ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Doplňující služby pro systém Zabbix
Student:	Martin Piták
Vedoucí:	Ing. Jiří Smítka
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Popište monitorovací systém Zabbix.

Navrhněte a implementujte aplikaci, která umožní uživatelům systému Zabbix simulovat vyhodnocování podmínek pro hlášení problémů.
Navrhněte a implementujte službu, která bude v reálném čase posílat informace o problémech ze systému Zabbix do internetové komunikační služby Slack.
Navrhněte a implementujte způsob distribuce logu ze systému Zabbix na jiné servery s využitím služby syslog.
Proveďte testování všech implementovaných funkcionalit v reálném provozu.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 14. října 2016

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

Bakalářská práce

Doplňující služby pro systém Zabbix

Martin Piták

Vedoucí práce: Ing. Jiří Smítka

15. května 2017

Poděkování

Chtěl byl poděkovat svému vedoucímu za pomoc při zpracování bakalářské práce. Svojí rodině za jejich neskutečnou podporu při studiu. Dále bych chtěl poděkovat Lence Kubičové za kontrolu práce.

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 15. května 2017 .....

Odkaz na tuto práci

Piták, Martin. *Doplňující služby pro systém Zabbix*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací tří doplňkových služeb k systému Zabbix konkrétně simulátoru podmínek pro systém Zabbix, spojení mezi systémem Zabbix a službou Slack a přeposílání záznamů ze systému Zabbix, naprogramovaných v různých programovacích jazycích. Výsledné doplňkové služby jsou kompletně implementované a otestované v reálném prostředí. Vytvořené aplikace umožňují snadné rozšíření o další funkce. Přínosem této práce je poskytnutí nástrojů komunitě, které zjednodušují práci se systémem Zabbix. V příloze práce jsou přiloženy zdrojové kódy všech tří doplňkových služeb.

Klíčová slova integrace mezi systémem Zabbix a službou Slack, serverová služba, simulátor podmínek, komunita systému Zabbix, Slack, Zabbix, Python, Java

Abstract

This work deals with analys, design and implementatin of three additional services for Zabbix, namely the Zabbix expression simulator, connection between Zabbix and Slack, and forwarding of Zabbix log by syslog, programmed

in various programming languages. The resulting additional services are implemented and tested in real environment. The created applications allow easy extension with additional features. The benefit of this work is to provide tools for community that would simplify working with Zabbix. In the appendix you can find source codes of all thee additional services.

Keywords integration for communication between Slack and Zabbix, service, expression simulator, Zabbix community, Slack, Zabbix, Python, Java

Obsah

Seznam obrázků

Úvod

Nacházíme se v době, kdy vzniká velké množství nových služeb a produktů, které usnadňují naši činnost a spolupráci ve skupině. Mezi těmito nově vznikajícími službami a produkty většinou zatím neexistuje žádná interakce. Ani přes jejich vzrůstající množství nelze vždy nalézt ty správné, které budou odpovídat konkrétním požadavkům klienta.

V této práci se zabývám analýzou, návrhem a implementací tří doplňkových služeb k systému Zabbix, zpracovaných v různých programovacích jazycích. Jsou jimi: simulátor podmínek pro systém Zabbix, spojení mezi systémem Zabbix a službou Slack a přeposílání záznamů ze systému Zabbix na jiné servery pomocí služby syslog. Téma jsem si zvolil proto, že monitorovacímu systému Zabbix chybí některé dle mého názoru důležité funkce a neexistuje žádné propojení mezi ním a službou Slack. Výsledky práce budou prospěšné hlavně stávajícím uživatelům monitorovacího systému Zabbix a služby pro online spolupráci Slack, ale také lidem postrádajícím tyto funkce u podobných služeb.

Nejdříve se věnuji popisu systému Zabbix a služby Slack. Následovat budou tři kapitoly, kdy každá bude věnována jednomu softwaru. Jednotlivě každou kapitolu rozděluji na analýzu, návrh, implementaci a testování.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je popis monitorovacího systému Zabbix, nalezení a analýza již existujících řešení pro simulování podmínek pro systém Zabbix, spojení mezi systémem Zabbix a službou Slack a přeposílání záznamů ze systému Zabbix na jiný server přes službu syslog. Dále je cílem prozkoumání možných technologií a analýza a výběr takových technologií, které budou nejlépe splňovat požadavky doplňků.

Cílem praktické části práce je návrh a implementace simulátoru podmínek pro systém Zabbix, spojení mezi systémem Zabbix a službou Slack a přeposílání záznamů ze systému Zabbix na jiný server přes službu syslog a jejich následné otestování v reálném prostředí.

Kapitola 2

Zabbix

Zabbix je *open source*[1] monitorovací systém určení do podnikového prostředí. Jeho autorem je Lotyš Alexei Vladishev, zakladatel a generální ředitel Zabbix company. První verze, 1.0, byla vydána v roce 2004 a poslední verze, 3.2, v říjnu 2016.

Backend[2] systému je naprogramován v jazyce C, *frontend*[3] je napsáno v jazyce PHP. Data jsou uložena v databázi.

Zabbix poskytuje několik možností jak kontrolovat programové a hardwarové vybavení. Jedna z nejpoužívanějších možností získávání dat ze sledovaného zařízení je přes Zabbix agent, *software*, který se instaluje na cílovou stanici.

Mezi další podporované technologie systému Zabbix na získávání údajů z monitorovaných zařízení patří například SNMP, IPMI, JMX nebo přes SSH a Telnet scripty. [6]

Na jiném principu než předchozí funguje program Zabbix trapper, který posílá informaci předanou uživatelským příkazem, nebo scriptem, na Zabbix server.

K získávání dat ze zařízení je nutné, aby mělo veřejnou IP adresu. Pokud je za Natem, tak je nutné použít proxy server, Zabbix proxy, který obstarává komunikaci mezi serverem a zařízeními na místní síti.

Mezi nejvýznamnější funkce Zabbixu patří automatické detekování určitých součástí monitorovaného zařízení a velmi komplexní, ale přesto jednoduché tvoření Triggerů.

[1] Aplikace u které byl poskytnut zdrojový kód.
[2] Část webové aplikace, sloužící k administraci a zpracování dat.
[3] Část webové aplikace, kterou vidí uživatel.

2.1 Nastavení - Configuration

2.1.1 Host

Host je objekt, který popisuje monitorované zařízení. Nastavují se zde základní informace o zařízení, IP adresa, *hostname*, výrobce a další. Host může být zařazen do skupiny pro snadnější vyhledávání a třídění. Déle se tu nastavují veškeré monitorované Itemy a Triggery, které mohou být přiřazeny ručně nebo pomocí template.[7]

2.1.2 Item

Item je monitorovaný údaj na zařízení. Každý Item má tři základní údaje, jméno, typ a klíč. Klíč slouží jako jeho identifikátor. Mezi další vlastnosti Itemu patří datový typ, jednotky a údaj o frekvenci jak se má hodnota aktualizovat. Itemy mohou být v aplikaci pro snadnější hledání a třídění.[7]

2.1.3 Trigger

Priorita	Operátor	Popis
1	-	unární mínus
2	not	logická negace
3	*	násobení
	/	dělení
4	+	aritmetické plus
	-	aritmetické mínus
5	<	méně než, definováno jako: A<B <=> (A<=B-0.000001)
	<=	méně než nebo rovno
	>	více než, definováno jako: A>B <=> (A>=B-0.000001)
	>=	více než nebo rovno
6	=	je rovno, definováno jako: A=B <=> (A>B-0.000001) a (A<B+0.000001)
	<>	není rovno, definováno jako: A<>B <=> (A<=>B-0.000001) nebo (A>=B+0.000001)
7	and	logický AND
8	or	logický OR

Tabulka 2.1: Podporované operace v Zabbix Triggeru

Trigger je předpis popisující stav, na který je potřeba upozornit. Jedná se o matematicko-logický předpis[4], jehož výsledkem je jestli situace nastala (jakékoliv číslo kromě nuly) nebo nenastala (nula). Do předpisu se vkládají klíče Itemů s funkcemi a makra[5].

