Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Webové rozhraní pro administraci programu Puppet

Plzeň 2016 Bc. Václav Štengl

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. června 2016

Bc. Václav Štengl

Abstrakt

Hlavním cílem této práce je vytvoření webového rozhraní pro správu nástroje Puppet. Komunitní verze Puppetu totiž neobsahuje grafické rozhraní, které by umožňovalo správu nástroje a zobrazení stavu serverů a infrastruktury.
Teoretická část práce představuje vybrané nástroje konfiguračního managementu a jejich principy. Konkrétně se jedná o nástroje CFEngine, Puppet, Chef a Ansible. Detailně je popsán zejména nástroj Puppet, který je pro tuto práci podstatný.
V praktické části jsou definovány požadavky na administrační rozhraní. Tyto požadavky jsou podrobně analyzovány a dle výsledků analýzy je navrhnuta a implementována webová aplikace. Výsledná aplikace je srovnána s nástroji *Puppet Enterprise Console*, *Foreman* a *Puppet Explorer*.

Abstract

The main goal of this diploma thesis is implementation of a web interface for administration of Puppet tool. The community edition of Puppet is missing graphical user interface, which would allow some administration tools and the identification of the servers and infrastructures status.
The theoretical part of the thesis introduces selected resources of configuration management and its principles, mainly CFEngine, Puppet, Chef and Ansible. The detailed description is for Puppet that is crucial for this diploma thesis.
The practical part introduces the definitions of requirements for administrative interface. Those requirements are deeply analyzed and results are used as a base for the implementation of the web application. The resulting application is then compared to the tools *Puppet Enterprise Console*, *Foreman* and *Puppet Explorer*.

Obsah

1 Úvod

Dle [1] bylo v roce 2014 odhadováno, že existuje 75 milionů serverů, které jsou součástí internetu. Společnost Microsoft vlastnila nejvíce těchto serverů, a to přibližně jeden milion. Společnost Google vlastnila přibližně 900 000 serverů. Další servery lze najít na úrovni organizací. S příchodem virtualizace a distribuovaných řešení jako jsou clustery či *cloud computing* neustále narůstá počet serverů a také potřeba je efektivně spravovat.
S rostoucím počtem serverů není možné navyšovat počty administrátorů, kteří by servery spravovali. V organizacích se stovkami či s tisíci servery se v rámci provádění standardních administračních operací již nelze spoléhat na manuální procesy administrátorů. Manuální metody mohou totiž vést k odchylkám na serverech způsobených lidskou chybou, snížené produktivitě a pomalé reakci na změny v systémech. Navíc jsou tyto metody v rozsáhlých prostředích velmi těžko sledovatelné, škálovatelné a udržovatelné.
V dnešní době existuje velké množství nástrojů pro hromadnou správu serverů. Pomocí těchto nástrojů lze zautomatizovat konfiguraci serverů, minimalizovat opakující se činnosti a maximálně využívat již vytvořená řešení. Jedním z nástrojů pro hromadnou správu je i Puppet. Při použití komunitní verze, která je k dispozici zdarma, lze narazit na problém s chybějícím grafickým administračním rozhraním. Administrační rozhraní by mělo sloužit ke kontrole stavu serverů a zobrazení jejich zpětné vazby. Dále by mělo být možné měnit konfiguraci serverů z jednoho centrálního prvku. Cílem této práce je navrhnout dle definovaných požadavků administrační nástroj pro komunitní verzi Puppetu a následně ho vytvořit.

Teoretická část této práce se nejprve věnuje konfiguračnímu managementu. Dále jsou představeny existující nástroje pro hromadnou správu serverů, jejich vývoj a principy. Podrobně je popsán především nástroj Puppet, který je stěžejní pro tuto práci. Také je představen nástroj *Puppet Enterprise*, který slouží zejména pro komerční využití a obsahuje webové uživatelské rozhraní *Puppet Enterprise Console* pro správu serverů a konfigurace.

V další části práce jsou pak definovány požadavky na webové administrační rozhraní. Tyto požadavky jsou poté analyzovány a je navrhnuto finální řešení. Toto řešení je dále implementováno. V závěru práce je vytvořené řešení porovnáno s existujícími nástroji a jsou zhodnoceny výsledky.

2 Teoretická část

2.1 Konfigurační management

V dnešní době existuje poměrně velké množství standardů pro konfigurační management. Navíc je nutné si uvědomit, že existuje více druhů konfiguračních managementů [2]:

- Softwarový konfigurační management, který se zaměřuje na řízení zdrojového kódu v průběhu systémového vývoje,
- hardwarový konfigurační management,
- operativní, který je nadefinován především v metodice ITIL. Operativní konfigurační management se zaměřuje na konfigurační položky jako je software, hardware, dokumentace a poskytování informačních služeb v informační infrastruktuře.

Tato kapitola je věnována operativnímu konfiguračnímu managementu a převážně metodice ITIL. Tato metodika je také v mnoha zemích využívána jako standard pro poskytování IT služeb (IT Service Management).

Metodika ITIL je sada knižních publikací, která obsahuje sbírku nejlepších zkušeností z oboru řízení služeb informačních technologií. ITIL byl publikován poprvé v letech 1989–1995 a byl obsažen v 31 knihách. Aktuálně je platná ITIL verze 3, která vyšla v roce 2007 a díky revizi předchozích verzí čítá pouze 5 knih. Tyto knihy kopírují životní cyklus služby:

- *Service Strategy* (strategie služeb),
- *Service Design* (návrh služeb),
- *Service Transition* (zavedení služeb),
- *Service Operation* (provoz služeb),
- *Continual Service Improvement* (neustálé zlepšování služeb).

Obecná definice služby metodiky ITIL je dle [3] „explicitně definovaná a popsaná funkcionalita poskytovaná informačními technologiemi, která podporuje, či přímo umožňuje chod nějakého podnikového procesu, resp. podnikové činnosti“.

S konfiguračním managementem úzce souvisí třetí kniha *Service Transition* (zavedení služeb). Cílem této knihy je dodat služby požadované businessem do produkčního prostředí. K dosažení tohoto cíle je v knize popsáno několik procesů [4]:

- *Change Management* (správa změn),
- *Service Asset and Configuration Management* (správa aktiv a konfigurace),
- *Knowledge Management* (správa znalostí),
- *Transition Planning and Support* (plánování a podpora přechodu),
- *Release and Deployment Management* (správa nasazení verzí),
- *Service Validation and Testing* (ověření a testování služby),
- *Evaluation* (vyhodnocení).

Důležitým procesem pro konfigurační management je *Service Asset and Configuration Management* (správa aktiv a konfigurace). V rámci tohoto procesu se totiž objevují důležité pojmy - konfigurační položka a konfigurační záznam.

Konfigurační položka (*configuration item*) může být jakýkoliv prvek řízeného systému. Konfigurační položky by měly být spravovány za účelem dodávky služby. Typicky zahrnují služby IT, hardware, software, stavby, lidské zdroje a další prvky.

Konfigurační záznam (*configuration record*) obsahuje detailní informace o konfigurační položce. Každý záznam dokumentuje životní cyklus jedné konfigurační položky.

V případě správy serverů si lze představit konfigurační položku jako softwarový balíček, existujícího uživatele, existující skupinu, soubor s určitými atributy a právy, běžící službu a jiné. Nástroje pro hromadnou správu serverů poskytují způsob, jak popsat konfigurační položky a jednotlivé závislosti mezi

nimi. Jedná se ovšem pouze o popis současného stavu. Pro udržení informací o životním cyklu konfigurační položky je vhodné využít libovolný nástroj pro správu verzí. S tímto nástrojem lze poté dohledat stavy konfiguračních položek v čase a sledovat vývoj a změny jednotlivých položek.

Přínosem konfiguračního managementu je existence spolehlivých informací o konfiguračních položkách, jejich vzájemných vazbách a jejich dokumentaci, což významně zvyšuje efektivitu téměř všech ostatních procesů. Navíc je možné získat okamžitou představu o stavu stroje a v případě potřeby vytvořit nový stroj s identickými položkami. Následující kapitola je věnována nástrojům, které slouží k hromadné správě serverů a umožňují popsat konfigurace jednotlivých strojů.

2.2 Nástroje hromadné správy

V dnešní době existuje na trhu velké množství nástrojů, které usnadňují automatizaci vytváření a správu konfigurace jednotlivých serverů. Většina těchto nástrojů obsahuje i definici vlastního jazyka, pomocí kterého dokáží popsat konfiguraci serveru. Tyto nástroje nejčastěji implementují architekturu klient–server, ve které je klient nainstalovaný na každém spravovaném stroji a komunikuje s centrálním prvkem – serverem. Server jako centrální prvek spravuje konfigurace a poskytuje je svým klientům. Architekturu klient–server využívá například Puppet, na který je zaměřena tato práce. Existují ovšem také nástroje, které fungují i bez předinstalovaných klientů. Zástupcem je například nástroj Ansible, který nepotřebuje na spravovaném serveru nic víc než nakonfigurovaného SSH klienta. Centrální prvek se poté připojí na daný stroj pomocí SSH protokolu a vykonává potřebné kroky.
V následujícím textu jsou popsány vybrané nástroje pro hromadnou správu serverů. Tyto nástroje poskytují základní přehled o možnostech hromadné správy a použitých principech.

2.2.1 CFEngine

CFEngine je nejstarším nástrojem pro hromadnou správu serverů. Tento projekt vytvořil Mark Burgess v roce 1993 na univerzitě v Oslu s prvotním cílem zautomatizovat konfigurace UNIX systémů, které byly různorodé a bylo k nim potřeba spravovat velké množství skriptů a zdrojových souborů [5].

Již od začátku je vyvíjen v nízkoúrovňovém jazyce C, který je velmi rychlý, ale především také platformě nezávislý. Další výhodou je to, že není potřeba instalovat na servery dodatečné závislosti k zprovoznění nástroje [5].

První verze *CFEngine 1* zahrnovala specializovaný jazyk, pomocí kterého bylo možné definovat pravidla založená na atributech systému (architektura, existující uživatel, existující skupina a jiné). Pokud systém vyhověl některému z pravidel, byly vykonány příslušné příkazy. Sada příkazů byla omezená, nicméně dovolovala např. modifikaci konfiguračních souborů, připojování disků, mazání dočasných souborů a další [6]. Nevhodným řešením se ukázalo opakované spouštění skriptů a příkazů.

S rostoucí popularitou byla v roce 2002 vydána nová verze *CFEngine 2*. Hlavní myšlenkou této verze bylo, aby systém, který je součástí konfiguračního managementu, byl konvergován do požadovaného stavu pouze nezbytnými změnami v systému. Pokud se systém již nacházel v požadovaném stavu, žádné akce nebyly vykonány. Tato idea byla velmi zásadní pro automatizaci a konfiguraci pomocí *CFEngine 2*, jelikož se již neprovádělo vícenásobné spouštění skriptů, které mohlo mít za následek i rozbití celého systému. Tím se velmi zjednodušily jednotlivé skripty a zároveň i nasazování na samotný systém [6].
Jak rostla popularita nástroje *CFEngine 2*, rozšiřoval se také jazyk a bylo přidáváno mnoho rozšíření pro potřeby jednotlivých systémů. To nakonec donutilo Marka Burgesse ke kompletnímu předělání nástroje. Výsledkem bylo v roce 2009 vydání *CFEngine 3*. Součástí této verze bylo i vytvoření nového deklarativního jazyka, který podporuje tzv. teorii slibů.

Teorie slibů je model dobrovolné spolupráce mezi dvěma autonomními činiteli, kteří zveřejňují své záměry jeden druhému ve formě slibů. Slib je deklarace záměru, jehož účelem je zvýšit příjemcovu jistotu ve tvrzení o minulém, přítomném nebo budoucím chování [6].
Pomocí deklarativního jazyka se zapisují jednotlivé sliby (*promises*), kterých je potřeba dosáhnout. Slib může být například to, že webový server bude naslouchat na portu 80. Další slib může být, že na tomtéž webovém serveru se bude nacházet příslušný adresář s příslušnými právy pro *Apache*. Kolekce slibů je nazývána *bundle* a umožňuje seskupení slibů do logických celků. Příklad *bundle*, který obsahuje sliby pro konfiguraci NTP služby je uveden na následujícím příkladu.

```
bundle agent ntp
{
  files:
    "/etc/ntp.conf"
      create => "true",
      copy_from => secure_cp("repository/files/ntp.conf",
        "192.168.114.1");

  services:
    "ntp"
      service_policy => "start";
}
```

Uvedený *bundle* obsahuje dva sliby:

- Na serveru musí existovat konfigurační soubor */etc/ntp.conf*, který má shodný obsah se souborem uloženým v definovaném adresáři na serveru.
- Na serveru musí být spuštěna služba NTP.

Pokud stav serveru neodpovídá uvedeným slibům, CFEngine vykoná potřebné kroky a tento stav napraví. Deklarací slibů tedy není určeno, jakým způsobem dosáhnout požadovaného stavu. Konverze systému do požadovaného stavu už je ponechána na samotném klientovi.

CFEngine používá architekturu klient–server. Server se liší od klienta tím, že jsou na něm uloženy jednotlivé politiky (popis konfigurací strojů). Klienti se pravidelně dotazují serveru na svou politiku a zda politika nebyla na serveru změněna. Pokud by byl server nedostupný, použije se poslední známá politika uložená lokálně na klientovi. To znamená, že klient může bez problému běžet samostatně nezávisle na serveru.
Jak je uvedeno v [9], lze princip CFEngine nástroje přirovnat k orchestru. Ten se skládá z více hudebníků (počítačů) a každý z nich má svoji kopii not, podle které hraje. Tyto noty poskytuje dirigent, který pomocí not řídí celý orchestr.

Na trhu existuje komerční verze nástroje *CFEngine 3*, která rozšiřuje otevřenou komunitní verzi. Komerční verze nástroje přináší navíc zejména zpětný sběr dat od klientů, grafický administrační nástroj, nástroje pro monitoring a jiné [6]. Stejně jako v případě nástroje Puppet komunitní verze nástroje neobsahuje nástroj pro zobrazení zpětné vazby klientů. Klienti sice mohou

odesílat emaily administrátorovi na základě událostí, nicméně neposkytují ucelený přehled o jednotlivých serverech.

2.2.2 Puppet

Nástroj Puppet byl vytvořen v roce 2005 Lukem Kaniesem, který několik let aktivně používal *CFEngine 2* a snažil se přispívat do tohoto projektu. Vývoj CFEngine byl v té době velmi uzavřený a přispívat do kódu bylo možné pouze opravou nalezených chyb [7]. Málokdo navíc sdílel CFEngine kódy pro administraci jednotlivých serverů a tak při složitější konfiguraci stroje narážel stále na problémy a nedostatky. Proto se Luke Kanies rozhodl vytvořit vlastní nástroj pro hromadnou správu – Puppet.

Puppet historicky vychází z projektu *CFEngine 2* a je napsán v jazyce Ruby. Používá vlastní modelově orientovaný doménový jazyk, který klade důraz na jednoduchost a snadnou interpretaci požadovaných vlastností. Stejně jako v případě nástroje CFEngine využívá Puppet architekturu klient–server, ve které server (Puppet master) spravuje veškerou konfiguraci pro jednotlivé stroje a klienti (Puppet agenti) se dotazují mastera na svoji vlastní konfiguraci. Tento popis je na masterovi uložen v tzv. manifestech. Nejčastěji obsahuje manifest konfigurace tzv. modulů, které umožňují logické seskupení celků konfigurace.
Ohromným přínosem modulů Puppetu je jejich obecné použití a možnost sdílení. Pro sdílení modulů existuje repozitář *Puppet Forge*. Tento repozitář obsahuje veškeré moduly sdílené mezi komunitou uživatelů Puppetu a tyto moduly lze libovolně používat ve vlastní konfiguraci. Repozitář je velmi rozsáhlý a udržovaný a je velmi pravděpodobné, že konfigurace, kterou potřebuje uživatel Puppetu vykonat, je zde již vytvořená. Velká uživatelská základna a díky ní i rozsáhlý repozitář modulů přináší pro Puppet obrovskou konkurenční výhodu.

Na strojích, které je potřeba spravovat, musí být nainstalován Puppet agent, který běží jako služba. Součástí instalace agenta je také nástroj *Puppet Facter*, který poskytuje informace (fakta) o daném stroji. Tato fakta mohou být buď systémová (operační systém, verze systému, velikost pamětí, IP adresa a další), ale může se jednat i o vlastní nadefinované hodnoty, které mohou být dále použity v manifestech.
Agent periodicky (implicitně každých 30 minut) odesílá fakta Puppet masterovi a zároveň se dotazuje na katalog, který obsahuje popis stavu pro jeho

stroj. Master na základě těchto faktů sestaví požadovaný katalog, který odešle zpět na agenta. Jakmile agent obdrží katalog, začne kontrola stavu systému oproti získanému popisu. Pokud je v systému nalezena odchylka od poskytnutého katalogu, vykoná agent potřebné změny k získání požadovaného stavu. Nakonec agent pošle masterovi report, ve kterém jsou obsaženy události, které se odehrály během posledního běhu.

