Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Podpora dynamické závažnosti v systému Logstash

Radim Dostál

Vedoucí práce: Ing. Tomáš Herout

11. května 2015

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu, kterým je pan Ing. Tomáš Herout, za trpělivost a konstruktivní návrhy při realizaci. Také bych rád poděkoval všem, kteří mě podporovali v průběhu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Dostál, Radim. *Podpora dynamické závažnosti v systému Logstash.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce řeší problém, kterým je dohled logů a určení jejich závažnosti na základě parametrů, které jsou dynamicky vypočítány a nastaveny od administrátora. Také detekuje periodicitu logů a rozhoduje o poslání zprávy správci. Práce realizuje rozšíření pro systém Logstash, který je silným nástrojem pro řízené zpracování logů.

Klíčová slova logstash, dohled, ruby, rails, log, elasticsearch, postgres

Abstract

This thesis solves problem of monitoring logs and determine their severity based on parameters that are dynamically calculated and set by the administrator. It also detects periodicity of logs and decide about sending a notification to administrator. The thesis realizes extension of the system Logstash which is powerful tool for controlled processing of logs.

Keywords logstash, monitoring, ruby, rails, log, elasticsearch, postgres

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jak je známo, tak spolehlivost síťových či softwarových služeb je v dnešní době finančně náročnější než provozování služeb samotných. Proto je pro administrátory velmi důležité provozovat nějaký typ dohledového systému, který zařídí aktivní dohled zařízení a služeb, a po zjištění problému ho oznámí administrátorovi. Většina dohledových systémů poskytuje jednoduché rozhraní, které je schopno kontrolovat službu aktivně tzn. každých pár minut zjistit, jestli je služba dostupná. Bohužel podpora na straně pasivního dohledu, a to konkrétně logů služeb či zařízení často značně kolísá viz kapitola Analýza. Existuje mnoho systémů, které provádí analýzu příchozích logů, ale většinou poskytují administrátorovi jen jednodušší a graficky přívětivější pohled na logy a žádné nebo velmi omezené možnosti detekce důležitých zpráv.

Cíl práce

V této práci se budu věnovat právě možnostem dohledu logů, které bývají prvním místem, na které se administrátor dívá při zjištění problému, bohužel nelze informovat aministrátora o každém logu, protože i v relativně malém prostředí jich může být vygenerováno až desítky tisíc během jednoho dne. Typy logů, které jsou pro administrátora důležité, se také často mění vzhledem ke konkrétnímu prostředí a konkrétnímu administrátorovi.

Tato práce řeší problém pomocí rozšíření pro systém Logstash, který je navrhnutý pro analýzu a pro perzistentní uložení logů. Hlavním účelem této práce je dynamické určení závažnosti určitého logu, která je závislá na konfiguraci ze strany administrátora, na četnosti příchodu a na závažnosti určené zdrojem logu. Další vlastností, kterou tato práce bude podporovat, je jednoduché webové rozhraní, které bude administrátorovi umožňovat konfiguraci závažnosti logů, závažnost hostů, kteří zasílají logy, a dále umožní zobrazení několika posledních přijatých logů. U výpisu posledních přijatých logů bude možné prostřednictvím známé části logu přejít k souvisejícím logům.

Související jsou logy, které obsahují stejnou MAC adresu, IP adresu nebo se týkají stejného rozhraní.

Kapitola 1

Analýza

V této kapitole popíši výsledky své analýzy existujících řešení pro dohled logů a v krátkosti napíši jejich nedostatky.

1.1 Existující řešení

1.1.1 Zabbix

Zabbix je jeden z nejlepších dohledových systému s otevřeným zdrojovým kódem[1], který řeší aktivní dohled služeb prostřednictvím kontroly otevřeného portu na zařízení nebo pomocí tzv. zabbix agenta, který je nainstalovaný v systému a umožňuje tak zpřístupnění veškerých dat pro dohled. Právě díky zabbix agentovi je možné zpřístupnit soubory, které jsou určeny pro logování, a kontrolovat nové řádky regulárním výrazem. Bohužel takto jednoduchý způsob kontroly logů může být velmi nevýhodný, protože nelze ovlivnit kolikrát je daná zpráva oznámena administrátorovi, může se tak lehce stát, že nějaké zařízení zaloguje určitou událost mnohokrát během pár minut a zabbix administrátorovi pošle informaci o každé z nich. Existují případy, kdy je takové chování žádané, ale těch není mnoho a žádné komplexní řešení zabbix bohužel nenabízí[2].

1.1.2 Graylog

Graylog je systém, který se zabývá dohledem a zpracováním logů. Tento systém nahlíží na příchozí logy jako na proudy, které si definuje administrátor. Administrátor má možnost dohledu logů v rámci jednoho proudu, který může být složen například jen z logů od jednoho zdroje. Lze nastavit trigger akci, která při neobvyklém počtu logů informuje administrátora. Rozšiřitelný je prakticky stejně jako systém Logstash, ale množství jeho rozšíření není tak velké jako v případě Logstashe, to je jeden z hlavních důvodů, proč se v téhle situaci nehodí[3].

1.1.3 log.io

Systém log.io poskytuje jiný pohled na dohledování logů. Pomocí tohoto nástroje lze příchozí logy nebo výběr z nich zobrazovat v reálném čase v prohlížeči. Tento systém však neposkytuje možnost perzistentního ukládání dat[4].

1.1.4 Proč právě Logstash

Protože je v případě problému potřeba informovat administrátora, jsou výstupní rozšíření systému Logstash velmi dobrými možnostmi. Logstash obsahuje výstupní rozšíření, která poskytují odeslání zprávy například prostřednictvím emailu, xmpp nebo lze odeslat zprávu do dohledovacího systému zabbix a prostřednictvím něj řídit doručení informace.

Dále kvalitní dokumentace a aktivní komunita přijde vhod při vývoji rozšíření. Dostupnost systému Logstash je velmi dobrá, existují balíčky pro většinu linuxových distribucí, které lze stáhnout z příslušných zdrojů. Další možností, jak získat systém Logstash, je jeho zkompilování ze zdrojového kódu.

1.2 Analýza vstupních dat

Vstupními daty jsou již zmíněné logy. Termín log je ve většině případů chápán jako záznam v souboru určeném pro logy. Log je většinou krátká zpráva, která nese informaci o stavu nějaké služby, systému nebo dokonce hardwaru, na kterém služba běží. Pravděpodobně nejvíce rozšířeným logovacím standardem je standard syslog, který byl definán IETF v RFC 5424. Bohužel velké množství vývojářů implementuje tento standard s chybami nebo s nějakými úpravami, které jsou pro učely analýzy méně výhodné.

Standardem definovaný formát logu[5]:
TIMESTAMP HOSTNAME TAGS MESSAGE

TIMESTAMP
: Formát časové značky je variabilní a určuje ho RFC 3339.

HOSTNAME
: Pole hostname určuje zdroj logu, který může být určen IP adresou, pomocí FQDN, ale existují i situace, kdy je tato hodnota NULL.

TAGS
: Pole tags není povinné, ale také může obsahovat několik polí, které obsahují libovolná data.

MESSAGE
: Zpráva v libovolném formátu.

Tyto logy je v první řadě nutné rozparsovat a pochopit, co znamenají jejich jednotlivé části. Parsování a strojové chápání řetězce je složitá disciplína v oblasti IT.

Kapitola 2

Použité prostředky

2.1 Logstash

Logstash je systém pro analýzu logů a má otevřený zdrojový kód. Tento systém se v poslední době velmi rozšiřuje a dá se říci, že jeho velkou výhodou je vysoká modularita a jednoduchá konfigurace. V současné době pro něj existuje přes 100 rozšíření, které ho obohacují o velké množství funkcí. Jeho rozšíření se řadí do několika různých skupin, které popíši později, toto třídění vychází z jeho architektury. Architektura systému Logstash je navržena jako řada doplňků či skriptů, které jsou propojeny rourou, kterou známe ze systémů založených na Unixu. Logstash však tuto rouru obohatil o několik významných funkcí, jako jsou například podmínky, dle kterých lze řídit následující zpracování určitého logu. Také v nich umožňuje jednoduché změny. Log je ve chvíli zpracování prezentován jako strukturovaný text ve formátu JSON[6].

U každé skupiny uvedu jen hlavní představitele, ostatní lze dohledat v dokumentaci projektu[7]:

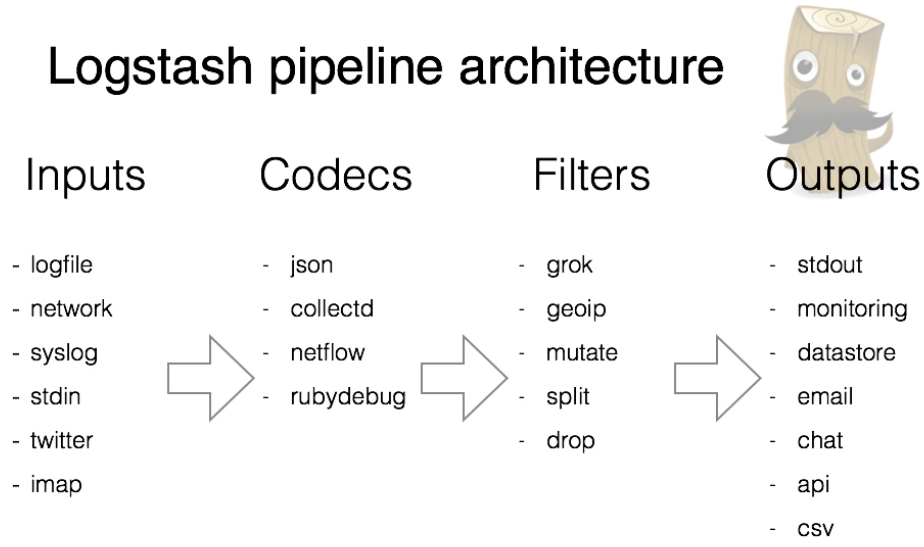
INPUT
: Doplňky, které umožňují přijímat logy do systému, mezi hlavní představitele patří doplněk syslog, UDP a například stdin.

CODEC
: Rozšíření z této skupiny umožňují dekódovat přijaté logy, hlavní představitel této skupiny je rozšíření netflow.

FILTER
: Jednou z nejpočetnějších skupin je jednoznačně skupina filtrů, která umožňuje nad logy provádět různé výpočty a změny, lze logy šifrovat, měnit a upravovat. Hlavními představiteli jsou grok, mutate, split a další.

OUTPUT
: Poslední skupinou jsou výstupní rozšíření, které umožňují logy ukládat

do nejrůznějších cílů nebo jen odeslat informace o konkrétním logu na konkrétní místo. Hlavními představiteli jsou rozšíření elasticsearch, zabbix a stdout.



Obrázek 2.1: Logstash pipeline architecture

Konfigurace systému Logstash se skládá jen z volání rozšíření, které poskytují veškerou funkcionalitu.

Příklad konfigurace:

```
input {
  stdin {}
  syslog {}
}
filter { grok { ...truncate... } }
output {
  if ["dynamic_priority"] > "6" {
    zabbix {
      host => "monitoring.example.com"
      zabbix_sender => "/usr/local/bin/zabbix_sender"
    }
  } else {
    file { path => "/var/log/logstash/unsent.log" }
  }
}
```

2.1.1 Kibana

Jen zkráceně zde zmíním i projekt Kibana, který je navržen jako webové rozhraní umožňující zobrazovat data z databáze Elasticsearch a je vývojáři systému Logstash doporučovaný. Bohužel není schopen dělat nad daty změny[7].

2.2 Programovací jazyky a frameworky

2.2.1 Ruby

Systém Logstash je naprogamovaný v programovacím jazyce Ruby, ve kterém je třeba programovat i jeho rozšíření. Proto ani mé rozšíření není vyjímkou a je napsáno v tomto jazyce.

Programovací jazyk Ruby je interpretovaný skriptovací jazyk, který vznikl v roce 1995 v Japonsku rukou programátora Yukihiro Matsumoto. Ruby je dnes velmi vyhledávaný pro svou jednoduchou syntaxi a výkonné zpracování. Dalším důležitým aspektem pro jeho rozšíření byla jeho platformová nezávislost, takže jeho programy lze relativně bez obtíží spustit na platformě Linux, Windows, ale i Mac OS X. Jeho popularita značně vzrostla ve chvíli, kdy vznikl framework Rails využívajícího pro svůj běh[8].

2.2.2 Ruby on Rails

Protože rozšíření, které je vyvíjeno v rámci této práce, musí mít možnost konfigurace prostřednictvím webového rozhraní, rozhodl jsem se využít tuto platformu, která se zdá být velmi efektivní co se vývoje týče.

Framework Ruby on Rails je přímo určený pro vyvíjení webových aplikací, které jsou připojeny k databázi. Jeho první verze pro veřejnost spatřila světlo světa v roce 2005 a od té doby se těší velké oblibě a kvalitní podpoře ze strany dokumentace a komunity. Další vlastností je využití architektury Model-View-Controller, která odděluje jednotlivé části aplikace, konkrétně části, které spolupracují s databází od těch, které zobrazují data uživateli. Tato architektura je velmi vhodná pro aplikace, které předpokládají budoucí vývoj. Rails se řídí několika základními pravidly, které se snaží ušetřit programátorovi psaní rutinních částí kódu. Tyto části lze jednoduše generovat pomocí přibalených rails skriptů[9].

2.3 Databáze

2.3.1 Elasticsearch

Pro perzistentní uložení logů v systému Logstash se většinou využívá databáze Elasticsearch.

Databáze Elasticsearch vychází z plně textového vyhledávače Apache Lucene[10], který vyvíjí společnost Apache Software Foundation. Databáze

Elasticsearch je naprogramována v jazyce Java a komunikace s ní probíhá pomocí RESTful API, prostřednictvím kterého se odesílají tzv. dokumenty, které plní roli příkazů a lze pomocí nich udělat prakticky jakoukoliv akci. Hlavním rysem Elasticsearch je rychlost a spolehlivost, kterou bezesporu poskytuje[11].

2.3.2 Postgres

Postgres je relační databáze, kterou toto rozšíření využívá pro perzistentní uložení dodatečných dat.

Postgres je jeden z nejrozšířenějších databázových systémů dnešní doby. Je vyvíjený od roku 1982. Primárně je vyvíjený pro platformu systémů založených na Unixu, ale lze ho provozovat i na platformě Windows. V současnosti je vyvíjen společností PostgreSQL Global Development Group. Z velké části je plně kompatibilní se standardy jazyka SQL, ale v mnoha částech ho záměrně porušuje, aby nabídl rozšířené funkce a plnohodnotně využil jeho objektovou část[12].

2.3.3 Plně textové vyhledávání

Plně textové vyhledávání je termín, pod kterým rozumíme vyhledávání pomocí části textového řetězce. Je to technika, při které jsou textové dokumenty ohodnoceny tzv. skóre, které je vysoké dle velikosti shody předaného vzoru. Elasticsearch i Postgres tuto techniku podporují, ale každý jiným způsobem. Nejdůležitějším rozdílem je jednotka skóre, kterým se pozná relevance daného objektu k předanému vzoru. Elasticsearch udává skóre bez jednotky, je to reálné číslo od nuly z hora neomezené, což může vést ke značným problémům, když je potřeba znát koeficient, jak moc si jsou dané řetězce či dokumenty podobné. V mém případě to bylo velmi důležité, protože některé části logu se mohly měnit, takže bylo důležité určit přesnou hranici podobnosti. Databáze Postgres určuje podobnost procentuélně, což bylo pro tyto účely výhodnější, a proto jsem využil plně textové vyhledávání v databázi Postges.

2.4 Knihovny

Pro zjednodušení vývoje jsem použil několik existujících knihoven, kterým se v terminilogii jazyku Ruby říká Gemy a stahují se pomocí nástroje gem.

2.4.1 Podpora Postgres pro Ruby

Pro komunikaci s databází Postgres existují dva hlavní Gemy, jeden je velmi komplexním řešením, které využívá pro přímou komunikaci s databází externí binární program, jmenuje se jednoduše pg. Systém Logstash má svůj vlastní upravený interpret jazyka Ruby, který nepodporuje knihovny s externími pro-

gramy. Proto jsem využil gem postgres-pr, který komunikuje s databází přímo a lze jej tedy využít v rozšíření programu Logstash.

2.4.2 Elasticsearch

Databáze Elasticsearch je hojně využívána v jazyku Ruby, takže opatřit si gem právě pro komunikaci s touto databází nebyl problém. Je to gem, který se jmenuje elasticsearch a komunikuje pomocí vnitřního nástroje curl s databází pomocí RESTful API. Protože některá rozšíření pro Logstash s databází Elasticsearch už komunikují, není tak třeba gem složitě instalovat pomocí Ruby Logstash interpreta.

2.4.3 Gem text

Gem text pro jazyk Ruby obsahuje několik algoritmů pro práci s texty. Jedním z těchto algoritmů je implementace Levenshteinovi vzdálenosti, kterou při porovnání vzorů se zprávami používám.

2.5 Levenshteinova vzdálenost řetězců

Levenshteinova vzdálenost je řetězcová metrika, která určuje míru rozdílnosti mezi řetězci. Je definována jako minimální počet operací vkládání, mazání nebo nahrazení, které jsou potřeba vykonat, aby byly řetězce totožné.[13]

2.5.1 Definice[13]

Matematická definice Levenshteinovi vzdálenosti mezi řetězci a a b je dána funkcí $lev_{a,b}(|a|, |b|)$ kde

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if min(i,j)} = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{jinak} \end{cases}$$

kde $1_{(a_i \neq b_j)}$ je funkce rovna nule pro $a_i = b_j$ a jinak rovna jedné.

První možnost pro minimum v tomto vzorci říká, že je třeba smazat znak, druhá možnost určuje vložení nového znaku a třetí možnost nerovnost znaků.

Levenshteinova vzdálenost má několik hraničních hodnot:

1. Maximální vzdálenost dvou řetězců je rovna délce delšího řetězce.

2. Vzdálenost je rovna nule jen v případě, že jsou oba řetězce totožné.

3. Pokud mají řetězce stejnou délku, je jejich maximální Levenshteinova vzdálenost rovna Hammingově vzdálenosti.

Protože známe horní hranici vzdálenosti mezi dvěma slovy, můžeme vyjádřit jejich rozdíl procentuélně.

2.5.2 Příklad

Ukáži zde výpočet Levenshteinovi vzdálenosti mezi slovy *abcdef* a *azced*.

Tabulka 2.1: Levenshteinova vzdálenost: Počáteční tabulka pro výpočet

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1						
z	2						
c	2						
e	3						
d	4						

Výpočet provedeme pomocí tabulky, která bude uchovávat veškeré mezivýpočty. Pro začátek potřebujeme připravenou tabulku 2.1, kterou budeme po řádcích postupně vyplňovat, až nám v pravém dolním rohu vyjde výsledná vzdálenost.

Tabulka 2.2: Levenshteinova vzdálenost: První část výpočtu

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0					
z	2						
c	2						
e	3						
d	4						

Pole na souřadnicích (a, a) vyplníme hodnotou z modrého pole, protože písmena jsou si rovna.

Tabulka 2.3: Levenshteinova vzdálenost: Druhá část výpočtu

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0	1				
z	2						
c	2						
e	3						
d	4						

Pole na souřadnicích (b, a) vyplníme nejmenší hodnotu z modrých polí, kterou o jedničku zvýšíme.

Tabulka 2.4: Levenshteinova vzdálenost: Výsledek

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
z	2	1	1	2	3	4	5
c	2	2	2	1	2	3	4
e	3	3	3	2	2	2	3
d	4	4	4	3	2	3	3

V pravém dolním rohu je výsledek, který určuje počet operací, které je třeba provést se slovy, aby se shodovala. Pomocí zpětného průchodu tabulky od výsledku lze jednoduše zjistit typy těchto funkcí. V tomto příkladě to znamená nahrazení písmena z ve slově v řádku písmenem b, vymazání písmena d ze slova v řádku a změna písmena f za písmeno d ve slově v řádku.

2.6 Testovací a vývojové prostředí

Veškerý vývoj a veškeré testování probíhalo v systému GNU/Linux v distribuci Debian 7.6 „wheezy“ (verze jádra 3.2.0).

Kapitola 3

Implementace

V následující části popíši, co vše by mělo toto rozšíření řešit a poskytovat. Hlavním smyslem je umění odfiltrovat zbytečné logy od těch důležitých i za cenu toho, že malé procento odflitrovaných logů budou logy důležité. Pro tuto schopnost jsem navrhl mechanismy, které dynamicky určují jejich důležitost. Pro porozumění některým jejich částem je bylo nutné rozparsovat. Pro systém Logstash existuje rozšíření, které pro to využívá otevřenou knihovnu Grok. Grok pro parsování textů využívá řadu vzorů, které jsou zapsány regulárními výrazy a samotné rozšíření pro Logstash jich při instalaci obsahuje několik desítek a ty jsou z velké části dostačující. Proto jsem se rozhodl využít rozšíření Grok i pro mé účely. Pomocí něho je log rozparsován a dále pokračuje v řetězu funkcí.

3.1 Implementace rozšíření

Rozšíření je implementováno v jazyce Ruby, stejně jako další. Toto rozšíření je specifické hlavně tím, že pro svůj běh využívá externí databázi Postgres. Rozšíření pro systém Logstash vyžadují implementaci dvou veřejných funkcí. Hlavičky těchto funkcí jsou předepsány vývojáři Logstash. První z funkcí zařizuje inicializaci běhu, jako je připojení do potřebné databáze nebo otevření potřebných soketů. Druhá a mnohem důležitější funkce, která musí být implementována, je funkce filter. Ta přebírá parametrem aktuálně zpracovávaný log. Každé rozšíření systému Logstash dědí z nadřazené třídy základní funkce. Některé z těchto funkcí umožňují základní manipulaci s logem. V metodě filter toto rozšíření zpracuje data z logu a vyhodnotí je s daty v databázi. Vyhodnocení rozepíši do několika podkapitol.

3.1.1 Příprava, inicializace a konfigurace rozšíření

Konfigurace rozšíření je zprostředkována přes konfigurační soubor systému Logstash. V následujícím seznamu popíši konfigurační možnosti rozšíření.

db_name
Jméno databáze.

db_host
IP adresa nebo hostname serveru s databází.

db_user
Uživatel pro přístup do databáze.

db_passwd
Heslo uživatele pro připojení do databáze.

db_port
Port, na kterém naslouchá databáze.

min_score
Tento parametr určuje, na kolik procent se log musí podobat svému vzoru. Tato hodnota je z intervalu 0–1. Výchozí hodnota je 0.9

residue_field
Parametr, který určuje název pole v objektu JSON. V tomto poli bude rozšíření hledat část logu pro porovnání se vzorem. Ta by měla být zbavena času, data a čísla sekvence, pokud ho obsahuje. Výchozí hodnota je residue.

host_ip_field
Parametr s názvem pole, které obsahuje ip adresu nebo hodnotu, která určuje zdroj logu.

field_with_timestamp
Parametr s názvem pole, které obsahuje časovou značku logu, tato hodnota je dále zpracována funkcí *strftime*, která přijímá timestamp jen ve formátech definovaných pomocí RFC 3339[14].

field_syslog_severity
Parametr s názvem pole, které obsahuje číselnou závažnost od zdroje (syslog severity). Výchozí hodnota je *severity*.

period_margin
Tento parametr určuje toleranci při určení periody. Ve výchozím stavu je jeho hodnota 1.

k1
Koeficient pro výpočet celkové závažnosti. Parametr k1 se týká závažnosti zdroje zprávy. Ve výchozím stavu je tato hodnota nastavena na 1.

k2
: Koeficient pro výpočet celkové závažnosti. Parametr k2 se týká závažnosti vzoru zprávy. Ve výchozím stavu je tato hodnota nastavena na 1.

k3
: Koeficient pro výpočet celkové závažnosti. Parametr k3 se týká určené závažnosti dle počtu příchodů. Ve výchozím stavu je tato hodnota nastavena na 8.

s1
: Koeficient, který určuje váhu celkové závažnosti spočtené ze závažnosti od hosta, vzoru a závažnosti dle počtu příchodů.

s2
: Koeficient, který určuje váhu závažnosti od zdroje.

debug
: Parametr, který povolí výpis ladících dat. Výchozí hodnota je false.

3.1.2 Určení vzoru a jeho závažnosti

Určení vzoru už se bez využití databáze neobejde, je třeba udělat 1 až 2 dotazy. Pro porovnání logu se vzorem v databázi se využije řetězec v poli, které nese název podle konfigurace direktivy *residue_field* nebo ve výchozím stavu název residue. Z tohoto řetězce se odstraní známé části, konkrétně IP adresa, MAC adresa a jméno síťového rozhraní, kterého se zpráva týka.

Po nahrazení se vykoná dotaz do tabulky patterns, který pomocí plně textového vyhledávání hledá maximálně shodný záznam. Maximálně shodný záznam se využije pro další porovnání pomocí Levenshteinovi vzdálenosti slov. Pokud existuje vzor, který je podobný alespoň z 80% logu, použije se jeho závažnost, která je konfigurovatelná administrátorem, pro další výpočty. Také se k samotnému logu vytvoří JSON atribut pattern, který nese jednoznačný identifikátor vzoru z tabulky patterns.

Pokud v tabulce patterns neexistuje vzor, který je s logem dostatečně podobný, tak je třeba vytvořit vzor nový, který je ohodnocen výchozí závažností 5.

3.1.3 Určení závažnosti hosta

Určení závažnosti hosta probíhá velmi podobně jako určení závažnosti logu dle vzoru. V první řadě se provede dotaz, který by měl vrátit závažnost již existujícího hosta. Pokud host v databázi neexistuje, je vytvořen s výchozí závažností 5. Host je identifikován IP adresou, která je ve výchozí konfiguraci hledána v JSON poli *ip*.

3.1.4 Určení periodicity

Existuje značné množství logů, které chodí v určité periodě, tedy že se opakují po konstantně určeném intervalu. Zejména síťová zařízení odesílají periodické logy, které informují o problémech, které musí být v co nejkratší době odstraněny. Toto rozšíření periodický log odhalí a neinformuje o problému při každém přijetí.

Periodicita logu se dá určit až po jeho třetím příchodu ve stejné periodě, proto i toto rozšíření, před určením periodicity, čeká na jeho třetí příchod. Pro určení periodicity je využívána tabulka vztahu mezi vzorem a zdrojem logu. Do této tabulky se průběžně ukládá čas příchodu logů, který se využívá pro počítání aktuální periody, a také je zde uložena hodnota předchozí periody. Po přijetí zprávy se tedy dotáži na předchozí hodnoty konkrétního typu zprávy od konkrétního hosta. A tyto hodnoty se použijí pro výpočet aktuální periody a pro srovnání s předchozí periodou. Protože může zapůsobit zpoždění sítě a zpoždění zpracování logu, může se perioda o několik sekund lišit, tak je možné pomocí parametru *period_margin* nastavit toleranci rozdílu period.

3.1.5 Určení dynamické závažnosti

Pro určení dynamické závažnosti slouží tabulka *times_of_messages*, která udržuje informace o času přijetí určitého typu zprávy z určitého zdroje. Díky této tabulce lze jednoduše zjistit, kolikrát už zpráva přišla za poslední dobu. Množství přijetí určité zprávy hraje zásadní roli pro určení dynamické závažnosti. Cílem je informovat administrátora o stavu, který se neděje často. Proto při přijetí logu, který nepřišel delší dobu, jeho závažnost roste nebo naopak pro log, který chodí příliš často, klesá.