Jako první se provádí operace s prioritou jedna a poslední osm. Všechny operátory jsou asociativní zleva až na unární mínus a logickou negaci. Každý Trigger má úroveň závažnosti, celkem je šest úrovní závažnosti.[7]

2.1.4 Template

Template je sada entit (Itemy, Triggery, grafy atd.), která se dá aplikovat na více Hostů zároveň. Většinou ho používáme ke sjednocení entit patřící k jedné službě nebo aplikaci (Apache, MySQL, Postfix atd.). Templaty poskytují jednu velkou výhodu. Když je potřeba upravit jednu entitu na všech Hostech, stačí změnit template a úpravy se projeví na všech Hostech, kteří mají danou template přiřazenou.[7]

2.1.5 Action

Action je akce, která vznikne v závislosti na nastalé události. Nejdůležitější z nich je změna stavu Triggeru, zda platí nebo neplatí. U akce se nastavuje zpráva pomocí maker, která se má a za jaké podmínky poslat. Nemusí se jen posílat zpráva, ale mohou se provádět i další operace, např. spustit *script* na monitorovaném zařízení.[7]

2.1.6 Maintenance

Maintenance je období údržby. Nastavuje se zde, jestli se budou sbírat data nebo ne a doba platnosti. Doby v obdobích, ve kterých údržby poběží, mohou být jednorázové nebo opakující se. Hosty, ale i skupiny Hostů, na které se tato údržba vztahuje, se také nastavují.[7]

2.2 Monitoring

2.2.1 Dashboard

Dashboard je úvodní stránka, kde jsou zobrazené souhrnné informace o stavu Zabbixu, monitorovaných skupinách a Hostech, posledních několik Triggerů, nejčastější Triggery a další jiné statistiky. Dashboard se dá filtrovat, takže každý uživatel může vidět jen ty údaje, které ho zajímají.

[4] Kombinace metematických a logickýc operací

[5] Proměná, která se mění na základě kontextu.

2.2.2 Lates data

Druhá nejpoužívanější stránka, která nám umožňuje zobrazit přijatá data, která se vyhledávají pomocí filtrů. Filtrovat se dá pomocí Hostů a skupiny Hostů, aplikace nebo jména Itemu.

2.2.3 Triggers

Nejvíce používaná stránka, která nám ukazuje přehled posledních Triggerů, které nastaly. Triggery se dají filtrovat podle jména, aplikace, stáří, potvrzení problému a závažnosti.

Ke každému Triggeru můžeme přidat *acknowledgement*, tedy informaci o tom, že jsme Trigger zaregistrovali a připojíme zprávu. U každého Triggeru může být i více zpráv.

2.2.4 Events

Stránka s přehledem všech událostí týkající se Triggerů. Dá se filtrovat pomocí času nebo pomocí vybraného Triggeru.

2.2.5 Screens

Stránka, na které si můžeme nastavit vlastní souhrn informací přehledně na jedné obrazovce.

Kapitola 3

Slack

Slack je *freemium*[6] webový nástroj po týmovou kooperaci a spolupráci. Vznikl ve firmě Tiny Speck jako interní nástroj na komunikaci při tvorbě hry Glitch, která nebyla nikdy dokončena. Firma později zjistila, že je to velmi dobrý nástroj a zpřístupnila ho veřejnosti. Nyní se plně věnuje rozvoji Slacku. Byl použit týmem, který pomáhal dostat sondu na Mars[7].

Podporuje dvoufázové ověřování[8]. Velmi připomíná IRC. Slack podporuje integrace a už dnes existuje více jak sto padesát integrací[9], mezi které patří například, Google Drive, Trello, Dropbox, Box nebo GitHub. Podporuje veřejné kanály, ve kterých mohou komunikovat i lidé bez registrace. Do Slacku lze zvát Hosty, kteří mají přístup do jednoho kanálu zdarma.

Firmě Tiny Speck se platí za počet uživatelů ve firmě. Počet kanálů není nijak omezen.

[6] Aplikace je zdarma v omezené míře , za více funkcionalit se musí platit.

[7] https://slack.com/customers/nasajpl

[8] Ověření nejen pomocí hesla ale i za pomoci další informace. Například ssh klíč nebo kód vygenerovaný pomocí jiné aplikace.

[9] https://slack.com/apps

Kapitola 4

Zabbix Trigger simulator

4.1 Analýza

4.1.1 Požadavky na aplikaci

Po aplikaci požaduji, aby uměla přijímat stejné výrazy, jaké přijímají Zabbix Triggery. Musí být schopna určit správnost těchto výrazů. Uživatel musí mít možnost otestovat zadaný výraz na objemu dat, jejichž velikost bude moci určit. Výsledky tohoto otestování budou přehledně znázorněny tak, aby uživatel poznal, kdy výraz platí a kdy ne. Aplikace musí být co nejvíce přenositelná, tzv. aby šla bez větších potíží spustit jak na Linux, tak i na Windows.

4.1.2 Analýza konkurence

Při vyhledávání již existujících řešení, které by mohly konkurovat mé aplikaci, jsem byl neúspěšný. Pokud existují, tak je nikdo nezveřejnil, nebo jsou špatně dohledatelné. V Zabbixu existuje testování Triggerů pouze tak, že se do jednotlivých Itemů dá dosadit jen jedna hodnota, ale už zde není žádná závislost na předchozích hodnotách nebo čase.

4.1.3 Analýza potřebných technologií

Cílem aplikace je, aby byla co nejvíce přenositelná. Musel jsem vybrat co nejvíce přenositelný jazyk. Nejznámější z nich je Java běžící na JVM, což je virtuální stroj. Ale na JVM nemusí běžet jen Java, ale i implementace jiných jazyků[5]. Nakonec jsem vybral Javu jako velmi podporovaný jazyk, který se dá bez větších potíží spustit na jakémkoliv počítači. Dnes se najde velmi málo počítačů, na kterých Java není nainstalovaná.

