



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Platforma průmyslové spolupráce

CZ.1.07/2.4.00/17.0041

Název

Nástroj na analýzu logů

Popis a využití

- use case pro analýzu aplikačních logů
- výuka: pokročilá Java, webové frameworky

Jazyk textu

- český

Autor (autoři)

- Tomáš Došek

Oficiální stránka projektu:

- <http://lasaris.fi.muni.cz/pps>

Dostupnost výukových materiálů a nástrojů online:

- <http://lasaris.fi.muni.cz/pps/study-materials-and-tools>

Obsah

1. Úvod.....	8
1.1. Členění práce.....	8
2. Přehled dostupných nástrojů.....	10
2.1. Komerční řešení.....	10
2.1.1. Papertrail.....	11
2.1.2. Loggly.....	11
2.1.3. Logscape.....	11
2.1.4. Otus siem.....	12
2.1.5. XpoLog.....	12
2.1.6. LogLogic.....	13
2.1.7. Logentries.....	13
2.2. Volně dostupná řešení – Open Source, Freeware.....	13
2.2.1. Splunk.....	13
2.2.2. Logstash.....	14
2.2.3. Graylog 2.....	14
3. Grok – popis, měření.....	15
3.1.1. Měření počtu řádků zpracovaných za jednotku času.....	17
3.1.2. Měření počtu pravidel, jimiž je Grok schopen obsloužit parsování.....	18
3.1.3. Vyhodnocení testů.....	18
4. Návrh nástroje pro generování pravidel pro parsování neznámého vstupu.....	19
4.1. Podrobný popis návrhu vytvářeného nástroje.....	20
4.2. Návrh schématu databáze.....	22
4.2.1. Požadavky na funkce systému.....	22
4.2.2. Analýza podstatných jmen.....	23
4.2.3. Schéma databáze.....	24
4.2.4. Definice databázových entit.....	24
5. Programové prostředky a důležité části nástrojů goracle a goracleServer.....	28
5.1. GrokWrap.....	28
5.2. GoracleServer.....	29
5.3. Goracle.....	30
5.4. Vývojový cyklus nástroje.....	36
6. Přehled funkcí navrženého systému.....	37
7. Zhodnocení práce.....	40
8. Použité zdroje.....	41
8.1. Citace.....	41
8.2. Seznam obrázků.....	42
8.3. Seznam použitých zkratk a výrazů.....	43
9. Přílohy.....	46
9.1. Kompilace a nasazení nástroje.....	46
9.1.1. Kde vzít zdrojový kód?.....	47
9.2. Vytvoření databáze postgres pro nástroj.....	47
9.3. Instalace Groku.....	48
9.4. Nasazení grokwrap.....	49
9.5. Nasazení goracleServer.....	50
9.6. Nasazení goracle.....	51
9.7. Jakým způsobem nahlásit problém?.....	53

1. Úvod

Informační technologie jsou v současnosti nejvýznamnějšími prostředky pro uchovávání a zpracování dat. Většina operací, které v takovýchto systémech probíhají, generují poměrně velké množství auditních a logovacích záznamů v různých formátech, které jsou podmnožinou přirozeného jazyka.

Účelem této práce je popsat nástroje, které umožní uživateli centralizovaně tyto záznamy zpracovávat a srozumitelně předkládat tak, aby bylo možno těchto záznamů využít například pro monitorování stavu systémů, předvídání rizik a včasnou odpověď na hrozby.

Nástroje, kterým se tato práce věnuje poskytují nejen statistiky, ale i funkce pro převod logovacích záznamů do uživatelem zvoleného strukturovaného formátu vhodného pro další zpracování.

V praktické části této práce je uveden způsob využití zdarma dostupných nástrojů pro analýzu logovacích záznamů založených na parsování vstupu pomocí vzorů. Součástí praktické části je také vypracování programových prostředků pro správu a automatickou detekci těchto vzorů pro neznámý vstup.

1.1. Členění práce

Práce je rozdělena do sedmi kapitol. Úvodní část je věnována stručnému představení diplomové práce, jejích cílů a členění práce.

Druhá kapitola obsahuje rešerši dostupných nástrojů pro řešení problematiky analýzy a parsování logovacích záznamů včetně popisu jejich výhod a nedostatků.

Ve třetí kapitole byl zvolen nástroj Grok vyvíjený Jordan(em) Sissel(em). Tento jednoduchý nástroj, který slouží jako základ Open Source řešení v dané oblasti, poskytuje programátorům rozmanité prostředky pro formátování výstupu a filtrování vstupních dat. Dále jsou v této části prezentovány výsledky měření vybraných vlastností tohoto nástroje.

Čtvrtá kapitola se zabývá návrhem nástroje pro generování pravidel pro neznámý vstup. Součástí této kapitoly je dále i jednoduchý popis architektury, které slouží jako ukázka možného nasazení nástroje.

Pátá kapitola popisuje prostředky použité k naprogramování nástroje pro automatické generování a správu parsovacích pravidel. Dále jsou v této kapitole detailněji popsány vybrané součásti návrhu řešení tohoto nástroje.

Šestá kapitola obsahuje přehled funkcí navrhovaného nástroje. Na ukázkových obrazovkách ze systému jsou znázorněny funkce správy parsovacích pravidel, nahrávání optimalizovaného souboru s pravidly pro parsování na vzdálené počítačové systémy a další funkce systému.

Obsahem sedmé kapitoly je shrnutí a diskuse dosažených výsledků.

Za sedmou kapitolou je uveden seznam použitých zdrojů, seznam obrázků, seznam použitých zkratk a přílohy této práce. Nedílnou netištěnou součástí této práce je CD nosič obsahující text této práce a kompletní zdrojový kód nástroje, jež byl vytvořen jako vypracování praktické části této diplomové práce. Nosič rovněž obsahuje skripty, které byly použity pro měření výkonnosti nástroje Grok a ukázkové konfigurace nástroje pro generování parsovacích vzorů.

2. Přehled dostupných nástrojů

V dnešní době složitých výpočetních systémů a rozsáhlých počítačových sítí se stále důležitějším faktorem stává efektivní monitorování systémů a rychlé nalezení problémů, které se mohou objevit během jejich provozu. Běžným způsobem nalezení problému je interpretace logovacích záznamů a hledání takových záznamů, které obsahují chybná data, varování, či výpisy chybových hlášení různého druhu. S množstvím těchto záznamů roste i jejich nepřehlednost a riziko přehlédnutí záznamů důležitých pro úspěšné odstranění závad.

K řešení této problematiky již bylo vyvinuto velké množství nástrojů. Tyto nástroje se buď na aktivním způsobem (příkladem může být populární software Nagios), za pomoci různých agentských aplikací, nebo na pasivně, analýzou logovacích záznamů, snaží poskytnout administrátorům infrastruktur dostatečný počet informací pro úspěšnou analýzu a odhalení potíží vyskytujících se v jejich infrastrukturách.

Tato práce se zaměřuje na druhou ze zmiňovaných skupin nástrojů. Nástroje, které analyzují logovací záznamy, mají značnou výhodu ve výpočetní nenáročnosti na hostovský systém. Ve většině případů je třeba pouze přesměrovat logovací záznamy pomocí některého protokolu, např. rsyslog (Remote System Logging), na stroj, který provádí analýzu dat.

Následující text představuje dostupné nástroje pro parsování logovacích záznamů, jejich zpracování a zobrazení výsledků v uživatelsky přívětivé formě. Jsou zde popsány řešení publikovaná v rámci některé z „Open Source“ licencí, ale i komerční nástroje řešící problematiku v dané oblasti.

2.1. Komerční řešení

Zde jsou popsána řešení, která lze dlouhodobě používat pouze za předpokladu zakoupení licence. Komerční řešení umožňují nasazení v různých podmínkách, ve většině případů umožňují parsování logovacích a auditních záznamů pocházejících pouze z omezené množiny aplikací. Některé z komerčně dostupných nástrojů jsou poskytovány i v neplacených variantách (většinou se však nejedná o skutečný freeware, ale pouze o časově limitovanou zkušební verzi, jejíž omezení v podstatě znemožňují její nasazení do reálného provozu).

U jednotlivých komerčních nástrojů jdou uvedeny informace o limitech a podmínkách jejich zakoupení, pokud jsou tyto informace veřejně dostupné.

2.1.1. Papertrail

Papertrail je komerčně nabízeným řešením založeným na prvcích cloudových výpočtů.

Základní neplacená licence Papertrail je zdarma, ale povoluje zpracování pouze 200 MB dat. Licence pro použití v rozsáhlejší produkčním prostředí je nabízena za 225 amerických dolarů a poskytuje úložný prostor pro 16 GB dat měsíčně.

Architektura papertrailu umožňuje uživateli zasílat logovací a auditní záznamy prostřednictvím protokolu rsyslog. Papertrail poskytuje přístup ke zpracovaným datům pomocí rozhraní splňujícího parametry architektury REST.

Bohužel tento nástroj umožňuje pouze zpracování omezeného počtu typů logovacích a auditních záznamů (například záznamy formátů Linux syslog, textové logovací záznamy, apache, MySQL, Ruby on Rails, záznamy systému Windows, tomcat, Heroku, routerů a firewallů).[1]

2.1.2. Loggly

Loggly je další z komerčních nástrojů poskytovaných ve veřejném cloudu. Sběr dat u loggly je omezen pouze na data zasílaná aplikacemi ve specifických programovacích jazycích, databázovými servery MySQL a PostgreSQL. Systémové logovací záznamy mohou být zaslány prostřednictvím rsyslog protokolu. Ke zpracovaným datům lze přistupovat pouze prostřednictvím internetového prohlížeče.

Tento nástroj byl již otestován a nasazen do produkčních prostředí velkých společností podnikajících v oboru informačních technologií, jako například Sony, Logitech, nebo Electronic Arts.

Zdarma poskytovaná licence tohoto nástroje, pojmenovaná Lite, poskytuje úložný prostor o velikosti 200 MB denně. Licence Pro, která poskytuje dostatek prostoru pro větší produkční prostředí (7 GB za den) je nabízena za 349 amerických dolarů měsíčně.[2]

2.1.3. Logscape

Logscape je „necloudová“ alternativa nástroje pro analýzu a monitorování dat na základě indexace vstupních logovacích záznamů. Tento nástroj vyžaduje instalaci na vlastní počítač.

Logscape umožňuje zpracovávat data z několika možných vstupů. Prvním z těchto vstupů je rsyslog protokol. Logscape dále umožňuje vystavit vybrané TCP porty a zpracovávat data příchozí na těchto portech. Dále je možné na-

čtení aplikační knihovny logscape v aplikacích napsaných v různých programovacích jazycích. Tyto programy pak ukládají své logovací záznamy přímo prostřednictvím „logscape manager“ aplikace. Posledním možným vstupem je přímé čtení logovacích záznamů ze stroje, kde je logscape manager nasazen.

Základní neplacená licence umožňuje nasadit logscape na jeden stroj a neumožňuje sběr dat ze vzdálených prostředků prostřednictvím sítě. Licence pro deset strojů s centralizovaným serverem pro analýzu dat lze zakoupit za 3000 amerických dolarů na rok.[3]

2.1.4. Otus siem

Otus siem je dalším nástrojem pro analýzu logovacích záznamů dostupným pouze na lokálním počítači. Otus siem nabízí rozsáhlé možnosti vstupů dat. Příkladem vstupu jsou serverové komponenty komunikující prostřednictvím snmp, sflow, rsyslogu. Dále lze logovací záznamy nahrát pomocí ftp, ssh nebo http protokolů.[4]

Otus siem zakládá svůj úspěch na distribuci výpočtů spojených s analýzou veškerých dat. Společnost Bitsteer provozující nástroj Otus siem garantuje redundanci všech dat, které byly tímto nástrojem sesbírány.[4] Zpracovaná data jsou však přístupná pouze prostřednictvím webového prohlížeče.

Společnost Bitsteer provozující Otus siem nepublikuje ceny jednotlivých licencí na žádné z oficiálních internetových stránek.

2.1.5. XpoLog

XpoLog je řešení, které oproti předchozím produktům nabízí i sestavování auditních zpráv o stavu systému podle různých metodik. Příkladem takových metodik jsou PCI, SOX, HIPPA a FISMA. Tento produkt také zvládá zpracovávat vstupy z různých zdrojů, ale neposkytuje žádné rozhraní pro přístup ke zpracovaným datům.

Výhodou tohoto nástroje je přímý monitoring dat v různých databázových serverech, jako jsou postgres a MySQL. Společnost poskytující XpoLog se snaží tuto funkci poskytnout pro jakýkoli databázový server, pro který existuje JDBC ovladač.

Pro lokální použití na jednom stroji lze využít zkušební verzi XpoLog. Nejsou však známy ceny pro komerční využití ve větším produkčním prostředí.[5]

2.1.6. LogLogic

LogLogic je škálovatelným řešením problému analýzy a zobrazení dat se sbíraných prostřednictvím logovacích záznamů určeným pro velké společnosti. Tento nástroj poskytuje množství analytických možností v závislosti na typu zpracovávaného logovacího záznamu.

