České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Matěj Židek**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Monitorování privátního cloudu**

Pokyny pro vypracování:

Nastudujte problematiku monitorování zátěže ve virtualizovaném linuxovém prostředí obecně (libvirt, collectd, RRDtool...) a privátních cloudech (collectd v OpenNebule, OpenStack Ceilometer). Dále se naučte používat API cloudu OpenNebula. V prostředí IaaS systému OpenNebula navrhněte a implementujte nadstavbu realizující monitoring výkonu jednotlivých virtuálních strojů, která bude uchovávat informace o zátěži po dobu minimálně jednoho roku pro počet aktivních strojů v řádu stovek a neaktivních v řádu desetitisíců. Tyto informace uživatelům prezentujte v jednoduchém webovém rozhraní, kde bude vidět i zátěž fyzických strojů vedle všech na nich běžících virtuálních strojů. Implementaci proveďte v jazyku Ruby.

Seznam odborné literatury:

BLACK, David A. The well-grounded Rubyist. Greenwich, Conn.: Manning, c2009,xxx, 487 s. ISBN 9781933988658.

OLSEN, Russ. Eloquent Ruby. Upper Saddle River, NJ: Addison-Wesley, c2011,xxvii, 413 p. ISBN 0321584104.

WLODARCZYK, Tomasz Wiktor. Performance of Data Extraction from OpenTSDB.[online]. [cit. 2015-04-20]. Dostupné z: http://twwlodarczyk.github.io/papers/opentsdb_extraction_performance_precamera.pdf

FRIEDMAN, Ellen a Ted DUNNING. Time Series Databases: New Ways to Store and Access Data. O'Reilly Media, Inc., 2014. ISBN 9781491914724.

OPENNEBULA PROJECT. OpenNebula 4.12 Documentation [online]. 2015[cit. 2015-04-20]. Dostupné z: http://docs.opennebula.org/4.12/

Vedoucí: Ing. Tomáš Vondra

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 4. 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Balářská práce

Monitorování privátního cloudu

Matěj Židek

Vedoucí práce: Ing. Tomáš Vondra

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové systémy

20. května 2015

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Tomáši Vondrovi za rady a postřehy, Mgr. Borisi Parákovi za připomínky a vstřícnou komunikaci při sběru požadavků, Leoně Židkové za pomoc s pravopisem a přátelům a rodině za podporu.
Výpočetní zdroje byly poskytnuty MetaCentrem v rámci programu LM2010005 a CERIT-SC v programem Centra CERIT Scientific Cloud, součástí Operačního programu pro Výzkum vývoj a inovace, reg. č. CZ.1.05/3.2.00/08.0144.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2015 .....

Abstract

The goal of this bachelor thesis is to create an enhanced monitoring system for a cloud platform Open-Nebula that would store data for a duration of at least one year and present them in graphs in a user-friendly environment.

The final solution uses one2influx daemon to gather monitoring data from OpenNebulas XML-RPC API and saves them to the InfluxDB database. The visualization is done using Grafana tool and after a customization of one2influx the solution is applicable to other cloud solutions as well, e.g. OpenStack.

At the moment the solution is in a state of limited functionality. The problem is caused by an undebugged version of InfluxDB. This work thus serves more as a demonstration of what can be achieved with the use of InfluxDB in a short period of time.

Abstrakt

Cílem této bakalářské práce je vytvoření nadstavby cloudového systému OpenNebula pro rozšířený monitoring, která bude uchovávat data po dobu minimálně jednoho roku a prezentovat je v grafech v přívětivém uživatelském rozhraní.

Výsledné řešení používá démona one2influx k získání dat o monitoringu z OpenNebula XML-RPC API a ukládá je do databáze InfluxDB. Zobrazení probíhá pomocí nástroje Grafana a řešení lze nasadit po úpravě one2influx i na další cloudové platformy např. OpenStack.

Momentálně je systém ve stavu s omezenou funkcionalitou. Problém je způsoben neodladěnou verzí InfluxDB. Práce tak má hlavně význam jako ukazatel, čeho půjde v nejbližší době s InfluxDB dosáhnout.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Vědecké výpočetní centrum CERIT-SC má cloud computing cluster postavený na platformě OpenNebula (ONE), která však neobsahuje pokročilejší podporu sledování zátěže a to především v dlouhodobém horizontu.

ONE má vlastní grafické uživatelské prostředí pro správu cloudu jménem Sunstone, které ale poskytuje jen pár základních grafů staticky ukazujících posledních dvanáct hodin vytížení. Požadavkem CERIT-SC je mít čitelná data z historie a tedy grafy, v kterých se dá vracet do minulosti, ale zobrazovat i aktuální vytížení. Navíc půjde o samostatný systém, který nebude integrovaný do prostředí Sunstone a tak bude možno z grafů vyčíst informace o virtuálních a fyzických strojích v rámci více instancí cloudů OpenNebula (nebo i jiných platforem), které budou integrovány do monitorovacího systému.

S obecným fungováním privátního IaaS cloudu seznámím čtenáře v kapitole Analýza, podrobněji se zaměřím na OpenNebulu i OpenStack a shrnu požadavky zadavatele.

Bude-li řešení někdy nasazeno, je jasné, že bude potřebovat údržbu. Pokud se správně pospojují existující řešení, pak se o údržbu budou starat uživatelé těchto projektů. Při použití existujících systémů je tedy kritické, vybrat ty správné a k tomu je vhodně pospojovat. Těmto úkolům se věnuji v kapitole Návrh.

Po výběru už nezbývá než spojovací článek naprogramovat a celkově uvést systém do chodu, což je společně s testováním předmětem kapitoly Implementace.

Shrnutí čeho jsem dosáhl a co je potřeba vylepšit, najdete v kapitole Závěr. V příloze A je nakonec seznam zkratek a pro zájemce o nasazení uvádím v příloze B Manuál.

Kapitola 2

Analýza

V této kapitole bych nejprve představil cloudové služby, kterými se budu v této práci zabývat. Poté v souladu se zadáním i možnosti monitoringu ve virtualizovaném linuxovém prostředí obecně a konkrétně. Na závěr požadavky na budoucí systém od zadavatele.

2.1 Privátní cloud

Zde uvedu několik základních informací o dvou privátních cloudových řešeních pro projekty OpenNebula a OpenStack. První je důležitý, protože jeho monitoringem se zabývám ve zbytku práce a druhý je tu kvůli možnosti rozšíření monitorovacího systému na další cloudové služby.

2.1.1 OpenNebula

OpenNebula je open-source cloudová platforma sponzorovaná společností OpenNebula Systems. Slouží k vytváření a spravování virtuální infrastruktury, kterou pak lze používat jako public, private nebo hybrid cloud v podobě IaaS. Za klíčové vlastnosti projektu jsou považovány například transparentnost procesů i technologií a inovace v nových technologiích pro pokrytí potřeb velkých cloudových nasazení. [12]

2.1.2 OpenStack

OpenStack je obdobný open-source projekt jako ONE, který začal jako společné dílo Rackspace Hosting a NASA. V současnosti je provozován společností OpenStack Foundation, která je sponzorována mnoha dalšími firmami. [17]

Narozdíl od ONE je OpenStack složen z několika nezávislých částí a poskytuje tak například i služby typu DNS a Hadoop, pro které je v ONE potřeba vytvořit a nastavit samostatné virtuální servery. [12] [28]

2.2 Obecné možnosti monitoringu ve virtualizovaném linuxovém prostředí

V této kapitole představím několik základních nástrojů a dá se říct i principů pro provoz a monitoring IaaS cloudů.

2.2.1 Hypervizor

Hypervizor je v kontextu této práce software, který umožňuje virtualizovat operační systém, neboli dovoluje několika operačním systémům najednou sdílet hardwarové prostředky stroje, na kterém běží. Mezi nejznámější patří například KVM, Xen nebo VirtualBox.

2.2.2 libvirt

Libvirt je skupina nástrojů pro správu virtualizace na jednom fyzickém stroji. Poskytuje jednotný přístup ke správě více různých hypervizorů, bez nutnosti používat jejich specifické nástroje pro kontrolu každého zvlášť. Díky možnosti vzdáleného přístupu je pak vhodný, jako základní stavební prvek rozsáhlejších virtualizačních IaaS systémů nasazených na více fyzických stojích, jako třeba Eucalyptus a OpenStack. [10] [11]

2.2.3 collectd

Collectd je démon, shromažďující data o vytížení stroje, na kterém právě běží. Jednou z výhod tohoto démona je množství pluginů, které lze většinou rozdělit do dvou velkých skupin:

- **vstupní**: Jsou periodicky dotazovány a vrací hodnotu nějakého měření.
- **výstupní**: Získávají z démona hodnoty a mohou je například zapisovat do RRD souborů (více v 3.1.3 RRDtool), CSV souborů nebo je posílat dál po síti, kde může další instance collectd naslouchat a ukládat data centrálně.

Více se lze dočíst v manuálových stránkách [24].