Aplikace musí srozumitelným způsobem zprostředkovávat výsledky uživateli. Musí se s ní relativně dobře pracovat, proto jsem zvolil grafickou aplikaci. Zde nastala otázka, jakou technologii použiji pro tvorbu GUI. QT a SWT jsem vyřadil, protože potřebují knihovny pro každý systém zvlášť a nejsou

natolik přenositelné. AWT a Swing jsou velmi zastaralé a dnes již nevyhovující. SwingX, jako znovuzrození Swingu, se dnes zda být projektem, který se už nebude rozšiřovat. JavaFX byla oficiálně přidána do Javy osm a v dnešní době se zdá jako nejlepší volba pro tvorbu jednoduchého GUI. Zvolil jsem JavaFX, protože je v základě Javy, a tím je její vývoj na nejbližší dobu zajištěn. Současně také vypadá moderně.[1]

Po podrobné analýze Zabbix Triggerů jsem potřeboval něco, co by je zvládlo vyhodnocovat, tedy výraz přečíst a přeložit do výsledné hodnoty pravda nebo nepravda. Po prvních neúspěšných pokusech najít již existující řešení, jsem se rozhodl začít tvořit vlastni gramatiku. Tu se mi nedařilo popsat, protože čím více operátorů jsem přidával do gramatiky, tím více se stávala složitější, nepřehlednější a často jsem při její tvorbě dělal chyby. Po mnohých nezdarech jsem objevil EvalEx[10] , knihovnu pro Javu, která umí počítat aritmeticko-logické výrazy a to dokonce i s prioritou operací, kterou potřebuji. Také se dají operace přidávat nebo upravovat. Knihovna je skoro dokonalá, ale má jeden problém. Operace *not* je brána jako funkce a musí mít za sebou závorky. V Zabbixu jsou závorky u operátoru *not* nepovinné, proto budu muset vyřešit v návrhu, jak budu do výrazu přidávat závorky tak, aby to EvalEx mohl vyhodnotit.

4.1.4 Analýza fungování Zabbixu

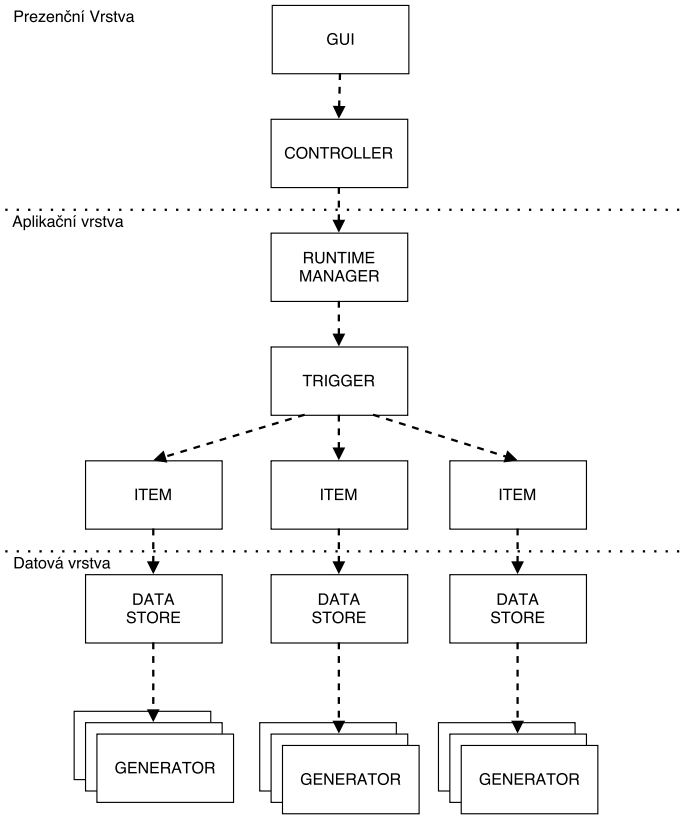
Agent posílá data všech jemu přiřazených Itemů do systému Zabbix. Ten vyhodnocuje Triggery na základě Itemů a provádí akce. Zabbix provede vyhodnocení Triggeru, když jakýkoliv Item obsažený v Triggeru dostane novou hodnotu. Pokud je ve výrazu časová funkce, tak se vyhodnocení provede každých třicet vteřin a zároveň když přijde nová hodnota. Trigger podporuje jedno makro[11], které říká jeho status, zda podmínka z minula platila nebo ne.

4.2 Návrh aplikace

Zabbix Triggery podporují spoustu funkcí, ne všechny je ale potřeba simulovat. Některé funkce pracují pouze s aktuální hodnotou a předchozí na ně nemají vliv. Jedná se hlavně o funkce pracující s textem. Další funkce, kterou není nutno simulovat, je funkce *nodata*. Říká, jestli Item dostal za zadanou dobu nějaká data. Nemá smysl simulovat funkci, která platí, když nejsou data, protože testujeme výrazy nad daty a taková situace nikdy nenastane.

Jako první jsem začal navrhovat spojení mezi agentem a systémem Zabbix. Potřeboval jsem navrhnout systém simulace posílání dat z agenta do Zabbixu. Nejdříve jsem chtěl navrhnout systém, díky kterému bude moci uživatel zvolit, nad jakými daty se bude Trigger analyzovat, a tím určit jaká data bude mít

[10] https://github.com/uklimaschewski/EvalEx
[11] {TRIGGER-STATUS})



Obrázek 4.1: Vnitřní reprezentace aplikace

Item. Rozhodl jsem se pro dva generátory určené předpisem. První generátor vytváří pole hodnot a jeho syntax je

$$[n1, n2, n3, n4, ..., nl] * k,$$

kde k je počet opakování. Pokud není určeno jinak, k je jedna. Druhý generátor vytváří řadu mezi dvěma hodnotama $<x,y>$. Jeho syntax je

$$\{x..y\} * k,$$

kde k je počet opakování. Pokud není určeno jinak, k je opět jedna. Generátory se dají řetězit, a to použitím zřetězení $+$, například

$$[n1, n2, n3] * k + x, y * l].$$

Další částí musí být simulace databáze, do které se data ukládají. Toto provádí DataStore, který zpřístupňuje data vyšší vrstvě a obsahuje generátory.

Item v Zabbixu simuluji objektem Item, který provádí operace nad daty a obstarává získávání dalších dat z DataStore. Protože je potřeba získávat data v určitých intervalech, tak za určitý počet iterací, Item požádá DataStore

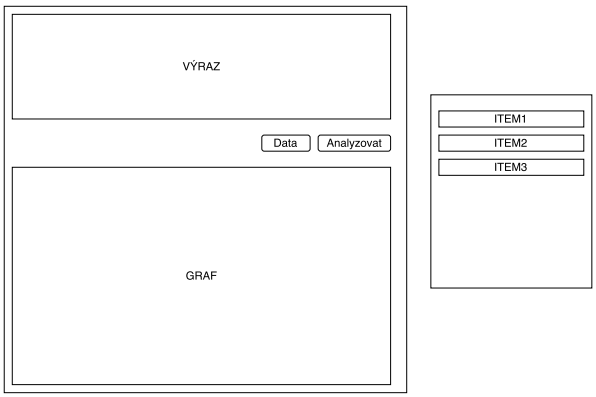
o další data a ten si nechá vygenerovat generátory další hodnotu. Simuluji čas pomocí cyklu, kde jedna iterace je jedna vteřina.

Trigger provádí vyhodnocování výrazu, tvorbu Itemů, kontrolu správnosti výrazu a slouží jako spojovací článek mezi RuntimeManagerem a Itemy.

