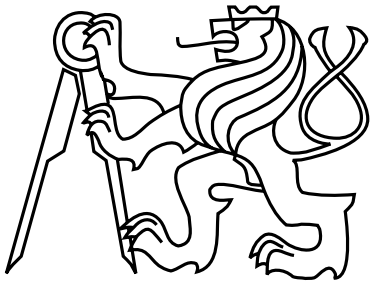
Zadanie diplomovej práce

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačovej grafiky a interakcie



Diplomová práca

Analyzátor a konvertor logov

Bc. Martin Kropuch

Vedúci práce: Ing. Pavel Strnad, Ph.D.

Študijný program: Otvorená informatika, štruktúrovaný, magisterský

Odbor: Softvérové inžinierstvo

5. januára 2015

Poďakovanie

Týmto by som sa chcel poďakovať svojmu vedúcemu Ing. Pavlovi Strnadovi, Ph.D. za zadanie a pomoc pri písaní mojej diplomovej práce, za všetok čas, ktorý mi vždy ochotne venoval a za poskytnuté rady týkajúce sa návrhu a implementácie riešenia. Taktiež ďakujem svojej rodine a blízkym, ktorí ma počas písania práce neustále podporovali. Bez nich by táto diplomová práca nevznikla.

Prehlásenie

Prehlasujem, že som svoju diplomovú prácu vypracoval samostatne s využitím podkladov v priloženom zozname. Nemám závažný dôvod proti použitiu tohoto diela v zmysle §60 Zákona číslo 121/2000 Zb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon).

V Prahe dňa 4. 1. 2015 .....

Abstract

Web log analysis is young and dynamically developing industry of IT, which helps to dig great amount of relevant information from log files to those working with web technologies. This thesis introduces us the issues connected with this process, describes basic principles of the industry and deals with analysis, implementation and configuration of modular log analyzer and converter. It focuses on processing log files from Apache, PostgreSQL and JBoss, but it also discusses future extensions to other technologies.

Abstrakt

Analýza webových logov patrí medzi mladé, veľmi dynamicky napredujúce IT odvetvie, ktoré pomáha z logových súborov získať obrovské množstvo relevantných informácií pre všetkých, ktorí pracujú s webovými technológiami. Táto práca nás uvádza do tejto problematiky, popisuje základy tohto odvetvia a zaoberá sa návrhom a implementáciou modulárneho analyzátora a konvertora logových súborov. Zameriava sa predovšetkým na spracovanie logových súborov z technológií Apache, PostgreSQL a JBoss, ale pojednáva aj o možnostiach rozšírenia o ďalšie technológie.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Logovanie je dnes v informatickom svete bežným a často používaným pojmom. Kapacita pevných diskov v súčasnosti umožňuje zaznamenávať, teda logovať, takmer všetko, čo sa v počítačoch a iných zariadeniach deje. Logovanie prebieha na úrovni servera, operačného systému či jednotlivých programov a procesov, ktoré v rámci týchto systémov bežia. Proces logovania prebieha vo väčšine prípadov neustále. Iba tak má totiž zmysel a poskytne nám komplexné informácie o tom, čo sa deje na pozadí toho ktorého systému, procesu či servera.

Získame tak obrovské množstvo údajov a informácií. Tieto informácie nás väčšinou až tak nezaujímajú, pokiaľ všetko funguje tak, ako má. Študovanie logov fungujúceho systému sa po chvíli stáva nudným a my sa radšej vrátime k využívaniu možností a funkcií, ktoré nám fungujúci systém poskytuje. Skutočnú hodnotu logovania si uvedomíme až vo chvíli, keď niečo fungovať prestane. Vtedy prichádza na rad skúmanie súborov vytvorených v procese logovania a ich analýza.

Práve analýza je tým mocným nástrojom, ktorý nám z neprehľadnej spleti údajov pomôže získať relevantné informácie, ktoré potrebujeme. Môžeme tak vypátrať zdroje rôznych chýb, odhaliť príčiny zvláštneho správania sa systému či vytvoriť prehľadné štatistiky v období, keď všetko funguje. Pomocou analýzy a nástrojov, ktoré nám poskytuje môžeme všetky tieto informácie filtrovať, kategorizovať či rozdeľovať. Až v tomto momente, teda po vykonaní analýzy, si uvedomujeme akú moc logovanie v skutočnosti má.

V tejto diplomovej práci sa zaoberáme procesom logovania a následnou analýzou logov na webových serveroch a pridružených technológiách. Konkrétne ide o webový server Apache, aplikačný server JBoss a systém na správu databáz PostgreSQL. Práve tieto tri systémy totiž využíva počítačový systém (celý systém zatiaľ nemá meno) zadávateľa. Logové súbory z tohto systému budú slúžiť na postavenie špecifikácie systému v algebre PEPA[1] (Performance Evaluation Process Algebra). Podrobné fungovanie tohto systému ani tejto algebry nie je pre našu prácu podstatné. O fungovaní systému nepotrebujeme vedieť nič a algebru PEPA si nebudeme nijak predstavovať.

Systém, ktorý navrhujeme a implementujeme v tejto diplomovej práci má za úlohu spracovať súbory logov zo všetkých troch týchto technológii a uložiť ich do spoločnej databázy. Údaje, ktoré z logov dostaneme, potom budeme môcť ďalej filtrovať a analyzovať. Nad systémom je postavené REST API, ktoré umožňuje manipuláciu s údajmi (napr. pridávanie

[1] http://www.dcs.ed.ac.uk/pepa/about/

údajov vo formáte JSON, získanie informácií o stave systému atď.). Navrhnutý systém využíva open source riešenia a je nakonfigurovaný tak, aby zodpovedal požiadavkám v zadaní. Systém je jednoducho rozšíriteľný pomocou rôznych, neustále vznikajúcich, modulov a pluginov.

Kapitola 2

Popis problému a špecifikácie cieľov

2.1 Popis problému

Zo zadania problému vyplýva, že systém, používaný zadávateľmi, beží na viacerých počítačových staniciach. Tieto spolu komunikujú v rámci väčšieho celku. V rámci celého projektu si teda medzi sebou tieto stanice vymieňajú údaje, posielajú požiadavky a zapisujú reťazce do databázy. Opäť si pripomeňme, že podrobnosti o fungovaní tohto systému pre ciele tejto diplomovej práce nie sú podstatné. Tieto stanice využívajú tri vyššie zmienené technológie (Apache, JBoss a PostgreSQL). Na všetkých týchto staniciach prebieha neustále proces logovania. Tieto logy sa následne zhromažďujú na jednom mieste. V tomto momente vstupuje do procesu aj problém, ktorým sa zaoberá naša diplomová práca. Logy, ktoré budú periodicky pribúdať je potrebné rozanalyzovať a riadok za riadkom uložiť do databázy. Tieto informácie potom majú byť využiteľné na prehliadanie a filtráciu v rámci komplexného systému. Logy v jednotnom formáte budú slúžiť na vybudovanie špecifikácie systému v algebre PEPA. Riešenie musí podporovať operačný systém UNIX, pretože parsovanie, analýza aj prehliadanie získaných údajov budú prebiehať iba na serveroch s týmto operačným systémom. Používateľ bude mať možnosť vybrať si, ktoré údaje chce zobraziť či importovať. Implementácia má byť ďalej schopná analyzované údaje poskytovať na ďalšie spracovanie vo formáte JSON. Vhodným riešením by bolo nad systémom vytvoriť REST API, ktoré by zaisťovalo posielanie údajov v tomto formáte niekde ďalej. V neposlednom rade má byť celé riešenie modulárne a jednoducho rozšíriteľné tak, aby bolo schopné spracovať ďalšie formáty logov.

Po zhrnutí dostávame nasledujúci zoznam požiadaviek:

- podpora operačného systému na platforme UNIX,
- spracovanie logov z technológií Apache, JBoss a PostgreSQL,
- ukladanie údajov do databázy,
- práca s údajmi v tejto databáze podľa používateľských kritérií,
- import údajov na ďalšie spracovanie vo formáte JSON,

- REST API,
- modulárnosť, rozšíriteľnosť.

2.2 Špecifikácie cieľov

Prvou požiadavkou riešenia je podpora operačného systému UNIX. Implementácia by teda mala využívať multiplatformný programovací jazyk Java, respektíve riešenia na ňom postavené. Vďaka tomu prestane byť otázka operačného systému relevantná. Toto riešenie zabezpečí kompatibilitu aj s inými operačnými systémami (napríklad s operačným systémom Windows), ak by bola v budúcnosti potrebná. Vďaka programovaciemu jazyku Java budeme môcť jednoducho zaistiť aj modulárnosť a rozšíriteľnosť systému. Pomocou tohto jazyka nebude následne problém vytvoriť REST API na správu údajov vo formáte JSON.

V rámci implementácie musíme zaistiť periodické spracovanie všetkých logových súborov, ktoré sa budú objavovať v spoločnom úložisku (v zložke na disku servera, kde systém pobeží). Systém bude teda kontrolovať pribúdajúce nové logové súbory s určitou frekvenciou a následne ich spracuje. Pri spracovaní musí rozlíšiť, z akej technológie daný log pochádza a rozparsovať ho do databázy.

Keďže naše riešenie bude pracovať s viacerými typmi logových súborov, ktoré budú periodicky pribúdať, predpokladáme, že naša databáza bude pracovať s veľkým množstvom údajov. Musíme si uvedomiť, že logové súbory môžu za 24 hodín obsahovať radovo niekoľko tisíc riadkov. Náš systém bude spracovávať logy z troch systémov a viacerých počítačových staníc. To všetko kladie značné nároky na databázu. Preto od použitej technológie požadujeme schopnosť pracovať s veľkým množstvom údajov s rozumnou rýchlosťou. Takisto by mala byť dobre škálovateľná, keďže objem údajov a počet požiadaviek môže rýchlo narastať. Vidíme, že pre naše potreby bude kľúčová najmä rýchlosť zobrazovania uložených údajov. Preto budeme hľadať vhodnú databázovú technológiu typu key/value (kľúč/hodnota), ktorá potrebnú rýchlosť zaručí.

Celé naše riešenie sa točí okolo spracovania a analýzy logov. Bez analýzy totiž logové súbory nemajú prílišnú výpovednú hodnotu a dá sa povedať, že sú bezcenné. Takisto naše riešenie stráca zmysel, ak budeme mať údaje z logov spracované, uložené v databáze a budeme schopní ich rýchlo zobraziť, ale forma zobrazenia bude neprehľadná a neposkytne nám určité možnosti filtrovania. Aby sme z potenciálu, ktorý logové súbory ukrývajú, vyťažili čo najviac, potrebujeme vhodne zvoliť aj používateľské prostredie. Malo by nám umožniť nielen filtrovať a prezerať údaje v textovej forme, ale zároveň zobraziť tieto informácie aj v grafickej podobe. Grafy by pre určitý typ štatistických údajov poslúžili najlepšie.

Po zhrnutí dostávame nasledujúci zoznam cieľov:

- použité technológie zaručia multiplatformnosť,
- funkčné spracovanie logových súborov z technológií Apache, JBoss a PostgreSQL,
- architektúra navrhnutého riešenia umožní jeho jednoduché rozširovanie,

- použitá technológia databázy bude dostatočne svižná,
- databáza bude bez obmedzenia funkčnosti zvládať stredne veľké objemy údajov,
- REST API zabezpečí dodatočnú správu údajov,
- funkčné používateľské prostredie,
- možnosť grafického zobrazenia štatistík.

Finálna implementácia systému by tak predstavovala na mieru šité riešenie problému so spracovaním a následným analyzovaním logových súborov z troch rôznych technológií. Prínosom by mala byť hlavne komplexnosť spracovania s tým, že sa nezabúda na budúcnosť. Konkrétne by to boli multiplatformnosť, rýchlosť, prehľadnosť a škálovateľnosť celého systému. Ďalej by sme umožnili analýzu troch typov logových súborov v rámci jedného systému. Architektúra systému by potom umožnila jednoduché rozšírenie o ďalšie typy logových súborov.

Kapitola 3

Logovanie a analýza logov

3.1 Logovanie a typy logov

V tejto sekcii si povieme niečo o logovaní a o typoch logov, ktoré poznáme. Viac sa zameriame na serverové logy, ktoré sú pre našu prácu najzaujímavejšie. Ukážeme si aj príklady logových súborov a vysvetlíme si, ako najlepšie pochopiť ich obsah.

3.1.1 Čo je to logovanie?

Logovanie [23] je schopnosť aplikácie alebo operačného systému zaznamenať nejakú udalosť, ktorá nastala, pre neskoršiu analýzu systémovým administrátorom alebo analyzačným softvérom. Väčšina operačných systémov, softvérových frameworkov a používateľských programov obsahuje nejakú formu logovania. Niektoré systémové procesy a aplikácie využívajú vlastné logovacie služby a logujú do vlastných logovacích súborov, vo väčšine prípadov sa však využívajú služby špeciálneho podsystému, ktorého jedinou úlohou je prijímať správy od iných procesov a aplikácií a zaznamenávať ich do súborov spolu s rôznymi informáciami o danej udalosti. Takáto hierarchia väčšinou funguje v rámci operačného systému aj v rámci aplikácie či technológie (v rámci operačného systému sa o logovanie stará jeden démon, v rámci technológie webového servera jeden podsystém)

Výhodou tohto riešenia fakt, že logovacie súbory má na starosti jediný proces, ktorý kontroluje prístup k nim (má právo zápisu) a serializuje správy od jednotlivých procesov. Logovací proces môže správy vhodne filtrovať, kategorizovať alebo ich rozdeľovať do samostatných súborov, podľa toho, čo umožní jeho konfigurácia. Spúšťa sa pri štarte servera/aplikácie a mal by neustále bežať, pretože iba vtedy má celé logovanie význam.

3.1.2 Logy udalostí

Tieto logy [22] zaznamenávajú udalosti v rámci systému kvôli tomu, aby sme mohli vykonať následnú kontrolu systému, porozumieť jeho aktivitám a diagnostikovať prípadné problémy. Sú veľmi dôležité pri porozumení komplexných systémov, hlavne v prípadoch, že sa jedná o systémy s malou používateľskou interakciou (napríklad serverové aplikácie). Veľmi užitočné môže byť aj skombinovanie týchto logov z viacerých zdrojov. Tento prístup, spolu so štatistickou analýzou, môže odhaliť zdanlivo nesúvisiace udalosti na rôznych serveroch.

3.1.2.1 Logy transakcií

Väčšina databázových systémov využíva nejaký spôsob logovania transakcií. Tieto typy logov sú určené na neskoršiu kontrolu v rámci štatistickej analýzy a nie sú určené na čítanie pre človeka ako vidíme na obr. 3.1. Zaznamenávajú zmeny vykonané na uložených údajoch a umožňujú databázovým systémom obnoviť pôvodnú štruktúru údajov v prípade zlyhania transakcií, rôznych chýb či pádov celého systému. Pomáhajú teda udržiavať uložené údaje v konzistentnom stave. Databázové systémy väčšinou obsahujú logovanie všeobecných udalostí aj logovanie transakcií.

	Current LSN	Operation	Context	Transaction ID	AllocUnitId	AllocUnitName
3576	0000008a:00000198:0010	LOP_EXPUNGE_ROWS	LCX_CLUSTERED	0000:00000000	281474979397632	sys.syscolpars.clst
3577	0000008a:00000198:0011	LOP_EXPUNGE_ROWS	LCX_CLUSTERED	0000:00000000	281474979397632	sys.syscolpars.clst
3578	0000008a:00000198:0012	LOP_EXPUNGE_ROWS	LCX_CLUSTERED	0000:00000000	281474979397632	sys.syscolpars.clst
3579	0000008a:00000198:0013	LOP_EXPUNGE_ROWS	LCX_CLUSTERED	0000:00000000	281474979397632	sys.syscolpars.clst
3580	0000008a:00000198:0014	LOP_SET_BITS	LCX_PFS	0000:00000000	281474979397632	sys.syscolpars.clst
3581	0000008a:00000198:0015	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3582	0000008a:00000198:0016	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3583	0000008a:00000198:0017	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3584	0000008a:00000198:0018	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3585	0000008a:00000198:0019	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3586	0000008a:00000198:001a	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3587	0000008a:00000198:001b	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3588	0000008a:00000198:001c	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3589	0000008a:00000198:001d	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3590	0000008a:00000198:001e	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	562949956108288	sys.syscolpars.nc
3591	0000008a:00000198:001f	LOP_SET_BITS	LCX_PFS	0000:00000000	562949956108288	sys.syscolpars.nc
3592	0000008a:00000198:0020	LOP_EXPUNGE_ROWS	LCX_INDEX_LEAF	0000:00000000	844424932360192	sys.sysschobjs.nc2
3593	0000008a:00000198:0021	LOP_SET_BITS	LCX_PFS	0000:00000000	844424932360192	sys.sysschobjs.nc2

Obr. 3.1: Príklad logu transakcií v programe určenom na jeho prehliadanie – vidíme, že ani takto naformátovaný nedáva človeku veľký zmysel

3.1.3 Serverové logy

Serverový log je súbor (alebo viacero súborov) automaticky vytvorený a spravovaný serverom obsahujúci aktivitu, ktorá sa na serveri odohrala. Typickým príkladom je webový serverový log [35], ktorý spravuje históriu požiadaviek jednotlivých stránok. Organizácia W3C má pre webové serverové logy zavedený štandardný formát (Common Log Format – jeho príklad vidíme na obr. 3.2), ale existujú aj iné formáty. Neskoršie záznamy sú vkladané na koniec súboru. Typicky je to riadok obsahujúci informácie o požiadavke, IP adrese klienta, čase a dátume požiadavky, požadovanej stránke, kóde HTTP, používateľskom agente a URI, z ktorého bola stránka navštívená (tzv. referer). Tieto údaje môžu byť skombinované do jedného súboru alebo rozdelené do viacerých logov, ako sú napríklad log prístupov (access log), chybový log (error log) či log URI, z ktorých boli stránky navštívené (referrer log). Serverové logy zvyčajne nezbierajú informácie špecifické pre používateľa. Tieto súbory sú prístupné iba správcovi webových stránok. S pomocou použitia štatistickej analýzy môžeme v serverových logoch skúmať návštevnosť v priebehu dňa, v priebehu jednotlivých dní v týždni, rozdeliť prístupy podľa URI či používateľských agentov. Analýza webových logov je v správnych rukách veľmi mocným marketingovým nástrojom.

```
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "MKACTIVITY /sql/!svn/act/8db7ca3c-4a26-3243-8385
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/trunk/FolderOne HTTP/1.1" 207 438
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/!svn/vcc/default HTTP/1.1" 207 388
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "CHECKOUT /sql/!svn/bln/4 HTTP/1.1" 201 338
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPPATCH /sql/!svn/wbl/8db7ca3c-4a26-3243-8385-
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PROPFIND /sql/trunk/FolderOne HTTP/1.1" 207 404
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "CHECKOUT /sql/!svn/ver/4/trunk/FolderOne/SQLFile
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "PUT /sql/!svn/wrk/8db7ca3c-4a26-3243-8385-ff3ec9
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "MERGE /sql/trunk/FolderOne HTTP/1.1" 200 687
10.1.2.75 - - [21/Feb/2007:14:46:54 -0600] "DELETE /sql/!svn/act/8db7ca3c-4a26-3243-8385-ff3
```

Obr. 3.2: Príklad serverového logu zo serveru Apache vo formáte Common Log Format

3.1.4 Obsah serverových logov

Logové súbory rôznych webových serverov obsahujú rôzne informácie. Podľa autorov knihy *Advanced Computing* [2], základné informácie nachádzajúce sa v serverovom logu sú:

- Používateľské meno: predstavuje identifikátor používateľa, ktorý navštívil vaše webové stránky. Vo väčšine prípadov ide o IP adresu pridelenú používateľovi poskytovateľom internetových služieb (ISP). Môže ísť o dočasne pridelenú adresu a preto jednoznačná identifikácia používateľa chýba. V niektorých prípadoch je to vyriešené tak, že webové stránky vyžadujú vytvorenie používateľského profilu. Tento profil sa skladá z mena a hesla, prípadne doplnkových informácií, a používateľ, ktorý webové stránky navštívi znova môže byť unikátne identifikovaný.

- Cesta návštevy: vyjadruje cestu, ktorou sa používateľ na webové stránky dostal. Môže to byť priamym zadaním adresy URL, kliknutím na odkaz alebo prostredníctvom vyhľadávačov.

- Traverzová cesta: zachytáva pohyb používateľa v rámci webových stránok používaním jednotlivých odkazov.

- Časový odtlačok: je čas, ktorý používateľ strávil na jednotlivých stránkach počas surfovania webu. Označuje sa ako relácia.

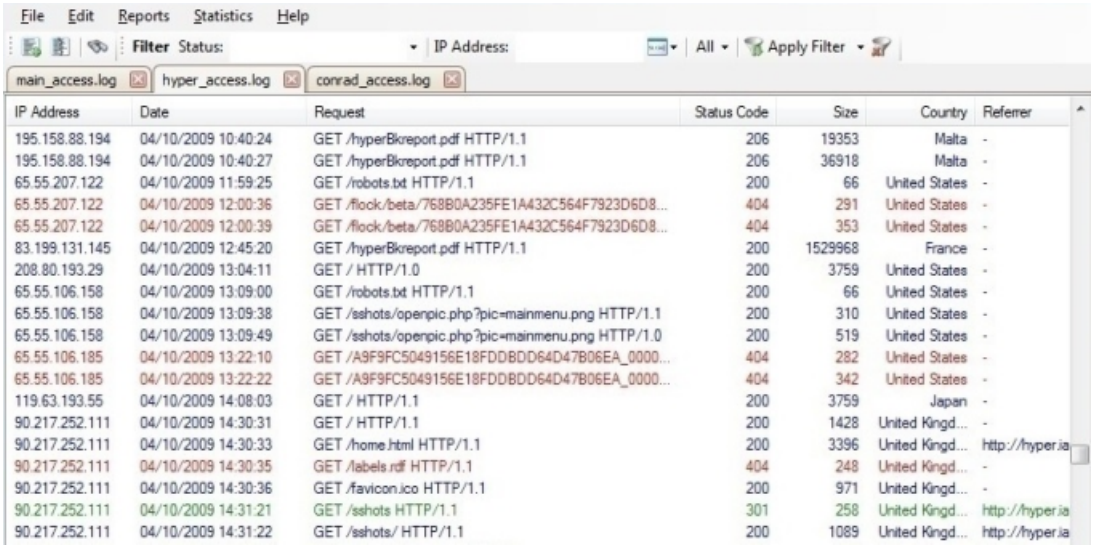
- Posledná navštívená stránka: je stránka, ktorú používateľ navštívil ako poslednú pred opustením webových stránok.

- Miera úspešnosti: predstavuje počet stiahnutí a vykonaní iných akcií používateľom (napríklad nákup tovaru alebo aplikácie). Tento ukazateľ má význam hlavne v marketingu a označuje sa aj ako miera konverzie.

- Používateľský agent: nepredstavuje vo väčšine prípadov nič iné ako internetový prehliadač, z ktorého používateľ odoslal požiadavku na webový server. Ide o textový reťazec popisujúci použitý typ a verziu použitého prehliadača.

- URL: zdroj, kam používateľ požadoval prístup. Môže ísť napríklad o stránku HTML, protokol CGI alebo skript.

- Typ požiadavky: metóda, ktorou bol začatý prenos informácií. Ide o metódy HTTP ako napríklad GET, POST atď.
- Kód odpovedi: ide o číselný kód HTTP, ktorý vyjadruje odpoveď na požiadavku. Medzi najbežnejšie kódy patria 200 – OK, 403 – Forbidden (zakázané) či 404 – Not Found (nenájdené)[17].