Tento program je plně proprietární a společnost TIBCO nezveřejňuje další informace o LogLogic.[6]

2.1.7. Logentries

Logentries je posledním z komerčních produktů, které jsou v této práci zmíněny. Oproti předchozím aplikacím nabízí širší integraci se světoznámými cloudovými službami, jako je Amazon Web Services cloud, nebo cloudová platforma Heroku. Pro přístup k datům lze zvolit například Rabbit message queue, která hranice tohoto produktu posouvá dále v porovnání s konkurencí.

Základní bezplatná licence Logentries nástroje nabízí 1 GB úložného prostoru pro data měsíčně a její platnost časově omezena. Platinum licence nabízí 50 GB prostoru měsíčně za cenu 99 amerických dolarů.[7]

2.2. Volně dostupná řešení – Open Source, Freeware

V kategorii nástrojů založených na analýze logů není nepřehledné množství nástrojů, které by byly distribuovány zcela zdarma. Následujících několik však svými funkcemi předčí mnohá komerční řešení. Některá komerční řešení navíc používají tyto nástroje, jako základ pro svůj model zpracování logovacích záznamů.

Oproti komerčním řešením poskytují „Open Source“ řešení většinou navíc vysokou agilitu co se týče rychlosti rozvoje nových funkcí, neboť na nich pracují skupiny vývojářů určitým způsobem zainteresované v dalším rozvoji nástroje.

2.2.1. Splunk

Společnost splunk nabízí řešení založené na zásuvných modulech, z nichž každý poskytuje určité funkce. Od analýzy logových záznamů tak může tento produkt plynule přejít k analýze tzv. velkých dat.

Tyto plug-iny společnost poskytuje pod různými licencemi a jejich ceny se liší. Tato společnost nabízí také takové zásuvné moduly, které umožní auditovat systémy podle několika standardů, jmenovitě standardu SEC, PCI, HIPAA a FISMA.

V porovnání s předchozími řešeními nabízí navíc i kompletní podporu pro produkty společnosti Microsoft. Zejména přímé monitorování Microsoft Ex-

change, jednoho z velmi často využívaných produktů pro správu e-mailové komunikace, a Microsoft Active Directory, produktu pro správu identit a oprávnění, mohou být pro společnosti stavící své výpočetní střediska na těchto technologiích, výhodou.

Splunk má časově neomezenou volnou verzi, která je poskytnuta k lokálnímu stažení a dále pak verze založené na cloudových prostředcích, které společnost poskytuje pod placenou licenci.[8]

2.2.2. Logstash

Logstash je první plně „Open Source“ alternativou ke komerčním řešením v této oblasti. K analýze dat využívá programu Grok a nabízí možnost vizualizace dat např. prostřednictvím známého javového frameworku Elastic search. [9]

Logstash umí zpracovávat libovolný vstup tak, jak jej produkuje program Grok, jeho možnosti jsou tedy téměř neomezené.

Logstash je však zajímavou alternativou ke komerčním řešením, neboť není vázaný pouze na linuxový operační systém. Lze jej poměrně pohodlně nasadit i na systému Microsoft Windows, který disponuje platformou Java. [10]

Pro uživatele, kteří touží po extrémním nasazení v cloudu, logstash nabízí integraci s populárním nástrojem pro automatizaci instalací – Puppet.

Jedinou významnou nevýhodou tohoto nástroje je, že parsovací pravidla pro Grok je třeba předložit v souboru, který uživatel manuálně sepíše. Toto může být pro některé administrátory poměrně složité a pokud chtějí např. zpracovávat data z aplikací, které jsou jejich společnosti vyvíjeny. K odstranění tohoto problému by bylo potřeba poskytnout uživateli alespoň základní komfort v procesu tvorby nových pravidel, například automatickým odvozováním. Tento problém budeme řešit v praktické části této práce.

2.2.3. Graylog 2

Graylog 2 je aplikace která rozvíjí možnosti nástroje logstash směrem pokročilejších funkcí pro odhalování chyb v běhu software. Graylog 2 poskytuje rozhraní REST pro přístup k analyzovaným datům.

Graylog 2 v porovnání s logstash nabízí poměrně zajímavou architekturu slibující možnost nasazení i v opravdu rozsáhlých prostředích. Veškeré operace v prostředí Graylog 2 jsou totiž distribuované prostřednictvím Apache Kafka clusteringu.

Bohužel Graylog2 sdílí stejný problém jako logstash a to tedy nepřívětivost při administrování a konfigurování pravidel pro parsování dat.[11]

3. Grok – popis, měření

Tato kapitola diplomové práce popisuje nástroj pojmenovaný Grok. Tento nástroj byl zvolen pro vypracování praktické části jako základ pro parsování logovacích záznamů, stejně tak poslouží jako nástroj pro automatickou analýzu neznámých záznamů a návrh parsovacích pravidel pro takovéto záznamy.

Tato aplikace byla publikován Jordan(em) Sissel(em) ve své původní podobě již v roce 2009, při čemž vyvíjena byla zhruba od roku 2007. Jedná se o poměrně jednoduchý program napsaný v jazyce C (obsahuje také knihovnu pro práci s tímto programem v jazyce Ruby), jehož vstupem je prostý textový soubor obsahující pravidla pro parsování logovacích záznamů, dalším vstupem jsou pak samotné logovací záznamy.[12]

Jakým způsobem Grok funguje? Nejjednodušší je vysvětlit princip fungování tohoto programu na jednoduchém vstupu a jednoduchém pravidlu. Mějme tedy následující vstup (logovací záznam webového serveru apache):

```
24.217.161.84 - - [31/Aug/2011:08:34:44 -0700] "GET /apple-touch-  
icon.png HTTP/1.1" 404 514 "-" "Reeder/1.5.1 CFNetwork/485.13.9  
Darwin/11.0.0"
```

Tento vstup zkusíme rozparsovat následujícím pravidlem – pravidlem pro datum, formát dle standardů Evropské unie, pro úplnost je nutné dodat, že YEAR, MONTHNUM a MONTHDAY jsou regulární výrazy, které jsou součástí základní báze pravidel programu Grok:

```
DATE_EU %{YEAR}[/-]%{MONTHNUM}[/-]%{MONTHDAY}
```

Nejprve se Grok pokusí aplikovat regulární výraz (pravidlo pro parsování bez definice jména pravidla), pokud se tato operace úspěšně ukončí vyvolá se reakce adekvátní definici úspěchu tzv. „match“. Nepodaří-li se na vstup regulární výraz aplikovat a zároveň, je-li definována nějaká akce pro větev neúspěchu tzv. „no-match“, je výstup předán ke zpracování této větvi. Dle platné konfigurace, která definuje požadavky na výstup pro větve „match“ a „no-match“, Grok naformátuje výstup a odešle jej.

V našem případě tedy Grok vyzkouší, zda lze aplikovat regulární výraz obsažený v pravidlu „DATE_EU“ na výše uvedený logovací záznam. Jelikož tento regulární výraz neodpovídá formátu logovacího záznamu – v tomto záznamu se například vyskytuje IP adresa, datum je ve formátu běžném pro Spojené státy americké, jsou zde i další data, která nejsou pravidlem „DATE_EU“ vůbec zadefinována – bude tato operace označena jako neúspěšná a záznam bude předán ke zpracování pomocí vyhodnocovací větve „no-match“ v případě, že je tato větev zadefinována v konfiguraci, se kterou byl program Grok volán.

Jak již bylo výše popsáno, Grok pracuje na bázi různých vyhodnocovacích větví. Jakým způsobem lze ale tyto větve definovat? Jak již víme, program grok se běžně načítá s určitou konfigurací. Tato konfigurace například stanovuje, jaké pravidlo pro parsování se používá, jakým způsobem se výstup programu formátuje a kam se tento výstup následně předává. Následující kód je ukázkou velmi jednoduché konfigurace programu Grok.

```
program {
  exec "since /var/log/messages"
  match {
    pattern: "this is not an error"
    reaction: none
    break-if-match: yes
  }
  no-match {
    reaction: none
  }
}
```

Tato konfigurace má tedy pouze dvě větve. Jednu větev kladnou nazvanou „match“, neboli shoda, která ukončí Grok ve chvíli, kdy vstup odpovídá vzoru „this is not an error“ a druhou zápornou, která ve své podstatě pouze nechá program běžet a nebude provádět žádné akce. Za vstup logovacích záznamů v tomto případě považujeme výstup příkazu „since /var/log/messages“, který vypíše kompletní obsah souboru /var/log/messages.

Standard konfigurace programu Grok umožňuje i definici více větví typu „match“ a „no-match“, při čemž jejich vyhodnocování probíhá lineárně, tedy v takovém pořadí, jaké je definováno v konfiguračním souboru.

Pro účely této diplomové práce později popisují konfiguraci, která umožní využít Grok jako základ pro případy, kterými se zabývá praktická část této práce.

Mezi dobré vlastnosti programu Grok patří i schopnost formátovat výstup do strukturovaného formátu, např. JSON, nebo jej zasílat jako parametr libovolného příkazu linuxové příkazové řádky, čehož budeme v dalších částech této práce hojně využívat.

Poměrně důležitým faktorem ovlivňujícím použitelnost nástroje Grok, jako základu pro naši práci je počet řádků, které bude Grok schopen zpracovat za vteřinu a maximální počet pravidel, který lze použít aniž by tento nástroj ukončil svou činnost s chybovou hláškou. V následující podkapitola se tedy zaměřuje na změření těchto důležitých vlastností.

Pro účely veškerých měření je v této práci použit následující skript, který je volán z linuxového terminálu.

```

#!/bin/bash
START=$(/usr/bin/date -u "+%s.%N")
/usr/bin/grok -f /thesis/meassureMe.grok >
/thesis/test_output.txt
END=$(/usr/bin/date -u "+%s.%N")
/usr/bin/echo "Start date/time is: " $START
/usr/bin/echo "End date/time is: " $END
DELTA=$(echo "($END - $START) * 1000" | bc)
/usr/bin/echo "Delta is: " $DELTA ms

```

Tento skript ve své podstatě zavolá program grok s určenou konfigurací, která je uložena v souboru /thesis/meassureMe.grok. Výstup operací, které Grok provede přeměruje do souboru /thesis/test_output.txt, kde lze ověřit, zda test dopadl správně, například za použití správného pravidla byl vstup naformátován do správného tvaru.

Před spuštěním libovolné operace zaznamená skript systémový čas v přesnosti na nanosekundy. Po provedení této operace opět zaznamená čas ve shodné přesnosti a na standardní výstup vypíše rozdíl těchto dvou hodnot. Při měření zanedbává výše uvedený skript délku volání příkazu „date“, neboť je tato hodnota v praxi invariantní díky způsobu jakým příkaz date získává čas z linuxového kernelu.

3.1.1. Měření počtu řádků zpracovaných za jednotku času

První důležitou vlastností je počet řádků, který je Grok schopný zpracovat. Pro začátek byla tato propustnost změřena při použití základní báze pravidel, které definují například formát běžného data, nebo textového obsahu. Při použití takovéto konfigurace byl Grok schopný zpracovat logovací soubor webového serveru apache o délce dvaceti tisíců řádků za čas zhruba 121 ms. Je nutno podotknout, že tento čas závisí také na definici testu ve smyslu hledání jednotlivého pravidla, nebo prohledávání nad všemi pravidly pro parsování. Při prohledávání na základě jednoho vybraného pravidla totiž Grok nejdříve seřadí pravidla, nalezne pravidlo, pomocí kterého se dále orientuje a teprve po té zahajuje porovnávání zda vstup tomuto pravidlu odpovídá, či ne. Při takovémto testu se doba porovnání vstupu o dvaceti tisících řádcích protáhne zhruba na 123 ms, při bazovém souboru s pravidly o délce 91 pravidel.

Výsledek tohoto měření nám říká, že Grok je schopný zpracovat více, než dostatečný počet logovacích záznamů za vteřinu. Nejedná se tedy o limitující faktor.

3.1.2. Měření počtu pravidel, jimiž je Grok schopen obsloužit parsování

Pro účely tohoto testu je třeba přidat další funkci a to generátor nových parsovacích vzorů.

Tento skript lze nalézt na CD nosiči přiloženém k této práci. Bohužel kvůli jeho rozsáhlosti nelze obsah tohoto skriptu vložit přímo do textu této práce. Zevrubný popis práce tohoto skriptu by však měl být dostačující pro pochopení formátu souboru s pravidly, který byl použit pro tento test.

Jak tedy generátor funguje? Nejdříve se do souboru zapíše 8 pravidel představující nutná bázová pravidla pro definici typů. Po té se vygeneruje uživatelem definovaný počet částí pravidla tak, že se náhodně zvolí délka pravidla (od délky jedna po délku 15) a dále se náhodně zvolí bázové pravidlo které se dosadí za danou část výsledného pravidla. Po dosazení všech částí se pak pravidlo pojmenuje a zapíše do souboru určeného pro test. Tento soubor pak je v konfiguraci zmíněn jako zdrojový pro soubor pravidel. Test po té zahrnoval vstup logovacích záznamů ze souboru o dvaceti tisících řádků a souboru s pravidly o generovaném počtu pravidel.