RuntimeManager provádí jednotlivé vyhodnocení Triggeru a vytváří data pro grafickou část.

Aplikace se dá označit za vícevrstvou, i když to nebylo plánované. První zobrazovací vrstva, neboli GUI, řešená pomocí JavaFX, komunikuje s Controllerem. Druhá aplikační vrstva je RuntimeManager, který vše řídí a pracuje s Triggerem. Trigger operuje nad Itemy. Třetí datová vrstva je DataStore, který pracuje s generátory dat.

Poslední část, kterou jsem potřeboval nasimulovat, bylo datum. Protože Zabbix podporuje funkce vracející čas, přemýšlel jsem o intervalu mezi daty. Nevymyslel jsem ale, jak se bude čas posouvat. Buď se bude čas posouvat po vteřině jako to dělají Itemy, nebo si uživatel bude moci zvolit, jak se má čas posouvat. Další možností je statické datum, které jsem nakonec zvolil. S posouváním po vteřině by uživatel musel počítat už při tvorbě generátorů a možnost vlastního kroku by uživatele vystavovala počítaní konečného času.

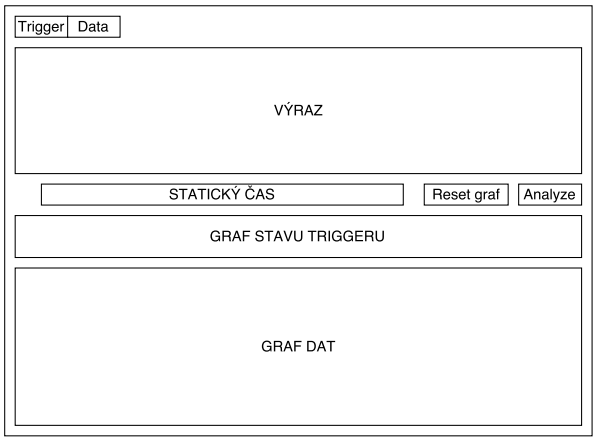


Obrázek 4.2: První verze GUI

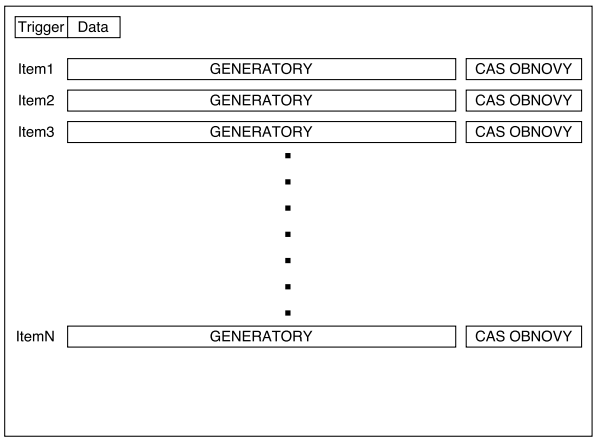
4.3 Návrh GUI

Protože jsem vybral grafickou aplikaci, tak bylo nutné udělat i návrh grafického rozhraní. Vytvořil jsem celkově dvě varianty. První varianta nebyla úplně optimální, protože otevírání nového okna pro zadávání předpisů pro Itemy bylo zmatečné. Uživatel by musel pracovat s dvěma okny, a kdyby měl obě maximalizované, tak by musel mezi nimi složitě přepínat. Také se může jedno okno schovat a uživatel by nemohl pracovat s výrazem, dokud by nezavřel okno s daty.

Druhý návrh místo s dvěma okny, pracoval se záložkami. Tento způsob rozložení mi připadá mnohem lepší. Tím, že je vše v jednom okně, stačí přepínat mezi záložkami. Lidé se záložkami pracují běžně, například v internetových prohlížečích, nebo tabulkových procesorech.



Obrázek 4.3: Druhý verze GUI, okno s výrazy



Obrázek 4.4: Druhý verze GUI, okno s daty

Optimální řešení bylo takové rozložení, kde vše bylo v jednom okně a zobrazovalo se dobře i při malé velikosti. Takové řešení by ale bylo velice nepřehledné.

Budu se snažit o relativně minimalistické rozložení bez zbytečných textů a všechny objekty budou mít k sobě popisek, který se zobrazí až po najetí kurzoru.

4.4 Implementace

Implementaci jsem začal Generátory. Zde jsem použil jednoduchý *polymorfismus*[12], abych nemusel v DataStore rozlišovat o jaký Generátor se jedná.

DataStore je jednoduchá třída, která obsahuje pole dat, nad kterými se dále pracuje.

Item je nesložitější třída, protože provádí všechny funkce nad daty. Řeší se zde převod textu z výrazu v Triggeru na volání funkce.

Trigger je třída, která provádí samotné vyhodnocování, mimo jiné se zde provádí potřebné úpravy výrazu, aby byl použitelný pro EvalEX a také tvorba potřebných Itemů.

RuntimeManager sbírá vypočítaná data a předává je Controlleru, který je zobrazí v grafech.

Controller je třída starající se o zobrazování chybových hlášek a o tvorbu potřebných částí grafického rozhraní.

Při implementaci jsem narazil na několik problémů. První problém byl, jak upravit výraz tak, abych ho mohl použít v EvalEx. Problémová část byla operace not, která v EvalEx potřebuje závorky kolem negovaného výrazu, kdežto v Zabbixu nejsou potřeba. Proto jsem vytvořil funkci, která mi pro zadanou otevírací závorku najde příslušnou zavírací závorku, abych mohl přidat svoje závorky pro not.

Druhý problém byl, když se ve výrazu vyskytovali podobné Itemy a lišily se například mezerou u parametru. Pro Itemy s klíči $item1.funkce(parametr)$ a $item1.funkce(parametr)$ se vytvořily dva rozdílné Itemy. Zpočátku jsem to chtěl řešit tím, že odstraním všechny mezery z výrazu a nahradím uživatelem zadaný výraz upraveným, ale to se ukázalo jako špatný nápad, protože to měnilo text uživateli pod rukama a působilo to nepříjemně pro uživatele a zároveň to komplikovalo analýzu Triggerů. Proto jsem se rozhodl udržovat originální a upravený výraz a jen s tím, že upravím originální výraz tak, že nahradím Itemy, Itemy bez mezer.

Třetí problém byla rychlost simulace pro velké množství dat. Původně jsem simuloval výraz každou vteřinu, ale i Zabbix to dělá jinak, vždy při nové hodnotě v Itemu, tak jsem to udělal stejně jako to má Zabbix.

Dalším největším problémem, který nastal v implementaci, byl ten, že jsem nedokázal naimplementovat dvě metody forecast[13] a timeleft[14] . Obě tyto metody pracují s předpovídáním vývoje přijatých dat v budoucnu. Ani z popisu či dokumentace těchto dvou funkcí jsem nebyl schopen replikovat jejich funkčnost. Po prozkoumání zdrojového kódu jsem usoudil, že tyto dvě funkce vynechám. Funkce jsou relativně nové, jsou podporované až od verze 3.0.0.

[12] Technika, která nám umožňuje tvořit stejné objekty ale s jiným chováním.

