třikrát a získané hodnoty byly zprůměrovány.

Následující tabulka ukazuje statistiky při generování 1000 zpráv za sekundu typu rails přístupový. Celkem bylo vygenerováno 100 000 zpráv, přičemž zpráva prošla dvěma parsery (parserem Basic a poté parserem rails) a proto jich bylo uloženo celkem 200 000. Během zpracování běžela aplikace generující logové záznamy, což ovlivnilo výsledky (v testech jsou záznamy o využití zdrojů generátorem).

Čas (sekund)	60	120	180	240	244
Spotřeba paměti (MB)					
Elasticsearch	352	353	356	344	344
Reader	332	334	334	335	335
Generator	182	179	147	0	0
Využití procesoru (%)					
Reader	36	38	38	34	33
Generator	18	13	8	0	0
Velikost fronty					
logs_from_pipe	3200	10753	1	0	0
processing	3	0	3	0	0
indexing	30 173	74 268	108 946	6 247	0
indexed	260	561	61	841	0
Zaindexovaných zpráv					
Počet	28 000	56 000	91 000	194 000	200 000

Tabulka 14.1: Test pro 1000 zpráv za sekundu typu RoR přístupový

Z výsledků vychází, že aplikace zaindexovala v průměru 820 zpráv za sekundu. Z průběhů testu je vidět, že aplikace záznamy rychle parsuje, ale má problém při jejich indexování, které zpomaluje celý proces a indexování se značně zrychlí po dokončení parsování.

Test probíhal totožně jako předešlý, s rozdílem počtu zpráv vygenerovaných za sekundu. Ten byl zmenšen na 500 zpráv za sekundu.

Čas (sekund)	60	120	180	240	264
Spotřeba paměti (MB)					
Elasticsearch	302	315	331	344	344
Reader	328	330	332	332	331
Generator	187	144	147	154	0
Využití procesoru (%)					
Reader	27	29	29	30	29
Generator	22	10	8	7	0
Velikost fronty					
logs_from_pipe	635	274	783	404	0
processing	3	0	3	0	0
indexing	2205	2459	2568	3700	0
indexed	260	1000	61	782	0
Zaindexovaných zpráv					
Počet	42 000	93 000	141 000	192 000	200 000

Tabulka 14.2: Test pro 500 zpráv za sekundu typu RoR přístupový

Z výsledků vychází, že aplikace zaindexovala v průměru 758 zpráv za sekundu. Z průběhů testu je vidět, že aplikace provoz stíhala bez problému a nemusela ukládat větší množství záznamů do fronty.

Pro získání více informací o rychlosti zpracování bylo předgenerováno 100 000 RoR přístupových logových záznamů do fronty logs_from_pipe. Následně byla aplikace spuštěna bez ovlivnění generátorem.

Čas (sekund)	60	120	180	210	224
Spotřeba paměti (MB)					
Elasticsearch	363	363	363	364	364
Reader	334	337	337	337	337
Využití procesoru (%)					
Reader	58	56	52	48	47
Velikost fronty					
logs_from_pipe	66 807	30 028	0	0	0
indexing	30 378	66 938	74 998	24 998	0
indexed	626	728	1000	366	0
Zaindexovaných zpráv					
Počet	37 000	73 000	125 000	176 000	200 000

Tabulka 14.3: Test pro 100 000 předgenerovaných zpráv typu RoR přístupový

Z výsledků vychází, že aplikace zaindexovala v průměru 893 zpráv za sekundu. Z průběhů testu je vidět, že aplikace záznamy rychleji parsují a při jejich ukládání do Elasticsearch vlákno pro indexování nestíhá.

Dalšímu testu bylo podrobeno pouze parsování záznamů. Opět bylo vygenerováno 100 000 zpráv. Parsování probíhalo společně s generováním záznamů.

Čas (sekund)	30	60	90	120	149
Spotřeba paměti (MB)					
Reader	329	327	329	329	329
Využití procesoru (%)					
Reader	35	36	36	36	36
Velikost fronty					
logs_from_pipe	1443	14	1727	921	0
indexing	24 804	69 966	112 536	157 978	200 000
Zaindexovaných zpráv					
Počet	24 804	69 966	112 536	157 978	200 000

Tabulka 14.4: Test parsování 100 000 zpráv typu RoR přístupový

Z výsledků vychází, že aplikace zparsovala v průměru 1342 zpráv za sekundu.