IP Address	Date	Request	Status Code	Size	Country	Referrer
195.158.88.194	04/10/2009 10:40:24	GET /hyperBkreport.pdf HTTP/1.1	206	19353	Malta	-
195.158.88.194	04/10/2009 10:40:27	GET /hyperBkreport.pdf HTTP/1.1	206	36918	Malta	-
65.55.207.122	04/10/2009 11:59:25	GET /robots.txt HTTP/1.1	200	66	United States	-
65.55.207.122	04/10/2009 12:00:36	GET /flock/beta/768B0A235FE1A432C564F7923D6D8...	404	291	United States	-
65.55.207.122	04/10/2009 12:00:39	GET /flock/beta/768B0A235FE1A432C564F7923D6D8...	404	353	United States	-
83.199.131.145	04/10/2009 12:45:20	GET /hyperBkreport.pdf HTTP/1.1	200	1529968	France	-
208.80.193.29	04/10/2009 13:04:11	GET / HTTP/1.0	200	3759	United States	-
65.55.106.158	04/10/2009 13:09:00	GET /robots.txt HTTP/1.1	200	66	United States	-
65.55.106.158	04/10/2009 13:09:38	GET /sshots/openpic.php?pic=mainmenu.png HTTP/1.1	200	310	United States	-
65.55.106.158	04/10/2009 13:09:49	GET /sshots/openpic.php?pic=mainmenu.png HTTP/1.0	200	519	United States	-
65.55.106.185	04/10/2009 13:22:10	GET /A9F9FC5049156E18FDDBDD64D47B06EA_0000...	404	282	United States	-
65.55.106.185	04/10/2009 13:22:22	GET /A9F9FC5049156E18FDDBDD64D47B06EA_0000...	404	342	United States	-
119.63.193.55	04/10/2009 14:08:03	GET / HTTP/1.1	200	3759	Japan	-
90.217.252.111	04/10/2009 14:30:31	GET / HTTP/1.1	200	1428	United Kingd...	-
90.217.252.111	04/10/2009 14:30:33	GET /home.html HTTP/1.1	200	3396	United Kingd...	http://hyper.ia
90.217.252.111	04/10/2009 14:30:35	GET /labels.rdf HTTP/1.1	404	248	United Kingd...	-
90.217.252.111	04/10/2009 14:30:36	GET /favicon.ico HTTP/1.1	200	971	United Kingd...	-
90.217.252.111	04/10/2009 14:31:21	GET /sshots HTTP/1.1	301	258	United Kingd...	http://hyper.ia
90.217.252.111	04/10/2009 14:31:22	GET /sshots/ HTTP/1.1	200	1089	United Kingd...	http://hyper.ia

Obr. 3.3: Príklad formátovaného serverového logu zo serveru Apache v programe Apache Log Viewer

3.1.5 Ako čítať logy?

Ako vidíme na obr. 3.2, logové súbory nie sú veľmi prehľadné. Naformátovanie logových súborov (obr. 3.3) nám síce uľahčí ich čítanie, nezaručí nám ale, že im porozumieme. Aby sme skutočne vedeli, čo logové súbory obsahujú, musíme rozumieť jednotlivým údajom, ktoré sa v nich môžu objaviť. Následne si tieto údaje môžeme začať spájať do zmysluplných informácií. Na nasledujúcom príklade, ktorý sme našli na webových stránkach *jakpsatweb.cz* [34], si ukážeme, ako by mohlo vyzerať získavanie zmysluplných informácií z logov. Aby sme zachovali dobrú čitateľnosť, sú pôvodné záznamy zalomené a rozdelené na dva riadky. Jeden záznam (pôvodne jeden riadok) v našom príklade obsahuje dátum a čas, IP adresu klienta, metódu HTTP požiadavky, požadované URL (to je relatívne v rámci lokality), kód odpovedi a odtlačok prehliadača (používateľského agenta). Kúsok logu:

```
2003-09-02 14:58:29 62.245.90.159 GET /pozadie.html 200
Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+.NET+CLR+1.1.4322)

2003-09-02 14:58:32 160.218.144.158 GET /javascript/priklady/skr_zalozky.html 200
Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+98;+Win+9x+4.90)
```

```
2003-09-02 14:59:09 194.228.206.162 GET /weblog/my_weblog.xml 304 Feedreader

2003-09-02 14:59:17 62.229.222.18 GET /images/dp.png 304
Mozilla/5.0+(X11;+U;+Linux+i686;+en-US;+rv:1.0.2)+Gecko/20030708
```

Príklad ukazuje štyri prístupy z 2. septembra 2003 popoludní. Z IP adresy 62.245.90.159 niekto metódou GET požiadal o stránku pozadie.html pomocou prehliadača Microsoft Internet Explorer 6.0 a dostal ju (kód HTTP 200 znamená OK). O chvíľu neskôr sa niekto iný (z IP adresy 160.218.144.158) pozeral na príklady skrývaných záložiek. Tretí záznam urobila nejaká čítačka RSS (Freereader), zistila ale, že požadovaný súbor /weblog/my_weblog.xml sa nezmenil (kód HTTP 304 znamená Not Modified – nezmenené), a preto ho nesťahovala. Takisto sa nezmenil obrázok /images/dp.png, na ktorý sa niekto pozeral z prehliadača Mozilla Firefox (renderovacie jadro Gecko) na operačnom systéme Linux.

3.2 Spracovanie logov

V tejto sekcii si povieme niečo o spracovaní logov. Pôjde o analýzu webových logov, teda proces skúmania záznamov udalostí, ktoré prebehli a získanie relevantných informácií z týchto záznamov. Budeme sa podrobnejšie venovať tomu, čo všetko sa dá logovať v technológiách, ktoré sú pre nás v tejto práci podstatné. Na základe toho zistíme, aké sú naše možnosti pri návrhu a implementácii riešenia. Dozvieme sa, aké štatistiky z logových súborov môžeme zostaviť a čo všetko z nich vieme zistiť.

3.2.1 Analýza webových logov

S nárastom popularity internetu a vznikom nových spôsobov jeho používania narastá aj potreba merania výkonnosti jednotlivých webových stránok tak presne, ako je to len možné [1]. V súčasnosti nestačí presne odmerať počet návštevníkov daných webových stránok. Dôležité je zistiť správanie návštevníkov na jednotlivých stránkach a vztiahnuť toto správanie na vytýčené ciele. V predchádzajúcej sekcii sme si predstavili, čo všetko je dnes možné pomocou logovania zaznamenávať. Analýza webových logov je ďalším krokom na ceste za relevantnými informáciami. Je to štúdium logových súborov z konkrétnych webových stránok. Dôvodom je pomocou softvéru na analýzu webových logov zmerať výkonnosť daných webových stránok, vytiahnuť z logov užitočné informácie a následne ich prezentovať v nespočetnom množstve užitočných foriem. Toto pomerne mladé odvetvie má natoľko interdisciplinárny charakter, že jeho výsledky sú užitočné pre všetkých, ktorí pracujú s webovými stránkami. Práca alebo zábava, učenie sa či socializácia, doma či na cestách, jednotlivo či v skupinách. Weboví používatelia sú vo všetkých týchto situáciách obklopení infraštruktúrou zariadení, sietí a aplikácií. Táto infraštruktúra spolu s neustále rastúcim množstvom všetkých typov informácií, ktoré si vieme predstaviť stimuluje intelektuálnu a psychickú aktivitu používateľov. Či už používatelia hľadajú, používajú, vytvárajú či získavajú informácie, nechávajú za sebou veľké množstvo údajov o ich informačných postojoch, náladách a potrebách. Práve preto môžu analýzu webových logov využiť nielen systémoví administrátori, ale aj obchodníci, marketingoví profesionáli či dizajnéri nových webov.

Prístupov a metód ako to urobiť je veľa a neustále vznikajú nové. Podľa autorov knihy *Handbook of Research on Web Log Analysis* [1] sú toto tie základné:

- Konceptuálne frameworky: Koncepty sú predstavené, definované a následne použité na vytvorenie konceptuálnych frameworkov, ktoré potom poskytujú smery štúdia údajov.
- Fenomenológia/Etnometodológia: Interpretačná metodológia, ktorá skúma správanie používateľov. Etnometodológia, ako rozšírenie fenomenológie, skúma interakcie jednotlivcov a skupín v rámci sociálnych štruktúr.
- Obsahová analýza: metodická a opakovateľná metodológia, ktorá určuje, kvantifikuje a analyzuje prítomnosť skúmaných objektov v rámci veľkých sád údajov.
- Etnografia: Kvalitatívna štúdia, v ktorej výskumník dlhší čas pozoruje členov určitej skupiny v ich prirodzenom prostredí.
- Historická metóda: Zbiera a skúma fakty, ktoré pochádzajú z minulosti, o udalostiach, ľuďoch a prostrediach.
- Analýza rozpravy: Vedecká metóda vyhodnocovania na základe argumentov.
- Prípadové štúdie: Komplexné štúdie jedného subjektu, ovplyvnené vhodným výberom analyzovaných skutočností.

3.3 Možnosti logovania – Apache

Server HTTP Apache je jeden z najsilnejších, ak nie najsilnejšie, open source riešení pre webové operácie. Apache poskytuje veľmi komplexné flexibilné možnosti logovania. Na nasledujúcich riadkov si s pomocou stránky s dokumentáciou najnovšej verzie tohto webového servera [3] vysvetlíme, čo všetko je tu možné logovať a ako to nakonfigurovať.

Apache totižto poskytuje paletu rôznych mechanizmov na logovanie všetkého, čo sa na serveri udeje, od prvotnej požiadavky, cez proces mapovania URL až po nadviazanie spojenia. Samozrejmosťou je logovanie chýb, ktoré v rámci tohto procesu môžu nastať. Naviac ponúka možnosť využiť moduly tretích strán, ktoré budú samy vykonávať logovanie alebo vložia záznamy do existujúcich logových súborov servera a aplikácie, ako napríklad programy CGI, skripty PHP a iné, ktoré môžu posielať správy do chybového logu (error log) servera.

3.3.1 Chybový log (error log)

Chybový log je najdôležitejším logovým súborom na serveri. Je to miesto, kam bude Apache httpd posielať diagnostické informácie a zaznamenávať akékoľvek chyby, ktoré nastanú pri spracovávaní požiadaviek. Je to prvé miesto, kam sa pozrieť, ak nastane problém pri štarte servera alebo serverovej operácii. Chybový log bude obsahovať podrobnosti problému a informácie, ako ho napraviť.

Chybový log je zapisovaný do súboru, ktorý sa zvyčajne nazýva *error_log* na unixových systémoch a *error.log* na systémoch Windows. Na unixových systémoch je taktiež možné posielať chyby systémovému procesu *syslog*.

Formát chybového logu je definovaný direktívou *ErrorLogFormat*, s pomocou ktorej si môžeme prispôsobiť, ktoré hodnoty sa budú logovať. V predvolenom formáte bude typický záznam logu vyzerať nasledovne (samozrejme bude na jednom riadku):

```
[Fri Sep 09 10:42:29.902022 2011] [core:error] [pid 35708:tid 4328636416]
[client 72.15.99.187] File does not exist: /usr/local/apache2/htdocs/icon.ico
```

Prvou položkou záznamu je dátum a čas správy. Ďalšou je modul, ktorý správu vyprodukoval (v tomto prípade je to core) a úroveň severity správy. Nasleduje ID procesu a ID vlákna, v ktorom proces bežal. Predposlednou položkou je adresa klienta, ktorý odoslal požiadavku a na záver vidíme podrobnú chybovú správu, ktorá v tomto prípade indikuje odoslanie požiadavky na neexistujúci súbor.

V logu sa môžu objaviť rôzne správy, avšak väčšina z nich bude vyzerať podobne, ako príklad uvedený vyššie. Chybový log môže taktiež obsahovať výstup ladenia od CGI skriptov. Všetky informácie, odoslané CGI skriptom na *stderr* sa skopírujú priamo do chybového logu.

Pripojenie tokenu %L do chybového logu aj prístupového logu (access log) vygeneruje ID záznamu logu, vďaka ktorému môžeme ľahšie priradiť záznamy z chybového logu k záznamom z prístupového logu.

3.3.2 Prístupový log (access log)

Prístupový log servera zaznamenáva všetky požiadavky spracované serverom. Umiestnenie a obsah prístupového logu sú kontrolované *CustomLog* direktívou. Direktíva *LogFormat* môže byť použitá na zjednodušenie výberu toho, čo má log obsahovať. Samotný formát logu je do značnej miery konfigurovateľný. Formát je špecifikovaný pomocou formátovacích reťazcov, ktoré sú podobné formátovacím reťazcom *printf* [6], používaných v programovacom jazyku C++. Na nasledujúcich riadkoch si predstavíme najviac používané príklady. Kompletný zoznam formátovacích reťazcov nájdete v tabuľke 3.1.

3.3.2.1 Common Log Format

Typická konfigurácia prístupového logu potom vyzerá nasledovne:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

Tento kúsok kódu definuje názov konfigurácie *common*. Formátovacie reťazce serveru presne povedia, aké informácie má zaznamenať. Konfigurácie uvedená vyššie bude zapisovať do logu záznamy vo formáte Common Log Format (CLF). Tento štandardný formát dokáže produkovať veľa webových serverov a prečíta ho väčšina programov na analýzu logov. Záznamy v logu vyprodukované vo formáte CLF vyzerajú nasledovne:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif
HTTP/1.0" 200 2326
```

Jednotlivé časti záznamu si bližšie vysvetlíme:

- **127.0.0.1 (%h)** – IP adresa klienta, ktorý poslal požiadavku na server. Ak je zapnuté nastavenie *HostnameLookups*, bude sa server namiesto IP adresy snažiť zistiť a zalogovať názov hostiteľa. Táto konfigurácia však môže výrazne spomaliť server. Názvy hostiteľa je lepšie zisťovať pri neskoršom spracovaní logov. Táto IP adresa nemusí byť nutne adresou zariadenia, ktoré používateľ používa. V prípade, že sa na ceste medzi zariadením používateľa a serverom nachádza proxy server, zaloguje sa práve IP adresa proxy servera.

- **- (%l)** – Pomlčka na výstupe znamená, že príslušná informácia nie je k dispozícii. V tomto prípade ide o RFC 1413 identitu klienta, zisťovanú prostredníctvom položky *identd* na zariadení klienta. Táto informácia je vysoko nespoľahlivá a nemala by sa, až na prípady prísne kontrolovaných interných sietí, používať.

- **frank (%u)** – Používateľské ID (userid) osoby, ktorá požadovala príslušný dokument, určené na základe overenia totožnosti HTTP. Ak má kód stavu požiadavky hodnotu 401 (viď. nižšie), tejto hodnote nie je možné veriť, pretože používateľ ešte nebol overený. Ak dokument nie je chránený heslom, aj v tejto časti sa zobrazí pomlčka.

- **[10/Oct/2000:13:55:36 -0700] (%t)** – Čas prijatia požiadavky. Formát je:

  ```
  [deň/mesiac/rok:hodina:minúta:sekunda časové pásmo]
  ```

  - deň – 2 číslice,
  - mesiac – 3 písmená,
  - rok – 4 číslice,
  - hodina – 2 číslice,
  - minúta – 2 číslice,
  - sekunda – 2 číslice,
  - časové pásmo – (+/-) 4 číslice.

- **"GET /apache_pb.gif HTTP/1.0"("%r")** – Riadok požiadavky od klienta v dvojitých úvodzovkách. Tento riadok obsahuje veľké množstvo informácií. Po prvé, klientom použitá metóda je GET. Po druhé, klient požiadal o zdroj */apache_pb.gif* a potretie, klient používa protokol HTTP/1.0.

- **200 (%>s)** – Kód stavu, ktorý server odoslal späť klientovi. Táto informácia je veľmi hodnotná, keďže nám ukazuje, či požiadavka bola úspešne zodpovedaná (kódy stavu začínajúce číslicou 2), presmerovaná (kódy stavu začínajúce číslicou 3), skončila chybou na strane klienta (kódy stavu začínajúce číslicou 4) alebo skončila chybou na strane servera (kódy stavu začínajúce číslicou 5).

- **2326 (%b)** – Posledná časť nám ukazuje veľkosť objektu vráteného klientovi v bajtoch (veľkosť nezahŕňa hlavičky odpovede). Ak server nevrátil klientovi žiaden obsah, hodnota bude „-“.

3.3.2.2 Combined Log Format

Ďalším rozšíreným formátom je Combined Log Format. Vyzerá nasledovne:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined
CustomLog log/access_log combined
```

Môžeme si všimnúť, že ide o totožný formát ako Common Log Format s dvoma pridanými poliami. Oba tieto formátovacie reťazce používajú zápis *%{header}i*, kde *header* znamená akúkoľvek hlavičku požiadavky HTTP. Záznam v prístupovom potom vyzerá takto:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0"
200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

Pridané polia sú:

- **"http://www.example.com/start.html"("%{Referer}i")** – Hlavička požiadavky HTTP typu Referer (URI, z ktorého bola stránka navštívená). Poskytuje informáciu, odkiaľ bolo na navštívenú stránku pristúpené (mala by to byť stránka, ktorá obsahuje alebo odkazuje na súbor */apache_pb.gif*).
- **"Mozilla/4.08 [en] (Win98; I ;Nav)"("%{User-agent}i")** – Hlavička požiadavky HTTP typu User-agent (používateľský agent). Poskytuje nám informácie, ktoré o sebe prezrádza klientsky prehliadač.

Formátovacie reťazce z tabuľky 3.1 môžu byť obmedzené, aby zaznamenávali iba odpovede s určitými špecifickými kódmi stavov HTTP. Dosiahneme to umiestnením zoznamu kódov stavov, oddelených čiarkou, za znak %. Umiestnením znaku vyjadríme, že ide o negáciu. Príklady modifikátorov nájdeme v tabuľke 3.2.

3.3.3 Rotovanie logov

Aj na priemerne zaneprázdnenom serveri, kvantita informácií, ktorá je ukladaná do logových súborov, veľmi rýchlo rastie. Každých 10 000 záznamov znamená približne 1 MB. Preto je potrebné periodicky logové súbory rotovať – existujúce logy odstrániť alebo presunúť. To nie je možné urobiť, pokiaľ server stále beží (log je otvorený kvôli zápisu). Server je potrebné reštartovať, aby po presunutí alebo odstránení starých súborov vytvoril nové. Apache poskytuje možnosť tzv. elegantného reštartu (graceful restart). Táto funkcia umožní otvoriť nové logové súbory bez straty existujúcich alebo čakajúcich spojení od klientov. Aby to však bolo možné, server musí dopísať požiadavky spred reštartu do starých súborov. Až potom otvorí nové logy, a preto je potrebné nechať po tomto druhu reštartu staré logy ešte chvíľu na pokoji. Rotovanie logov jednoducho zapíšeme takto:

```
mv access_log access_log.old
mv error_log error_log.old
apachectl graceful
sleep 600
gzip access_log.old error_log.old
```

Server Apache toho dokáže ešte oveľa viac. Pre úplnosť odporúčame prečítať si stránky s dokumentáciou [3].

3.4 Možnosti logovania – JBoss

V aplikačnom serveri JBoss je celková konfigurácia logovania reprezentovaná logovacím subsystémom [20]. Ten pozostáva zo štyroch podstatných častí: konfigurácie handlera, logger, root logger (konfigurácie logov) a logovacie profily. Každý logger sa odvoláva na handler (alebo skupinu handlerov) a každý handler deklaruje formát logu a jeho výstup:

```
<subsystem xmlns="urn:jboss:domain:logging:1.0">
   <console-handler name="CONSOLE" autoflush="true">
       <level name="DEBUG"/>
       <formatter>
           <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
       </formatter>
   </console-handler>
   <periodic-rotating-file-handler name="FILE" autoflush="true">
       <formatter>
           <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
       </formatter>
       <file relative-to="jboss.server.log.dir" path="server.log"/>
       <suffix value=".yyyy-MM-dd"/>
   </periodic-rotating-file-handler>
   <logger category="com.arjuna">
       <level name="WARN"/>
   </logger>
   [...]
   <root-logger>
       <level name="DEBUG"/>
       <handlers>
           <handler name="CONSOLE"/>
           <handler name="FILE"/>
       </handlers>
   </root-logger>
</subsystem>
```

3.4.1 Konfigurácie logovania

Logovanie v aplikačnom serveri JBoss má široké možnosti konfigurácie. Používaný je napríklad open source framework Log4J. Celkovo môžeme konfiguráciu logovania nastavovať až v piatich rôznych súboroch:

- logging.properties
- jboss-logging.properties
- log4j.properties

- log4j.xml
- jboss-log4j.xml

Môžeme sa rozhodnúť, či tieto konfigurácie budeme používať iba pre jednotlivé nasadenia (per-deployment logging) alebo v rámci celého systému.

3.4.2 Logovacie profily

JBoss nám dovoľuje používať rôzne logovacie profily. Každý profil je ako nový logovací subsystém skladajúci sa z konfigurácii handlera, loggera a deklarácií root loggera. Logovací profil môže byť priradený k viacerým nasadeniam. používanie profilov umožňuje aj zmeny v konfiguráciách logovania za behu servera, bez nutnosti opätovného nasadenia.

3.4.3 Popis logovacieho subsystému

V nasledujúcej sekcii si stručne popíšeme jednotlivé handlery a následne si povieme, aké sú jednotlivé logovacie úrovne.

Zoznam jednotlivých handlerov

- async-handler – handler, ktorý píše sub-handlerom v asynchrónnom vlákne,
- console-handler – handler, ktorý píše do konzoly,
- custom-handler – definuje vlastný logovací handler,
- file-handler – handler, ktorý zapisuje do súboru,
- logger – definuje kategóriu loggera,
- periodic-rotating-file-handler – handler, ktorý zapisuje do súboru a periodicky logový súbor rotuje (podľa nastavenej periódy),
- root-logger – definuje root logger pre daný obsah,
- size-rotating-file-handler – ktorý zapisuje do súboru a logový súbor rotuje pri prekročení určitej nastavenej veľkosti logu,
- syslog-handler – definuje syslog handler.

JBoss využíva 6 základných úrovní logovania [19]:

- FATAL – Táto úroveň označuje kritické zlyhania služieb. Ak služba zaznamená tento druh chyby, je neschopná akokoľvek obsluhovať požiadavky.
- ERROR – Táto úroveň označuje narušenie požiadavky alebo schopnosti obsluhovať požiadavky. Služba by však mala byť schopná v prítomnosti chýb tohto typu aj naďalej v obmedzenej kapacite obsluhovať požiadavky.

- WARN – Nekritické chyby služieb. Do tejto kategórie spadajú chyby, ktoré umožňujú službe ďalej pokračovať, očakávania a jemné narušenia fungovania. Existuje veľmi tenká hranica medzi chybami typu WARN a chybami typu ERROR.
- INFO – Udalosti životného cyklu a ďalšie dôležité informácie. Používajú sa hlavne na oboznámenie sa so stavom služieb.
- DEBUG – Typy správ, ktoré obsahujú informácie o udalostiach životného cyklu servera. Väčšinou majú význam pre vývojárov a obsahujú hĺbkové informácie potrebné pre ladenie a podporu. Správy typu DEBUG a INFO nám presne povedia, v akom stave sa služba nachádza a aké serverové zdroje používa (porty, rozhrania, logové súbory atď.).
- TRACE – Správy, ktoré sú priamo spojené s príslušnými požiadavkami. Používajú sa na skutočne hlboké preniknutie do fungovania servera. Logovať sa bude každá jedna maličkosť. Mali by sme byť pripravení na rapídne zväčšovanie sa logového súboru.