2.2.4 Ganglia

Ganglia je, dá se říci, alternativa ke collectd pro monitorování větších skupin serverů a clusterů. Tento systém nejprve rozdělí síť na jednotlivé skupiny, kde každý člen má v jeden okamžik uloženy všechny naměřené metriky ostatních členů a v rámci skupiny je i sdílí. Centrálnímu shromažďovacímu serveru se pak stačí dotázat kteréhokoliv zařízení ze skupiny, aby získal informace o všech členech. Nasbíraná data jsou ve finále ukládána pomocí RRDtool (více v 3.1.3 RRDtool) a lze k jejich vizualizacím přistupovat i přes webové rozhraní.

Přístup k předávání informací je hlavní rozdíl oproti collectd, kde v nastavení pro větší sítě ví o ostatních zařízeních jen centrální server, který data shromažďuje. Collectd tak musí mít spojení s každým uzlem, naproti tomu Ganglii stačí pouze jedno spojení s každou skupinou. [25]

2.3 Možnosti monitoringu v privátních cloudech

2.3.1 OpenStack Ceilometer

Tento subsystém se používá pro sběr infromací o infrastruktuře cloudu, konkrétně například vytížení fyzických a virtuálních strojů. Tyto informace pak lze použít k vytváření fakturací, pouze za to, co zákazník opravdu využil. Také se dají vytvořit upozornění při přesáhnutí určitého procenta vytížení, které pak může řešit škálovací program nebo administrátor. Kromě základních měřitelných metrik jdou vytvořit i nové, třeba pro získávání dat z vlastních aplikací. Proces, který data shromažďuje, je v základním nastavení ukládá do MongoDB. Použít lze i jiné databáze nebo uložení do souboru a posílání dál po síti protokolem HTTP. Pokud je pro uložení použita jedna z podporovaných databází, lze pak pro získání starších dat použít unifikované REST API. Data jsou ukládána s neklesající granularitou navždy, ovšem u některých databázových úložišť lze nastavit pro data TTL (time-to-live).

Ve webovém rozhraní OpenStacku jménem Horizon pak lze zobrazit, oproti možnostem, ONE pokročilejší grafy jednotlivých metrik. [15] [16]

2.3.2 Monitoring v OpenNebule

ONE poskytuje dvě základní možnosti monitoringu, kde obě mají společné to, že uchovávají data na řídícím serveru, který budu nazývat front-endový. Možnosti jsou následující:

- **UDP-push model**: Lehčí varianta, kde v jednotlivých hostech běží collectd, které periodicky posílá data v UDP paketech řídícímu serveru. Jeho nevýhodou však je, že síť, ve které komunikace probíhá, musí být bezpečná, protože se jedná o nešifrovanou komunikaci.
- **SSH-pull model**: Oproti předchozímu jde o těžkopádnější řešení vhodnější pro menší cloud. V tomto případě se front-endový server periodicky dotazuje sond, např. libvirt v hostech. Výhodou je pak šifrovaná komunikace, ale nevýhodou rychlost získání dat, protože je omezen počet souběžných SSH spojení a kvůli čekání pak jsou nutné i větší intervaly mezi jednotlivými dotazy.

Alternativou k nativním možnostem může být použití systému Ganglia, který má ze strany ONE podporu pro získávání dat, ale jeho konfigurace se musí provádět pro každý stroj zvlášť pomocí úloh v cronu, což z něj dělá zbytečně složitější alternativu k fungujícímu implementovanému systému. Navíc už nativní systém sám sondy má, ty sbírají data podobně jako Ceilometer a jsou rozšiřitelné.

Problémem je, že ONE ukládá tato data ve své interní databázi (SQLite nebo MySQL) v nedokumentovaném formátu po dobu pouze dvanácti hodin. Existuje ale opět API typu XML-RPC, kterého se lze dotázat na aktuální a omezeně i historické údaje. Pro programátorův komfort pak lze použít obalující knihovny v Ruby a Javě. [13]

Poslední alternativou, která by mohla ulevit řídícímu serveru je možnost odchytávat na něm pakety nějakým dalším programem, který je bude posílat dál a nebude využívat robustní XML API OpenNebuly, viz ONESnooper Server[1]. Jelikož však ještě není vyzkoušené, jak

[1] <https://github.com/arax/onesnooper-server>

moc dá užití API front-endovému serveru zabrat, budu používat k získání dat API, což lze protože v CERIT-SC používají monitoring UDP-push modelem.

Nakonec bych chtěl jasně definovat část objektů, jež lze získat z API OpenNebuly a které budu dále využívat:

- **fyzický stroj (host)**: Základní stavební prvek cloudu. Stroj, na němž běží OpenNebula a lze na něm stroje virtualizovat.
- **virtuální stroj (VM)**: Virtualizovaný host, který obecně nemá přímý přístup ke všem hardwarovým prostředkům.
- **datastore**: Úložné médium, které se používá k uchování obrazů disků pro VM. Liší se typy ukládání a možnostmi, jak poskytují data hostům.
- **template (pro VM)**: Šablona pro vytváření VM. Každý VM má svou šablonu a tou jsou definovány jeho parametry, např. CPU, RAM, sítě.
- **uživatel (user)**: Jde uživatelský účet s určitými právy. Může být vlastníkem nějakého virtuálního stroje nebo datastoru.
- **skupina (group)**: Skupina uživatelů.
- **cluster**: Skupina hostů a datastorů.

2.4 Požadavky na monitorovací systém

V následující sekci uvádím obecné a funkční požadavky zadavatele.

2.4.1 Obecné požadavky

1. Ukládat data monitoringu po delší dobu, optimálně tři roky (minimálně jeden rok) s klesající granularitou
2. Uchovávat data i pro už neexistující stroje
3. Nad databází musí existovat agregace starších dat
4. Systém musí být škálovatelný. Předpokládá se rostoucí objem dat
5. Možnost přidání skriptů pro získávání dat z jiných cloudových platforem, a tedy i zobrazení grafů z nich
6. Dokumentované API poskytující historická data
7. Implementace v jazyku Ruby
8. Použité nástroje musí být open-source

2.4.2 Funkční požadavky

1. Možnost čerpání dat z více OpenNebula instancí
2. Uložená data v určitém intervalu se musí dát manuálně smazat pro zmenšení zabraného prostoru
3. Zobrazovat informace o počtu běžících virtuálních strojů
4. Vlastnosti zobrazení grafů musí být nastavitelné, např. výběr časového intervalu
5. Podporovat grafy pro VM, hosty, datastory a clustery
6. Podporovat následující filtrovací podmínky pro zobrazení grafů:
   (a) Pro VM - ID, jméno, ID hosta, jméno hosta, ID clusteru, jméno clusteru, ID uživatele, jméno uživatele, ID skupiny, jméno skupiny
   (b) Pro Hosta - ID, jméno, ID clusteru, jméno clusteru, ID používaných datastorů
   (c) Pro Datastore - ID, jméno, ID clusteru, jméno clusteru, ID uživatele, jméno uživatele, ID skupiny, jméno skupiny, ID používajících hostů, transfer manager, typ
   (d) Pro Cluster - ID, jméno
7. Překládání grafů několika metrik v jednom obrázku
8. V grafech musí být možnost zobrazit tyto metriky:
   (a) Pro fyzické stroje:
      i. Volné a obsazené místo v datastoru
      ii. Alokované a dostupné zdroje CPU/RAM
      iii. Využité a dostupné zdroje CPU/RAM
      iv. Využité a alokované zdroje CPU/RAM
   (b) Pro virtuální stroje platí vše, co pro fyzické a navíc:
      i. Poslaná data do sítě
      ii. Přijatá data ze sítě

Kapitola 3

Návrh

V produkčním prostředí zadavatele mají momentálně více než 200 běžících fyzických strojů a virtuálních v řádů stovek až nízkých tisíců, kde obě čísla budou s postupem času růst. Jelikož chceme data ze všech serverů získávat v co nejkratších časových intervalech, tak se nejedná o triviální úkol a svým rozsahem by vydal na samostatnou bakalářskou práci.

Naštěstí, jak jsem zmínil v 2.3 Možnosti monitoringu v privátních cloudech mají OpenNebula i OpenStack systém pro centralizované shromažďování dat o monitoringu, přístupná přes API. Shromaždování dat mám tedy vyřešené a stačí vymyslet, jak uchovat a vizualizovat tato data.