Pro soubor s tisíci pravidly, při výběru jednoho z pravidel zpracování vstupu trvá zhruba 564 ms. Pro soubor s deseti tisíci pravidel trvalo zpracování zhruba 572 ms, zpracování 100 000 pravidel při daném vstupu trvalo zhruba 744 ms. Závěrečný test, který používal soubor s jedním milionem pravidel trval při stejném testu zhruba 1399 ms.

3.1.3. Vyhodnocení testů

Účelem předcházejících testů bylo zjistit zda je Grok schopen zpracovat vysoký počet pravidel pro parsování a ověřit propustnost tohoto programu. Dříve popsané scénáře ověřily, že nedojde ke zhroucení programu Grok při vysokém počtu pravidel. Dojde k ovlivnění rychlosti zpracování a to k takovému zpomalení, které stále neohrožuje celkovou použitelnost tohoto nástroje jako základu pro praktickou část této práce. Ověřená propustnost programu Grok se také nezdá být limitujícím faktorem pro použití v této diplomové práci.

4. Návrh nástroje pro generování pravidel pro parsování neznámého vstupu

Tato kapitola nejdříve popíše problém, který řešíme a po té návrh samotného nástroje.

Komerční řešení jsou pro běžné účely bohužel placená a navíc jsou schopná zpracovávat jenom logovací záznamy pocházející ze značně omezeného počtu aplikací, případně kladou nároky na formátování logovacích záznamů aplikací, jejichž data jsou analyzována.

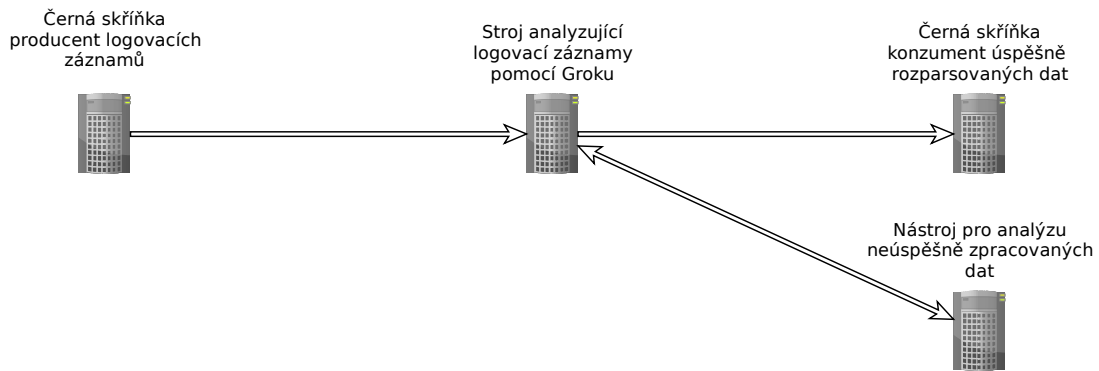
Nekomerční řešení na druhé straně nejsou placená a jsou schopná zpracovat libovolné formáty logovacích záznamů, bohužel však kladou přehnané požadavky na znalosti daného problému a tím omezují počet uživatel, kteří po takovémto řešení sáhnou.

Jako cíl praktické části této diplomové práce byl tedy stanoven návrh a implementace nástroje, který by těžil z dobrých vlastností nekomerčních nástrojů, jako je Graylog 2, nebo Logstash, a zároveň by vyplnil díru v podobě uživatelské nepřívětivosti správy těchto nástrojů.

Dle těchto požadavků si tedy kladu za cíl této práce vytvoření nástroje, který by uživateli zobrazoval návrhy podoby parsovacích pravidel pro vstup, který byl Grokem identifikován jako neznámý (tedy nevyskytující se v souboru s pravidly pro parsování).

Jedním z požadavků Laboratoře softwarových architektur a informačních systémů byla implementace, která by umožnila přesměrování výstupu pro úspěšně zpracovaná data do externího nástroje ve formátu JSON. Tento nástroj je pro účely této práce vnímán jako tzv. „černá skříňka“, neboli nástroj, kterým se tato práce zajímá pouze do míry předání úspěšně rozparsovaného výstupu programu Grok tomuto nástroji.

Architekturu takového nástroje lze tedy jednoduše popsat schématem, jež lze nalézt na následující straně.



Ilustrace 1: Zevrubná architektura požadovaného nástroje

4.1. Podrobný popis návrhu vytvářeného nástroje

Jak již bylo zmíněno v druhé kapitole, řešení problému parsování a analýzy existuje mnoho. Některá z nich jsou založena na instalaci specifického software na stroje, ze kterých pocházejí logovací záznamy, jiná jsou založena na cloudových výpočtech. Účelem této práce je navrhnout nástroj, který nebude uživatele limitovat ve způsobu přístupu k aplikaci jako takové.

Jako základ pro nástroj vytvořený v této diplomové práci byl zvolen nástroj Grok, který je schopen zasílat výstup jako parametr libovolného příkazu linuxového terminálu. Nyní je třeba rozvrhnout možné případy nasazení zamýšleného nástroje.

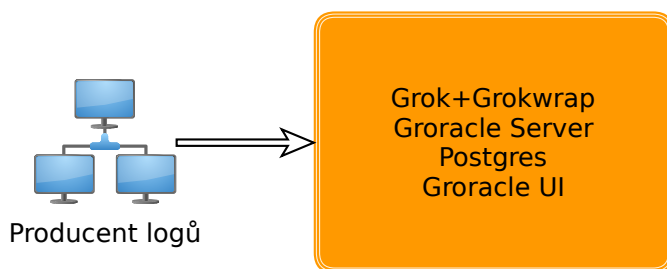
Prvním krajním případem je nasazení nástroje na jeden jediný počítač. V takovém případě by stačilo pouze výstup Groku směřovat na nástroj, který provádí analýzu a vizualizaci dat.

Druhým krajním případem je nasazení nástroje jako cloudového řešení. V takovém případě je nutné řešit sběr dat z Groku, jejich zpracování a následné zobrazení při čemž výsledná architektura z bezpečnostních důvodů může vyžadovat, aby stroje produkující logovací záznamy nepřistupovaly přímo do cloudového úložiště, ale přistupovaly do něj pomocí programu, který by určitým způsobem obalil a zabezpečil komunikaci.

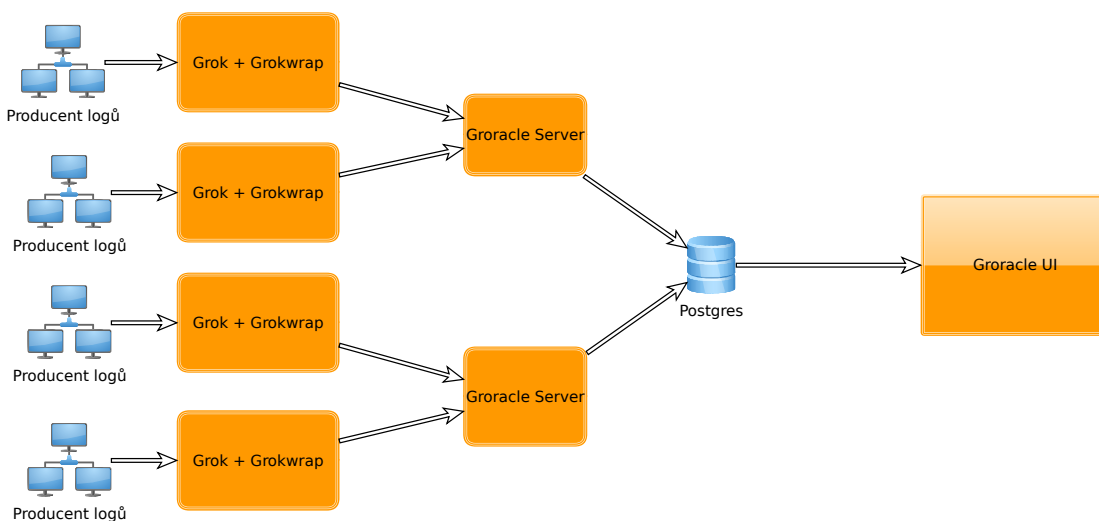
Když tyto požadavky dáme dohromady, vznikne nám potřeba pro nástroje Grok, jako základ celého řešení, nástroje pro sběr/zpracování/uložení dat a zobrazení dat uživateli. Dle této dekompozice pojmenujme nástroje vyvinuté v rámci této diplomové práce jako grokwrap (skript spouštějící Grok s definovanou konfigurací jako systémového daemona), goracleServer (serverová komponenta zodpovědná za shromáždění, zpracování a následné uložení dat) a na závěr goracle (UI komponenta, která bude reprezentovat data z databáze,

řešit přístup a oprávnění uživatel, atd.). Pro samotné uložení dat pak byl zvolen databázový engine postgres, který je jedním z mála zdarma dostupných databázových serverů, jehož škálovatelnost umožňuje i velmi vysokou zátěž. Na následujících schématech jsou zobrazeny již dříve zmíněné případy.

Ilustrace 2 vyobrazuje případ, kdy všechny nástroje nasadíme na jednotlivý stroj. Veškeré komponenty jsou přímo propojené a zároveň poskytují uživateli uživatelské rozhraní pro interakci změny dat a správu konfigurací.



Ilustrace 2: Jednoduchá architektura



Ilustrace 3: Příklad škálování navrhovaného nástroje

Ilustrace 3 pak zobrazuje příklad škálování této architektury do vyšších rozměrů. Na tomto schématu lze vidět, několik serverových komponent, které sbírají data do jedné databáze, se kterou lze interagovat prostřednictvím UI rozhraní.

Tím, že je `goracleServer` poměrně pohyblivou složkou lze vytvořit nepřehledné množství architektur o velmi různorodé podobě. Dříve uvedené ilustrace jsou tedy pouze jednoduchá ukázka možností požadovaných po návrhu nástroje.

Pod pojmem producent logů je možné si představit pouze jeden jednotlivý počítač, ale může se jednat i o kompletní počítačovou síť, která předává logovací záznamy Groku prostřednictvím vhodného protokolu. Příkladem takového protokolu může být `rsyslog`, nebo i jednoduché zasílání logovacích záznamů prostřednictvím TCP portu.

4.2. Návrh schématu databáze

Jak je z předloženého návrhu již zřejmé, velmi důležitou roli v tomto projektu hraje databáze. Jako databázový server byl zvolen `postgres` díky svým dobrým vlastnostem a jednoduché škálovatelnosti. Tento databázový server je navíc poskytován zdarma téměř ve všech linuxových distribucích.

V extrémním případě lze databázi `postgres` i replikovat, nebo clusterovat a tak zvýšit výkonnost databázového serveru a poskytnout větší množství spojení na databázi, než je standardních 100 databázových spojení, redundanci uložených dat a jiné vlastnosti distribuovaného systému. [13]

Při návrhu a implementaci nástrojů `goracle` a `goracleServer` byl brán ohled na to, aby uživatelské rozhraní používalo co nejméně databázových spojení a tudíž zůstalo co nejvíce těchto prostředků dostupných pro `goracleServer`, a tedy bylo možné cílové architektury nasazení nástroje co nejlépe škálovat.

4.2.1. Požadavky na funkce systému

Tato podkapitola rozebere funkce, které budeme po systému požadovat a na jejich základě následně odvodíme důležité databázové entity.

První a zároveň zřejmou funkcí a zároveň entitou je uložení vstupu. U vstupu patrně bude třeba uložení data, kdy došlo k uložení záznamu, host ze kterého byl záznam sebrán, samotný záznam a po té návrh pravidla pro parsování daného vstupu.

Dalším požadavkem zadání bylo, aby tento nástroj obsahoval pokročilé funkce. První takovou funkcí je správa uživatel a autorizace uživatel vůči částem systému. Logicky lze odvodit, že pro větší počet uživatel již bude poměrně výhodné tyto oprávnění pro uživatele také řešit na úrovni rolí.

Poměrně žádanou funkcí by mohlo být nahrání nového souboru s pravidly na hosty, na kterých je nasazen `grokwrap`. Pro velký počet hostů by opět mohlo být vhodné tyto host nějakým způsobem seskupit a umožnit hromadné nahrání na více hostů zároveň do takovéto skupiny.

Poslední funkcí systému by mohla být definice černé listiny, která by umožnila skrytí záznamů, o které nemá uživatel zájem a nechce je již vídat, či jinak s nimi pracovat.