[13] Předpověď další hodnoty Itemu na základě předchozích hodnot.

[14] Odhad doby potřebné k tomu aby hodnota Itemu překročila danou mez.

Při implementaci jsem přišel na to, že když se testuje výraz s hodně Itemy a velkým množstvím dat, tak není přesně vidět, v jaký moment výraz platí nebo neplatí. Našel jsem knihovnu jfxutils[15] , která toto řeší. Navazuje na JavaFX a přidává možnost posunu a přibližování v grafech.

4.5 Testování

Kód aplikace byl průběžně testován Unit testy[16] . Devadesát dva procent řádků aplikace je pokrytých Unit testy. Podle mě je toto velmi dobré pokrytí, ale nejde úplně určit, jestli je devadesát dva procent hodně nebo málo, protože záleží jaký kód je testován a kolik nejpoužívanějšího kódu je pokryto.[2] [3] [4]

Chování aplikace jako takové se testovalo ručně na výrazu pro hysterezi použitém v reálném prostředí.

```
({TRIGGER.VALUE}=0 and {AGENT U-D trigger without proxy:agent.ping.
    nodata.avg(#7)}>0.7) or ({TRIGGER.VALUE}=1 and ({AGENT U-D trigger
     without proxy:agent.ping.nodata.avg(#14)}>0.2 or {AGENT U-D
    trigger without proxy:agent.ping.nodata.avg(#3)}=1))
```

Další testovací výraz byl s dvanácti různými Itemy. Poslední testovací scénář byl výraz s dvěma Itemy a generátory generující dvacet tisíc údajů, takzvaně čtyřicet tisíc údajů celkem.

4.6 Budoucí rozšíření

Myslím si, že možností jak rozšířit aplikaci je velmi mnoho. První věc, která by stála za rozšíření, je přidat implementace funkcí forecast a timeleft.

O další rozšíření by se zasloužilo i generování dat. Mělo by se zvážit, jestli by bylo užitečné přidání náhodně generovaných dat z daného rozsahu. Na základě tohoto zvážení by se pak rozhodlo, jestli se tato funkcionalita bude implementovat. Řada je pouze jednosměrná a to vzestupně. Měla by se přidat možnost definovat i řadu klesající.

Pro některé uživatele by mohlo být užitečné, aby se přidala podpora pro funkce pracující s textem. Pro ty by se muselo vymyslet, jak by se texty generovaly. Napadá mě syntax podobná regulárním výrazům, ale to by bylo nutno podrobněji prozkoumat.

Nejpomalejší část je přidávání dat do grafu. Pokud by se v budoucnosti vymyslela optimalizace přidávání dat do grafu a paralelizace vyhodnocování výrazů, tak by to zrychlilo běh aplikace. Pokud by se podařilo zrychlit přidávání dat do grafu, tak další část, která by stála za zrychlení je část kdy se simuluje čas.

[15] https://github.com/gillius/jfxutils

[16] Test, který ověřuje dílčí funkcionalitu kódu.

Kapitola 5

Slabbix

5.1 Analýza

5.1.1 Analýza konkurence

Při hledání již existujících řešení spojení těchto dvou služeb jsem našel pouze jedno řešení[17], které bylo naprosto nedostačující a špatně rozšířitelné. Jedná se o jednoduchý Shell script, který posílá nové udaje do Slacku přes webhooky[18] do jednoho kanálu. Tento způsob řešení používá pouze jednostrannou komunikaci, a také je velice nepraktický. Při velkém množství údajů by byl kanál ve Slacku zahlcený.

5.1.2 Požadavky na aplikaci

Aplikace musí běžet jako služba na serveru Linux, která se spouští se startem systému. Aplikaci je možno konfigurovat pomocí souboru a musí umět logovat[19] informace o aktivitě do souboru. Při Eventu se do Slacku do příslušného kanálu pošle zpráva, která obsahuje informace o události. Ze Slacku je možno udělat *acknowledgement* události. Na základě této akce se všechny zprávy o daném problému upraví a připíše se do nich zpráva a kdo *acknowledgement* udělal. Zároveň se do Zabbixu k události přidá *acknowledgement*, ve kterém je jméno uživatele a zpráva. Aplikace musí být snadno rozšiřitelná o další příkazy. Po vyřešení problému se upraví všechny zprávy, aby bylo poznat, zda je problém vyřešený. V každý pracovní den se vypíšou všechny aktivní problémy. Zprávy musí být strukturovány tak, aby zabíraly co nejméně místa, ale obsahovaly co nejvíce informací.

[17] https://github.com/ericoc/zabbix-slack-alertscript
[18] Způsob komunikace se serverem pomocí webového serveru.
[19] Zaznamenávat udaje o aktivitě nebo o chybách do souboru.

5.1.3 Analýza potřebných technologií

Aplikace musí běžet jako služba na serveru Linux a existuje několik možností jak to zajistit. Může se spouštět pomocí init scriptu[20] nebo pomocí správce služeb. Rozhodl jsem se pro správce služeb a to konkrétně system-d[21].

Existuje několik různých knihoven, které lze použít pro komunikaci se Slackem. Její optimální výběr byl velmi složitý, ale velkým faktorem byla rozsáhlost knihovny, její popularita a moje znalost jazyka.

SDK	Jazyk	Popis
Node.js SDK	node.js	oficiální SDK, relativně nové
Python SDK	Python	oficiální SDK, dlouho vyvíjené, syntax podobná SQL příkazům
SlackAPI	C#	podpora C# není na Linuxu úplně optimální
Slack	GO	nutnost kompilace, rychlé
BotKit	node.js	velmi obecná knihovna není vhodná
slapp	node.js	potřebuje webový server
slack-client	php	potřebuje webový server
Slacker	python	dlouhá historie, velmi dobrá syntax, velice populární
slack-ruby-bot	ruby	velice podobné Slackeru

Tabulka 5.1: Knihovny pro komunikaci se Slackem

Vybral jsem Slacker, protože má velkou popularitu v komunitě a jeho syntax je podobná vnitřní syntaxi Slacku. Má velmi aktivní vývoj, na němž se podílí velká skupina lidí. SDK přímo od Slacku nejsou dostatečné, protože jsou složité a mají nízkoúrovňové funkce. SlackAPI C# není ideální, protože podpora pro C# na Linuxu není dokonalá. Knihovny nad node.js a php potřebují pro jejich funkčnost webový server, který by byl zbytečně běžící proces na serveru. Slack-ruby-bot je jediný velký konkurent vybraného Slackeru, ale Ruby je primárně určen na práci s webem. Jeho vývoj je velmi rychlý, často se mění a s novými verzemi se většinou rozbíjí starší části kódu.

Slacker je knihovna pro Python, proto jsem potřeboval knihovnu, která by zajišťovala komunikaci se Slackem a byla napsaná v Pythonu. Zvažoval jsem i možnost, že bych si potřebné části napsal sám, ale rozhodl jsem se použít již napsanou a odzkoušenou knihovnu.

Vybral jsem pyzabbix, protože je velmi populární a na jeho vývoji se podílí premium zákazník Zabbixu. Ostatní možnosti se jeví jako neaktuální a bez dalšího vývoje.