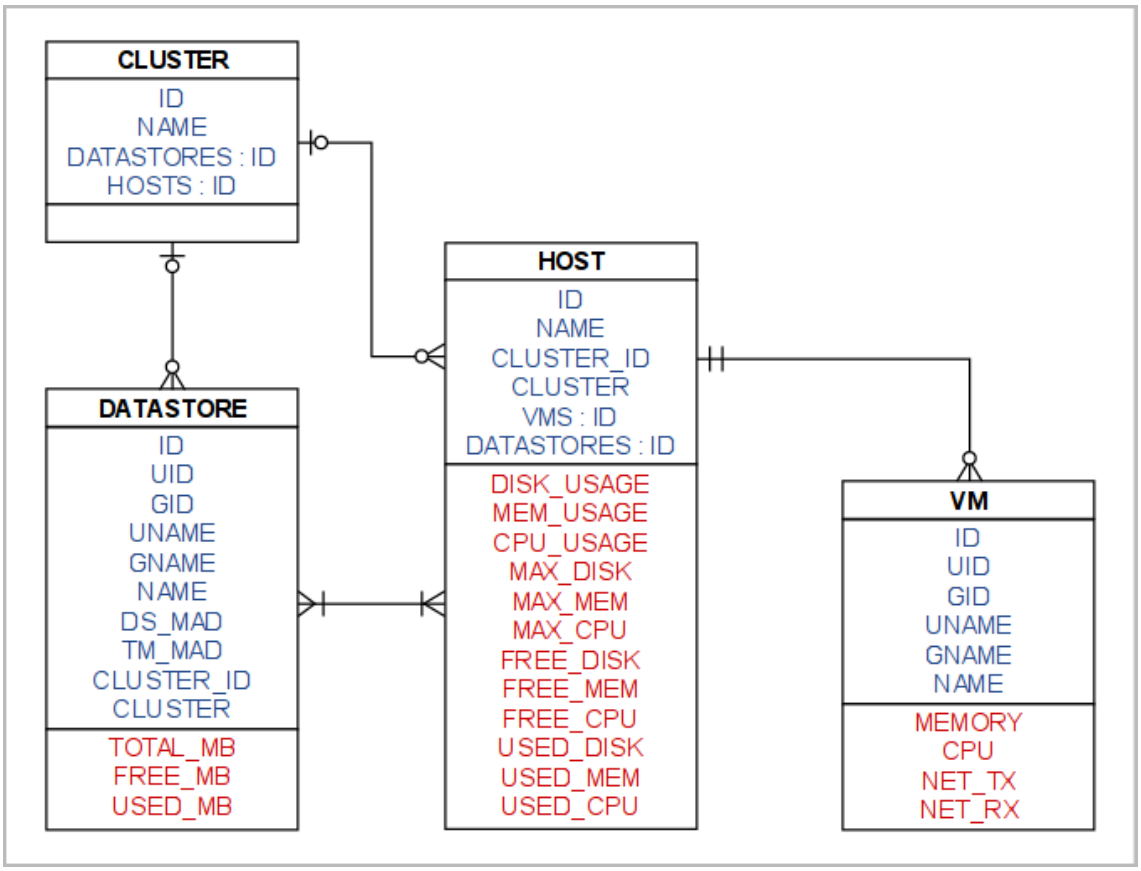
V této kapitole se tedy budu věnovat návrhu především těchto dvou částí a to pouze pro případ OpenNebuly. OpenStack má obdobné API a data se liší víceméně pouze názvy a formátem, proto pak stačí data ukládat ve formátu shodném s OpenNebulou a výsledek práce může být použit pro oba cloudy.

Ještě na začátek bych rád zavedl několik pojmů:

- **downsampling**: Funkce, která dokáže převést body s danou granularitou na hrubší. Uvádím příklad, kde jsou dvě časové série se záznamy každou minutu (série A) a deset minut (série B): Pomocí nějaké agregační funkce (percentil, průměr) se downsamplingem z deseti bodů série A spočítá hodnota jednoho nového bodu v sérii B.
- **OpenNebula objekt**: Stavební prvek OpenNebula cloudu, který lze získat z ONE XML-RPC API.
- **tag**: Je popisek, neboli filtrační kritérium, které identifikuje jakému objektu naměřená metrika patří a jaké jsou další parametry objektu. Kromě ID se však v libovolném časovém okamžiku může změnit, proto je potřeba jej zaznamenat ke každému bodu v čase.

3.1 Ukládání dat

V obrázku 3.1 jsou zobrazeny vztahy mezi jednotlivými hlavními objekty monitoringu v ONE. Modře jsou označeny tagy a červeně jsou označeny metriky, které se pro příslušný objekt měří.



Obrázek 3.1: Schéma objektů a jejich atributů, které lze získat z API OpenNebuly

Potřebuji tedy ukládat časové body, které reprezentují hodnotu nějaké metriky a navíc jsou k nim uloženy tagy, reprezentující vztahy mezi objekty. Co se pro takové potřeby skvěle hodí jsou time series databáze. V této podkapitole bych se proto rád věnoval jejich přehledu, který bylo nutné vytvořit, abych mohl vybrat nejlepší možné řešení. Ukládání do souborů a implementaci čistě vlastních řešení jsem vynechal. Agregační funkce, vyhledávání apod. pracující v nějakém důstojném čase není triviální naprogramovat a navíc u stávajících projektů je větší pravděpodobnost toho, že se o ně někdo bude starat i po ukončení mé práce. Pokud se nezajímáte o detaily jednotlivých řešení, v tabulce 3.1 je pak porovnání těch nejvhodnějších současných time series databázových nástrojů.

3.1.1 OpenTSDB

OpenTSDB je jeden z nejpoužívanějších nástrojů umožňujících shromažďování a ukládání velkého množství časových bodů, které mohou nést informace o naměřených hodnotách.

Na serverech, ze kterých chce získávat údaje, běží jeho klienti, kteří v intervalech posílají naměřené hodnoty sběrným démonům (TSD) a ti už se starají o ukládání dat v NoSQL databázi HBase.

Tento systém je výhodný hlavně svou škálovatelností, protože lze nasadit tolik TSD, kolik

může být poskytnuto serverů. Bohužel tato vlastnost mi zde moc nepomůže, protože zdrojem jsou instance cloudů a ta je momentálně jedna. Navíc aktuální verze 2.1.0 doporučuje ukládat pouze kolem 6-7 tagů k časovému bodu, než dojde ke zhoršení rychlosti vyhledávání a tohle množství nestačí. [14]

Další důležitou vlastností je, že data ukládá s neklesající granularitou, až pokud chcete data přečíst z HTTP API, můžete použít funkcí pro downsampling k rychlejšímu vykreslení dat. Hrubší data existují pouze v okažiku dotazu a při dalším dotazu se výpočty budou provádět znovu. [23] [29]

3.1.2 SQL databáze

Řešení s použitím relačních databází MySQL či PostgreSQL, asi jako jediné dovoluje jednoduše uchovat many-to-many vazbu mezi hosty a datastory, bohužel znamená také návrh složitého databázového schématu.

Možností je například jedna tabulka pro jednu metriku, jednoho objektu v jeden časový interval. Záznamy budou ve dvou sloupcích: čas a hodnota metriky. Počet tabulek jsou pak kombinace přes všechny metriky, objekty a granularity. To je velké číslo a měřené metriky i objekty se s postupem času mohou měnit, což by znamenalo upravovat tohle komplexní schéma. Navíc je potřeba zaznamenat všechny tagy. To lze například přidáním všech příslušných tagů do jména tabulky ve tvaru `název_tagu=hodnota`, kde pro vybrání správných bodů pak stačí vybrat všechny tabulky, které by pasovaly regulárnímu výrazu pro výběr určených hodnot tagů. Nakonec bych ještě musel implementovat vlastní funkci pro převedení na hrubší granularitu.

3.1.3 RRDtool

Název tohoto nástroje je akronymem round-robin database tool a jde o databázový systém s předem zadanou velikostí. Funguje tak, že když se zapíše nový bod a archiv je plný, přepíše se bod nejstarší. Jednotlivé databáze se ukládají do souborů, kde jeden může obsahovat několik měřených zdrojů/metrik, například velikost poslaných dat do sítě. Databáze navíc očekává, že nová data se zapíší v předem zadanou periodu a pokud ne, zapíše se že hodnota není známá, aby nedošlo ke zkreslení ve funkcích pracujících s těmito daty. Jeho předností je podpora agregačních funkcí, které spočítají například průměr z nejnovějších hodnot v zadaném intervalu. RRDtool není jen back-end, ale lze jím i vykreslovat grafy. [26]

3.1.4 Graphite

Je podobně jako OpenTSDB projekt složený z několika nástorů:

- démon carbon, kterému se posílají nová data
- databáze whisper na systému RRD
- webová aplikace pro vykreslování grafů

Graphite v podstatě vychází z projektu RRDtool. Liší se hlavně tím, že RRDtool očekává data v předem daných intervalech, kdežto Graphite je zvládá ukládat i v nepravidelných. Dalším rozdílem je pak to, že Graphite je modernější aplikace napsaná Pythonu, což jej i přes všechny optimalizace činí pomalejším než RRDtool, který je napsaný v C. Kvůli lepším možnostem vizualizace zde však pro mě má nakonec větší hodnotu než RRDtool. [4]

3.1.5 Druid

Druid je pokročilý systém pro ukládání časosběrných údajů v reálném čase. Místo měření metrik se spíše používá na zaznamenání různých logů nebo třeba měření počtu kliknutí na reklamy. Uvádím ho jako zástupce ze skupiny analytických nástrojů pro big data, které by se zde daly použít.

Jeho asi největší výhodou je rychlost zápisu i čtení. Pro zvýšení dostupnosti je celý systém možno rozložit na několik serverů a přesto bude data poskytovat ihned, co jsou dostupná v cachi na některém z uzlů. Jeho komplexnost je však i jeho slabinou, protože se skládá z několika v podstatě samostatných částí a každá se může rozbít. Některé systémy sice mohou vypadnout a podstatná část bude kvůli dobré implementaci fungovat, nicméně platí, že více pohyblivých částí je větší riziko.

Komplexní systém také znamená složitější konfiguraci a údržbu. Také přímo k němu neexistuje žádný nástroj pro vizualizaci uložených dat a neposkytuje vhodnou funkci pro downsampling, která by šla zautomatizovat.