4.2.2. Analýza podstatných jmen

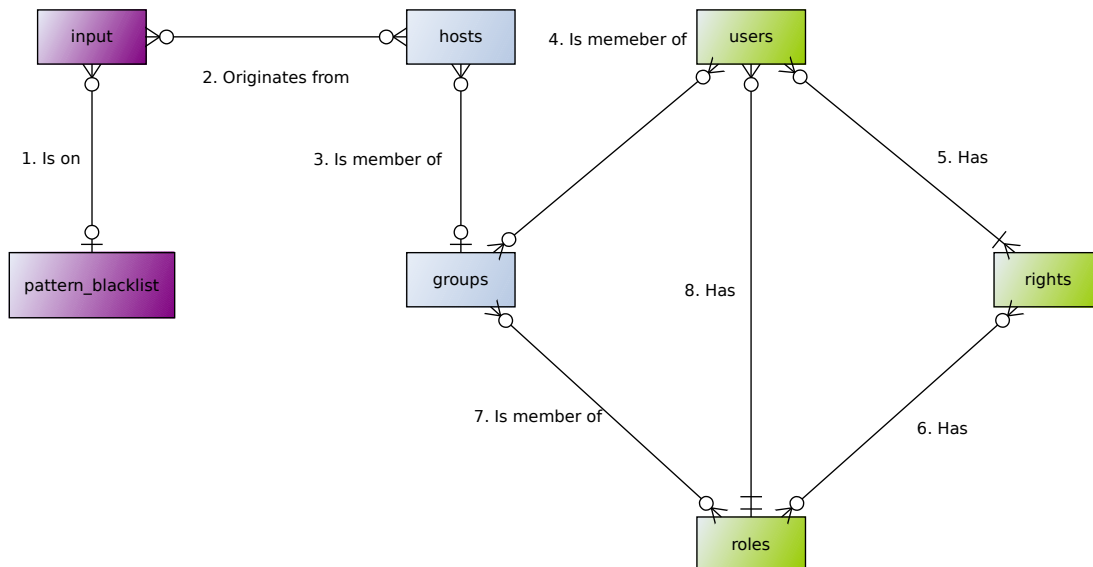
Poměrně známou metodou pro odhalení databázových entit je tzv. analýza podstatných jmen. Tato metoda zakládá na výpisu podstatných jmen z textu, či dokumentu, který obsahuje požadavky na funkce systému. Tato podstatná jména pak nejspíše tvoří dobrý základ pro databázové entity pro daný systém. Takto lze do jisté míry odhalit i vztahy mezi entitami.[14]

Po aplikaci této metody na kapitulu 4.2.1 tedy dostáváme následující klíčová slova, která jsou kandidáty na entity: Vstup, host, uživatel, pravidlo, oprávnění, role, soubor s pravidly, skupina, černá listina.

Po opětovném analyzování prvotní analýzy podstatných jmen pak pravidlo a soubor s pravidly jsou ve skutečnosti pouze odvozené od vstupu a pravděpodobně tedy nebudou entitou, ale spíše atributem, popřípadě aplikací funkce na entitu. Ve výsledném seznamu tedy zůstávají tedy následující podstatná slova pravděpodobně identifikující entity: Vstup, host, uživatel, oprávnění, role, skupina, černá listina.

4.2.3. Schéma databáze

Databázové schéma odvozené z podstatných jmen, která byla identifikována v kapitole 4.2.2 lze pak vyobrazit následovně, viz Ilustrace 4.



Ilustrace 4: Databázová architektura

4.2.4. Definice databázových entit

Následující text popisuje databázovou architekturu pomocí prostředků metodiky HIT. V první části bude popsáno, které objekty náleží do určitých kategorií, v druhé části pak budou zmíněna spojení a vztahy mezi jednotlivými entitami.[15]

Ke každé entitě je zde přiřazen slovní popis a to zejména z důvodu zjednodušení porozumění toho, jaký má entita v databázi účel.

Definice entit:

Jméno entity: pattern_blacklist

Typ: Kernel

Popis: Objekt kategorie (#pattern_blacklist) je každé parsovací pravidlo, které (#users) potvrdil(i) jako platné, či neplatné a filtrované.

Jméno entity: input

Typ: Kernel

Popis: Objekt kategorie (#input) je každý vstup, který byl nástrojem Grok identifikován jako neznámý, tedy takový, pro který doposud neexistuje parsovací pravidlo.

Jméno entity: hosts

Typ: Kernel

Popis: Objekt kategorie (#hosts) je každý počítač, který je zaregistrován do systému a který sbírá data prostřednictvím programu Grok.

Jméno entity: groups

Typ: Kernel

Popis: Objekt kategorie (#groups) je každá skupina určená ke sdružování hostů z kategorie (#hosts).

Jméno entity: users

Typ: Kernel

Popis: Objekt kategorie (#users) je každá osoba oprávněná provádět operace v modelovaném systému.

Jméno entity: rights

Typ: Kernel

Popis: Objekt kategorie (#rights) je každé oprávnění k provedení operace v modelovaném systému.

Jméno entity: roles

Typ: Kernel

Popis: Objektem kategorie (#roles) je každá pojmenovaná skupina umožňující hromadné přidělení oprávnění (#rights).

Nyní budou popsána veškerá zamýšlená spojení mezi databázovými entitami. Tímto se slovně vyjádří vztahy, jež mezi sebou tyto entity mají. Prosím mějte na paměti, že implementační řešení zvolené pro praktické vyhotovení této práce nemusí nutně vyžadovat, aby tyto vztahy byly přímo implementovány na úrovni databáze.

Následující popis také obsahuje kardinality spojení entit, číselné vyjádření kardinality následující za popisem vztahu je vždy zapsáno ve směru cíl:zdroj.

Definice spojení:

Jméno spojení: 1. Is on

Kardinalita: (#input)->(#pattern_blacklist)

Popis: Vstup (#input) je filtrován, je-li zapsán na černé listině (#pattern_blacklist).

\0,1:0,m

Jméno spojení: 2. Originates from

Kardinalita: (#input)->(#hosts)

Popis: Vstup (#input) pochází z některého hosta, který může, ale nemusí být zapsán v kategorii (#hosts).\0,m:0,n

Jméno spojení: 3. Is member of

Kardinalita: (#hosts)->(#groups)

Popis: Host zapsaný na seznamu (#hosts) může být členem některé ze skupin (#groups).\0,1:0,m

Jméno spojení: 4. Is member of

Kardinalita: (#users)->(#groups)

Popis: Uživatel zapsaný na seznamu (#users) může být členem jedné a více skupin (#groups).\0,m:0,n

Jméno spojení: 5. Has

Kardinalita: (#users)->(#rights)

Popis: Uživatel zapsaný na seznamu (#users) má přiděleno jedno, či více oprávnění (#rights).\1,m:0,n

Jméno spojení: 6. Has

Kardinalita: (#roles)->(#rights)

Popis: Uživatelská role zapsaná na seznamu (#roles) může mít přidělena nějaká oprávnění (#rights).\0,m:0,n

Jméno spojení: 7. Is member of

Kardinalita: (#roles)->(#groups)

Popis: Uživatelská role zapsaná na seznamu (#roles) může být členem některé ze skupin (#groups).\0,m:0,n

Jméno spojení: 8. Has

Kardinalita: (#users)->(#roles)

Popis: Uživatel zapsaný na seznamu (#users) musí být příslušníkem některé role (#roles) v systému. \1,1:0,m

Nyní tedy práce dospěla do stavu, kdy je popsáno, jakým způsobem bude nástroj fungovat a jaká bude struktura databáze uchovávající důležitá data. Přesuňme se proto k samotným programovým prostředkům použitým k vytvoření nástroje a ke struktuře samotné aplikace, která byla pro účely této práce vytvořena. Tomuto tématu se věnuje příští kapitola nazvaná „Programové prostředky a důležité části nástrojů groracle a groracleServer“.

5. Programové prostředky a důležité části nástrojů groracle a groracleServer

V této kapitole budou přiblíženy programové prostředky využitě při tvorbě nástroje pro automatický návrh parsovacích pravidel. Dle pravidla neprogramuj, co již někdo naprogramoval, tento projekt využívá frameworky a nástroje, které ve značné míře zjednodušují naprogramování výsledné aplikace.

Další obsah této kapitoly rozebírá jednotlivé části výsledného nástroje a programové prostředky využitě při jejich vytvoření. Všechny vytvořené části budou rozebrány hierarchicky a to zdola nahoru.

5.1. GrokWrap

GrokWrap je jednoduchý skript, který ve své podstatě pouze obaluje Grok, startuje jej jako daemona linuxového systému za použití konfiguračního souboru. Podoba tohoto skriptu je následující.

```
#!/bin/bash
echo "Killing netcat if it already exists"
pkill -f "nc -l 999"
echo "Starting grok using growrap configuration"
grok -d -f /etc/groracle/grokwrap-config.grok
```

Důležitým prvkem tohoto nástroje je však samotná konfigurace nástroje Grok. Právě v této konfiguraci je upraveno, jakým způsobem jsou filtrována a zasílána úspěšně rozparsovaná data. Podoba této konfigurace tak, jak byla testována během této práce je následující (jedná se o obsah souboru /etc/groracle/grokwrap-config.grok):

```
program {
  debug: false
  load-patterns: "/etc/groracle/rules"
  #rules"
  # By default we use netcat to listen on TCP port 999. Note
  # that if the port is already in use,
  # the wrapper would not work correctly.
  exec "nc -l 999"
  match {
    # All patterns are used for matching, when some pattern
    # matches, the output of grok is redirected
    # from grok to a TCP port as JSON object.
    pattern: ".*"
    reaction: "json: { matched: \"%{@MATCH|jsonencode}\"}"
    shell: "/usr/bin/nc localhost 9997"
    break-if-match: no
  }
  no-match {
    # When no match is detected, the input line is redirected
```

```

# to another TCP port for further processing by groracle
reaction: "%{@LINE|shellescape}"
shell: "/usr/bin/nc localhost 9998"
}
}

```

Takováto konfigurace načte soubor s parsovacími pravidly dostupný v cestě /etc/groracle/rules. Pro implementaci dříve zmiňovaného filtru byl zvolen standardní příkaz linuxových systémů, který se jmenuje netcat (zkratkou nc). Ten příkaz umožňuje zaslat text prostřednictvím TCP, UDP protokolů, stejně tak umožňuje předat výstup prostřednictvím Unix Domain Socketu, který je ideální pro lokální nasazení nástroje na jeden jediný stroj.

GroracleServer v současné době umožňuje sbírání dat na TCP portu, nebo na Unix Domain socketu. Na jakém socketu si budeme data předávat je tedy otázkou konfigurace nástroje grokwrap (specificky větev no-match položka shell) a nástroje groracleServer, kterému se věnuje podkapitola 5.2.

5.2. GroracleServer

Jak lze již logicky odvodit z předchozího textu, groracleServer je část projektu, která je zodpovědná za sběr dat a to buď z TCP portu, nebo z Unix Doménového socketu (ve variantě bezstavového tzv. STREAM socketu). Tato jednoduchá aplikace napsaná v jazyce Java sbírá data z programu Grok. Tato data jsou po té zpracována tak, že je identifikován původce těchto dat (host), je zjištěno datum sběru. Dalším postupem je automatický odhad pravidla pro parsování vstupu. K tomuto účelu využíváme volání programu discogrok, který je součástí Groku. Discogrok, nebo-li Grok Discovery je nástroj, který je schopný odhadnout strukturu vstupu na základě souboru s pravidly pro grok a vstupní řádky. Takto připravená data jsou po té uložena do databáze postgres.

Veškerá konfigurace pro tento nástroj je uložena v /etc/groracle/config.xml, což je soubor ve formátu XML. Jeho vzor lze nalézt na příloženém CD nosiči ve složce config projektů groracleServer a groracle.

Poměrně důležitým návrhovým prvkem je zpracování dat příchozích buďto prostřednictvím TCP portu, nebo Unix Doménového socketu. Pro zvýšení výkonnosti celého nástroje využívá dříve uvedené zpracování vstupu metody tzv. „Stream gobblingu“. Tato metoda je všeobecně známá mezi javovými programátory pro její dobré vlastnosti. Využití metody dovoluje otevření datového toku, tento tok je pak zpracován v samostatném vlákně. Toto dovoluje čtení a zpracování z více zdrojů současně.[16] Tato metoda byla nejprve používána pro účely čtení z výstupu systémových příkazů a její využití pro účely groracleServeru tedy vyžadovalo značné změny a poměrně velký odklon od její původní podoby, neboť je třeba vzít v potaz např. blokaci TCP portu,

nebo Unix Doménového socketu. Pro komunikaci s Unixovými Doménovými sockety GroracleServer používá knihovnu pojmenovanou JUDS (Java Unix Domain Sockets), která má dvě části. První částí je javová implementace předávající datový proud a druhá část je tvořena jednoduchým programem v jazyce C, který je nutné nejdříve zkompileovat v místě nasazení. O tom se ale více zmíníme v kapitole probírající nasazení nástroje do provozu.

Pro shrnutí jsou v následujícím textu stručně uvedeny funkce vykonávané GroracleServerem. GroracleServer převezme data z TCP portu, nebo Unix Doménového socketu, v závislosti na konfiguraci, tato data zpracuje a vloží do databáze. Do databáze jsou data vkládána prostřednictvím prostého volání SQL funkce za využití JDBC postgres ovladače pro jazyk Java.

Data v databázi lze následně vizualizovat a upravovat prostřednictvím nástroje groracle, který si nyní přiblížíme.

5.3. Groracle

Groracle, jak již bylo zmíněno, je nástroj, který uživateli umožní manipulovat s uloženými daty. V případě této součásti navrhovaného nástroje již není situace tak jednoduchá. Tento nástroj musí splňovat několik požadavků, aby mohl představovat naplnění dříve uvedených cílů.

Prvním z těchto požadavků je nasaditelnost na jednotlivý stroj, ale zároveň možnost nasazení jako cloudového nástroje. Dalším požadavkem je poskytnout uživateli pokročilé funkce. Oba tyto požadavky poměrně dobře naplňuje nástroj, jež by byl napsaný v jazyce Java Enterprise Edition. Takovýto nástroj bude možné nasadit jak lokálně, prostřednictvím aplikačního serveru, tak i vzdáleně, případně do některého z komerčních cloudů, který umožňuje nasazení aplikací naprogramovaných v jazyce Java.

Pro zjednodušení programování aplikací na platformě Java Enterprise Edition již publikováno nepřeberné množství různých frameworků, při čemž každý z nich má své výhody a nevýhody. Jedním z nejznámějších je framework GWT (Google Web Toolkit). Tento framework umožňuje vytvoření uživatelského rozhraní tak, že programátor vytvoří zdrojový kód v jazyce Java, za použití speciálních objektů, při kompilaci je tento zdrojový kód přeložen do jazyků HTML a JavaScript. Tento framework má však jednu významnou nevýhodu a to je nutnost separace zdrojového kódu uživatelského rozhraní a samotného funkčního kódu. Dále je zde pak nutnost ošetřovat bezpečnostní rizika při komunikaci uživatelského rozhraní se serverovými komponentami.

Po hledání vhodného řešení které by mělo podobné vlastnosti jako GWT, nicméně nemělo jeho nevýhody, a konzultaci s vedoucím této práce, bylo rozhodnuto, že jako základní framework pro napsání nástroje groracle bude sloužit framework Vaadin.

Vaadin je objektově orientovaný framework, který využívá GWT ke kompilaci uživatelského prostředí, nicméně komponenty, které jsou zobrazeny v UI přímo interagují s kódem, který není specifický UI. Tato komunikace je prováděna prostřednictvím JSONP, při čemž každý objekt zasílaný z UI na server je automaticky šifrován RSA algoritmem. Tento framework tedy poskytuje pohodlí jazyka Java při programování uživatelského prostředí, automaticky šifruje data, která jsou exponována vnějšímu světu a nevyžaduje separaci kódu na striktně serverový zdrojový kód a kód uživatelského rozhraní. Z těchto důvodů je tedy Vaadin dokonalá volba pro účely této diplomové práce.[17]

Jednou z dalších výhod frameworku Vaadin je také kompatibilita UI generovaných tímto nástrojem s téměř všemi současnými webovými prohlížeči. Toto umožní uživatelům našeho nástroje také využívat jeho služeb na většině dnešních chytrých telefonů. V době, kdy se stále více cloudové služby dostávají i na tuto platformu komunikace člověka s počítačem, je tato výhoda také velmi podstatná.

Další prvky použité při programování groracle aplikace jsou standardní a odpovídají standardu dnešní doby. Pro práci s hesly využíváme v tomto nástroji knihovnu apache commons-codec. Tato zavedená knihovna nepodléhá dramatickým změnám v porovnání se standardní knihovnou java digest. Z tohoto důvodu se jedná o lepší variantu pro větší nasazení s pravidelnými aktualizacemi. Pro práci s databází využíváme v tomto projektu JPA (Java Persistence Api), pro zjednodušení práce s tímto schématem je v projektu využit framework Hibernate, který poskytuje základ pro objektově-relační mapování dat z databáze na objekty.

Pro parsování konfiguračního souboru ve formátu XML byla zvolena standardní knihovna XMLparsing.

Jedinou do jisté míry nestandardní knihovnou je dcharts. Tato Open Source knihovna poskytuje přidané funkce pro zobrazování statistik. V současné verzi (dcharts-widget-1.7.0) bohužel tato knihovna obsahuje kritickou chybu při přidávání položek do grafu. Tato práce však obsahuje záplatu, která tento problém přemísťuje a v rámci řešení této práce jsem se rozhodl přispět do dcharts knihovny touto opravou.

Následující výstup příkazu „tree com | grep -v 'java\|xml\|gif'“ popisuje adresářovou strukturu hlavního balíku projektu:

```
com
├── example
│   ├── goracle
│   │   ├── Brokers
│   │   ├── DAL
│   │   │   ├── DAO
│   │   ├── Entity
│   │   │   ├── Aggregated
│   │   ├── Views
│   └── widgetset
```

Jaký je tedy obsah jednotlivých složek? Základní složka projektu goracle obsahuje tzv. navigátor. Navigátor je speciální třída implementující Servlet API ve verzi 3.0 a je zodpovědná za navigaci v aplikaci a iniciální přihlášení uživatele. Tato třída také umožňuje automatickou konstrukci fragmentů URI a veškeré součásti nástroje goracle jsou tak přístupné i prostřednictvím přímého odkazu. Kromě této třídy je v základní složce také třída Menu.java, která je zodpovědná za konstrukci menu pro uživatelské prostředí, dále tato třída ověřuje, zda má uživatel právo přistupovat k jednotlivým částem systému.

V podbalíku nazvaném „Brokers“ jsou uloženy třídy, které slouží pro provedení složitějších funkcí, nebo kooperující s operačním systémem, jako je například nahrávání souborů na vzdálené hosty, čtení konfigurací z XML souboru, pokročilá správa hesel, generování statistik, atd.

Další v pořadí je podbalík DAL. Tento obsahuje speciální třídu implementující komunikaci Hibernate s databází. V jeho podsložce DAO se pak nacházejí třídy reprezentující objektovou abstrakci operací prováděných s databází. V DAO vrstvě z důvodu lepšího škálování aplikace používáme speciální mechanismus, který umožňuje transakční přístup k databázovým operacím a dále umožňuje přímou správu spojení, které Hibernate vytváří za účelem přístupu k databázi. Jedná se o přímou implementaci jednoho z doporučení uvedeného inženýry společnosti Red Hat na konferenci Jboss World.[18]

Podbalík Entity obsahuje třídy, které poskytují objektovou reprezentaci dat v databázi. Zde je využito poměrně nové schéma reprezentace objektů pomocí Hibernate, které je založeno na anotacích jednotlivých proměnných daného objektu. Tímto se vyvarujeme potřeby správy a úprav různých XML souborů, které se dříve využívali pro definici objektově-relačního mapování. V podsložce „Aggregated“ lze pak nalézt třídy reprezentující takové objekty, které nejsou

přímou reprezentací dat v některé z databázových tabulek, ale vznikají jako výsledek databázových agregací.

Jakým způsobem tedy mapování objektů v tomto projektu vypadá? Jednoduchý popis je vidět na následujícím jednoduchém názorném příkladu. Níže uvedený výpis popisuje atributy databázové tabulky groups:

```
groracle=> \d groups
          Tabulka "public.groups"
-----+-----+-----
  Sloupec | Typ | Modifikátory
-----+-----+-----
  groupid | character varying(20) | not null
  description | text |
```

Následující úryvek zdrojového kódu popisuje jak vypadá reprezentace položky uložené v této tabulce pomocí anotované třídy, která využívá technologie Hibernate. Prosím mějte na paměti, že z následujícího ukázkového kódu záměrně vypouštíme komentáře a klauzule importující jiné třídy:

```
@Entity
@Table(name="groups")
public class Group {
    public Group(){}
    @Id
    @Column(name="groupid")
    String groupid;

    @Column(name="description")
    String description;
    public String getGroupid(){
        return this.groupid;
    }
    public String getDescription(){
        return this.description;
    }
    public void setGroupid(String groupid){
        this.groupid = groupid;
    }
    public void setDescription(String descriptpion){
        this.description = descriptpion;
    }
}
```

Základním znakem anotovaného stylu práce s Hibernate je vytvoření globálních proměnných, které odpovídají sloupcům v databázových tabulkách a další nutností je pak vytvoření metod, které dovolí z takto vytvořeného objektu data vyjímat a také je do něj vkládat, nebo upravovat.

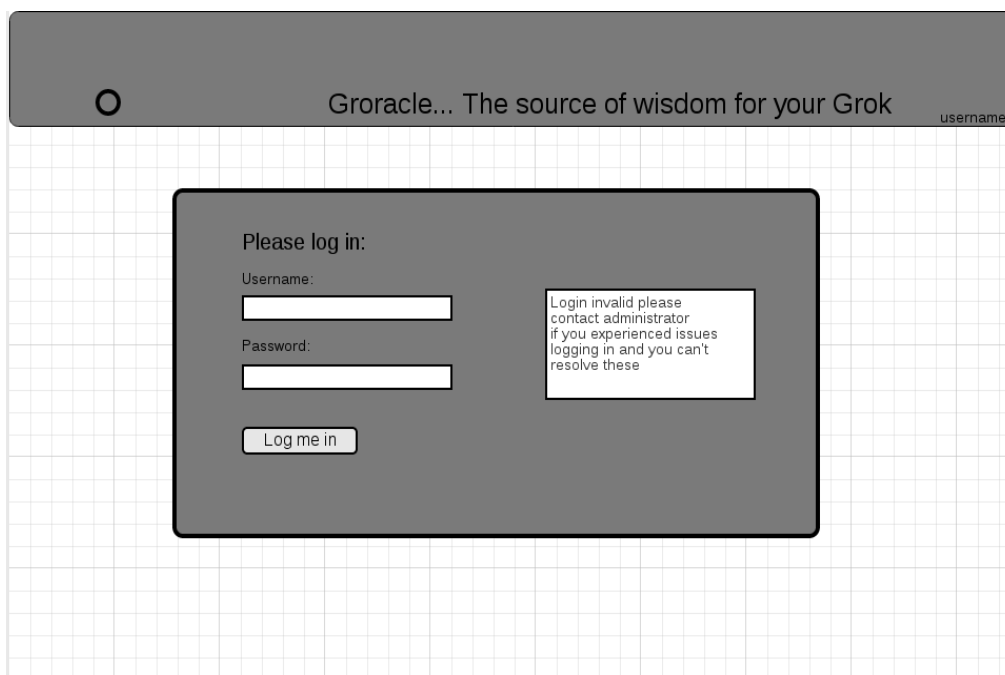
Poslední podbalík je formovaný složkou „Views“. Tato složka obsahuje samotné pohledy utvářející uživatelské rozhraní nástroje. Třídy v této složce jsou tedy zodpovědné za zobrazení dat, poskytnutí funkcí pro změnu dat a samozřejmě také přímou validaci některých dat zadávaných uživatelem.

Úplně poslední složkou je složka widgetset. Tato však obsahuje pouze zkompi-
lovaný obsah, který slouží pro vykreslování statistik prostřednictvím knihovny dCharts. Tato knihovna vyžaduje, aby složka widgetset byla obsažena v základní složce projektu

Samotné tvorbě těchto pohledů předcházela jednoduchý návrh prostřednictvím online nástroje Moqups, který je dostupný na adrese <https://moqups.com>. Ilustrace číslo 5 je ukázkou návrhu přihlašovacího dialogu. Ilustrace číslo 6 pak zobrazuje původní návrh dialogu pro odstranění parsovacího pravidla z černé listiny.

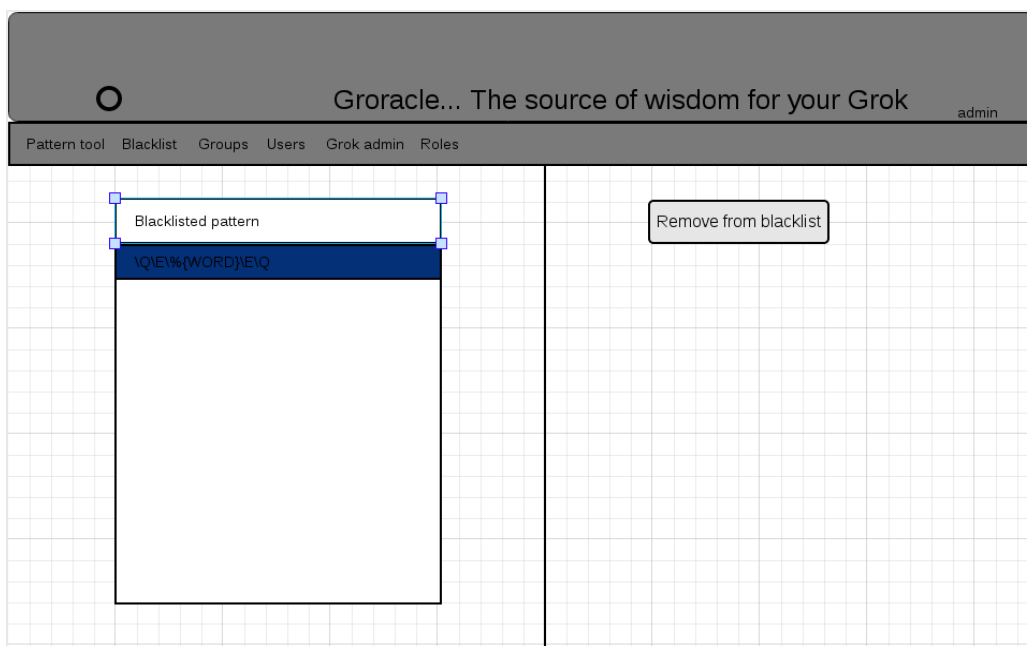
Během samotného vývoje nástroje však došlo k určitým změnám, zejména v reprezentaci dat tak, jak jsou zobrazena uživateli. Ne všechny původní návrhy jsou tedy implementovány v takové formě, v jaké byly navrženy.