[20] Příkazy a procedury, které se provedou při startu systému.

[21] https://www.freedesktop.org/wiki/Software/systemd/

SDK	Popis
py-zabbix	nepotřebuje žádné další knihovny
pyzabbix	nejpopulárnější, spravováno premium klientem Zabbixu, Systematica
ZabbixPythonApi	bez podpory Pythonu 3
zabbix	přepsaná ruby knihovna, velmi málo populární
zabbix-client	podobné pyzabbixu, ale špatně zdokumentováno a málo používané
zabbix-api-erigones	špatná syntax, podobná SQL příkazům

Tabulka 5.2: Python knihovny pro komunikaci se Zabbixem[8]

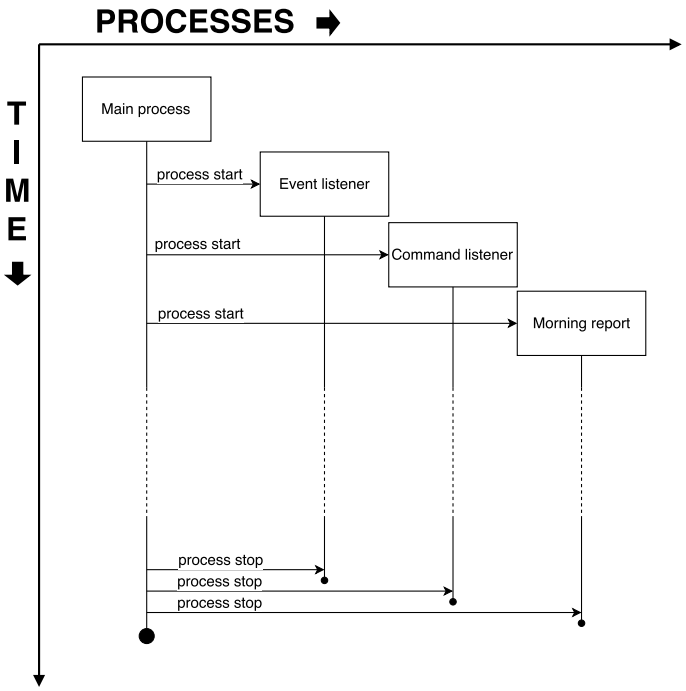
Protože Slacker neuměl websockety, které Slack používá pro přenos dat v reálném čase, musel jsem nalézt knihovnu, která to bude dělat za něj. Po několika nefunkčních knihovnách jsem narazil na websocket-client, která zprostředkovává spojovací kanál mezi Slackerem a Slackem.

Dalším tématem bylo ukládání dat. Nejdříve jsem se rozhodl ukládat data lokálně pomocí knihovny Pickle, ale při testování této knihovny jsem narazil na několik vad. Když se přerušil zápis dat, tak soubor s daty byl poškozený a jeho obnova byla nadmíru zdlouhavá a pracná. Další nevýhoda byla velmi dlouhá doba zápisu velkého množství dat při ukončování aplikace. Nabízela se i možnost průběžného zapisovaní, ale ta by jen zbytečně zpomalovala chod aplikace. Zvažoval jsem také Průběžné zapisování pomocí jiného vlákna, ale přišlo mi to jako zbytečně překombinované. Potřeboval jsem úložiště, které je rychlé a vydrží nečekaný pád serveru a podobné kritické situace. Proto jsem se rozhodl pro SQL databázi a pro tu jsem potřeboval další knihovnu. Zvolil jsem pymysql. Přestože patří mezi jednodušší knihovny, její používání je naprosto dostačující.

Knihovna	Popis
pymysql	implementováno jen za použití Pythonu, náhrada za MySQLdb
sqlite3	primárně určená pro SQLite databázi
MySQLdb	bez podpory Pythonu 3
SQLalchemy	ORM řešení, vhodné pro velké a složité projekty ne pro malý projekt

Tabulka 5.3: Python knihovny pro komunikaci s MySQL databází

5.2 Návrh Aplikace



Obrázek 5.1: Průběh života aplikace

Aplikace se skládá ze dvou částí, klienta a ze serveru. Klient je spuštěn Zabbixem s parametry. Klient na základně parametrů rozhodne, jestli má serveru předat informace nebo ne. Server se skládá z několika procesů. První, hlavní, proces se stará o spuštění ostatních procesů a o korektní ukončení celé aplikace. Druhý proces poslouchá na localhostu na příchozí spojení od Systému Zabbix a posílá informace o Eventech do Slacku. Třetí proces poslouchá na websocketu Slacku, kde přijímá veškeré zprávy na všech kanálech, ve kterých je přidán bot[22]. Hledá v nich příkazy a ty následně vyhodnocuje. Poslední, čtvrtý proces, v pravidelných intervalech opakuje všechny nevyřešené Eventy v kanálu.

Problém rozšířitelnosti o další příkazy se dá řešit více způsoby. Prvním je vytvoření vlastního nahrávání dalších rozšíření, nebo druhým je použití již existující řešení na nahrávání rozšíření, například Yapsy[23], Plug n' Play[24], pluginloader[25]. Protože všechny existující postupy se mi zdály moc složité a omezující, tak jsem se rozhodl pro svůj vlastní. Rozšíření se načítají ze speci-

[22]Robot, automat, reagující na uživatelské podněty.
[23]http://yapsy.sourceforge.net/
[24]https://github.com/daltonmatos/plugnplay
[25]https://github.com/magmax/python-pluginloader

ální složky a mají určitou strukturu. Jsou v nich uvedeny důležité parametry. První důležitý parametr je FN, jméno příkazu. Druhý parametr ALIAS, kde jsou uvedeny všechny zástupné příkazy. Poslední je TYPE, určující, k jaké události ve Slacku se vztahuje.

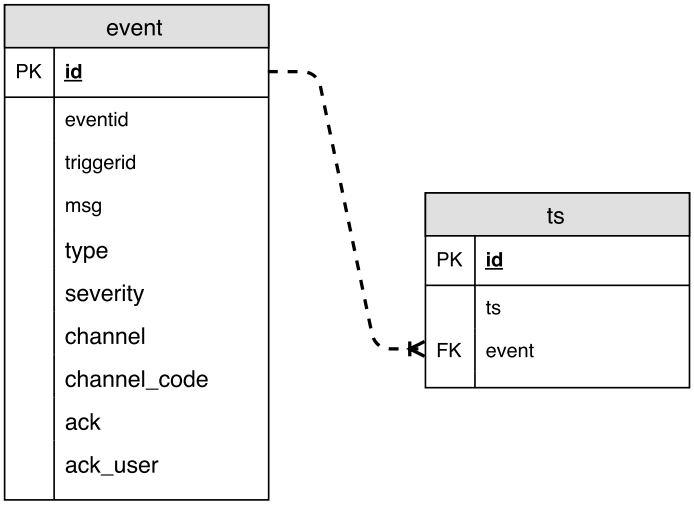
Zprávy posílané do Slacku musejí být co nejkratší, ale co nejinformativnější. Ze zprávy by bylo dobré na první pohled poznat, o jaký typ zprávy se jedná. Protože Slack nepodporuje různé barvy textu nebo podbarvení, zvolil jsem jako rozlišovací znak obrázek následován úrovní závažnosti. Další informace ve zprávě jsou jméno Triggeru, který nastal, kdy Trigger nastal a odkaz do Zabbixu na příslušný Event. Další údaje jsou podle typu zprávy, která se posílá, nebo na kterou se zpráva upravuje. U zprávy typu *acknowledgement* je navíc uveden uživatel, který jej provedl a jeho zpráva. U zprávy typu problém vyřešen je navíc datum, kdy byl Trigger vyřešen a jak dlouho trvalo jeho řešení. U každodenního hlášení všech nevyřešených problému je navíc uvedeno jak dlouho je problém nevyřešený. A protože se ve hlášení vyskytují i zprávy o acknowledgementu, tak u nich navíc, kromě informací o tom jak dlouho je problém nevyřešený, je i kdo udělal acknowledgement a zpráva o něm.