Nakonec bych pak zmínil že mimo rychlosti disponuje i možností přidat k datovým bodům neomezené množství tagů. [30]

3.1.6 InfluxDB

InfluxDB je time series databáze v nejnovější verzi postavená nad BoltDB. Její předností je oproti ostatním jednoduchý dotazovací jazyk, který vychází ze syntaxe SQL, i když databáze nevyžaduje předem určené schéma. Skupiny bodů se shromažďují v měřeních a existují funkce, které z měření automaticky vytváří nové s hrubší granularitou. Největším nedostatkem InfluxDB je, že tagy jsou implementované až v nejhovější verzi 0.9, která se momentálně ještě testuje. Tato verze by však měla být hotová v co nejbližší době. Potřebné funkce jsem vyzkoušel a ve většině případů byly funkční, proto to nepovažuji za závažný nedostatek. [9]

3.1.7 Blueflood

Systém vytvořený v Rackspace ke zpracování metrik generovaných jejich managed cloudem. Blueflood je napsán v Javě nad NoSQL databází Cassandra a podporuje automatický downsampling. Bohužel projekt, alespoň navenek, z pohledu na jejich repozitář na GitHubu, není nijak aktivní. Navíc nepodporuje tagy. [1]

3.1.8 NoSQL databáze

Někteří používají pro ukládání i samostatnou NoSQL databázi. Znamená to však návrh vlastního schématu, které se podobně jako u SQL řešení může měnit. Oproti klasickým SQL

databázím by se však měly chovat lépe pro větší množství dat, ve smyslu rychlejšího přístupu a jdou i lépe škálovat. Nakonec je zde tedy potřeba podobné množství práce jako u SQL. Konkrétní případ možné implementace neuvádím, protože NoSQL databáze se od sebe silně liší a jeden postup nelze aplikovat na všechny.

3.1.9 Prometheus

Prometheus je nástroj pro monitoring, který vyvinuli v SoundCloudu. Body jsou zde ukládány vícedimenzionálně, tedy každá metrika může mít více tagů (dimenzí). Není to ale jen databáze, je to celý monitorovací systém. Od ostatních se tak liší hlavně přednostním použitím pull modelu, který jsem zmínil v 2.3.2 Monitoring v OpenNebule. V nastavené periodě se dotazuje koncových bodů, na kterých běží jeho klientské knihovny. Ty pak lze rozšířit tak, aby vracely hodnoty požadovaných metrik. Jako bonus má v sobě zabudované pěkné prostředí pro vykreslování grafů. Hlavní nevýhodou pak je, že nepodporuje downsampling. [18] [19]

3.1.10 Další řešení

Existuje více time series databází a zde ještě heslovitě dvě uvedu, aby bylo jasné, že jsem se zabýval i jimi. Pro naše použití se však nehodí.

- **Newts**: Menší projekt postavený nad Cassandrou, umí si vypočítat downsampling, ale neumí ho uchovat.
- **KairosDB**: Fork OpenTSDB podporující jako back-end mimo HBase i H2 a Cassandru.

3.1.11 Shrnutí

Do uzšího výběru jsem nakonec zvolil OpenTSDB, jako zažitý standard, InfluxDB, Graphite a Prometheus. V tabulce 3.1 si můžete prohlédnout, jak obstály proti kritickým požadavkům. Je patrné, že nejlepší volba je InfluxDB, i když je ještě ve vývoji.

	OpenTSDB	InfluxDB	Graphite	Prometheus
Horizontální škálovatelnost	ANO	ANO	NE	ANO
Automatický downsampling	NE	ANO	ANO	NE
Podpora většího množství tagů	NE	ANO	NE	ANO
Nástroje pro vykreslování grafů	ANO	ANO	ANO	ANO

Tabulka 3.1: Plnění kritických požadavků u výběru nejslibnějších projektů

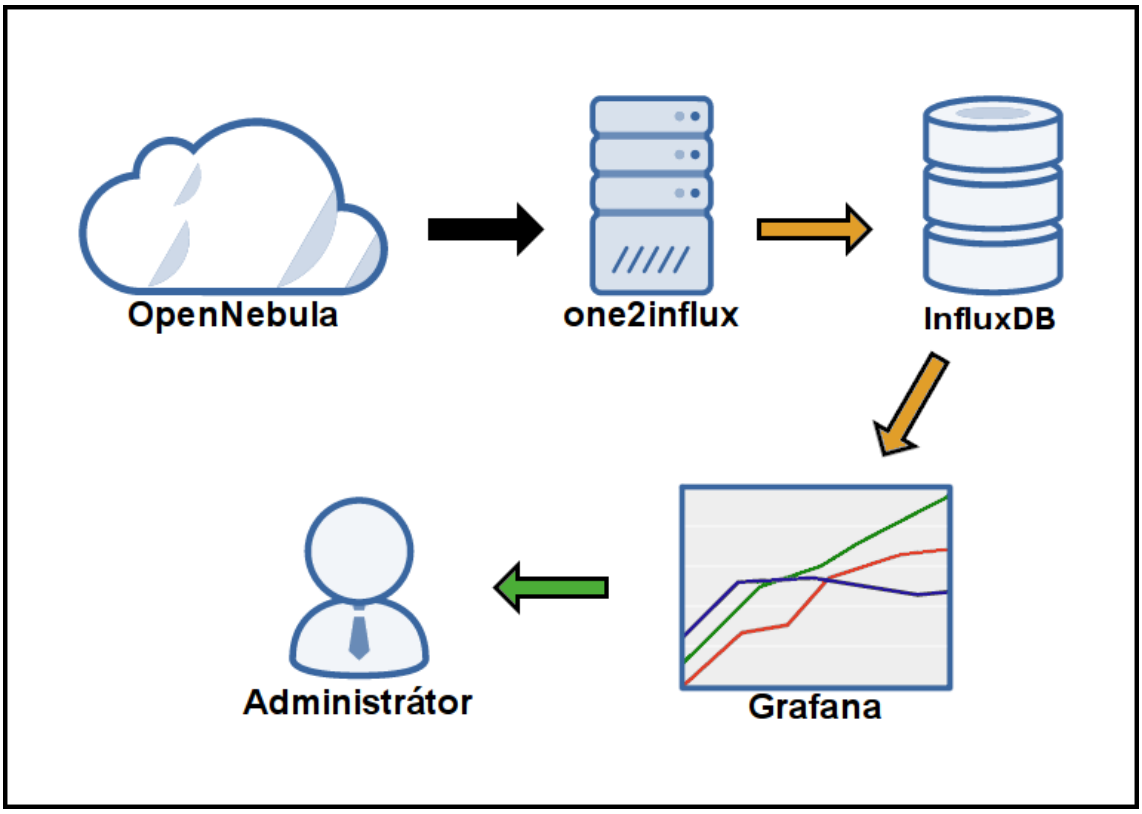
3.2 Model nasazení

V obrázku 3.2 je zachycen model nasazení. Prostředníka mezi existujícími řešeními tvoří démon one2influx, kterého jsem naprogramoval. Získává data z ONE API a ukládá je do InfluxDB, odkud si je může nahrát Grafana pro vykreslení grafů. Šipky znázorňují, jestli může být spojení šifrované nebo ne. Zelená znamená ano. Oranžová je spojení s InfluxDB, které jsem otestoval a momentálně nefunguje, ovšem v nejbližších verzích je plánovaná jeho podpora, viz diskuze[1] na GitHubu. OpenNebula API SSL nepodporuje, proto černá šipka.

Jednotlivé komponenty lze spustit na samostatných serverech. Pokud bude komunikace probíhat v zabezpečené síti, nejlepší je asi řešení, kde všechny tři komponenty (démon, databáze i Grafana) běží na jednom serveru. Jestliže by zabezpečená síť nebyla k dispozici, pak by se pouze démon přesunul na front-endový server OpenNebuly.

Podrobnější informace o jednotlivých komponentách uvedu v následujících podkapitolách.