V případě návrhu uživatelského rozhraní je nutné podotknout, že bohužel tato disciplína nepatří mezi autorovy silné stránky a po domluvě s vedoucím této práce byl kladen hlavní důraz spíše na architekturu funkcí a funkčního kódu, který je zodpovědný za modifikaci dat, než na přímou podobu nástroje.



Ilustrace 5: Návrh přihlašovacího dialogu – neúspěšné přihlášení

Uživatelské rozhraní tak, jak je tvořeno programem Vaadin vyžaduje nejenom definici objektu, ale jednotlivé elementy uživatelského rozhraní odvozují svoji pozici a styl z definice kaskádového stylu (CSS), který je buď přímo definován uživatelem, nebo je odvozen ze základních stylů, jež Vaadin doručuje jako součást knihoven tohoto frameworku. Pro účely našeho nástroje bylo zvoleno základní odvození ze stylu Vaadin Reindeer a definice některých elementů byly následně upraveny prostřednictvím tzv. Zdrojových souborů kaskádového stylu, jež jsou umístěny v souboru `groracle/WebContent/VAADIN/themes/groracle/groracle.scss`.

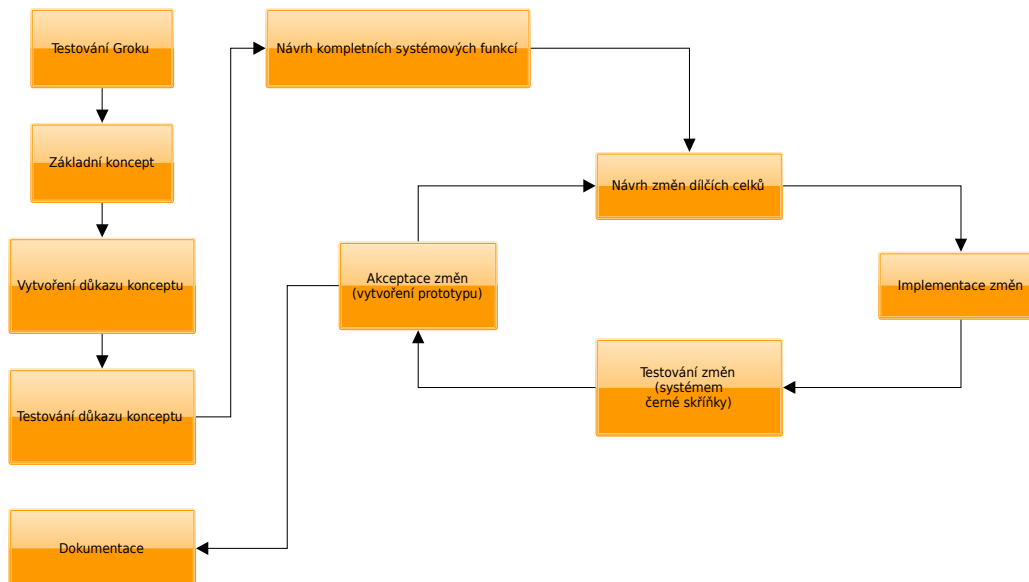


Ilustrace 6: Návrh dialogu pro odstranění pravidla z černé listiny

Tímto byly tedy probrány základní rysy navrženého nástroje a programové prostředky použité při jejich realizaci. V následující podkapitole je popsáno, jaký vývojový cyklus byl pro vývoj tohoto nástroje zvolen.

5.4. Vývojový cyklus nástroje

S ohledem na celkový objem času, po který byla tato práce vyvíjena a který činil méně, než 6 měsíců, bylo nutné uzpůsobit projektový cyklus a způsob předávání výsledků tomuto relativně krátkému období. Kompletní vývojový cyklus je popsán následujícím schématem.



Ilustrace 7: Vývojový cyklus

Jak lze z tohoto schématu vyčíst, jedná se o upravenou verzi vývojového cyklu založeného na prototypování. Změny vyplývají předně z počátečního testování škálovatelnosti Groku a určité míry nejistoty, zda bude Grok vůbec možné pro účely vytvoření nástrojů, prezentovaných touto prací, použít.

Během doby trvání celého projektu byl vytvořen prvotní prototyp, který dokazoval použitelnost groku. Ten se sestával v podstatě z nástroje grokwrap, groracle server ve variantě pouze TCP serveru a jednoduchého zobrazení ve webovém rozhraní. Následující prototypy již přidávaly některé významné funkce jako například autentizaci a autorizaci uživatel, implementací rolí a delegace oprávnění na bázi rolí, implementaci generování optimalizovaných souborů s pravidly, nahrávání souborů, generování statistik. Poslední prototyp pak obsahoval změnu systému práce s databází z JDBC implementace na objektově-relační mapování databáze pomocí Hibernate.

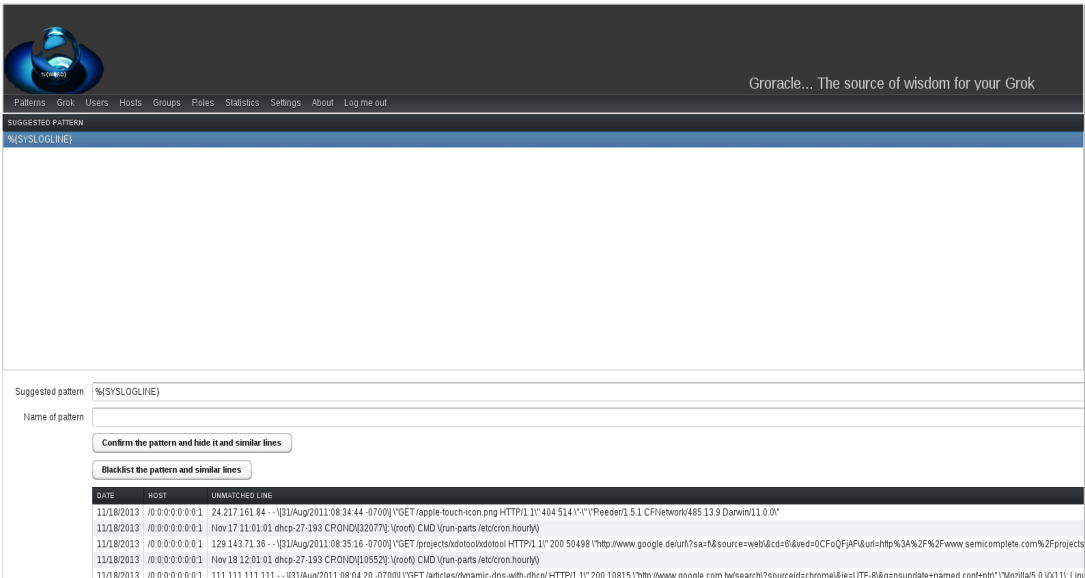
6. Přehled funkcí navrženého systému

V této kapitole je popsáno, jak nástroj vytvořený pro účely této práce využívat. Jsou zde uvedeny ukázkové obrazovky tohoto systému a na praktickém příkladu jsou popsány základní funkce systému.

Nejprve tedy zpět k zadání této práce. Účelem této práce bylo vytvořit nástroj, který by automaticky odhadoval parsovací pravidla pro neznámý vstup. Dříve již bylo ukázáno, jakým způsobem k účelu automatického odhadu používáme nástroj discogrok a poměrně detailně jsme probrali, jak jsou tato data ukládána do databáze. Nyní si popíšeme, jak lze s těmito daty pracovat prostřednictvím webového rozhraní a jaké další funkce nám toto prostředí nabídne.

Po přihlášení uživatele je prvním zobrazeným pohledem list doposud neod-souhlasený, či nefiltrovaných návrhů. Po výběru jedné položky se zobrazí de-taily, mezi kterými lze vidět IP adresu hosta, ze kterého tyto návrhy pochází, datum vložení a samotnou položku, které návrh odpovídá. Ukázkou tohoto pohledu je ilustrace číslo 8. Každý návrh lze manuálně upravit, pojmenovat a buď potvrdit jako platné pravidlo, nebo jej zahrnout na černou listinu pravidel, které chceme filtrovat.

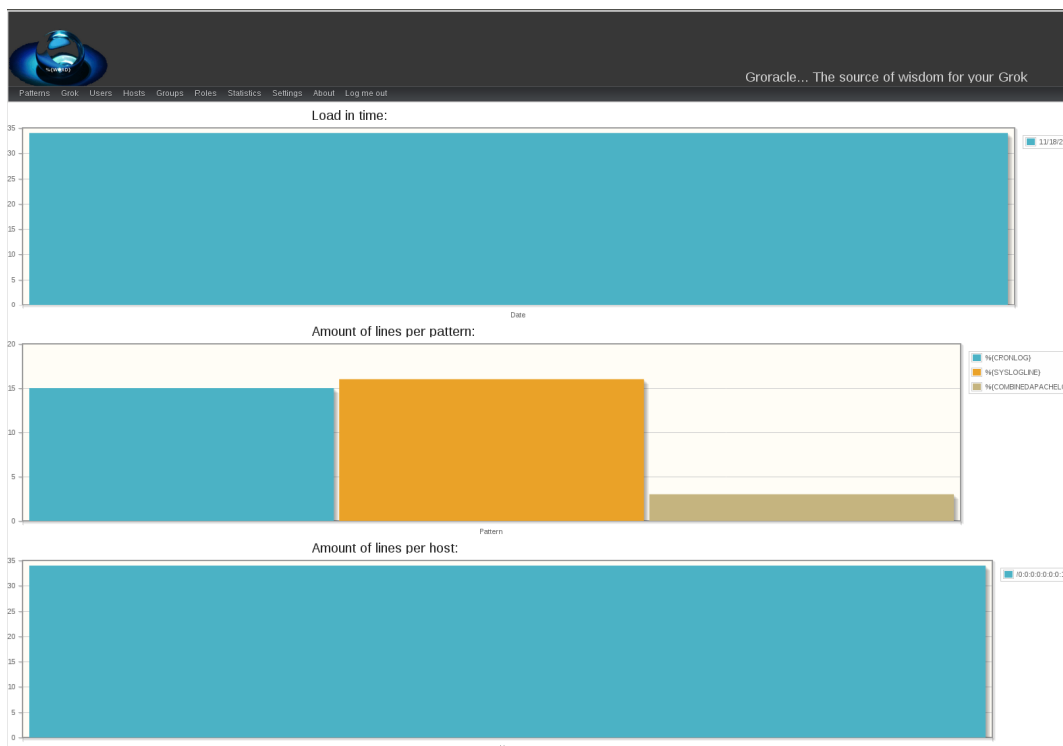
Jak pravidla zahrnutá na černé listině, tak pravidla potvrzená uživatelem jako platná je možné přidat k aktuálnímu souboru pro parsování/filtrování dat. Tyto soubory lze vygenerovat prostřednictvím funkcí přístupných pod



The screenshot shows the Groacle web interface. At the top, there is a navigation menu with links: Patterns, Grok, Users, Hosts, Groups, Roles, Statistics, Settings, About, and Log me out. The main content area displays a 'SUGGESTED PATTERN' section with the text '%(SYSLOGLINE)'. Below this, there is a form with a 'Suggested pattern' field containing '%(SYSLOGLINE)', a 'Name of pattern' field, and two buttons: 'Confirm the pattern and hide it and similar lines' and 'Blacklist the pattern and similar lines'. At the bottom, there is a table with the following data:

DATE	HOST	UNMATCHED LINE
11/18/2013	/0.0.0.0:0.0.1	24.217.161.84 - [31/Aug/2011:08:24:44 -0700] "GET /apple-touch-icon.png HTTP/1.1" 404 514 "-" Firefox/1.5.1 CFNetwork/485.13.9 Darwin/11.0.0
11/18/2013	/0.0.0.0:0.0.1	Nov 17 11:01:01 dhcp-27-193 CROKID[22077] [root] CMD [run-parts /etc/cron.hourly]
11/18/2013	/0.0.0.0:0.0.1	129.142.71.38 - [31/Aug/2011:08:25:16 -0700] "GET /projects/ideotoolstool HTTP/1.1" 200 50498 http://www.google.de/url?sa=i&source=web&cd=6&ved=0CFoQFjAFI&url=http%3A%2F%2Fwww.semicomplete.com%2Fprojects
11/18/2013	/0.0.0.0:0.0.1	Nov 18 12:01:01 dhcp-27-193 CROKID[28552] [root] CMD [run-parts /etc/cron.hourly]
11/18/2013	/0.0.0.0:0.0.1	111.111.111.111 - [31/Aug/2011:08:04:20 -0700] "GET /articles/dynamic-dns-with-dhcp/ HTTP/1.1" 200 10815 http://www.google.com/websearch?sourceid=chrome&ie=UTF-8&q=nsupdate+named.conf+plm%20Mozilla5.0%20Linux

Ilustrace 8: Ukázka základního pohledu



Ilustrace 10: Ukázka dialogu - Statistiky

Statistiky jsou další funkcí poskytovanou systémem. Nyní jsou dostupné pouze statistiky pro zobrazení zátěže denně, neboli počet řádků, jež je do systému uložen denně, počtu uložených řádků, které odpovídají stejnému návrhu pravidla a počet řádků pocházejících z určitého hostovského systému.