Obrázek 5.2: Ukázka zpráv ve Slacku

Informace o problémech se musí poslat do příslušného kanálu. Musel jsem přijít na způsob, jak předat informaci o kanálu aplikaci. Přišel jsem jen na dva možné způsoby. Musí se použít jedna z vlastností Hosta, třeba *alias* nebo *notes*, nebo použít skupinu, ve které je Host umístěn. První možnost by vyžadovala úpravu každého Hosta a to v systému s více Hosty je zbytečné, zdlouhavé, neefektivní. Změna kanálu pro jednoho Hosta je příjemná, stačí změnit jeden údaj. Přejmenování kanálu by znamenalo velké množství uprav a hledaní, který další Host ještě není změněn. Druhou možností je přidat Hosta do skupiny Hostů, kde by jméno skupiny bylo jméno kanálu. Tato možnost přináší mnoho výhod v přidávání Hostů do skupin, je jednodušší než úprava jejich údajů a hlavně Hosti už v některé skupině budou. Skupiny mohou složit jako rozdělení projektů ve firmě a vybírat se může jen pomocí prvního slova. Mohou existovat skupiny, které jsou ignorované. Rozlišovat se to může pomocí speciálního prvního znaku. Přejmenování kanálů je jen přejmenování skupiny a Host může být ve více skupinách zároveň, takže se zprávy mohou posílat do více kanálů.

5.3 Návrh Databáze

V databázi potřebuji uchovat informace o problému a o všech zprávách, ve kterých byl problém poslán do Slacku. Databáze je velice jednoduchá, jsou to pouze dvě tabulky, kde v jedné uchovávám informace o Eventu a v druhé uchovávám všechny časy zpráv o Eventu. Tyto časy jsou tvořeny převážně každodenním hlášením. U Eventu je přidaná informace o kódu kanálu, protože když Slack upravuje zprávy, tak nepracuje s jménem kanálu ale právě s jeho kódem.



Obrázek 5.3: Návrh databáze pro Slabbix

5.4 Implementace

Server byl původně napsán s vlákny, ale z důvodu rychlosti byl předělán na procesy. V průběhu implementace jsem musel řešit několik úskalí. První z nich bylo, když server, na kterém běží Zabbix a Slabbix, ztratil přístup na internet, bylo zbytečné něco posílat do Slacku, když se to neodešle. Proto jsem potřeboval zachytit informace o všech problémech, které nastaly v době, kdy byl vypadlý internet. Protože Zabbix spouští pro každý Event nového klienta, tak jsem se rozhodl, že pokud nebude mít server přístup k internetu, tak Klient bude čekat, než ho bude mít. V tomto byl malý problém, protože Zabbix čeká maximálně třicet vteřin na provedení scriptu a tento čas se nedá změnit. Proto jsem musel přidat takovou funkcionalitu, že když se spustí Klient a z něj se spustí nový proces, který teprve bude vykonávat dané operace, tak se hlavní proces zavře.

Druhý problém bylo regulérní ukončovaní aplikace, protože websocket-client je blokující, takzvaně čeká na spojení a pak teprve pokračuje v chodu.

Tím pádem by se aplikace ukončila, až kdyby přišla zpráva. To trvalo v některých případech i několik minut. Dokumentace k websocket-clientu neříkala nic o asynchronním používání, ale povedlo se mi ve zdrojových kódech najít informaci o timeoutu[26], díky které se řádné ukončení velmi zrychlilo.

Implementoval jsem tyto moduly. Ack modul, který na zprávu se specifickým příkazem přepošle *acknowledgement* do Zabbixu a upraví zprávu ve Slacku a potřebné informace. Další modul je podobný předchozímu tím, že odpovídá na reakci zprávy. Další modul je Clear, který odstraní specifikované problémy z databáze. Tento příkaz se hodí, právě tehdy když nastane problém a pak se v Zabbixu zakáže. Tím pádem by byl stále ve Slacku. Modul Error, který zaznamená všechny problémy, co nastanou při komunikaci se Slackem

5.5 Testování

Aplikace se velice špatně testuje lokálně, protože velmi závisí na vstupu z třetích stran. Lokálně jsem aplikaci testoval scriptem, který simuloval Zabbix a posílal problémy. Problémy se posílaly do jednoho kanálu. Služba byla také testována v reálném prostředí po dobu dvou měsíců. Testování probíhalo na Systému Zabbix, který monitoroval více jak dvě stě zařízení s více jak dvanácti tisíci různými Triggery a vyhodnocoval více jak dvě stě Triggerů za sekundu.

5.6 Budoucí rozšíření

Většina budoucích rozšíření by znamenala přepsání velkého množství kódů. I když je aplikace rozdělena do několika procesů, tak vně procesu je to imperativní. První rozšíření, které by velice urychlilo chod aplikace, by byla optimalizace ukončování a paralelizace zpracování příkazů ze Slacku a nových problémů.

[26] doba čekání na zprávu

Kapitola 6

Zabbix log to syslog

6.1 Analýza

6.1.1 Aktuální konkurence

Systém Zabbix umí posílat své logy[27] od verze tři do syslogu, proto stačí jen toto nastavit a pak nastavit syslog, aby logy posílal na jiný server.

6.1.2 Požadavky na aplikaci

Aplikace musí být schopná detekovat použitý syslog *daemon*[28] a pokud je podporovaný, tak povolit v Zabbixu logovaní do syslogu a upravit konfiguraci syslogu tak, aby se log přeposílal na jiný server. Aplikace by měla podporovat nejběžnější syslog daemony.

6.1.3 Analýza potřebných technologií

Mezi nejběžnější syslog daemony patří syslog-ng, rsyslog, syslogd a sysklogd. Rsyslog patří mezi nejpoužívanější, protože je používán na Debianu, Ubuntu a Fedore. Syslog-ng se nejčastěji používá na Gentoo a na SuSE a má docela složitou configurační syntax. Syslogd a sysklogd je používán převážně na Unixech.

Musel jsem vybrat, v jakém programovacím jazyce budu aplikaci implementovat. Vybíral jsem z Perlu a Bashe a Pythonu. Velké a složité jazyky jako C nebo C++ jsem vůbec nezvažoval. Přišlo mi, že se jedná o relativně snadný úkon, na který není potřeba používat složitý jazyk. Ne všechny systémy jsou stejné. Dobrá výhoda Bashe je, že každý je schopen si ho otevřít a přizpůsobit svým potřebám.