[1] <https://github.com/influxdb/influxdb/issues/2115>



Obrázek 3.2: Model monitorovacího systému

3.3 Podrobněji o InfluxDB

Zde bych rád nastínil architekturu a zásadní funkce InfluxDB, pro které jsou klíčové tyto pojmy:

- **Databáze**: Největší správní celek, podobný pojmu databáze z MySQL. Slouží k omezení přístupu jednotlivým uživatelům a omezení oblasti působení ostatních funkcí.
- **Měření**: Označení měřené metriky.
- **Série**: Kombinace měření a skupiny tagů.
- **Retention policy**: Část databáze s pravidlem, které zajišťuje funkci round robin archivu z RRDtool. Při vytváření se určí časový interval, jméno a replikační faktor (použitelné pro cluster režim) a všechny body starší než povolený interval se odtud vždy automaticky odstraní. Čas vynucení retention policy lze nastavit. [7]
- **Continuous query**: Jsou pojmenované dotazy, které se po zavedení počítají automaticky a lze je přesměrovat do nové série i v jiné retention policy. Zde je budu používat pouze pro downsampling. To nejlépe vysvětlím na následujícím příkladu, kde CQ jménem `cq101` fungující nad databází `db`, které z retention policy jménem `one_day` vybere vždy každých sedm minut nejnovější data. Z nich vytvoří průměr a přidá k nim tagy

které byly v sérii `one_day` a mají se zachovat a tento nový bod vloží do retention policy `week_day` a série se stejným jménem jako výchozí, tedy `MEM`. [8]

```
1 CREATE CONTINUOUS QUERY "cq101" ON "db"
2 BEGIN
3   SELECT mean(value) AS value INTO "db"."one_week"."MEM"
4     FROM "db"."one_day"."MEM"
5     GROUP BY time(7m), CLUSTER_ID, CLUSTER_NAME, HOST_ID
6 END;
```

- **HTTP API**: Jde o nejvíce nízkoúrovňový způsob komunikace s InfluxDB. Používají se HTTP metody POST pro zápis a GET pro čtení. Tato funkce mi řeší požadavek na API poskytující historická data.
- **Webové uživatelské rozhraní**: Jde o uživatelsky přívětivější způsob spojení s InfluxDB. Lze zde například vytvářet nové databáze nebo retention policy a provádět dotazy prostřednictvím dotazovacího jazyka InfluxDB. Výsledky se zde oproti základnímu JSON formátu zobrazí v tabulce.
- **Cluster režim**: Jde o konfiguraci InfluxDB pro několik serverů, kvůli zvýšení výkonu pro zápis i čtení a zvýšení dostupnosti. Už v minimálním nastavení pro tři servery, lze ustát chybu jednoho, bez celkového výpadku služby. [6]
- **Zálohování**: Je možné klasicky pro veškerá data na aktuálním systému nebo inkrementální, které uloží pouze změnu od poslední zálohy. [5]

3.4 Podrobněji o ukládání

Kvůli zbytečnosti uchovávat starší data v původním rozlišení, bych zde rád probral téma vrstvy granularity. Každá vrstva odkazuje na jeden časový interval a o čím delší interval jde, tím hrubší je granularita. Následuje příklad vrstev které by se daly použít, ale ve finále, bych chtěl nechat tuto volbu na administrátorovi:

- den se záznamy po jedné minutě
- týden se záznamy po sedmi minutách
- měsíc se záznamy po půl hodině
- rok a déle se záznamy po šesti hodinách

Pro tento příklad jsem vycházel jsem z první granularity, která by měla být pro monitoring dostačující a potřebuji na ni zaznamenat 1440 bodů. Zbylé granularity jsem dopočítal tak, aby při význačných časových intervalech (týden, měsíc, rok) obsahovaly stejný počet bodů.

Každé vrstvě granularity pak vytvořím retention policy. Kromě poslední, která si uchová data navždy. Pro manuální mazání je možné přidat nakonec další vrstvy, např. pro dva a tři roky. InfluxDB momentálně podporuje, pro tento projekt vhodné, mazání v časovém

intervalu pouze pro série v retention policy a tak pak můžu smazat pouze celé vrstvy, což by u těch dvou a tří letých nemuselo vadit.

Zapisovat pak budu data každou minutu do první retention policy a nastavím CQ pro každou měřenou metriku tak, aby se data automaticky přelévala z jedné retention policy do druhé.

3.5 Vykreslení grafů

Zvolením InfluxDB jsem si tuto část značně usnadnil, protože lze použít vizualizační nástroj Grafana.

Grafana je uživatelské prostředí pro vykreslení dat z OpenTSDB, Graphite a InfluxDB. Bylo původně psané pouze jako webová aplikace, kde byl pro její provoz potřeba webový server, například Apache. Pro uchování nastavení grafů se musela použít nějaká externí databáze, třeba InfluxDB. Aktuální verze 2.0 je však přepsaná v Go, s vlastním webovým serverem i databází. [2] [3]

Prvním krokem k použití je zadání zdroje dat. Poté už lze v prostředí vytvořit vlastní panel (v terminologii Grafany dashboard), kde lze na řádky umístit grafy. Co se na grafu zobrazí se nastavuje pro InfluxDB buď přímo jejich dotazovacím jazykem nebo Grafana podporuje výběr na vyšší úrovni, kde zkrátka naklikáte, co za metriky, v jakém čase a s jakými podmínkami se má zobrazit. Tato více vizuální a člověku přirozenější možnost však omezuje možnosti přesného určení zobrazených dat. [3]

Veškerou konfiguraci dashboardů lze exportovat a importovat ve formátu JSON, takže lze dobře naskriptovat prostředí pro uložené metriky, aniž by ho člověk musel nastavovat celé ručně.

3.6 one2influx

One2influx je démon, kterého jsem napsal v jazyce Ruby. Periodicky se dotazuje ONE API na aktuální hodnoty z monitoring systému. ONE API sice vrací i hodnoty za posledních 12 hodin, takže by šel použít i přístup, kdy se perioda dotazů místo řekněme základní jedné minuty změnila na 12 hodin. Z historie však nejdou vyčíst všechny informace (například příslušnost datastoru ke clusteru), a proto volím možnost získávání aktuální hodnoty.

Mým cílem bylo umožnit administrátorovi výběr přesně toho, co chce sledovat. Proto má one2influx podrobné nastavení, kde lze povolit nebo zakázat všechny důležité metriky. Více o implementaci v další kapitole.

Kapitola 4

Implementace

V této kapitole uvedu podrobnosti o tom, jak probíhala implementace tohoto projektu, od nastavení prostředí pro vývoj, přes naprogramování démona, konfiguraci Grafany, až po testování a nalezené problémy.

4.1 Implementační prostředí

Přístup přímo k OpenNebula API v CERIT-SC jsem pochopitelně nedostal, ale naštěstí ONE poskytuje ke stažení předkonfigurovaný virtuální disk pro virtualizační nástroj VirtualBox, který byl pro mé potřeby naprosto postačující. Virtuální OpenNebulu ve verzi 4.12 jsem měl spuštěnou u sebe na notebooku s operačním systémem Arch Linux.

Metacentrum, které s CERIT-SC spolupracuje, mi poskytlo uživatelský přístup k OpenNebule, kde jsem si vytvořil virtuální stroj pro InfluxDB a Grafanu k testování. Jako operační systémy jsem zde používal Debian ve verzích 7.8 a 8, protože Grafana i InfluxDB poskytují balíčky pro Debian a instalace je pak opravdu jednoduchá.

Jelikož, jak už jsem zmínil v 3.2 Model nasazení, InfluxDB zatím nemá implementováno SSL, tak jsem vždy při testování vytvořil pomocí SSH tunely. Ty port forwardingem mapovaly porty pro přístup a zápis do InfluxDB na porty u mě v notebooku přístupné pouze z localhostu. Porty InfluxDB jsem pak firewallovým programem iptables na vzdáleném serveru zablokoval, aby mi nikdo nepověřený nemohl mazat data, protože při četném restartování a mazání instalace jsem často používal přednastavené přihlašovací údaje.

4.2 Implementace one2influx

K implementaci démona jsem vybral požadovaný jazyk Ruby ve verzi 2.2.2p95, ale ke spuštění stačí jakákoliv verze vyšší než 2.0.0. Protože jsem byl v době psaní bez předchozích zkušeností s Ruby, čerpal jsem základy z knih The Well-Grounded Rubyist [21] a Eloquent Ruby [27]. K nim jsem využíval oficiální online dokumentaci [20], abych držel krok s aktuální verzí 2.2.2. Komentáře byly psány pro nástroj YARD[1], který slouží ke generování dokumentace z komentářů ve zdrojovém kódu. K vývoji jsem dále používal verzovací nástroj

[1] <http://yardoc.org/>

git a změny nahrával na hosting git repozitářů GitHub, kde lze zdrojové kódy one2influx i s dokumentací dohledat[2].

Celý program je napsán jako gem, což je jednotný formát knihoven v Ruby, zjednodušující instalaci a správu závislostí. Funkční verzi jsem umístil na hosting gemů RubyGems.org, kde je veřejně dostupný[3].

V pseudokódu ve výpisu 4.1 je shrnuta v jednoduchosti funkcionalita programu.

```
1   načti konfiguraci
2   otestuj spojení s ONE
3   otestuj jestli databáze a retention policy existují
4
5   if nepovedlo se cokoliv z výše uvedeného
6    ukonči program a vypiš chybu
7   end
8
9   while true do
10   získej data z ONE
11   if nepodařilo se získat data
12    zkus to znovu, maximálně ještě 4x
13   end
14
15   ulož data v InfluxDB
16   if nepodařilo se uložit data
17    zkus to znovu, maximálně ještě 4x
18   end
19
20   uspi program na zadaný interval
21  end
```

Výpis 4.1: Pseudokód funkcionality one2influx

Program lze rozšířit nebo upravit na sbírání jiných metrik resp. jejich poměrů. Ve třídě `Config` (4.2.2.1) lze nastavit kromě základních údajů pro funkčnost i které metriky budou sbírány a které tagy přiřazeny. Navíc jsou třídy, které dědí z `OneObject`, implementovány tak, aby do nich šly přidat nové funkce. Pak je možné snadno tvořit například poměry mezi metrikami a tedy metriky nové.

4.2.1 Použité gemy

Zde je pouze výčet použitých knihoven. Výhodné by bylo i využití gemu pro usnadnění komunikace s InfluxDB. Takové sice existují, ale ještě nejsou kompatibilní s verzí 0.9. Nicméně komunikace s HTTP API není nijak složitá.