3.4.4 Prístupový log

JBoss je v možnostiach logovania veľmi flexibilný a ponúka nám aj vytvorenie klasického prístupového logového súboru [18]. Čo všetko je možné doň zapísať nám ukazuje tabuľka 3.3

3.5 Možnosti logovania – PostgreSQL

Aj databázová technológia PostgreSQL nám ponúka široké spektrum možnosti logovania. Konfigurácia celého procesu prebieha v súbore *postgresql.conf* pomocou rôznych parametrov. Tieto parametre slúžia ako prepínače (vypnuté/zapnuté) alebo nadobúdajú príslušnú množinu hodnôt. Výsledné fungovanie logovania je kombináciou týchto parametrov. Na parametre sa teraz pozrieme bližšie.

3.5.1 Kam logovať

logging_collector(boolean) Tento parameter zapína proces bežiaci na pozadí, ktorý zachytáva logové správy posielané na výstup stderr a presmerováva ich do logovacích súborov. Je to základný parameter na vytváranie klasických logov v PostgreSQL. Nastavuje sa pri štarte servera.

log_destination(string) PostgreSQL podporuje niekoľko metód logovania serverových správ, vrátane výstupov stderr, csvlog a syslog. Na operačnom systéme Windows je podporovaný aj eventlog. Tento parameter nastavujeme ako zoznam destinácií logovania, oddelených čiarkou.

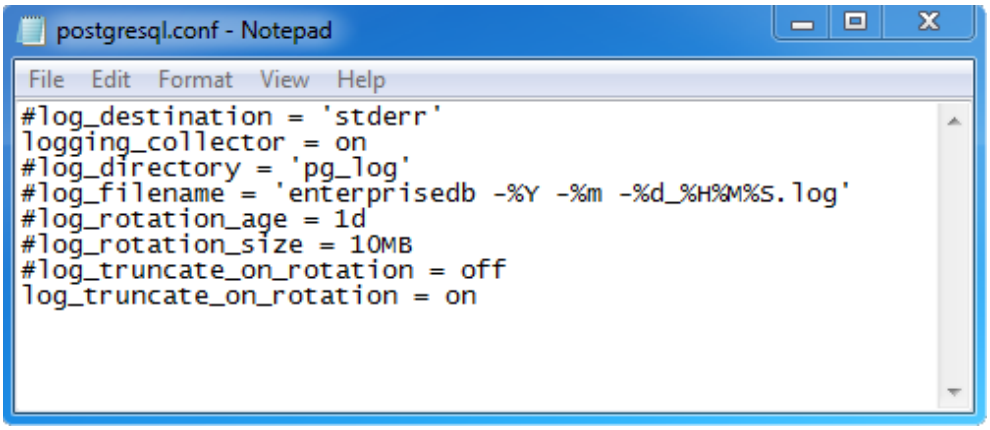
log_directory(string) Tento parameter určuje, v akej zložke budú vytvorené logy. Môže byť zadaný ako relatívna alebo absolútna cesta.

log_filename(string) Tento parameter určuje názvy vytvorených súborov. Hodnota je považovaná za reťazec špecifikácie *strftime* [36]. Podporované formátovacie reťazce sú veľmi podobné tým uvedeným v špecifikácii. Vďaka tomu môžeme vytvoriť názvy súborov s automaticky meniacou sa časovou zložkou.

log_rotation_age(integer) Parameter určujúci maximálnu dĺžku života individuálneho logového súboru. Po uplynutí uvedených minút je vytvorený nový logový súbor. Ak je hodnota 0, vytváranie nových logov na základe dĺžky ich existencie je vypnuté.

log_rotation_size(integer) Parameter určujúci maximálnu veľkosť individuálneho logového súboru. Po zapísaní uvedených kilobajtov je vytvorený nový logový súbor. Ak je hodnota 0, vytváranie nových logov na základe veľkosti je vypnuté.

log_truncate_on_rotation(boolean) tento parameter spôsobí, že PostgreSQL bude prepisovať (namiesto napájania) akýkoľvek existujúci log s rovnakým názvom. Prepísanie však nastane iba v prípade, že nový log je otvorený kvôli rotácii na základe dĺžky existencie logu. Príklad: Ak napríklad chceme vytvoriť 7 logových súborov, pomenovaných podľa dní (server_log.Mon, server_log.Tue atď.) a automaticky prepisovať minulotýždňové logy novými, nastavíme *log_filename* na hodnotu *server_log.%a*, *log_truncate_on_rotation* na hodnotu *on* a *log_rotation_age* na hodnotu *1440*.



Obr. 3.4: Príklad konfigurácie logovania v súbore postgresql.conf

Takto nakonfigurované logy si potom môžeme zobraziť na 4 rôzne výstupy. Výstup stderr je nastavený ako predvolený. Výstup eventlog je podporovaný iba na operačných systémoch Windows [32].

3.5.1.1 eventlog

- predvolené posielanie správ typu štart/stop,
- jednoduchá úprava súboru postgresql.conf,
- reštart služieb pomocou menu Služby,
- nástroj Event Viewer na zobrazovanie udalostí,
- iba pre OS Windows.

3.5.1.2 stderr

- veľmi jednoduché spravovanie,
- Postgre zvláda rotáciu logov automaticky,
- potreba manuálne odosielať logy na centrálny server,
- potreba manuálne čistiť logové súbory.

3.5.1.3 csvlog

- Postgre zvláda rotáciu logo automaticky,
- CSV - rýchle a jednoduché načítavanie do databázy,
- potreba manuálne odosielať logy na centrálny server,
- potreba manuálne čistiť logové súbory,
- logy prestávajú byť čitateľné,
- dostupné polia nemusia byť dostačujúce.

3.5.1.4 syslog

- jednoduchá centralizácia logov,
- flexibilné možnosti prostredníctvom nástroja syslog-nf,
- potreba prístupových práv k súboru syslog.conf,
- potreba manuálne vykonávať rotáciu logov.

3.5.2 Kedy logovať

V PostgreSQL existujú tieto úrovne logovania: DEBUG, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC. Podrobne nám ich predstavuje tabuľka 3.4. Tieto úrovne fungujú s nasledujúcimi parametrami.

client_min_messages(enum) Tento parameter kontroluje, správy akej úrovne sa posielajú používateľovi. Každá úroveň obsahuje úrovne, ktoré ju nasledujú (podľa poradia uvedeného vyššie). Čím neskôr v poradí sa úroveň nachádza, tým menej správ je posielaných. Predvolená hodnota je NOTICE.

log_min_messages(enum) Tento parameter kontroluje, správy akej úrovne sa posielajú do serverového logu. Každá úroveň obsahuje úrovne, ktoré ju nasledujú (podľa poradia uvedeného vyššie). Čím neskôr v poradí sa úroveň nachádza, tým menej správ je posielaných. Predvolená hodnota je WARNING.

log_error_statement(enum) Tento parameter kontroluje, aké chybové SQL príkazy sa zaznamenávajú do serverového logu. Každá úroveň obsahuje úrovne, ktoré ju nasledujú

(podľa poradia uvedeného vyššie). Čím neskôr v poradí sa úroveň nachádza, tým menej správ je posielaných. Predvolená hodnota je ERROR.

log_min_duration_statement(integer) Tento parameter spôsobí, že trvanie všetkých dokončených príkazov, ktoré dosiahne aspoň uvedenú hodnotu v milisekundách, bude zaznamenané. Hodnota 0 zaznamená všetky príkazy. Hodnota -1 naopak vypne zaznamenávanie všetkých príkazov. Ak napríklad nastavíme tento parameter na hodnotu 250 ms, všetky SQL príkazy trvajúce aspoň 250 ms sa zaznamenajú.

3.5.3 Čo logovať

Možnosti logovania v PostgreSQL je veľmi veľa. Aj Postgre nám napríklad poskytuje možnosť logovať pomocou formátovacích reťazcov, zaznamenávať časové pásmo, odkiaľ požiadavka prišla či samotné znenie požiadavky. My si predstavíme len tie najzaujímavejšie, zvyšok nájdeme v dokumentácii [31].

log_connections(boolean) Tento paramater zaznamenáva všetky pokusy o pripojenie na server a zároveň úspešne dokončené overenia klientov. Nemôže byť zmenený po štarte relácie.

log_disconnections(boolean) Tento paramater zaznamenáva všetky ukončenia relácie a zároveň dĺžku ich trvania. Nemôže byť zmenený po štarte relácie.

log_duration(boolean) Tento paramater po zapnutí zaznamenáva všetky dokončené príkazy.

log_statement(enum) Tento parameter určuje, ktoré príkazy SQL sa zaznamenajú. Prípustné hodnoty sú *none*, *ddl*, *mod* a *all*. Hodnota *ddl* loguje všetky príkazy definície údajov, ako napríklad *CREATE, ALTER* a *DROP*. Pri nastavení hodnoty na *mod* sa logujú príkazy typu *ddl* a zároveň príkazy modifikujúce údaje, ako napríklad *INSERT, UPDATE, DELETE, TRUNCATE* a *COPY FROM*. Analogicky, príkaz *all* potom zaznamenáva úplne všetky typy príkazov SQL.

log_line_prefix(string) Tento paramater zaznamenáva výstup naformátovaný ako funkcia printf, teda ako množina za sebou idúcich formátovacích reťazcov. Rozpoznané reťazce sa nahradia informáciou, tie nerozpoznané sa ignorujú. Parameter sa dá nastaviť v súbore *postgresql.conf*. Viac informácií nájdeme v tabuľke 3.5.

Formátovací reťazec	**Popis**
%%	Znak %
%a	IP adresa klienta, ktorý odoslal požiadavku
%{c}a	IP adresa pripojenia základného peera
%A	Lokálna IP adresa
%B	Veľkosť odpovede v bajtoch, bez hlavičiek HTTP
%b	Veľkosť odpovede v bajtoch, bez hlavičiek HTTP vo formáte CLF (t. j. v prípade, že sa neodoslali žiadne bajty sa namiesto „0“ zobrazí „-“)
%{VARNAME}C	Obsah súboru cookie s názvom VARNAME v požiadavke zaslanej na server. Iba súbory cookie verzie 0 sú plne podporované
%D	Čas potrebný na obslúženie požiadavky, v milisekundách
%{VARNAME}e	Obsah premennej prostredia s názvom VARNAME
%f	Názov súboru
%h	Názov vzdialeného hostiteľa. Zaznamená IP adresu, ak je možnosť *HostnameLookups* nastavená na *Off)*. Takto je táto možnosť nastavená predvolene.
%H	Protokol požiadavky
%{VARNAME}i	Obsah premennej s názvom VARNAME: riadky hlavičky v požiadavke zaslanej na server
%k	Počet požiadaviek, ktoré sa majú „udržiavať pri živote“ (keepalive)
%l	Vzdialené prihlasovacie meno (od identd, ak je poskytnuté). Ak nie je prítomný parameter *mod_ident* a parameter *IdentityCheck* nie je zapnutý, vráti pomlčku
%L	ID logu požiadavky z chybového logu (alebo „-“, ak do chybového logu nebola pre túto požiadavku zalogovaná žiadna chyba). Po vyhľadaní zhodného riadka v chybovom logu môžeme zistiť, čo chybu spôsobilo
%m	Metóda požiadavky
%{VARNAME}n	Obsah poznámky s názvom VARNAME z iného modulu
%{VARNAME}o	Obsah premennej s názvom VARNAME: riadky hlavičky v odpovedi
%p	Kánonický port servera obsluhujúceho požiadavku
%{format}p	Kánonický port servera obsluhujúceho požiadavku alebo skutočný port servera alebo skutočný port klienta. Prípustné formáty sú *canonical*, *local* a *remote*
%P	ID procesu „dieťaťa“, ktoré obsluhovalo požiadavku
%{format}P	ID procesu alebo ID vlákna „dieťaťa“, ktoré obsluhovalo požiadavku. Prípustné formáty sú *pid*, *tid* a *hextid*
%q	Reťazec žiadosti (query)
%r	Prvý riadok požiadavky
%R	Handler generujúci odpoveď (ak existuje)

Formátovací reťazec	**Popis**
%s	Stav. Pre požiadavky, ktoré boli interne presmerované je toto stav originálnej požiadavky. Ak chceme zobraziť konečný stav, použijeme %>s
%t	Čas prijatia požiadavky, vo formáte *[18/Sep/2011:19:18:28 -0400]*. Posledné číslo indikuje časový posun od GMT
%{format}t	Čas v danom formáte. Mal by byť uvedený vo formáte *strftime(3)* [36]. Ak formát začína príkazom *begin:* (predvolená hodnota), zaznamená sa čas na začiatku spracovávania požiadavky. Ak formát začína príkazom *end:*, zaznamená sa čas blízko pri konci spracovávania požiadavky. Ako doplnok k formátu *strftime(3)* sú podporované ďalšie formáty, ktoré nájdeme na stránkach dokumentácie [4]
%T	Čas potrebný na obslúženie požiadavky, v sekundách
%u	Ak bola požiadavka overená, zobrazí sa tu vzdialený používateľ
%U	Požadovaná cesta URL, bez žiadosti (query)
%v	Kánonický názov servera *ServerName*, ktorý obslúžil požiadavku
%V	Názov servera podľa možnosti *UseCanonicalName*
%X	Stav pripojenia po dokončení odpovede. Existujú tri prípustné hodnoty: • X – spojenie bolo prerušené pred dokončením odpovede, • + – spojenie môže byť po odoslaní odpovede naďalej udržiavané, • - – spojenie bude po odoslaní odpovede ukončené.
%I	Prijaté bajty, vrátane požiadavky a hlavičiek. Nemôže nadobudnúť hodnotu 0
%O	Prijaté bajty, vrátane hlavičiek
%S	Prenesené bajty (prijaté a odoslané), vrátane požiadavky a hlavičiek. Nemôže nadobudnúť hodnotu 0. Ide o kombináciu %I a %O

Tabuľka 3.1: Formátovacie reťazce servera Apache

Formátovací reťazec	**Význam**
%400,501{User-agent}i	Zaznamená do logu iba položku *User-agent* s chybovými kódmi 400 a 501. Pre ostatné kódy bude zaznamenaná pomlčka
%!200,304,302{Referer}i	Zaznamená do logu iba položku *Referer* s požiadavkami, ktoré **nevrátia** jeden z troch špecifikovaných kódov. Pre ostatné kódy bude zaznamenaná pomlčka

Tabuľka 3.2: Príklady modifikátorov formátovacích reťazcov servera Apache

Formátovací reťazec	**Popis**
%a	Vzdialená IP adresa
%A	Lokálna IP adresa
%b	Odoslané bajty, bez hlavičiek HTTP alebo „-“, ak je hodnota 0
%B	Odoslané bajty, bez hlavičiek HTTP
%h	Názov vzdialeného hostiteľa (alebo jeho IP adresa, ak je vypnutá funkcia *resolveHostis*)
%H	Protokol požiadavky
%l	Vzdialené používateľské meno z parametra identd
%m	Metóda požiadavky (GET, POST atď.)
%p	Lokálny port, na ktorom bola požiadavka prijatá
%q	Reťazec žiadosti (query)
%r	Prvý riadok požiadavky (metóda a URI požiadavky)
%s	Kód stavu HTTP odpovede
%S	ID používateľskej relácie
%t	Čas a dátum, vo formáte CLF
%u	Vzdialený používateľ (ak bol overený) alebo „-“
%U	Požadovaná cesta URL
%v	Názov lokálneho servera
%D	Čas potrebný na spracovanie požiadavky, v milisekundách
%T	Čas potrebný na spracovanie požiadavky, v sekundách
%I	Názov vlákna súčasnej požiadavky

Tabuľka 3.3: Možnosti logovania prístupového logu v aplikačnom serveri JBoss

Úroveň	**Použitie**	**syslog**	**eventlog**
DEBUG	Poskytuje podrobné informácie pre vývojárov	DEBUG	INFORMATION
INFO	Poskytuje informácie požadované používateľom	INFO	INFORMATION
NOTICE	Poskytuje informácie, ktoré môžu byť nápomocné pre používateľov	NOTICE	INFORMATION
WARNING	Poskytuje varovania pred pravdepodobnými problémami	NOTICE	WARNING
ERROR	Poskytuje záznam o chybe, ktorá spôsobila zlyhanie aktuálneho príkazu	WARNING	ERROR
LOG	Poskytuje informácie, ktoré sú nápomocné pre administrátorov	INFO	INFORMATION
FATAL	Poskytuje záznam o chybe, ktorá spôsobila zlyhanie aktuálnej relácie	ERR	ERROR
PANIC	Poskytuje záznam o chybe, ktorá spôsobila zlyhanie všetkých databázových relácií	CRIT	ERROR

Tabuľka 3.4: Tabuľka vysvetľujúca úrovne logovania v PostgreSQL

Formátovací reťazec	**Popis**
%a	Názov aplikácie
%u	Používateľské meno
%d	Názov databázy
%r	Názov vzdialeného hostiteľa alebo IP adresa a vzdialený port
%h	Názov vzdialeného hostiteľa alebo IP adresa
%p	ID procesu
%t	Časový odtlačok s milisekundami
%m	Časový odtlačok bez milisekúnd
%i	Značka príkazu: typ aktuálneho príkazu relácie
%e	Chybový kód SQLSTATE
%c	ID relácie
%l	Počet riadkov pre každú reláciu alebo proces, začínajúc od 1
%s	Časový odtlačok štartu procesu
%v	ID virtuálnej transakcie
%x	ID transakcie
%q	Neprodukuje výstup, ale vraví procesom mimo relácie, aby na tomto mieste v reťazci skončili. Je ignorovaný procesmi v rámci relácie
%%	Znak %

Tabuľka 3.5: Možnosti logovania prostredníctvom formátovacích reťazcov v databázovej technológii PostgreSQL

Kapitola 4

Analýza a návrh riešenia

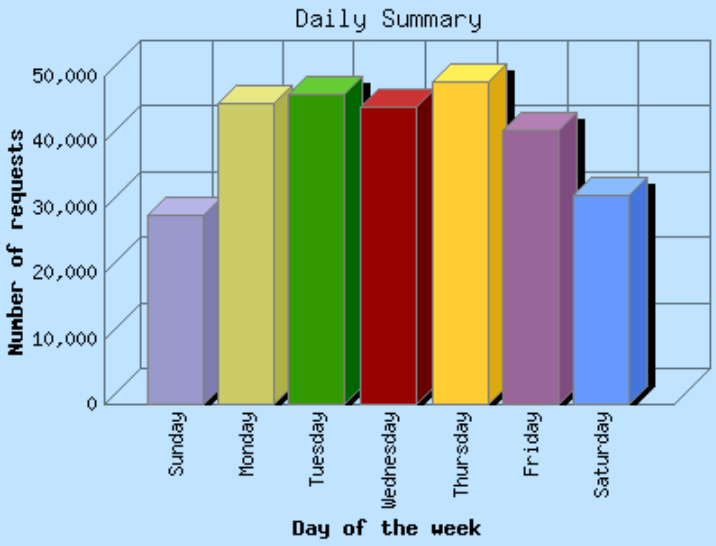
V tejto kapitole sa bližšie pozrieme na nástroje, ktoré nám môžu pomôcť pri riešení problému. Povieme si, aké máme možnosti, predstavíme si existujúce riešenia a zhrnieme si ich plusy a mínusy. Kapitolu sme napísali tak, aby ukazovala ako prebiehalo naše postupné prenikanie do problematiky a objavovanie nových nástrojov a systémov, ktoré nám môžu pomôcť pri riešení problému. Po predstavení týchto technológií si na základe požiadavkov od zadávateľov. Potom si zdôvodníme, prečo sme sa rozhodli použiť práve tie technológie, ktoré sú využívané vo finálnom produkte.

4.1 Webová analýza

V tejto sekcii si predstavíme zopár základných nástrojov na webovú analýzu logov. Tieto nástroje sa však zameriavajú na spracovanie logov na jednom mieste (jednom serveri) a riešia niektoré požiadavky zadávateľa. Dokážu spracovávať logy z požadovaných technológií, zvládajú stredne veľké objemy údajov, majú grafické používateľské rozhranie a väčšinou sú mulitplatformné. Z princípu sú však nerozšíriteľné, neobsahujú REST API a sú skôr zamerané na SEO a analýzu. Zhromažďovanie a transport logov by sme museli riešiť pomocou iných nástrojov. Venujeme im pár riadkov, pretože počas analýzy riešenia sme ich použitie zvažovali. Ukázalo sa však, že riešia len malú časť našich problémov, niekomu však môžu prísť vhod.

4.1.1 Analog

Prvým v našom zozname je nástroj s názvom Analog. Je pomerne rozšírený a jeho tvorcovia dokonca tvrdia, že najrozšírenejší na svete. Tento nástroj sa špecializuje na tvorbu prehľadov z logov vo formáte HTML. Je rýchly, škálovateľný, multiplatformný a veľmi konfigurovateľný. Tento nástroj je zadarmo. Zaujímavosťou je, že poskytuje tvorbu prehľadov až v 32 jazykoch. Po spojení s nástrojom Report Magic vytvára aj grafické prehľady. Príklad môžeme vidieť na obrázku 4.1. Nevýhodami tohto nástroja sú nutnosť využívať databázu tretej strany a chýbajúce REST API.



	Day of the week	Number of requests	Number of page requests
1.	Sunday	28,955	2,494
2.	Monday	45,765	4,236
3.	Tuesday	47,284	4,516
4.	Wednesday	45,303	4,804
5.	Thursday	49,057	4,480
6.	Friday	41,650	4,123
7.	Saturday	31,796	2,614
	Total Weekdays	**229,059**	**22,159**
	Total Weekend	**60,751**	**5,108**

Obr. 4.1: Príklad vygenerovaného denného prehľadu v programe Analog

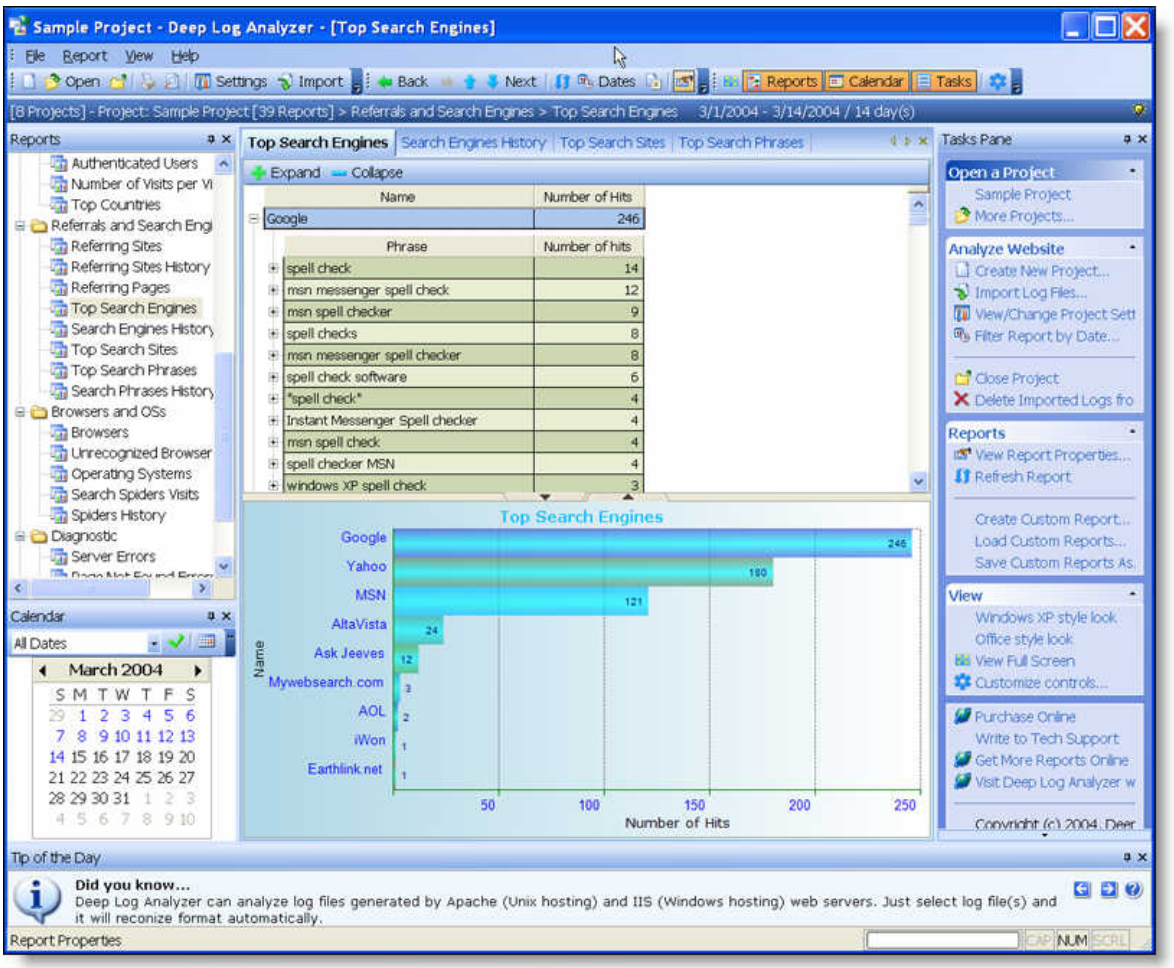
4.1.2 Apache Log Viewer

Jeden z najpoužívanejších nástrojov na analýzu logov zo serveru Apache. Medzi jeho výhody patrí to, že nie je potrebné ho inštalovať na Apache/IIS server, dokáže farebne rozlíšiť riadky logu podľa kódov stavu HTTP a umožňuje preložiť IP adresu na krajinu. Podporuje aj klasické funkcie ako vyhľadávanie (podľa IP adresy, reťazca požiadavky, dátumu atď.), filtrovanie podľa kódov stavu HTTP či vizuálne prehľady a štatistiky. Očividnou nevýhodou

je v našom prípade chýbajúca podpora ďalších dvoch technológií a nutnosť dodatkového vkladania do databázy. Chýbajúce REST API takisto nie je výhodou.