7. Zhodnocení práce

V první části této práce byla představena problematika, kterou se tato diplomová práce zabývá, kterou následoval přehled dostupných komerčních a nekomerčních řešení v dané oblasti. Práce představila výhody a nevýhody jednotlivých produktů a pokračovala návrhem a popisem řešení, které bylo pro účely této diplomové práce a zadání laboratoře LaSArIS vytvořeno.

Jednotlivé kapitoly této práce popisovaly, jakým způsobem byla navržena architektura nástroje tak, aby jej bylo možné nasadit na jeden jediný stroj, ale zároveň i například do některého z komerčních cloudů. V této práci bylo popsáno i jakým způsobem byla navržena databáze pro nástroj a jakým způsobem jsou sbírána a ukládána data pro účely zpracování nástrojem gromacle.

Na ukázky poskytly i vzhled na způsob, jakým bylo navrženo uživatelské rozhraní, jakým způsobem bylo naprogramováno a předně jakých prostředků bylo k naprogramování uživatelského rozhraní využito.

V předposlední kapitole byly popsány některé funkce systému a jakým způsobem je uživatelům umožněn přístup k těmto funkcím.

Kromě textu této diplomové práce, který právě čtete, bylo naprogramováno kompletní řešení daného problému o celkovém rozsahu větším, než 8000 řádků zdrojového kódu včetně kompletní dokumentace prostřednictvím technologie JavaDoc a přídatných komentářů v důležitých bodech zdrojového kódu. Dále jsou s prací přiloženy ukázkové konfigurace vytvořených nástrojů, schémata a návrhy uživatelského rozhraní a schémata referenčních architektur zobrazujících možné nasazení nástroje.

Tento nástroj byl uvolněn pod jednou z „Open Source“ licencí. Jak laboratoř LaSArIS, tak libovolný návštěvník mají tedy možnost do tohoto projektu přispívat a využívat vytvořeného nástroje pro své potřeby, jedná se tedy o řešení, jehož hranice nekončí s odevzdáním této práce, ale autor této práce zamýšlí do projektu samozřejmě nadále přispívat a to jak případnými záplatami, tak i rozvojem nových funkcí.

8. Použité zdroje

8.1. Citace

- [1] Papertrail. *Frustration-free log management* [online]. Dostupné na World Wide Web: <https://papertrailapp.com>, cit. 2.12.2013.
- [2] Loggly. *Solve operational problems faster* [online]. Dostupné na World Wide Web: <http://www.loggly.com>, cit. 2.12.2013.
- [3] Logscape. *Search without limits* [online]. Dostupné na World Wide Web: <http://logscape.com>, cit. 2.12.2013.
- [4] BITSTEER. *What is Otus siem?* [online]. Dostupné na World Wide Web: http://www.bitsteer.com/otus_about.html, cit. 2.12.2013.
- [5] XPLG. *The next big thing in big data* [online]. Dostupné na World Wide Web: <http://www.xpolog.com>, cit. 2.12.2013.
- [6] TIBCO. *Log management for the enterprise* [online]. Dostupné na World Wide Web: <http://www.tibco.com/products/event-processing/log-management/default.jsp>, cit. 2.12.2013.
- [7] Logentries. *Analyzing billions of log events every day* [online]. Dostupné na World Wide Web: <https://logentries.com>, cit. 2.12.2013.
- [8] Splunk. *Listen to your data* [online]. Dostupné na World Wide Web: <http://www.splunk.com>, cit. 2.12.2013.
- [9] Logstash. *Logstash* [online]. Dostupné na World Wide Web: <http://logstash.net>, cit. 2.12.2013.
- [10] Logstash. *Running logstash as a windows service* [online]. Dostupné na World Wide Web: <http://cookbook.logstash.net/recipes/windows-service>, cit. 5.12.2013.
- [11] Torch. *Graylog 2 is for data analysis* [online]. Dostupné na World Wide Web: <http://www.graylog2.org>, cit. 2.12.2013.
- [12] Jordan Sissel (Semicomplete). *Introduction to Grok* [online]. Dostupné na World Wide Web: <https://code.google.com/p/semicomplete/wiki/Grok>, cit. 3.12.2013.
- [13] Postgres. *Replication, clustering, and Connection Pooling* [online]. Dostupné na World Wide Web: http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling, cit. 7.12.2013.

- [14] R.C.M. Consulting. *Analysis by Noun Lists and Use Cases: A Case Study* [online]. Dostupné na World Wide Web: <http://www.objectmentor.com/resources/articles/casestud.pdf>, cit. 3.12.2013.
- [15] Winkler, Marek; Oškera, Michal a Staníček, Zdenko. *Conceptual modeling using the HIT method* [online]. Dostupné na World Wide Web: http://is.muni.cz/www/39391/HIT_tutorial.pdf, cit. 4.12.2013.
- [16] Daconta, Michael C. (Java world). *Navigate yourself around pitfalls related to the Runtime.exec() method* [online]. Dostupné na World Wide Web: <http://www.javaworld.com/jw-12-2000/jw-1229-traps.html>, cit. 4.12.2013.
- [17] Vaadin. *Comparison* [online]. Dostupné na World Wide Web: <https://vaadin.com/comparison>, cit. 5.12.2013.
- [18] Momjian, Bruce a Mlodgenski, Jim. *Scaling Hibernate Applications with Postgres* [online]. Dostupné na World Wide Web: http://www.redhat.com/f/pdf/jbw/jmlodgenski_940_scaling_hibernate.pdf, cit. 5.12.2013.

8.2. Seznam obrázků

Zevrubná architektura požadovaného nástroje.....	20
Jednoduchá architektura.....	21
Příklad škálování navrhovaného nástroje.....	21
Databázová architektura.....	24
Návrh přihlašovacího dialogu – neúspěšné přihlášení.....	34
Návrh dialogu pro odstranění pravidla z černé listiny.....	35
Vývojový cyklus.....	36
Ukázka základního pohledu.....	37
Ukázka dialogu - Soubor s parsovacími pravidly, upload.....	38
Ukázka dialogu - Statistiky.....	39

8.3. Seznam použitých zkratk a výrazů

Aplikační server	Speciální framework poskytující abstrakci mezi vrstvou operačního systému a samotné aplikace. Tento server umožňuje spuštění samotných aplikací.
Cloudové výpočty	Angl. Cloud computing. Architektura budování výpočetních systémů na základě modelu podobném internetu. Jednotlivé služby jsou dostupné pomocí webového prohlížeče, popřípadě klientské aplikace, přičemž výpočetní systém lze dynamicky rozšiřovat o nové výpočetní prostředky.
CSS	Cascading Style Sheets. Česky kaskádový styl je soubor pravidel, která upravují různé znaky uživatelského rozhraní založeného na HTML. Jedná se například o pozici objektu, jeho barvu, atd.
DAL	Database Access Layer. Programová vrstva pro přístup k databázi.
DAO	Data Access Object. Speciální objekt, který je abstraktní reprezentací databázových operací s určitými objekty.
Framework	Anglický výraz pro knihovny funkcí, či podprogramy, které poskytují programátorovi pokročilé funkce a datové struktury.
GB	Gigabajt. Násobek jednotky velikosti paměti (1024 MB).
Groracle	Nástroj pro vizualizaci dat uživateli povolující pokročilé operace v navržené infrastruktuře nástroje pro generování pravidel.
GroracleServer	Nástroj pro sběr a zpracování dat a jejich uložení do data báze.
GWT	Google Web Toolkit. Framework pro zjednodušení programování uživatelských rozhraní aplikací na bázi Java Enterprise Edition.
Hibernate	Framework pro objektově-relační mapování databázových dat na objekty.
HTML	HyperText Markup Language. Značkovací jazyk pro tvorbu webových aplikací.
HTTP	Hyper Text Transfer Protocol. Protokol pro přenos dat v internetu.

Java	Objektově orientovaný programovací jazyk.
JPA	Java Persistence Api. Specifikace pro práci s relačními daty.
JDBC	Java Database Connectivity. Architektura pro přístup programů, napsaných v jazyce Java, k databázím.
JSON	JavaScript Object Notation. Formát zápisu objektu a jeho vlastností do prostého textového souboru.
JSONP	JavaScript Object Notation with Padding. Komunikační technika umožňující komunikaci UI komponent se servery pomocí JSON objektů.
MB	Megabajt. Běžně užívaná jednotka velikosti paměti.
Open Source	Souhrnné označení pro software vyvíjený v rámci licence poskytující volný přístup k tomuto software a jeho zdrojovému kódu.
Parsování	Proces přeměny vstupních dat na data výstupní za využití předem definovaných pravidel kladených na výstup.
Pattern	Angl. ekvivalent pro vzor.
REST	Representational State Transfer. Styl programování založený na přístupu k datům prostřednictvím HTTP protokolu a URI odkazů. Umožňuje přístup k datům, prostřednictvím odpovědi na HTTP dotaz.
Rsyslog	Remote System Logging. Protokol používaný v linuxových systémech pro přenos logovacích záznamů z jednoho počítače na počítač jiný, který data ukládá, či zpracovává.
SNMP	Simple Network Management Protocol. Protokol pro správu a monitorování výpočetních prostředků.
Stream Gobbling	Metoda přenesení zpracování datového doku uvnitř samotného vlákna
UI	User Interface. Označení pro rozhraní aplikace se kterým komunikuje uživatel.
URI	Uniform Resource Identifiers. Řetězec tvořící odkaz na webový zdroj. Specifikace je dostupná na: http://www.ietf.org/rfc/rfc3986.txt

Velká data

Souhrnné označení pro data produkovaná dnešními IT systémy. Nejedná se pouze o data o samotných systémech, ale například data a statistiky sebrané ze sociálních sítí, reklamních kampaní, atd.

9. Přílohy

Hlavní přílohou této práce je CD nosič obsahující tuto práci a dále i zdrojový kód veškerých součástí nástroje pro účely této práce vytvořeného. Tento CD-ROM také obsahuje zdrojový kód pro testování nástroje Grok a ukázková data použitá při testování.

9.1. Kompilace a nasazení nástroje

Následující text popisuje, jakým způsobem nástroj vytvořený pro tuto práci zkompilovat a nasadit. Postupně tato kapitola popíše, jakým způsobem nasadit grokwrap, jak zkompilovat a spustit groacleServer a na závěr bude ukázáno, jak zkompilovat, nasadit a spustit samotné uživatelské rozhraní nástroje groacle.

Před samotným popisem kompilace a nasazení je třeba nainstalovat nainstalovat položky z následujícího seznamu. Veškeré součásti projektu, které jsou naprogramované v jazyce Java lze zkompilovat prostřednictvím nástroje pro správu závislostí Maven. Pokud jej nemá uživatel nainstalovaný, je třeba tento nedostatek odstranit. Maven lze nainstalovat snadno například pomocí správce balíků yum a to příkazem „yum install maven“. V případě, že chce uživatel používat nejnovější zdrojový kód tohoto nástroje, bude ještě zapotřebí nainstalovat nástroj pro správu verzí git. Tento nástroj lze opět nainstalovat například příkazem „yum install git“. Na stroji, který je zamýšlen jako databázové úložiště je třeba nainstalovat a spustit databázový server postgres a klientskou aplikaci postgresql. Toto je opět možné učinit pomocí „yum install postgresql-server postgresql“. Pokud bude uživatel chtít využívat funkce spojené s nahráváním souborů s parsovacími pravidly, pak je třeba na veškeré hosty instalovat openssh, openssh-server a na stroj, kde bude nasazeno UI je třeba nainstalovat openssh a openssh-clients. Na strojích, kde poběží aplikace naprogramované v jazyce Java je také třeba nainstalovat balík java-1.7.0-openjdk-devel

Pokud by uživatel chtěl instalovat vše zároveň pak lze využít následující příkaz:

```
yum install git maven openssh openssh-clients openssh-server postgresql postgresql-server java-1.7.0-openjdk-devel
```

9.1.1. Kde vzít zdrojový kód?

Zdrojový kód veškerých součástí nástroje lze nalézt buďto na přiloženém CD nosiči, v informačním systému Masarykovy univerzity, kde je uložena i tato práce, nebo na serveru github jako repozitáře pro verzovací nástroj. Následující příkazy nástroje git zkopírují repozitář ze serveru github na lokální disk.

Grokwrap: `git clone https://github.com/tdosek/grokwrap.git`

GroracleServer: `git clone https://github.com/tdosek/groracleServer.git`

Groracle: `git clone https://github.com/tdosek/groracle.git`

Pro stažení těchto repozitářů může být zapotřebí registrace na serveru github.