Tabulka 3.1: Určení dynamické závažnosti

Závažnost	Počet příchodů logu
10	30 dní nepřišel
9	14 dní nepřišel
8	7 dní nepřišel
6	1 den nepřišel
5	méně než 4× za den
4	více než 4× za den
3	více než 6× za den
2	více než 12× za den
1	více než 20× za den

3.1.6 Určení celkové závažnosti

Celková závažnost logu se vypočítá pomocí závažnosti zdroje, závažnosti vzoru a závažnosti určené dle počtu přijatých zpráv tohoto typu v rámci jednoho zdroje. Celková závažnost je vypočítána pomocí vzorce váženého průměru, který má konfigurovatelné koeficienty tedy váhy konkrétních závažností. Ve výchozím nastavení jsou váhy nastaveny ve prospěch závažnosti určené dle počtu příchodů.

$$dynamic\_priority = \frac{k1*host\_priority+k2*pattern\_priority+k3*rating\_priority}{k1+k2+k3}$$

3.1.7 Určení celkové závažnosti, včetně závažnosti od zdroje

Pokud existuje pole se závažností od zdroje, tak se také zařadí do výpočtu celkové závažnosti. Protože závažnost od zdroje je v rozsahu 0–7 a závažnost v tomto rozšíření je 0–10, tak je třeba ji znormalizovat normalizačním koeficientem, který je vypočítaný podílem maximálních hodnot obou závažností. Následující vzorec ukazuje výpočet dynamické závažnosti s existencí závažností od zdroje.

$$dynamic\_priority = \frac{s1*dynamic\_priority+s2*norm\_coefficient*syslog\_severity}{s1+s2}$$

3.1.8 Rozhodnutí maximálního počtu odeslání konkrétní zprávy

Tato funkcionalita umožňuje administrátorovi určit maximální množství odeslaných zpráv konkrétního typu v určitém intervalu. Ve výchozím nastavení je možné odeslat maximálně 1 zprávu za 24 hodin, tyto hodnoty jsou konfigurovatelné prostřednictvím webového rozhraní.

Tento mechanismus využívá pro svůj účel tabulku vztahů mezi hosty a vzory, ve které je uložen interval, během kterého počet odeslaných zpráv ovlivňujeme, s časovou značkou začátku intervalu, počet momentálně odeslaných zpráv v intervalu a maximální počet odeslaných zpráv.

3.1.9 Problémy implementace

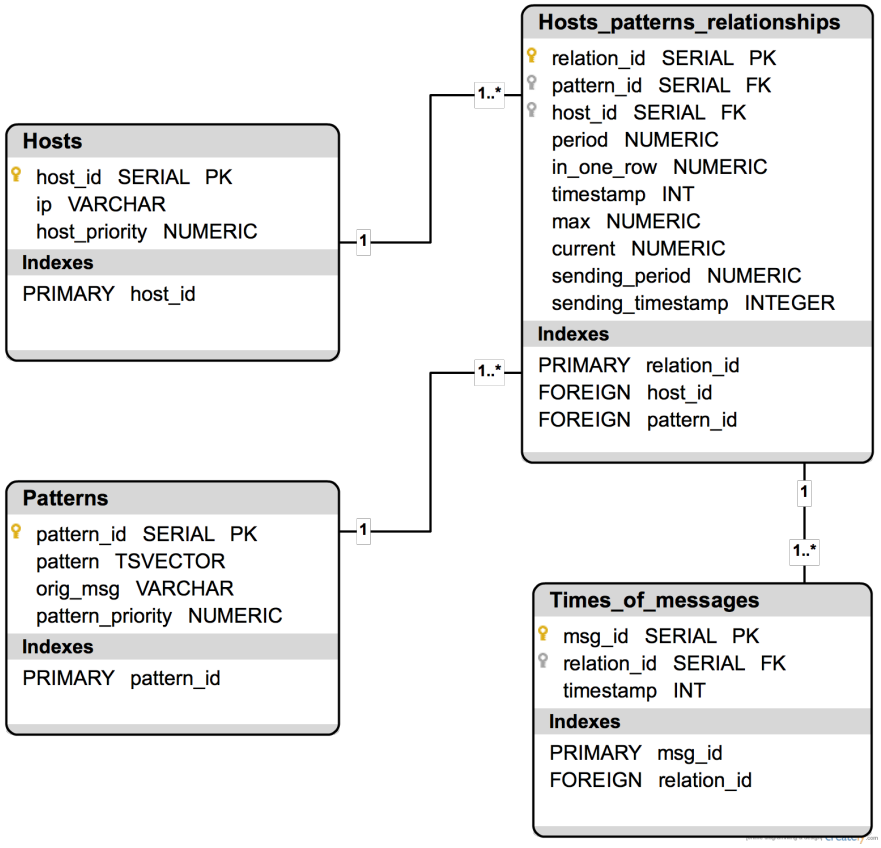
Při implementaci jsem narazil na několik problémů. Popíši zde jejich příčiny a řešení.

Prvním problémem byla instalace některých gemů do systému Logstash, který využívá pro běh rozšíření svůj vlastní zdroj gemů. Tato vlastnost je důsledek využití vlastního interpreta jazyka Ruby, který je naprogramován v jazyce Java. Jeho vývojáři však na tuto možnost mysleli a vytvořili parametry, pomocí kterých se knihovna stáhne a prolinkuje se složkou, která je využívána jako zdroj gemů.

Dalším zásadnějším problémem bylo rozhodnout, kterou databázi použít. V počáteční verzi jsem pro ukládání vzorů a pro plně textové vyhledávání mezi nimi využíval databázi Elasticsearch. To se posléze stalo hlavním problémem, protože databáze Elasticsearch hodnotí podobnosti pomocí skóre, které je reálným číslem zhora neomezeným, a proto nelze říci, kdy jsou si řetězce dostatečně podobné. Tato vlastnost je zcela úmyslná a vysvětlena právě vývojáři ve [15]. Po prostudování dokumentace databáze Postgres jsem se rozhodl využít pro plně textové vyhledávání právě ji.

3.1.10 Databáze

Následující obrázek popisuje strukturu databáze, kterou využívá rozšíření a webové rozhraní. Veškeré funkce jsou podporovány za pomoci 4 tabulek, které podrobněji popíši v následující části. Databázi lze vytvořit pomocí create skriptu, který se nachází na přiloženém CD.



Obrázek 3.1: Diagram databáze Postgres

Hosts
Tabulka Hosts udržuje IP adresy všech zdrojů logů.

Patterns
Patterns je tabulka, která udržuje vzory, se kterými jsou logy porovnávány. Za zmínku zde stojí hlavně sloupec pattern, který je datového typu TSVECTOR, tento typ se využívá pro plně textové vyhledávání.

Hosts_patterns_relationships
Tabulka vztahů vznikla při dekompozici vztahu m:n mezi tabulkami Hosts a Patterns. V této tabulce jsou data, která určují, zda je zpráva periodická, případně kolik zpráv s danou periodou přišlo. Další data určují, jestli už bylo odesláno maximální množství konkrétní zprávy od konkrétního hosta.

Times_of_messages
Tabulka, která pomáhá určit dynamickou závažnost logu. V této tabulce jsou záznamy o všech, které byly přijaty. Samotné logy zde nejsou, protože původní zadání tato data nevyžaduje.

3.2 Implementace webového rozhraní

Pro zjednodušení vývoje jsem využil vygenerované skripty od frameworku Rails, které sestaví celou strukturu aplikace. Rozšíření obsahuje jednoduché webové rozhraní, které umožňuje konfigurovat závažnosti vzorů, hostů a maximální hodnotu odeslaných zpráv v určitém intervalu. Na hlavní stránce jsou vypsány logy, které byly vyhodnoceny jako důležité a aktuálně chodící periodické logy. Další možností je zobrazení posledních přijatých logů a možnost nalezení jiných s nimi souvisejících.

3.2.1 Architektura aplikace

Aplikace je napsána v jazyce Ruby za pomocí frameworku Rails. Důležitou vlastností tohoto frameworku je architektura. Rails jsou postaveny na bázi architektury Model-View-Controller. Aplikace očekává budoucí vývoj, a proto je více než vhodné logické rozdělení některých částí[16].

3.2.1.1 Model

Část Model v architektuře MVC mimo jiné zařizuje komunikaci s databází a mapování databázových objektů na objekty v kódu. Také poskytuje uživatelská data vrstvě Controller.

3.2.1.2 View

View je vrstva, která je z velké části zodpovědná za data, která vidí uživatel. Ta komunikuje prostřednictvím vrstvy Controller s vrstvou Model.

3.2.1.3 Controller

Controller připravuje data pro zobrazení uživateli nebo-li pro vrstvu View, v této části se s daty provádí potřebné výpočty a jejich změny.

3.2.2 Možnost nastavení závažnosti vzorů a hostů

Nejdůležitější vlastností webového rozhraní je nastavení administrátorovy závažnosti k jednotlivým vzorům a hostům.

3.2.3 Možnost nastavení maxima pro odeslání zpráv v intervalu

Další mechanismus, který lze konfigurovat z webového rozhraní, je mechanismus počítání odeslaných zpráv v rámci určitého intervalu a v případném překročení maxima tyto zprávy potlačit. Administrátor určuje maximální počet odeslání určité zprávy v intervalu pro každého hosta a jeho konkrétní typ logu.

Kapitola 4

Testování s předanými daty

Testování probíhalo v několika různých variantách. Ohodnocení logů je přímo zavislé na konfiguraci daného spuštění. Data od vedoucího práce byla z prostředí síťového poskytovatele, takže se týkala převážně síťových prvků, v tomto případě přavážně od společnosti Cisco Systems, a skládala se z několika souborů, které měly dohromady přes 500 000 řádků a byla nasbíraná zhruba za 14 dní.

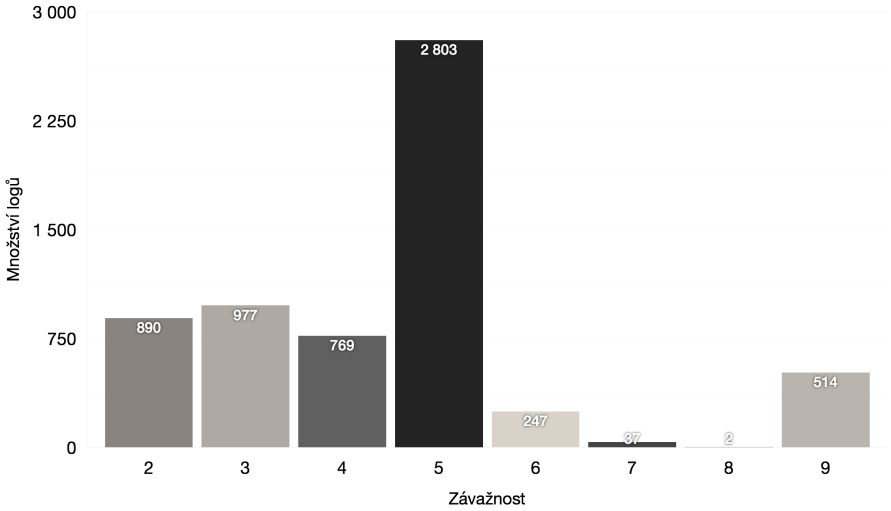
4.1 Test dynamické závažnosti s velkým důrazem na závažnost podle počtu příchodů

4.1.1 60 000 logů ve výchozím nastavení

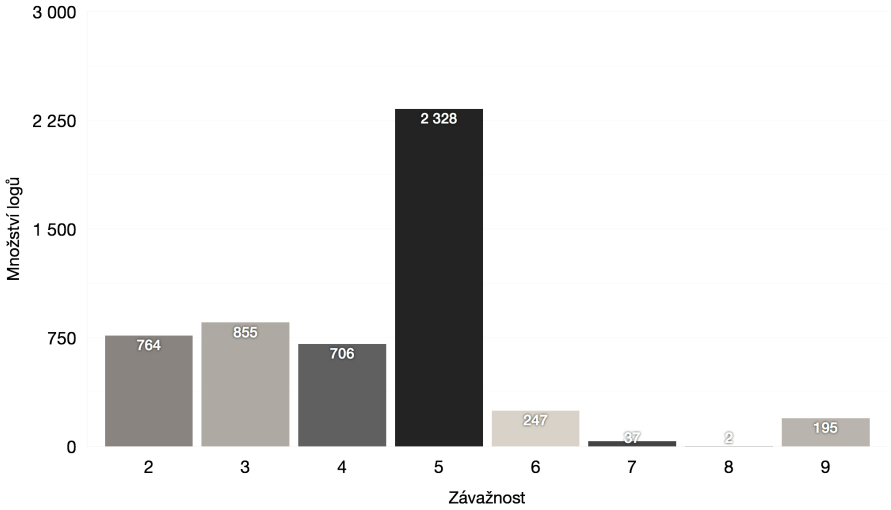
Následující data jsou změřena a spočítána při výchozí konfiguraci pouze pomocí 60 000 logů, které jsem vybral z celkového počtu. Výchozí nastavení klade největší důraz na dynamické ohodnocení dle počtu příchodů určité zprávy.

Na obrázku č. 4.1 jsou zobrazeny dynamické závažnosti, které rozšíření vypočítalo. Při začátku výpočtu byla databáze prázdná. Logů se závažností 1 bylo 54 121, ty jsem do grafu nezahrnul kvůli přehlednosti. Zajímavostí je, že 514 logů bylo ohodnoceno závažností 9, tyto logy jsou z většiny logy, které důležité nebyly, ale protože byla databáze prázdná, tak se nenašlo žádné jejich předešlé přijetí. Na následujícím obrázku č. 4.2 jsou zobrazena data bez počátečních 5 000 logů, jak je vidět, tak logů se závažností 9 radikálně ubylo.

Mezi daty bylo vyhodnoceno 6 logů, které chodí v pravidelných periodách, 5 z nich chodí v intervalu 1 minuty a jeden z nich v periodě 5 minut.



Obrázek 4.1: Test 60 000 logy



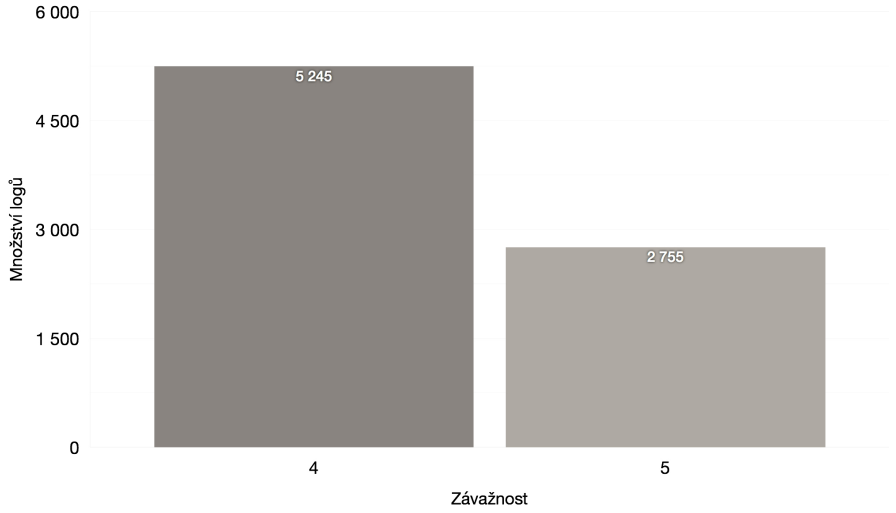
Obrázek 4.2: Test 60 000 logy bez počátečních 5 000 logů

4.2 Test celkové závažnosti s velkým důrazem závažnosti vzorů

4.2.1 8 000 logů s výchozími závažnostmi vzorů

Test zobrazený na následujícím grafu proběhl pomocí výběru zhruba 8 000 logů z předaných dat. Nastavení pro tento běh bylo upraveno jen pomocí koeficientů, které byly nastaveny tak, aby kladly velký důraz na závažnost vzoru. Konkrétně koeficient pro závažnost vzoru měl hodnotu 8, koeficient

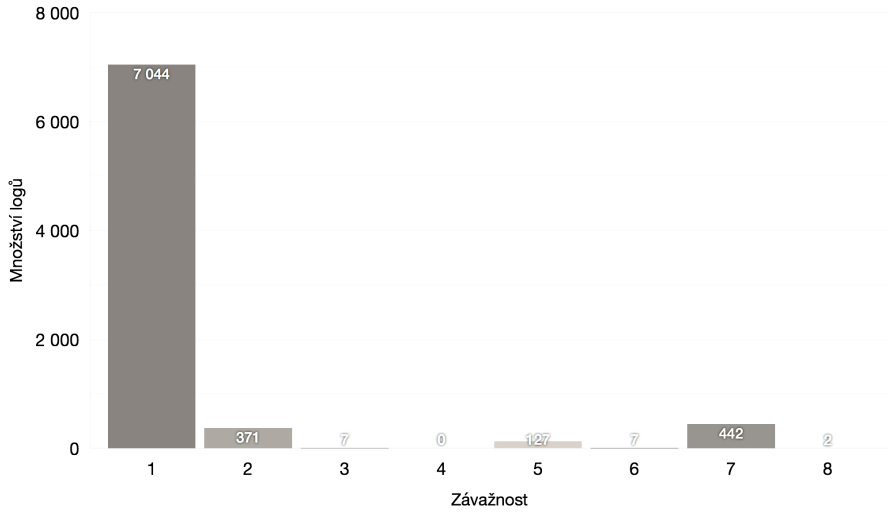
pro závažnost hosta byl nastaven na hodnotu 1 a koeficient pro dynamicky spočítanou závažnost měl též hodnotu 1.



Obrázek 4.3: Test 8 000 logy ve výchozím nastavení

4.2.2 8 000 logů s upravenými závažnostmi vzorů

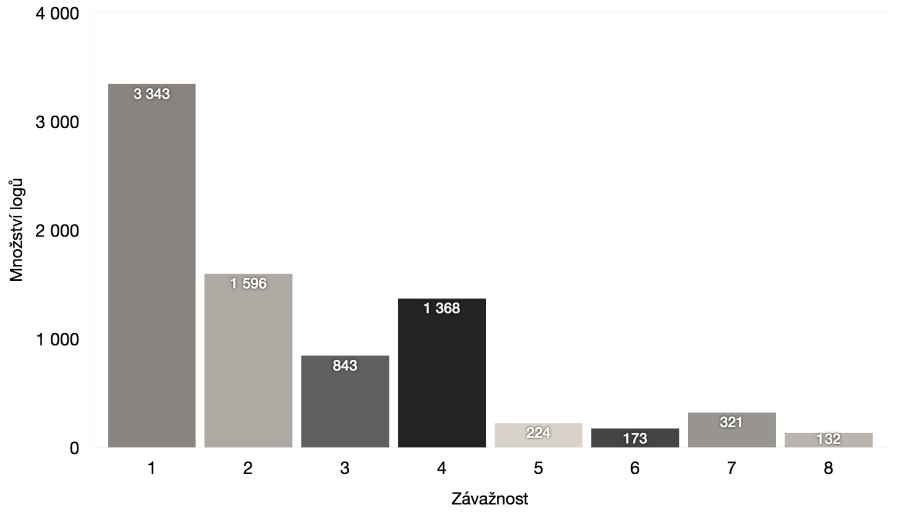
Data zobrazená na následujícím grafu jsou výsledkem běhu programu, který měl nastavené koeficienty stejně jako v předchozím případě, ale závažnosti vzorů se změnily podle potřeby administrátora.



Obrázek 4.4: test 8 000 logy s upravenými závažnostmi vzorů

4.3 Test celkové závažnosti s velkým důrazem na závažnost odesílatele logu

Na následujícím grafu jsou zobrazena data testu, který proběl ve výchozím nastavení závažnosti vzorů a v nastavení největší váhy pro závažnosti hosta. Závažnosti odesílatelů logů byly nastaveny podle potřeby administrátora. Graf výchozího nastavení závažností hostů není uveden, protože jeho hodnoty by byly stejné jako při výchozím nastavení v předešlém testu.



Obrázek 4.5: Test 8 000 logy s upravenými závažnostmi hostů

4.4 Detekce periodicity ve zmíněné části dat

Ve vzorku použitém pro testování byly detekovány 3 případy periodických zpráv. Dva z nich s periodou 5 minut a jedna z nich s periodou 1 minuty.

4.5 Závěr testování

Na závěr testování vyhodnotím zjištěné výsledky. V první části testování byl proveden test pomocí 60 000 logů, které přišly během 14 dní, protože dynamické hodnocení závažnosti počítá s databází předešlých logů. Předpokládáme, že závažnosti logů, které se budou odesílat administrátorovi, budou v rozmezí 6–9.

Jak je vidět z grafu na obrázku 4.1, tak velké množství logů bylo vyhodnoceno jako průměrně závažných, z velké části je tato hodnota složena z logů, které dosud nepřišly a nebylo tedy možné zjistit jejich závažnost podle minulých příchodů. Ale na první pohled je možné vidět, že velké množství logů

bylo potlačeno a nebudou se tedy hlásit administrátorovi. Nutno podotknout, že přes padesát tisíc logů skončilo se závažností jedna.

Následující graf na obrázku číslo 4.2 je vygenerovaný pomocí stejných dat, ale je odebráno prvních 5000 logů, které z velké části zapříčinily velké množství maximálně závažných logů a průměrně závažných logů.

Z grafu 4.3 je vidět, že při změně koeficientů a zvýhodnění závažnosti vzoru, která je ve výchozím nastavení, je nemožné správně ohodnotit dynamickou závažností. Následující graf číslo 4.4 zobrazuje data při úpravě závažnosti vzorů, z grafu je vidět, že velké množství logů bylo potlačeno.

Graf číslo 4.5 je výsledek nastavení závažností zdrojů zpráv, tedy hostů, které logy posílají. Jak je vidět, tak i touto metodou je možné výhodně závažnost ovlivnit a důležitých logů zbyde jen malé množství.

Závěr

Cílem této práce bylo vytvořit rozšíření pro systém Logstash, který je určený hlavně pro perzistentní ukládání logů a pro jejich další zpracování. Toto rozšíření mělo obohatit systém Logstash o možnost určení dynamické závažnosti jednotlivých logů, která je vypočítána ze závažnosti zdroje logu, závažnosti typu logu, závažnosti určené zdrojem a z dynamicky určené závažnosti podle počtu příchodů konkrétního logu. Tímto mechanismem lze odfiltrovat často chodící a nedůležité logy, které zpravidla administrátora nezajímají a nechat mu tak zasílat informace jen o těch důležitých. Další funkce rozšíření je určení periodicity logu. Závažnost typu logu a závažnost zdroje logu lze konfigurovat pomocí webového rozhraní, které je součástí rozšíření. Ve webovém rozhraní lze zobrazit periodické logy včetně period a počtu dosud přijatých.

V první části práce se věnuji již existujícím možnostem dohledu logů, ve zkratce jsem popsal jejich možnosti a zhodnotil jejich nedostatky. Těchto řešení není mnoho a žádné z nich nenabízí možnost dynamické závažnosti logů.

Po analýze jsem popsal použité technologie a také důvody jejich použití. Detailněji jsem zde popsal Levenshteinovu metodu pro určení rozdílu mezi řetězci, kterou využívám pro určení správného vzoru logu.

Nejdelší částí této práce je kapitola o implementaci, která je rozdělena do dvou podsekcí, ve kterých zvlášť popisuji implementaci rozšíření a implementaci webového rozhraní. V sekci o rozšíření detailně popisuji techniky určení závažnosti. Také zde uvádím strukturu databáze, kterou rozšíření využívá pro uložení pomocných dat. Rozšíření poskytuje několik konfigurovatelných parametrů, které jsou v této kapitole též popsány a vysvětleny. V následující podsekci jsem popsal implementaci webového rozhraní a jeho architekturu. Jeho hlavním účelem je možnost nastavení závažnosti zdroje logu a typu logu. Pomocí webového rozhraní lze také nastavit maximální množství odeslání konkrétního logu od konkrétního zdroje.

V poslední části se věnuji testování vytvořeného rozšíření s daty od pana vedoucího. Těchto dat je několik set tisíc, takže jsem vytvořil jen vzorek dat, který jsem využil. Testování dopadlo dle očekávání a většina logů byla i ve

výchozí konfiguraci potlačena. Ve vzorku dat bylo také nalezeno několik logů, které chodí s konkrétní periodou.

Osobní přínos práce hodnotím velice kladně, protože implementace rozšíření je provedena v jazyce Ruby a implementace webového rozhraní pomocí frameworku Ruby on Rails. Byla to tedy možnost, jak se naučit a otestovat tyto technologie, které jsou v poslední době velmi rozšířené.

Do budoucna je předpokládáno praktické nasazení tohoto rozšíření do prostředí poskytovatele připojení se zhruba tisící zdroji logů.

Literatura

[1] Marik, O.; Zitta, S.: Comparative analysis of monitoring system for data networks. In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, April 2014, s. 563–568, doi:10.1109/ICMCS.2014.6911307.

[2] Zabbix SIA: Zabbix documentation. 2015, [Online; accessed 10-Feb-2015]. Dostupné z: `https://www.zabbix.com/documentation/2.4/`

[3] Graylog: Graylog documentation. 2015, [Online; accessed 11-Feb-2015]. Dostupné z: `http://docs.graylog.org/en/1.0/`

[4] Smathers, M.: Log.io github readme. 2015, [Online; accessed 14-Feb-2015]. Dostupné z: `https://github.com/NarrativeScience/Log.io`

[5] Gerhards, R.: The Syslog Protocol. 2009, [Online; accessed 3-May-2015]. Dostupné z: `https://tools.ietf.org/html/rfc5424`

[6] Logstash documentation. 2015, [Online; accessed 14-Feb-2015]. Dostupné z: `http://logstash.net/docs/1.4.2/`

[7] Reelsen, A.: Using elasticsearch, logstash and kibana to create realtime dashboards. [Online; accessed 1-Mar-2015]. Dostupné z: `https://secure.trifork.com/dl/goto-berlin-2014/GOTO_Night/logstash-kibana-intro.pdf`

[8] Ruby community: About Ruby. [Online; accessed 3-May-2015]. Dostupné z: `https://www.ruby-lang.org/en/about/`

[9] Viswanathan, V.: Rapid Web Application Development: A Ruby on Rails Tutorial. *Software, IEEE*, ročník 25, č. 6, Nov 2008: s. 98–106, ISSN 0740-7459, doi:10.1109/MS.2008.156.

[10] Kang, H. H.; Song, C. H.; Kim, Y. C.; aj.: Metadata for efficient storage and retrieval of life log media. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, Aug 2008, s. 687–690, doi:10.1109/MFI.2008.4648025.

[11] solid IT: DB-Engines Ranking - popularity ranking of search engines. [Online; accessed 10-Mar-2015]. Dostupné z: `http://db-engines.com/en/ranking/search+engine`

[12] The PostgreSQL Global Development Group: What is PostgreSQL? [Online; accessed 25-Mar-2015]. Dostupné z: `http://db-engines.com/en/ranking/search+engine`

[13] Rane, S.; Sun, W.: Privacy preserving string comparisons based on Levenshtein distance. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, Dec 2010, s. 1–6, doi:10.1109/WIFS.2010.5711449.

[14] G. Klyne: Date and Time on the Internet: Timestamps. 2002, [Online; accessed 3-May-2015]. Dostupné z: `https://www.ietf.org/rfc/rfc3339.txt`

[15] Scores As Percentages. 2009, [Online; accessed 8-Feb-2015]. Dostupné z: `http://wiki.apache.org/lucene-java/ScoresAsPercentages`

[16] Mcheick, H.; Qi, Y.: Dependency of components in MVC distributed architecture. In *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on*, May 2011, ISSN 0840-7789, s. 000691–000694, doi:10.1109/CCECE.2011.6030542.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

FQDN Fully Qualified Domain Name

JSON JavaScript Object Notation

IETF Internet Engineering Task Force

IP Internet Protocol

MAC Media Access Control

MVC Model, View a Controller

RFC Request For Comments

SQL Structured Query Language

Příloha B

Obsah přiloženého CD

- `readme` ..... stručný popis obsahu CD
- `implementation` ..... zdrojové kódy implementace
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
- `thesis` ..... text práce ve formátu PDF