4.1.3 Deep Log Analyzer

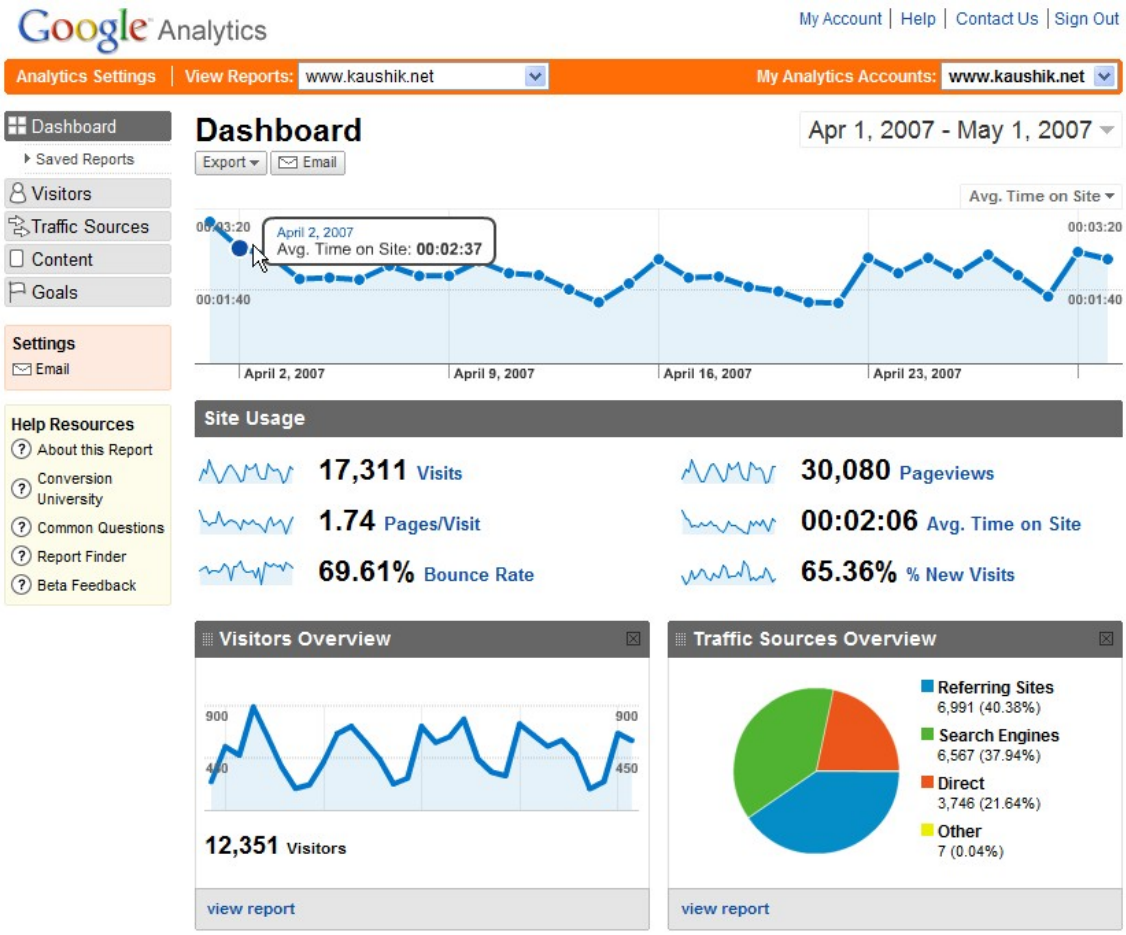
Mnohými vyhlasovaný za najlepší nástroj na webovú analýzu vôbec. Funguje lokálne bez nutnosti inštalácie alebo vkladania kúskov kódu sledovania na stránku. Zameriava sa na malé až stredné webové stránky a okrem klasických funkcií vyhľadávania, filtrovania a tvorby štatistík ponúka aj niekoľko zaujímavých vecí navyše. Medzi tie najzaujímavejšie patria flexibilita a výber až zo 40 druhov prehľadov, hierarchické štatistické prehľady webových stránok a možnosť exportu údajov vo formáte databázy MS Access. Hlavnými nevýhodami pre nás sú používanie technológie závislej na OS, prílišná orientácia na SEO a v neposlednom rade to, že za pokročilejšie funkcie je potrebné zaplatiť. Ako program vyzerá môžeme vidieť na obrázku 4.2.



Obr. 4.2: Zobrazenie hlavného pracovného prostredia programu Deep Log Analyzer

4.1.4 Google Analytics

Spoločnosť Google netreba nikomu predstavovať. Sada nástrojov od tejto spoločnosti tiež patrí k tomu najlepšiemu, čo môžeme v tomto odvetví nájsť. Medzi hlavné výhody patrí naozaj rozsiahla dokumentácia vo veľkom počte dostupných jazykov, široká paleta prehľadov a grafov a možnosti priameho prepojenia s inými nástrojmi od tejto spoločnosti. To nám môže výrazne pomôcť pri dosiahnutí našich cieľov spojenými s webovými technológiami (zvýšenie návštevnosti, zlepšenie konverzií, odhalenie a náprava príčin slabšej výkonnosti). Hlavnou nevýhodou je odovzdanie prístupu k súkromným údajom veľkej korporácii akou je Google a nutnosť pridať na webové stránky tzv. kód sledovania. Tento kód doslova sleduje všetko, čo sa na našich stránkach deje a na základe toho potom zostavuje všetky štatistiky. Pracovné prostredie môžeme vidieť na obrázku 4.3.



Obr. 4.3: Zobrazenie hlavného pracovného panelu nástroja Google Analytics

4.1.5 Webalizer

Päticu používaných nástrojov nám uzatvára Webalizer. Webalizer je malým nástrojom, ktorý je zadarmo. Jeho veľkosť mu však neuberá na kvalite. Je napísaný v jazyku C, je multiplatformný a veľmi rýchly pri spracovávaní logov. Poskytuje veľké množstvo štatistík, ktoré sa môžu objaviť vo vytváraných prehľadoch. Prehľady ponúka vo viac ako 30 jazykoch.

Na záver podkapitoly uvádzame prehľadnú tabuľku, ktorá sumarizuje výsledky porovnania týchto nástrojov vzhľadom na požiadavky práce.

Názov tech.	Multiplat-formnosť	Pokrytie všetkých 3 tech.	Kapacita a rýchlosť DB	Podpora formátu JSON	Integrované REST API	Grafická vizualizácia	Modularita, rozšíriteľnosť
Analog	áno	áno	áno	nie	nie	áno	nie
Apache Log Viewer	nie	nie	áno	nie	nie	áno	nie
Deep Log Analyzer	nie	áno	áno	nie	nie	áno	áno, za poplatok
Google Analytics	áno	nie	áno	áno	nie	áno	nie
Webalizer	áno	áno	áno	áno	nie	áno	nie

Obr. 4.4: Porovnávacia tabuľka jednotlivých nástrojov

4.2 Centralizované logovanie

Ako sme už povedali, logy predstavujú kritickú časť systému, poskytujú dôležité informácie a odpovedajú na otázky, čo systém robí a čo sa udialo. Drvivá väčšina procesov systému vytvára logové súbory v nejakej forme. Tieto logy sú zhromažďované ako súbory na disku a zvyčajne je tu nastavená ešte forma rotovania. Keď je systém hostený na jednom počítači (serveri), k logom máme jednoduchý prístup a pomerne jednoducho ich dokážeme aj analyzovať. Avšak po rozšírení systému na viacero hostov sa logovanie stáva problémom. Bez správnych nástrojov je ťažké vyhľadávať určitú chybu naprieč stovkami súborov a desiatkami serverov. Bežným prístupom, ktorý tento problém rieši, je nakonfigurovanie centralizovaného logovacieho systému, takže údaje z jednotlivých súborov sú prenesené do centrálneho úložiska. Existuje niekoľko spôsobov konfigurácie centralizovaného logovacieho systému.

4.2.1 Replikovanie súborov

Najjednoduchším spôsobom je naplánovať replikovanie logových súborov na centrálny server pomocou skriptov alebo špecializovaných nástrojov ako cron a rsync. Tento prístup však má

množstvo nevýhod a v konečnom dôsledku údaje neštrukturalizuje, slúži iba ako centralizované úložisko.

4.2.2 Syslog

Ďalšou možnosťou je použitie nástroja syslog. Väčšina ľudí používa jednu z dvoch implementácií: rsyslog alebo syslog-ng. Tieto daemony umožnia procesov posielať im logové správy a konfigurácia nástroja syslog určí, ako sa budú tieto správy ukladať. V rámci centralizovaného logovania je na sieti nastavený centrálny syslog démon a klientské logovacie daemony, ktoré centrálnemu preposielajú správy. Viac informácií o tomto prístupe si môžeme prečítať tu [8].

Syslog je dobrý, pretože ho používa veľké množstvo procesov a aplikácií a pravdepodobne je už nainštalovaný na vašom systéme. S týmto prístupom však budete musieť vyriešiť otázku škálovateľnosti servera a zabezpečiť jeho vysokú dostupnosť.

4.2.3 Distribuované kolektory logov

Pred pár rokmi vznikla v súvislosti s týmto problémom nová trieda riešení – distribuované kolektory logov. Tie sú navrhnuté špeciálne pre zhromažďovanie logov a udalostí s vysokým objemom údajov a veľkou priepustnosťou. Väčšina týchto riešení slúži na zhromažďovanie všeobecných udalostí a ich spracovávanie v reálnom čase a logovanie je len jeden z prípadov, ktoré s ich pomocou môžeme vyriešiť. Všetky majú rôznu funkcionalitu a svoje odlišnosti, no ich architektúra je veľmi podobná. Pozostávajú z logovacích klientov a/alebo agentov na každom hostiteľskom počítači. Agenty preposielajú logy do bloku kolektorov, ktoré na oplátku preposielajú jednotlivé správy do škálovateľnej vrstvy úložiska. Hlavnou myšlienkou je to, že vrstva kolektorov je horizontálne škálovateľná a rastie s pribúdajúcim množstvom hostiteľských počítačov a logovaných správ. Podobne je navrhnutá aj vrstva úložiska – s narastajúcim objemom údajom je horizontálne škálovateľná. Poskytnuté vysvetlenie predstavuje veľké zjednodušenie toho, čo všetky tieto nástroje dokážu, no vidíme, že oproti nástroju syslog sú o krok vpred. Na nasledujúcich riadkoch si toto riešenie predstavíme bližšie.

4.2.4 Architektúra centralizovaného logovacieho systému

Ak chceme vybudovať funkčný systém centralizovaného logovania, musíme adresovať štyri základne aspekty: zhromažďovanie, prenos, ukladanie a analýzu údajov (logov, udalostí atď.). Vidíme to na obrázku 4.5. Počas rešerše sme sa dozvedeli, že každý aspekt zvyčajne pokrýva iný nástroj, takže na vybudovanie robustného riešenia ich potrebujeme niekoľko prepojiť dohromady.

4.2.4.1 Zhromažďovanie

Aplikácie vytvárajú logy rôznymi spôsobmi. Niektoré používajú syslog, niektoré logujú priamo do súborov. Ak vezmeme do úvahy typickú aplikáciu bežiacu na linuxových hostoch, pár desiatok súborov sa bude nachádzať v priečinku /var/log a pár v priečinkoch špecifických pre danú aplikáciu v priečinkoch home.

Pri zhromažďovaní sa musíme zamyslieť nad tým, či údaje logy potrebujeme takmer v reálnom čase, alebo nám stačí dostávať súbory s miernym oneskorením a pracovať s nimi až neskôr. V našom prípade počítame s druhou alternatívou a budeme predpokladať, že logy nemusíme mať k dispozícii takmer v reálnom čase.

4.2.4.2 Prenos

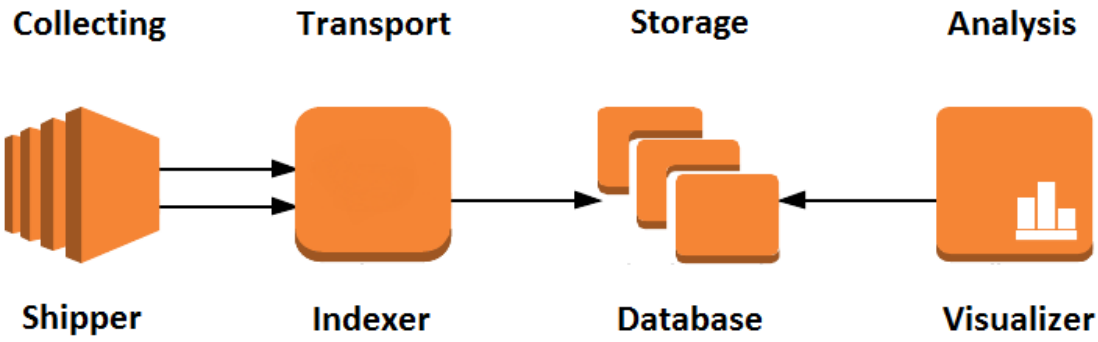
Logové súbory sa môžu na viacerých hostoch rýchlo nazhromaždiť. Spoľahlivý a rýchly prenos do centralizovaného úložiska si vyžaduje využitie ďalších nástrojov a ich konfiguráciu. Hlavne, ak chceme prenášať efektívne a zabezpečiť, že sa údaje nestratia.

Frameworky ako napríklad Scribe [33], Flume [13], Heka [16], Logstash [28], Chukwa [9], Fluentd [12], nsq [30] a Kafka [21] sú špeciálne navrhnuté na spoľahlivý prenos veľkého objemu údajov. Každý z týchto frameworkov adresuje problém prenosu, robia to však odlišne [7].

Frameworky ako Scribe, nsq a Kafka potrebujú, aby klienty logovali údaje pomocou ich API. Zvyčajne sa napíše kúsok aplikačného kódu, ktorý loguje priamo do zdrojov, čo umožňuje znížiť odozvu a zvýšiť spoľahlivosť. Následne sa logy centralizujú prostredníctvom kódu postavenom na príslušných API. Ak máme pod kontrolou aplikáciu, ktorej logy chceme zhromažďovať, tieto frameworky môžu byť efektívnejšie. V našom prípade však počítame s tým, že nebudeme mať prístup k jednotlivým aplikáciám.

Logstash, Fluentd a Flume poskytujú množstvo vstupov a zároveň umožňujú prácu so súbormi a spoľahlivý prenos. Preto predstavujú lepšiu alternatívu pre všeobecné zhromažďovanie a prenos logov, čo je aj náš prípad.

Napadá nás ešte možné použitie technológií rsyslog a syslog-ng, ktoré sú na zhromažďovanie a prenos logov taktiež určené, avšak nie každá aplikácia používa syslog ako logovaciu platformu.



Obr. 4.5: Diagram znázorňujúci architektúru centralizovaného logovacieho systému

4.2.4.3 Ukladanie

Logové súbory sa teda prenášajú, teraz potrebujú cieľovú destináciu. Centralizované úložisko musí byť schopné zvládnuť postupný nárast za určitý čas. Každý deň bude pridané určité

množstvo údajov, ktoré závisí na počte hostov a procesov, ktoré generujú logové údaje.

Ukladanie závisí od niekoľkých faktorov:

Ako dlho majú zostať uložené – Ak logy potrebujeme na dlhotrvajúcu archiváciu a nevyžadujú okamžitú analýzu, vhodným riešením je použitie technológií S3, AWS Glacier, prípadne zálohovanie na veľkokapacitné pásky, keďže tieto riešenia poskytujú relatívne nízku cenu za objem údajov. Ak potrebujeme logy za posledných niekoľko týždňov či mesiacov, dobre nám poslúžia distribuované úložné systémy ako HDFS, Cassandara, MongoDB alebo ElasticSearch. Ak potrebujeme iba posledných pár hodín na analýzu takmer v reálnom čase, poslúži nám taktiež technológia Redis. V našom prípade si zvolíme zlatú strednú cestu.

Veľkosť údajov nášho systému – Týždenný objem logov spoločnosti Google a lokálneho obchodu s rybárskymi potrebami sú dve rôzne veci. Systém úložiska preto musí počítať s tým, že veľkosť našich údajov bude narastať. Musí preto umožňovať horizontálnu škálovateľnosť.

Ako budeme k logom pristupovať – Niektoré úložiska nie sú vhodné na analýzu takmer v reálnom čase, ani na sériovú analýzu. Načítanie súboru z veľkokapacitnej pásky alebo pomocou technológie AWS Glacier môže trvať hodiny. Tieto riešenia teda nie sú vhodné napríklad na riešenie problémov s produkčnou verziou (a nie sú vhodné ani v našom prípade). Interaktívnejšiu analýzu môžeme dosiahnuť, ak údaje uložíme pomocou technológií ElasticSearch, MongoDB alebo HDFS. Tie nám umožnia efektívnejšiu prácu so surovými údajmi. Niektoré logované údaje sú však tak veľké, že potrebujeme frameworky orientované na sériovú analýzu. V tomto prípade je štandardne používaným riešením Apache Hadoop (HDFS).

4.2.4.4 Analýza zhromaždených údajov

Po uložení údajov do centralizovaného úložiska ich potrebujeme nejakým spôsobom analyzovať. Najbežnejším spôsobom je nastavenie periodického procesu sériovej analýzy. To sa týka hlavne úložísk ako sú Hive, HDFS a Pig.

Ak potrebujeme na analýzu grafické rozhranie (v našom prípade to tak je)tento článok [7] spomína použitie technológie ElasticSearch ako úložiska a frameworkov Kibana alebo Graylog2 na dopyt a inšpekciu údajov. Parsovanie logov by v tomto prípade mali zabezpečiť frameworky Logstash, Fluentd, Flume či Heka. Tento prístup umožňuje priblíženie sa analýze v reálnom čase, nie je však vhodný na spracovanie obrovského množstva údajov prostredníctvom sériovej analýzy.

4.3 Využiteľnosť existujúcich riešení

V nasledujúcej sekcii si rozoberieme požiadavky zadávateľa problému a na základe vytýčených cieľov si navrhneme vhodné riešenie. Predchádzajúca sekcia ukázala, že centralizovaný logovací systém predstavuje vhodného kandidáta na vyriešenie nášho problému. V prípade, že sa nám podarí nájsť vhodné technológie, naším riešením by bolo navrhnúť a implementovať práve takýto systém. Sekcia Architektúra centralizovaného logovacieho systému nám na základe požiadaviek už zúžila výber dostupných technológií. Poďme si to teda zopakovať:

- Zhromažďovanie – žiadne zúženie kritérií, zhromažďovanie zvládajú všetky frameworky

- Prenos – všeobecné zhromažďovanie a prenos logo bez prístupu k aplikáciám. Vhodní kandidáti: Flume, Fluentd, Logstash
- Ukladanie – zlatá stredná cesta v prípade rýchlosti prístupu aj veľkosti objemu údajov. Vhodní kandidáti: Elasticsearch, HDFS, MongoDB
- Analýza– potreba určitej vizualizácie údajov. Vhodní kandidáti: Heka, Flume, fluentd, Logstash ako parsery, ElasticSearch ako databáza a jeden z frameworkov Kibana alebo Graylog2 ako grafické rozhranie.

Vyzerá to tak, že rozhodovať bude vhodnosť jedného z troch frameworkov na zhromažďovanie a analýzu logov – Flume, Fluentd a Logstash. K nim následne vyberieme vhodné úložisko a grafické rozhranie. Teraz si ešte raz zhrnieme požiadavky zadávateľa a špecifikácie cieľov, ktoré budú slúžiť ako kritéria na výber jedného z troch nástrojov pri implementácii riešenia problému.

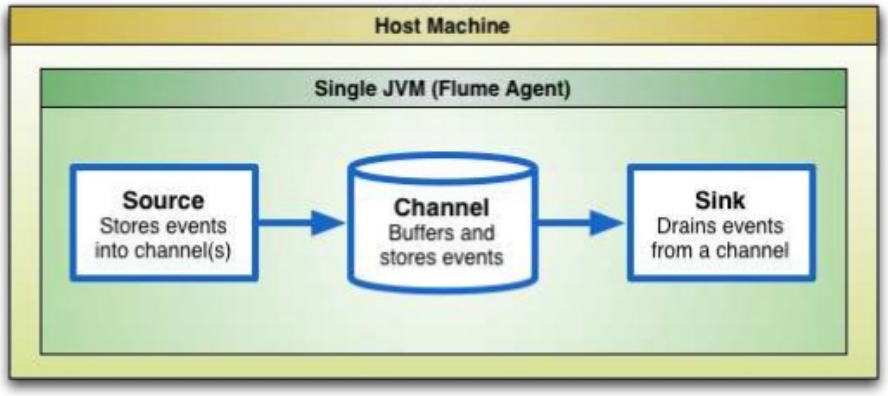
- podpora operačného systému na platforme UNIX a iných platforiem,
- funkčné spracovanie logových súborov z technológií Apache, JBoss a PostgreSQL,
- ukladanie stredne veľkého objemu údajov do databázy a uspokojivá rýchlosť použitých technológií,
- práca s údajmi v tejto databáze podľa používateľských kritérií,
- import údajov na ďalšie spracovanie vo formáte JSON,
- REST API, ktoré zabezpečí import údajov na ďalšie spracovanie vo formáte JSON,
- funkčné používateľské prostredie s možnosťou vizualizácie štatistík,
- modulárnosť, jednoduchá rozšíriteľnosť.

Ako ďalší postup pri riešení problému sa ponúka naprogramovanie všetkých častí riešenia samostatne. To by nám umožnilo upravovať jednotlivé parametre riešenia podľa dôležitosti. Neboli by sme viazaní rozsahom hodnôt, polí, kapacitou atď. nejakých iných nástrojov.