[2] <https://github.com/zidekmat/one2influx>
[3] <https://rubygems.org/gems/one2influx>

4.2.1.1 Pro běh

- **json**: Knihovna pro vytváření řetězců ve formátu JSON z Ruby asociativních polí. Potřebná, protože InfluxDB přijímá nové body v tomto formátu.
- **opennebula**: OpenNebula XML-RPC API nadstavba pro použití v Ruby kódu.
- **nokogiri**: Knihovna ke zpracování XML.

4.2.1.2 K testování

- **gyoku**: Knihovna pro vytváření XML, použitá ke generování falešných údajů z OpenNebuly k testování.
- **rubystats**: Knihovna obsahující metody pro generování normálního rozdělení.

4.2.2 Třídy

Zde uvedu k čemu slouží jednotlivé třídy one2influx. Více lze najít v komentářích přímo ve zdrojovém kódu.

4.2.2.1 Config

Tato třída je v programu doslupná přes globální proměnnou `$CFG`, která obsahuje její instanci. Pro správný běh programu je potřebné nastavit přístupové body k ONE API a InfluxDB, ostatní nastavení není potřeba měnit.

Ve výpisu 4.2 je pak, jako příklad nastavení zaznamenávaných metrik a tagů, konfigurace pro virtuální stroj. Přidání či odebrání monitorované metriky nebo tagu se provádí odkomentovaní resp. zakomentováním příslušného řádku.

Tagy získané z XML-RPC API by v databázi nebyly jedinečné, protože každý objekt obsahuje např. atribut NAME. Proto jsou tagy v nastavení reprezentovány asociativním polem, kde klíč je jméno tagu, které se uloží do databáze a hodnota je jméno tagu v XML získaného z ONE. Klíče lze měnit podle přání uživatele. Tag `ID` u žádného z objektů není uveden, protože jde o tag povinný a nesmí se zakázat.

Speciální tagy `DSS_IDS` a `HOST_IDS` jsou realizací many-to-many vazby mezi hosty a datastory. `DSS_IDS` obsahuje ID všech datastorů, které tento host používá. Jedná se o řetězec ve tvaru: `,,ID_1,,ID_2,,...,,`, protože InfluxDB podporuje jednoduché regulární výrazy pro filtrování tagů a lze pak například filtrem `WHERE DSS_IDS='.*,5,.*,7,.*'` zobrazit nějakou metriku hosta, který používá datastory s ID 5 a 7. `HOST_IDS` se chová stejně, jsou to ID všech hostů, kteří používají datastore.

U metrik nebylo asociativní pole potřeba, protože ty jsou už tak jedinečné v rámci celé databáze. Je tedy použito normální pole.

Zvláštní skupinu pak tvoří ještě pole `cust_metrics`, které obsahuje názvy nových metrik. Například v 4.2 je metrika `MEMORY_PERC`, která udává percentuální využití RAM. Podobné nové metriky lze implementovat v jednotlivých třídách dědících z `OneObject`. O nich více v 4.2.2.4.

```
1 # Virtual machine
2 vm:    {
3  tags: {
4   CLUSTER_ID: 'CLUSTER_ID',
5   CLUSTER_NAME: 'CLUSTER_NAME',
6   HOST_ID: 'HOST_ID',
7   HOST_NAME: 'HOST_NAME',
8   VM_NAME: 'NAME',
9   UID: 'UID',                # [int] user's ID
10  GID: 'GID',                # [int] group's ID
11  UNAME: 'UNAME',            # [string] user's name
12  GNAME: 'GNAME',            # [string] group's name
13  #VM_STATE: 'STATE',        # [int] virtual machine state
14  #LCM_STATE: 'LCM_STATE' # [int] substates for ACTIVE state
15  },
16  metrics: [
17   'MEMORY', # [kB] memory consumption
18   'CPU',    # [%] 1 VCPU consumed
19                          #   (two fully consumed cpu is 200)
20   #'NET_TX', # [B] sent to the networ
21   #'NET_RX'  # [B] received from the network
22  ],
23  cust_metrics: [
24   #'MEMORY_PERC' # Computes percentage usage of memory for VM
25  ]
26 }
```

Výpis 4.2: Konfigurace tagů a metrik pro virtuální stroj

4.2.2.2 Data

Třída `Data` slouží pouze k zapouzdření procesu získání dat z OpenNebuly a uchování všech bodů, které se mají v aktuální interval nahrát.

4.2.2.3 Influx

Třída pro komunikaci s InfluxDB. Ověřuje existenci databáze s retention policy a ukládá nové body do databáze. Body jsou kvůli lepšímu výkonu rozděleny na dávky po 2500 bodech a každá se v případě chyby pokusí uložit ještě čtyřikrát.

4.2.2.4 OneObject

Generalizace OpenNebula objektů. Slouží k získávání dat z XML dokumentů, přepisuje tagy z XML na jména v konfiguraci, získává zavedené i nové metriky.

Nové metriky jsou implementovány ve funkcích, které začínají `get_` a následuje název nové metriky. Například tedy: `get_MEMORY_PERC`. Kromě konvence pro pojmenování také musí vracet hodnotu metriky.

4.2.2.5 Host

Dědí z `OneObject` a od ostatních se liší tím, že má speciální tag `DSS_IDS`, určený pro filtrování regulárními výrazy.

4.2.2.6 Datastore

Dědí z `OneObject` a podobně jako `Host` implementuje metodu pro speciální tag `HOSTS_IDS`.

4.2.2.7 Cluster

Dědí z `OneObject` a od ostatních se liší tím, že jeho XML reprezentace získaná z OpenNebuly neobsahuje žádné metriky, pouze ID hostů a datastorů. Proto je zde navíc při vytváření instance přijímán parametr obsahující všechny hosty, ze kterého se vyberou ty náležící aktuálnímu clusteru a z nich se jako součty spočítají metriky výkonu celého clusteru.

4.2.2.8 VirtualMachine

Dědí z `OneObject` a jeho XML reprezentace z ONE neosahuje příslušnost ke clusteru a hostovi, proto se při inicializaci předává ještě jako další parametr host, na kterém virtuální stroj běží a odtud se získají tagy jméno a ID pro hosta a cluster.

4.2.2.9 FakeOne

Třída určená k testování. Nahrazuje spojení s ONE a generuje náhodné XML reprezentace OpenNebula objektů. Počty obsažených objektů jde nastavit v konstruktoru třídy.

4.2.2.10 FakeData

Třída určená k testování, jde o úpravu třídy `Data`, aby byla schopná fungovat bez spojení s reálným ONE XML-RPC API a místo toho používala `FakeOne`.

4.3 Konfigurace InfluxDB

InfluxDB před spuštěním není potřeba nijak zvlášť nastavovat. Jediná pro vývoj zajímavá možnost byla nakonfigurovat cluster režim, pro zvýšení počtu zapsaných bodů za sekundu a rychlejší odezvy při dotazování. Bohužel dokumentace k tomuto nastavení ještě plně neexistuje a sám jsem jej zprovoznit nedokázal.

Bylo ale potřeba provést nastavení databáze, což je ale jednoduše proveditelné přes webové GUI. Databázi jsem vytvořil a přiřadil k ní retention policy podle příkladu vrstev granularity z 3.4 Podrobněji o ukládání.

Pak už jen zbývalo nastavit mezi nimi CQ, které se však ukázaly jako ne úplně funkční, a proto jsem je ne vždy využíval, více v 4.6.

4.4 Konfigurace Grafany

Grafanu jsem po většinu času používal v základním nastavení, jen jednou jsem otestoval SSL a to nečiní problém. Konfigurace pro zobrazení grafů je už poměrně intuitivní záležitost, která se ale musí odklikat. Nakonec jsem si vystačil s jedním dashboardem na obrázku 4.1 s poměrně nezajímavými, ale pravdivými grafy o zátěži mé testovací OpenNebuly ve VirtualBoxu. Dashboardy lze nahrávat ze souborů a tento konkrétní jsem přidal na přiložené CD.

Obrázek 4.1: Ukázkový dashboard v Grafaně

4.5 Testování

Testování jsem prováděl s InfluxDB ve verzích 0.9RC25 až 0.9RC29 a Grafanou 2.0.0 až 2.0.2. Protože je program zamýšlen jako démon, tak nevypisuje chyby na standardní chybový výstup, ale do souboru `one2influx.log`, jehož umístění lze nastavit v konfiguraci. Logy navíc každý den začnou s prázdným souborem stejného jména a záznamy z předchozího dne uloží do nového souboru, kde název má za příponu datum předchozího dne.

4.5.1 Test funkčnosti

Pro odladění funkčnosti jsem využíval virtualizovanou OpenNebulu (viz 4.1), kde byl nastavený jeden fyzický a dva virtuální stroje. Continuous query jsem nastavil z počátku pouze pro jednu metriku a ukázalo se, že i tak fungují jak se jim zachce, proto jsem je odstranil. Toto nastavení, kde se data ukládala v podstatě pouze jeden den jsem měl spuštěné nepřetržitě 5 dní a nevyskytly se žádné problémy. Zápis i čtení probíhaly v řádech maximálně stovek milisekund. One2influx by se i při fungujících CQ choval stejně, proto se domnívám, že stačí počkat až na oficiální vydání verze 0.9, které by mohlo být během června/července. [22]

4.5.2 Test se zátěží

Abych přiblížil podmínky provozu ostrého nasazení, potřeboval jsem ve své virtuální OpenNebule více fyzických a virtuálních strojů. ONE ve VirtualBoxu však má své limity, a tak jsem navrhl třídu, která generuje falešné XML reprezentace pro nastavitelný počet ONE objektů. Testování jsem tak prováděl pro 300 hostů, 600 VM, 500 datastorů, 100 clusterů, 300 skupin a 600 uživatelů.

Díky falšování výstupu OpenNebuly jsem mohl přenést one2influx na server, kde běží InfluxDB a Grafana. Opět jsem vynechal CQ a data tak nahrával pouze do první vrstvy granularity. Tady ovšem byly výsledky mnohem horší. Testování jsem prováděl na stroji s 2GB RAM a 1GB swapovacím oddílem a už po přibližně šestnácti hodinách byla paměť i swapovací prostor serveru kompletně vytíženy a tedy celková odezva serveru byla mnohonásobně delší. Na InfluxDB se to projevilo tak, že čtení i zápis měly odezvy v minutách a většina zápisů se ani neprovedla. Úspěšnější jsem nebyl ani s přidáním RAM na 4GB, až zvýšení na 8GB pomohlo a zápisy byly opět ve stovkách milisekund, složitější dotazy maximálně v jednotkách sekund a to i po déle než jednom dni.

O tomto stavu jsem napsal zprávu[4] a umístil ji na git repozitář InfluxDB. Žádné odezvy jestli jsou to reálné systémové nároky nebo jestli se jedná o nějakou chybu se mi zatím nedostalo.

4.6 Problémy při testování

Jelikož InfluxDB z verze 0.8 na verzi 0.9 prodělal v mnoha oblastech značné změny a ani funkcionalita, která dříve nečinila problémy, nyní nebyla zaručená. Při implementaci mého řešení se tedy objevily spousty problémů. Většina pramenila z nedostatečně aktuální dokumentace. Zásadním problémem se pak ukázaly být continuous query, které měly být předností InfluxDB. Fungovaly například pouze pro jednu metriku a pokud se přidala další už nefungovalo žádné. InfluxDB je ale stále ve vývoji a stále více se blíží stabilní verzi 0.9 a vývojáři ještě před vydáním slibují jejich důkladné testování, proto celé řešení postavené nad InfluxDB považuji v dlouhodobějším měřítku za použitelné. One2influx se předělávat nemusí a může být nasazen až s vydáním stabilní verze InfluxDB.

[4] <https://github.com/influxdb/influxdb/issues/2519>

Kapitola 5

Závěr

Co se týče zadání, tak navržená kombinace one2influx, InfluxDB, Grafana splňuje v teoretické rovině všechny požadavky. I když ještě není stabilní, stále je o krok napřed oproti řešením s použitím ostatních databázových systémů a vyplatí se počkat. Kritický požadavek na agregaci starších dat v praxi, v aktuální verzi InfluxDB 0.9RC29, nelze považovat za splněný. Nicméně dá se říci, že InfluxDB by se už do vydání stabilní verze nemělo nijak výrazně měnit. Lze předpokládat, že řešení bude plně funkční s odladěným vydáním, tedy v rámci jednoho až dvou měsíců a to i bez mého zásahu.

Do budoucna lze systém vylepšit především nahrazením nebo přepsáním one2influx. Nahrazení by bylo možné, jako plugin collectd, který by byl vstupně-výstupní a se stejnou funkcionalitou jako one2influx, s tím že by využíval zabudované podpory v InfluxDB pro načítání dat z collectd. Výhoda by byla v tom, že collectd je klasický linuxový démon, jehož funkčnost je prověřena a není tak experimentální jako one2influx. Další možnost je pak využít v one2influx nějakou knihovnu pro připojení k InfluxDB, což by opět přineslo, více uživateli ověřenou funkčnost, a tedy více ošetřené chování v chybových situacích. Bohužel žádná není momentálně kompatibilní s InfluxDB 0.9. Pokud by používání ONE API bylo v praxi na front-endový server příliš náročné, stálo by za to zvážit nasazení systému Ganglia nebo odchytávat pakety z monitoringu posílané na front-end nějakým dalším programem. Například ONESnooper Server z 2.3.2.

Nakonec bych chtěl poznamenat, že řešení se povedlo navrhnout tak, aby se dalo použít i na jiné cloudové platformy, třeba na zmiňovaném OpenStacku. Je třeba pouze doimplementovat prostředníka mezi API OpenStacku a InfluxDB. Tomu se nelze nijak vyhnout, protože každý cloud má API a reprezentaci dat jiné.

Literatura

[1] Blueflood: Distributed, Fault-tolerant Metrics Processing.
https://github.com/rackerlabs/blueflood/wiki, stav z 7. 5. 2015.

[2] Grafana Installation, .
http://docs.grafana.org/v1.9/installation/, stav z 7. 5. 2015.

[3] What's New in Grafana v2.0, .
http://docs.grafana.org/v1.9/installation/, stav z 7. 5. 2015.

[4] Graphite: FAQ, .
http://graphite.readthedocs.org/en/0.9.x/faq.html, stav z 7. 5. 2015.

[5] InfluxDB: Backup and Restore, .
http://influxdb.com/docs/v0.9/concepts/backup_and_restore.html, stav z 7. 5. 2015.

[6] InfluxDB: Clustering, .
http://influxdb.com/docs/v0.9/advanced_topics/clustering.html, stav z 7. 5. 2015.

[7] InfluxDB: Key Concepts, .
http://influxdb.com/docs/v0.9/concepts/key_concepts.html, stav z 7. 5. 2015.

[8] InfluxDB: Continuous Queries, .
http://influxdb.com/docs/v0.9/concepts/continuous_queries.html, stav z 7. 5. 2015.

[9] InfluxDB Overview, .
http://influxdb.com/docs/v0.9/introduction/overview.html, stav z 7. 5. 2015.

[10] Applications using libvirt, .
http://libvirt.org/apps.html, stav z 7. 5. 2015.

[11] Libvirt FAQ, .
http://wiki.libvirt.org/page/FAQ, stav z 7. 5. 2015.

[12] About the OpenNebula Project, .
http://opennebula.org/about/project/, stav z 7. 5. 2015.

[13] OpenNebula 4.12 Documentation, .
`http://docs.opennebula.org/4.12/`, stav z 7. 5. 2015.

[14] OpenTSDB: Writing.
`http://opentsdb.net/overview.html`, stav z 7. 5. 2015.

[15] OpenStack Cloud Administrator Guide, .
`http://docs.openstack.org/admin-guide-cloud/admin-guide-cloud.pdf`, stav z 18. 5. 2015.

[16] Ceilometer developer documentation, .
`http://docs.openstack.org/developer/ceilometer/index.html`, stav z 7. 5. 2015.

[17] Companies Supporting The OpenStack Foundation, .
`https://www.openstack.org/foundation/companies/`, stav z 7. 5. 2015.

[18] Prometheus: Comparsion to Alternatives, .
`http://prometheus.io/docs/introduction/comparison/`, stav z 7. 5. 2015.

[19] Prometheus Overview, .
`http://prometheus.io/docs/introduction/overview/`, stav z 7. 5. 2015.

[20] Ruby 2.2.2 Documentation.
`http://ruby-doc.org/core-2.2.2/`, stav z 7. 5. 2015.

[21] BLACK, D. A. *The Well-Grounded Rubyist.* 1. : Manning Publications, 1 edition, 2009.

[22] DIX, P. InfluxDB v0.9.0 release update.
`http://influxdb.com/blog/2015/05/01/InfluxDB-v0_9_0-release-update.html`, stav z 7. 5. 2015.

[23] DUNNING, T. – FRIEDMAN, E. *Time Series Databases: New Ways to Store and Access Data.* 1. : O'Reilly Media, 2014.

[24] FORSTER, F. collectd(1) - Linux man page.
`http://linux.die.net/man/1/collectd`, stav z 7. 5. 2015.

[25] MASSIE, M. et al. *Monitoring with Ganglia.* 1. : O'Reilly Media, 2012.

[26] OETIKER, T. rrdtool(1) - Linux man page.
`http://linux.die.net/man/1/rrdtool`, stav z 7. 5. 2015.

[27] OLSEN, R. *Eloquent Ruby.* 1. : Addison-Wesley Professional, 1 edition, 2011.

[28] SERRA, J. et al. *OpenStack Operations Guide.* 1. : O'Reilly Media, 2014.
online edice z 16.5.2015.

[29] WLODARCZYK, T. W. Performance of Data Extraction from OpenTSDB. 2014, 1, 1.