Stejně jako v případě nástroje CFEngine, existuje také komerční verze nástroje Puppet. Komerční verze obsahuje navíc grafické uživatelské rozhraní pro správu serverů, statistiky, zpracování reportů, nástroje pro monitoring, technickou podporu a další. Zpětnou vazbu lze v případě komunitní verze řešit odesíláním emailů nebo notifikací při reakci na události či externími nástroji. Nástroj Puppet bude dále detailně popsán v samostatné kapitole.

2.2.3 Chef

Nástroj Chef byl vytvořen v roce 2009 Adamem Jacobem, který nebyl spokojený s některými vlastnostmi Puppetu. Největší překážkou pro něj bylo nedeterministické pořadí prováděných operací, které v některých případech vedlo k odlišnostem na agentech, jelikož každý agent mohl vykonat příkazy v různém pořadí [8]. Puppet si totiž sestavuje pro jednotlivé položky konfigurace a jejich závislosti vlastní model formou grafu. Tento model již ale nedefinuje pořadí, v jakém postupovat k získání požadovanému stavu. To je ponecháno na samotném agentovi.
Nástroj Chef naopak provádí operace v pořadí, v jakém jsou deklarovány. Popis konfigurace stroje je uveden v receptech. Kolekce receptů se nazývá kuchařka. Kuchařka může být složena z několika receptů a dohromady tvoří zdrojový kód ve vlastním doménovém jazyce, ve kterém mohou být použity prvky jazyka Ruby (nejčastěji se používá pro podmínky a větvení). Příklad jednoduchého receptu pro instalaci služby *Nginx* je uveden níže.

```
include_recipe "apt"

package 'nginx' do
  action :install
end
```

```
service 'nginx' do
  action [ :enable, :start ]
end
```

Historicky vychází Chef z nástroje Puppet, a proto je také z větší části napsaný v jazyce Ruby. Typická instalace se běžně skládá ze tří základních komponent, kterými jsou *Chef Server*, *Chef Workstation* a *Chef Node*.

Chef Server

Chef Server je centrální prvek, který uchovává uložená konfigurační data dostupná pro každý spravovaný *node* (stroj) a zároveň k nim spravuje přístupová práva. Dále udržuje veškeré informace o dané infrastruktuře a o každém spravovaném nodu, který je registrovaný právě u *Chef Serveru*.
Server dále uchovává a ukládá všechny kuchařky, recepty a metadata, které dohromady popisují všechny spravované nody.

Workstation

Na stroji, kde je nainstalován *Chef Workstation*, je možné vytvářet a spravovat veškerou konfiguraci serverů. Jedná se o vytváření kuchařek a receptů, aktualizace *chef–repo* repozitáře, interakce s *Chef Serverem* a další. Workstation je místo, kde uživatel stráví většinu času při používání Chef nástroje. Obvyklé činnosti, které je třeba provádět, jsou následující:

- Vytváření kuchařek a receptů,
- udržování *chef–repo repozitáře* s dostupnými konfiguracemi a jeho synchronizace s aktuální konfigurací,
- používání nástroje *knife*, který dokáže nahrát novou či změnit stávající konfiguraci z repozitáře na server,
- komunikace s jednotlivými nody, například zajištění *bootstrap* operace pro nabootování nově vytvářeného systému.

Chef Node

Node může být fyzický, virtuální či cloudový stroj, na kterém je nainstalován *chef–client*. Nástroj *chef–client* slouží k zajištění požadovaného stavu dle získané konfigurace. Node si od serveru vyžádá aktuální konfiguraci a popis a poté dle získaného receptu zajistí požadovaný stav. Pořadí operací je na všech nodech velmi přísně dodržováno.

Chef není jen nástrojem pro popis a správu jednotlivých serverů, ale je možné pomocí něj popsat kódem a dále automatizovat celou infrastrukturu. Je možné vytvářet, instalovat a dále konfigurovat virtuální servery. Další přínos automatizace spočívá v automatizovaném nasazování verzí na fyzické či virtuální servery a také do cloudů.

2.2.4 Ansible

Ansible je nástroj pro hromadnou centrální správu serverů používaný pro automatické konfigurování serverů, nasazování software a tzv. orchestraci kontinuálních úloh, jako je např. nasazování na více serverů nebo instalace aktualizací napříč servery. Jeho obrovskou výhodou je minimalistická implementace a díky tomu velmi snadné použití. Ansible totiž nepotřebuje na spravovaných serverech mít nainstalované agenty, ale veškerá konfigurace a správa se provádí z centrálního prvku pouze s použitím SSH protokolu.

Na serveru lze definovat tzv. inventář. Inventář je soubor v `INI` formátu, ve kterém lze definovat cílové stroje a lze je dávat do skupin (i rekurzivně – jedna skupina může obsahovat další skupiny). Příklad jednoduchého inventáře, který obsahuje skupinu *servers*, je uveden na následujícím příkladu.

```
[servers]
server_1 192.168.20.101 ansible_ssh_user=root
```

Pro popis konfigurace úloh či kroků se používá tzv. *playbook*. Jedná se o předpis konfigurace psaný v jazyce YAML, ve kterém jsou uvedené jednotlivé kroky, které je potřeba na daném stroji či strojích provést. *Playbook* se skládá z několika *plays* (her), které vymezují jednotlivé celky úloh pro server či skupiny serverů. Na následujícím příkladu je uveden playbook, který popisuje

synchronizaci adresáře `src` na serveru s adresářem `/tmp/dest` na cílovém stroji.

```
---
- name: nasazeni verze
  hosts: servers
  tasks:
    - name: synchronizace slozky
      synchronize:
        src: '{{ playbook_dir }}/../src'
        dest: /tmp/dest
        delete: yes
        recursive: yes
```

Ansible umí dobře pracovat s moduly, které jsou obecné a vícenásobně použitelné. Ve výše uvedeném případě je použit modul *synchronize*, který je určen pro přenos souborů mezi stroji. Existuje ohromné množství již vytvořených modulů, které jsou uživatelům k dispozici. Samozřejmostí je možnost vytvoření vlastního modulu.

Existuje také komerční verze nástroje Ansible, která se nazývá *Ansible Tower*. Tato verze přináší snadnou správu a ovládání pomocí grafického rozhraní, generování grafů a statistik, technickou podporu a jiné. Existují ale i externí aplikace, které se snaží grafické rozhraní pro komunitní verzi vytvořit. Jednou z nich je například *Semaphore*.

2.3 Puppet

V předchozí kapitole byly popsány některé významné nástroje hromadné správy a jejich principy. Existujících nástrojů je na trhu daleko více, nicméně pro pochopení záměru a popis jednotlivých principů by měl být popis dostačující. Tato kapitola je již zaměřena na podrobnější popis nástroje Puppet a jeho vlastnosti a možnosti.

Puppet je open source nástroj operativního konfiguračního managementu. Lze pomocí něj spravovat většinu operačních systémů typu Linux, *nix (Solaris, AIX, *BSD), Windows a MacOS [16]. Na libovolném operačním systému, který splňuje předchozí klasifikaci, umožňuje Puppet instalaci služby Puppet agent, která vykonává potřebné kroky pro získání požadovaného stavu.

K zajištění kompletní funkčnosti nástroje je vhodné projení agentů s Puppet masterem.
Na stroji s Puppet masterem se nachází aktuální konfigurace jednotlivých serverů. Dále master udržuje informace o agentech a jejich reportech, které obsahují zpětnou vazbu agentů. Instalace služby Puppet master je podporována pouze na operačních systémech typu Linux [16]. Systém Windows ani jiné další nejsou pro instalaci mastera podporovány.

2.3.1 Certifikáty

Veškerá komunikace mezi agentem a masterem probíhá pomocí zabezpečené komunikace. Ke komunikaci je použit protokol HTTPS s SSL šifrováním. K zajištění bezpečné komunikace je potřeba mít správné certifikáty na agentovi i na masterovi. Tato konfigurace je nezbytná před samotnou výměnou zpráv.

Puppet master slouží jako hlavní certifikační autorita, jejíž jméno odpovídá jménu stroje. Pod tímto jménem musí být master přístupný všem agentům, kteří s ním chtějí komunikovat. Zároveň je potřeba, aby také master mohl přistupovat k agentům dle jejich jména. Je tedy potřeba mít na všech strojích správně nakonfigurovaný `hosts` soubor, který obsahuje přiřazení zvolených IP adres k uvedeným jménům.

Agent pokusem o první připojení požádá mastera o vygenerování nového certifikátu. Výsledkem této akce je čekající požadavek, který musí být na masterovi schválen manuálním potvrzením administrátora. Po schválení požadavku je vygenerován certifikát se jménem agenta, který je následně podepsán certifikační autoritou. Privátní část vygenerovaného certifikátu je odeslána příslušnému agentovi. Veřejná část certifikátu je uložena na masterovi a slouží k ověření příchozí komunikace, zda přichází ze známého stroje. Po tomto nastavení jsou obě služby připravené komunikovat. Díky použití certifikátů k šifrování dat a přístupu k serverům dle nakonfigurovaného `hosts` souboru lze získat data z mastera pouze od známých strojů.

2.3.2 Domain Specific Language

Puppet používá vlastní deklarativní jazyk pro popis jednotlivých stavů, které jako celek popisují konfiguraci a vlastnosti serveru. Tento jazyk patří mezi

zástupce doménových specifických jazyků (DSL). Dle obecné definice DSL se jedná o jazyk navržený a určený k plnění určitého specifického úkolu [10]. V praxi se může jednat například o jakýkoliv framework či právě popis systému. Mezi nejznámější zástupce DSL patří například SQL, Ruby on Rails či Make program. Dle [11] jsou jeho významnými přednostmi například v porovnání s XML formátem dobrá čitelnost pro člověka, možnost použití konstrukcí programovacího jazyka a rychlé parsování.
Na následujícím příkladu je uveden jednoduchý zdrojový kód jazyka Puppet, ve kterém je definována existence souboru `test.txt` v zadaném adresáři. Dále jsou uvedeny parametry souboru: vlastník, skupina, přístupová práva a statický obsah.

```
#/etc/puppetlabs/puppet/manifests/site.pp
#ukázka jednoduchého manifestu
file { "/var/tmp/test.txt":
        ensure  => present,
        owner   => root,
        group   => root,
        mode    => 664,
        content => "Toto je testovací soubor vytvořený
                    Puppetem.",
}
```

Pomocí doménového jazyka se popisují požadované stavy, ve kterých by se měly jednotlivé položky (*resources*) nacházet. Zápis stavů těchto položek se tedy provádí deklarací jednotlivých *resource*. V uvedeném příkladě je *resource* typu *file*, tedy soubor. Slovo *resources* znamená anglicky zdroje či prostředky, ale v tomto kontextu jsem se rozhodl ho nepřekládat.

2.3.3 Resources

Deklarace *resource* vždy popisuje nějaký stav systému. Příkladem se může jednat například o následující stavy:

- Zadaný uživatel existuje na daném serveru,
- služba *Apache2* je spuštěná a naslouchá na portu 8080,

- daný soubor existuje v zadané cestě,
- definovaný balíček je nainstalovaný na serveru, a jiné.

Jednotlivé *resource* se zapisují a formátují následovně:

```
typ_resource { 'název resource'
  atribut1 => hodnota1,
  atribut2 => hodnota2,
}
```

Na předchozích dvou příkladech si lze všimnout, jakým způsobem jsou zapisovány jednotlivé atributy souboru. Pokud by se správce rozhodl například smazat soubor *test.txt* z vybraných serverů, stačilo by přepsat atribut **ensure** na **absent** a Puppet by se o tento požadovaný stav postaral.

Dostupné typy *resource* jsou k nalezení v dokumentaci Puppetu[1] nebo je možné vypsat je na stroji, na kterém je nainstalovaný Puppet. K tomu slouží příkaz:

```
$ puppet resource --types
```

Vybrané typy *resource* jsou uvedeny v tabulce 2.1.

Tabulka 2.1: Puppet – typy *resource*.

Resource	**Základní popis**
package	Správa softwarových balíčků.
service	Správa služeb běžících na lokálním stroji.
file	Správa lokálních souborů.
user	Správa uživatelských účtů.
group	Správa uživatelských skupin.
exec	Spouštění příkazů na lokálním stroji.

[1] https://docs.puppet.com/puppet/latest/reference/type.html

2.3.4 Třída

Kolekce či množina jednotlivých *resource* je většinou organizovaná do tříd, které mohou být volané opakovaně z ostatních kódů. Zatímco jednotlivé *resource* popisují základní typy jako je balíček, uživatel, soubor a další, třída již může popisovat větší celek konfigurace. Může se jednat například o popsání stavu služby či běžící aplikace. K tomu je běžně potřeba zajistit různé softwarové balíčky, konfigurační soubory, služby nebo také systémové úlohy. Menší třídy mohou být dále kombinovány s jinými třídami, které poté dohromady mohou popsat kompletní systémovou roli, jako je například databázový server či aplikační server webové aplikace [12].

Na dalším příkladu je definice třídy, která zajistí, aby se na systému nacházela vždy nejnovější verze webového serveru *Apache2*. Dále bude zajištěno, že služba *Apache2* bude na daném serveru vždy spuštěná.

```
class webserver {

  package { 'apache2':
    ensure => latest,
  }

  service { 'apache2':
    ensure    => running,
    subscribe => Package['apache2'],
  }
}
```

K zajištění aktuální verze balíčků slouží atribut **ensure** s uvedenou hodnotou **latest**. Stav služby je dále popsán atributem **ensure** s hodnotou **running**. Veškeré parametry k jednotlivým *resource* jsou k nalezení v online dokumentaci[2].
Existují také tzv. meta–parametry, které se zapisují jako atributy a lze je použít u libovolného *resource* typu. V uvedeném příkladu výše lze najít atribut **subscribe**, který je právě jedním z meta–parametrů. Slouží k vytvoření pouta mezi jednotlivým *resource*. V tomto konkrétním případě se při změně balíčku *Apache2* provede následně obnovu (restart) služby *Apache2*. Dalším příkladem meta–parametru je například **require**, pomocí kterého se definují

[2]https://docs.puppet.com/puppet/latest/reference/type.html

závislosti mezi jednotlivými *resource*. Není potřeba tedy řešit pořadí provádění jednotlivých *resource*, ale se správně propojenými závislostmi se o to agent postará sám. Kompletní přehled meta–parametrů lze nalézt v Puppet dokumentaci[3].

Přestože definice třídy může být součástí hlavního manifestu, je to z praktického hlediska nevhodné, protože by se manifest stal záhy nepřehledný. Pro každý server by se musela definice zkopírovat zvlášť a v konfiguraci by vznikla redundance, u které vždy hrozí riziko možné nekonzistence dat.
Z toho důvodu se definice tříd udržují odděleně v takzvaných modulech.

2.3.5 Modul

Puppet má v sobě integrovaný režim pro práci s moduly, a proto veškeré třídy definované v modulech jsou viditelné jak v hlavním manifestu, tak v ostatních třídách. Výhodou modulu je to, že definice požadovaných stavů se nachází pouze na jednom místě a je viditelná právě z ostatních modulů. Je běžným zvykem udělat třídy parametrizované, takže pro každý stroj lze v rámci konfigurace nastavovat odlišné hodnoty. Většinou se v definici uvedou implicitní parametry, které lze při deklaraci v manifestu nahradit vlastními. Předání parametrů třídě lze vidět na následujícím příkladu v kapitole *2.3.6 Manifest*.

Modul obsahuje vždy jednu hlavní třídu, jejíž název je shodný s názvem modulu. K této třídě většinou existují další podtřídy, které zobecňují a rozkládají celek na více dílčích částí.

Největší předností modulů je ovšem *Puppet Forge* repozitář. Tento repozitář obsahuje veškeré moduly sdílené mezi komunitou uživatelů Puppetu a tyto moduly lze libovolně používat ve vlastní konfiguraci. Repozitář je velmi rozsáhlý a udržovaný a je velmi pravděpodobné, že konfigurace, kterou potřebuje uživatel Puppetu vykonat, je zde již vytvořená. Příkladem nejvíce stahovaných modulů jsou *ssh, postgresql, java, nginx, ntp, apache, wordpress, mysql* a další [17]. Každý modul měl být v maximální míře univerzální, tak aby jej bylo možné distribuovat a sdílet právě v repozitáři *Puppet Forge*.

[3]https://docs.puppet.com/puppet/latest/reference/metaparameter.html

2.3.6 Manifest

Základní soubor pro popis stavu systému se u Puppetu nazývá manifest. V manifestu se již rozlišuje, jaká konfigurace se má uplatnit pro různé stroje. Manifest může obsahovat přímo definice *resource* či tříd, nicméně z důvodů zmíněných v předchozím textu je vhodnější používat pouze deklarace existujících modulů a tříd.