Pôvodne sme chceli postupovať týmto spôsobom. Najväčším prínosom by bolo totiž práve ušitie riešenia na mieru. So zadávateľom by sme neustále komunikovali a mohli sa v priebehu procesu prikláňať k tým riešeniam čiastkových problémov, ktoré by nám najviac vyhovovali. Pri hlbšom preniknutí do problematiky v rámci výskumu pre teoretickú časť tejto práce sme však zistili, že pokryť všetky požiadavky na takej úrovni, aby sme boli spokojní a mohli riešenie odprezentovať ako konečné, je nereálne. Aspoň nie za obmedzený čas a v danom počte osôb (1). Pri rešerši v tejto kapitole sme však objavili aj využiteľné, a dostatočne komplexné nástroje na to, aby sme sa k požadovanému výsledku dopracovali. Nemusíme tak všetko programovať od nuly a môžeme sa viac zamerať na detaily riešenia a jeho reálnu využiteľnosť pre zadávateľa. Poďme si ich dôkladnejšie porovnať.

4.3.1 Apache Flume

Apache Flume predstavuje škálovateľné riešenie zapisovania logov do súborového systému Apache Hadoop. Je to distribuovaný systém na prijímanie údajov navrhnutý na efektívne zhromažďovanie, štruktúrovanie a prenos veľkého množstva údajov do HDFS. V terminológii Flume je správa nazvaná udalosť. Udalosti prúdia cez jedného alebo viacerých agentov a potom dorazia do cieľovej destinácie (zvyčajne ide o HDFS). Udalosť pozostáva z tela správy a hlavičky, ktorá nesie metaúdaje, napríklad časový odtlačok, pôvodcu udalosti a iné. Agent je proces jazyka Java, ktorý hostí komponenty, cez ktoré udalosti prúdia. Tieto komponenty sú zdroje (sources), kanály (channels) a spotrebiče (sinks).



Obr. 4.6: Diagram princípu fungovania technológie Apache Flume

Zdroj prijíma správy odoslané externými zdrojmi. Zdroj po prijatí udalosť uloží v jednom alebo viacerých kanáloch. Kanál predstavuje jednoduché úložisko, ktoré prechováva udalosť, pokiaľ nie je spracované spotrebičom. Spotrebič vezme udalosť z kanála a odošle ho do externého úložiska (HDFS) alebo prepošle ďalšiemu agentu.

Apache Flume podporuje okrem iného tieto zdroje: Avro, Thrift, syslog aj textové riadky, z ktorých vytvorí udalosti. Na výstupe potom figurujú napríklad tieto spotrebiče: Elasticsearch, HBase, HDFS (nie je teda vylúčené použitie úložiska Elasticsearch s niektorým grafickým rozhraním). Okrem toho podporuje aj pluginy tretích strán, napríklad na zápis do databázy MongoDB. Výrobcovia tvrdia, že na vstupe sa môže objaviť takmer čokoľvek, keďže zdroje sú veľmi flexibilné a konfigurovateľné.

Model škálovateľnosti a rozšíriteľnosti technológie Apache Flume môže byť použitý na dosiahnutie analýzy údajov takmer v reálnom čase. Flume obsahuje klienta log4j, takže doň môžeme jednoducho ukladať aj správy z aplikácie jazyka Java. Komunita naviac neustále vyvíja nové klienty, takže si Flume poradí s väčšinou formátov.

Výhody:

- garantované doručenie údajov,
- horizontálna škálovateľnosť,
- vysoká priepustnosť,

- odolnosť voči chybám.

4.3.2 Fluentd

Fluentd je aplikácia jazyka CRuby vyžadujúca Ruby vo verzii 1.9.2 alebo vyššej. Existuje verzia open-source, aj platená verzia – td-agent. Fluentd podporuje Linux a Mac OS X, v súčasnosti však nepodporuje Windows, čo môžeme považovať za nevýhodu.

Fluentd používa vstupy a výstupy prijímajúce a odosielajúce údaje a mechanizmus spravujúci destinácie jednotlivých logov. Interne sú správy konvertované do formátu JSON, čo poskytuje štruktúru neštrukturovaným logovým údajom. Správy môžeme značkovať a smerovať na rôzne výstupy.

Rozšíriteľnosť je zabezpečená pluginmi, ktorých je v repozitári veľké množstvo. Vstupy a výstupy majú naprogramované užitočné funkcie ako vyrovnávanie (buffering), rovnomerné rozloženie záťaže a poradia si aj s vypršaním časových intervalov (timeouts) a opätovným získaním údajov (retries). Tieto funkcie sú dôležité pre stabilné a spoľahlivé doručovanie údajov.

Fluentd je veľmi podobný nášmu poslednému kandidátovi – technológii Logstash. Asi najväčším rozdielom je prístup k dizajnu. Logstash zdôrazňuje flexibilitu a súčinnosť, zatiaľ čo Fluentd prioritizuje jednoduchosť a robustnosť. To však neznamená, že Logstash nie je robustný a Fluentd nie je flexibilný. Ide jednoducho o rozličnú prioritizáciu funkcionality.

textbfVýhody:

- jednoduchosť a robustnosť,
- horizontálna škálovateľnosť,
- rozšíriteľnosť pluginmi,
- vysoká prispôsobiteľnosť.

4.3.3 Logstash

Logstash je aplikáciou postavenou na jazyku JRuby, ktorá vyžaduje na prevádzku JVM. Keďže beží na JVM, môže byť prevádzkovaná všade tam, kde je podporovaná JVM, čo znamená Linux, Mac OS X aj Windows. Balíček je dodávaný ako spustiteľný súbor jar, ktorý umožňuje veľmi jednoduchú inštaláciu.

Jednou zo známych nevýhod JVM môže byť vyššia pamäťová náročnosť, akú by sme očakávali od prenosu a zhromažďovania logov. Logstash našťastie obsahuje vstavný plugin Logstash forwarder (predtým Lumberjack), ktorý zabezpečuje zhromažďovanie z jednotlivých hostov, a ktorý má naozaj skvelú pamäťovú náročnosť (niekoľko desiatok kB). V prípade potreby je ho možné nakonfigurovať.

Logstash podporuje veľké množstvo vstupov, kodekov, filtrov a výstupov. Vstupy predstavujú zdroje údajov. Kodeky konvertujú prichádzajúci formát na vstupe do internej reprezentácie, ako aj späť na formát výstupov. Používajú sa v prípade, že prichádzajúce správy nepozostávajú iba z jedného riadka textu. Filtre vykonávajú na udalostiach akcie, ktoré nám umožňujú udalosti modifikovať alebo neprijať. Výstupy predstavujú destinácie, kam môžu

udalosti smerovať. Jednou z najlepších funkcií technológie Logstash je to, že pomocou filtrov a kodekov môžeme udalosti filtrovať, upravovať a transformovať z jedného formátu do iného.

Podobne ako Flume a Fluentd, aj Logstash dokáže prijímať údaje z veľkého množstva zdrojov – syslogu, logov udalostí Windows, štandardného vstupu, logových súborov atď. Logstash momentálne obsahuje najviac prídavných pluginov a pluginov tretích strán umožňujúcich jeho rozšíriteľnosť.

textbfVýhody:

- flexibilita a interoperabilita,
- horizontálna škálovateľnosť,
- rozšíriteľnosť pluginmi,
- vysoká prispôsobiteľnosť,
- schopnosť spracovávania údajov za pochodu.

Všetky tri frameworky sú dobrými nástrojmi na centralizované logovanie. Dokážu prenášať a zhromažďovať logy z viacerých hostov a každý poskytuje nejaké možnosti ďalšej úpravy či filtrovania údajov. Logstash v tomto vyhráva, no Fluentd a Apache Flume poskytujú trochu lepšiu spoľahlivosť pri prenášaní údajov. Každý z nich dokážeme prepojiť s databázou Elasticsearch a grafickým rozhraním. Väčšine našich požiadaviek teda vyhovuje ktorýkoľvek z nich. Logstash preferujeme trochu viac kvôli množstvu pluginov a jeho flexibilite. Počas porovnávania však vyšlo najavo ešte jedna skutočnosť. Spoločnosť Elasticsearch prezentuje trojicu Elasticsearch, Logstash a Kibana ako komplexné riešenie centralizovaného logovania. Nazývajú ho ELK. Pri bližšom preskúmaní a rešerši zistíme, že riešenie ELK je veľmi obľúbeným práve kvôli tomu, že tieto tri nástroje sú špeciálne vytvorené na spoluprácu a komunikáciu medzi sebou, čo prináša veľmi jednoduché nastavovanie. Na internete nájdeme veľké množstvo príručiek a diskusií týkajúcich sa konfigurácie, čo rozhodne nie je na zahodenie. Práve preto sme sa rozhodli zvoliť do nášho systému centralizovaného logovania Elasticsearch ako databázu, Logstash ako kolektor a parser logov a Kibanu ako grafické rozhranie – a teda riešenie ELK. Komplexnosť všetkých systémov je vidieť aj z nasledujúceho obrázka 4.7. Nevybrali sme teda iba podľa funkcionality, ale aj podľa zamerania, čo vysvetľujeme počas celej sekcie vyššie.

Názov tech.	Multiplatformnosť	Pokrytie všetkých 3 tech.	Kapacita a rýchlosť DB	Podpora formátu JSON	Integrované REST API	Grafická vizualizácia	Modularita, rozšíriteľnosť
Flume	áno	áno	áno	áno	áno	áno	áno
Logstash	áno	áno	áno	áno	áno	áno	áno
Fluentd	obmedzené	áno	áno	áno	áno	áno	áno

Obr. 4.7: Porovnávacia tabuľka nástrojov centralizovaného logovania

4.3.4 Elasticsearch

Elasticsearch je flexibilný, mocný, distribuovaný open source vyhľadávač a analytický engine [10]. Je schopný prehľadávať údaje a tvoriť štatistiky v reálnom čase. Od začiatku vývoja boli na prvom mieste spoľahlivosť a škálovateľnosť. Táto technológia nám dovoľuje vďaka sade API a DSL oveľa viac ako len fulltextové vyhľadávanie. Spolu s pridruženými technológiami Logstash a Kibana mysleli tvorcovia tohto softvéru na podobné situácie, v akej sa nachádza zadávateľ problému.

Tento engine chceme použiť v našom riešení ako databázový systém. Technológia Elasticsearch bola vyvíjaná práve na použitie v systémoch s veľkým množstvom údajov a na svižné a funkčné vyhľadávanie v týchto systémoch. Naviac má vbudované RESTful API, takže spĺňa aj túto požiadavku. Nasledujúci zoznam zhrnie jednotlivé pozitívne funkcie a ich prípadne využitie v našom riešení:

- Údaje a ich analýza v reálnom čase – logové súbory budú neustále pribúdať. Elasticsearch dokáže tieto údaje spracovávať a v reálnom čase ich zapracovať do existujúcich štatistík, čo vyhovuje požiadavke o svižnosti a rýchlosti celého riešenia.
- Distribuovanosť – celý systém umožňuje nasadenie v malom a postupné zväčšovanie sa s časom a pribúdajúcimi údajmi. Pridanie nových uzlov umožňuje získať výhodu v podobe dodatočného výkonu, ak by ho naše riešenie v budúcnosti potrebovalo.
- Dostupnosť – v prípade poruchy nejakého z uzlov ho systém automaticky detekuje, reorganizuje sa a zaistí prístup k údajom a ich bezpečnosť.
- Fulltextové vyhľadávanie – ako sme už veľakrát povedali, samotné údaje sú bezcenné, ak nemáme nástroje na ich organizáciu, filtrovanie a analýzu. Táto technológia podporuje vyhľadávanie vo viacerých jazykoch, jazyk na zadávanie žiadostí, geolokáciu, funkciu automatického dokončovania a funkciu opráv na základe obsahu (mysleli ste ...?).
- Orientácia na dokumenty – systém nám umožňuje ukladať entity z reálneho sveta v podobe dokumentov JSON. Všetky polia sú indexované, takže výsledky vyhľadávania sú veľmi rýchlo dostupné.
- Jednoduchosť – Elasticsearch umožňuje do databázy vkladať dokumenty JSON bez toho, aby sme museli pracne navrhovať štruktúru databázy. Systém vytvorí najvhodnejšiu štruktúru na základe samotných údajov a umožní vyhľadávanie v nich. Používateľ má stále možnosť špecifikovať, ako sú jeho údaje indexované.
- RESTful API – jedna zo základných požiadaviek zadávateľa. Prístup k údajom prostredníctvom JSON cez HTTP je preto veľmi jednoduchý.
- Technológie – Elasticsearch je postavený na technológii Apache Lucene, ktorá je napísaná v jazyku Java. To nám zaručuje multiplatformnosť.
- Open source licencia – konkrétne licencia typu Apache 2. Táto licencia umožňuje bezplatné nasadenie nášho riešenia a je jednou z najflexibilnejších dostupných licencií.

4.3.5 Logstash

Je nástroj, na centralizáciu logov z rôznych systémov na jednom mieste. Umožňuje na vstupe prijímanie takmer akéhokoľvek typu logov z veľkého množstva zdrojov. Následne interne tieto logy spracuje (rozparsuje, filtruje atď.) podľa toho, ako je nakonfigurovaný a následne logy odovzdá na požadovanú lokalitu či výstup.

Logstash dokáže spracovať všetky typy logov, od systémových, logov z webových serverov, cez chybové logy, až po aplikačné logy [25]. Pri použití s technológiou Elasticsearch ako databázy na ukladanie údajov a technológie Kibana ako používateľského rozhrania na vizualizáciu týchto údajov nám poskytuje presne to, čo potrebujeme pre vyriešenie nášho problému.

Logstash využijeme ako nástroj, ktorý spracuje všetky typy logov z požadovaných technológií a uloží ich do databázy. Vo fáze implementácie bude dôležité tento nástroj správne nakonfigurovať a vytvoriť tak komplexné riešenie. Logstash nám totiž umožňuje na vstupe prijímať nielen súbory (logy), dovoľuje nám vstup brať aj zo služieb ako eventlog, syslog, prostredníctvom tcp či z technológie log4j. Následne možnosti spracovania sú takmer nekonečné a veľký je aj počet výstupov, kam môžeme spracované údaje odoslať. Ak by nám ani to nestačilo, Logstash nám umožňuje napísať si vlastné pluginy, čím pokryjeme požiadavku jednoduchého rozšírenia.

V rámci riešenia budeme teda mať vytvorených jeden alebo niekoľko konfiguračných súborov, v ktorých nastavíme, aby Logstash urobil z údajmi to, čo potrebujeme. Čo všetko sa bude nachádzať v konfiguračnom súbore si povieme v kapitole Implementácia. Názornú predstavu o tom, ako Logstash funguje, nám dáva obrázok 4.8.

4.3.6 Kibana

Trojicu použitých nástrojov dopĺňa Kibana. Je to webové rozhranie na vizualizáciu logov a časových údajov. Kibana dokáže porovnávať počty žiadostí (queries) v rámci určitého časového rozsahu a umožňuje zobrazovať záznamy logov v podobe rôznych druhov grafov, takže aj pri väčšom objeme údajov si môžeme zvoliť to, čo nám pomôže údajom najlepšie porozumieť.

Webové rozhranie umožňuje v uložených údajoch fulltextovo vyhľadávať a zároveň písať žiadosti (queries) pomocou syntaxe jazyka Lucene. Táto syntax je veľmi jednoduchá, napríklad:

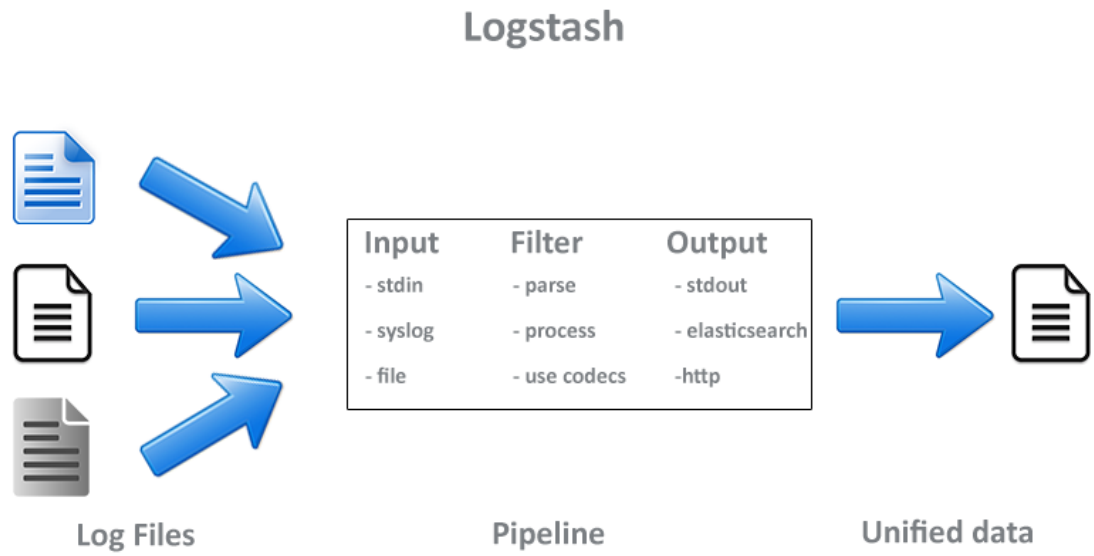
```
(apples AND oranges) OR "fruit"
```

alebo

```
snack:* AND calories:[300 TO *]
```

Viac o tejto syntaxi sa môžeme dozvedieť z dokumentácie projektu Apache Lucene [29].

Okrem toho nám Kibana ponúka možnosť zobraziť logový záznam ako celú správu alebo prehľadne zobraziť a porovnávať iba polia, ktoré potrebujeme. To isté platí pre rôzne druhy logov. Môžeme si napríklad zobraziť záznamy zo serveru Apache aj prístupy od databázy PostgreSQL alebo skúmať tieto záznamy osobitne. Ukážku pracovného prostredia z technológie Kibana môžeme vidieť na obrázku 4.9.



Obr. 4.8: Princíp fungovania technológie Logstash

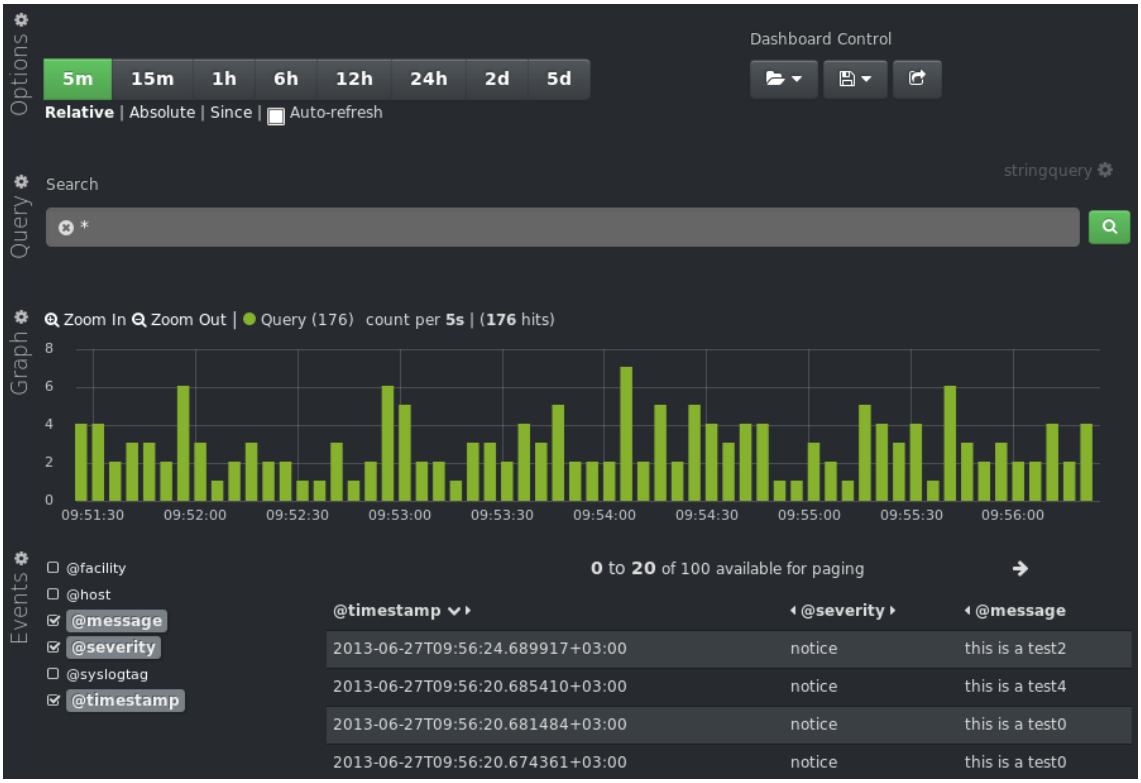
4.3.7 ELK

Tieto tri nástroje (Elasticsearch, Logstash a Kibana) spolu vytvárajú komplexné riešenie nášho problému. Táto trojica bola navrhnutá preto, lebo vytvára živé náhľady z takmer akéhokoľvek zdroja údajov v reálnom čase. Okrem toho spĺňa požiadavky zadávateľa a pomáha nám jednoduchšie dosiahnuť špecifikované ciele.

V navrhovanom riešení sa počíta s nástrojom Logstash ako hlavným parserom údajov z logových súborov, z technológiou Elasticsearch ako databázou, kde všetky tieto údaje uložíme a Kibana následne slúži ako doplňujúce webové rozhranie, schopné uložené údaje prehliadať, vizualizovať a filtrovať. Návrh architektúry si môžeme bližšie prezrieť na obrázku 4.10.

4.3.8 Podrobnosti návrhu

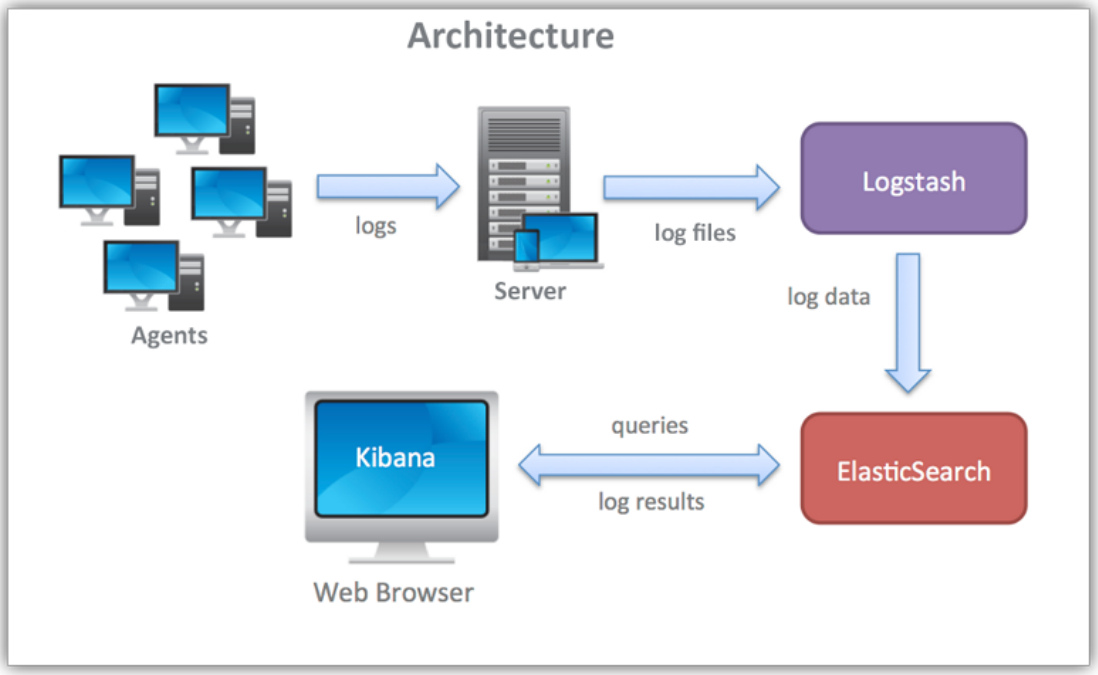
Všetky systémy sa pokúsime nainštalovať a spojazdniť na operačnom systéme Ubuntu. Po nainštalovaní a nakonfigurovaní nástrojov vytvoríme obraz celého operačného systému pomocou aplikácie VirtualBox. Vďaka tomu bude jednoduché naimportovať obraz celého riešenia prostredníctvom virtuálneho emulátora na akomkoľvek serveri. Zadávateľ tak bude môcť celé riešenie rýchlo presúvať medzi servermi. Po naimportovaní obrazu systému si zadávateľ môže prispôsobiť používateľské prostredie (napríklad odstrániť grafické rozhranie) a pozmeniť konfigurácie jednotlivých komponentov. My budeme mať možnosť otestovať riešenie v podmienkach, ktoré budú veľmi podobné reálnemu nasadeniu.



Obr. 4.9: Ukážka pracovného prostredia webového rozhrania Kibana

Ako prerekvizitu pred inštaláciou jednotlivých nástrojov budeme v operačnom systéme Ubuntu potrebovať iba nainštalovať Javu. Inštaláciu si bližšie popíšeme v kapitole Implementácia riešenia.

Po naimportovaní sa obraz správa ako plnohodnotný virtuálny operačný systém. Je možné ku nemu pristupovať aj cez internet. To pre nás bude veľmi vhodné pri pristupovaní k REST API, ktoré bude bežať na určitých portoch nad databázou Elasticsearch. Zároveň to poskytne viacero možností, ako do systému dopraviť logové súbory.



Obr. 4.10: Návrh architektúry riešenia ELK

Kapitola 5

Implementácia riešenia

V tejto kapitole si popíšeme spôsob inštalácie riešenia ELK, následne si rozoberieme, ako toto riešenie používať tak, aby sme z neho vyťažili čo najviac a ukážeme si v akom stave sa nachádza konfigurácia jednotlivých komponentov v rámci obrazu operačného systému. Predstavíme si zaujímavé možnosti konfigurácie spracovania logov, ktoré pre nás môžu byť užitočnými v budúcnosti a plynule na to nadviažeme na možné rozšírenia implementovaného riešenia.

5.1 Obraz operačného systému Ubuntu

Na nasledujúcich riadkoch si opíšeme stav implementácie riešenia v priloženom obraze operačného systému Ubuntu vo formáte .ova. Pomocou programu VirtualBox sme vytvorili obraz celého riešenia. Tento obraz obsahuje čistú inštaláciu operačného systému Ubuntu 14.04 v 64 bitovej verzii. V systéme sú nainštalované nástroje v týchto verziách:

- Elasticsearch 1.1.1
- Logstash 1.4.2
- Kibana 3.0.1

5.1.1 Inštalácia jednotlivých nástrojov

V tejto sekcii si stručne povieme, ako dospieť do stavu, aký je momentálne na obraze operačného systému. Je to pre prípad, že by nebolo možné používať obraz operačného systému pomocou virtualizácie. Predpokladáme inštaláciu na operačný systém unixového typu (tak, ako to bolo uvedené v požiadavkách). Ako sme spomínali v predchádzajúcej kapitole, jedinou prerekvizitou pred inštaláciou jednotlivých nástrojov je Java. Nainštalujeme verziu Oracle Java 7, pretože to stránky Elasticsearch odporúčajú. Systém by však mal fungovať aj s OpenJDK.

Pridanie Oracle Java PPA do apt:

```
sudo add-apt-repository -y ppa:webupd8team/java
```

Aktualizácia indexu balíčkov:

```
sudo apt-get update
```

Inštalácia Oracle Java 7 (následne potvrdíme zmluvné podmienky):

```
sudo apt-get -y install oracle-java7-installer
```

Teraz prejdeme na inštaláciu systému Elasticsearch. Najnovšia stabilná verzia systému Logstash 1.4.2 odporúča inštaláciu systému Elasticsearch vo verzii 1.1.1. My sa budeme tohto odporúčania držať.

Import verejného kľúča GPG od Elasticsearch:

```
wget -O - http://packages.elasticsearch.org/GPG-KEY-elasticsearch |
sudo apt-key add -
```

Vytvorenie zoznamu zdrojov:

```
echo 'deb http://packages.elasticsearch.org/elasticsearch/1.1/debian stable main' |
sudo tee /etc/apt/sources.list.d/elasticsearch.list
```

Aktualizácia indexu balíčkov:

```
sudo apt-get update
```

Inštalácia systému Elasticsearch 1.1.1:

```
sudo apt-get -y install elasticsearch=1.1.1
```

Po inštalácii mierne pozmeníme konfiguráciu:

```
sudo vi /etc/elasticsearch/elasticsearch.yml
```

Do súboru pridáme nasledujúce riadky. Prvý z nich zakáže prístup k inštancii Elasticsearch na porte 9200 (predvolený port), aby sme znemožnili čítanie z databázy alebo jej vypnutie cez HTTP API komukoľvek zvonku. Druhý riadok zakáže dynamické skripty:

```
network.host: localhost
script.disable_dynamic: true
```

Nasledujúcim príkazom nastavíme, aby sa Elasticsearch spúšťal ako daemon (service) pri štarte OS Ubuntu:

```
sudo update-rc.d elasticsearch defaults 95 10
```

Poslednou úpravou v prípade ES bude inštalácia jedného veľmi často používaného pluginu s názvom **head**. Je to webové rozhranie na prehliadanie a interakciu s ES.

```
sudo /usr/share/elasticsearch/bin/plugin -install mobz/elasticsearch-head
```

Plugin bude dostupný na adrese

```
http://localhost:9200/_plugin/head/
```

Localhost v adrese platí v našom prípade. V prípade inej konfigurácie môže byť nahradený hostom, kde beží ES.

Následne sa pustíme do inštalácie nástroja Kibana. Stiahneme si balíček do domovského priečinka a následne ho rozbalíme:

```
cd ~; wget https://download.elasticsearch.org/kibana/kibana/kibana-3.0.1.tar.gz
```

```
tar xvf kibana-3.0.1.tar.gz
```

Otvoríme a upravíme konfiguračný súbor

```
sudo vi ~/kibana-3.0.1/config.js
```

Vyhľadáme riadok ktorý špecifikuje pripojenie k elastic seacrh a nahradíme predvolený port 9200 portom 80. Je to dôležité, pretože plánujeme k nástroju Kibana pristupovať na porte 80 (t.j. http://verejná ip adresa serveru so systémom logstash, napr. http://localhost):

```
elasticsearch: "http://"+window.location.hostname+":80",
```

Potom vytvoríme nový priečinok a skopírujeme súbory na vhodnejšie miesto:

```
sudo mkdir -p /var/www/kibana3
sudo cp -R ~/kibana-3.0.1/* /var/www/kibana3/
```

Kvôli spôsobu, akým Kibana poskytuje používateľovi rozhranie s Elasticsearch (používateľ potrebuje priamy prístup k Elasticsearch), potrebujeme nakonfigurovať nástroj Nginx, aby požiadavky portu 80 presúval na port 9200 (predvolený port, kde počúva Elasticsearch). Kibana našťastie poskytuje konfiguračný súbor Nginx, ktorý väčšinu tohto nastavuje.

Inštalácia Nginx:

```
sudo apt-get install nginx
```

Stiahnutie konfiguračného súboru Nginx do domovského pričinka:

```
cd ~; wget https://gist.githubusercontent.com/thisismitch/2205786838a6a5d61f55/
raw/f91e06198a7c455925f6e3099e3ea7c186d0b263/nginx.conf
```

Otvoríme ho na úpravu:

```
vi nginx.conf
```

Nájdeme hodnotu server_name a zmeníme ju na presne stanovené meno servera (FQDN). V našom prípade je to localhost. Podobne vyhľadáme hodnotu root a zmeníme ju na našu cestu k priečinku, takže to vyzerá nasledovne:

```
server_name localhost;
root /var/www/kibana3;
```

Súbor skopírujeme namiesto predvoleného serverového bloku Nginx:

```
sudo cp nginx.conf /etc/nginx/sites-available/default
```

Nakoniec Nginx reštartujeme:

```
sudo service nginx restart
```

Ako posledný nainštalujeme Logstash. Balíček je dostupný v rovnakom repozitári ako Elasticsearch, a my sme už nainštalovali verejný kľúč, takže nám stačí vytvoriť zoznam zdrojov:

```
echo 'deb http://packages.elasticsearch.org/logstash/1.4/debian stable main' |
sudo tee /etc/apt/sources.list.d/logstash.list
```

Klasická aktualizácia indexu balíčkov:

```
sudo apt-get update
```

Inštalácia nástroja Logstash:

```
sudo apt-get install logstash=1.4.2-1-2c0f5a1
```

Logstash sa nainštaloval, musíme ho však ešte nakonfigurovať. To si ukážeme na ďalších riadkoch, keďže to predstavuje nosnú časť riešenia.

V tomto momente máme nainštalované všetky technológie rovnako ako v obraze operačného systému. Teraz si povieme niečo o jednotlivých systémoch, ich používaní a o ich konfigurácii (v rámci obrazu systému sú vytvorené určité vzorové konfiguračné súbory – tie sa nachádzajú aj na priloženom DVD). Odteraz budeme predpokladať používanie priloženého obrazu systému Ubuntu a jeho načítanie pomocou virtualizačného programu.

Na tomto obraze máme vytvorený používateľský účet **user**, heslo k účtu je \. Rovnaké heslo (\) je prednastavené aj pre superpoužívateľský účet **sudo**.

5.2 Elasticsearch

Systém Elasticsearch je nastavený tak, že sa spúšťa ako daemon spolu so štartom OS. Potom beží a počúva na porte 9200. V našom prípade si môžeme beh systému overiť zadaním http://localhost:9200 do prehliadača. V prípade, že servis beží, dostanem o ňom späť základné informácie. Teraz sa poďme pozrieť na niektoré základné pojmy.

5.2.1 Skupina (cluster)

Skupina je kolekcia jedného alebo viacerých uzlov (serverov), ktoré spolu uskladňujú vaše údaje a vykonávajú federatívne indexovanie a vyhľadávanie naprieč všetkými uzlami. Predvolený názov skupiny je elasticsearch. Názov je dôležitý, pretože ho potrebujeme vedieť, ak chceme do skupiny pripojiť nový uzol. Nie je problém, ak máme iba jeden uzol v rámci jednej skupiny.

5.2.2 Uzol (node)

Uzol je server, ktorý je súčasťou skupiny. Ukladá údaje a podieľa sa na indexovaní a vyhľadávaní údajov. Podobne ako skupina je jednoznačne identifikovaný názvom, konkrétne náhodným menom charakteru vybraného z komiksového univerza Marvel. Názvy sa samozrejme dajú meniť. Pri vytváraní uzlov môžeme špecifikovať, do akej skupiny sa majú pripojiť. Predvolene sa uzly pokúšajú pripojiť do skupiny s názvom elasticsearch (za predpokladu, že sú uzly jeden pre druhý viditeľné). Počet uzlov v skupine nie je obmedzený.

5.2.3 Index (index)

Index je kolekcia dokumentov, ktoré majú nejakým spôsobom podobné charakteristiky. Môžeme mať napríklad index pre používateľské údaje, index pre katalóg produktov a ďalší index pre údaje z objednávok. Index je identifikovaný názvom (napísaným iba malými písmenami) a tento názov sa používa pre referenciách indexu, keď s ním pracujeme. Môže ísť o indexovanie, vyhľadávanie, aktualizácie (update operations) a odstraňovanie dokumentov v tomto indexe. Počet indexov v skupine nie je obmedzený.

5.2.4 Typ (type)

V rámci indexu si môžeme nadefinovať jeden alebo viacero typov. Typ je logická kategória/rozdelenie indexu, ktorého sémantika je na nás. Vo všeobecnosti sa typ definuje pre dokumenty, ktoré majú skupinu zvyčajných polí. Predstavme si napríklad, že prevádzkujeme blogovaciu platformu a ukladáme všetky údaje v rámci jedného indexu. V rámci tohto indexu môžeme definovať typ pre používateľské údaje, typ pre blogové údaje a ďalší typ pre komentáre.

5.2.5 Dokument (document)

Dokument je základná jednotka, ktorá môže byť indexovaná. Môžeme mať napríklad dokument pre jedného konkrétneho zákazníka, ďalší pre konkrétny produkt atď. Dokumenty sú ukladané vo formáte JSON (JavaScript Object Notation), ktorý je štandardom na výmenu informácii na internete. Počet dokumentov v rámci indexu/typu nie je obmedzený.

5.2.6 Takmer reálny čas (near realtime)

Elasticsearch je vyhľadávacou platformou, ktorá pracuje takmer v reálnom čase. To znamená, že dokumenty sú v skutočnosti vyhľadávateľné s malým oneskorením po zaindexovaní. Zvyčajne ide o jednu sekundu.

5.2.7 REST API

Komunikácia a interakcia s uzlom, respektíve so skupinou, prebieha v Elasticsearch pomocou REST API. Prostredníctvom tohto API je možné:

- zistiť zdravie, stav a štatistiky skupiny, uzlu alebo indexu;
- spravovať údaje a metaúdaje skupiny, uzlu a indexu;
- vykonávať operácie CRUD (Create, Read, Update, Delete – vytvoriť, čítať, aktualizovať, odstrániť) a vyhľadávať v indexoch;
- vykonávať pokročilé operácie vyhľadávania, napríklad zoradzovanie, filtrovanie, skriptovanie, agregácie a iné.

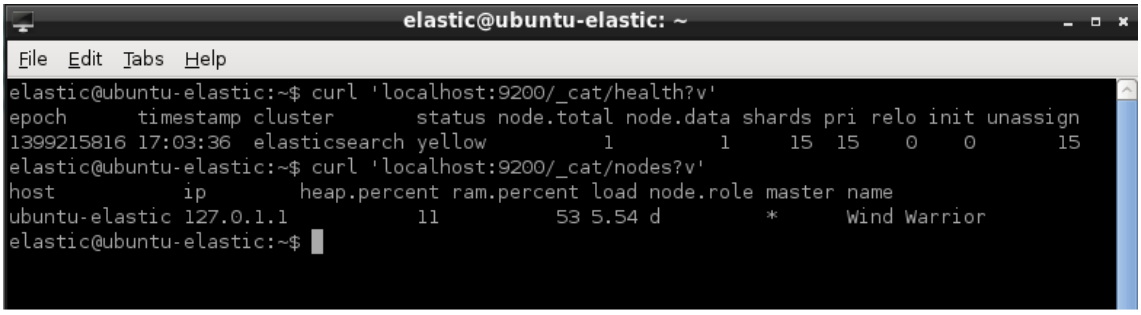
Ako sme už spomínali vyššie, Elasticsearch beží predvolene na porte 9200. Uvedieme si niekoľko užitočných príkazov, ktoré využijeme pri práci s Elasticsearch. Predpokladáme, že pracujeme v prostredí obrazu OS Ubuntu, ktorý sme virtualizovali. Inak by localhost v príkladoch nahradila adresa IP.

```
curl 'localhost:9200/_search?pretty=1&q=*'
```

Kúsok kódu _search použije API na vyhľadávanie, pretty=1 znamená, že máme zapnutý formátovaný výpis a q= znamená žiadosť (query), čo v je v našom prípade *, a teda všetko. Tento príkaz by teda vrátil formátované všetky záznamy v databáze.

```
curl 'localhost:9200/_cat/health?v'
```

API _cat sa používa napríklad na zisťovanie zdravia skupiny. Odpovede na tento príkaz a následne na príkaz, ktorým zisťujeme zdravie uzlov priamo z obrazu implementovaného riešenia v systéme Ubuntu môžeme vidieť na obrázku 5.1. Podrobnosti o jednotlivých príkazoch a celú používateľskú príručku nájdeme na týchto stránkach [11].



Obr. 5.1: Odpovede na príkazy zisťujúce zdravie skupiny a uzlov Elasticsearch

5.3 Logstash

Logstash taktiež beží po spustení OS ako daemon. Je možné spustiť ho aj z príkazového riadka. Najprv sa pozrieme na základné príkazy a následne sa prepracujeme k príkazom s hotovou konfiguráciou. Predtým si vysvetlíme ako vlastne Logstash funguje.

5.3.1 Potrubie (pipeline)

Základným stavebným kameňom pri konfigurácii nástroja Logstash sú 4 kategórie parametrov: vstupy (input), výstupy (outputs), kodeky (codecs) a filtre (filters). Vytvorením potrubia na spracovanie udalostí je Logstash schopný vydolovať z logov relevantné informácie a sprístupniť ich nástroju Elasticsearch na uloženie a vyhľadávanie v nich. Teraz sa podrobnejšie pozrieme na najpoužívanejšie možnosti nastavenia potrubia. Kompletné možnosti konfigurácie nájdeme na stránkach dokumentácie [24].

5.3.2 Vstupy (inputs)

Vstupy slúžia na pridávanie logov do nástroja Logstash. Najpoužívanejšie sú tieto:

- **file:** číta zo súboru v súborovom systéme, podobne ako unixový príkaz „tail -0a“. Tento vstup detekuje a obsluhuje aj rotáciu logových súborov;
- **syslog:** načúva na dobre známom porte 514 pre správy typu syslog a parsuje ich podľa formátu RFC3164;
- **redis:** číta zo serveru typu redis, používajúc kanály aj zoznamy redis. Redis sa často používa ako „prostredník“ v centrálnej inštalácii na spracovanie udalosti od vzdialených Logstash „dodávateľov“;
- **lumberjack:** spracováva udalosti vo formáte protokolu lumberjack. V novších verziách sa nazýva logstash-forwarder.

5.3.3 Filtre (filters)

Filtre sú reťazi Logstash používané ako prostredné nástroje, ktoré slúžia na spracovanie údajov. Často sú kombinované s podmienkami, preto umožňujú vykonať určitú v udalosti, ak tá spĺňa určité kritériá. Medzi najužitočnejšie patria:

- **grok:** parsuje ľubovoľný text a štruktúruje ho. Grok v súčasnosti predstavuje najlepšiu možnosť ako rozparsovať logovú údaje do štruktúrovaného a prehľadávateľného textu. V nástroji Logstash nájdeme viac ako 140 predlôh (patterns),
- **mutate:** tento filter umožňuje v poliach vykonať všeobecné zmeny, ako sú premenovanie, odstránenie, nahradenie či modifikácia daného poľa v udalosti,
- **drop:** umožňuje odstrániť udalosť úplne, napríklad udalosti typu debug,
- **clone:** vytvorí kópiu udalosti s možnosťou pridať či odstrániť polia,

- **geoip:** pridáva informácie o geografickej polohe IP adries (a umožňuje zobrazenie pekných diagramov v nástroji Kibana).

5.3.4 Výstupy (inputs)

Výstupy predstavujú finálnu časť potrubia. Udalosť môže počas spracovania prejsť viacerými výstupmi, po skončení zoznamu je spracovanie udalosti skončené. Najpoužívanejšie sú:

- **elasticsearch:** ukladanie údajov do databázy Elasticsearch. Je to najjednoduchšia možnosť na následné zobrazenie a prehľadávanie uložených údajov,
- **file:** zapisuje udalosti do súboru na disku,
- **graphite:** odosiela údaje do populárneho open source nástroja graphite,
- **statsd:** služba, ktorá načúva a čaká na štatistiky, ako počítadla a časovače, odoslané cez UDP a odosiela množiny ďalším službám. Vhodné najmä pre používateľov, ktorí už službu statsd používajú.

5.3.5 Kodeky (codecs)

Kodeky sú v podstate základné filtre, ktoré môžu fungovať ako časť vstupu alebo výstupu. Kodeky umožňujú jednoducho oddeliť transport správ od serializačného procesu. Najpopulárnejšie sú tieto:

- **json:** zakódovanie z rozkódovanie údajov vo formáte JSON,
- **multiline:** sa stará o spájanie viacriadkových udalostí, ako sú napríklad hromadné sledovacie správy (stacktrace messages) v jazyku Java.

5.3.6 Prípadová štúdia konfigurácie

Na nasledujúcich riadkoch si ukážeme ako spúšťať nástroj Logstash. Veľká sila tohto nástroja je práve v tom, akú konfiguráciu použijeme. Začneme tým najjednoduchším príkazom:

```
/opt/logstash/bin/logstash -e 'input { stdin { } } output { stdout {} }'
```

Príznak **-e** nám umožní prijímať konfiguráciu priamo z príkazového riadka. Vo zvyšku príkazu definujeme ako vstup štandardný vstup stdin a ako výstup štandardný výstup stdout. Filtre ani kodeky sa vo vyššie uvedenom príkaze nevyskytujú. Táto konfigurácia neurobí nič iné, ako vypísanie textu, ktorý vpíšeme do príkazového riadka, v štruktúrovanom formáte.

Uveďme si zložitejší príklad:

```
/opt/logstash/bin/logstash -e 'input { stdin { } } output { stdout { codec => rubydebug } }'
```

Oproti predchádzajúcemu príkazu sme na výstupe definovali kodek rubydebug, ktorý vypíše výstup údaje udalosti používajúc knižnicu Ruby Awesome Print. Po vpísaní textu „logstash test“ sa zobrazí toto:

```
logstash test
{
  "message" => "logstash test",
  "@timestamp" => "2014-04-26T23:48:05.335Z",
  "@version" => "1",
  "host" => "my-laptop"
}
```

Pridaním kodeku na výstupe teda môžeme zmeniť formát výstupu celého nástroja Logstash. Poďme sa pozrieť na ďalší príklad, ktorý už zahŕňa aj databázu Elasticsearch. Príkaz počíta s tým, že nástroj Elasticsearch je na danom stroji spustený.

```
/opt/logstash/bin/logstash -e 'input { stdin { } } output { elasticsearch { host => loca
```

Ak niečo napíšeme do konzoly teraz, na výstupe konzoly sa nezobrazí nič, pretože logstash ako výstup používa Elasticsearch, t. j. správa sa uloží tam.

Teraz sa pozrieme na niečo, čo už môže byť užitočné. Dostávame sa ku konfigurácii pomocou perzistentných konfiguračných súborov. Najprv sa pozrieme na to, ako vyzerá prázdny konfiguračný súbor:

```
input {  }
filters {  }
output {  }
```

Vidíme tri kategórie parametrov – vstup, filtre a výstup. Kodeky sa následne aplikujú do týchto kategórií. Na predchádzajúcich riadkoch sme Logstash spúšťali s konfiguráciou z príkazového riadka. Teraz si konfiguráciu vložíme do súboru a Logstash si ju následne z tohto súboru načíta. To určí, čo sa bude diať s prichádzajúcimi logmi. Súbor pomenujeme „logstash-usecase.conf“. Na začiatku je konfiguračný súbor prázdny, ako sme videli vyššie. Teraz si ukážeme, ako ho naplniť, aby sledoval na vstupe súbor /var/log/custom/file.log a zapisoval logy z neho do databázy Elasticsearch. Logy v tomto súbore budú v nasledujúcom formáte:

```
10.121.123.104 kevin - [29/Nov/2014:21:01:04 +0100] "GET c.p.k.ClassName HTTP/1.1" 200
10.121.123.104 - - [29/Nov/2014:21:01:17 +0100] "GET etc.log.java.Author HTTP/1.1" 302
10.121.123.104 user main-PC [29/Nov/2014:21:01:18 +0100] "GET cvut.cz.Book HTTP/1.1" 302
10.121.123.104 - - [29/Nov/2014:21:01:18 +0100] "GET a.b.c.Example HTTP/1.1" 302
```

Je to vymyslený formát, ktorý nedáva veľký zmysel, dobre ukazuje však, čo všetko Logstash zvládne. Pozrime sa na tento formát bližšie. Na začiatku vidíme klientskú IP adresu stroja, ktorý požiadavku zadal. Nasledujú údaje o používateľovi a stroji, na ktorom používateľ požiadavku zadal. Ďalší je časový údaj. Potom vidíme počiatočné úvodzovky a za nimi metódu požiadavku, triedu jazyka Java a verziu HTTP. Po ukončujúcich úvodzovkách nasleduje odpoveď servera a počet bytov. Vidíme, že používateľ, stroj a počet bytov môže nahradiť pomlčka.

Vráťme sa späť k nášmu súboru a začnime ho napĺňať, aby vykonával to, čo potrebujeme. Najprv vyplníme požadovaný vstup a výstup. Potrebujeme na vstupe načítať súbor z určitého

umiestnenia, spracovať ho a zapísať do Elasticsearch. Pozrime sa na stránky s pluginmi Logstash [26] a skúsime vybrať niečo vhodné. Logstash samozrejme obsahuje plugin na čítanie logov zo súboru s názvom file. Zanesme ho do nášho konfiguračného súboru. Syntax zápisu je parameter => hodnota, prípadne parameter => "hodnota". Budeme musieť identifikovať aj cestu k súboru (povinné) a označíme si aj typ logu, aby sme vedeli, že pochádza z tohto vstupu (voliteľné):

```
input {
  file {
    path => "/var/log/custom/file.log"
    type => "custom_log"
  }
}
filters {  }
output {  }
```

Logstash po spustení s týmto konfiguračným súborom začne sledovať súbor file.log a v prípade, že sa v ňom objavia logy, niečo s nimi urobí (zatiaľ neurobí nič, pretože sekcie filters a output v konfiguračnom súbore sú prázdne). Poďme nastaviť zapisovanie logov do databázy Elasticsearch. V dokumentácii nájdeme výstupný plugin elasticsearch (prípadne by sme mohli použiť aj elasticsearch_http pre staršie verzie a zápis vo formáte HTTP/REST). Jeho nastavenie je veľmi jednoduché a konfiguračný súbor potom vyzerá takto:

```
input {
  file {
    path => "/var/log/custom/file.log"
    type => "custom_log"
  }
}
filters {  }
output {
  elasticsearch { host => localhost }
}
```

V tejto podobe konfiguračný súbor spôsobí to, že vezme všetky riadky, ktoré sa objavia v súbore file.log a nerozparsované ich zapíše do databázy elasticsearch, ktorá beží na lokálnom stroji. Poďme sa teraz pozrieť na tú najzaujímavejšiu časť a náš vymyslený logový formát dôkladne rozparsujme. Najpoužívanejším filtrom na parsovanie je v Logstashi filter grok. Obsahuje veľké množstvo predpripravených vzorov pre rôzne technológie [27] a možnosť vytvárať si vlastné pomocou jazyka regulárnych výrazov. To si ukážeme neskôr. Syntax zápisu je vo filtri grok je match => ["message", "vzor"]. Vzor sa potom zapisuje ako %{PATTERN:IDENTIFIER}. Najlepšie bude ukázať si príklad. Veľkou pomocou pri ladení a parsovaní logov pomocou filtra grok sú tieto webové stránky s nástrojom grok-debugger [15]. Používanie je veľmi jednoduché. Do horného poľa zadáme príklad logového riadka v požadovanom formáte a do dolného poľa potom vpisujeme parsovacie vzory z stránok uvedených vyššie. Môžeme si zapnúť aj funkciu automatického dokončovania. V prípade, že

vpísaný vzor nachádza zhodu v hornom riadku (v logu), zobrazí sa nám to v tele stránky. Postupne tak môžeme pohodlne vytvoriť parsovacie vzory pre akékoľvek kombinácie údajov v logu.

Začneme teda parsovaním nášho formátu logov. Na vyššie uvedenom odkaze nájdeme vhodný vzor. Na začiatku záznamu nášho logu je vždy IP adresa klienta, odkiaľ požiadavka prichádza. Použijeme teda vzor IPORHOST a zapíšeme to takto:

```
input {
  file {
    path => "/var/log/custom/file.log"
    type => "custom_log"
  }
}
filters {
  grok {
    match => [ "message", "^%{IPORHOST:clientip}"]
  }
}
  output {
  elasticsearch { host => localhost }
}
```

Tento zápis spôsobí niekoľko vecí. Na začiatku očakávaného vzoru musí byť IP adresa alebo host. Logstash pre ňu vytvorí značku clientip. Musí byť na začiatku vždy. O to sa stará znak ^ ktorý pochádza z jazyka regulárnych výrazov (jeho úlohou je práve to, že údaj na začiatku výrazu je povinný). Druhým údajom v poradí nášho formátu logu je názov používateľa daného počítača. Zdá sa však, že sa neobjavuje vždy. Musíme preto filter nastaviť tak, aby bol tento údaj voliteľný a namiesto neho sa mohla objaviť pomlčka. Vhdoným vzorom bude vzor USER.

```
input {
  file {
    path => "/var/log/custom/file.log"
    type => "custom_log"
  }
}
filters {
  grok {
    match => [ "message", "^%{IPORHOST:clientip} (?:-|%{USER:ident})"]
  }
}
output {
elasticsearch { host => localhost }
}
```

Vidíme, že náš vzor presne zodpovedá formátu logu. IP adresa, medzera, názov používateľa alebo pomlčka (voľbu zabezpečuje ternárny operátor). Podobným vyhľadávaním medzi predvolenými vzormi sa dopracujeme k finálnej podobe vzorca pre náš formát:

```
10.121.123.104 kevin - [29/Nov/2014:21:01:04 +0100] "GET c.p.k.ClassName HTTP/1.1" 200

^%{IPORHOST:clientip} (?:-|%{USER:ident}) (?:-|%{USER:auth}) \[%{HTTPDATE:timestamp}\] \"
(?:%{WORD:verb} %{JAVACLASS:class}(?: HTTP/%{NUMBER:httpversion})?|-)\" %{NUMBER:response}
```

Vidíme, že napríklad v prípade uvedenej triedy jazyka Java nemôžeme použiť vzor WORD. Je to preto, že oddeľovačom je bodka, čo by pri formáte zápisu tried jazyka Java spôsobilo problém. V príklade uvedenom vyššie by sme objavili iba prvé c. Potom by sa filter grok zastavil, pretože vzor by nepasoval na danú správu. Preto namiesto toho použijeme vzor JAVACLASS.

Finálny konfiguračný súbor teda bude vyzerať takto:

```
input {
  file {
    path => "/var/log/custom/file.log"
    type => "custom_log"
  }
}
filters {
  grok {
     match => [ "message", "^%{IPORHOST:clientip} (?:-|%{USER:ident}) (?:-|%{USER:auth})
     \[%{HTTPDATE:timestamp}\] \"(?:%{WORD:verb} %{JAVACLASS:class}
     (?: HTTP/%{NUMBER:httpversion})?|-)\" %{NUMBER:response}"]
  }
}
output {
elasticsearch { host => localhost }
}
```

Vytvorili sme si konfiguračný súbor pre vlastný formát logu. Logstash ho teraz bude parsovať a ukladať do databázy. Pozrime sa, čo sa teda stane po vložení prvého riadka do pripraveného súboru a rozparsovaní nástrojom Logstash. Vidíme to v tabuľke 5.2.

5.4 Kibana

Rozhranie nástroja Kibana môžeme vidieť na obrázku 4.9. Navrchu vidíme možnosti filtrovania podľa časových úsekov. Najnovšia verzia Kibany (nainštalovaná na obraze OS Ubuntu nám dovoľuje vytvoriť si aj vlastné filtre). Pod týmito možnosťami sa nachádza panel vyhľadávania. Cez tento panel Kibana veľmi rýchlo fulltextovo prehľadáva spracované udalosti, typy, značky atď. Nasleduje graf s pridanými udalosťami a potom sekcia udalostí, kde môžeme začiarknutím vyberať jednotlivé polia či študovať kompletné znenie jednotlivých udalostí. Všetky filtre a vyhľadávania sa samozrejme prejavujú na zobrazených položkách.

Na záver ešte raz zopakujeme, že implementácia konfiguračného súboru je momentálne nastavená určitým spôsobom. Nástroj Logstash parsuje všetky tri typy logov podľa tejto

10.121.123.104 kevin - [29/Nov/2014:21:01:04 +0100] "GET c.p.k.ClassName HTTP/1.1" 200 1272	
httpversion	1.1
timestamp	29/Nov/2014:21:01:04·+0100
auth	
clientip	10.121.123.104
bytes	1272
response	200
ident	kevin
verb	GET
class	c.p.k.ClassName

Obr. 5.2: Tabuľka zobrazujúca rozparsovaný jeden riadok nášho formátu logu

konfigurácie. Konfigurácia nie je v žiadnom prípade konečná, je možné (a pravdepodobne aj potrebné) ju upravovať podľa reálne nasadených údajov. V tom ale spočíva komplexnosť tohto riešenia. Môže slúžiť ako na mieru šité riešenie pred predvolené nastavenia logových súborov a veľmi jednoducho ho môžeme modifikovať, pridať na spracovanie nové typy logov, vytvárať nové polia v databáze atď.

5.5 Konfiguračný súbor main.conf

V tejto sekcii si predstavíme obsah hlavného konfiguračného súboru s názvom „main.conf“, ktorý zodpovedá za spracovanie logov zo všetkých troch technológií uvedených v špecifikácii. Konfigurácia nástroja Logstash je témou tak rozsiahlou, že by vystačila na napísanie ďalšej diplomovej práce. Preto jej venujeme osobitnú sekciu. Konfiguráciu je možné rýchlo a jednoducho modifikovať a upravovať podľa požiadaviek. Obsah súboru vyzerá takto:

```
input {
  file {
    path => "/var/log/elk/apache/apache_access.1"
    type => "apache_access"
  }
  file {
    path => "/var/log/elk/apache/apache_access.2"
    type => "apache_access"
  }
  file {
    path => "/var/log/elk/apache/apache_access.3"
    type => "apache_combined"
  }
  file {
    path => "/var/log/elk/apache/apache_error.log"
    type => "apache_error"
  }
  file {
    path => "/var/log/elk/apache/postgresql.log"
    type => "postgresql"
  }
```

```
  file {
    path => "/var/log/elk/apache/jboss.log"
    type => "jboss"
  }
}

filter {
  if [type] == "apache_access" {
    ################################################
    ##   APACHE
    ################################################
    grok {
      match => { "message" => "%{COMMONAPACHELOG}" }
    }
    date {
    # in this format: 2014-04-23T11:27:14.177+0200
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [type] == "apache_error" {
    grok {
      match => { "message" => "%{COMMONAPACHELOG}" }
    }
    date {
    # in this format: 2014-04-23T11:27:14.177+0200
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [type] == "apache_combined" {
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
    # in this format: 2014-04-23T11:27:14.177+0200
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [type] == "postgresql" {
    ################################################
    ##   POSTGRESQL
    ##
    ## Assumes the log prefix is set to:
    ## log_line_prefix = '%t [%p]: [%l-1] user=%u-%h,db=%d '
    ## which is the pgbadger 'default'
    ################################################
    ## Join the lines

    multiline {
      pattern => "^\s"
      what => "previous"
    }
    ## Ignore backups
    if "COPY" in [message] { drop {} }
    ##
    ## Tag lines and parse additionally
    ##
    else if "duration" in [message] {
      if "execute" in [message] {
      mutate { add_tag => ["sql-query"] }
```

```
    grok {
      match => ["message", "%{DATESTAMP:timestamp} CET \[%{POSINT:pid}\]: \[%{POSINT}-1]
      user=%{NOTSPACE:connection_id} %{WORD:log_level}:  duration: %{NUMBER:execution_ms} ms
      execute %{GREEDYDATA:sql-statement}"]
    }
  } else if "parse" in [message] {
      mutate { add_tag => ["sql-parse"] }
      grok {
        match => ["message", "%{DATESTAMP:timestamp} CET \[%{POSINT:pid}\]: \[%{POSINT}-1]
        user=%{GREEDYDATA:connection_id} %{WORD:log_level}:  duration: %{NUMBER:parse_ms} ms
        parse %{GREEDYDATA:sql-statement}"]
      }
  } else if "bind" in [message] {
      mutate { add_tag => ["sql-bind"] }
      grok {
        match => ["message", "%{DATESTAMP:timestamp} CET \[%{POSINT:pid}\]: \[%{POSINT}-1]
        user=%{GREEDYDATA:connection_id} %{WORD:log_level}:  duration: %{NUMBER:bind_ms} ms
        bind %{GREEDYDATA:sql-statement}"]
      }
    }
  } else {
    mutate { add_tag => "db-message" }
    grok {
      match => ["message", "%{DATESTAMP:timestamp} CET \[%{POSINT:pid}\]: \[%{POSINT}-1]
      user=%{NOTSPACE:connection_id} %{WORD:log_level}: %{GREEDYDATA:message}"]
    }
  }
  date {
  # in this format: 2014-04-23T11:27:14.177+0200
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
} else if [type] == "jboss" {
  ################################################
  ##   JBOSS
  ## Assuming we ar logging something like this:
  ## ~dc-devt 2013-12-06T17:43:04.234+0100 [0.0.0.0-http-10.32.92.147-8080-3]
  INFO  b.v.a.d.l.PreProcessLoggingInterceptor -
  ## Service: GET http://10.32.92.147:8080/appContext/rest/service
  ## UserId: itsmeagain
  ## Response types application/json
  ## Query Parameters:
  ##  limit -> [10]
  ##  sortColumn -> [number]
  ##  start -> [0]
  ## Path parameters:
  ## Reply type: class myapp.PagedList
  ## Output document:
  ## {...contents snipped...}
  ## Duration: 0.078s
  ################################################
  multiline {
    pattern => "^[\[~]"
    negate => true
    what => "previous"
  }
  grok {
```

```
      match => [ "message", "~%{NOTSPACE:application} %{TIMESTAMP_ISO8601:timestamp}
      \[%{DATA:server}\-%{DATA:thread}\] %{LOGLEVEL:severity}\s+%{JAVAFILE:category}
      \- %{GREEDYDATA:shortmessage}"]
    }

    grok {
      match => [ "message", "Duration: %{NUMBER:duur:float}s" ]
      tag_on_failure => [_grokparsefailure]
    }

    grok {
      match => [ "message", "UserId: %{WORD:ldapid}" ]
      tag_on_failure => [_grokparsefailure]
    }

    grok {
      match => [ "message", "Service: %{WORD:http_command} %{URI:endpoint}" ]
      tag_on_failure => [_grokparsefailure]
    }

    date {
    # in this format: 2014-04-23T11:27:14.177+0200
      match => ["timestamp", "yyyy-MM-dd'T'HH:mm:ss.SSSZ"]
    }
  }

  if "_grokparsefailure" in [tags] { drop {} }
}

output {
  stdout { codec => rubydebug }
  elasticsearch {
    host => localhost
  }
}
```

Na vstupe definujeme cestu k jednotlivým súborom (predpokladáme, že sa nazývajú apache_access.1, apache_access.2, apache_access.3, apache_error.log, postgresql.log a jboss.log). Logstash tieto súbory sleduje a spracováva zmeny. Následne každému druhu logu priradíme príslušný typ. Sekcia filter definuje, ako budeme postupovať pri spracovaní jednotlivých typov logov. Používame tu základné filtre a kodeky, ktoré Elasticsearch ponúka. Využívame napríklad filter grok, konkrétne jeho metódu match, ktorá nám dovolí vybudovať si vlastné regulárne výrazy, pomocou ktorých môžeme logové udalosti pekne rozparsovať do jednotlivých polí. Okrem toho používame napríklad kodek multiline, ktorý nám umožní spracovať logové udalosti, ktoré zaberajú aj niekoľko riadkov. Logy zo serveru Apache rozparsujeme podľa formátu Common Format alebo Combined Log Format a definujeme, aby sa za časový údaj udalosti zvolil reálny čas prebehnutia udalosti a nie čas jej spracovania. Pri databázových logoch PostgreSQL najprv definujeme pomocou príkazu multiline, že môžeme spracovať aj viacriadkové logy, potom ignorujeme udalosti, ktoré obsahujú výraz COPY (zálohy) a následne ich podľa obsahu správy delíme na ďalšie inštancie. Každej inštancii pridáme značku (tag). V každej správe potom očakávame určité formátovanie a odchytávame jednotlivé zložky. Konkrétne ide o časový údaj a časové pásmo (pevne nastavené na CET), ID procesu, ID pripojenia, úroveň správy, trvanie požiadavky a text požiadavky. Je

nutné pripomenúť, že konfiguračný súbor a filter grok počíta s určitou konfiguráciou logových súborov. V prípade, že formát logu nebude dodržaný, filter grok zlyhá a správa nebude rozparsovaná do dodatočných polí. Udalosti sa buď priradí značka _grokparsefailure alebo sa celý riadok zapíše ako jedna správa (bez rozparsovania do jednotlivých polí). Udalosti so značkou _grokparsefailure následne nezapisujeme do databázy. V prípade logov zo servera JBoss takisto očakávame určitý formát logov. Pre iné formáty logov musíme zvoliť inú konfiguráciu. Toto je základná konfigurácia, ktorá pracuje s predvolenými hodnotami logovania. Pre logy zo servera JBoss takisto očakávame viacriadkové logy (Java stacktrace messages). Následne udalosť ďalej parsujeme a pomocou regulárnych výrazov hľadáme reťazce duration, userId a service.

5.6 Riešenia častých problémov

Táto sekcia slúži na náčrt riešení s častými problémami a otázkami týkajúcimi sa riešenia ELK. Táto problematika je veľmi rozsiahla a nie všetko sme schopní pokryť v rámci témy DP. Preto tieto sekcie iba načrtneme a nebudeme zachádzať do podrobností. Riešenie jednotlivých problémov aj tak závisí od mnohých faktorov a je dôležité poznať presné špecifikácie nasadenia (tieto špecifiká momentálne nepoznáme – napríklad nastavenie iných serverov, z ktorých budeme zbierať logové súbory). Nasledujúce riadky slúžia skôr ako ukážka toho, že ELK ponúka aj na tieto otázky odpovede.

5.6.1 Dôležité umiestnenia

Nie je na škodu poznať v prípade potreby umiestnenia jednotlivých konfiguračných a binárnych súborov našich technológií. Uvádzame preto nasledujúci zoznam.

Elasticsearch

- Binárne súbory: /usr/share/elasticsearch
- Správca pluginov: /usr/share/elasticsearch/bin/plugin
- Konfigurácia: /etc/elasticsearch/elasticsearch.yml
- Údaje: /var/lib/elasticsearch/<názov-skupiny>

Logstash

- Binárne súbory: /opt/logstash
- Správca pluginov: /usr/share/elasticsearch/bin/plugin
- Konfigurácia: /etc/logstash/conf.d

Kibana Kibana je „len“ skupina súborov HTML a JavaScriptu. Po stiahnutí archívu sme si ju počas inštalácie komponentov rozbalili a premiestnili do pripraveného adresára.

- Umiestnenie: /var/www/kibana3/

5.6.2 Rotácia logov

Logstash podporuje rotáciu logov automaticky. Keď používame vstup file, ten sleduje príslušný súbor a zmeny v ňom. Je naprogramovaný tak, aby bez problémov zvládol výmenu za nový súbor.

5.6.3 Spracovanie logov z viacerých serverov

Túto bežnú požiadavku rieši v prípade ELK doplnok logstash-forwarder (predtým lumberjack). Je to malý nástroj napísaný v jazyku Go umožňujúci bezpečne prenášať komprimované logové údaje s minimálnymi nárokmi na zdroje prostredníctvom protokolu Lumberjack. Funguje tak, že po inštalácii nástroja Logstash na domovský server vygenerujeme SSL certifikát a súkromný kľúč. Logstash potom nakonfigurujeme, aby na vstupe naslúchal protokol lumberjack na určitom porte a očakával SSL certifikát a kľúč. Na vzdialené servery, z ktorých chceme zbierať a odosielať logy, nainštalujeme logstash-forwarder a nakonfigurujeme mu ako cieľovú destináciou domovský server s nástrojom Logstash (resp. jeho IP adresu a port) a na každý z týchto serverov skopírujeme aj SSL certifikát. Podrobný návod s uvedeným príkladom konfigurácie nájdeme tu [14].

Kapitola 6

Testovanie

Konfigurácia počítača použitého na testovanie bola nasledujúca:

- CPU: Intel Core i7-3520M 2,9 GHz
- Pamäť RAM: 12 GB (pre obraz OS Ubuntu s nainštalovanými nástrojmi boli vyhradené 4 GB)
- Pevný disk: Hitachi, 7200 ot./min.
- OS: Win 8 Professional 64 bit

Testovanie prebiehalo v programe na virtualizáciu Oracle VM VirtualBox vo verzii 4.3.20. Virtualizovaný bol obraz OS Ubuntu 14.04 64bit (vo formáte .ova) s nainštalovanými verziami nástrojov Elasticsearch 1.1.1, Logstash 1.4.2 a Kibana 3.0.1. Testovanie prebiehalo spôsobom BlackBox, teda testovací scenár na základe vstupných hodnôt očakáva nejaké návratové hodnoty. Všetky testovacie scenáre počítajú so spustenými nástrojmi Elasticsearch a Logstash s konfiguračným súborom main.conf (ak nie je uvedené inak). Obmieňané sú vstupné údaje. Prvé štyri testovacie scenáre sa zameriavajú na to, či je správne nastavený konfiguračný súbor. Testujeme s malým objemom údajov z každej technológie (1 – 10 riadkov) a len overujeme správne rozparsovanie. Zvyšné scenáre sa zameriavajú na rýchlosť spracovania väčšieho objemu údajov. Testovanie overujeme podľa rozhrania Kibana (http://localhost:80). Po načítaní základnej obrazovky zvolíme možnosť Logstash Dashboard. Pri prehliadaní odporúčame zvoliť rozsah posledných 7, 30 dní či vlastnú hodnotu (Kibana predvolene nastaví iba posledný deň), aby sa zobrazili všetky logy, ktoré v databáze existujú. Existujúce logy totiž pokrývajú rozsah dátumov 27.1.2014 až 10.1.2015. Druhým nástrojom na kontrolu vložených logov je plugin head (http://localhost:9200/_plugin/head/). Po otvorení je najlepšie prepnúť sa hore na kartu Browser, kde naboku vidíme jednotlivé indexy (Elasticsearch si vytvára pre každý deň nový index) a jednotlivé udalosti v nich. Intuitívne rozhranie nám umožní udalosti filtrovať.