[30] YANG, F. et al. Druid: A Real-time Analytical Data Store. 2014, 1, 1.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CERIT-SC CERIT Scientific Cloud

CQ Continuous Query

CSV Comma-separated values

DNS Domain Name System

HTTP Hypertext Transfer Protocol

IaaS Infrastructure as a Service

JSON JavaScript Object Notation

KVM Kernel-based Virtual Machine

NoSQL Not Only SQL

ONE OpenNebula

REST Representational State Transfer

RPC Remote Procedure Call

RRA Round-robin Archive

RRD Round-robin Database

SQL Structured Query Language

SSH Secure Shell

SSL Secure Sockets Layer

TSD Time Series Daemon

UDP User Datagram Protocol

VM Virtual Machine (virtuální stroj)

XML Extensible Markup Language

Příloha B

Manuál

V tomto manuálu pro nasazení monitorovacího systému přepokládám funkční OpenNebulu ve verzi 4.12, kde front-endový server má operační systém Debian 8 a ONE má na `http://localhost:2633/RPC2` dostupný přístupový bod pro XML-RPC API. Pro jednoduchost provedu instalací právě na tento server, i když v praxi doporučuji vlastní server pro InfluxDB dohromady s Grafanou.

Následující postup platí pro aktuální verze InfluxDB (0.9RC29) a Grafany (2.0.2).

B.1 Požadavky

- Operační systém GNU/Linux
- `Ruby` verze 2 a vyšší
- `zlib 1.2.8`, jako závislost `nokogiri` (gem použitý v one2influx 4.2.1.1)

B.2 Instalace

Instalace InfluxDB a Grafany je v prostředí Debianu jednoduchá, protože existují připravené balíčky. Doporučuji je ale místo repozitářů stáhnout přímo z oficiálních stránek jednotlivých produktů. Všechny závislosti by měly být přímo v balíčcích a instalaci lze provést jednoduše programem `dpkg`, správcem balíčků pro Debian.

Ve výpisu B.1 jsou konkrétní potřebné příkazy pro instalaci a spuštění, protože pro oba programy není potřeba žádná konfigurace před spuštěním. Výjimečně jsem zde použil opravdu malé písmo, aby odkazy na vždy aktuální verzi, které by měly nějaký čas vydržet, zůstaly nezalomené.

```
$ wget https://grafanarel.s3.amazonaws.com/builds/grafana_latest_amd64.deb
$ wget https://s3.amazonaws.com/influxdb/influxdb_latest_amd64.deb
$ sudo dpkg -i influxdb/influxdb_latest_amd64.deb grafana_latest_amd64.deb
$ sudo service grafana-server start
$ sudo service influxdb start
```

Výpis B.1: Instalace one2influx

One2influx je program psaný formou gemu, díky čemuž se o závislosti stará správce balíčků RubyGems, který je umí automaticky nainstalovat. Ve výpisu B.2 je uveden příklad instalace závistlostí one2influx, které je třeba provést ručně.

```
$ sudo apt-get update
$ sudo apt-get install ruby ruby-dev libghc-zlib-dev -y
```

Výpis B.2: Instalace závislostí one2influx

Neuvádím zde spuštění, protože one2influx nejprve potřebuje nastavit spojení s OpenNebulou a InfluxDB, kterým provedu až v podkapitole B.3.

B.2.1 Instalace one2influx - pro produkční prostředí

Tato instalace neobsahuje možnost testování bez OpenNebuly. Jelikož jsem gem umístil na komunitní hosting RubyGems.org, jeho instalace se provede pouze příkazem: `gem install one2influx`.
Program pak bude uživateli pro spuštění dostupný globálně.

B.2.2 Instalace one2influx - pro testování

Tato instalace je vhodnější pro testování a vychází ze stažení zdrojových kódů, které jsem umístil na hosting git repozitářů GitHub a nainstalování závislostí pomocí gemu `bundler`. Příklad instalace je ve výpisu B.3.

```
$ wget https://github.com/zidekmat/one2influx/archive/master.zip
$ unzip master.zip
$ cd one2influx-master && gem install bundle && bundle install
```

Výpis B.3: Instalace one2influx k testování

Program bude spustitelný pouze přes soubor `bin/one2influx`.

B.3 Konfigurace

B.3.1 InfluxDB

Následující nastavení je možné provést přímo přes HTTP požadavky na API nebo pomocí GUI, které je dostupné na adrese: `http://localhost:8083`. Prvními kroky jsou vytvoření databáze a retention policy. Oba jdou realizovat přes GUI rozhraní a není potřeba znát příkazy. Pro vytvoření retention policy doporučuji použít vrstvy granularity z 3.4 Podrobněji o ukládání a replikaci zvolit 1, vyšší číslo slouží pro cluster režim.

Zbývá už jen nastavit continous query pro agregace dat mezi jednotlivými vrstvami. Je potřeba nastavit jedno CQ pro každou metriku a agregaci. Jedna metrika tedy bude mít tři pravidla pro příklad z 3.4. Protože je to dohromady spousta pravidel, nebudu zde vypisovat celý konkrétní příklad, ale uvedu ve výpisu B.4 jedno pravidlo, ze kterého lze vycházet ve skriptu pro generování všech zbylých. Název `jmeno_db` je potřeba upravit, stejně jako

`jmeno_query`, které musí být nad databází jedinečné. Dále je třeba změnit `gran_z` a `gran_do`, které udávají z jaké a do jaké retention policy se má agregace provést. Místo `int_agregace` se zadá interval, nad kterým se spočítá agregační funkce, zde `mean` průměr, ale tu lze také nahradit. Nakonec se místo `seznam_tagu` uvede seznam tagů, oddělených čárkou, které se vztahují k metrice `jmeno_metriky` a chceme je zachovat.

```
CREATE CONTINUOUS QUERY "jmeno_query" ON "jmeno_db"
 BEGIN
   SELECT mean(value) AS value
     INTO "jmeno_db"."gran_do"."jmeno_metriky"
     FROM "jmeno_db"."gran_z"."jmeno_metriky"
     GROUP BY time(int_agregace), seznam_tagu
 END;
```

Výpis B.4: Obecné continuous query

Vytvářením CQ procházím tak dopodrobna, protože pro ně ještě není na aktuální verzi opravená oficiální dokumentace.

Upozornění

InfluxDB má po prvním spuštění otevřené porty `8083` (GUI) a `8086` (API), proto je doporučuji zvenčí blokovat například pomocí `iptables` nebo alespoň změnit heslo administrátora a požadovat autentizaci.

B.3.2 Grafana

Nastavení Grafany silně závisí na preferencích administrátora, podle toho o jaké statistiky má zájem. Na přiloženém CD je v souboru `dashboard.json` příklad konfigurace, která generuje dashboard na obrázku 4.1 a z níž lze vycházet při vytváření vlastních dashboardů.

Konfigurační soubor lze nahrát přes GUI, které je dostupné na portu 3000, kokrétně pro nahrání je odkaz: `http://localhost:3000/dashboard/import`.

Jako v případě InfluxDB i zde doporučuji změnit počáteční heslo a navíc použít SSL.

B.3.3 one2influx

Nastavení one2influx se provádí v souboru `config.rb` v adresáři `one2influx/lib/one2influx/`, který se pro instalaci v produkčním prostředí nachází v jedné ze složek pro instalaci gemů a ty lze zjistit příkazem `gem environment`. Pro instalaci k testování je tam, kde jste rozbalili soubor `master.zip`.

Pro používání programu musíte nastavit spojení s OpenNebulou a InfluxDB, kde je navíc potřeba doplnit jméno použité databáze a retention policy, ze které se bude vycházet.

Ostatní položky jsou v souboru bohatě komentované, a proto je zde nebudu rozebírat. Jen dodám, že při nastavení metrik a tagů se odstraňuje přidáním `#` před první znak na řádku a přidává odebráním prvního znaku `#` na řádku. Tag ID je u všech objektů povinný, tudíž není nastavitelný.

B.4 Testování

Testování slouží k vyzkoušení systému nanečisto bez funkčního nastavení OpenNebuly. Program generuje XML, které by normálně získával z ONE, to zpracovává a posílá klasickým způsobem do InfluxDB. Nastavit počty jednotlivých objektů, které se budou generovat je možné v souboru `one2influx/test/fake_lib/fake_one.rb`. Nastavení metrik je možné jako v případě klasického běhu programu, ale spuštění se provádí příkazem: `test/one2influx_fake`.

Upozornění

Pro správný běh bez zbytečných chybových hlášek doporučuji v nastavení one2influx zakázat zaznamenání vlastních metrik (`cust_metric`), protože ne všechny vztahy jsou ve falešných XML dokumentech generované.

B.5 Spuštění

Pokud jste provedli alespoň základní nastavení InfluxDB a one2influx. Doporučuji program spustit na pozadí jedním z následujících příkazů, protože program je zamýšlen jako démon, tak je chybový výstup ukládán do souboru `one2influx.log` a ne na standartní chybový výstup.

`one2influx & disown` - pro spuštění produkční instalace

`one2influx/bin/one2influx & disown` - pro spuštění instalace k testování

Příloha C

Obsah přiloženého CD

```
cd/
|-- dashboard.json # Vzorový dashboard pro otestování funkčnosti
|                  #   one2influx
|
|-- latex/         # Adresář obsahující zdrojové kódy pro vytvoření
|                  #   pdf bakalářské práce
|
|-- one2influx/    # Adresář se zdrojovými kódy a dokumentací aplikace
|                  #   one2influx
|
|-- README.txt     # Návod pro použití CD
|
`-- text/          # Adresář obsahující pdf bakalářské práce
```