Poslednímu testu bylo podrobeno pouze indexování záznamů. Aplikaci bylo předpřipraveno 200 000 zpráv připravených k zaindexování.

Čas (sekund)	30	60	90	115
Spotřeba paměti (MB)				
Elasticsearch	364	364	364	364
Reader	322	328	328	328
Využití procesoru (%)				
Reader	37	31	28	27
Velikost fronty				
indexing	152 998	98 787	43 998	0
indexed	891	223	1000	0
Zaindexovaných zpráv				
Počet	47 002	101 213	156 002	200 000

Tabulka 14.5: Test zaindexování 100 000 zpráv typu RoR přístupový

Z výsledků vychází, že aplikace zaindexovala v průměru 1739 zpráv za sekundu. Po testech zabíralo 9 000 000 logových záznamů v Elasticsearch celkem 2191,2MB.

Po tomto testu byla podrobena dalšímu na spolehlivost zpracování dat. Bylo vygenerováno 100 000 záznamů, během jejichž zpracovávání byla aplikace několikrát násilně ukončena. Následně po spuštění obnovila záznamy z front a pokračovala ve zpracování. Po dokončení se nacházelo v Elasticsearch 200 000 záznamů a byla tak potvrzena spolehlivost zpracování dat.

Zhodnocení testu

Předvedené testy byly ukázány pro RoR přístupový. Aplikace byla podrobena testování i pro další typy logových záznamů (RoR výkonnostní, Tomcat přístupový). Získané výsledky byly téměř totožné s předvedeným testem. Na základě výsledků je schopna aplikace zvládnout provoz o objemu nejméně 758 logových záznamů za sekundu (při souběžném generování záznamů generátory zvládla 500 záznamů). Při větším provozu si aplikace data připraví a během menší zátěže zvládne zprávy doindexovat.

Z výsledků testu je vidět, že nastává při větším zatížení problém s indexováním dat. Problémem může být to, že o parsování záznamů se stará více vláken (v závislosti na nastavení - v testu 3), zatímco data do Elasticsearch zapisuje pouze jedno vlákno (indexer). Proto pro možné zlepšení a dosažení většího množství zpracovaných záznamů je vhodné optimalizovat tuto část.

15 Závěr

V práci byla probrána problematika pořizování a analyzování logových záznamů. Na základě nastudování této problematiky bylo vytvořeno řešení pro centralizovaný sběr logových záznamů. Řešení využívá obvykle používaný nástroj pro sběr logových záznamů (syslog) a tím umožňuje snadnou implementaci do existujícího řešení. Použitím syslogu lze zasílat logové záznamy z libovolného zařízení, neboť odesílaní záznamů do Syslogu podporuje naprostá většina aplikací pro logování. Zpracování příchozích záznamů provádí parsery, které získávají podstatné informace a ukládají je ve strukturované podobě do Elasticsearch. Během zpracování jsou kontrolovány přednastavená upozornění a v případě pozitivního testu je uživatel informován. Pro zpracování záznamů byla důsledně zvolena architektura, která umožňuje jejich rychlé zpracování. Problémem není ani neznámý formát logového záznamu, protože lze jednoduše rozšířit aplikaci o další parser pro daný typ. Skrze webovou aplikaci je přístupné vyhledávání a filtrování v logových záznamech, ve kterém je umožněno komplexní filtrování a vyhledávání. Výsledky lze přehledně vizualizovat do předkonfigurovaných nebo ručně vytvořených grafů.

Během práce byly získány zkušenosti se zpracováním velkého množství dat. Bylo nutné umožnit rychlé zpracování záznamů. Tím pádem musela být optimalizována architektura zpracování zprávy. Jednotlivé části zpracování byly rozděleny do funkcí, které jsou spouštěny ve vláknech. Časově náročné funkce jako parsování zprávy je spuštěno vícekrát. Problémem byl paralelismus v Ruby, který ve skutečnosti neumožňoval současný běh vláken. Z tohoto důvodu bylo Ruby nahrazeno JRuby, které má stejné vlastnosti jako Ruby a navíc umožňuje skutečný paralelismus. Dalším problémem byla rychlost ukládání zpráv při použití architektury, ve které parser při úspěšném zpracování zprávu ihned ukládal do Elasticsearch. Tímto přístupem ukládal maximálně 170 záznamů za sekundu. Proto byl přidán indexer, který shromažďuje zpracované záznamy a při dosažení definovaného počtu nebo uplynutí času hromadně odesílá záznamy k uložení do Elasticsearch. Indexer značně zrychlil proces ukládání a umožnil zpracování kolem 700 záznamů za sekundu. V první fázi vývoje byly jako fronty použity Queue struktury v Ruby, což mělo za následek při nečekaném ukončení aplikace ztrátu všech záznamů. Proto byly tyto struktury nahrazeny databází Redis, která je použita jako fronta.

Výsledkem práce je splněné zadání. Nástroj je užitečný pro vyhledávání konkrétních záznamů a umožňuje nalézt požadované informace. Vizualizace poskytuje přehledné statistiky, které mohou pomoci pro optimalizaci stránek, získání přehledu o oblíbenosti jednotlivých stránek nebo analyzování chyb a útoků. Je snadné používat nástroj pro dodané typy logových záznamů a lze rozšířit aplikaci o další formát logového záznamů. Vytvořené řešení bylo testováno na reálných datech a byla ověřena jeho funkčnost. Pro další rozvoj je možné napsat více parserů a umožnit tak zpracování více typů logových záznamů. Nebo učinit webovou aplikaci přehlednější a lépe pochopitelnou pro běžného uživatele.

Literatura

[1] *Erik Troan, Preston Brown:* **Logrotate** [on-line]
4th Berkeley Distribution, 5.11.2002
Dostupné z URL: <http://linuxcommand.org/man_pages/logrotate8.html>

[2] *Balder Van Camp:* **The State of Logging in Java 2013** [on-line]
6.8.2013, Dostupné z url: <http://zeroturnaround.com/rebellabs/the-state-of-logging-in-java-2013>

[3] **Logging** [on-line] [cit. 30.4.2014]
Dostupné z URL: <https://www.ruby-toolbox.com/categories/Logging>

[4] **Syslog.org Home** [on-line]
Dostupné z URL: <http://www.syslog.org>

[5] **The syslog-ng Premium Edition 5 LTS Administrator Guide** [on-line]
15.3.2014, Dostupné z URL: <http://www.balabit.com/sites/default/files/documents/syslog-ng-pe-5.0-guides/en/syslog-ng-pe-v5.0-guide-admin/html-single/index.html>

[6] *R. Gerhards,:* **The Syslog Protocol** [online]
RFC 5424, Adiscon GmbH, Network Working Group, březen 2009, Dostupné z URL: <http://tools.ietf.org/html/rfc5424>

[7] *Jiří Václavík :* **Perl (16) - Regulární výrazy - začínáme** [online] 19.10.2005,
Dostupné z URL: <http://www.linuxsoft.cz/article.php?id_article=947>

[8] *Jeffrey E. F. Friedl:* **Mastering Regular Expressions, 3rd Edition**
O'Reilly Media, Sebastopol 2006, ISBN: 0-596-52812-4

[9] *Roger Hughes:* **The Producer Consumer Pattern** [on-line] 26.02.2013,
Dostupné z url: <http://java.dzone.com/articles/producer-consumer-pattern>

[10] *Darryl Grove:* **Programování aplikací pro vícejádrové procesory**
Computer Press, 2011, ISBN: 978-80-251-3487-0

[11] **Redis** [on-line]
Dostupné z url: <http://redis.io>

[12] **MongoDB** [on-line]
Dostupné z url: <http://www.mongodb.com/>

[13] *Brian Proffitt:* **When NoSQL Databases Are — Yes — Good For You And Your Company** [on-line] 25.03.2013,
Dostupné z url: <http://readwrite.com/2013/03/25/when-nosql-databases-are-good-for-you>

[14] *Luke P. Issac* **SQL vs NoSQL Database Differences Explained with few Example DB** [on-line] 14.01.2014,
Dostupné z URL: <http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db>

[15] **Apache Solr** [on-line]
Dostupné z URL: <http://lucene.apache.org/solr>

[16] **Elasticsearch** [on-line]
Dostupné z URL: <http://www.elasticsearch.org>

[17] **Getting Started with Rails** [on-line]
Dostupné z URL: <http://guides.rubyonrails.org/getting_started.html>

[18] *Sam Ruby, Dave Thomas, David Heinemeier Hansson:* **Agile Web Development with Rails Fourth Edition**
The Pragmatic Programmers LLC, 2010, ISBN: 1-934356-54-9

[19] **Ruby on Rails/Getting Started/Model-View-Controller** [on-line]
Dostupné z URL: <http://en.wikibooks.org/wiki/Ruby_on_Rails/Getting_Started/Model-View-Controller >

[20] **Highcharts** [on-line]
Dostupné z URL: <http://www.highcharts.com>

[21] **jqPlot** [on-line]
Dostupné z URL: <http://www.jqplot.com>

[22] **Rsyslog** [on-line]
Dostupné z URL: <http://www.rsyslog.com>

[23] **Syslog-ng** [on-line]
Dostupné z URL: <http://www.balabit.com/network-security/syslog-ng>

[24] *Ilya Grigorik* **Parallelism is a Myth in Ruby** [on-line] 13.11.2008,
http://www.igvita.com/2008/11/13/concurrency-is-a-myth-in-ruby>

[25] *Steven Holzner:* **Javascript-profesionálně**
Mobil Media A.S, Brno 2003, ISBN: 80-86593-40-1

[26] *Lee Babin:* **Beginning Ajax with PHP - From Novice to Professional**
Springer-Verlag, New York 2007, ISBN: 1-59059-667-6

[27] **Active Admin** [on-line]
Dostupné z URL: <http://www.activeadmin.info/documentation.html>

Seznam tabulek

Seznam obrázků

Seznam zdrojových kódů a konfigurací

Příloha

A Instalace a konfigurace

Instalace je rozdělena do několika kroků, které je třeba provést v uvedeném pořadí. Další nastavení jsou přiložena v CD. Konfigurace je předvedena pro Linux a konkrétně pro distribuci Ubuntu.

A.1 Instalace JRuby

Pokud nemáte k dispozici Ruby interpret , pak jsou doporučeny dvě možnosti. První je nainstalovat ho pomocí balíku apt-get jruby:

```
sudo apt-get install jruby
```

Druhou možností je nainstalovat ho pomocí rvm. Rvm umožňuje snadnou práci s více ruby interprety. Nejprve v konzoli zadejte příkaz pro nainstalování curl balíku, který umožní stáhnout nejnovější verzi rvm:

```
sudo apt-get install curl
```

Příkaz stáhne a nainstaluje stabilní verzi rvm.

```
\curl -sSL https://get.rvm.io | bash -s stable
```

Poté nainstalujte jRuby:

```
rvm install jruby
```

A.2 Instalace Elasticsearch

Do konzole zadejte následující příkazy, které stáhnou a nainstalují Elasticsearch:

```
wget https://download.elasticsearch.org/elasticsearch/elasticsearch/
elasticsearch-1.1.1.deb
```

```
sudo dpkg -i elasticsearch-1.1.1.deb
```

Pro nastavení spuštění Elasticsearch jako služby po startu systému.

```
sudo update-rc.d elasticsearch defaults 95 10
sudo /etc/init.d/elasticsearch start
```

A.3 Instalace Redisu

Nainstalování balíku obsahující Redis server.

```
apt-get install redis-server
```

Spuštění Redis serveru.

```
redis-server
```

A.4 Instalace databáze

V případě že již používáte databázi, stačí nastavit konfiguraci v adresáři webové aplikace v config/database.yml a v souboru database.yml nacházející se v adresáři, kde je aplikace pro parsování zpráv. V případě existující databáze vyberte adaptér z následující stránky https://github.com/jruby/activerecord-jdbc-adapter a řiďte se uvedenými pokyny.

Zde bude předvedena instalace a konfigurace pro Mysql. Nejprve nainstalujte následující balíky:

```
sudo apt-get install mysql-server mysql-client
sudo apt-get install libmysql-ruby
```

Přihlaste se do Mysql a proveďte následující příkazy:

```
přihlásí se do mysql
mysql -u root -p

# vytvoří uživatele ('username' - nahradit jménem, 'password' -
nahradit heslem)
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';

# vytvoří databázi pojmenovanou databaze
create database databaze;

# nastaví uživateli s username práva pro přistup k databází databaze
grant all privileges on databaze.* to username@localhost ;
```

Tyto údaje je nutné nakonfigurovat v adresáři webové aplikace v config/database.yml a v souboru database.yml nacházející se v adresáři, kde je aplikace pro parsování zpráv. Konfigurace pro výše zmíněné by měla vypadat:

```
adapter: jdbcmysql
host: localhost
database: databaze
username: username
password: password
encoding: utf8
```

Zdr. A.1: Konfigurace databáze

A.5 Instalace webové aplikace

V adresáři kde se nachází webová aplikace je nutné provést stažení balíku nodejs:

```
sudo apt-get install nodejs
```

Nainstalovat gem bundler, který umožňuje spuštění bundleru:

```
gem install bundler
```

Nainstalovat požadované gemy ze souboru *Gemfile* a provést migrace do databáze:

```
bundle install
rake db:migrate
```

Pokud chceme vytvořit připravené grafy, je nutné spustit:

```
rake create_graphs
```

A.6 Instalace a nastavení syslogu

Pokud nemáte nainstalován rsyslog, postupujte podle instalace uvedené na stránce: http://www.rsyslog.com/tag/installation/

V případě že není vytvořena roura, je nutné ji nejprve vytvořit ('jmeno' - jméno vytvářené roury), doporučeno je vytvoření v adresáři, kde se nachází aplikace pro parsování:

```
mkfifo 'jmeno'
```

Konfigurace pro Rsyslog se nachází v souboru */etc/rsyslog.conf*. Do tohoto souboru stačí přidat následující řádky. Pro povolení příjímání zpráv pomocí UDP protokolu a nastavení maximální podporované délky zpráv (je zvolena maximální nastavitelná délka zprávy v RSyslogu):

```
$ModLoad imudp
$UDPServerRun 514
$MaxMessageSize 32k
```

Zdr. A.2: Konfigurace pro UDP a maximální délku zprávy

Ukládání všech příchozích zpráv do roury dle definované šablony (CESTA musí být nahrazena umístěním, ve kterém se nachází roura, pipe - musí být nahrazena jménem, kterým je pojmenována vytvořená roura):

```
\$template MessageFormat,'%syslogtag% %syslogseverity% %syslogfacility% %timestamp
    :::date-rfc3339% %fromhost% %fromhost-ip% %msg% \n'

*.* |CESTA/pipe;MessageFormat
```

Zdr. A.3: Konfigurace šablony

Pro načtení aktuální konfigurace je třeba v konzoli restartovat Rsyslog:

```
sudo service rsyslog restart
```

B Spuštění

B.1 Spuštění aplikace pro parsování

V adresáři kde se nachází aplikace pro parsování by jste měli nejprve provést nastavení parserů v souboru *config.yml*, které mají být použity. Parser je možné zvolit podle jména aplikace, priority a zařízení:

```
rails:
  # jméno třídy
  class_name: Parser
  # adresa k souboru
  file_location: parser.rb
  # typ logu
  log_type: rails
  # parser bude použit pro zprávy, jejichž jméno aplikace bude z následujícího
      seznamu
  app_name:
    - "rails_test"
    - "rails_app"
    - "rails_app2"
    - "rails_app3"
    - "rails_app4"
  level:
    - 6
```

Zdr. B.1: Konfigurace parserů

Následně můžete spustit aplikaci:

```
ruby reader.rb
```

Spuštění parseru jako služby

Aplikaci pro parsování lze spustit jako službu. V adresáři reader je připraven skript *log_parser*, který je nutné přesunout do */etc/init.d*. Ve skriptu je nutné upravit následující údaje a nahradit je správnými hodnotami.

```
# jméno spuštěné služby
APP_NAME=reader
# cesta k adresáři obsahující parser
APP_PATH=~/Projekt/Parser/reader.rb

# cesta k jruby
JRUBY=~/.rvm/rubies/jruby-1.7.12/bin/ruby
export GEM_HOME=~/.rvm/gems/jruby-1.7.12
```

Zdr. B.2: Konfigurace skriptu pro parser

Spuštění služby:

```
sudo /etc/init.d/log_parser start
```

Zastavení služby:

```
sudo /etc/init.d/log_parser stop
```

B.2 Spuštění webové aplikace

V adresáři webové aplikace zadejte:

```
rails s
```

Aplikace je přístupná na adrese: http://localhost:3000