Všechny manifesty mají příponu `.pp` a hlavním manifestem, který je zpracováván jako první je *site.pp*. Manifesty jsou uloženy v cestě, která je nakonfigurována na Puppet masteru. Implicitně je použita `/etc/puppet/modules`.

```
node 'web_as1' {
  class { 'apache': }              # použití apache modulu
  apache::vhost { 'example.com':   # předání parametru třídy
    port    => 80,
    docroot => /var/www/html
  }
}
```

V manifestu jsou již rozlišeny jednotlivé stroje, na které jsou aplikovány definované stavy. Klasifikace serverů může být např. podle přesného názvu stroje, jak lze vidět na příkladu výše. Na název počítače může být také aplikován regulární výraz nebo může být porovnáván libovolný fakt, který Puppet agenti zasílají Puppet masterovi.

2.3.7 Rest API

Puppet master i agent poskytují několik REST API metod, které slouží k vzájemné komunikaci a výměně dat. Puppet agent využívá toto API například k získání aktuálního katalogu, získání dat ze souborů uvedených v manifestech, odesílání aktuálních reportů masterovi a jiné. Jednotlivé metody jsou velmi detailně popsány v dokumentaci[4] k HTTP API a lze je v případě správného nastavení Puppetu volat i z jiných externích nástrojů.

Použitý formát pro výměnu dat pomocí HTTP API je PSON. PSON je variantu typu JSON, která ovšem používá jiné kódování. K nedostatkům JSON

[4]https://docs.puppet.com/puppet/latest/reference/http_api/http_api_index.html

formátu patří to, že neumožňuje definovat znakovou sadu přenášeného obsahu a implicitně předpokládá, že znaky budou ze sady unicode (většinou UTF–8). PSON naproti tomu používá pro reprezentaci znaků 8–bitový zápis ASCII znaků, a proto dovoluje zapsat libovolnou sekvenci bytů. Puppet používá PSON formát pro serializovaná data a v tomto formátu je odesílá do sítě či ukládá na disk. Pro použití v HTTP požadavku je použit MIME typ *pson* nebo *text/pson* [13].

Příkladem HTTP metod, které poskytuje Puppet master agentům, jsou získání aktuálního katalogu, získání obsahu souboru, odeslání reportu a další. V příkladu níže je uveden požadavek pro nahrání reportu na mastera. K požadavku je přiložen YAML report, který obsahuje veškeré důležité informace o posledním běhu agenta.

```
PUT /production/report/puppetagent HTTP/1.0
ContentType: text/pson
Content-Length: 1428

{"host"=>"puppetagent1",
 "time"=>"2013-09-12T03:50:59.009301000+02:00",
 "configuration_version"=>1357986,
 "puppet_version"=>"3.3.0",
 "kind"=>"apply",
 "status"=>"unchanged",
 "metrics"=>
  {"resources"=>
    {"name"=>"resources",
     "label"=>"Resources",
     "values"=>
      [["total", "Total", 1],
       ...
}
```

Další skupinou jsou služby poskytující informace ohledně SSL certifikátů. Jedná se například o získání veřejného certifikátu pro zvolený server, získání revokačního listu, získání všech certifikátů, zjištění stavu certifikátu a jiné. Tyto služby jsou také velmi důležité, protože například seznam všech připojených agentů lze zjistit právě pomocí certifikátů. Pro zjištění připojených agentů je třeba vytvořit požadavek na seznam všech platných certifikátů, které jsou uloženy na masterovi. Z vrácených hodnot lze poté rozpoznat,

které certifikáty patří k jednotlivým strojům a díky tomu získat seznam serverů.

2.3.8 Reporty

Pro automatizovanou konfiguraci serverů je potřeba mít nainstalovaného Puppet agenta na každém serveru, který má být spravován. Agenti musí být správně propojeni s Puppet masterem, se kterým komunikují pomocí zabezpečené komunikace přes HTTPS protokol. Puppet master zahrnuje správu manifestů, ze kterých je vytvářen aktuální katalog pro každý spravovaný server. O tento katalog si agent periodicky žádá a konfiguruje (ověřuje) dle něj stav jednotlivých *resource* na serveru. Výsledkem každého běhu je report, který agent odešle na svého mastera.
Pro administrátora, který spravuje infrastrukturu serverů, je důležité mít přehled o aktuálních, ale také předchozích stavech serverů. Tyto informace lze získat z obdržených reportů, se kterými umí Puppet master pracovat. Pro absolutní přehled nad servery je nutné mít absolutní kontrolu nad těmito reporty. Existují čtyři základní způsoby, jak lze reporty zpracovávat [14]:

- Uložení reportu jako soubor ve formátu YAML do definovaného adresáře,
- odeslání reportu jako soubor ve formátu YAML pomocí HTTP nebo HTTPS protokolu na definovanou URL adresu,
- uložení reportu do logovacího souboru na lokálním stroji,
- zavolání vlastního obslužného kódu.

Uložení reportu na disk je implicitní nastavení mastera po instalaci. Pokud je vyžadováno jiné nebo dodatečné zpracování, je možné si napsat vlastní obslužný kód v jazyce Ruby či využít již hotová řešení, mezi která patří například notifikace na *Twitter, HipChat, SNMP* protokol a další [15]. Většinou se notifikace o běhu odesílají pouze v případě neúspěšného provedení některého z běhů, ale není problém posílat veškeré reporty. Je ovšem nutné počítat s tím, že každý agent generuje implicitně report každých 30 minut a při větším množství spravovaných serverů může objem notifikací způsobit nepřehlednost.

Open source nástroj, který lze také použít pro zpracování a uložení reportů na masterovi je *PuppetDB*. Tento nástroj ukládá reporty do *PostgreSQL* databáze, ke které se lze také připojit z externího nástroje a s daty pracovat. V databázi je dále uložen aktuální katalog a fakty jednotlivých nodů.

2.3.9 PuppetDB

Jedná se o open source projekt, který slouží k ukládání a poskytování dat o Puppet agentech. Mezi tato data patří aktuální katalogy, fakta o jednotlivých agentech, reporty a obsah reportů (události, časové značky a jiné). Propojením nástroje *PuppetDB* s Puppet masterem lze získat další možnost dodatečné úpravy konfigurace systémů. Implementace totiž umožňuje manuálním přidáním faktů do databáze či manuálním přidáním tříd do příslušné tabulky ovlivnit výsledný katalog, který bude poslán agentovi.
Konkrétní implementace je taková, že fakta, která odešle agent masterovi, jsou nejdříve uložena do databáze. K těmto faktům jsou dále přidána uživatelem vytvořená fakta, která jsou společně s ostatními použita pro vygenerování katalogu. Dále je katalog, který se vygeneruje na masterovi z uložených manifestů, před odesláním agentovi nejprve uložen do *PuppetDB* databáze. K uloženému katalogu mohou být následně manuálně přidány záznamy o třídách a tím lze ovlivnit finální podobu katalogu.

PuppetDB nástroj je použit v komerční verzi *Puppet Enterprise*. Slouží zde pro ukládání dat o agentech, ale právě také pro dodatečnou kontrolu nad konfigurací. Součástí komerční verze Puppetu je i grafické uživatelské rozhraní pro správu nástroje, které se nazývá *Puppet Enterprise Console*. Tento nástroj umožňuje dynamicky přidávat a spravovat fakta a třídy v *PuppetDB* databázi. Tím je možné měnit finální podobu katalogu bez zásahu do manifestů.

2.4 Puppet Enterprise

Puppet Enterprise je komerční verze nástroje Puppet. Zdrojové kódy tohoto nástroje jsou uzavřené a vychází ze zdrojových kódů komunitní verze. Tento nástroj je zpoplatněn podle počtu serverů, které jsou spravovány nástrojem Puppet. Počet serverů odpovídá počtu podepsaných certifikátů na Puppet masteru. Pokud spravovaných serverů není více než deset, není vyžadována

žádná licence a lze tento nástroj používat bezplatně. V případě většího počtu serverů se cena odvíjí od počtu spravovaných serverů a záleží na konkrétní nabídce. Dle [18] začíná cena za jeden server na částce 100$ ročně, nicméně finální cena je vždy předmětem dohody s Puppet týmem. Výhody, které přináší komerční verze, jsou uvedeny níže:

- Stabilní a otestovaná instalace na podporovaných verzích operačních systémů,
- technická podpora Standard (9–17, 5 dní v týdnu) nebo Premium (24x7x365),
- *Enerprise Console* – uživatelské GUI rozhraní pro správu serverů,
- zpracování reportů a uložení do *PuppedDB* databáze,
- monitoring serverů a událostí,
- možnost autentizace přes *LDAP, Active Directory* nebo *Google Apps Directory*,
- podpora vytváření a přiřazování serverů do skupin (*group*). [20]

2.4.1 MCollective

Významným nástrojem pro *Puppet Enterprise* je *MCollective* framework (*Marionette Collective*). Jedná se o framework, který slouží ke koordinaci a řízení spravovaných serverů. Zajišťuje interakci s jednotlivými servery a umožňuje kontrolu nad nasazováním na skupiny serverů, na kterých je nainstalován Puppet agent [22].
Příkladem reálné infrastruktury mohou být webové, databázové a monitorovací servery. Pro koordinaci (orchestraci) nasazení lze pomocí *MCollective* frameworku např. hromadně vypnout monitorovací servery, aby neohlašovaly chyby na serverech během nasazení. Dále lze nasadit aplikaci na webové a databázové servery. Po úspěšném nasazení lze opět zapnout monitorovací servery. Není tedy třeba řídit každý server zvlášť, ale je možné definovat skupiny, které jsou poté řízeny právě *MCollective* frameworkem.
Zdrojem veškerých informací o serverech a skupinách je *PuppetDB* nástroj. Není tedy třeba konfigurovat tento nástroj. Skupiny lze editovat v nástroji *Puppet Enterprise Console*.

2.5 Puppet administrace a zpětná vazba

2.5.1 Puppet Enterprise Console

Puppet Enterprise Console (dále jen konzole) je webový administrační nástroj pro správu Puppetu, který běží na stejném stroji jako master. Je součástí instalace *Puppet Enterprise* edice. Grafické rozhraní nástroje odstiňuje uživatele od práce s příkazovou řádkou na Puppet masterovi. Dále zobrazuje zpětnou vazbu agentů a umožňuje jejich dynamickou správu [22].

Role a účty

V tomto nástroji je možné vytvářet uživatele a přiřazovat jim definované role. Dostupné role jsou následující:

- *Administrators* – Role, která přiřadí uživateli veškeré pravomoci v nástroji.
- *Operators* – Role, která umožní uživateli vytvářet skupiny pro servery, přiřazovat servery do skupin a potvrzovat podepsání certifikátů Puppet masterem.
- *Viewers* – Role pro prohlížení reportů, serverů a stavu prostředí.

Jednotlivé role lze přiřazovat jak uživatelům, tak i skupinám uživatelů. Není možné se jako uživatel zaregistrovat. Uživatel musí být manuálně vytvořen administrátorem, který také uživateli přidělí příslušná práva.

Skupiny

Pomocí Puppet konzole lze vytvářet skupiny (*grupy*), do kterých lze přiřazovat jednotlivé servery. Skupinám je poté možné přiřazovat různé moduly a tím vlastně dynamicky měnit konfigurace jednotlivých serverů. Je možné například vytvořit skupinu *dev* pro vývojové servery, na které lze ladit nové moduly. Když jsou moduly odladěné, je možné je nasadit na ostatní servery.

Certifikáty

Pomocí administračního rozhraní je možné potvrzovat příchozí požadavky na podepsání certifikátů. Právo na tuto činnost mají uživatelé s rolí *Operators* a *Administrators*.

Externí klasifikace

Externí klasifikace (*External Node Classifiers*) je dynamické přiřazování modulů a *resource* k jednotlivým serverům či skupinám. Pomocí této funkčnosti je možné doplňovat či přepisovat nastavení, které je uvedené v manifestech. Moduly musejí být přítomné na Puppet master stroji v definované složce a poté je lze dynamicky přiřazovat pomocí Puppet konzole.

Live Management

Live Management je pojmenování komponenty, která dokáže ovládat Puppet agenta. Tato komponenta je součástí nástroje *Puppet Enterprise* a lze pomocí ní spravovat a sledovat jednotlivé agenty. Ovládání této komponenty je poté přístupné z Puppet konzole. Pro uživatele s příslušnými právy lze ovládat na agentu následující funkce:

- Zastavení Puppet agenta či agentů,
- spuštění Puppet agenta či agentů,
- vynucení nového běhu Puppet agenta či agentů,
- zjištění aktuálního stavu agenta či agentů.

Pomocí tohoto nástroje není třeba se přihlašovat na jednotlivé agenty a používat příkazovou řádku. Z centrálního bodu (Puppet konzole) lze všechny agenty ovládat a spravovat.

Zpětná vazba

Pro zjištění zpětné vazby jednotlivých agentů slouží v Puppet nástroji reporty. Tyto reporty jsou v komerční verzi *Puppet Enterprise* zpracovávány nástrojem *PuppetDB* a ukládány do relační databáze. Informace o reportech a jejich obsah jsou poté zobrazovány uživatelům v Puppet konzoli. Jedná se o metriky posledního běhu (doba běhu, doba stahování konfigurace, počet *resource*, počet událostí, ...), události, zdroje událostí (třídy, servery, *resource*), logy, chybné stavy a jiné [23].

Dále lze na detailu serveru nalézt příslušná fakta, nainstalované moduly, reporty, skupiny a nedávnou aktivitu serveru. Všechny tyto informace jsou uloženy v *PuppetDB* databázi.

2.5.2 Puppet Explorer

Jedná se o nástroj s webovým rozhraním, který umožňuje zobrazovat data uložená v *PuppetDB* databázi. Slouží tedy pouze ke sledování stavu Puppetu. *Puppet Explorer* je napsán v jazycích CoffeeScript a AngularJS a podporuje komunikaci s více *PuppetDB* instancemi [24].

Tento nástroj umožňuje filtrování serverů dle specifického dotazovacího jazyka, který je použit v existujícím Puppet modulu *dalen-puppetdbquery* [24]. Tento dotazovací jazyk umožňuje filtrovat jednotlivé servery a zobrazovat události či fakta serverů. Při vytváření filtrů se lze odkazovat na fakta a *resources*. V dotazech lze používat logické operátory, operátory porovnání či regulární výrazy. Příklady dvou jednoduchých dotazů jsou uvedeny na příkladu níže [25]:

```
#Všechny servery s amd64 architekturou,
#na kterých je nainstalován mysql-server.
package["mysql-server"] and architecture=amd64

#Všechny servery, na kterých je třída Postgresql::Server
#s verzí 9.3.
class[postgresql::server]{ version="9.3" }
```

Nástroj umožňuje primárně zobrazit uživateli jednotlivé servery a k nim příslušné informace. Konkrétně se jedná o reporty, fakta a události. Dále posky-

tuje zajímavé interaktivní grafy pro všechny *resources*, které jsou na serveru spravovány.

2.5.3 Foreman

Dle [26] se jedná o největší open source projekt týkající se Puppetu, který není vyvíjen společností *Puppet Labs*. Nástroj původně vznikl jako uživatelské rozhraní pro nástroj Puppet a postupem času byl rozšiřován o další funkcionality. Současné funkcionality nástroje pokrývají celý životní cyklus serveru. Nástroj poskytuje prostředky pro instalaci serveru a jeho následnou konfiguraci, ale také monitoring v čase a případné zrušení serveru. K vytváření nových serverů je možné nástroj integrovat k různým virtualizačním nástrojům (*VMware vCenter Server, OpenStack* a jiné) a nástrojům veřejného cloudu (*Amazon EC2, Rackspace* a další.) [26]. Vytvořené servery je možné dále konfigurovat pomocí nástroje Puppet, který *Foreman* integruje ve svém administračním rozhraní. Pro správu nástroje Puppet lze využít následující funkcionality:

- Vytváření skupin pro jednotlivé servery a skupiny,
- externí klasifikace tříd a modulů,
- správa certifikátů na masterovi,
- zpracování a zobrazení reportů včetně možnosti dotazování,
- zobrazení faktů, událostí, metrik a tříd,
- spuštění běhu agenta.

Nástroj *Foreman* obsahuje několik obrazovek s různými přehledy, které slouží pro monitoring a zjištění stavu serverů. Na těchto obrazovkách lze rychle nalézt servery, na kterých nastala chybová událost či server, který již delší dobu neposkytl žádný report. Dále je u serverů držena veškerá historie příkazů, které byly provedeny uživateli pomocí nástroje *Foreman*. Zajímavou obrazovkou je dále statistika jednotlivých faktů, které se týkají všech serverů. Zde jsou procentuelně vyjádřeny počty serverů, např. pro nainstalovaný operační systém, architekturu, počet procesorů, nainstalované moduly a jiné.

3 Administrace pro Puppet

3.1 Požadavky na aplikaci

V této části textu je uvažována situace, kdy je na všech spravovaných serverech nainstalována služba Puppet agent, která korektně komunikuje s Puppet masterem. Dále se předpokládá, že konfigurace modulů, tříd a manifestů na masterovi je již funkční. V této situaci potřebuje administrátor, který je zodpovědný za dané servery, monitorovat stav serverů, provádět změny v konfiguracích serverů a mít přehled o všech stavech a změnách. Tento přehled by měl být centralizovaný. To znamená, že uživatel nalezne veškeré potřebné informace na jednom místě. K tomuto účelu by měla sloužit právě webová aplikace, která je předmětem této práce.

Ověření aktivity serverů

Základní informace, kterou uživatel při správě systémů jistě ocení, je zjištění, zda jsou všechny servery dostupné. Může totiž nastat situace, kdy některý ze serverů může být například vypnutý či odpojený od sítě. Monitoring hardware sice není záležitostí Puppetu (k monitorování systémových prostředků a infrastruktury slouží jiné nástroje, např. *Centreon*, *Zabbix* či *Nagios*), ale přehled o jednotlivých strojích rozhodně přinese užitečnou informaci.

Přehled o aktuálních či nedávných stavech serverů

Uživatele může zajímat, zda jsou servery ve stavu odpovídajícímu popsaným stavům. V rámci konfigurace může například na serveru běžet služba, která vykonává užitečný kód. Pokud by se služba z nějakého důvodu zastavila (např. díky neošetřené výjimce, která by celý program zastavila, nebo díky omylu administrátora), měl by být uživatel schopen tuto situaci zjistit, aby zamezil případnému budoucímu opakování. Puppet sice dokáže automaticky službu restartovat a nežádoucí stav napravit, ale včasná informace a oprava příčiny je na produkčních serverech ve většině případů vyžadována.

Přehled reportů a jejich detailní zobrazení

Přehled o událostech jednotlivých běhů, stav jednotlivých *resource* a délka trvání událostí jsou informace obsažené ve vygenerovaném reportu. Aplikace by měla tyto reporty přehledně zobrazovat formou tabulek a grafů. Detailní informace obsažené v reportu budou využity primárně v případě neúspěšného běhu, kdy bude mít uživatel možnost zobrazit detailní informace a zjistit, které akce se nepovedly a z jakého důvodu.

Zobrazení faktů o serverech

Puppet master má k dispozici fakta o jednotlivých serverech. Tato fakta popisují daný stroj a zároveň mohou sloužit k rychlému přehledu o konfiguraci stroje. Aplikace by měla umět zobrazit fakta jednotlivých serverů a poskytnout tím lepší přehled o jednotlivých strojích.

Editace manifestů

Dalším požadavkem na aplikaci je možnost editace manifestů. Manifest soubory obsahují přiřazení modulů, tříd a *resource* ke konkrétním strojům a lze pomocí nich měnit konfiguraci libovolných serverů. Možnost editace manifestů by neměla sloužit přímo k definování nové konfigurace, ale spíše k malým rychlým změnám, které lze provést bez vytváření nových manifestů. Je tedy vyžadována možnost editace stávajících manifestů.

Spuštění běhu agenta

Každý z agentů kontroluje implicitně každých 30 minut svoji konfiguraci vůči katalogu vygenerované masterem. Pokud by byl změněn manifest pro vybraný server trvalo by v nejhorším případě právě 30 minut, než by se změna na daném stroji promítla. Aplikace by proto měla umožňovat okamžitou propagaci nové konfigurace na server. Výsledkem spuštění běhu by měl být report, který obsahuje detailní informace o uskutečněném běhu.

Administrace uživatelů

Aplikace by měla být dostupná pro různé uživatele s různými uživatelskými rolemi. Měla by umožňovat přiřazení vybraných serverů pouze vybraným uživatelům a nastavovat práva na různé akce. Základním právem je pouhé zobrazení informací o daném stroji. Rozšířeným právem může být manuální spuštění běhu agenta.

3.2 Analýza požadavků

V předchozí kapitole byly definovány základní požadavky, které by měla výsledná aplikace splňovat. V této kapitole budou všechny uvedené body analyzovány a budou uvedeny možnosti řešení. Z definovaných požadavků byl vytvořen následující výčet funkčností, které by měla aplikace uživateli poskytovat:

- Ověření aktivity serverů,
- přehled o aktuálních či nedávných stavech serverů,
- přehled o reportech a jejich detailní zobrazení,
- zobrazení faktů o serverech,
- editace manifestů,
- spuštění běhu a vygenerování nového reportu,
- administrace uživatelů.

Cílem této práce je vytvořit aplikaci s grafickým rozhraním pro správu nástroje Puppet. Tato aplikace poběží na webovém serveru a bude přístupná uživateli z webového prohlížeče. Z hlediska možných technologií a programovacího jazyka jsou preferována open source řešení.

Editace manifestů

Jednou z funkčností, která je od aplikace vyžadována, je editace manifestů. Tyto konfigurační soubory jsou standardně uloženy na masterovi v definované složce. Puppet master ovšem neposkytuje žádnou možnost, jak obsah těchto souborů číst či dokonce editovat. Jedinou možností editace je manuální úprava těchto souborů administrátorem.
Alternativou k manuální úpravě manifestů je použití *PuppetDB* nástroje a modifikace záznamů v tabulkách. Jak již bylo popsáno výše, s *PuppetDB* lze dodatečně přidávat jednotlivé moduly do katalogu a tím ovlivnit jeho finální podobu.

Pro jednoduchost řešení a splnění požadavku je zvoleno řešení, ve kterém webový server s aplikací pro správu Puppetu poběží na serveru společně s Puppet masterem. Na tomto stroji se nachází požadované soubory a aplikace s dostatečně nastavenými přístupovými právy k nim bude mít přímý přístup. Zobrazení obsahu manifestů a jejich editace tedy bude probíhat v rámci úpravy souborů na lokálním stroji.

Jak bylo zmíněno v předchozí analýze, instalace Puppet mastera je podporována pouze na systémech typu Linux, a proto i webový server s aplikací musí být spustitelný na operačním systému typu Linux.

Přehled o reportech a jejich detail

Reporty plní u nástroje Puppet velmi důležitou roli, neboť obsahují důležité informace o stavech serverů a také o událostech, které se při posledním běhu odehrály. Pro absolutní přehled nad spravovanými servery je nutné mít kontrolu nad těmito reporty. Jak již bylo zmíněno v předchozím textu, report je ve formátu YAML a existují čtyři základní způsoby, jak lze reporty zpracovávat:

- Uložení reportu jako soubor do definovaného adresáře,
- odeslání reportu pomocí HTTP či HTTPS protokolu na definovanou URL adresu,
- uložení reportu do logovacího souboru na lokálním stroji,
- zavolání vlastního obslužného kódu.

Implicitní nastavení Puppet mastera je ukládání souborů do definovaného adresáře. Open source řešením pro vlastní zpracování reportu a uložení do relační databáze je *PuppetDB* nástroj, který reporty ukládá do *PostgreSQL* databáze. V této databázi je report již zpracován v příslušných tabulkách a k těmto datům lze v případě potřeby přistupovat. Data jsou v databázi rozdělena na události, které během běhu nastaly, stavy jednotlivých *resource*, logy, časové značky a jiné. V této databázi jsou navíc obsažena i fakta jednotlivých nodů a také třídy, které jsou součástí katalogu.
Použití *PuppetDB* nástroje by znamenalo splnění několika dalších požadavků a zjednodušení práce, nicméně ve výsledném řešení nástroj *PuppetDB* není použit. Během první instalace, konfigurace *PuppetDB* a následném propojování s Puppet masterem se totiž objevilo několik chyb, které ovlivňovaly správnou funkci Puppet mastera. Jednalo se hlavně o chyby v aplikační části nástroje, která zpracovávala katalog a fakta jednotlivých strojů. Chyby byly způsobeny převážně nekompatibilitou s aktuální verzí nástroje Puppet, ale také například nefunkční aplikační částí na stroji s nastavenou doménou. Přestože v té době již *Puppet Enterprise Console* používala vlastní nástroj, který vycházel ze zdrojových kódů *PuppetDB*, samotné open source řešení mi v době analýzy nepřišlo vhodné, a proto není použito pro tuto aplikaci.

Jelikož již bylo rozhodnuto, že aplikace poběží na stejném stroji jako Puppet master, lze jednoduše zpracovávat reporty uložené v lokální složce. Další alternativou by mohlo být vytvoření webové služby, na kterou by Puppet master odesílal doručené reporty. Toto řešení by s sebou neslo riziko, že reporty vygenerované v době, kdy webová služba neběžela nebo nebyla dostupná, by nebyly zpracovány a následkem toho by aplikace zobrazovala nekompletní data.

Vybraným řešením s převažujícími výhodami je zpracování reportů přímo z definované složky. To znamená kontrolovat složku s reporty periodicky úlohou, která bude ověřovat přítomnost nových reportů. V případě nalezení nových reportů provede jejich zpracování a uložení do databáze.

Jelikož lze konfigurovat periodu generování reportů na agentech, musí být možné konfigurovat také periodu úlohy pro kontrolu reportů. Toto řešení je prospěšné v tom, že pokud aplikace nebude v běžícím stavu, reporty se budou korektně ukládat do definované složky. Při spuštění aplikace se všechny reporty zpracují dodatečně. Tím bude zaručena konzistence dat mezi Puppet masterem a aplikací.

Spuštění běhu agenta

Po editaci manifestu je potřeba, aby se provedené změny co nejrychleji promítly na server. To má na starost Puppet agent, který rozpozná změnu v katalogu a provede na serveru potřebné kroky. Z důvodu bezpečnosti zahajuje běh agenta sám agent a žádá mastera o svoji aktuální konfiguraci. Není to tedy tak, že master vynutí běh na agentovi pouhým odesláním katalogu. Iniciátor komunikace musí být vždy agent.

Puppet agent obsahuje v REST API metodu pro vynucení běhu agenta. Po zavolání této metody zahájí agent komunikaci s masterem a vyžádá si aktuální katalog. Po získání katalogu se inicializuje běh agenta, který po skončení běhu vygeneruje report ve formátu YAML. Vygenerovaný report je automaticky odeslán na mastera, který report zpracuje dle definované konfigurace.

Další možností, jak inicializovat běh agenta je spustit příkaz přímo na stroji. K tomu slouží následující příkaz:

```
$ puppet agent –test
```

K spouštění příkazů na agentech by bylo ovšem potřeba používat SSH protokol pro připojení z mastera. Výhodným řešením je tak použití REST API metody, kterou lze volat pro jednotlivé servery.
Pro získávání dat pomocí REST API z externích nástrojů je potřeba nastavit přístup k jednotlivým metodám volání na masterovi a agentovi. Zabezpečení přístupu se provádí v konfiguračním souboru `auth.conf`. V tomto souboru je nutné nastavit přístupová práva k jednotlivým metodám, které budou využívány.

V rámci implementace je tedy třeba zpracovat report ve formátu YAML. Možným řešením by mohlo být napsání vlastního kódu pro zpracování daného formátu. Toto řešení je ovšem časově náročné a přináší riziko možných implementačních chyb. Vhodnějším řešením je zřejmě použití vhodných open source knihoven, které jsou již vyzkoušeny v praxi několikanásobným použitím v existujících projektech. Je tedy velká pravděpodobnost, že plní očekávanou funkčnost bez chyb.

Zobrazení faktů o serverech

Pro zjištění faktů o daném serveru lze použít metodu REST API, která vrátí aktuální fakta patřící k serveru. Tuto metodu poskytuje již Puppet master, který má přehled o aktuálních faktech. Jelikož bylo v předchozí analýze rozhodnuto, že aplikace bude využívat REST API, je získání faktů z REST API logickou volbou.
Odpověď na požadavek je okamžitá a získání aktuálních hodnot je téměř okamžité. Z toho důvodu není potřeba ukládat tato data do databáze pro zobrazení v aplikaci. Při požadavku na zobrazení lze zavolat příslušnou metodu REST API a aktuální data zobrazit.

Aktivita serverů

Výsledkem každého běhu agenta je report, který je odeslán masterovi. Dle předchozí analýzy jsou reporty k daným strojům zpracovány automatickou úlohou, která zajistí uložení nových reportů do databáze. Důležitou informací, kterou každý report obsahuje je časová značka, která vyjadřuje čas spuštění reportu. Pokud by byly v databázi uloženy poslední reporty daného stroje a zároveň je známá perioda, s jakou Puppet agent reportuje, lze z časů jednotlivých reportů získat dobrý přehled o aktivitě daného serveru. Ideálním stavem je pravidelné periodické vytváření reportů. Pokud je ale např. čas posledního reportu v databázi starší než 30 minut, může být příčinou nedostupný server či neběžící Puppet.
Z uložených informací o reportech lze dále poskytovat statistiku o přijatých reportech formou grafů a tabulek. Samozřejmostí je tabulka s reporty za zvolené období. Dále lez vhodným databázovým dotazem získat četnost jednotlivých stavů reportů (*Changed, Unchanged, Failed*) za určité období.

Administrace uživatelů

V infrastruktuře spravované Puppetem lze mít desítky, stovky, ale klidně také tisíce serverů. Při tak velkém množství serverů je vhodné pro lepší kontrolu nad spravovanými servery mít také více Puppet masterů, kde každý z nich má na starost jiné stroje. V rámci požadavků na aplikaci pro administraci těchto serverů je vyžadováno, aby bylo možné přiřazovat jednotlivým uživatelům práva na jednotlivé servery. Tím je možné rozdělit kontrolu nad různými stroji mezi více lidí.

Přiřazování práv k serverům jednotlivým uživatelům je pravomocí administrátora. Zároveň má administrátor kontrolu nad všemi uživateli, jejich rolemi a zmíněnými právy k serverům. Dále je pravomocí administrátora zvolit, které servery jsou sledovány aplikací a které ne. Také editace manifestů a s tím spojená změna konfigurace serverů je záležitostí administrátora. Základní role aplikace jsou tedy:

- Administrátor (ADMIN),
- uživatel (USER).

Na základě analýzy je z výše uvedených požadavků vytvořen seznam obrazovek a jsou přiřazena práva jednotlivých rolí. Obrazovky jsou vypsány v tabulce 3.1.

Tabulka 3.1: Seznam obrazovek a příslušné role.

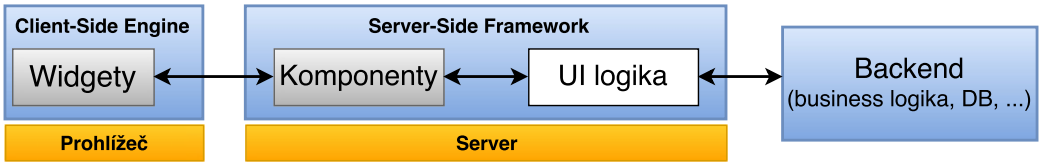
Obrazovka	**USER**	**ADMIN**
Přehled o nedávné a současné aktivitě serverů.	✓	✓
Editace a administrace uživatelů a jejich rolí.		✓
Editace a administrace serverů a přiřazení práv k serverům.		✓
Zobrazení a filtrování reportů.	✓	✓
Editace manifestů.		✓
Zobrazení informací o serveru, spuštění běhu agenta.	✓ ✓	✓ ✓

Právo na spuštění běhu agenta je přístupné také uživatelům s rolí USER, nicméně jedná se o speciální volbu, která musí být nastavena administrátorem při zakládání či editaci uživatele. Časté spuštění běhu totiž může server vytěžovat a představuje tedy možné riziko pro administrátory. Administrátor má tedy možnost nastavit právo na zobrazení všech informací o serveru a navíc přiřadit právo na spuštění běhu agenta.

4 Implementace

4.1 Technologie

Aplikace je psána ve webovém frameworku *Vaadin*. Jedná se o softwarový framework pro tvorbu webových aplikací. Zdrojový kód *Vaadin* aplikací je psán v jazyce Java a je dále za pomocí GWT frameworku překládán do JavaScriptu. Kód v jazyce JavaScript je následně interpretován v internetovém prohlížeči. Obrovskou výhodou tohoto frameworku je to, že aplikační i prezentační vrstva aplikace jsou programátorem psány ve stejném jazyce. Není tedy třeba znát ani používat front–end technologie, to *Vaadin* obstará sám. Zároveň tento framework řeší práci s DOM objekty, odstiňuje programátora od odlišností jednotlivých prohlížečů a zajišťuje asynchronní volání pomocí AJAX technologie. Psaní prezentační vrstvy ve *Vaadin* frameworku je velmi podobné psaní desktopové aplikace pomocí knihovny *Swing*. Veškerý zápis zdrojového kódu je zapsán v jazyce Java a o interpretaci do prohlížeče si již postará framework.



Obrázek 4.1: Architektura Vaadinu.

Na obrázku 4.1 je zobrazena architektura frameworku *Vaadin*. V hierarchii komponent uložených na serveru je nejvyšší komponentou *UI*. Ta může reprezentovat celé okno prohlížeče (tab), nebo část stránky, ve které je zasazena *Vaadin* aplikace. Zároveň složí jako přístupový bod pro prvky stránky, které nejsou reprezentovány komponentou (notifikace, podokna a vlastní JavaScriptový kód spouštěný v prohlížeči). Při zadání URL aplikace do prohlížeče se na serveru vytvoří nová instance *UI*.

Do prohlížeče klienta je nahrána klientská část frameworku *Client–Side Engine*, která je napsána v JavaScriptu. Klientská část má za úkol vykreslení *UI* a jeho komponent a dále zachytávání událostí od uživatele (kliknutí, stisk

klávesy s jiné) a jejich asynchronní odeslání na server. Prohlížeč slouží tedy jako tzv. tenký klient, který zachycené požadavky pouze odešle na server. Server následně zajistí potřebné změny a ty pošle zpět klientovi k zobrazení.

Na serverové části frameworku (*Server–Side Framework*) jsou uchovávány všechny instance *UI* a stav všech klientů. Veškerá aplikační logika je vykonávána zde. Každá serverová komponenta (*Button, TextField, Table*, ...) má na klientské straně svůj protějšek, tzv. *Widget*. Jedná se o reprezentaci komponenty, která pouze zobrazuje aktuální stav uložený na serveru.

Výhodou frameworku *Vaadin* je to, že programátor implementuje logiku *UI* a popíše stavy jednotlivých komponent. Nemusí se tedy starat ani o vykreslení obsahu pomocí jiných technologií (HTML, JavaScript), ale ani o komunikaci mezi klientem a serverem (HTTP, AJAX, JSON, *session*).

Jednotlivé stránky aplikace jsou potomci třídy *View* nebo sami implementují rozhraní *View*. Každá stránka je jednoznačně identifikována dle jména, které slouží k přístupu na stránku. V uvedeném případě je název stránky *home-page*. Na ukázce kódu je uvedena stránka s layoutem, který obsahuje jedno tlačítko.

```
@VaadinView(VIEW_NAME)
public class HomePageView implements View {

    public static final String VIEW_NAME = "home-page";

    public HomePageView() {
        VerticalLayout layout = new VerticalLayout();
        layout.setSizeFull();
        layout.addComponent(new Button("Klikni!");
        ...
        addComponent(layout);
    }
    ...
}
```

Navigaci mezi jednotlivými *Views* zajišťuje třída *Navigator*. Každé *UI* obsahuje svoji vlastní instanci třídy *Navigator*, která obsahuje seznam všech zaregistrovaných *Views* patřící k danému *UI*. Při vytváření *UI* je tedy potřeba všechna *View* registrovat a v rámci běhu aplikace lze mezi nimi libovolně přepínat. Programová změna obsahu stránky je zobrazena na dalším

příkladu.

```
public MainView() {
    Navigator navigator = UI.getCurrent().getNavigator();

    navigator.addView(        //přidání první stránky
        HomePageView.VIEW_NAME,
        HomePageView.class
    );
    navigator.addView(        //přidání druhé stránky
        AboutView.VIEW_NAME,
        AboutView.class
    );
    //zobraz úvodní stránku
    navigator.navigateTo(HomePageView.VIEW_NAME);
}
```

V kapitole analýza požadavků byl navržen seznam obrazovek, který odpovídal požadavkům na aplikaci. V rámci implementace je každá z těchto obrazovek implementována jako samostatné *View*. Seznam obrazovek s anglickým názvem, který odpovídá názvu použitému v aplikaci, je uveden v tabulce 4.1.

Tabulka 4.1: Seznam Views.

Obrazovka	**Titulek**	**Role**	
		USER	**ADMIN**
Přehled o nedávné a současné aktivitě serverů.	Status	✓	✓
Editace a administrace uživatelů a jejich rolí.	User management		✓
Editace a administrace serverů a přiřazení práv k serverům.	Node management		✓
Zobrazení a filtrování reportů.	Reports	✓	✓
Editace manifestů.	Manifests		✓
Zobrazení informací o serveru.	Node detail	✓	✓
Spuštění běhu agenta.			

Jednotlivá *Views* jsou viditelná uživatelům na základě přiřazených rolí. Roli přiřazuje uživateli administrátor při zakládání či editaci uživatele. Zabezpečení přístupu na základě rolí je v aplikaci implementováno pomocí *Spring Security*. Jedná se o framework, který zajišťuje autentizaci a autorizaci uživatele v rámci celé aplikace. Na uvedeném příkladu lze vidět vytvoření *beany*

pro definici připojení k databázi. Tato instance je předána jako parametr pro vytvoření služby *jdbc–user–service*, která zajišťuje získání dat pro *Spring Security*. V prvním selectu je získání všech uživatelů, kteří se mohou přihlásit k aplikaci. Druhý příkaz zajišťuje získání rolí k daným uživatelům. S těmito daty pracuje *Spring* framework po celou dobu běhu aplikace a zajišťuje jak přihlášení uživatele, tak následný přístup k jednotlivým *Views*. Přístup k *View* je řízen anotací `@Secured("ROLE")`. Takto konkrétně anotované *View* je přístupné pouze uživatelům s rolí ROLE.

```
<bean id="dataSource"
   class="org.springframework.jdbc.DriverManagerDataSource">

   <property name="driverClassName" value="${db.driver}" />
   <property name="url" value="${db.url}" />
   <property name="username" value="${db.user}" />
   <property name="password" value="${db.password}" />
</bean>

<jdbc-user-service data-source-ref="dataSource"
   users-by-username-query =
   "select email, password, active " +
    "from user_t where email=? "
   authorities-by-username-query =
   "select u.email, ur.role from user_t u " +
   "inner join user_role ur on u.userid = ur.userid " +
   "where u.email =? "
/>
```

4.2 Databáze

Aplikace ukládá veškerá data do relační databáze *PostgreSQL*. *PostgreSQL* databáze byla zvolena pro svoji snadnou instalaci a konfigurovatelnost, dobrý výkon a dobrou podporou na operačních systémech typu Linux.

Pro komunikaci aplikace s databází je využit *Java Persistence API* (JPA) framework, který umožňuje objektově relační mapování tříd (entit) na databázové tabulky. Jednotlivým tabulkám v ERA modelu odpovídají implementačně tzv. entity. Jedná se o třídy, které reprezentují databázovou tabulku a každá instance této třídy reprezentuje řádek tabulky. Jednotlivé privátní

proměnné třídy tedy odpovídají příslušnému sloupečku v databázi.

Pro manipulaci s daty je použit framework *Spring Data Core*. Tento framework řeší za programátora veškerou práci s databázovými transakcemi a také obsahuje generické metody pro vložení, úpravu či smazání záznamů. Rozhraní pro manipulaci s entitami se nazývá *CrudRepository* a je znázorněno v následujícím příkladu.

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);          (1)
    T findOne(ID primaryKey);                (2)
    Iterable<T> findAll();                   (3)
    Long count();                            (4)
    void delete(T entity);                   (5)
    boolean exists(ID primaryKey);           (6)
}
```

(1) Uložení dané entity.

(2) Vrácení dané entity dle primárního klíče.

(3) Vrácení všech řádků tabulky (entit).

(4) Vrácení počtu řádků tabulky.

(5) Odstranění dané entity z databáze.

(6) Zjištění, zda existuje entita s daným primárním klíčem.

Spring Data framework již obsahuje implementace výše zmíněných generických metod, proto se programátor nemusí o implementace těchto metod starat. Pro každou entitu je pouze potřeba vytvořit vlastní *repository*, které se implementuje jako rozhraní. Toto *repository* musí dědit *CrudRepository* rozhraní. Příklad vytvoření vlastního *repository* pro třídu (entitu) *Node* je uveden na následujícím příkladu.

```
@Repository
public interface NodeRepository extends
    CrudRepository<Node, Integer>{

    @Query("SELECT n FROM Node n WHERE n.name = :name")
    public Node findByNodeName(@Param("name") String name);


    @Query("SELECT n FROM Node n WHERE n.nodeId = :nodeId")
    public Node findById(@Param("nodeId") Integer nodeId);

}
```

Spring Data framework zařídí při spuštění programu vytvoření implementací všech definovaných *repository* pomocí *bean*. Pro získávání dat z dané tabulky je ještě možné specifikovat vlastní SQL příkazy, které slouží k výběru dat dle libovolných kritérií.

ERA model databáze lze prohlédnout v přílohách této práce. Tento model se skládá z šesti tabulek, do kterých jsou ukládána pouze nezbytná data. Data, která se mohou měnit (fakta o serverech, třídy na serverech) jsou dostupná pomocí REST API a jsou zobrazována vždy aktuální. Také místo ukládání velkých reportů do databáze jsou uloženy pouze základní parametry. Konkrétně se jedná o název reportu, čas vygenerování a výsledný status. Tyto informace jsou dostačující pro výpis do tabulek a generování grafů. Pokud by si uživatel chtěl zobrazit detail libovolného reportu, načte aplikace tento report z dostupné složky na disku a report zobrazí.

4.3 Konfigurace

Obecné nastavení pro aplikaci se nachází v souboru `Config.properties`. Tento soubor je logicky rozdělen do tří částí. V první části souboru lze najít veškerou konfiguraci, která souvisí s nástrojem Puppet. V druhé části souboru se nachází konfigurace databáze a v třetí části se nachází konfigurace mailového serveru. Konfigurační hodnoty jsou uvedeny v tabulce 4.2 a k nim je uvedena i implicitní hodnota, která je pro danou konfigurační položku nastavena. Detailní popis jednotlivých položek lze najít v sekci přílohy na konci

tohoto dokumentu.

Tabulka 4.2: Konfigurace aplikace.

Klíč	**Implicitní hodnota**
puppet.master.domain	https://puppetmaster
puppet.master.port	8140
puppet.master.report	/var/lib/puppet/reports
puppet.master.manifest	/etc/puppet/manifests
puppet.agent.port	8139
puppet.reports.history	30
db.driver	org.postgresql.Driver
db.url	jdbc:postgresql://localhost:5432/puppet
db.user	postgres
db.password	password
mail.sender.address	puppet2@email.cz
mail.sender.password	Passw0rd1
mail.smtp.server	smtp.email.cz
mail.smtp.port	465
mail.smtp.starttls.enable	false
mail.smtp.ssl.enable	true

Uvedená konfigurace je načtena při startu aplikace a po celou dobu běhu aplikace je neměnná. Při změně parametru je nutné aplikaci restartovat, aby se načetly nové hodnoty.

4.4 Obrazovky

V následující kapitole jsou popsány jednotlivé obrazovky vytvořené aplikace. Jsou zde primárně popsány funkčnosti obrazovky a použité principy implementace.

4.4.1 Status

Obrazovka *Status* se zobrazí uživateli jako první po přihlášení. Slouží k získání okamžitého přehledu o serverech a jejich stavech. Na první pohled je

možné zjistit, které servery neodpovídají definované konfiguraci či které servery jsou nedostupné. Dále lze získat informace o posledních reportech a zobrazit statistiku reportů za definovaný časový interval.

Aktuální stav serverů

Pro zjištění okamžitého stavu infrastruktury slouží tabulka se seznamem možných stavů. Ke každému stavu je poté uveden počet serverů, jejichž aktuální stav odpovídá právě danému stavu. Možné stavy jsou uvedeny v tabulce 4.3.

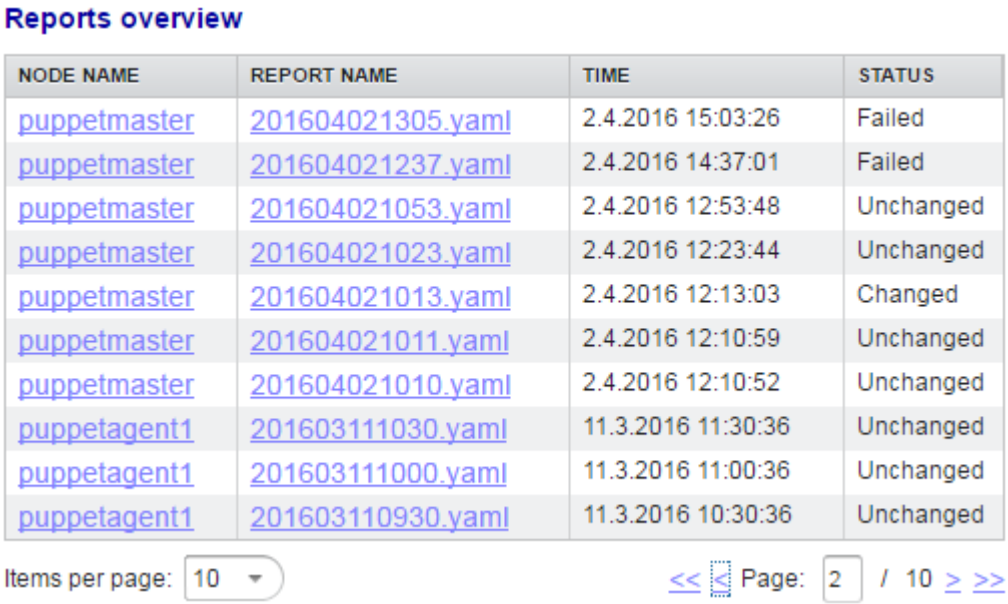
Tabulka 4.3: Možné stavy běhů na agentech.

Status	Popis
CHANGED	Během posledního běhu byl alespoň jeden *resource* změněn. Stav serveru před spuštěním běhu tedy neodpovídal popsané konfiguraci a byl obnoven Puppetem.
UNCHANGED	Stav serveru před spuštěním běhu odpovídal definované konfiguraci a nebylo třeba provádět změny na serveru.
FAILED	Puppet nemohl dostat systém do požadovaného stavu. Tento stav může být způsoben například nedostatečnými právy, překlepem v konfiguračních souborech, chybějícími zdroji atd.
UNRESPONSIVE	Puppet master za poslední půlhodinu (nebo jinou hodnotu dle konfigurace) neobdržel od agenta report.
ALL	Zahrnuje všechny stavy uvedené výše.

Pokud se server nachází ve stavu UNRESPONSIVE, znamená to, že za definovanou časovou periodu (implicitně 30 minut) nebyl nalezen příslušný report. Tento případ může být způsoben například neběžícím serverem či nefunkční konektivitou. Pro podrobnější informace o posledních reportech se pod tabulkou stavů nachází výčet všech dostupných serverů, ke kterým je dále uveden čas posledního vygenerování reportu. To umožní uživateli ihned zjistit, kdy přesně byly přijaty poslední reporty od všech serverů. Tato informace je nejvíce užitečná právě pro UNRESPONSIVE servery, kde uživatel dostane časovou informaci o nedostupnosti serveru (hodiny, dny či měsíce).

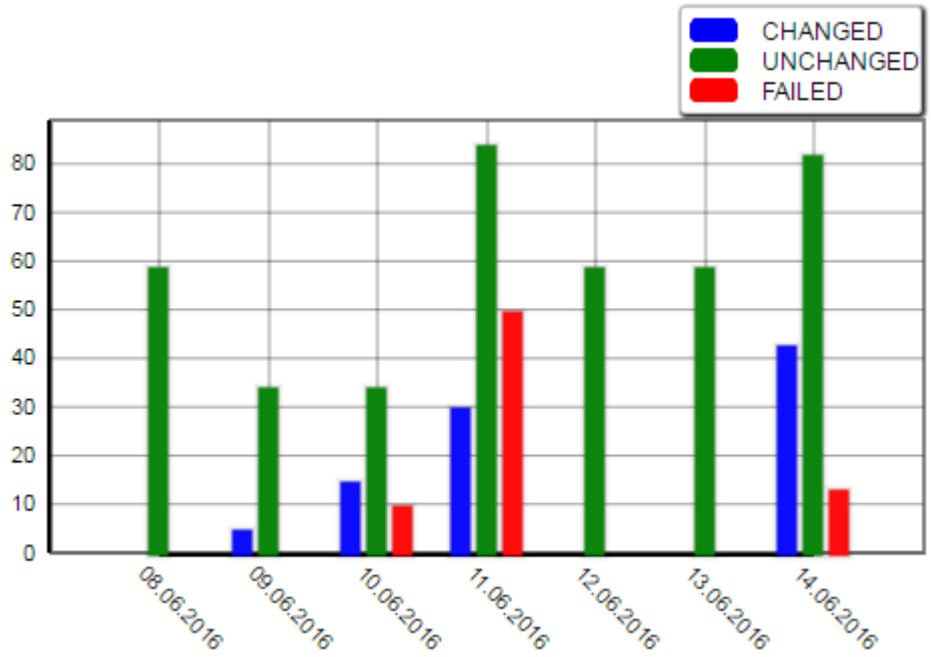
Reporty

Pro rychlý přehled o nedávných reportech slouží tabulka s přehledem posledních reportů. V tabulce jsou zobrazeny reporty za období, které lze nastavit v konfiguraci aplikace. Jak lze vidět na obrázku 4.3, informace o reportech v této tabulce jsou pouze základní. Konkrétně se jedná o název stroje, jméno reportu, čas vygenerování a výsledný stav běhu. Po kliknutí na jméno reportu je otevřena nová obrazovka s konkrétními detaily reportu. Také název serveru slouží jako odkaz na jeho detailnější popis.



Obrázek 4.2: Přehled reportů.

Dále se na obrazovce nachází sloupcový graf, který zobrazuje informace o přijatých reportech pro jednotlivé dny. Reporty jsou zde pro každý uvedený den rozlišeny dle stavu a jejich celkový počet je vynesen na svislé ose. Časové období, za které jsou reporty nastaveny lze nastavit v konfiguraci aplikace. Ukázka zmíněného grafu je zobrazena na obrázku 4.4.



Obrázek 4.3: Denní statistika reportů.

Kontrola nových reportů

Reporty jsou zpracovávány ze složky *puppet.master.report* uvedené v konfiguraci aplikace. V této složce existuje pro každý server podsložka, která odpovídá jménu serveru a obsahuje uložené reporty právě pro daný server. Kontrolu existence nových reportů provádí periodická úloha, která je součástí frameworku *Spring Core*. Její konfiguraci lze vidět na následujícím příkladu.

```
<task:scheduled-tasks scheduler="reportScheduler">
    <!-- check reports every 15 minutes -->
    <task:scheduled ref="reportCheck" method="check"
        fixed-delay="900000" />
</task:scheduled-tasks>
```

Úloha je definována pomocí třídy *ReportCheck* a kód úlohy je uveden v metodě *check()*. Konfigurační čas pro opakování úlohy je v milisekundách a odpovídá 15 minutám. Každých 15 minut je spuštěna kontrola reportů a nové reporty jsou uloženy do databáze.

Kontrolu nových reportů lze provést také manuálně pomocí tlačítka na hlavní obrazovce.

Pro procházení složek a čtení z disku je použita knihovna *Java NIO*. Pro zpracování reportů je dále použita knihovna *Yamlbeans*, která dokáže převést soubor na POJO třídu.

4.4.2 User management

Tato obrazovka je přístupná pouze uživatelům s rolí ADMIN. Slouží k zakládání nových uživatelů a také k jejich editaci. Povinné informace o uživateli jsou následující:

- Emailová adresa,
- jméno uživatele,
- příjmení uživatele,
- heslo,
- uživatelská role,
- status.

Emailová adresa slouží jako jednoznačná identifikace pro přihlášení uživatele a také pro obnovu hesla. V případě zapomenutí hesla je uživateli zaslán na emailovou adresu vygenerovaný odkaz, pomocí kterého je možné heslo změnit. Uživatel je tedy identifikován emailovou adresou a heslem, které je uloženo v databázi jako otisk hashovací funkce *Bcrypt*. Ta je implementována ve *Spring Security* třídou *BCryptPasswordEncoder*.

Dále je nutné přiřadit uživateli jednu z rolí (ADMIN, USER). Tyto role mají vliv na zobrazení jednotlivých *View*, viz tabulka 4.1. Dále je možné nastavit status uživatele. Pomocí statusu je možné zneaktivnit uživatelský účet a v případě potřeby ho opět aktivovat.

4.4.3 Node management

Tato obrazovka slouží pro výběr serverů, které bude aplikace zobrazovat. Uživateli jsou v tabulce nabídnuti všichni agenti, kteří jsou připojeni k Puppet masterovi. Na administrátorovi je ponecháno rozhodnutí, které servery bude aplikace sledovat a které ne. Zvolené servery lze odebírat a opětovně zase přidávat. Po odebrání serveru jsou smazána veškerá uživatelská práva na daný server a také veškeré reporty, které jsou uložené v databázi.

Vybrané servery se objeví v tabulce pro nastavení práv pro jednotlivé uživatele. Pro vybraného uživatele se v tabulce objeví všechny sledované servery a je na administrátorovi, jaká práva uživateli přidělí. Administrátoři mají automaticky práva na všechny servery, proto je toto nastavení účinné pouze pro uživatele s rolí USER. Dostupná práva jsou uvedena v tabulce 4.4.

Tabulka 4.4: Dostupná práva uživatelů na nody.

Právo	**Popis**
NONE	Uživatel nemá žádné právo na server a v aplikaci tento server vůbec nevidí.
READ	Uživatel vidí aktuální stav serveru, všechny reporty, detail reportů, informace o serveru, třídy a fakta.
READ+RUN	Nastaví stejná práva jako READ, ale uživatel může navíc manuálně spustit běh agenta na daném serveru.

4.4.4 Reports

Tato obrazovka slouží k zobrazení reportů dle kritérií zadaných uživatelem. Obrazovka obsahuje tabulku reportů, které jsou filtrovány dle zadaných kritérií. Filtrování lze provést pomocí následujících kritérií:

- Server či množina serverů,
- stav reportů,
- datum od,
- datum do,

- libovolnou kombinací výše uvedených kritérií.

Na základě vybraných kritérií se zobrazí tabulka s reporty, která obsahuje základní údaje o reportu. Tabulka má nastavitelný počet zobrazených reportů a umožňuje stránkování. Z tabulky reportů se lze odkazem dostat na detail vybraného reportu.

Detail reportu

Detail reportu umožňuje zobrazit uživateli podrobné informace o běhu serveru. Obrazovka obsahuje tři logicky oddělené stránky, mezi kterými se lze navigovat pomocí samostatných tabů. Konkrétně se jedná o metriky, logy a události.

První stránka s metrikami zobrazuje souhrnné informace o *resources*, časech, událostech a změnách. Celkem obsahuje čtyři tabulky, které slouží k získání souhrnného přehledu o počtech a časech reportu. Přehled tabulek a jejich popis je uveden v tabulce 4.5.

Tabulka 4.5: Metriky reportu.

Tabulka	**Popis**
Resources	Obsahuje seznam všech možných stavů *resources* a k nim příslušný počet *resource*, jejichž stav po skončení běhu odpovídal příslušnému stavu. Možné stavy jsou *Skipped, Failed, Failed to restart*, *Restarted, Changed, Out of sync* a *Scheduled*.
Time	Obsahuje seznam *resource* typů (*User, File, Package, Service*, *Config retrieval*, ...) a k nim časové informace o tom, jak dlouho jednotlivé typy trvalo zpracovat při běhu agenta.
Events	Zobrazuje počet a celkový stav událostí, které jsou vygenerovány jednotlivými *resource* při běhu agenta. Tabulka zobrazuje počty událostí k výsledným stavům (*Failure, Success*).
Changes	Obsahuje počet *resource*, které byly změněny při běhu agenta.

Druhá stránka s logy obsahuje logy Puppet agenta, které byly vytvořeny při generování reportu. Každý jednotlivý log má svoji prioritu, která je vybrána z následujícího výčtu (*info, notice, warning, error*). Dle priority je možné logy řadit a rychle vyhledat například všechny logy s nejvyšší prioritou *error*.

Třetí obrazovka s událostmi zobrazuje veškeré události, které se odehrály během generování reportu. Události jsou rozdělené do sekcí dle konkrétního stavu (*Changed, Failed, Skipped, Out of sync, Other*) a každá sekce obsahuje detailní popis jednotlivých událostí.

4.4.5 Manifests

Tato obrazovka je přístupná pouze administrátorům a slouží k editaci konfiguračních souborů Puppetu, tzv. manifestů. Ty jsou uloženy ve složce *puppet.master.manifest*, která je uvedena v konfiguračním souboru aplikace.
Obrazovka obsahuje interaktivní tabulku, pomocí které je možné přehledně zobrazit a procházet adresářovou strukturu s manifesty. Pro procházení podsložek dovoluje tabulka rozbalovat jednotlivé podsložky a zpětně jejich obsah skrýt. Při kliknutí na soubor v tabulce je otevřen editor, pomocí kterého lze prohlížet obsah souboru a také ho modifikovat. Editor obsahuje číslování řádků pro lepší orientaci v souboru. K dispozici jsou tlačítka pro uložení provedených změn (*save*) či obnovení původní verze souboru (*discard*).

4.4.6 Node detail

Node detail obrazovka slouží pro získání podrobných informací o daném serveru. Obsahuje graf s přehledem reportů za určité období. Tento graf je stejný jako na domovské obrazovce *Status*, kde graf zobrazuje statistiku všech serverů, na které má daný uživatel právo. Graf na této obrazovce zobrazuje statistiku o reportech pouze pro vybraný server. Dále je uvedena tabulka s posledními reporty pro daný server.

Pomocí tlačítka *Report run* je možné manuálně vytvořit požadavek na spuštění běhu agenta. Po stisknutí tlačítka je zavoláno přímo REST API agenta, který daný požadavek obslouží. Po dokončení požadavku odešle agent report Puppet masterovi, který ho uloží do příslušné složky dle konfigurace. Pro okamžité zobrazení reportu je vhodné manuálně zkontrolovat složku s reporty. To lze provést tlačítkem na obrazovce *Status*. Po zpracování reportu aplikací lze tento report vidět v aplikaci.

Na další stránce *Facts* je uvedena tabulka s veškerými fakty, která jsou na daném serveru dostupná. Tato fakta jsou v okamžiku zobrazení získána pomocí REST API Puppet mastera, takže jsou hodnoty vždy aktuální. Pokud není

server Puppet master dostupný, je zobrazena příslušná chybová hláška o nedostupnosti.

Třetí stránka *Classes* obsahuje seznam všech tříd, které jsou na daném serveru aplikovány. Tento seznam je stejně jako v případě faktů získáván pomocí REST API mastera, proto vždy odpovídá aktuálnímu stavu serveru. V případě nedostupnosti Puppet mastera je zobrazena příslušná hláška o nedostupnosti.

5 Ověření instalace a konfigurace

V této kapitole je uvedena instalace nástroje Puppet, konfigurace nástroje a také instalace vytvořené aplikace. Dále je uveden postup testování aplikace. Instalační postup a jednotlivé příkazy uvedené v této kapitole jsou otestovány na operačním systému *Debian 8.2 (Jessie)*.

5.1 Puppet master

Před samotnou instalací mastera je důležité zvolit jméno, pod kterým bude master v síti vystupovat. Toto jméno je většinou použito také pro certifikační autoritu, která je v rámci instalace na serveru vytvořena. V tomto návodu bude použito jméno *puppetmaster*. Toto jméno je potřeba svázat s adresou serveru. V souboru `/etc/hosts` je potřeba přidat řádek s odpovídající IP adresou a zvoleným názvem. Již dopředu je vhodné přidat také agenty, kteří budou připojeni k danému masterovi:

```
#/etc/hosts - puppetmaster
10.0.2.15     puppetmaster
10.0.2.16     puppetagent1
```

Nainstalování Puppet mastera lze provést pomocí nástroje *apt* pro instalaci balíků a knihoven:

```
$ apt-get install puppet puppetmaster
```

Předchozím příkazem jsou nainstalovány veškeré potřebné závislosti, balíčky a knihovny. Pro správný běh nástroje je dále potřeba provést několik konfiguračních nastavení. Nastavení nástoje Puppet se provádí v konfiguračním souboru `/etc/puppet/puppet.conf`. Tento konfigurační soubor je rozdělen do několika sekcí:

- `main` – globální sekce nastavení, které je platné pro všechny příkazy a služby.
- `master` – sekce platná pro službu Puppet master a pro příkazy týkající se certifikátů na masterovi.

- `agent` – sekce platná pro službu Puppet agent.

V konfigurační sekci [master] je nutné nastavit jméno serveru, které bylo zvoleno před začátkem instalace. To lze provést přidáním následujícího řádku:

```
dns_alt_names=puppetmaster
```

Dále je potřeba nastavit toto jméno také pro certifikační autoritu. To lze provést přidáním řádku:

```
certname=puppetmaster
```

V tuto chvíli zbývá ještě nastavit přístup k REST API Puppet mastera. Veškeré přístupy k REST API se nastavují v souboru `/etc/puppet/auth.conf`. Pro přístup aplikace k REST API Puppet mastera je potřeba mít v konfiguračním souboru povolen přístup k základní cestě `/`. Ten lze nastavit pro specifickou IP adresu, ale také například pro název serveru odpovídající regulárnímu výrazu. Konkrétní případ nastavení IP adresy Puppet mastera vypadá například takto:

```
path /
auth any
allow_ip 10.0.2.0/24      # povolí adresy 10.0.2.*
#allow puppetmaster        # povolí přístup pro stroj
#allow *                   # povolí přístup všem
```

Tím je základní instalace Puppet mastera kompletní.

5.2 Puppet agent

Před instalací agenta je důležité zvolit jméno, pod kterým bude agent v síti vystupovat. Toto jméno je stejně jako u předchozí instalace mastera potřeba svázat s adresou serveru v souboru `/etc/hosts`. V tomto instalačním návodu je zvoleno jméno agenta *puppetagent1*. Po dokončení konfigurace souboru `/etc/hosts` na agentovi je vhodné ověřit správnost souboru také na masterovi a případně přidat položku nového agenta, pokud tak nebylo učiněno během instalace mastera. Příklad nastavení souboru `/etc/hosts` na agentovi vypadá následovně:

```
#nastavení stroje puppetagent1
10.0.2.16     puppetagent1
10.0.2.15     puppetmaster
```

Nainstalování Puppet agenta lze podobně jako v případě mastera provést pomocí nástroje *apt*:

```
$ apt-get install puppet
```

Tímto příkazem jsou nainstalovány veškeré potřebné balíčky a knihovny potřebné pro instalaci agenta. Po instalaci agenta je nutné nastavit v konfiguračním souboru adresu mastera, ke kterému se má agent připojit. Do sekce [agent] v konfiguračním souboru se přidá následující parametr:

```
server=puppetmaster
```

Dále je potřeba nastavit jméno agenta (certifikátu), pod kterým bude agent komunikovat s Puppet masterem. To se provede přidáním parametru:

```
certname=puppetagent1
```

Jako poslední nastavení je třeba povolit přístup k REST API agenta. Tato možnost je nutná k manuálnímu spouštění běhu agenta prostřednictvím aplikace. K povolení naslouchání služby na zvoleném portu slouží parametr:

```
listen=true
```

Poslední nastavení spočívá v úpravě souboru `/etc/puppet/auth.conf` pro přístup k REST API agenta. Úprava souboru je stejná jako v případě mastera a spočívá v povolení přístupu k základní cestě `/`.

5.3 Propojení agenta s masterem

V tuto chvíli je instalace Puppet mastera a agenta téměř kompletní. Pro správné fungování obou nástrojů je ovšem potřeba tyto nástroje vzájemně propojit. Pro vzájemnou komunikaci agenta s masterem je po samotné konfiguraci nutné manuální potvrzení výměny certifikátů mezi stroji. V prvním kroku musí agent poslat masterovi požadavek na vygenerování certifikátu

podepsaného vytvořenou certifikační autoritou. Požadavek je nutné provést spuštěním následujícího příkazu na agentovi:

```
$ puppet agent –waitforcert 60 –test
```

Na masterovi je poté možné zobrazit příchozí požadavky a manuálně je potvrdit. K zobrazení aktuálních požadavků na připojení slouží příkaz:

```
$ puppet cert –list
```

Po zobrazení všech požadavků je potřeba potvrdit vytvoření a podepsání certifikátu pro vybraný server. K tomu slouží příkaz:

```
$ puppet cert –sign puppetclient
```

Tím je výměna certifikátu ukončena a agent s masterem jsou propojeny.

5.4 Instalace vytvořené aplikace

5.4.1 PostgreSQL

Data aplikace jsou ukládána do relační databáze *PostgreSQL*. Pro vývoj a testování byly použity verze *PostgreSQL 9.3* a *PostgreSQL 9.4*, které byly v době testování aktuální. Databáze byla instalována na stejný stroj jako Puppet master, tedy *Debian 8.2 (Jessie)*. Instalaci databáze lze provést pomocí nástroje *apt* zadáním následujícího příkazu:

```
$ apt-get install postgresql postgresql-client
```

Tento příkaz zajistí základní instalaci databázové služby. Dále je potřeba zvolit uživatelské jméno a databázi, do které se bude aplikace připojovat. Stejné údaje je poté nutné nastavit v konfiguraci aplikace. V tomto návodu bude použit implicitní uživatel *postgres* a stejně pojmenovaná databáze. Pro implicitního uživatele je nejprve nutné nastavit nové heslo, které je také součástí konfigurace aplikace. To lze provést následujícími příkazy:

```
#Přepnutí za uživatele v terminálu.
$ su - postgres
#Spuštění příkazu pod uživatelem postgres.
$ psql
#Změna hesla uživatele postgres.
$ ALTER USER "postgres" WITH PASSWORD 'password';
```

S tímto uživatelským účtem je možné se připojit k databázi např. pomocí nástroje *pgAdmin*, který poskytuje grafické rozhraní pro databázi *PostgreSQL*. V rámci instalace aplikace je nutné vytvořit několik tabulek a sekvencí. Dále je nutné vytvořit administrátorský účet, pomocí kterého lze poté aplikaci nakonfigurovat. Veškeré uvedené kroky lze provést instalačním skriptem, který je přiložen ve zdrojových kódech k této práci.

5.4.2 Tomcat 7

Pro běh aplikace je potřeba servletový kontejner, který běží v rámci webového serveru. Pro vývoj i testování byl použit *Tomcat 7*. Pro jeho instalaci lze opět použít nástroj *apt*:

```
$ apt-get install tomcat7
```

Základní balíčky a služba jsou nainstalovány předchozím příkazem. Součástí instalace je vytvoření uživatele *tomcat7*, pod kterým poté služba běží.
Pro snadnější ovládání webového serveru a nasazování aplikací lze doinstalovat webové rozhraní. Pomocí webového rozhraní lze poté spravovat celý webový server a jeho aplikace. Instalace nástroje se provede příkazem:

```
$ apt-get install tomcat7–admin
```

Pro tento administrační nástroj je nutné vytvořit uživatele s příslušnými rolemi pro správu jednotlivých aplikací. Nastavení se provádí v konfiguračním souboru `tomcat-users.xml` a uživatele s právy admina lze vytvořit následovně:

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

Po restartu služby *Tomcat 7* se lze přihlásit do instalovaného webového rozhraní, které je dostupné na portu 8080. Přihlašovací údaje odpovídají údajům uloženým v konfiguračním souboru `tomcat-users.xml`. Po přihlášení je možné aplikaci nasadit nahráním *war* souboru, který obsahuje přeložené kódy aplikace a je přiložen k této práci.

5.4.3 Konfigurace aplikace

Po nasazení aplikace je nutné nastavit správné hodnoty v konfiguraci aplikace. Po nasazení na *Tomcat* server se aplikace nachází v cestě `/var/lib/tomcat7/webapps/zcu-puppet-2015/`. U souborů aplikace lze nalézt konfigurační soubor `Config.properties`, ve kterém je potřeba správně nastavit zejména následující hodnoty:

- Adresa Puppet mastera a port,
- port, na kterém poslouchají agenti,
- cesta ke složce s reporty a manifesty na disku,
- připojení k databázovému serveru,
- konfigurace SMTP serveru.

Jako poslední je třeba nastavit aplikaci práva na soubory na disku, se kterými aplikace pracuje. Konkrétně se jedná o složku s reporty `/var/lib/puppet/reports` a složku s manifesty `/etc/pupppet/manifests/`. Oba tyto soubory vlastní uživatel *puppet* a je nutné přiřadit práva uživateli *tomcat7*. To lze provést tak, že se uživateli *tomcat7* přidá skupina *puppet*. Příkaz je následující:

```
$ usermod -a -G puppet tomcat7
```

Po tomto nastavení by měla být aplikace funkční a přístupná. Pro přihlášení lze použít jeden ze dvou účtů, které byly vytvořeny v rámci instalace databáze. Je možné přihlásit se pod následujícími uživateli:

- admin@admin.com/admin (ADMIN),
- user@user.com/user (USER).

5.5 Testování

Testování aplikace probíhalo v prostředí virtualizovaných serverů. Pro testování byl použit nástroj *VirtualBox*, který dokáže vytvářet a spravovat virtualizované systémy. V tomto nástroji byla také nakonfigurována NAT síť, do které byly připojeny všechny testovací servery. Pro vývoj a testování aplikace bylo spravováno celkem deset serverů, na kterých byl nainstalován nástroj Puppet. Puppet agent je běžně nainstalován i na masterovi, takže je v tomto počtu také zahrnut.

Uživatelské role a práva

V rámci instalace databáze jsou vytvořeni uživatelé s rolí USER a ADMIN. Tito uživatelé byli použiti při testování aplikace. Nejprve bylo ověřeno, že jsou pro oba uživatele zpřístupněné pouze ty obrazovky, na které mají dle své uživatelské role právo. Výsledek kontroly odpovídal tabulce 4.1, která obsahuje seznam *Views* s uvedenými uživatelskými rolemi.
Dále byla uživateli s rolí USER přidána práva na pět vybraných serverů. Na uživateli přístupných obrazovkách bylo zkontrolováno, že zobrazené informace se týkají pouze serverů, pro které má daný uživatel nastavená práva. Jednalo se o tabulky s reporty, tabulku výčtu serverů, graf reportů a tabulku celkového stavu serverů.
Tlačítko pro spuštění běhu agenta na detailu serveru bylo přístupné pouze s nastavenou rolí READ+RUN. S přiřazenou rolí READ se tlačítko vůbec nezobrazilo a nebylo tedy možné spustit běh na agentovi.

Správa serverů

Ke správě serverů slouží obrazovka *Node Management*, která je přístupná uživatelům s rolí ADMIN. Na této obrazovce jsou zobrazeny servery připojené k Puppet masteru. Všechny záznamy o serverech byly úspěšně uloženy do databáze po kliknutí na příslušná tlačítka. Také byla otestována možnost odebírání záznamů z databáze. V tomto případě se server znovu objevil jako možný k uložení a byly smazány veškeré reporty a nastavená uživatelská práva pro daný server. Na dané obrazovce byla dále nastavena různá práva na servery pro uživatele s rolí USER. Správná reakce na nastavená práva byla úspěšně zkontrolována v předchozím bodě.

Editace manifestů

Na obrazovce *Manifests* bylo nejprve zkontrolováno, že jsou zobrazeny pouze ty soubory, na které má dle instalace právo uživatel *tomcat7*. Zobrazení souborů odpovídalo právům nastaveným pro skupinu *puppet*, která je přiřazená právě uživateli *tomcat7*. Pro všechny soubory byla vyzkoušena editace souboru s následným uložením, ale také smazáním nově vytvořeného obsahu před samotným uložením. Po editaci souboru byl pro kontrolu manuálně vyvolán běh agenta a dle vygenerovaného reportu byla změna promítnuta na serveru. Konkrétně se jednalo o vytvoření uživatele. Existence vytvořeného uživatelského účtu byla následně ověřena připojením na server a kontrolou souboru `/etc/passwd`.
Pro kompletní otestování práv manifestů bylo pro vybrané soubory odebráno právo na zápis pro skupinu *puppet*. Následně byl proveden pokus o změnu obsahu těchto souborů pomocí aplikace. Na obrazovce se zobrazila chybová zpráva o nedostatečných právech.

5.5.1 UNIT testy

Významnou informaci při hromadné správě serverů pomocí nástroje Puppet představuje zpětná vazba od spravovaných serverů. Ta je v Puppetu řešena pomocí reportů, které agenti periodicky odesílají na mastera. Z těchto reportů lze poté zjistit současný stav definovaných *resources*, metriky, logy a události, které se odehrály při posledním běhu. Také lze dle času posledního reportu poznat aktivitu serveru.

Zpracování reportů je velmi důležitou funkcionalitou vytvořené aplikace a je klíčové pro její používání. Z toho důvodu bylo vytvořeno několik jednotkových testů, které se zaměřují právě na správné zpracování reportu. Pro různé reporty lze ověřit zpracování následujících informací:

- Zpracování reportu a konverze na POJO třídu,
- verze konfigurace,
- název prostředí pro Puppet agenta,
- název serveru a také certifikátu,
- instalovaná verze Puppetu,
- výsledný status reportu po skončení běhu,
- čas generování reportu a celková doba běhu,
- načtení logů,
- načtení metrik.

Všechny testy byly úspěšné a uložené a zobrazené hodnoty odpovídaly hodnotám uloženým v reportech.

6 Diskuze

Cílem této práce bylo vytvoření aplikace pro správu komunitní verze nástroje Puppet. Během analýzy požadavků na aplikaci se vycházelo z již existujících nástrojů *Puppet Enterprise Console* a *Foreman*. Během vývoje aplikace se na trhu objevil ještě další nástroj *Puppet Explorer*, který spíše než ke správě Puppetu slouží k monitorování stavu infrastruktury spravované Puppetem. V následujícím textu je shrnuto vytvořené řešení, které je dále porovnáno se zmíněnými existujícími nástroji.

6.1 Dosažené výsledky

Výsledkem práce je webová aplikace napsaná ve frameworku *Vaadin*. Aplikace využívá pro ukládání dat *PostgreSQL* databázi. Žádné jiné další nástroje či dodatečné závislosti nejsou pro běh aplikace potřeba.
Veškeré informace o serverech a infrastruktuře, které jsou zpracovávány a zobrazovány v aplikaci, jsou získávány pomocí REST API Puppet mastera, nebo přímo z reportů, které jsou uloženy v definované složce přímo na Puppet masterovi.

Pomocí vytvořené aplikace lze sledovat, případně upravovat, stav serverů v infrastruktuře spravované Puppetem. Na úvodní stránce jsou přehledně zobrazeny informace o stavu jednotlivých serverů. Na první pohled lze vidět, které servery odpovídají konfiguraci definované Puppetem či na kterých serverech nastaly nějaké události či chyby. Podrobné informace o serveru a detailní rozepsání jednotlivých operací, které se odehrály na Puppet agentu, je obsaženo v reportech, které jsou uloženy na masterovi. Výsledná aplikace umí tyto reporty zpracovat, přiřadit je ke konkrétnímu serveru a přehledně zobrazit veškeré obsažené informace.

Pro rychlou změnu konfigurace na serveru či serverech lze využít implementovaný editor, pomocí kterého lze upravovat manifesty. Po úpravě manifestu není změna okamžitě promítnuta na daný server, ale projeví se až po skončení běhu Puppet agenta. Puppet agent při běhu zajišťuje konfiguraci serveru dle obdrženého katalogu. Implicitně je běh plánován každých 30 minut, ale lze použít libovolnou periodu opakování. Pomocí aplikace je možné vynutit běh agenta okamžitě a výsledek běhu si zobrazit v obdrženém reportu.

Aplikace umožňuje definovat role k jednotlivým uživatelům. Uživatel s rolí ADMIN má plnou kontrolu nad servery, manifesty a uživatelskými účty. Může přidávat agenty do aplikace a také je zpětně odebírat. Má možnost editovat manifesty a tím měnit konfigurace jednotlivých serverů. Dále má plnou kontrolu nad uživatelskými účty.
Uživatelé s rolí USER mohou pouze sledovat stav serverů a zobrazovat reporty pro servery, na které mají explicitně přiřazená práva. Základní právo READ slouží ke zpřístupnění všech informací o daném serveru. Rozšířené právo READ+RUN dovoluje navíc vynucení běhu agenta.

6.2 Porovnání s existujícími nástroji

Aktuální stav serverů

Nástroj Puppet agent má na starost konfiguraci serveru do stavu, který odpovídá poskytnutému katalogu. Puppet agent řeší veškerou konfiguraci na serveru a je za ní zodpovědný. Zpětnou vazbu ohledně aktuálního stavu zasílá formou reportu Puppet masterovi, který dle své konfigurace může report uložit na disk, odeslat na zvolenou adresu či jinak zpracovat.
Všechny porovnávané konfigurační nástroje používají ke zjištění aktuálního stavu serverů právě tyto reporty. Reporty obsahují výčet *resources* s odpovídajícím stavem, logy vykonaných či neúspěšných změn, veškeré události a metriky. Nástroje *Puppet Explorer* a *Puppet Enterprise* používají pro ukládání reportů *PuppetDB* databázi. Nástroj *Foreman* používá svůj vlastní procesor pro zpracování reportů, které následně uloží do relační databáze. Vytvořená aplikace ukládá data z reportů do *PostgreSQL* databáze. Do databáze se ovšem ukládají základní informace (čas vygenerování, název a výsledný stav) a v případě požadavku na zobrazení reportu jsou zbylé informace načteny z definované složky na disku.
Přestože jsou data v porovnávaných nástrojích uložena v různých formátech, v aplikacích jsou zobrazována podobně. Z posledního obdrženého reportu pro daný server je zjištěn aktuální stav všech *resource* a tím vlastně také celého serveru. Dle času vygenerování posledního reportu lze dále určit, zda server odeslal report v pravidelném intervalu a je–li tedy dostupný. Dále report obsahuje metriky, logy a události, které nastaly při posledním běhu agenta. Všechny tyto informace jsou také zobrazovány všemi srovnávanými nástroji.

Detail serveru

Detail serveru poskytuje základní informace o stavu serveru, jeho historii a také aktuální konfiguraci. Pro vyhledání konkrétního serveru či serverů lze u existujících nástrojů použít možnost filtrů pro nalezení serverů, např. dle jména, architektury, operačního systému či instalovaného modulu. Vytvořená aplikace neumožňuje filtrování serverů, ale zobrazuje všechny servery v samostatné tabulce.
Detail konkrétního serveru poté obsahuje seznam posledních reportů, fakta o daném serveru, nainstalované moduly, různé statistiky běhů atd. Tyto informace jsou stejně jako v případě zobrazení aktuálního stavu uvedeny ve všech srovnávaných nástrojích. Nástroje se liší pouze odlišnou prezentací dat, ale hodnota informace je totožná.

Uživatelské role a práva

Nástroj *Puppet Explorer* nepoužívá žádné uživatelské účty či role pro přihlášení k aplikaci. Místo toho zobrazuje veškeré uložené informace o serverech bez nutnosti přihlašování. K ostatním srovnávaným nástrojům se lze přihlásit pomocí uživatelských účtů, ke kterým jsou i navíc přiřazeny různé uživatelské role. Jedná se o standardní administrační a uživatelské role, které jsou používány u většiny dnešních aplikací. Nástroje *Puppet Enterprise* a *Foreman* dovolují navíc autorizaci pomocí protokolu LDAP.

Vytvořená aplikace poskytuje navíc jako jediná nastavení práv uživatelů k jednotlivým serverům. Díky tomu je možné přiřadit uživatelům práva na libovolnou podmnožinu serverů. Lze tím například rozdělit odpovědnost za sledování většího množství serverů mezi více uživatelů.
Nástroje *Puppet Enterprise* a *Foreman* neumožňují rozlišení serverů mezi uživateli aplikace. Implementované role se vztahují na všechny uložené servery.

Změna konfigurace serveru

Jedním z implementovaných požadavků na vytvořenou aplikaci je možnost manuální editace manifestů. V aplikaci je tento požadavek řešen editorem, který dovoluje zobrazit a případně změnit obsah souboru v definované složce, ve které se nacházejí manifesty. Díky této funkcionalitě je možné editovat soubory, které ovlivňují konfiguraci jednotlivých serverů. To lze pomocí vy-

tvořené aplikace provádět z jednoho centrálního bodu bez nutnosti přihlášení k terminálu. Nástroje *Puppet Enterprise* a *Foreman* ovšem používají jinou funkčnost pro dynamickou konfiguraci serverů. Používají externí klasifikaci, kde veškeré změny a nově přidané úpravy jsou uloženy v databázi nástroje. Při následném generování katalogu na Puppet masteru je původní katalog, který je vygenerován z manifestů, modifikován (přepsán či doplněn) hodnotami z databáze.

Použití externí klasifikace přináší zjednodušení konfigurace, obzvlášť pokud je prováděno společně pro skupiny serverů. Pomocí formuláře lze vybrat modul a přiřadit ho k vybranému serveru či skupině serverů. O instalaci se už postarají příslušní Puppet agenti. Přestože je tato konfigurace velmi efektivní, stále je potřeba použité moduly instalovat či vytvářet. To je nutné provádět na Puppet masterovi manuálně před samotným nastavováním v aplikaci.

Externí klasifikace je užitečná pro rychlou a pohodlnou konfiguraci serverů, přináší ovšem také určitá rizika. Konfigurace serverů je totiž uložena na dvou místech (manifesty a databáze) a lze ji obtížněji sledovat a udržovat. Zatímco samotné manifesty lze udržovat ve verzovacích nástrojích, které dovolují sledovat jednotlivé položky v čase, hodnoty v databázi lze sledovat obtížněji. Oba nástroje neposkytují možnost exportu či zálohy konfigurace, ale je možné řešit tuto situaci exportem databázových tabulek či zálohováním databáze.

Spuštění běhu agenta

Jak již bylo napsáno výše, kromě nástroje *Puppet Explorer* umožňují všechny srovnávané nástroje měnit konfiguraci jednotlivých serverů. Tato konfigurace se ovšem projeví až po dokončení běhu agenta. Aby nemusel uživatel čekat na automatické spuštění každého běhu, implementují nástroje možnost manuálního vyvolání běhu agenta.
Puppet Enterprise používá k tomuto úkolu nástroj *MCollective*, pomocí kterého lze přímo ovládat Puppet agenta. Pomocí administračního rozhraní, které komunikuje s *MCollective* nástrojem, je možné zastavit či spustit službu Puppet agent a také zjistit její současný stav. Pro vynucení běhu agenta je využit také tento nástroj.
Foreman a vytvořená aplikace umožňují také manuální spuštění běhu agenta. K vynucení běhu ovšem používají REST API na agentu, který tuto funkčnost dovoluje. Po dokončení běhu je standardně vygenerován report, který obsahuje aktuální stav serveru.

Shrnutí

Z předchozího textu je pro zpřehlednění jednotlivých funkčností vytvořena tabulka 6.1. Z tabulky lze vyčíst možnosti jednotlivých nástrojů. Konkrétní informace o jednotlivých rozdílech jsou uvedeny v předchozím textu.

Uvedená tabulka ukazuje, že výsledná aplikace nepodporuje externí klasifikaci a správu skupin. Nástroje, které implementují externí klasifikaci, umožňují právě vytvářet skupiny a přiřazovat do nich jednotlivé servery. Externí klasifikace má totiž největší význam právě při dynamické konfiguraci skupin. Není totiž nutné přiřazovat moduly jednotlivým serverům zvlášť, ale již jednotlivým skupinám.
Vytvořená aplikace obsahuje editor manifestů, který slouží pro změnu konfigurace serverů. Tato funkcionalita nahrazuje externí klasifiaci a není tedy nutné používat skupiny pro jednotlivé servery.

Tabulka 6.1: Porovnání funkčností nástrojů.

Funkcionalita	**Puppet Enterprise**	**Foreman**	**Puppet Explorer**	**DP aplikace**
Aktuální stav serverů	✓	✓	✓	✓
Historie změn a událostí	✓	✓	✓	✓
Zobrazení reportů	✓	✓	✓	✓
Detail a informace o serveru	✓	✓	✓	✓
Spuštění běhu agenta	✓	✓		✓
Uživatelské role	✓	✓		✓
Práva pro jednotlivé servery				✓
Externí klasifikace	✓	✓		
Editace manifestů				✓
Vytváření skupin pro servery	✓	✓		

Možným rozšířením do budoucna by mohlo být verzování změn provedených v manifestech. To by bylo možné buď uchováváním předchozích verzí a jejich správou, nebo integrací již existujícího nástroje, např. nástroje GIT. Dále by bylo užitečné umožnit přiřazování serverů do uživatelem spravovaných skupin. Pro všechny servery ve skupině by bylo poté možné manuálně vyvolat běh agentů pomocí tlačítka.

7 Závěr

Cílem této práce bylo vytvoření aplikace pro správu komunitní verze nástroje Puppet. Výsledná aplikace je napsána ve frameworku *Vaadin* a využívá pro ukládání dat *PostgreSQL* databázi. K získávání dat využívá aplikace REST API Puppet mastera a reporty, které jsou uloženy v definované složce přímo na Puppet masterovi.
Vytvořená aplikace umožňuje sledovat a upravovat stav serverů. Na první pohled lze zjistit, které servery odpovídají konfiguraci definované Puppetem a na kterých serverech nastaly nějaké události či chyby. Podrobné rozepsání jednotlivých operací, které se odehrály na Puppet agentu, je obsaženo v reportech. Tyto reporty umí aplikace zpracovat, přiřadit je ke konkrétnímu serveru a přehledně zobrazit veškeré obsažené informace. Dále je možné editovat manifesty a tím měnit konfiguraci serverů. Aplikace umožňuje přihlášení s různými uživatelskými rolemi a definovanými právy na jednotlivé servery.

Výsledná aplikace je porovnána s nástroji *Puppet Enterprise Console*, *Foreman* a *Puppet Explorer*. Všechny srovnávané nástroje poskytují základní přehled o serverech a jejich současném a předchozím stavu. Kromě nástroje *Puppet Explorer* umožňují také změnu konfigurace serverů pomocí uživatelského rozhraní. Zatímco *Puppet Enterprise* a *Foreman* umožňují externí klasifikaci, vytvořená aplikace dovoluje přímo editovat manifesty v definované složce. Dalším rozdílem oproti srovnávaným nástrojům je přiřazování práv uživatelů k jednotlivým serverům. Vytvořená aplikace jako jediná umožňuje přiřazovat práva k serverům pro jednotlivé uživatele. To může být výhodné např. při velkém množství spravovaných serverů, kdy lze odpovědnost za určité servery delegovat určitým uživatelům. *Puppet Enterprise* a *Foreman* umožňují navíc filtrování a klasifikaci serverů dle implementovaného dotazovacího jazyka.
Nástroje *Puppet Enterprise* lze využít bezplatně pro správu maximálně deseti serverů. *Foreman* oproti tomu je open source řešení, které lze využít pro správu libovolného množství serverů. Oba nástroje zastupují velké projekty a lze je doporučit pro správu Puppetu. Vytvořená aplikace zastupuje minimalistické řešení, které bez nutnosti dodatečných nástrojů a závislostí poskytuje srovnatelné funkčnosti z hlediska nástroje Puppet. Aplikace byla testována s deseti agenty, nicméně počet agentů není nijak omezen.

Přehled zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DOM	Document Object Model
DSL	Domain Specific Language
ERA	Entitiy Relation Attributes
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
INI	Informal standard for configuration files
IT	Information technology
ITIL	Information Technology Infrastructure Library
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MIME	Multipurpose Internet Mail Extensions
NAT	Network Address Translation
NTP	Network Time Protocol
POJO	Plain Old Java Object
PSON	Protocol JSON
REST	Representational state transfer
SSH	Secure Shell
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
WAR	Web application ARchive
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Přílohy

Příloha 1 – Popis konfiguračních parametrů

puppet.master.domain
IP adresa či doménové jméno serveru, na kterém běží Puppet master. Hodnota musí povinně obsahovat protokol pro komunikaci s REST API (http či https).

puppet.master.port
Port pro komunikaci s Puppet masterem pomocí REST API.

puppet.master.report
Složka na serveru, kam Puppet master ukládá veškeré obdržené reporty. Tato složka obsahuje podsložky, které odpovídají jednotlivým serverům. V těchto podsložkách se poté nachází samotné reporty. Je důležité, aby aplikace měla práva k uvedené složce a také práva na čtení všech reportů.

puppet.master.manifest
Složka na serveru, kde se nachází manifesty. Manifesty mohou být modifikovány aplikací, proto musí mít aplikace dostatečná práva. Potřebná práva jsou čtení a úprava obsahu.

puppet.agent.port
Port pro komunikaci s Puppet agenty pomocí REST API. Na tomto portu komunikují všichni agenti, kteří jsou připojeni k masteru.

puppet.reports.history
Počet dní do minulosti, po které jsou reporty sledovány.

db.driver
Použitý řadič pro komunikaci s databází.

db.url
Connection string pro připojení k databázi. Tento řetězec musí obsahovat i jméno databáze.

db.user
Uživatel, pod kterým se aplikace přihlašuje k databázi.

db.password
Heslo uživatele, pod kterým se aplikace přihlašuje k databázi.

mail.sender.address
Emailová adresa, ze které jsou odesílány maily z aplikace.

mail.sender.password
Heslo k emailové adrese, ze které jsou odesílány maily z aplikace.

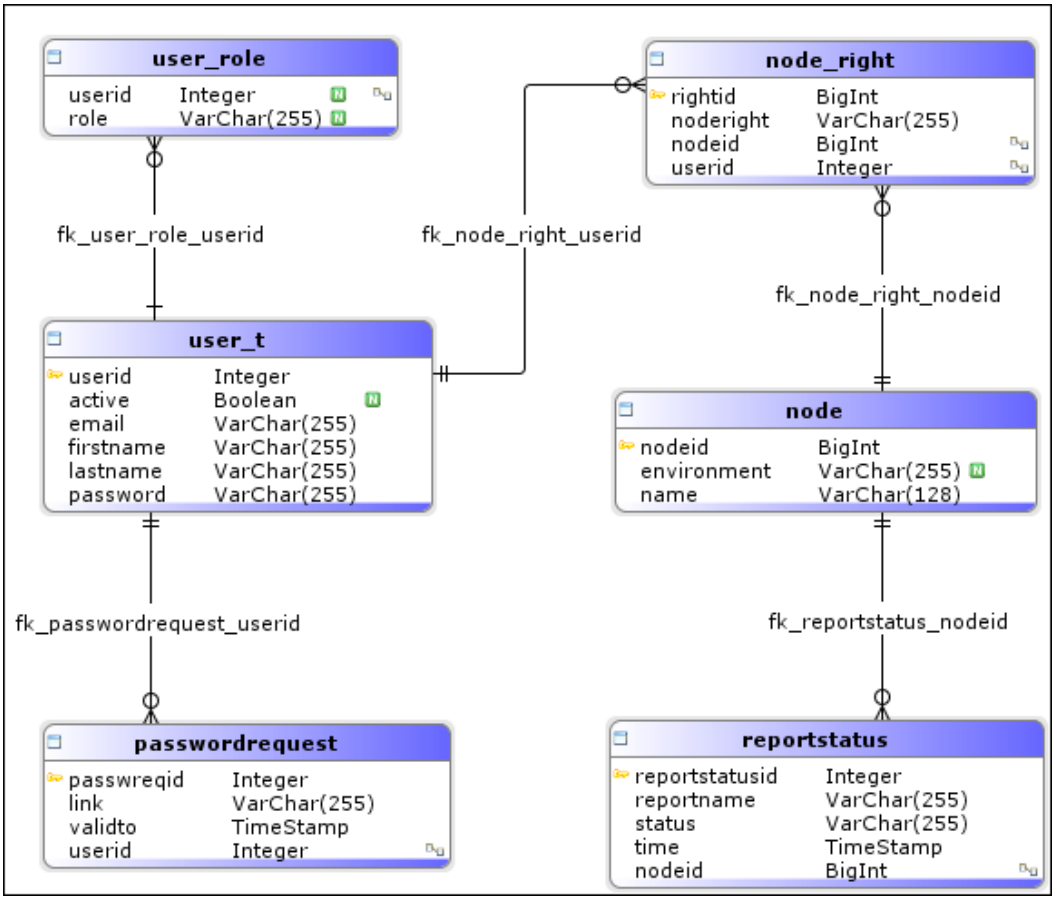
mail.smtp.server
SMTP server, který slouží k odesílání mailů.

mail.smtp.port
Port, na kterém příslušný SMTP server komunikuje.

mail.smtp.starttls.enable
TLS zabezpečení komunikace se SMTP serverem.

mail.smtp.ssl.enable
SSL zabezpečení komunikace se SMTP serverem.

Příloha 2 – Databázové schéma



Obrázek 1: Databázové schéma – ERA diagram.

Seznam obrázků

Seznam tabulek

Literatura

[1] VASANTH, R. *The Internet Holds 5 Million Terabytes Of Data, But Weighs As Much As A Grain Of Sand!* [online]. 22.4.2014 [cit. 1.4.2016]. Dostupné z: `http://dazeinfo.com/2014/04/22/internet-comprises-5-million-terabytes-data-weighs-much-grain-sand/`

[2] Pelikán, Robert. *Konfigurační management.* Praha, 2009. Diplomová práce. Vysoká škola ekonomická. Fakulta informatiky a statistiky. Vedoucí práce Jan Pour.

[3] Skála, Jiří. *ITSM & ITIL* [online]. 1.12.2011 [cit. 3.4.2016]. Dostupné z: `https://www.bestpractice.cz/cs/Best-practice/-ITSM-ITIL-.alej`

[4] Čermák, Miroslav. *ITIL tajemství zbavený.* [online]. 01.12.2009 [cit.3.4.2016] Dostupné z: `http://www.cleverandsmart.cz/itil-tajemstvi-zbaveny/`

[5] Tsalolikhin, Aleksey. *Automating System Administration with Cfengine 3.* Vertical Sysadmin [online]. 21.12.2009 [cit. 6.4.2016]. Dostupné z: `http://verticalsysadmin.com/cfengine3/`

[6] Zamboni, Diego. *Learning CFEngine 3: Automated System Administration for Sites of Any Size.* O'Reilly Media, Sebastophol, 2012. ISBN: 978-1-449-31220-6

[7] Akita, Fabio. *Chatting with Luke Kanies.* AkitaOnRails [online]. 18.12.2009 [cit. 21.5.2016] Dostupné z: `http://www.akitaonrails.com/2009/11/18/chatting-with-luke-kaines`

[8] Tsalolikhin, Aleksey. *State of the Art of Automating System Administration with Open Source Configuration Management Tools* Ver-

tical Sysadmin [online]. 9.7.2010 [cit. 26.3.2016]. Dostupné z: http://www.verticalsysadmin.com/config2010/

[9] Petko, Marek. *Hromadná správa výpočetních systémů v heterogenním prostředí.* Plzeň, 2014. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Inženýrská informatika. Vedoucí práce Michal Švamberg; Jiří Ledvina.

[10] van Deursen, Arie; Klint, Paul; Joost, Visser. *Domain-Specific Languages: An Annotated Bibliography.* ACM SIGPLAN Notices [online]. 1.6.2000 [cit. 16.2.2016]. Dostupné z: http://www.st.ewi.tudelft.nl/ arie/papers/dslbib.pdf

[11] Ford, Neal; Fowler, Martin. *Language Oriented Programming* [online]. 6.9.2007 [cit. 6.12.2015]. Dostupné z: http://nealford.com/downloads/conferences/Neal_Ford_Martin_Fowler-Language_Oriented_Programming-keynote.pdf

[12] Puppet. *Puppet Documentation – Reference Manual: Language Basics.* Puppetlabs [online]. 2016 [cit. 1.2.2016]. Dostupné z: https://docs.puppet.com/puppet/3.8/reference/lang_summary.html

[13] Puppet. *Puppet Documentation – Reference Manual: Puppet HTTP API – PSON.* Puppetlabs [online]. 2016 [cit. 2.2.2016]. Dostupné z: https://docs.puppet.com/puppet/latest/reference/http_api/pson.html

[14] Puppet. *Puppet Documentation – Reference Manual: About Puppet Reporting.* Puppetlabs [online]. 2016 [cit. 28.1.2016]. Dostupné z: https://docs.puppet.com/puppet/latest/reference/reporting_about.html

[15] Puppet. *When Puppet Reports: Part 1.* Puppet blog [online]. 2016 [cit. 6.6.2016]. Dostupné z: https://puppet.com/blog/when-puppet-reports-part-1

[16] Puppet. *Puppet Documentation – Reference Manual: Supported Platforms and System Requirements.* Puppetlabs [online]. 2016 [cit. 20.5.2016]. Dostupné z: https://docs.puppet.com/guides/platforms.html

[17] Puppet Forge. *Writing Great Modules.* Puppet Forge [online]. 2016 [cit. 20.5.2016]. Dostupné z: https://forge.puppet.com/

[18] Puppet. *Puppet FAQ - How much does Puppet Enterprise cost?* Puppet Product [online]. 2016 [cit. 3.3.2016]. Dostupné z: `https://puppet.com/product/faq`

[19] Puppet. *Puppet Documentation – Reference Manual: About Puppet Reporting.* Puppetlabs [online]. 2016 [cit. 28.1.2016]. Dostupné z: `https://docs.puppet.com/puppet/latest/reference/reporting_about.html`

[20] UpGuard. *Open Source Puppet vs. Puppet Enterprise* UpGuard Articles [online]. [cit. 6.6.2016]. Dostupné z: `https://www.upguard.com/articles/open-source-puppet-vs.-puppet-enterprise-which-is-right-for-you`

[21] Ansible. *Ansible Documentation.* Docs [online]. 2016 [cit. 25.3.2016]. Dostupné z: `http://docs.ansible.com/ansible/`

[22] Heinonen, Jussi. *Learning Puppet.* Packt Publishing, August 2015. ISBN: 9781784399832

[23] Duffy, Michael. *Puppet Reporting and Monitoring.* Packt Publishing, June 2014. ISBN: 9781783981427

[24] Puppet Explorer. *Puppet Explorer.* Oficial project page [online]. 2016 [cit. 1.6.2016]. Dostupné z: `http://puppetexplorer.io/`

[25] Puppet Explorer Demo. *Puppet Explorer Online Demo.* Puppet Explorer [online]. 2016 [cit. 6.6.2016]. Dostupné z: `http://demo.puppetexplorer.io/#/dashboard`

[26] Franceschi, Alessandro. *Extending Puppet.* Packt Publishing, June 2014. ISBN: 9781783981441