9.2. Vytvoření databáze postgres pro nástroj

Pokud uživatel spouští poprvé databázový server, je třeba jej inicializovat následujícím příkazem:

postgresql-setup initdb

Dále je také vhodné databázový server startovat přímo po startu operačního systému, což lze nastavit příkazem:

chkconfig postgresql on

Pro nástroj groracle je třeba nejprve vytvořit databázi v databázovém serveru postgres. Pro ideální zabezpečení databáze je důležité vytvořit speciálního uživatele, který bude tuto databázovou instanci vlastnit. Toto lze vykonat pomocí následujícího příkazu:

adduser groracle

Nyní, když máme uživatele groracle, je třeba pro něj vytvořit heslo:

passwd groracle

Po té lze přistoupit k vytvoření databázového uživatele a databáze pro nástroj. Nejdříve je nutné spustit konzoli pro databázi postgres. Spuštění je standardně přístupné z příkazové řádky uživatele root, který se přepne na uživatele postgres, tedy následujícím způsobem:

root# su - postgres

bash\$ psql -U postgres

Nyní přistoupíme k vytvoření uživatele:

postgres=# CREATE USER groracle WITH PASSWORD '123456';

Mějte na paměti, že 123456 by bylo vhodné nahradit jiným, silnějším heslem.

Dalším krokem je vytvoření databáze a přidělením práv na databázi uživateli `goracle`:

```
postgres# CREATE DATABASE goracle;
```

```
postgres# GRANT ALL PRIVILEGES ON DATABASE goracle to goracle ;
```

Nyní je potřeba databázovou strukturu inicializovat pro použití nástrojem `goracle`. Jak repozitář nástroje `goracle`, tak repozitář nástroje `goracleServer` obsahují složku `dbscript`, která obsahuje soubor `goracle.sql` s příkazy pro inicializaci databáze.

Pro spuštění tohoto skriptu je třeba, aby uživatel přešel opět do příkazové řádky uživatele `root` a spustil následující příkaz:

```
root# psql -U postgres goracle -f /cesta/k/souboru/goracle.sql
```

Tímto je databáze inicializována a připravena k dalšímu použití.

Veškeré součásti nástroje vyvíjeného v této práci používají k připojení k databázi autentifikaci heslem. Pokud tedy tato není nastavena, je třeba ještě změnit soubor `/var/lib/pgsql/data/pg_hba.conf` tak, aby toto umožňoval. Příkladem správně nastaveného záznamu tohoto souboru je následující řádek:

```
host all all 127.0.0.1/32 password
```

Pokud tato změna bude provedena je třeba ještě restartovat databázový server příkazem:

```
service postgresql restart
```

Pokud bude nástroj `goracle` nasazen na jiném stroji, než samotná databáze, pak je ještě nutné nastavit přístup k databázi v souborech `postgresql.conf` a `pg_hba.conf`. Kompletní návod, jak tuto operaci provést lze najít na různých internetových stránkách. V anglickém jazyce je tento návod dostupný například na adrese: <http://www.cyberciti.biz/tips/postgres-allow-remote-access-tcp-connection.html>

9.3. Instalace Groku

V samotné diplomové práci je popsán nástroj `Grok`, jako pomyslný středobod této práce. Nejinak je tomu při nasazení nástroje. `Grok` musí být přítomen na stroji, kde je nasazen `grokwrap` a `goracle Server`.

Bohužel z důvodu licenčního ujednání tohoto nástroje jej nelze svobodně distribuovat a proto jej nelze nalézt na přiloženém CD nosiči.

Tento nástroj je třeba stáhnout z gitového repozitáře na následující adrese: <https://github.com/jordansissel/grok.git>.

Dále je nutné nainstalovat závislosti pro zkompileování tohoto nástroje jmenovitě se jedná o:

Závislost:	Verze:
bison	>= 2.3
gnu flex	>= 2.5.35
gperf	>= 3.0
GNU make	>= 3.81
libevent	>= 1.3
libpcre	>= 7.6
Tokyo Cabinet	>= 1.4.9
Cunit	>= 2.1

Po instalaci těchto závislostí je ještě nutné zkontrolovat, že je na stroji přítomen nástroj gcc a cpp pro kompilaci zdrojového kódu v jazyce C.

Veškeré závislosti zmíněné výše včetně kompilátorů lze nainstalovat příkazem:

```
yum install bison gperf make libevent libevent-devel CUnit tokyocabinet  
tokyocabinet-devel pcre pcre-devel cpp gcc
```

Dále je již nutné pouze přejít do složky kopie repozitáře a spustit příkazy „make“ a „make install“

9.4. Nasazení grokwrap

Nástroj grokwrap je jednoduchý skript napsaný v jazyce Bash, jeho jedinou závislostí je tedy přítomnost programu Grok (jak jej nainstalovat jsme si představili v kapitole 9.3).

Pro ideální funkčnost je však třeba opět vytvořit uživatele groracle podobným postupem, jako při inicializaci databáze, tedy příkazy:

```
adduser groracle
```

```
passwd groracle
```

Jakmile je toto hotovo, z příkazové řádky uživatele root vytvoříme složku /etc/groracle a přiřadíme jí oprávnění pro uživatele groracle:

```
root# mkdir /etc/groracle
```

```
root# chown -R groracle /etc/groracle
```

```
root# chmod -R 775 /etc/groracle
```

Po té do této složky zkopírujeme soubory grokwrap-config.grok a rules:

```
root# cp grokwrap-config.grok /etc/groracle/
```

```
root# cp rules /etc/groracle/
```

Soubor grokwrap-config.grok obsahuje konfiguraci se kterou bude grok spuštěn, standardní ukázková konfigurace spustí grok tak, že tento nástroj bude naslouchat na TCP portu 999, úspěšně rozparsovaný vstup je poslán na TCP port 9997 ve formátu řádky s obalením speciálních znaků, nerozpoznaný vstup je pak přeměřován na port 9998, kde standardně naslouchá groracle-Server.

Toto chování lze změnit v relevantních klauzulích této konfigurace. Pro vstup se jedná o klauzuli „exec“, pro výstupy pak klauzule „shell“.

Je třeba mít na paměti, že změna výstupu nerozpoznaných dat musí být reflektována i do konfigurace gracleServeru.

Pro použití Unix Doménového socketu je třeba změnit parametry příkazu „nc“ v konfiguraci na „nc -U /cesta/k/socketu“ popřípadě s využitím parametru -l, pokud chceme na socketu naslouchat.

9.5. Nasazení groracleServer

GroracleServer je jednoduchá aplikace napsaná v jazyce Java, která čte konfiguraci v lokaci /etc/groracle/config.xml. Ukázkový soubor s popisem jednotlivých parametrů je dostupný ve složce nazvané „config“.

Opět je doporučeno vytvořit uživatele groracle, který bude tento program spouštět, pokud je groracleServer nasazen mimo lokaci grokwrapu, je třeba tedy vytvořit tohoto uživatele a složku /etc/groracle stejným způsobem, jako bylo instruováno v případě grokwrapu.

Pokud chce uživatel využívat funkcí spojených s Unix Doménovými sockety, pak je třeba nejprve zkompilovat soubor v adrese groracleServer/src/com/etsy/net/UnixDomainSocket.c. Toto lze provést následujícím příkazem:

```
cd groracleServer/src/com/etsy/net/
```

```
gcc UnixDomainSocket.c
```

Po provedení tohoto kroku a nastavení nutné konfigurace prostřednictvím souboru /etc/groracle/config.xml již stačí pouze zkompilovat Javové zdrojové kódy pomocí příkazu:

```
mvn verify
```

Tento příkaz zkompiluje a otestuje sanitu balíku, tento balík se při úspěchu pak nachází ve složce target a má příponu .jar.

Před spuštěním je nezbytně nutné provést správné nastavení tohoto nástroje prostřednictvím /etc/groacle/config.xml, jedná se hlavně o nastavení přístupů k databázi a lokací souborů s parsovacími pravidly.

Spuštění tohoto balíku lze pak provést příkazem:

```
java -jar groacleServer-1.1.jar &
```

Pro účely dlouhodobého nasazení je vhodné vytvořit init skript, nebo skript pro systemd, který by tento příkaz spouštěl automaticky při startu systému. Jelikož je toto však závislé na typu operačního systému, necháváme tento krok na uživateli, neboť nelze předvídat, jaký formát uživatel vyžaduje.

9.6. Nasazení groacle

Nasazení nástroje groacle, který poskytuje uživatelské rozhraní je opět vyžaduje konfiguraci umístěnou ve složce /etc/groacle/config.xml, z této konfigurace však čte pouze položky, které říkají, kde jsou na hostech uloženy soubory s parsovacími pravidly.

Před kompilací tohoto nástroje je třeba vložit údaje pro přihlášení k databázi v souboru groacle/src/hibernate.cfg.xml.

V případě, že by byl nástroj groacle nasazen na jinou architekturu, než-li 64-bitovou platformu založenou na operačním systému linux, je třeba před samotnou kompilací programu provést kompilaci tzv. widgetset. Toto lze vykonat příkazem:

```
mvn vaadin:compile
```

Pokud je toto hotovo, lze tento nástroj zkompilovat opět po vstupu do základní složky repozitáře spuštěním příkazu:

```
mvn verify
```

Výsledkem je pak .war archiv, jež je vytvořen ve složce target. Tento archiv je nutné nasadit na webový server dle vlastního uvážení. Podmínkou spuštění však je schopnost webového serveru interpretovat třídy za použití servlet-api 3.0.

Nasazení tohoto balíku si můžeme ukázat na webovém server tomcat 7. Pro jednoduchost popisují nasazení tohoto nástroje pomocí aplikace tomcat-manager, která je dostupná ve většině linuxových distribucí.

Nejprve je třeba nainstalovat balíky webového serveru:

```
yum install tomcat tomcat-webapps tomcat-admin-webapps
```

Po té je třeba editovat soubor `/etc/tomcat/tomcat-users.xml` a přidat následující řádky:

```
<role rolename="manager-gui"/>
```

```
<user username="tomcat" password="tomcat" roles="manager-gui"/>
```

Pro nahrání souboru většího, než 50 MB (velikost zkompilevaného groracle projektu včetně závislostí je asi 100 MB) prostřednictvím tomcat manageru je třeba ještě editovat soubor `/var/lib/tomcat/webapps/manager/WEB-INF/web.xml` tak, aby obsahoval následující parametry:

```
<multipart-config>
  <!-- 500MB max -->
  <max-file-size>524288000</max-file-size>
  <max-request-size>524288000</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>
```

Nyní je třeba spustit webový server příkazem „service tomcat start“ a ve webovém prohlížeči otevřít odkaz <http://localhost:8080/manager/html>. Po úspěšném přihlášení pak lze vidět webový manažer nasazených aplikací. Pro vzdálený přístup je třeba nahradit localhost IP adresou vzdáleného stroje.

Pro nasazení zkompilevaného .war archivu již stačí se navigovat na položku pojmenovanou „WAR file to deploy“. Tomcat manager obsahuje jednoduché menu pro výběr souboru a tlačítko pro jeho nasazení.

V menu „Applications“ se pak zobrazí naše nová aplikace a její cesta „Path“ bude nastavena na `/groracle-verze-archivu/`. Aplikace je tedy přístupná pomocí odkazu: <http://ip.nebo.fqdn.stroje:8080/groracle-verze-archivu/>.

Pro první přihlášení je v databázi dostupný uživatel admin s heslem admin. Je doporučeno toto heslo co nejdříve změnit. Takto lze učinit v dialogu „Settings – Reset password“.

Dříve prezentované úkony vedou k vytvoření kompletní infrastruktury nástroje groracle.

Alternativou k výše uvedenému nasazení na tomcat je přímé nasazení pomocí tomcat pluginu pro nástroj maven. Postup jak využít tento plugin je popsán na adrese: <http://tomcat.apache.org/maven-plugin-trunk/tomcat7-maven-plugin/usage.html>.

9.7. Jakým způsobem nahlásit problém?

Pokud některá z komponent nekomunikuje tak, jak by se čekalo, pak by měl uživatel nejprve zkontrolovat nastavení veškerých systémových prostředků, jako databáze postgres, firewallu systému, zda například neblokuje porty TCP pro zasílání logů groracleServerem, či dokonce porty pro komunikaci s databází.

Pokud ani po té uživatel neuspěje ve spuštění aplikace, je možné kontaktovat autora této práce ať již prostřednictvím informačního systému Masarykovy univerzity, tak i prostřednictvím githubu.

Na následujících adresách lze také zakládat problémy, které by případně v nástrojích uživatelé našli. Pokud se k uživatel k nahlášení problému rozhodne, prosím o poskytnutí co nejvíce dat, zejména logovacích záznamů z webových serverů, chybové výpisy, atd.

Adresy pro zakládání nových problémů:

<https://github.com/tdosek/groracle/issues>

<https://github.com/tdosek/groracleServer/issues>

<https://github.com/tdosek/grokwrap/issues>

Doufám však, že tyto adresy budou spíše sloužit k ukládání myšlenek a návrhů na nové funkce, než na zakládání problémů se současným systémem.