Je nutné poznamenať ešte jednu skutočnosť. Je jasné, že počas testovania riešenia sme skúšali rôzne verzie logov, formátov atď. Všetky logové súbory boli do Ubuntu skopírované, neboli vytvorené programom bežiacom na tomto OS. Počas toho sme objavili nasledujúci problém (bug v programe Logstash). Logstash si o každom sledovanom súbore vytvorí v

adresári /home skrytý súbor s názvom since_db+id_súboru. Tieto súbory slúžia na to, aby Logstash vedel, pokiaľ v príslušnom súbore už logy rozparsoval. Je to teda niečo ako zarážka. Keď sme počas testovania jednotlivé súbory s logmi otvorili v aplikácii vim a súbor upravili (vymazali, pozmenili či dodali riadky), Logstash po uložení rozparsoval celý súbor odznova. Nepamätal si teda miesto, kde skončil a vytvoril kópie niektorých riadkov. To nastalo preto, lebo ručné otvorenie súborov s logmi v aplikácii vim nejakým spôsobom znehodnotilo údaj o pozícii v súbore since_db. Bližšie je bug popísaný na tomto fóre [5]. Preto sme si vytvorili pomocné súbory, ktoré sme mohli upravovať ručne, a požadovaný obsah sme do sledovaných súborov pripojili pomocou príkazu cat, napr.

```
sudo bash -c 'cat postgresql_bkp.log >> postgresql.log'
```

6.1 Testovací scenár č. 1

Konfigurácia: Štandardný vstup, bez konfiguračného súboru a filtrov, výpis na štandardný výstup v štruktúrovanej podobe pomocou kodeku rubydebug. Logstash ako daemon bol ručne zastavený a spustený nasledujúcim príkazom.

```
/opt/logstash/bin/logstash -e 'input { stdin { } } output
{ stdout { codec => rubydebug } }'
```

Vstupné údaje: Textový reťazec „testovaci scenar 1“

Očakávaný výsledok: Štruktúrovaný výpis, kde vstupné údaje predstavujú pole správa udalosti a ďalšie polia, napríklad časový údaj.

Získaný výsledok:

```
{
  "message" => "testovaci scenar 1",
  "@version" => "1",
  "@timestamp" => "2015-01-02T20:10:07.183Z",
  "host" => "user-main"
}
```

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný. Všimnime si, že reťazec „testovaci scenar 1“ nenájdeme ani v databáze, pretože spúšťací príkaz použil ako výstup iba obrazovku, nie databázu Elasticearch.

6.2 Testovací scenár č. 2

Konfigurácia: Vstup zo súboru apache_access.2, filter parsujúci logy Apache, konfiguračný súbor test2.conf, výpis na štandardný výstup v štruktúrovanej podobe pomocou kodeku rubydebug, zápis do databázy Elasticsearch.

```
sudo /opt/logstash/bin/logstash -f /etc/logstash/conf.d/test2.conf
```

Vstupné údaje: Prvý riadok v súbore apache_access.2 „71.141.244.242 - aa [18/May/2011:01:48:10 -0700] "GET /admin HTTP/1.1"301 566 Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"“. Riadok bol zo súboru následne odstránený kvôli ďalšiemu testovaniu. Záznam v databáz zostal, pomocou pluginu head ho môžeme vidieť v indexe logstash-2011.05.08.

Očakávaný výsledok: Štruktúrovaný výpis udalosti zo serveru Apache vo formáte CLF a zápis tejto udalosti do databázy.

Získaný výsledok:

```
{
        "message" => "71.141.244.242 - aa [18/May/2011:01:48:10 -0700] \"GET
        /adminaaaaa HTTP/1.1\" 301 566 \"-\" \"Mozilla/5.0 (Windows; U;
        Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3\"",
       "@version" => "1",
     "@timestamp" => "2011-05-18T08:48:10.000Z",
           "type" => "apache",
           "host" => "ubuntu-elastic",
       "clientip" => "71.141.244.242",
          "ident" => "-",
           "auth" => "aa",
      "timestamp" => "18/May/2011:01:48:10 -0700",
           "verb" => "GET",
        "request" => "/admin",
    "httpversion" => "1.1",
       "response" => "301",
          "bytes" => "566",
       "referrer" => "\"-\"",
          "agent" => "\"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
          rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3\""
}
```

Zápis sa úspešne uložil aj do databázy Elasticsearch. Súbor test2.conf bol zo zložky /etc/logstash/conf.d/ odstránený, aby nekolidoval s hlavným súborom main.conf

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

6.3 Testovací scenár č. 3

Konfigurácia: Logstash a Elasticsearch bežiace ako daemony, vstup zo súboru postgre-sql.log, konfiguračný súbor main.conf, zápis do databázy Elasticsearch.

Vstupné údaje: Obsah súboru postgre.sql. Obsah tu neuvádzame, pretože je veľmi rozsiahly.

Očakávaný výsledok: Automatický zápis do databázy Elasticsearch.

Získaný výsledok: Uvádzame iba prvý riadok.

```
{
        "message": "26-12-2014 16:44:49 CET [4470]: [1-1] user=postgres,db=mydb LOG:
        duration: 4.550 ms  parse statement: SELECT id FROM location WHERE name =
        'enigma->/var/log/messages' AND server_id = '1'",
        "@version": "1",
        "@timestamp": "2015-01-04T02:27:26.869Z",
        "type": "postgresql",
        "host": "user-main",
        "path": "/var/log/elk/apache/postgresql.log",
        "tags": [
        "sql-parse"
        ],
        "timestamp": "26-12-2014 16:44:49",
        "pid": "4470",
        "connection_id": "postgres,db=mydb",
        "log_level": "LOG",
        "parse_ms": "4.550",
        "sql-statement": "statement: SELECT id FROM location WHERE name =
        'enigma->/var/log/messages' AND server_id = '1'"
}
```


Filter grok rozparsoval logový súbor podľa očakávaní, prebehol automatický zápis do databázy Elasticsearch, boli vytvorené nové záznamy typu postgresql. Pomocou pluginu head ich môžeme zobraziť v indexe logstash-2015.01.04. Nový index nebol vytvorený preto, lebo príslušný formát dátumu nezodpovedá klasickému údaju timestamp, ktorý máme nastavený v konfiguračnom súbore. To bolo urobené schválne, aby sme zistili, ako si s tým Logstash poradí. Tieto záznamy majú teda ako timestamp uvedený dátum parsovania, nie dátum uskutočnenia udalosti.

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

6.4 Testovací scenár č. 4

Konfigurácia: Logstash a Elasticsearch bežiace ako daemony, vstup zo súboru jboss.log, konfiguračný súbor main.conf, zápis do databázy Elasticsearch.

Vstupné údaje: Obsah súboru jboss.log. Obsah tu neuvádzame, pretože je veľmi rozsiahly.

Očakávaný výsledok: Štruktúrovaný výpis udalosti zo serveru JBoss v definovanom formáte a zápis tejto udalosti do databázy.

Získaný výsledok: Iba prvý riadok


```
{
         "message" => "~dc-devt 2013-12-06T17:43:04.234+0100 [0.0.0.0-http-10.32.92.147-8080-3]
          INFO  b.v.a.d.l.PreProcessLoggingInterceptor - \nService: GET
          http://10.32.92.147:8080/appContext/rest/service\nUserId:
          itsmeagain\nResponse types application/json\nQuery Parameters:
          \n\tlimit -> [10]\n\tsortColumn -> [number]\n\tstart -> [0]\nPath parameters: \n
          Reply type: class myapp.PagedList\nOutput document:\n{...contents snipped...}\nDuration: 0.078s",
        "@version" => "1",
      "@timestamp" => "2013-12-06T16:43:04.234Z",
            "type" => "jboss",
            "host" => "ubuntu-elastic",
```

```
            "tags" => [
        [0] "multiline"
    ],
     "application" => "dc-devt",
       "timestamp" => "2013-12-06T17:43:04.234+0100",
          "server" => "0.0.0.0",
          "thread" => "http-10.32.92.147-8080-3",
        "severity" => "INFO",
        "category" => "b.v.a.d.l.PreProcessLoggingInterceptor",
            "duur" => 0.078,
          "ldapid" => "itsmeagain",
    "http_command" => "GET",
        "endpoint" => "http://10.32.92.147:8080/appContext/rest/service",
            "port" => "8080"
}
```

Filter obsah súboru rozparsoval podľa očakávaní, podarilo sa mu nájsť podľa regulárnych výrazov aj kúsky udalosti, ktoré sme definovali a vytvoriť na ich základe nové polia, napr. duur. Záznam sa úspešne uložil aj do databázy Elasticsearch.

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

6.5 Testovací scenár č. 5

Konfigurácia: Elasticsearch aj Logstash bežia ako daemony (stav po spustení OS Ubuntu), konfiguračný súbor main.conf. Súbor apache_access.1 bol naplnený približne 1500 riadkami logov vo formáte Apache Common Format, ktoré boli nazbierané v priebehu 5 dní (od 27.12.2014 do 31.12.2014). Predposledný riadok súboru schválne obsahuje chybu, ktorá by mala spôsobiť zlyhanie parsovania tohto riadku (_grokparsefailure). Keď sa táto značka objaví pri nejakom riadku, ES by ho nemal zapísať do databázy. Testujeme rýchlosť zápisu a nezapisovanie chybných riadkov.

Vstupné údaje: Obsah súboru apache_access.1. Obsah tu neuvádzame, pretože je veľmi rozsiahly.

Očakávaný výsledok:Zápis udalostí do databázy, vynechanie chybného riadka.

Získaný výsledok:Zápis trval službe Logstash približne 1 minútu. Do databázy boli zapísané všetky záznamy. Chybný záznam bol vynechaný.

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

6.6 Testovací scenár č. 6

Konfigurácia: Elasticsearch beží ako daemon Logstash spustíme ručne príkazom so zapnutým príznakom –debug (daemona logstash predtým zastavíme), konfiguračný súbor main.conf. Súbor apache_access.3 bol naplnený približne 20500 riadkami logov vo formáte Apache Combined Log Format, ktoré boli nazbierané v priebehu 3 dní (od 2.1.2015 do 4.1.2015). Je to najväčší záťažový test systému.

```
sudo /opt/logstash/bin/logstash -f /etc/logstash/conf.d/main.conf --debug
```

Vstupné údaje: Obsah súboru apache_access.3. Obsah tu neuvádzame, pretože je veľmi rozsiahly.

Očakávaný výsledok:Zápis udalostí do databázy, požadované rozparsovanie a vytvorenie indexov (na každý deň jeden index). Vypísanie údajov na obrazovku, pretože je zapnutý príkaz –debug.

Získaný výsledok:Zápis trval službe Logstash približne 16 minút. Do databázy boli zapísané všetky záznamy. So zapnutým výpisom na obrazovku teda dostávame priemernú rýchlosť približne 1200 riadkov / minútu, t. j. 20 riadkov / sekundu Logové udalosti môžeme nájsť v indexoch pre dátumy 2.1.2015 až 4.1.2015.

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

6.7 Testovací scenár č. 7

Konfigurácia: Elasticsearch aj Logstash bežia ako daemony, konfiguračný súbor main.conf. Súbor apache_access.3 bol naplnený novými 20500 riadkami logov vo formáte Apache Combined Log Format, ktoré boli nazbierané v priebehu 3 dní (tie isté logy so zmeneným dátumom – tentokrát od 8.1.2015 do 10.1.2015). Je to najväčší záťažový test systému. Rozdiel je, že nič nebudeme vypisaovať do terminálu, všetko beží v reálnych prevádzkových podmienkach. Budeme porovnávať rýchlosť zápisu do databázy.

Vstupné údaje: Novo pridaný bsah súboru apache_access.3. Obsah tu neuvádzame, pretože je veľmi rozsiahly.

Očakávaný výsledok:Zápis udalostí do databázy, požadované rozparsovanie a vytvorenie indexov (na každý deň jeden index).

Získaný výsledok:Zápis trval službe Logstash približne 100 sekúnd. Do databázy boli zapísané všetky záznamy. Bez výpisu na obrazovku teda dostávame priemernú rýchlosť približne 12350 riadkov / minútu„ t. j. 205 riadkov / sekundu. Rozdiel v rýchlosti bez zapnutých I/O operácií je teda 10-násobný. Logové udalosti môžeme nájsť v indexoch pre dátumy 8.1.2015 až 10.1.2015.

Zhodnotenie testu: Test dopadol podľa očakávaní, považujeme ho za úspešný.

Ak chceme získať lepšiu predstavu o výkonnosti riešenia, najlepšie bude vykonať určité benchmarky systému a zmerať používané zdroje, aby sme odhalili slabé miesto. V pokoji používa systém Ubuntu v priemere 4 % zdrojov CPU a približne 1500 MB RAM z dostupných 4048 MB. Vykonali sme aj benchmarky z programu System Profiler and Benchmark, ktorý sme na OS Ubuntu nainštalovali. Ten ponúka základné informácie o systéme a benchmarky CPU. Vygenerovaný ýstup z tohto programu nájdeme na obraze OS Ubuntu v adresári /home. Je taktiež priložený na DVD v zložke benchmark. Najviac nás asi bude zaujímať posledná sekcia Benchmarks, kde prebehli testy CPU. Vidíme tam, ako dlho trvali jednotlivé testy na našom počítači, koľko na iných strojoch. Nižší výsledok je lepší (ide o počet sekúnd trvania banchmarku). Počas oboch posledných testov daemon logstash zamestnal procesor na 100 %. Počas testu č. 6 bol limitujúcim faktorom výpis na obrazovku. Počas testu č. 7 bola limitujúcim faktorom rýchlosť procesora. Využívanie pamäti RAM stúplo iba asi o 50 MB. Teoreticky by sme väčšiu rýchlosť zápisu do databázy mohli dosiahnuť aj výmenou disku, musel by to však najprv zvládnuť procesor.

Kapitola 7

Záver

Implementované riešenie predstavuje komplexný nástroj na spracovanie logových súborov z technológií Apache, JBoss a PostgreSQL. Logové súbory vieme pomocou nástroja Logstash rozparsovať, vybrať si ktoré údaje uložíme a ktoré sú pre nás nepodstatné. Všetko to prebieha v jednotnom internom formáte databázy Elasticsearch, v rámci ktorého si môžeme vytvárať nové polia, značky a kategorizovať údaje podľa typu. Následne môžeme uložené údaje analyzovať a prehliadať pomocou grafického rozhrania Kibana či plugine elasticsearch-head, rýchlo v nich vyhľadávať a vytvárať rôzne štatistiky. Keďže nad celým riešením beží REST API, môžeme pomocou neho zadávať príkazy na správu aj z terminálu. Riešenie nie je platformovo viazané, keďže celý obraz operačného systému Ubuntu, kde je riešenie vytvorené, vieme virtualizovať na akomkoľvek počítači. Celé toto riešenie sme vytvorili na stolnom počítači s operačným systémom Windows 8. Vďaka rýchlej výmene konfiguračných súborov nie je problém mať vytvorených viacero riešení presne podľa toho, ako to v danom momente potrebujeme. To je hlavným prínosom v prípade riešenia problému zadávateľa. Zároveň nám tieto vytvorené konfiguračné súbory umožňujú nepoužívať celý obraz OS Ubuntu, ale pár jednoduchými príkazmi nainštalovať nástroje ELK kdekoľvek inde. Konfiguračné súbory budú plne funkčné aj v tomto prípade. V neposlednom rade môžeme riešenie ľahko rozšíriť o ďalšie typy logových súborov.

7.1 Možnosti ďalšieho vývoja

Keďže celé riešenie bolo navrhnuté ako modulárne, ako prvé nám napadne rozšírenie o nové typy logov. Môžeme si však vytvoriť aj rôzne konfigurácie pre jeden typ logov, podľa toho, aký typ informácii z nich potrebujeme získať. Riešenie ELK ponúka obrovské množstvo možností. Štandardný vstup a súbor zďaleka nie sú jedinými vstupmi, systém môžeme nakonfigurovať napríklad na spracovanie syslog správ. Rovnako je to s možnosťami výstupu. Spracované informácie môžeme posielať dokonca priamo ďalším aplikáciám. Vďaka radu kodekov a filtrov nie je problém vytvoriť si vlastný formát logov pre rôzne príležitosti. Pomocou nainštalovania plugine logstash-forwarder by sme systém mohli jednoducho nastaviť na prijímanie logov z viacerých serverov. Logstash by taktiež mohol naslúchať na TCP portoch. Možností je naozaj veľa. Myslíme si, že navrhnuté riešenie je už teraz komplexným nástrojom, dalo by sa však povedať, že je stále na začiatku. Jeho potenciál môžeme ďalej rozvíjať a vytvoriť z neho naozaj mocný nástroj.

Literatúra

[1] B. Jansen. *Handbook of Reasearch on Web Log Analysis.* IGI Global, 2008.

[2] N. Meghanathan. *Advanced Computing.* Springer Berlin Heidelberg, Heidelberger Platz 3, Berlín, Nemecko, 2011.

[3] Apache http server version 2.4 – log files.
Dostupné z `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html/`, stav z 2. 4. 2014.

[4] Apache module mod_log_config.
Dostupné z `http://httpd.apache.org/docs/2.4/mod/mod_log_config.html#formats/`, stav z 2. 4. 2014.

[5] Logstash bug.
Dostupné z `https://logstash.jira.com/browse/LOGSTASH-1875`, stav z 31. 12. 2014.

[6] C++ documentation – printf function.
Dostupné z `http://www.cplusplus.com/reference/cstdio/printf/`, stav z 20. 3. 2014.

[7] Centralized logging architecture.
Dostupné z `http://jasonwilder.com/blog/2013/07/16/centralized-logging-architecture/`, stav z 8. 12. 2014.

[8] Centralized logging using rsyslog.
Dostupné z `http://urbanairship.com/blog/2010/10/05/centralized-logging-using-rsyslog/`, stav z 10. 12. 2014.

[9] Chukwa.
Dostupné z `https://chukwa.apache.org/`, stav z 22. 12. 2014.

[10] What is elasticsearch?
Dostupné z `http://www.elasticsearch.org/overview/elasticsearch`, stav z 29. 4. 2014.

[11] Elasticsearch guide.
Dostupné z `http://www.elasticsearch.org/guide/`, stav z 29. 4. 2014.

[12] Fluentd.
Dostupné z `http://www.fluentd.org/`, stav z 22. 12. 2014.

[13] Apache flume.
Dostupné z http://flume.apache.org/, stav z 22. 12. 2014.

[14] Logstash forwarder.
Dostupné z https://www.digitalocean.com/community/tutorials/how-to-use-logstash-and-kibana-t
stav z 29. 12. 2014.

[15] Grok debugger.
Dostupné z http://grokdebug.herokuapp.com/, stav z 22. 12. 2014.

[16] Heka.
Dostupné z http://hekad.readthedocs.org/en/v0.8.1/, stav z 22. 12. 2014.

[17] Http 1.1 – status code definitions.
Dostupné z http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html/, stav z 20. 4. 2014.

[18] Jboss 7 access log configuration.
Dostupné z http://chandank.com/application-server/tomcat/jboss-7-access-log-configuration,
stav z 17. 3. 2014.

[19] Jboss documentation – logging conventions.
Dostupné z http://docs.jboss.org/process-guide/en/html/logging.html, stav z 16. 3. 2014.

[20] Jboss documentation – logging configuration.
Dostupné z https://docs.jboss.org/author/display/AS72/Logging+Configuration/,
stav z 16. 3. 2014.

[21] Kafka.
Dostupné z http://kafka.apache.org/, stav z 22. 12. 2014.

[22] Computer data logging.
Dostupné z http://en.wikipedia.org/wiki/Computer_data_logging/, stav z 20. 3. 2014.

[23] Príručka systémového administrátora – logovanie.
Dostupné z http://deja-vix.sk/sysadmin/log.html/, stav z 12. 3. 2014.

[24] Logstash guide.
Dostupné z http://logstash.net/docs/1.4.2/tutorials/getting-started-with-logstash,
stav z 29. 4. 2014.

[25] Manage events and logs?
Dostupné z http://www.elasticsearch.org/overview/logstash, stav z 28. 4. 2014.

[26] Logstash – dokumentácia.
Dostupné z http://logstash.net/docs/1.4.2/, stav z 22. 12. 2014.

[27] Logstash – vzory.
Dostupné z https://github.com/elasticsearch/logstash/tree/v1.4.2/patterns/,
stav z 22. 12. 2014.

[28] Logstash.
Dostupné z `http://logstash.net/`, stav z 22. 12. 2014.

[29] Apache lucene – query parser syntax.
Dostupné z `http://lucene.apache.org/core/3_5_0/queryparsersyntax.html`, stav z 2. 5. 2014.

[30] Nsq.
Dostupné z `http://nsq.io/`, stav z 22. 12. 2014.

[31] Postgresql documentation – error reporting and logging.
Dostupné z `http://www.postgresql.org/docs/9.3/static/runtime-config-logging.html`, stav z 26. 4. 2014.

[32] Postgresql logging.
Dostupné z `http://wiki.postgresql.org/images/9/9d/Logging_pgopen_withnotes.pdf`, stav z 6. 3. 2014.

[33] Scribe.
Dostupné z `http://www.scribesoft.com/`, stav z 22. 12. 2014.

[34] Logy ze serveru.
Dostupné z `http://www.jakpsatweb.cz/seo/logy.html`, stav z 3. 4. 2014.

[35] Server log.
Dostupné z `http://en.wikipedia.org/wiki/Server_log/`, stav z 20. 3. 2014.

[36] strftime – convert date and time to string.
Dostupné z `http://pubs.opengroup.org/onlinepubs/009695399/functions/strftime.html`, stav z 7. 4. 2014.

Dodatok A

Inštalačná a používateľská príručka

A.1 S použitím priloženého obrazu OS Ubuntu

Na virtualizáciu obrazu použijeme virtualizačný softvér, napríklad Oracle VM VirtualBox. Po nahraní a spustení súboru použijeme vytvorený používateľský profil user, s heslom \. Rovnaké heslo použijeme v prípade potreby aj pre superpoužívateľský účet sudo. Po nabehnutí operačného systému sa automaticky spustia nástroje Logstash a Elasticsearch. Logstash sa spustí s konfiguračným súborom main.conf a sleduje príslušné logové súbory. Kibanu a plugin elasticsearch-head spustíme vo webovom prehliadači na adresách http://localhost:80 a http://localhost:9200/_plugin/head/.

Súbor main.conf môžeme nahradiť ľubovoľným konfiguračným súborom podľa potreby.

A.2 Bez použitia priloženého obrazu OS Ubuntu

V tejto sekcii si stručne povieme, ako dospieť do stavu, aký je momentálne na obraze operačného systému. Je to pre prípad, že by sme chceli riešenie používať bez obrazu operačného systému pomocou virtualizácie. Predpokladáme inštaláciu na operačný systém unixového typu (tak, ako to bolo uvedené v požiadavkách). Jedinou prerekvizitou pred inštaláciou jednotlivých nástrojov je Java. Celý postup je totožný s návodom na inštaláciu uvedenom na strane 45.

Dodatok B

Zoznam použitých skratiek

PEPA Performance Evaluation Process Algebra,

JSON JavaScript Object Notation,

IRC Instant Relay Chat,

IM Instant Messaging,

MMORPG Massive Multiplayer Online Role Playing Game,

VoIP Voice over IP,

HTTP Hypertext Transfer Protocol,

URI Uniform Resource Identifier,

URL Uniform Resource Locator,

CGI Common Gateway Interface,

ID Identifier,

GMT Greenwich Mean Time,

IIS Internet Information Services,

SEO Search Engine Optimization,

JRE Java Runtime Environment,

JDK Java Development Kit,

HDFS Hadoop Distributed File System,

FQDN Fully Qualified Domain Name,

OS Operating System.

Dodatok C

Obsah priloženého DVD

- **source** — adresár obsahuje obraz OS Ubuntu s názvom Ubuntu wit ELK vo formáte .ova
- **text** — adresár obsahuje text diplomovej práce vo formáte PDF
- **config** – adresár obsahuje konfiguračné súbory používané na rozbehnutie riešenia (main.conf) a testovanie (test2.conf)
- **log** – adresár obsahuje pár príkladov logových súborov (apache_access.1, apache_access.2, apache_access.3, postgresql.log, jboss.log)
- **benchmark** – adresár obsahuje vygenerovaný report z programu System Profiler a Benchmark vo formáte HTML
- **readme.txt** – súbor obsahuje text s obsahom adresárov a ďalšie dôležité informácie, ako sú inštalačná a používateľská príručka