[27] Soubor se záznamy o aktivitě nebo chybách aplikace.

[28] Program, který běží dlouhodobě, se obvykle spouští se startem systému. Funguje bez interakce uživatele.

Vybral jsem Bash jakožto jazyk, který je dostupný skoro v každé linuxové distribuci. Zná ho skoro každý a je dostatečně obsáhlý, abych v něm mohl vše udělat. Python jsem vyřadil, protože i když je velice populární jazyk, ne vždy může být dostupný na všech systémech. Perl jsem nevybral, i když to byl velký kandidát, protože není tolik populární jako Bash.

6.2 Návrh

Nejprve aplikace zjistí, jaký na systému běží syslog server a pak změní nastavení Zabbix serveru tak, aby posílal logy do syslogu. Poté upraví configuraci syslogu tak, aby se data posílala na uživatelem zadaný server.

6.3 Implementace

Implementace byla relativně snadná. Snažil jsem se o co největší čitelnost, aby byl kód co nejvíce upravitelný.

6.4 Testování

Aplikaci jsem otestoval na reálném serveru s rsyslogem a s konfiguračními soubory pro syslog-ng a pro syslogd a sysklogd.

6.5 Budoucí rozšíření

Aplikace momentálně podporuje pouze tři syslog daemony v budoucnu by proto bylo dobré přidat podporu i pro další.

Závěr

Cílem práce bylo navrhnout a implementovat simulátor podmínek pro monitorovací systém Zabbix, který ulehčí práci uživatelům monitorovacího systému Zabbix. Záměrem bylo i spojení mezi systémem Zabbix a službou pro online spolupráci Slack, která by zvýšila efektivitu práce v podnikovém prostředí. Dalším cílem bylo rovněž přeposílání záznamů ze systému Zabbix na jiný server pomocí služby syslog.

Simulátor podmínek pro systém Zabbix byl navrhnut tak, aby co nejpřesněji simuloval systém Zabbix. Aplikace je napsaná v Javě. Umožňuje komplexní testování výrazů. Syntax pro tvorbu generátorů je snadná, tvoří ji dva generátory jeden na tvorbu pole a druhý na tvorbu řady. Generátory se dají zřetězovat. Přestože jsou jen dva generátory, tak pomocí nich se dají vygenerovat všechny možná data. Aplikace má relativně intuitivní uživatelské prostředí vytvořené pomocí JavaFX. Výsledky se zobrazují na dvou grafech, kde na prvním grafu je informace o výsledku výrazu platí nebo neplatí. Na druhém grafu jsou data jednotlivých Itemů. Druhý graf se dá posouvat a přibližovat, aby mohl uživatel vidět při jakých hodnotách testovaný výraz platí a kdy neplatí.

Integrace mezi systémem Zabbix a službou Slack byla navržená tak, aby podporovala rozšíření o uživatelské příkazy. Je napsaná v jazyce Python a skládá se ze čtyř procesů. Služba funguje na principu klient-server. Zabbix spouští klienta s údaji o problému a klient je posílá na server. Server pak zpracuje udaje, vytvoří a pošle zprávu do Slacku. Služba připomíná nevyřešené události a umožňuje *acknowledgement* vzniklých událostí, které se zpětně projeví v systému Zabbix.

Zabbix podporuje ukládaní logů do syslogu, proto navrhnutí a vytvoření aplikace, která by přeposílala Zabbix log na jiný server pomocí syslogu, bylo celkem snadné. Aplikace přijme od uživatele server, na který se má log přeposílat a pokud detekuje podporovaný syslog *daemon*, tak nastaví Zabbix a syslog, aby se log posílal na zadaný server.

V budoucnosti by bylo možné rozšířit propojení mezi systémem Zabbix a službou Slack o další funkce v podobě pluginů, případně webové rozhraní,

pro lepší možnosti nastavení. Simulátor podmínek pro systém Zabbix je největší a nejsložitější aplikace ze všech tří. Proto si zaslouží největší pozornost. Věřím, že v budoucnosti se přidá podpora pro všechny funkce. Jistě se najde vylepšení vzhledu s možností různých stylů. Zrychlí se běh simulace a přidá se více generátorů dat pro ještě komplexnější možnosti testovaní. Pro zvýšení užitečnosti aplikace na přeposílaní Zabbix logu na jiný server pomocí syslogu by bylo dobré v budoucnu přidat podporu pro více syslog daemonů.

Bibliografie

1. AARON DIGULLA, aj. *Java GUI frameworks. What to choose? Swing, SWT, AWT, SwingX, JGoodies, JavaFX, Apache Pivot?* [online] [cit. 2017-03-23]. Dostupné z: http://stackoverflow.com/questions/7358775/java-gui-frameworks-what-to-choose-swing-swt-awt-swingx-jgoodies-javafx.
2. KILLSCREEN Gishu, aj. *What is a reasonable code coverage % for unit tests (and why)?* [online] [cit. 2017-04-02]. Dostupné z: http://stackoverflow.com/questions/90002/what-is-a-reasonable-code-coverage-for-unit-tests-and-why.
3. SAVOIA, Alberto. *How Much Unit Test Coverage Do You Need? - The Testivus Answer* [online] [cit. 2017-04-02]. Dostupné z: http://www.artima.com/forums/flat.jsp?forum=106&thread=204677.
4. SAVOIA, Alberto. *The Way of Testivus - Unit Testing Wisdom From An Ancient Software Start-up* [online] [cit. 2017-04-02]. Dostupné z: http://www.artima.com/weblogs/viewpost.jsp?thread=203994.
5. WAMPLER, Dean. *Adopting New JVM Languages in the Enterprise* [online] [cit. 2017-03-22]. Dostupné z: https://web.archive.org/web/20090522071822/http://blog.objectmentor.com/articles/2009/01/15/adopting-new-jvm-languages-in-the-enterprise.
6. ZABBIX LLC. *Data Collection* [online] [cit. 2017-03-20]. Dostupné z: http://www.zabbix.com/data_collection.
7. ZABBIX LLC. *Zabbix Manual* [online] [cit. 2017-03-21]. Dostupné z: https://www.zabbix.com/documentation/3.0/manual.
8. *Zabbix Python API libraries* [online] [cit. 2017-04-05]. Dostupné z: https://www.zabbix.org/wiki/Docs/api/libraries.

Příloha A

Seznam použitých zkratek

SNMP Simple Network Management Protocol

IPMI Intelligent Platform Management Interface

JMX Java Management Extensions

SSH Secure Shell

Init Initialization

TELNET Teletype network

NAT Network address translation

IP Internet Protocol

IRC Internet Relay Chat

ORM Object-relational mapping

JVM Java virtual machine

GUI Graphical user interface

SWT Standard Widget Toolkit

AWT Abstract Window Toolkit

SDK Software development kit

ACK Acknowledgement

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnými aplikacemi
- `src`
  - `ZTS` ..... zdrojové kódy implementace simulátoru
  - `Slabbix` ..... zdrojové kódy implementace Slabbixu
  - `Zabbix-syslog` ..... zdrojové kódy implementace přeposílání logu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS