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

Diplomová práce

Hromadná správa výpočetních systémů v heterogenním prostředí

Plzeň, 2014 Marek Petko

Zadání

1. Analyzujte současné možnosti hromadné správy výpočetních systémů v heterogenním prostředí.
2. Vyberte vhodné existující řešení hromadné správy s ohledem na současný stav IT na ZČU.
3. Přizpůsobte vybrané řešení podmínkám na ZČU.
4. Otestujte realizovaný systém na skupině stanic.
5. Řešení zhodnoťte a vypracujte dokumentaci.

Prohlášení

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

V Plzni dne 10. 5. 2014

Abstrakt

Tato diplomová práce analyzuje současné možnosti hromadné správy výpočetních systémů v heterogenním prostředí. Popisuje jednotlivá existující řešení, tato řešení srovnává a doporučuje nejvhodnější řešení k nasazení v prostředí ZČU. V praktické části se zabývá otestováním zvoleného řešení v laboratorním prostředí a návrhem vhodných prostředků pro produkční nasazení. Výsledky z testování jsou zhodnoceny a na jejich základě je vybrané řešení nasazeno v produkčním prostředí. Popis nasazení je detailně popsán včetně příkladů použití. V závěru práce jsou dosažené výsledky shrnuty a zhodnoceny.

Mass management of computing systems in heterogeneous environment

This thesis analyzes current options in mass management of computing systems in heterogeneous environment. It describes existing solutions, compares them and suggests the most suitable solution to be deployed at UWB. In practical part, it deals with testing of the selected solution in a laboratory environment and with design of suitable resources for production deployment. Results of the testing are evaluated and based on this results, selected solution is deployed in production environment. Description of the deployment is described in detail including usage examples. At the end of the document, achieved results are summed up and evaluated.

Obsah

1 Úvod

Masový rozvoj internetu a online služeb vede k neustálému zvyšování počtu serverů na celém světě. Od okamžiku, kdy k zajištění jedné služby přestal stačit jeden server, jsou vyvíjena různá distribuovaná řešení jako clustery, virtualizace a v poslední době cloud. Tato řešení mají společně za cíl distribuovat zátěž mezi více uzlů, přičemž celý systém musí být vysoce odolný proti výpadku. Virtualizace a cloud toto dále rozšiřuje o vysokou přenositelnost a flexibilitu. Ve výsledku je možné navyšovat výkon systému navýšením počtu serverů v řádu minut dle obchodních požadavků. Technické překážky tedy byly překonány, ale jak spravovat takto rozsáhlé systémy efektivně? Pokud počet serverů v čase roste exponenciálně, není možné, aby počet administrátorů byl přímo úměrný počtu spravovaných serverů. Pokud by tomu tak bylo, pak by firmy jako Google, Microsoft, Akamai, Facebook, Amazon a další pravděpodobně nikdy nedosáhly úspěchu. Cílem administrátora tedy musí být oprostit se od opakujících se činností, aby se mohl soustředit na vytváření nových řešení. K tomuto účelu slouží nástroje pro distribuovaný konfigurační management.

Práce v první části popisuje hromadnou správu obecně, potom se zabývá jednotlivými existujícími řešeními. Pro každé řešení je popsán princip konfigurace, princip funkce v síťovém prostředí a podporované platformy pro použití v heterogenním prostředí. Všechna popsaná řešení jsou srovnána a je vybráno jedno řešení nejvhodnější pro prostředí ZČU. Vybrané řešení je detailně popsáno.

V druhé části práce popisuje průběh laboratorního nasazení vybraného řešení. V laboratorních podmínkách jsou otestovány dvě varianty navrženého řešení. Výsledky testování jsou zhodnoceny a na jejich základě je doporučeno řešení pro produkční prostředí ZČU. Doporučené řešení je nasazeno a jsou v něm implementovány výsledky testovacího nasazení. Dále je vyřešeno několik ukázkových příkladů.

V závěru práce jsou shrnuta vytvořená řešení a jsou zhodnoceny její výsledky.

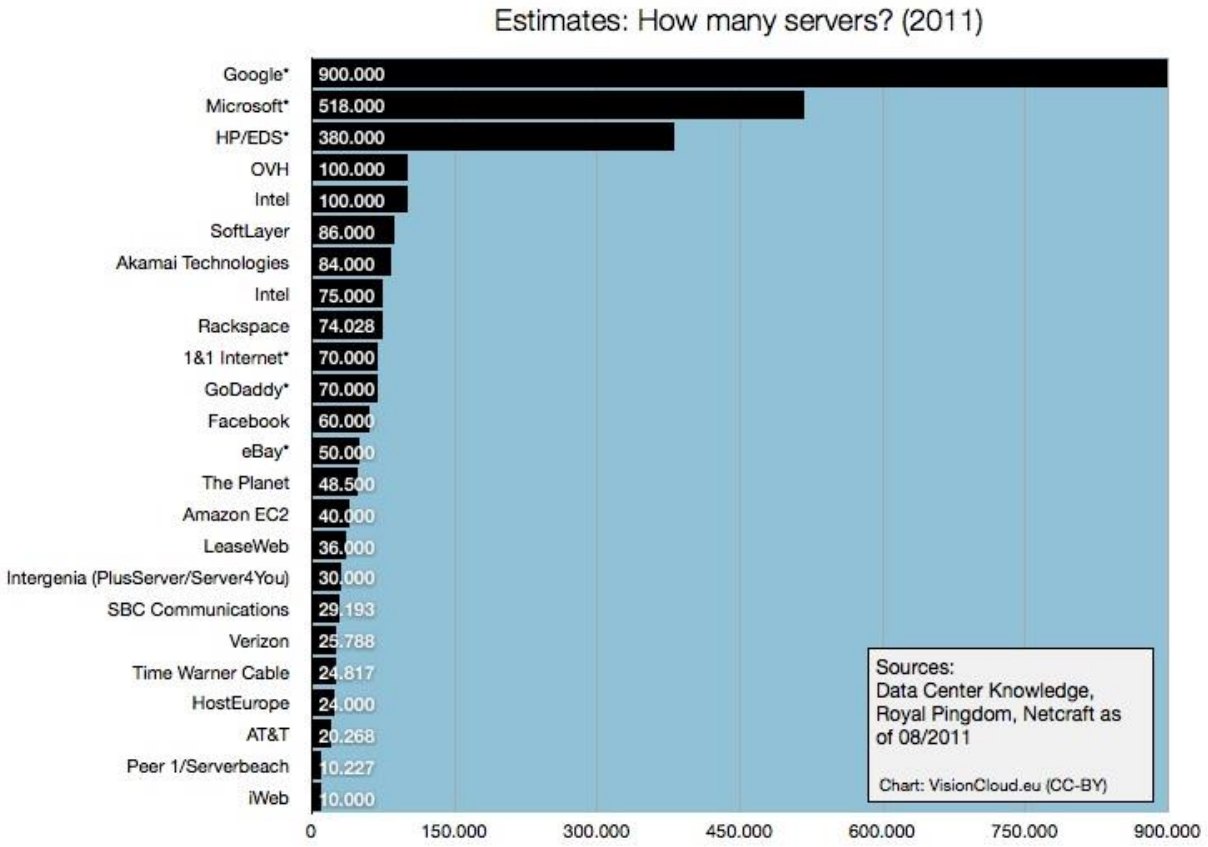
2 Teoretická část

Tato část popisuje hromadnou správu obecně, následně se věnuje popisu jednotlivých nástrojů. Nástroje jsou srovnány a nástroj vhodný pro nasazení na ZČU je popsán detailně. Součástí detailního popisu je popis funkce nástroje, jeho komponent, jazyka pro konfiguraci a bezpečnostních hledisek jeho nasazení.

2.1 Hromadná správa

Hlavní motivací pro hromadnou správu systémů je neustále rostoucí počet zařízení, která obsahují nějakou formu konfigurace. Organizace s desítkami až stovkami zaměstnanců dnes běžně používají řešení pro hromadnou správu koncových počítačů. V oblasti informačních systémů a serverů tento trend nastává až v posledních letech s příchodem clusterů, virtualizace a cloudů. Vzniká více serverů, než jsou administrátoři schopni ručně spravovat a tak přichází na řadu hromadná správa konfigurace a automatizace.

Na Obr. 1 jsou znázorněny odhadované počty serverů používaných předními světovými IT firmami v roce 2011. [1] Pokud by jeden administrátor dokázal manuálně spravovat např. 100 serverů, pak by Google musel takových administrátorů zaměstnávat 9000.



Obr. 1

Správa serverů je v porovnání se správou koncových zařízení odlišná. Servery bývají na rozdíl od koncových zařízení více účelově heterogenní, obsahují výrazně více konfigurace a požadavky na jejich správnou funkci jsou násobeny počtem uživatelů používajících službu, kterou poskytují.

Účelem hromadné správy je minimalizovat opakující se činnosti, maximálně využívat již vytvořená řešení a systematickým přístupem ke konfiguraci omezit možnost výpadku služby.

2.1.1 Automatizace v hromadné správě

Lidé jsou dobří v rozhodování, ale jsou nespolehliví při implementaci svých rozhodnutí. Naproti tomu stroje nejsou dobré v rozhodování, ale jsou velice spolehlivé při implementaci. Smyslem je ponechat každému činnost, ve které je dobrý.

Hlavním problémem při správě systémů je ztráta sebekázně. Kvalifikovaní pracovní si často myslí, že stačí být chytrý. Opak je ovšem pravdou. Chytří lidé bývají řešitelé problémů a ochotně vyřeší stejný problém několikrát za sebou a ztrácejí při tom čas a peníze.

Cílem automatizace v hromadné správě je ponechat vykonávání opakovaných činností, které musí být provedeny odpovědně a spolehlivě, počítačům. Zároveň je cílem omezit vliv ad hoc úprav systému, které narušují chápání systému ostatními administrátory. [2]

2.1.2 Standardy

IT Infrastrucure Library (ITIL) je celosvětově chápána jako reference nejlepších praktik pro procesy v IT Service Managementu. Do posledních let nebylo možné, aby ti, kteří aplikovali ITIL Framework, získali oficiální certifikaci, protože neexistoval formální standard. V roce 2000 britský úřad pro standardizaci BSI vytvořil standard BS 15000. Na konci roku 2005 byl tento standard přijat jako nový celosvětový standard ISO/IEC 20000. Přestože ISO/IEC 20000 formálně neobsahuje ITIL, popisuje procesy, které jsou v souladu s ITIL nebo jej doplňují.

Cílem ISO/IEC 20000 je poskytnout jednotný referenční standard pro podniky, které nabízí IT služby vnitřním nebo vnějším zákazníkům. Protože komunikace hraje zásadní roli v servisním managamentu, jedním z nejdůležitějších cílů standardu je vytvoření společné terminologie pro poskytovatele služeb, jejich dodavatele a zákazníky.

Configuration Management a *Change Management* jsou ve standardu ISO/IEC 20000 definovány jako součást hlavního kontrolního procesu. [3]

Configuration Management používá tzv. *Configuration Items* (CI). Configuration Items jsou spravovatelné prvky řízeného systému, mohou to být služby, hardware, software, budovy, osoby, atd. [4]

2.1.3 Hromadná správa v heterogenním prostředí

Heterogenní prostředí je značnou překážkou při nasazování většiny informačních systémů. Klasická nativní aplikace využívá API operačního systému a je přeložena pro některou architekturu procesorů. Pro zvýšení přenositelnosti aplikací jsou vytvářeny další abstraktní vrstvy, které mají zakrýt odlišnosti jednotlivých platforem.

V posledních letech citelně narůstá počet zařízení, která nepoužívají tradiční procesorovou architekturu x86, ale používají energeticky úspornější procesory ARM. Nárůst počtu mobilních zařízení s těmito procesory umožnil jejich všeobecné rozšíření, nově tak mohou

být použity i pro servery jako varianta šetrná k životnímu prostředí. [5] Současně narůstá počet spotřebitelských zařízení, která obsahují nějaký operační systém. Tato zařízení většinou používají některou variantu GNU/Linux, který má pro výrobce výhodné licenční podmínky a může být upravován na míru konkrétním zařízením, aniž by bylo nutné znovu vynalézat kolo. [6] Zvyšuje se tedy počet zařízení, která mohou být cílem konfiguračního managamentu a zároveň se zvyšuje jejich heterogenita.

V oblasti serverů jsou nejvíce zastoupeny operační systémy GNU/Linux a Microsoft Windows. [7] GNU/Linux je nejčastěji využíván pro provoz webových aplikací, veřejných mailserverů a databází. [8] Microsoft Windows jsou používány pro provoz Microsoft Active Directory specifických aplikací jako korporátní mailservery, fileservery, apod. Dále bývají používány pro podporu produktivních aplikací pracujících na Microsoft Windows, příkladem takových serverů mohou být Microsoft SQL databáze nebo licenční servery.

Moderní nástroje pro konfigurační managament umožňují spravovat více různých platforem. Které platformy jsou preferovány, závisí na jejich primárním určení. Obvykle jsou podporovány některé z operačních systémů GNU/Linux, Microsoft Windows, Mac OS X a Unix systémy jako Oracle Solaris, IBM AIX. Běžně je podporována procesorová architektura x86 (IA-32) resp. x86-64. Otevřené nástroje lze přeložit na jiných architekturách, pokud obsahují potřebné nástroje pro překlad a jejich zdrojový kód je přenositelný. Některé nástroje tento problém řeší použitím interpretovaného jazyka.

2.2 Technologie

Nejjednodušší implementací distribuovaného konfiguračního managementu je vytvoření kompletního obrazu funkčního systému, který je pak dále rozšiřován na další počítače. Nevýhody tohoto řešení se stanou zřejmými při změně konfigurace. Změny musí být provedeny jak do referenčního obrazu systému, tak na již běžících počítačích nebo musí být počítače znovu nainstalovány.

Další možností jednoduchého konfiguračního managementu jsou skripty. Pokud je vyřešen problém distribuce na klientská zařízení, pak je toto řešení výrazně lepší než použití kompletního obrazu. Problém skriptů je, že zpravidla mohou být bezpečně spuštěny pouze jednou, což je překážkou jejich periodickému spouštění za účelem udržení konzistentního stavu konfigurace.

Nejpokročilejší možností je použití některého z nástrojů pro hromadný konfigurační management. V následující části jsou vybrané nástroje popsány.

2.2.1 CFEngine

CFEngine (ConFiguration Engine) je historicky prvním nástrojem pro konfigurační management. [9]

CFEngine je distribuované řešení, které je nezávislé na operačních systémech, síťové topologii nebo na systémových procesech. Administrátor popíše ideální stav systému vytvořením

tzv. slibů a klient provede nezbytné kroky k dosažení tohoto stavu. Automatizace je v CFEngine dosažena s pomocí komponent, které běží lokálně na klientovi.

CFEngine používá pro účely popsání požadovaného stavu systému sliby a politiky. Politika je kolekce slibů zapsaných v deklarativním jazyce specifickém pro CFEngine a uložená v textovém souboru. Klienti tyto politiky autonomně stahují proprietárním přenosovým protokolem. [10]

2.2.1.1 Historie

Projekt CFEngine začal v roce 1993 na *University of Oslo* v Norsku, jeho zakladatelem byl Mark Burgess z oddělení teoretické fyziky. Prvotním cílem projektu bylo usnadnění práce administrátora rozmanitých Unix systémů a nahradit tak velký počet různých skriptů pro různé činnosti a různé platformy.

V roce 1998 začal Mark Burgess na *Oslo University College* s výzkumem počítačové imunologie. Tento výzkum se stal teoretickým základem pro vytvoření nové verze CFEngine 2. Verze 2 byla dále rozšiřována na základě poznatků z výzkumu a zpětné vazby uživatelů.

V roce 2003 se podklady pro CFEngine sestávaly pouze z oddělených dokumentů a specifikací a neexistoval komplexní model jeho fungování. Z tohoto důvodu později vznikla *Promise Theory* – teorie slibů, která byla prezentována v roce 2004. Na tomto teoretickém základě byla vyvíjena nová verze.

V roce 2008 byl dokončen CFEngine 3. Byla založena společnost CFEngine AS, která začala vyvíjet komerční verzi jako rozšíření otevřené komunitní verze. [11]

2.2.1.2 Komerční a komunitní verze

CFEngine 3 je dostupný ve dvou variantách:

- Community Edition,
- Enterprise Edition (Nova).

Community Edition je verze vyvíjená jako open source projekt s licencí *GNU General Public License 3*. Zdrojové kódy jsou veřejně dostupné na serveru Github.com[1]. Uživatelé mají možnost hlásit chyby software na webové stránce projektu[2]. [12] Opravy chyb a úpravy zdrojových kódů jsou po schválení správci projektu přijímány do hlavní vývojové větve.

Enterprise Edition je komerční verze rozšiřující funkcionalitu komunitní verze. Základem je komunitní zdrojový kód, který je dále rozšířen o neveřejný kód implementující placené funkce. Uživatelský kód (souhrnně nazývaný politika) spustitelný v komunitní verzi je vždy spustitelný ve verzi komerční, naopak je tomu pouze v případě, není-li použita žádná placená funkce. [13]

[1] https://github.com/cfengine
[2] https://cfengine.com/dev/projects/core/issues

Komerční verze je rozšířená o:

- zpětný sběr dat od klientů,
- *Mission Portal GUI* rozhraní,
    - monitorování stavu klientů,
    - vytváření zpráv o souladu s IT politikou,
    - možnost vytváření uživatelsky definovaných zpráv,
    - podporu uživatelských účtů a rolí včetně schvalování úprav,
    - editor s podporou zvýraznění syntaxe a doplňování,
    - systém správy verzí,
    - *Design Center GUI* rozhraní,
- funkce pro orchestraci distribuovaného systému,
- nativní podporu Access Control Listů (ACL),
- možnost autentizace přes LDAP,
- kompresi a pokročilé šifrování komunikace,
- podporu dalších platforem.

V komerční verzi je dále v ceně zahrnuta technická podpora a konzultace při nasazování. [14]

Cenové podmínky komerční verze nejsou zveřejněny, pro jejich získání je potřeba vytvořit poptávku u firmy CFEngine AS.

2.2.1.3 Podporované operační systémy

CFEngine Community Edition může být přeložen a spuštěn na většině POSIX kompatibilních operačních systémů. Musí obsahovat překladač jazyka C99, nástroj GNU `make` a knihovny PAM, OpenSSL, PCRE, pthread a Tokyo Cabinet. Zdrojový kód je běžně překládán a testován na operačních systémech:

- GNU/Linux,
- Solaris,
- Windows s použitím MinGW[3].

Kód je testován a překládán na platformách:

- x86,
- x86-64,
- SPARC. [15]

Pro rychlou instalaci a aktualizaci nových verzí jsou udržovány repozitáře s instalačními balíky pro operační systémy Debian[4], Red Hat[5] a jejich klony. [16]

[3] http://www.mingw.org/
[4] http://cfengine.com/pub/apt/packages
[5] http://cfengine.com/pub/yum/

CFEngine Enterprise Edition je dostupný pouze jako přeložený instalační balík. Podporované operační systémy jsou rozděleny do tří úrovní, podle četnosti testování a úrovně automatizace procesu vydávání nových verzí.

Úroveň 1 jsou operační systémy, pro které je proces testování a vydávání nových verzí plně automatizovaný, včetně nasazení do testovacího prostředí. Tyto systémy jsou:

- Red Hat (x86-64),
- Red Hat (x86) – pouze klient,
- Solaris (x68-64, UltraSPARC) – pouze klient.

Úroveň 2 jsou operační systémy, kde je proces testování pouze částečně automatizován. Testování v testovacím prostředí probíhá pravidelně, ale nikoliv neustále jako u úrovně 1. Tyto systémy jsou:

- SUSE Enterprise Server (x86-64),
- Debian (x86-64),
- Ubuntu (x68-64),
- SUSE Enterprise Server (x86) – pouze klient,
- Debian (x86) – pouze klient,
- Ubuntu (x68) – pouze klient,
- Windows Server 2008 (x86-64, x86) – pouze klient.

Úroveň 3 jsou operační systémy, kde není proces vydávání a testování vůbec automatizován. Tyto operační systémy jsou podporovány s podmínkou, že některé funkce nemusí pracovat správně a že mohou pracovat pouze jako klient. Instalační balíky jsou vytvářeny na vyžádání. Tyto systémy jsou:

- IBM AIX,
- HP-UX,
- Open Indiana,
- SmartOS.

Je možné nechat vytvořit instalační balík pro další operační systémy, pokud splňují podmínky pro překlad Community Edition. [17]

2.2.2 Puppet

Puppet je nástroj pro konfigurační management, který historicky vychází z projektu CFEngine 2. Byl založen v roce 2005. Jeho zakladatelem je Luke Kanies, který byl významným uživatelem a přispěvatelem do kódu CFEngine 2.

Puppet je vytvořen v jazyce Ruby. Pro zápis politiky používá modelově orientovaný deklarativní jazyk, který byl navržen s důrazem na jednoduchost a snadnou interpretaci uživatelem. Pokročilé funkce je možné vytvářet v Ruby. [18]

Kód je organizován do tzv. manifestů, které jsou klienty stahovány ze serveru a interpretovány lokálně.

2.2.2.1 Komerční a komunitní verze

Puppet je dostupný ve dvou variantách:

- Open Source
- Enterprise

Open Source je komunitní verze vyvíjená s licencí *Apache 2.0*. Zdrojové kódy jsou veřejně dostupné na serveru Github.com[6]. Počet přispěvatelů je odhadován na více než 5000.

Enterprise je placená verze s uzavřeným kódem, který je založen na zdrojovém kódu komunitní verze. [19]

Komerční verze je rozšířená o:

- pokročilé GUI rozhraní,
- podporu uživatelských účtů a rolí,
- možnost autentizace přes LDAP, Active Directory nebo Google Apps Directory,
- nástroj *Event Inspector* pro vytváření zpráv,
- možnost správy virtuálních serverů VMWare. [20]

V komerční verzi je dále zahrnuta technická podpora v jedné z variant:

- *Standard*, tj. podpora e-mailem v pracovní dny od 6 do 18 hodin PST;
- *Premium*, tj. telefonická podpora 24 hodin denně, 7 dnů v týdnu se zvýšenou prioritou požadavků. [21]

[6] https://github.com/puppetlabs/puppet

2.2.2.2 Podporované operační systémy

Puppet Open Source podporuje operační systémy Linux:

- Red Hat Enterprise a kompatibilní tj.
  - CentOS,
  - Scientific Linux,
  - Oracle Linux,
  - Ascendos;
- Debian a kompatibilní tj.
  - Ubuntu;
- Fedora;
- SUSE Enterprise Server;
- Gentoo;
- Mandriva Corporate Server;
- ArchLinux.

Dále podporuje operační systémy BSD a Unix a to:

- FreeBSD,
- OpenBSD,
- Mac OS X,
- Oracle Solaris,
- IBM AIX,
- HP-UX.

Také podporuje operační systémy Windows:

- Windows Server;
- Windows Vista, 7, 8. [22]

Puppet Enterprise podporuje pouze podmnožinu operačních systémů podporovaných verzí Open Source. Všechny funkce jsou podporovány na operačních systémech:

- Red Hat Enterprise,
- CentOS,
- Ubuntu,
- Debian,
- Oracle Linux,
- Scientific Linux,
- SUSE Enterprise Server.

Pouze klientská část *Enterprise* aplikace je podporována na operačních systémech:

- Windows Vista, 7, 8, Server;
- Oracle Solaris;
- IBM AIX. [23]

Obě verze, komerční i komunitní, potřebují pro své spuštění interpret jazyka Ruby. Open Source verze vyžaduje separátní instalaci interpretu uživatelem v požadované verzi. Enterprise verze obsahuje interpret jako součást instalace. [22]

2.2.3 Chef

Chef je automatizační framework, který historicky vychází z nástroje Puppet a který slouží ke zjednodušení nasazování aplikací na fyzické a virtuální servery a do cloudů. Klient používá abstraktní definice zvané *cookbooks* a *recipes*, které jsou vytvořené v jazyce Ruby a jsou vytvářeny jako zdrojový kód. Každá definice popisuje, jak by měla být specifická část infrastruktury sestavena a spravována. Klient potom tyto definice aplikuje na servery a aplikace, výsledkem je plně automatizovaná infrastruktura. Pokud je připojen nový klient, musí být pouze definováno, jaké cookbooks a recipes na něj mají být aplikovány. [24]

Distribuce cookbooks a recipes je prováděna přes HTTPS, klient je stahuje v pravidelných intervalech ze serveru a interpretuje.

2.2.3.1 Komerční a komunitní verze

Chef je dostupný ve dvou variantách:

- Open source Chef,
- Enterprise Chef.

Komerční verze Enterprise Chef je postavena na zdrojovém kódu komunitní verze Open source Chef.

Komerční verze přináší proti komunitní verzi:

- možnost provozu jako hostovaná služba,
- rozšířenou správcovskou konzoli,
- centralizované vytváření reportů,
- podporu správy více organizací jedním serverem,
- uživatelská oprávnění na bázi rolí,
- jednotnou autentizaci přes LDAP nebo Active Directory,
- rozšířenou technickou podporu.

Rozšířená technická podpora může být objednána ve variantách Standard nebo Premium, tj. 24 hodin denně, 7 dní v týdnu. [25]

2.2.3.2 Podporované operační systémy

Chef podporuje pro instalaci serveru operační systémy:

- Red Hat Enterprise,
- CentOS,
- Ubuntu.

Pro instalaci klientů Chef podporuje:

- Debian,
- Ubuntu,
- Red Hat Enterprise,
- openSUSE,
- SUSE Enterprise,
- Solaris,
- Mac OS X,
- Windows Server.

Podmínkou pro běh klienta je nainstalované prostředí jazyka Ruby. [26]

2.2.4 Ansible

Ansible je IT automatizační nástroj, určený pro konfigurování systémů, nasazování software a orchestraci složitých úkonů, jako je kontinuální nasazování software nebo aktualizace bez výpadku. Cílem Ansible je maximální jednoduchost a snadnost použití. [27]

Ansible staví na následujících principech:

- Běží bez agentů, konfigurace se provádí pomocí masově paralelizovaného SSH.
- Pro aktualizaci konfigurace se používá push, nikoliv pull.
- Konfigurace je snadno čitelná, tvoří ji YAML soubory.
- Psaní nových modulů je snadné.

Jedná se o minimalistické řešení, na spravovaných počítačích nemusí běžet žádná dodatečná služba, přístup umožňuje SSH, které se také stará o zabezpečení komunikace. Ansible podporuje `sudo`, není tedy nutné se připojovat jako superuživatel `root`. [28]

Soubor s konfigurací kroků se u Ansible nazývá *playbook*, krok ve scénáři *play*. [29]

2.2.4.1 Komerční a komunitní verze

Ansible je open source projekt, hlavní část konfiguračního nástroje se nazývá *Ansible Core*. Na rozdíl od jiných nástrojů pro konfigurační management nerozlišuje rozšířenou placenou verzi a verzi zdarma. Ansible Core je vždy zdarma a placené jsou pouze dodatečné nástroje.

Rozšiřující produkty jsou:

- Ansible Guru,
- Ansible Tower,
- Ansible Enterprise Tower.

Ansible Guru je placená podpora při nasazování a provozu Ansible Core. Zákazník může požádat o radu při vytváření playbooků a návrhu architektury.

Ansible Tower je webové rozhraní, které umožňuje provádět všechny úkony a dále obsahuje REST API. Ansible Enterprise Tower je shodný produkt rozšířený o telefonickou podporu a garantované SLA. [30]

2.2.4.2 Podporované operační systémy

Ansible může běžet na operačních systémech Linux/Unix, které podporují Python. Microsoft Windows nejsou podporovány. Instalaci je možné provést přeložením zdrojových kódů. Instalační balíky jsou připraveny operační systémy:

- Red Hat,
- Ubuntu,
- Debian,
- FreeBSD,
- Mac OS X.

Spravovaní klienti musí pouze podporovat SSH protokol. [31]

2.2.5 Microsoft System Center 2012 R2 Configuration Manager

Microsoft System Center 2012 R2 Configuration Manager je uzavřený komerční nástroj pro konfigurační management. Základní platformou pro aplikaci je Microsoft Windows Server 2012 R2 a volitelně *Microsoft Intune*. Klientskými zařízeními za předpokladu použití cloudové aplikace Microsoft Intune mohou být počítače a mobilní zařízení na platformách:

- Windows 8/8.1,
- Windows RT,
- Windows Phone,
- Mac OS X,
- iOS,
- Android,
- Unix/Linux. [32]

Možnost provozování klientského software na různorodých platformách vychází z předpokladu nasazení v BYOD (Bring Your Own Device) prostředí. To předpokládá použití privátních zařízení zaměstnanců pro firemní účely ve firemním prostředí. [33]

Configuration Manager na klientských zařízeních umožňuje:

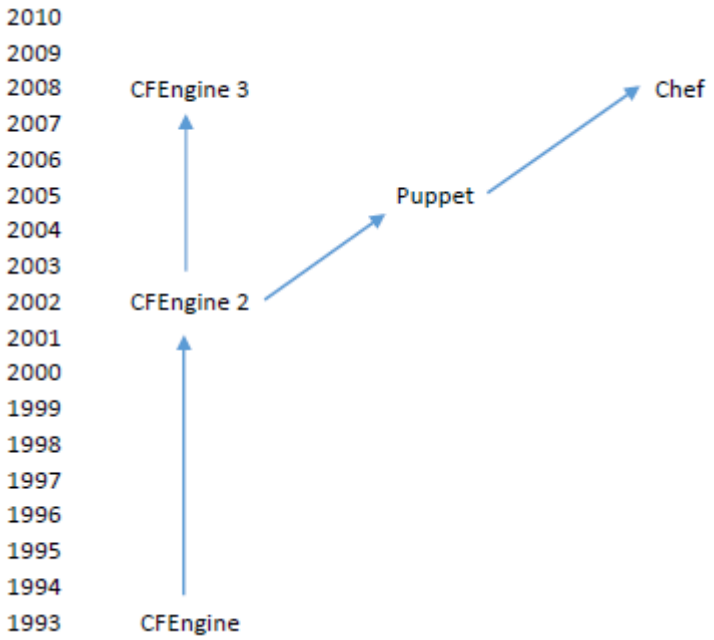
- instalaci software,
- podání zprávy o nainstalovaném software,
- nastavení sítí včetně Wi-Fi a VPN,
- detekci jailbreak[7]/rooting[8] na iOS a Android zařízeních,
- kontrolu souladu s bezpečnostní politikou organizace,
- odstraňování škodlivého software,
- automatickou instalaci aktualizací.

Správcovská konzole umožňuje rozlišování různých uživatelských rolí správců a jejich privilegií. [32]

2.2.6 Srovnání

Většina řešení pro konfigurační management je primárně určena pro některý typ operačního systému a další operační systémy, pokud jsou podporovány, jsou chápány pouze jako doplňující. Pro nástroje CFEngine, Puppet, Chef a Ansible jsou primárními operačními systémy Unix-like systémy, naopak pro Microsoft System Center je primárním operačním systémem Microsoft Windows.

Konfigurační nástroje CFEngine 3, Puppet a Chef mají všechny základ v původním nástroji CFEngine verze 1 z roku 1993, viz Obr. 2. Princip jejich fungování je proto velice podobný, každý nástroj ale má své specifické vlastnosti, které mohou být pro různá prostředí vhodné nebo nevhodné.



Obr. 2

[7] Odstranění kontroly podpisů u aplikací na zařízeních firmy Apple. [80]
[8] Získání superuživatelského přístupu v operačním systému Android. [81]

CFEngine je vytvořen jako nativně překládaná aplikace v programovacím jazyce C, Puppet a Chef jsou vytvořeny v interpretovaném jazyce Ruby. [18] Pro přeložení CFEngine stačí pouze GNU nástroje. Pokud jsou instalační soubory už přeloženy v balíčku, pak pro CFEngine nenastávají žádné další závislosti. Puppet a Chef potřebují pro svůj běh interpret jazyka Ruby a další knihovny. [34] Ansible je vytvořen v jazyce Python a pro svůj běh vyžaduje jeho interpret. [35] Nevýhodou konfiguračních nástrojů založených na interpretovaných jazycích je nutnost instalace dodatečného software interpretu na všechny klienty. Interpret i jeho knihovny musí být v rozsahu podporovaných verzí na všech klientech, přičemž udržování takového systému a jeho aktualizací může být komplikované. [36] Rozdíly v efektivitě interpretovaných a překládaných jazyků jsou obecně známy.

Konfigurace je v nástrojích CFEngine, Puppet a Chef zapisována programovacím jazykem. V CFEngine je používán deklarativní DSL (Domain Specific Language) jazyk postavený na základě teorie slibů, rozšiřující moduly mohou být vytvořeny v libovolném jazyce, pokud je dodržen stanovený formát výstupu. Puppet používá DSL jazyk na základě Ruby, rozšiřující moduly mohou být vytvořeny v Ruby přímo. Chef nepoužívá DSL, konfigurace je prováděna přímo v Ruby, proto nepodporuje moduly. [37] Chef záměrně na rozdíl od CFEngine a Puppetu zpracovává jednotlivé části konfigurace striktně sekvenčně. [18] Ansible používá konfiguraci zapsanou ve formátu YAML, v porovnání s předchozími nepoužívá programovací jazyk. Moduly pro Ansible mohou být vytvořeny v libovolném jazyce, výstupní formát modulu musí být ve formátu JSON. Na rozdíl od předchozích umožňuje kromě konfigurace ze souboru provádět i interaktivní zásahy v CLI (Command Line Interface). [37]

Soubory konfigurace jsou v CFEngine na klienty přenášeny jako prostý text v původním formátu vlastním protokolem specifickým pro CFEngine. Server je dimenzován i pro větší instalace. Autentizace se provádí pomocí veřejného a soukromého klíče. [38] Puppet používá vlastní protokol na bázi HTTPS, soubory konfigurace jsou předkompilovány na serveru a poté přeneseny na klienta. Pro větší instalace se doporučuje použít Apache s modulem Passenger místo výchozího WEBrick webserveru. [39] Chef používá pro přenos Ruby kódu protokol HTTPS bez úprav. [40] Ansible nepoužívá žádnou klientskou aplikaci, změny jsou prováděny přímo serverem přes protokol SSH. Protokol SSH je velmi bezpečný, ale pro potřeby hromadné automatizované konfigurace je výpočetně náročný. [41]

2.3 Volba vhodného řešení

V prostředí CIV ZČU již existuje řešení pro hromadný konfigurační management koncových stanic pracujících na různých verzích Microsoft Windows. Toto řešení bylo vyvinuto na CIVu na míru pouze pro potřeby ZČU. Pro hromadnou správu serverů, kterých většina pracuje na Unix-like operačních systémech, zatím žádné řešení neexistuje. Cílem bylo vybrat vhodné existující řešení na základě přechozí analýzy, které by odpovídalo potřebám ZČU.

Microsoft System Center 2012 R2 Configuration Manager je primárně určen pro zařízení pracující na operačním systému Microsoft Windows. Je zaměřen především na správu koncových počítačů a mobilních zařízení. Pro správu Unix-like serverů není příliš vhodný.

Primárně pro účely správy Unix-like serverů jsou určeny nástroje CFEngine, Puppet, Chef a Ansible. První tři jmenované sdílejí výhodu distribuovaného zpracování na klientech. Ansible má výhodu nulové stopy na spravovaných zařízeních, bohužel ale používá nepříliš efektivní SSH protokol, zatěžuje server centralizovaným zpracováním a je vysoce závislý na kvalitě síťového připojení. Spojení je navazováno směrem ke klientovi, tím jsou omezeni klienti za NAT (Network Address Translation) routery. Proto není pro zamýšlené použití vhodný.

Zbývající trojice si je funkčně velice podobná. Hlavním cílem nástroje Chef je především nasazování webových aplikací. [18] Striktně sekvenční zpracování konfigurace a použití všeobecného jazyka Ruby je pro toto použití výhodné, bohužel pro potřeby všeobecného konfiguračního managementu jsou Puppet a CFEngine navrženy lépe.

Puppet a CFEngine jsou přímou konkurencí. Zatímco Puppet vznikl po odtržení od CFEngine jako ryze komerční projekt na bázi otevřeného kódu, CFEngine přišel s komerční variantou více jak 3 roky později. [18] Z toho důvodu jsou placené funkce výrazně pokročilejší v Puppetu než v CFEngine.

Placené varianty CFEngine, Puppet a Chef jsou účtovány podle počtu nainstalovaných klientů. Licence je vždy časově omezená na jeden rok. Cena za uzel se odvíjí od počtu zakoupených licencí a od délky smlouvy. V Tab. 1 jsou ceny za licence srovnány. V případě CFEngine byl navíc kontaktován obchodní zástupce společnosti, který ZČU nabídl slevu pro akademické organizace. Tato cena je v tabulce také uvedena.

Servery	Počet	CFEngine 1 rok	CFEngine 3 roky	CFEngine Education 1 rok	CFEngine Education 3 roky	Puppet 1 rok	Chef 1 rok
GNU/Linux Debian	250	482 500 Kč	335 000 Kč	241 250 Kč	167 500 Kč	502 500 Kč	323 000 Kč
Windows Server	25	48 250 Kč	33 500 Kč	24 125 Kč	16 750 Kč	50 250 Kč	32 300 Kč
Oracle Solaris	11	21 230 Kč	14 740 Kč	10 615 Kč	7 370 Kč	22 110 Kč	14 212 Kč
Celkem	**286**	**551 980 Kč**	**383 240 Kč**	**275 990 Kč**	**191 620 Kč**	**574 860 Kč**	**369 512 Kč**
1 licence	*1*	*1 930 Kč*	*1 340 Kč*	*965 Kč*	*670 Kč*	*2 010 Kč*	*1 292 Kč*

Tab. 1

CFEngine má oproti Puppetu následující výhody:

- Deklarativní programovací jazyk má silný teoretický základ teorie slibů a umožňuje jednotně popsat požadovaný stav systému.
- Algoritmus distribuce konfigurace je bezpečný i pro nespolehlivé sítě a nezatěžuje centrální server.
- Instalace CFEngine je minimalistická a jeho provoz není závislý na dalším software.

Pro instalace s velkým počtem klientů je CFEngine výhodný, protože používá vlastní efektivní protokol pro přenos dat, který je provozován na vlastním nativním serveru. Puppet pro velké instalace doporučuje použití Apache s modulem Passenger. Puppet s rostoucím počtem klientů stále více vytěžuje server, protože provádí předkompilování konfigurace pro jednotlivé klienty. Řešením tohoto problému může být použití Puppetu v masterless režimu, kdy jsou politiky na klienty distribuovány jinou cestou, např. git nebo rsync. Předkompilování se pak provádí až na klientovi. Testy provedené s 50 až 300 klienty ukazují, že CFEngine pro stejnou činnost spotřebovává méně výpočetních prostředků a vytížení serveru roste velice pomalu. [42]

Na základě zjištěných faktů byl pro otestování a nasazení na CIV ZČU zvolen CFEngine. Za hlavní výhody je považována minimalistická instalace, jednotný deklarativní jazyk na solidním akademickém základu a efektivní přenosový protokol.

2.4 CFEngine

CFEngine byl navržen, aby umožnil škálovatelný konfigurační management v celém životním cyklu systému v jakémkoliv prostředí. Většina ostatních konfiguračních systémů předpokládá, že vždy bude k dispozici spolehlivé síťové připojení a že změny budou přesouvány shora-dolů z autoritativního uzlu. Takové systémy nejsou použitelné v prostředích, kde je síťové připojení nespolehlivé, s malou propustností nebo v systémech, které mají malý výpočetní výkon. CFEngine nepotřebuje spolehlivou infrastrukturu, pracuje oportunisticky a spotřebovává velice málo prostředků. CFEngine tedy může pracovat nejen v klasických scénářích, ale i ve zcela neobvyklém nespolehlivém prostředí. [2]

2.4.1 Teorie slibů

Teorie slibů (promise theory) je model dobrovolné spolupráce mezi individuálními autonomními činiteli, kteří zveřejňují své záměry jeden druhému ve formě slibů. Slib je deklarování záměru, jehož účelem je zvýšit příjemcovu jistotu ve tvrzení o minulém, přítomném nebo budoucím chování. [43] Aby slib mohl zvýšit příjemcovu jistotu, příjemce musí původci slibu důvěřovat. Důvěra může být vytvořena na ověření, že některé předchozí sliby jsou splněny. Důvěra je tudíž v symbiotickém vztahu se sliby. [44]

Teorie slibů se odklání od logiky závazků a povinností. Její myšlenkou je, že všichni činitelé v systému by měli mít autonomii řízení, tzn., že nemohou být nuceni do konkrétního chování. V teorii slibů může činitel vytvářet sliby jen o svém vlastním chování. Pro autonomní činitele je nesmyslné vytvářet sliby o chování někoho jiného.

Přes svůj obecný charakter je teorie slibů hlavním modelem chování CFEngine. [45]

2.4.2 Princip funkce

Systém CFEngine lze přirovnat k orchestru. Sestává se z více počítačů (hudebníků), kde každý má svoji vlastní kopii not a ví jak je zahrát. Může, ale nemusí mít dirigenta, který pomáhá koordinovat samostatné jedince.

Komponenty CFEngine samostatně běží na každém počítači. Pokud potřebují, mohou spolu komunikovat. Pokud dojde k přerušení síťového připojení, CFEngine komunikaci přeskočí a pokračuje ve své činnosti.

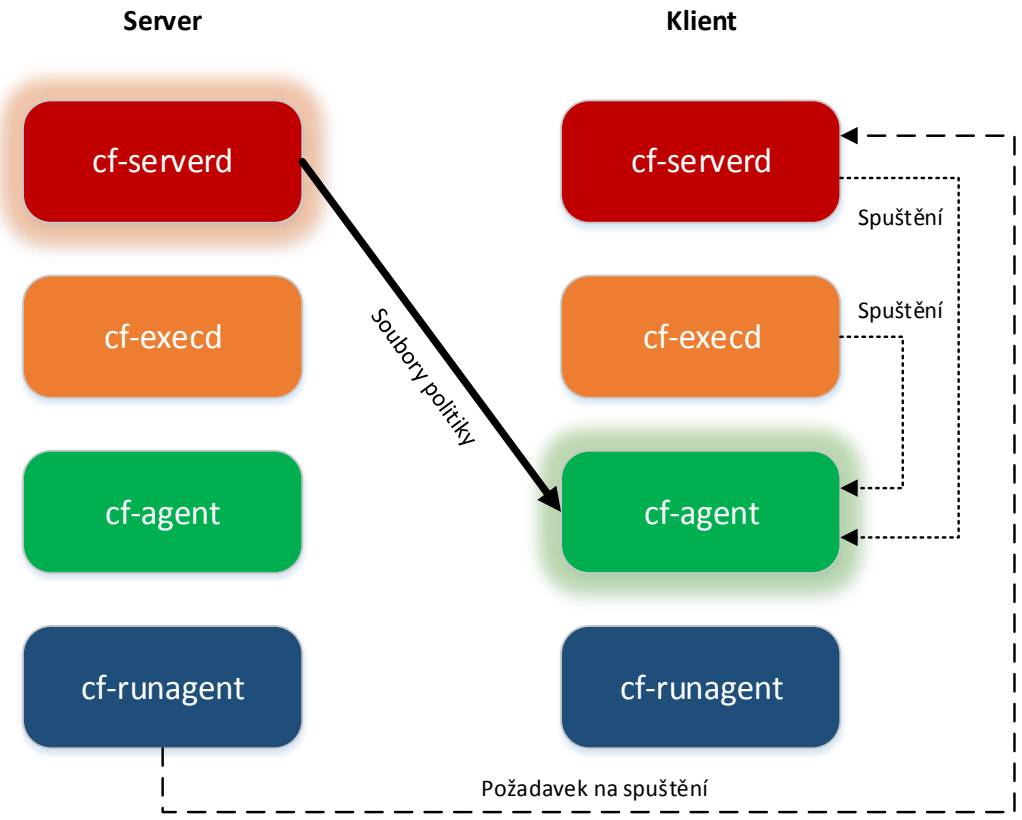
Z pohledu CFEngine není technický rozdíl mezi klientem a serverem. Všechny instalace CFEngine obsahují shodné komponenty a mohou zastávat všechny činnosti[9]. Volba, který z uzlů bude server, zůstává na administrátorovi. Server ale stále zůstává klientem, který používá sám sebe jako svůj server. Ostatní klienti mohou, pokud je to politikou vyžádáno, zastávat funkci serveru.

Každá instalace CFEngine obsahuje následující komponenty:

- `cf-serverd`, démon pracující jako fileserver;
- `cf-execd`, démon spouštějící v pravidelných intervalech `cf-agent`;
- `cf-agent`, agent vykonávající politiku;
- `cf-runagent`, pomocný program na ruční spuštění `cf-agent` na jiném počítači.

[9] Neplatí v plném rozsahu pro Enterprise Edition.

Schéma komunikace mezi komponentami je znázorněno na Obr. 3. [46]



Obr. 3

Administrátor obvykle uloží politiku na jeden z počítačů a tím z něj vytvoří server. Ostatní počítače jsou nastavené jako klienti a znají jeho IP adresu. Na serveru běží démon `cf-serverd`, na klientovi pak démon `cf-execd`. Démon `cf-execd` spouští v pravidelných intervalech (výchozí 5 min.) nástroj `cf-agent`, který se připojí k `cf-serverd` a ze serveru stáhne soubory s politikou. Potom politiku interpretuje. Pokud soubory s politikou nebyly na serveru změněny nebo se nelze připojit, pak `cf-agent` soubory nestahuje a pouze interpretuje již dříve stažené.

Z toho vyplývají dvě důležité vlastnosti CFEngine:

- Politika je interpretována lokálně bez použití serveru. Server slouží pouze jako úložiště pro soubory politiky. Politika může být na klienta zkopírována jakkoliv jinak a bude fungovat, dokonce i když bude odpojený od sítě. Politika může na klientovi být i vytvořena.
- Klient politiku stahuje dobrovolně metodou pull, nikoliv push, což se považuje za bezpečnější řešení.

Program `cf-agent` lze vzdáleně spustit nástrojem `cf-runagent`. Ten kontaktuje `cf-serverd` na klientovi, který spustí `cf-agent` mimo nastavený interval. Tak lze simulovat přenos push.

Nastavení serveru je zapsáno jako kód politiky, stejně tak jako proces stahování souborů na klientovi. To znamená, že schéma komunikace může být administrátorem kdykoliv upraveno, pokud to situace vyžaduje. Role počítače v systému je tak zcela ovlivnitelná politikou.

2.4.3 Souborová struktura

Všechny soubory CFEngine jsou v Unix-like systémech uloženy v adresáři `/var/cfengine/`. Hlavní adresář obsahuje podadresáře:

- `bin`, spustitelné binární soubory;
- `inputs`, soubory politiky pro lokální vykonání;
- `lastseen`, logy příchozích a odchozích spojení;
- `lib`, knihovny nutné pro běh CFEngine;
- `masterfiles`, soubory politiky pro distribuci na klienty;
- `modules`, soubory modulů politik;
- `outputs`, výstupy nástroje cf-agent;
- `ppkeys`, klíče pro vzájemné ověření uzlů;
- `reports`, seznamy opravených promise (pouze Enterprise);
- `share`, dokumentace a příklady;
- `state`, stavová databáze. [47]

2.4.4 Aktualizace a vykonání politiky

CFEngine pracuje na principu dobrovolné spolupráce. Jednou z vlastností, která takové chování umožňuje, je oddělení aktualizace politiky klienta od jejího vykonání. Ve skutečnosti teorie slibů ani žádný pojem aktualizace politiky nezná.

Uzel CFEngine vůbec nemusí být uzlem distribuovaného systému. CFEngine bude stejně dobře pracovat na počítači, který je nainstalován na pustém ostrově, je napájený z baterií a nikdy nebyl připojený k síti, jako v moderním datacentru. Na každém počítači, kde je nainstalován, je pravidelně spouštěn výkonný program `cf-agent`. Při každém spuštění `cf-agent` spustí politiku v adresáři `inputs`. Pokud jsou detekovány změny konfigurace systému, které nejsou v souladu s politikou, `cf-agent` je opraví a potom se ukončí. Tento proces se pravidelně opakuje každých 5 minut a je zcela nezávislý na vnějším prostředí. Proto lze politiku vytvořit přímo na lokálním počítači a nechat CFEngine dohlížet nad jejím dodržováním.

Kopírování politiky na každý počítač ručně nebo s pomocí skriptu je možné, ale značně nepraktické. Proto je před spuštěním výkonné politiky vždy vyhledán soubor `update.cf` v adresáři `inputs`, který obsahuje pokyny pro aktualizaci. Program `cf-agent` se na jejich základě spojí se serverem a stáhne z něj novou politiku. Pokud to není možné, pokračuje dále v činnosti.

Na serveru je politika pro klienty uložena v adresáři `masterfiles`. Protože je server zároveň klientem, i on si politiku z vlastního adresáře `masterfiles` kopíruje do adresáře `inputs`, odkud jí spouští. Pracuje tudíž stejně jako každý jiný klient s rozdílem, že kopírování probíhá lokálně, nikoliv po síti. [2]

2.4.5 Jazyk

CFEngine umožňuje správci definovat požadovaný stav IT systému, který je vyjádřen politikou zapsanou pomocí CFEngine jazyka. Politika (policy) je soubor slibů (promises), které deklarují požadované záměry, kterých je potřeba dosáhnout, aby byly naplněny obchodní cíle podniku. Politika je automaticky stahována klienty, kteří danou politiku lokálně vykonávají a autonomně zajišťují soulad s požadovaným stavem. [48]

2.4.5.1 Koncepce

CFEngine pracuje na principu teorie slibů (promise theory). Slib je zdokumentování záměru něco vykonat nebo se chovat daným způsobem. Cokoliv v CFEngine může být chápáno jako slib. Příkladem slibu může být webserver, který slibuje, že na něm bude nainstalována a poběží služba *httpd* na TCP portu 80.

Naplnění těchto slibů není zajišťováno direktivně směrem shora dolů, vždy musí existovat jistá míra důvěry. Ekvivalentem takového chování může být řízení větší firmy. Větší firmu nemůže nejvyšší management přímo řídit, aniž by oprávnil nižší úrovně managementu k provádění samostatných rozhodnutí a důvěřoval jim.

V systému CFEngine se klient slibem dobrovolně zavazuje splnit pravidla, za která je odpovědný. To umožňuje vytvářet rozsáhlé systémy, protože je eliminována nespolehlivá centrální autorita.

Lidé jsou dobří ve vymýšlení řešení a v rozhodování, ale nejsou spolehliví při provádění opakovaných operací. Naproti tomu stroje nejsou příliš dobré v rozhodování, ale jsou velice spolehlivé při implementaci připravených řešení. Proto jsou v CFEngine obě tyto role odděleny. Uživatel zná požadovaný cíl a umí ho deklarovat, CFEngine umí takovou deklaraci přeložit na posloupnost kroků nutnou pro dosažení cíle. Uživatel tedy definicí slibu rozhoduje co, kde, kdy, za jakých podmínek a CFEngine řeší otázku jak. Po dosažení požadovaného stavu CFEngine dohlíží, aby tento stav zůstal zachován. Dojde-li k odchylce, CFEngine provede takové kroky, aby byl systém opět v požadovaném stavu. [49]

2.4.5.2 Promise

Vše v CFEngine je interpretováno jako slib. Sliby mohou být vztaženy k různým subjektům, jako jsou např. soubory a jejich práva, příkazy, přístupová oprávnění, atd.

Sliby mohou být různých typů, viz 2.4.5.9 Typy slibů.

Každý slib se skládá z objektu, ke kterému se slib vztahuje (promiser) a z atributů, které musí být pro tento objekt splněny. Každý atribut se skládá z typu atributu a hodnoty. [50]

Příklad slibu:

```
files:

   "/tmp/test_file"
      comment => "Soubor test_file musí existovat a mít práva 644.",
       create => "true",
        perms => m("644");
```

Slib je typu `files` a definuje, že musí existovat soubor `test_file` v adresáři `/tmp`, který bude mít práva 644. Pokud neexistuje, má být vytvořen. Komentář je součástí slibu a popisuje, co slib vyjadřuje.

2.4.5.3 Bundle

Bundle je kolekce, která umožňuje seskupení souvisejících slibů do pojmenovaného celku. Bundle si lze představovat jako subrutinu nebo jako ekvivalent procedury u procedurálních programovacích jazyků. Např. skupina slibů vztahujících se k webovému serveru může být sdružena do bundle pojmenovaného `webserver`. Bundle je definovaného typu, který rozhoduje v jakých částech běhu CFEngine bude interpretován resp. který nástroj je odpovědný za sliby zapsané v bundle. Např. bundle typu `agent` obsahuje sliby pro `cf-agent`, zatímco bundle typu `server` obsahuje sliby pro `cf-serverd`. [51]

Příklad bundle:

```
bundle agent ntp
{
   files:
      "/etc/ntp.conf"
          create => "true",
       copy_from => secure_cp("/repo/config-files/ntp.conf","daidalos.civ.zcu.cz");

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

Bundle `ntp` seskupuje dva sliby související s během *ntp* démona:

1. V sekci slibů typu `files` je slib, podle kterého musí existovat konfigurační soubor, který je totožný se souborem uloženým v repozitáři na serveru. Pokud tomu tak není, CFEngine soubor stáhne a opraví tím soubor do požadovaného stavu.
2. V sekci slibů typu `services` je slib, podle kterého musí být spuštěna služba `ntp`. Pokud tomu tak není, CFEngine službu spustí.

2.4.5.4 Body

Pro každý typ slibu existuje velký počet definovatelných atributů. Body je seskupení souvisejících atributů do znovu použitelné komponenty, která může být volána s parametry. Tím má zjednodušit a zkrátit zápis slibu. Body neumožňuje uživateli shrnout libovolné atributy, definice atributů zahrnutých v body je jasně definována. [52]

Příklad použití následuje:

```
bundle agent example
{
   files:
      "/etc/passwd"
         perms => system;
}

body perms system
{
   mode => "644";
   owners => { "root" };
   groups => { "root" };
}
```

2.4.5.5 Class

Třídy (classes) umožňují aplikovat sliby pouze na konkrétní prostředí v závislosti na kontextu. Slib může být aplikován např. pouze na systémy typu Linux, pouze ve zvolený den v týdnu, nebo pokud proměnná nabývá určité hodnoty. Třídy jsou fakta, která reprezentují stav nebo kontext systému. Nastavené třídy klasifikují prostředí běhu klienta. Mohou nabývat stavu nastavená nebo nenastavená. V CFEngine Enterprise jsou všechny nastavené třídy reportovány zpět na server, kde mohou být použity na vytváření zpráv o klientech.

Třídy se rozdělují na pevné třídy (hard classes) a uživatelské třídy (soft classes). Pevné třídy jsou nastaveny v okamžiku spuštění klienta a charakterizují lokální systém[10]. Uživatelské třídy jsou definované uživatelem.

Pevné třídy mohou být využity jako v příkladu:

```
bundle agent tvrde_tridy
{
   reports:

      linux::
         "Tento klient používá Linux!";

      solaris::
         "Tento klient používá Solaris!";

      windows::
         "Tento klient používá Windows!";
}
```

Promise typu `report` vypíše při ručním spuštění klienta do konzole zadaný řetězec. V příkladu jsou definovány tři takové sliby, každý obsahuje řetězec odpovídající konkrétnímu typu operačního systému. Slib je interpretován pouze, pokud je dříve zapsaná třída nastavena. V opačném případě je ignorován. Spuštěním kódu příkladu na různých platformách dojde

[10] Seznam tvrdých tříd lze nalézt v manuálu na stránce https://cfengine.com/docs/3.5/reference-classes.html.

k vypsání vždy jen správného řetězce pro konkrétní platformu. Z příkladu vyplývá, že třídy jsou v CFEngine hlavním nástrojem pro větvení běhu programu.

Obdobně je možné kód větvit s použitím uživatelských tříd, které musí být v kódu předem definovány[11]. Příklad použití následuje:

```
bundle agent mekke_tridy
{
   classes:

      "monday_linux"  and => {"linux", "Monday"};
      "apache" expression => fileexists("/etc/apache2");

   reports:

      monday_linux::
        "Je pondělí a tento klient používá Linux!";

      apache::
        "Tento klient má konfiguraci Apache!";
}
```

Definice uživatelských tříd se většinou provádí v sekci slibů typu `classes`, ale není to podmínkou. V příkladu jsou nastaveny dvě uživatelské třídy: `monday_linux` a `apache`. Třída `monday_linux` vznikne logickou operací *and* nad třídami `linux` a `Monday`. Pokud obě existují, tzn., pokud je současně pondělí a lokální operační systém je Linux, pak je třída `monday_linux` nastavena. Druhá uživatelská třída `apache` je nastavena pokud funkce `fileexists()` nalezne adresář `/etc/apache2`.

Logické operace mohou být alternativně zapisovány přímo v místě použití třídy:

```
reports:

   linux.Monday::
      "Logická operace AND.";

   windows|solaris::
      "Logická operace OR.";

   !linux::
      "Logická operace NOT.";
```

2.4.5.6 Proměnné

Proměnné jsou v jazyce CFEngine definovány slibem podobně jako třídy. Mohou být definovány v bundle libovolného typu a rozlišují se tři základní druhy:

- skalární proměnná,
- seznam (kolekce skalárních proměnných),
- asociativní pole.

[11] CFEngine nezpracovává kód sekvenčně viz 2.4.5.8 Pořadí spuštění

Skalární proměnná může být typu:

- `string`,
- `int`,
- `real`.

Seznam může podle typu obsahu být:

- `slist` (string),
- `ilist` (int),
- `rlist` (real).

Skalární proměnná obsahuje jednu hodnotu, různé datové typy se definují jako v příkladu:

```
bundle agent promenne
{
   vars:

      "retezec"     string => "Toto je řetězec.";
      "celociselna" int    => "999";
      "realna"      real   => "1234.56";
}
```

Proměnné jsou ve všech typech bundle lokální, jedinou výjimkou je bundle typu `common`. Všechny v ní nastavené proměnné jsou globální. Lokální nebo globální proměnná se referencuje následovně:

```
$(promenna)
```

Obdobně je možné přistupovat k lokálním proměnným jiného bundle:

```
$(nazev_bundle.promenna)
```

Seznamy obsahují více hodnot daného typu, např.:

```
bundle agent seznamy
{
   vars:

      "retezce"     slist => {"První", "Druhá", "Třetí"};
      "celociselne" ilist => {"1", "2", "3"};
      "realne"      rlist => {"1.23", "4.56", "7.89"};
}
```

K seznamu jako celku je možné přistupovat následovně:

```
@(seznam)
```

Předchozího zápisu lze využít k vnořování seznamů:

```
bundle agent vlozeny_seznam
{
   vars:

      "sladina" slist => {"voda", "slad"};
      "mladina" slist => {@(sladina), "chmel"};
}
```

První definovaný seznam je vložen do druhého, přičemž dojde k jednoduchému spojení prvků prvního seznamu s prvky druhého seznamu. Není možné vytvářet vícerozměrné seznamy.

Asociativní pole je kolekce proměnných, kde klíč odkazuje na hodnotu. [53] Použití ukazuje následující příklad:

```
bundle agent asociativni_pole
{
   vars:

      "pole_uzivatelu[jgroll]"   string => "Josef Groll";
      "pole_uzivatelu[aholecek]" string => "Antonín Holeček";
      "pole_uzivatelu[flesner]"  string => "František Lešner";

   reports:

      "V městě Plzni působil $(pole_uzivatelu[jgroll])";
}
```

2.4.5.7 Iterace

V CFEngine neexistují explicitní cykly. Pro iteraci jsou namísto nich použity seznamy. Pokud je seznam referencován jako skalární proměnná, CFEngine automaticky iteruje přes všechny jeho položky. Chování je podobné cyklu for-each v některých imperativních programovacích jazycích. Příklad demonstruje základní způsob použití:

```
bundle agent iterace
{
   vars:
      "maso" slist => {"vepřové", "kuřecí", "hovězí", "jehněčí"};

   reports:
      "Máma mele $(maso) maso.";
}
```

Po spuštění kódu příkladu `cf-agent` vypíše:

```
Máma mele vepřové maso.
Máma mele kuřecí maso.
Máma mele hovězí maso.
Máma mele jehněčí maso.
```

Z výstupu vyplývá, že cyklus je vždy prováděn zopakováním celého slibu. Z toho vyplývají jistá omezení, popsaná dále.

První příklad lze upravit pro iteraci přes pole:

```
bundle agent iterace
{
   vars:
      "zvire" slist => {"prase", "kure", "krava", "jehne"};

      "maso[prase]" string => "vepřové";
      "maso[kure]" string => "kuřecí";
      "maso[krava]" string => "hovězí";
      "maso[jehne]" string => "jehněčí";

   reports:
      "Máma mele $(maso[$zvire]) maso.";
}
```

Výstup po spuštění kódu bude shodný s první příkladem. Explicitní vyjmenování klíčů není ve většině případů vhodné, seznam klíčů je možné získat z pole funkcí `getindices()`[12] [54]:

```
"zvire" slist => getindices("maso");
```

Koncept implicitní iterace je intuitivní, má ale svá omezení. Kód následujícího příkladu ukazuje možný problém:

```
bundle agent problem
{
   vars:
      "pivo" slist => {"budvar", "prazdroj", "birell"};
      "napojovy_listek" => {"mineralka", "kofola", "pivo_$(pivo)"};
}
```

Na první pohled se může zdát, že výsledné pole by odpovídalo zápisu:

```
"napojovy_listek" => {"mineralka", "kofola", "pivo_budvar", "pivo_prazdroj",
                      "pivo_birell"};
```

[12] K funkci `getindices()` existuje ekvivalentní funkce `getvalues()` pro získání seznamu hodnot.

Ve skutečnosti ale spuštění skončí chybou:

```
Redefinition of variable "napojovy_listek"
```

Protože je vždy zopakován celý slib, nikoliv jeho část, iterace bude při spuštění interpretována jako:

```
"napojovy_listek" => {"mineralka", "kofola", "pivo_budvar"};
"napojovy_listek" => {"mineralka", "kofola", "pivo_prazdroj"};
"napojovy_listek" => {"mineralka", "kofola", "pivo_birell"};
```

Takový kód se opakovaně pokouší inicializovat proměnnou se stejným názvem a to je příčinou chybového hlášení. [55]

2.4.5.8 Pořadí spuštění

Deklarativní programovací jazyk CFEngine nepoužívá pevné pořadí vykonávání řádků kódu za sebou jako například skripty v jazyce Bash apod. Namísto toho používá pragmatický způsob spouštění tak, aby při interpretaci slibu byly splněny všechny jeho závislosti, např. aby byly nastaveny proměnné a třídy.

Sliby jsou interpretovány ve výchozím pořadí podle jejich typu. Tento přístup je založen na jednoduché úvaze, že některé typy slibů jsou zpravidla závislé na některém jiném typu slibů. Např. třídy často vznikají na základě proměnných. V rámci bundle je výchozí pořadí zpracování slibů podle typu následující:

1. `meta`
2. `vars`
3. `defaults`
4. `classes`
5. `files`
6. `packages`
7. `guest_environments`
8. `methods`
9. `processes`
10. `services`
11. `commands`
12. `storage`
13. `databases`
14. `reports`

Speciálním případem jsou bundle typu `edit_line`, které slouží k úpravě obsahu souborů. Výchozí pořadí typů slibů v bundle typu `edit_line` je:

1. `meta`
2. `vars`
3. `defaults`
4. `classes`
5. `delete_lines`
6. `field_edits`
7. `insert_lines`
8. `replace_patterns`
9. `reports`

U souborů zpravidla záleží na pořadí vkládaných nebo mazaných řádků. Z toho důvodu je dodržováno pořadí interpretace jednotlivých slibů v rámci jednoho typu slibu. Naproti tomu u běžných bundle není pořadí interpretace definováno a může být prováděno libovolně. Výjimkou je, pokud je odhalena závislost mezi sliby v rámci jednoho typu. V takovém případě je pořadí zpracování upraveno tak, aby byla závislost splněna. Pokud nastane situace, že v jednom běhu kódu nemohou být splněny všechny závislosti, je kód interpretován znovu. Opakování běhu může být provedeno maximálně třikrát.

Aby bylo dosaženo co nejvyšší efektivity při zpracování kódu, je test závislostí proveden před interpretací kódu, ihned po ověření správnosti syntaxe. Tím se předem určí pořadí interpretace slibů uvnitř typů. Sliby jsou pak v rámci bundle interpretovány nejdříve podle typů a následně podle stanoveného pořadí. Styl zápisu může ovlivnit rychlost a efektivitu běhu kódu. Např. není vhodné nastavovat hodnotu proměnné na základě třídy, protože proměnné jsou interpretovány dříve než třídy. To vede k opakovanému interpretování kódu.

Jedinou výjimkou, kdy je interpretace kódu striktně sekvenční je parametr `bundlesequence` v body `common control`. Body `common control` může být přirovnáno k funkci `main()` v procedurálních programovacích jazycích. Parametr `bundlesequence` je seznam názvů bundle, které mají být interpretovány v pořadí zápisu. Tak mohou být např. knihovny interpretovány před uživatelským kódem. [56]

2.4.5.9 Typy slibů

Všechno v CFEngine je slib. CFEngine jich rozeznává několik typů, podle typu subjektu, ke kterému se slib vztahuje. Jednotný přístup zajišťuje přehlednost a jednotnou logiku politiky.

Ve zdrojovém kódu název typu slibu zapsaný s dvojtečkou, např. `files`, uvozuje sekci v rámci bundle. Sekce je chápána od uvození do konce bundle nebo do uvození dalšího typu. Všechny sliby v rámci této sekce jsou interpretovány jako uvozený typ.

Typ slibu `vars` již byl popsán v 2.4.5.6 Proměnné, typ `classes` v 2.4.5.5 Class. Následující text popisuje zbývající typy. Kompletní popis všech přípustných atributů k jednotlivým typům slibů lze nalézt v referenčním manuálu[13].

2.4.5.10 Commands

Sliby typu `commands` slouží ke spuštění příkazu v CLI operačního systému. Pokud není atributem určeno jinak, příkaz je spouštěn přímo z CFEngine a nepodporuje rozšíření shellu operačního systému jako např. roury, přesměrování apod. Příkaz slibuje, že bude spuštěn, ale slib nemá dále žádný vztah ke spuštěnému nebo již běžícímu procesu. K tomu účelu slouží promise typu `processes`. Výstup příkazu není nijak zpracováván ani jinak dostupný k použití, k tomuto účelu lze použít funkci `execresult()`. Jedinou zpětnou vazbou je volitelné nastavení třídy na základě návratové hodnoty příkazu.

V nejjednodušším případě lze slib použít bez atributů:

```
bundle agent update_apt
{
   commands:
      "/usr/bin/apt-get update";
}
```

Speciálním případem příkazu je tzv. modul, který výrazným způsobem rozšiřuje možnosti CFEngine. Příkaz je považován za CFEngine modul, pokud je nastaven atribut `module`. Modul je skript nebo program vytvořený v libovolném programovacím jazyce, který vypisuje výstup v definovaném formátu:

- Řádky začínající znakem `+` přidávají třídu (např. `+nazev_tridy`).
- Řádky začínající znakem `-` odebírají třídu (např. `-nazev_tridy`).
- Řádky začínající znakem `=` nastavují skalární proměnnou.
- Řádky začínající znakem `@` nastavují seznam. [57]

Moduly lze využít jako zásuvné moduly CFEngine k provádění složitějších operací, např. v případech, kde je praktičtější využití imperativního programovacího jazyka.

2.4.5.11 Processes

Sliby typu `processes` se vztahují k položkám v tabulce procesů OS. Promiser v tomto případě vyjadřuje regulární výraz hledaný v tabulce procesů tj. spuštěných příkazů v určitém stádiu běhu. Logicky pak promiser vyjadřuje proces určitého názvu. CFEngine rozlišuje mezi `processes` a `commands`. Sliby typu `commands` jsou sliby vztažené ke spustitelnému souboru, tzn., že soubor:

- může být spuštěn,
- nemůže mu být poslán signál,
- nemá PID (Process ID).

[13] https://cfengine.com/docs/3.5/reference-promise-types.html

Naopak sliby typu `processes` se vztahují k dané instanci procesu v systému. Procesy mají zcela opačné vlastnosti, protože proces:

- nemůže být spuštěn,
- může mu být poslán signál,
- má PID.

Z toho vyplývá, že restartování procesu musí být zapsáno jako samostatný slib typu `commands`. [58]

Slib typu `processes` v kombinaci se slibem `commands` lze využít k zajištění běhu procesu:

```
bundle agent apache2_enabled
{
   processes:
      "apache2"
         restart_class => "restart_apache";

   commands:
      restart_apache::
         "/etc/init.d/apache2 start";
}
```

Dojde-li z libovolného důvodu k ukončení procesu, pak CFEngine nastaví třídu `restart_apache`, která následně povolí spuštění příkazu pro opětovné spuštění Apache.

Obdobně lze slib typu `processes` využít k zajištění stavu, kdy požadovaný proces neběží:

```
bundle agent apache2_disabled
{
   processes:
      "apache2"
         process_stop => "/etc/init.d/apache2 stop",
         signals => {"term", "kill"};
}
```

Na příkladu je vidět nejednotnost postupu pro spuštění a ukončení procesu. Spuštění je prováděno logicky správně odděleně, podle principu popsaného v úvodu. V kontrastu s tím je ukončení procesu prováděno přímo ve slibu `processes`.

2.4.5.12 Services

V předchozích odstavcích bylo popsáno použití slibů typu `commands` a `processes`. Sliby typu `services` jsou kombinací předchozích a jsou vyjádřením abstraktního pohledu na provoz služby. Službou je v tomto případě míněna množina nula a více procesů. Prázdná množina je stav, kdy služba neběží. Služba vždy běží na pozadí a nevyžaduje uživatelský vstup. [59]

Následující příklad zajišťuje obdobný výsledek jako v příkladu v 2.4.5.11 Processes:

```
bundle agent apache2_service
{
   services:
      "www"
         service_policy => "start";
}
```

Takto jednoduchý zápis je možný, pokud je služba předdefinována ve standardní knihovně CFEngine. Ta v současné době obsahuje 93 služeb, přičemž tento počet se neustále rozšiřuje díky příspěvkům uživatelské komunity. Pokud služba v knihovně neexistuje, je nutné si vytvořit vlastní body typu `service_method` a atributem se na něj ve slibu odkázat:

```
bundle agent custom_service
{
   services:
      "custom123"
         service_policy => "start",
         service_method => service_custom;
}

body service_method service_custom
{
   service_bundle => my_services("$(this.promiser)","$(this.service_policy)");
}

bundle agent my_services
{
   vars:
      "startcommand[custom123]" string => "/etc/init.d/cutom123 start";
      "restartcommand[custom123]" string => "/etc/init.d/custom123 restart";
      "reloadcommand[custom123]" string => "/etc/init.d/custom123 reload";
      "stopcommand[custom123]" string => "/etc/init.d/custom123 stop";
      "pattern[custom123]" string => ".*custom123.*";

   processes:
   .
   .
   .

   commands:
   .
   .
   .
}
```

V knihovně nebo vlastním bundle je slib typu `services` vždy interpretován jako dvojice `commands` a `processes`. V příkladu je zhruba nastíněno, jak se definuje nová vlastní služba. Formát odpovídá formátu zápisu ve standardní knihovně. Ten je dobré dodržovat, pokud by výsledná služba měla být někdy v budoucnu přidána do standardní knihovny. Kompletní definici vlastní služby lze vytvořit podle souboru `processes.cf` ve standardní knihovně nebo podle příkladu v manuálu CFEngine[14].

[14] https://cfengine.com/docs/3.5/examples-policy-ensure-service-is-enabled-and-running.html

2.4.5.13 Files

Sliby typu `files` jsou určeny k manipulaci se soubory. Subjektem je vždy lokální soubor, který může být vytvořen, smazán nebo upraven. [60]

Správa konfiguračních souborů s CFEngine může být prováděna třemi způsoby:

- Kompletní soubor může být beze změny zkopírován z CFEngine serveru.
- Může být zkopírována šablona, která podporuje některé konstrukce CFEngine jazyka a která se na lokálním počítači doplní.
- V souboru mohou být prováděny řádkové operace.

Každý ze způsobů má své výhody a nevýhody. Kopírování celého souboru je nejjednodušší varianta, bohužel soubor musí být pro všechny klienty stejný nebo musí administrátor spravovat více variant. Naopak provádění řádkových úprav umožňuje naprostou volnost, ale politika je pak velmi komplikovaná. Každý soubor odpovídá více či méně složité gramatice a je tudíž obtížné se v souboru programově zorientovat. Částečným řešením toho problému je připravení speciálních řádků tzv. kotev, před nebo za které se bude řádek vkládat. Kompromisem mezi předchozími způsoby je použití šablony. Šablony jsou předem připravené soubory, které umožňují do předem připraveného statického textu v daném formátu, doplňovat proměnné, případně mohou být celé úseky textu použity jen pro konkrétní klienty. [61]

Kopírování souboru z CFEngine serveru (obecně z libovolného CFEngine uzlu) se provádí následovně:

```
bundle agent soubor_ze_serveru
{
  files:
     "/tmp/soubor"
        copy_from => remote_cp("/var/cfengine/masterfiles/soubor",
                               "$(sys.policy_hub)");
}
```

Ke zkopírování souboru se používá body `remote_cp` ze standardní knihovny, viz 2.4.5.17 Standardní knihovna. Všechny cesty jsou z důvodů bezpečnosti vždy uváděny absolutně.

Vytvoření souboru na základě šablony se provádí takto:

```
bundle agent soubor_ze_sablony
{
  files:
     "/tmp/soubor2"
        create => "true",
        edit_template => "/tmp/sablona";
}
```

Soubor se vytvoří z lokálně uložené šablony, tzn., že šablona musí být nejdříve ze serveru zkopírována na lokální disk např. postupem uvedeným výše.

Šablona může obsahovat názvy proměnných, které jsou nahrazeny jejich hodnotou:

```
listen_address = $(sys.ipv4)
```

Případně mohou být použity třídy k omezení vybraných úseků na konkrétní servery:

```
[%CFEngine linux:: %]
listen_address = $(sys.ipv4[eth0])

[%CFEngine solaris:: %]
listen_address = $(sys.ipv4[e1000g0])
```

Řádkové úpravy souborů umožňují přidávat nebo odebírat jednotlivé řádky z libovolných souborů, které mohou být ručně editovány kýmkoliv.

Úpravy se provádí s pomocí bundle typu `edit_line`:

```
bundle agent ssh_config
{
  files:
     "/etc/ssh/sshd_config"
        edit_line => disable_root;
}

bundle edit_line disable_root
{
  delete_lines:
     ".*PermitRootLogin yes.*";

  insert_lines:
     "PermitRootLogin no";
}
```

První sekce bundle `disable_root` odstraní řádek, který odpovídá zadanému PCRE regulárnímu výrazu. Druhá sekce vloží řádek v uvozovkách. Úpravy jsou prováděny vždy v zapsaném pořadí v rámci sekce, přičemž odstranění řádků je prováděno vždy před vkládáním řádků.

2.4.5.14 Packages

CFEngine umožňuje spravovat softwarové balíky použitím nativního balíčkovacího systému v každém podporovaném operačním systému. CFEngine dohlíží na kontrolu a naplnění slibů, ale kontrolu stavu, instalaci a odebrání balíků ponechává na balíčkovacím sytému.

S balíčkem se zachází jako s černou skříňkou mající vlastnosti:

- název balíčku,
- verzi balíčku,
- název architektury.

Správci balíčků jsou považováni taktéž za černou skříňku, která podporuje funkce:

- vypsání seznamu nainstalovaných balíčků,
- instalaci balíčku,
- odebrání balíčku,
- opravení instalace z balíčku,
- aktualizace instalace z balíčku,
- záplatování instalace z balíčku,
- kontrolu instalace z balíčku.

Správci balíčků se dělí na chytré a hloupé. Chytří správci balíčků např. apt jsou schopni vytvořit strom závislostí a všechny závislosti naplnit. Balíčky v nich mohou být instalovány použitím symbolických jmen bez uvedení verze nebo architektury. U hloupých balíčkovacích systémů např. *msi* je potřeba závislosti ověřit a naplnit ručně. [62]

Slib, který zajistí, že na klientovi bude nainstalován balík, může být zapsán následovně:

```
bundle agent apache2_package
{
   packages:
      "apache2"
         package_policy => "add",
         package_method => apt-get;
}
```

CFEngine opět dohlíží na naplnění slibu a pokud je balík z libovolného důvodu odstraněn, pak je na základě slibu opět nainstalován.

2.4.5.15 Methods

Sliby typu `methods` slouží pro vnoření bundle dovnitř jiného bundle. Bundle je možné vnořovat s parametry a je tak možné je použít jako znovupoužitelné celky. [63]

Příklad použití:

```
bundle agent tokoukate
{
   vars:
      "jidla" slist => {"Knedlíky se zelím, se zelím kyselým",
                        "Pak sem jed u stolu kdo ví co v rosolu",
                        "Kapustu vařenou jedli sme s Mařenou",

                        "Sám jsem si za pecí zadělal telecí"};

   methods:
      "any" use_bundle => predstavtesi("$(jidla)");
}

bundle agent predstavtesi(jidlo)
{
   vars:
      "ref" string => "Představte si, představte si, co jsem měl dnes k obědu";

   reports:
      "$(ref). $(jidlo).";
}
```

Použití metody k vnoření bundle lze přirovnat k volání funkce uvnitř jiné funkce v procedurálních jazycích. Tak lze vytvářet obecné bundle, které mohou být použity jako stavební komponenty konkrétnějších bundle. Sliby typu `methods` mají také velký vliv na možnosti strukturování politiky.

2.4.5.16 Reports

Funkce slibu typu `reports` byla již dříve v textu stručně popsána. Slib tohoto typu při interaktivním spuštění do konzole vypíše zadaný řetězec. Pokud je CFEngine spuštěn automaticky neinteraktivně a pokud je v konfiguraci nastavena e-mailová adresa administrátora, pak administrátor obdrží od klienta e-mail s textem reportu. V obou případech je výstup zapsán do souboru `syslog`. Taktéž je možné atributem stanovit výstupní soubor, kam jsou výstupy zapsány. [64]

Použití slibu tohoto typu je intuitivní:

```
bundle agent vystup
{
   reports:
      "Toto je informace pro administrátora.";
}
```

2.4.5.17 Standardní knihovna

Standardní knihovna nazývaná *Community Open Promise Body Library* je kolekcí často používaných body a bundle, kterou vytváří a spravuje komunita CFEngine. Cílem je umožnit uživateli psát politiky bez nutnosti vytvářet vlastní body a bundle pro často opakované činnosti. Některá body slouží jako znalostní báze o různých operačních systémech, balíčkovacích systémech apod. Psaní politik pro heterogenní prostředí je pak zjednodušeno na použití připraveného body. Není tudíž nutné zkoumat umístění konkrétních spustitelných souborů v konkrétním OS, vstupy a výstupy různých balíčkovacích systémů apod.

Knihovna je součástí politiky a je uložená v adresáři `lib/`. Nově od verze 3.5 je pro každou verzi CFEngine vytvořen vlastní adresář s číslem verze, čímž je zajištěna zpětná kompatibilita při aktualizaci. Dále byla knihovna rozdělena na více souborů, podle druhu body a bundle.

2.4.6 Bezpečnost

Jednou z klíčových vlastností CFEngine 3 je důraz kladený na bezpečnost. Architektura, komunikační protokoly a použitý programovací jazyk C, který nezávisí na externích knihovnách skriptovacích jazyků, dělají CFEngine velice odolným proti potenciálním útokům. [65]

CFEngine 3 nemá žádný záznam v databázi zranitelností NVD amerického národního institutu pro standardizaci a technologie NIST. Pro CFEngine 2 je v databázi evidováno 9 záznamů[15].

Aby byla bezpečnost udržována stále na vysoké úrovni, řídí se vývojáři CFEngine následujícím kodexem:

1. Už v návrhu by mělo být zaručeno, že není možné CFEngine klientovi zaslat data, která by měnila politiku. Každý klient by si měl zachovat právo kdykoliv navrhovanou politiku odmítnout. To se nazývá model dobrovolné spolupráce.
2. CFEngine by měl vždy podporovat šifrování přenášených dat.
3. Každý host by měl pokračovat ve své funkci, tak dlouho jak to bude možné, bez nutnosti komunikace s ostatními uzly.
4. CFEngine by měl používat jednoduchý model důvěryhodnosti na bázi klíčů (podobný SSH). Neměla by být použita žádná centrální autorita. SSL a TLS by nemělo být použito.
5. CFEngine by měl vždy poskytovat bezpečné výchozí hodnoty, které neposkytují přístup k ostatním uzlům.

[15] http://web.nvd.nist.gov/view/vuln/search-results?query=cfengine&search_type=all&cves=on

Splnění bodů kodexu lze velice snadno ověřit:

- Body 1., 3. a 5. jsou splněny díky principu funkce CFEngine. Každý klient stahuje politiku ze serveru dobrovolně, v dobrovolně zvoleném čase. Směr přenosu politiky je vždy pull, nikoliv push. Politika je vykonávána lokálně bez jakýchkoliv vazeb na server nebo ostatní klienty. Politika je prováděna i v okamžiku, kdy je klient odpojen od sítě.
- Bod 2. je splněn do té míry, že přenášené soubory jsou šifrovány volitelně. V případě Community Edition je používána šifra Blowfish 128, v Enterprise Edition je používán AES 256.
- Bod 4. je splněn použitím ověřováním s pomocí veřejného RSA 2048 klíče.

Dalším bezpečnostním prvkem je omezení poskytování politik ze serveru pouze na rozsah IP adres definovaný atributem `allowconnects` v nastavení serveru. [66]

2.4.7 Vliv na bezpečnost informačního systému

Použití CFEngine má zásadní vliv na zvýšení bezpečnosti informačního systému jako celku. CFEngine pracuje autonomně na pozadí a ve stanoveném intervalu (výchozí 5min) ověřuje soulad klienta s definovanou politikou. Pokud je u některého slibu zjištěn nesoulad, pak je tento opraven s možností nahlášení události administrátorovi. Taková situace může nastat:

- nevhodným zásahem administrátora,
- činností nainstalovaného software,
- manipulací útočníkem.

V krátkém intervalu je tedy prováděn kompletní a pravidelný bezpečnostní audit všech uzlů systému s okamžitou automatickou nápravou.

CFEngine může pomoci zvýšit bezpečnost informačního systému neustálou kontrolou:

- běžících procesů,
- otevřených portů,
- obsahu a oprávnění souborů,
- nastavení SSH,
- firewallu,
- verzí balíčků,
- atd.

Tento přístup je v CFEngine nazýván *Active Security*. [67]

2.4.8 Dokumentace

Dokumentace CFEngine 3.5 je strukturována do tří základních kategorií:

- základní manuál[16],
- referenční manuál[17],
- příklady[18].

Základní manuál popisuje architekturu, design, komponenty, princip fungování a koncepci jazyka. Slouží k prvotnímu seznámení se s prostředím CFEngine a k pochopení jeho fungování.

Referenční manuál do detailu popisuje jednotlivé prvky jazyka a slouží především jako programátorská příručka pro vytváření politik.

Příklady řeší ukázkové problémy od nejjednoduššího uvedení CFEngine do provozu po konkrétní praktické problémy. Cílem je procvičení použití programovacího jazyka, kde lze čerpat inspiraci pro vlastní řešení.

Nevýhodou dokumentace verze 3.5 je upřednostnění designu před přehledností a účelností. V některých případech je vhodnější použít původní dokumentaci verze 3.0[19], která je přehlednější a ucelenější. Původní dokumentace obsahuje také více příkladů a například popis standardní knihovny existuje pouze v této verzi[20].

[16] https://cfengine.com/docs/3.5/manuals.html
[17] https://cfengine.com/docs/3.5/reference.html
[18] https://cfengine.com/docs/3.5/examples.html
[19] https://cfengine.com/archive/manuals
[20] https://cfengine.com/archive/manuals/CfengineStdLibrary

3 Realizační část

Tato část popisuje testování CFEngine Community Edition a Enterprise Edition ve dvou laboratorních prostředích. První laboratorní prostředí ověřuje základní vlastnosti otevřené varianty, druhé laboratorní nasazení se zabývá otestováním rozšiřujících funkcí placené verze a nasazením CFEngine v heterogenním prostředí na různých platformách.

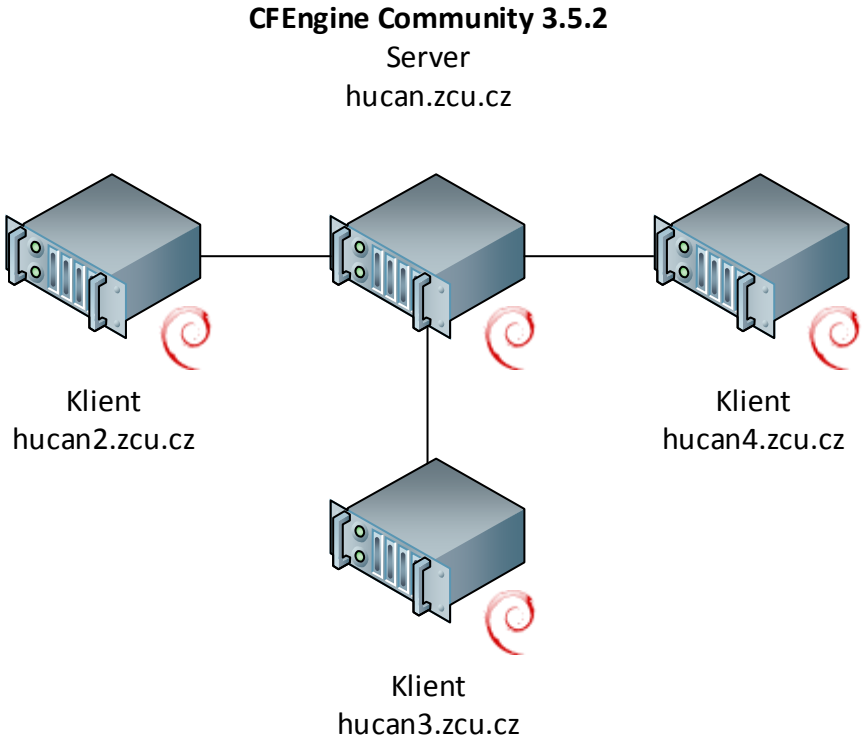
Dále tato část obsahuje porovnání vlastností CFEngine s nástrojem Puppet nasazeným v reálném produkčním systému.

Na závěr je popsáno produkční nasazení ve výpočetním prostředí ZČU.

Uvedené ukázky zdrojových kódů jsou v některých případech záměrně zjednodušeny. Jsou syntakticky správné a funkční, ale v některých případech mohla být odstraněna kontrola okrajových podmínek apod. pro zachování názornosti ukázky. V některých případech rozsáhlých zdrojových kódů je použit pouze úryvek. Kompletní zdrojové kódy jsou v příloze na CD.

3.1 Laboratorní nasazení 1 - CFEngine 3 Community Edition

Pro účely testování CFEngine 3 Community Edition byly ve virtualizačním prostředí CIV ZČU vytvořeny čtyři virtuální servery s operačním systémem Debian GNU/Linux 7.1 Wheezy.



Obr. 4

3.1.1 Instalace serveru a klientů

První instalace *CFEngine 3 Community Edition* proběhla nástrojem `apt` z balíku `cfengine3` ze standartního repozitáře operačního systému. Tato instalace měla ihned z počátku několik nevýhod:

1. Nainstalovaná verze CFEngine 3.2.4 byla zastaralá proti aktuální verzi 3.5.2. Ve verzi 3.4 a především ve verzi 3.5 byly do CFEngine implementovány některé důležité nové funkce např. podpora slibů typu `methods:` (od 3.4).
2. Byla výrazně změněna adresářová struktura. Soubory byly přesunuty z `/var/cfengine` do `/var/lib/cfengine` a adresář `/var/cfengine/inputs` byl změněn na symbolický odkaz na umístění `/etc/cfengine`.
3. Čistá instalace neobsahovala žádné předem připravené politiky, pouze prázdný adresář *masterfiles*. Kritická byla především absence souborů s nastavením serveru, souborů pro aktualizování klientů a absence knihoven.

Na základě těchto nevýhod bylo od používání balíku z repozitáře operačního systému upuštěno. Pro další testování byl využíván balík `cfengine-community` z oficiálního repozitáře[21] projektu CFEngine. Oficiální repozitář odpovídá formátu pro program `apt`, je tedy možné jeho adresu vložit do souboru `/etc/apt/sources.list`:

```
# cfengine.com
deb http://cfengine.com/pub/apt stable main
```

Instalace nebo aktualizace CFEngine na libovolném serveru s operačním systémem Debian je pak triviální:

```
apt-get install cfengine-community
```

Virtuální server `hucan.zcu.cz` byl prohlášen za CFEngine server. Každý CFEngine server je zároveň klientem, proto není třeba explicitně nastavovat roli serveru.

Spárování se serverem a první aktualizace politiky klienta se provádí spuštěním programu `cf-agent` s parametrem `--bootstrap`:

```
cf-agent --bootstrap --policy-server 147.228.52.126
```

Pokud `cf-agent` detekuje vlastní IP adresu, začne se považovat za CFEngine server. Použití IP adresy lokální smyčky (`127.0.0.1`) není doporučováno. [68]

Nastavení serveru je součástí politiky, je zapisováno ve stejném jazyce a je uloženo v adresáři `masterfiles`. V předchozích verzích bylo nutné před provedením bootstrapu provádět nastavení, od verze 3.5 je výchozí politika připravena tak, že většina nastavení je zapsána

[21] http://cfengine.com/pub/apt/packages

dynamicky a není potřeba do nich na začátku zasahovat. Nejdůležitější proměnné v nastavení byly přesunuty do souboru `def.cf`, zejména pak:

```
vars:
   "acl" slist => { "$(sys.policy_hub)/16" };
   "domain" string => "zcu.cz";
```

Proměnná `acl` je použita v `body server control`, tj. v nastavení serveru, v atributech:

- `allowconnects`, který definuje, z jakých IP je možné se k serveru připojit;
- `trustkeysfrom`, který definuje IP adresy, ze kterých jsou veřejné klíče pro ověřování považovány za důvěryhodné;
- `skipverify`, který definuje, pro které IP adresy není potřeba provádět ověření identity s použitím DNS.

Ve výchozím nastavení je proměnná *acl* nastavena použitím systémové proměnné `sys.policy_hub` na `/16` adresu sítě ve které se server nachází. Toto nastavení bylo ponecháno, protože adresa sítě ZČU je `147.228.0.0/16`[22].

Proměnná `domain` se nevztahuje pouze k serveru, ale ke všem klientům. Reprezentuje doménové jméno klientů a je používána například pro nastavení odchozí adresy pro odesílání e-mailů. Tato proměnná byla nastavena na adresu `zcu.cz`. Používání této proměnné v politikách je bezpečné pouze tehdy, jsou-li všichni klienti i server ve stejné doméně. V prostředí ZČU ale existují i servery v subdoménách např. `metalist.civ.zcu.cz`. Pro takové je proměnná nastavená špatně a jak bylo později zjištěno, vede k problémům s některými funkcemi, viz 3.6.2 Chyba odesílání e-mailů.

Servery `hucan2.zcu.cz`, `hucan3.zcu.cz` a `hucan4.zcu.cz` byly prohlášeny za CFEngine klienty a byly spárovány se serverem `hucan.zcu.cz` bootstrap příkazem shodným s dříve uvedeným příkladem.

3.1.2 Správa verzí politiky

Systém správy verzí politiky není v CFEngine standardně obsažen, ale jeho používání je doporučené. [69] Politika je zapsána v deklarativním programovacím jazyce, proto může být použit některý z běžných verzovacích systémů. V testovacím prostředí byl použit systém správy verzí GIT, protože je považován za nejpokročilejší verzovací systém v prostředí OS Linux. [70] Kromě běžných výhod verzovacích systémů, kterými jsou:

- udržování kompletní historie úprav,
- víceuživatelský přístup,
- vytváření vlastních větví konfigurace,

[22] https://apps.db.ripe.net/search/query.html?searchtext=147.228.0.0

může být využito distribuovaného principu fungování GITu pro lokální vývoj a testování politik. V principu je fungování GITu a CFEngine jako distribuovaného systému velice podobné, každý uzel má vlastní kopii dat získanou z centrálního repozitáře. Změna lokálních dat v uzlu neovlivní data v ostatních uzlech, ale v případě GITu je možné tuto změnu nahrát zpět do repozitáře, odkud může být rozšířena na ostatní uzly. Této myšlenkové shody bylo s výhodou využito při návrhu funkce verzovacího systému.

Základní požadavky na produkční nasazení verzovacího systému byly:

1. Zajistit přístupová oprávnění ke GIT repozitáři v návaznosti na autentizační mechanismy ZČU.
2. Umožnit transparentní členění CFEngine klientů do různých vývojových větví.
3. Automaticky rozšiřovat změny nahrané do GIT repozitáře na CFEngine klienty.
4. Umožnit lokální vývoj a testování politik na libovolném počítači v síti ZČU.

V testovacím prostředí byl na serveru `hucan.zcu.cz` zřízen lokální testovací repozitář[23]:

```
mkdir /var/cfengine/repository
git init --bare
```

Dále byly stanoveny tři základní vývojové větve, do kterých mohou být CFEngine klienti přiřazeni a to:

- *master* – produkční větev,
- *testing* – testovací větev,
- *development* – vývojová větev.

Klienti jsou rozděleni do jednotlivých větví správcem, přičemž:

- vývojová větev *development* by měla obsahovat 1 až 5 čistě testovacích serverů bez produkčního použití a měla by být použita pro vytvoření nové politiky,
- vývojová větev *testing* by měla obsahovat 5 až 10 produkčních serverů a měla by být použita pro testovací nasazení nově vytvořené politiky,
- vývojová větev *master* by měla obsahovat všechny zbývající produkční servery.

Stanovené počty serverů nejsou kritické, pouze větev *testing* by měla být udržována dostatečně velká na to, aby dostatečně pokryla heterogenitu prostředí (různé typy a verze OS, různé aplikační použití, atd.) a zároveň dostatečně malá, aby rozsah případné havárie byl omezený a zároveň bylo možné případné způsobené škody opravit manuálně.

Aby bylo možné klienty rozdělovat do vývojových větví a byl automatizován proces vydávání nových verzí politiky, a tudíž byl splněn požadavek 2. a 3., bylo nutné změnit výchozí adresářovou strukturu CFEngine a upravit proces aktualizace politik klientů.

[23] Detailní popis použití systému GIT lze nalézt např. v knize *Version Control with GIT*, ISBN: 978-0-596-52012-0.

V kořenovém adresáři CFEngine byl vytvořen nový adresář `masterfiles_branches`, který slouží k uložení nových CFEngine repozitářů pro jednotlivé vývojové větve. Dále byly vytvořeny adresáře pro každou vývojovou větev, každý je obdobou adresáře `masterfiles` v původní konfiguraci:

```
mkdir /var/cfengine/masterfiles_branches/

mkdir /var/cfengine/masterfiles_branches/master/
mkdir /var/cfengine/masterfiles_branches/testing/
mkdir /var/cfengine/masterfiles_branches/development/
```

Do hlavní vývojové větve bylo potřeba naklonovat prázdný testovací GIT repozitář, aby bylo možné používat nástroj `git`:

```
git clone /var/cfengine/repository/ /var/cfengine/masterfiles_branches/master/.
```

V dalším kroku byly zkopírovány soubory původní politiky a proveden počáteční commit:

```
cp -r /var/cfengine/masterfiles/* /var/cfengine/masterfiles_branches/master/

cd /var/cfengine/masterfiles_branches/master/
git init
git add *
git commit -m "Vložení výchozího zdrojového kódu"
git branch --set-upstream master origin/master
git push
```

Větev testing byla odvozena od hlavní větve naklonováním po předchozím commitu:

```
git clone /var/cfengine/repository/ /var/cfengine/masterfiles_branches/testing/.

cd /var/cfengine/masterfiles_branches/testing/
git branch testing
git commit -m "Vytvoření větve testing"
git branch --set-upstream testing origin/testing
git push
```

Větev *development* byla vytvořena analogicky k větvi *testing*. Všem větvím byl nastaven parametr `--set-upstream` tak, aby každý příkaz `push` aktualizoval příslušnou větev v repozitáři.

Stahování jednotlivých větví CFEngine klienty je možné řešit dvěma různými způsoby:

1. Klient si stáhne všechny větve a pak se sám v okamžiku běhu `promises.cf` rozhodne, kterou větev spustit, podle třídy do které patří.
2. Klient si stáhne pouze soubory konkrétní větve, podle třídy do které patří.

Pro testovací prostředí byl zvolen druhý způsob. Ten je výhodný, protože vlastní běh `promises.cf` s výkonnou politikou je naprosto odstíněn od rozhodování o větvích a je tak

naprosto transparentní. Rozhodování o stažení větve je provedeno už při aktualizaci politiky v souboru `update.cf`. Tím se minimalizuje možnost narušení výběru větví uživatelskou chybou v souboru `promises.cf`, která by mohla způsobit rozšíření testované politiky na všechny produkční stroje. Další výhodou je snížení objemu přenesených dat. Tato výhoda je ovšem pouze zdánlivá, protože CFEngine klient stahuje pouze ty soubory, u kterých došlo ke změně, přičemž testování souborů na změnu je prováděno jen v případě, signalizuje-li server změnu politiky změnou souboru `cf_promises_validated`. Úspora přenesených dat oproti prvnímu způsobu se tak projeví pouze při prvním stažení politiky po instalaci nového klienta a je zanedbatelná.

Zvolený způsob stahování vyžadoval několik úprav souboru `update.cf`. Nejprve bylo nutné definovat třídy pro rozdělení strojů. Testovací server `hucan2.zcu.cz` byl přiřazen do vývojové větve, testovací server `hucan3.zcu.cz` byl přiřazen do testovací větve, zbylé servery nejsou definovány:

```
classes:

   "branch_development" or => { "hucan2_zcu_cz" };

   "branch_testing" or => { "hucan3_zcu_cz" };
```

Na základě definovaných tříd byla deklarována proměnná `branch`[24]:

```
vars:

   any::
      "branch" string => "master",
               policy => "overridable";

   branch_testing::
      "branch" string => "testing",
               policy => "overridable";

   branch_development::
      "branch" string => "development",
               policy => "overridable";
```

Dále byly deklarovány proměnné definující cestu k adresáři `masterfiles_branches` a k adresáři zvolené větve:

```
vars:

   "branches_root" string => "/var/cfengine/masterfiles_branches";

   "my_branch_location" string => "$(branches_root)/$(branch)";
```

[24] Taková konstrukce je v rozporu s dříve popsanými doporučeními pro vytváření proměnných. V tomto případě se není doporučením vhodné řídit, protože jeho slepé dodržování by vedlo ke zbytečné komplikovanosti politiky.

Deklarované proměnné byly využity v modifikované verzi algoritmu kopírování souborů politiky ze serveru na klienta:

```
files:

   am_policy_hub|validated_updates_ready::

      "$(inputs_dir)"
            copy_from => u_rcp("$(mybranch_location)","$(sys.policy_hub)"),
         depth_search => u_recurse("inf"),
          file_select => u_input_files,
               action => u_immediate,
              classes => u_if_repaired("update_report");
```

Po přidání předchozího kódu bylo nutné upravit definici třídy `validated_updates_ready`. Tato třída je nastavena, pokud došlo ke změně souboru `cf_promises_validated` na serveru. Princip zápisu do souboru je ukázán na Obr. 5. Na serveru je pravidelně spouštěn klient `cf-agent`, který při svém spuštění provádí kontrolu politiky na změnu a syntaktickou správnost. Pokud došlo ke změně, zapíše aktuální datum a čas do souboru `cf_promises_validated`. Ostatní klienti zkontrolovanou politiku stáhnou, pouze pokud došlo ke změně tohoto souboru.

Obr. 5

V nově definované adresářové struktuře však takový postup není možný, protože `cf-agent` provádí kontrolu adresáře `inputs` tzn. pouze vlastní větve. Pokud dojde ke změně v jiné větvi, není tato změna detekována a klienti si aktualizovanou politiku nestáhnou. Tento proces je bohužel v `cf-agentu` pevně naprogramován a nelze jej měnit konfigurací. Náhradním řešením by mohlo být automatizované spouštění nástroje `cf-promises`, který provádí identickou kontrolu kódu jako `cf-agent`, ten ale soubor `cf_promises_validated` nevytváří. Z tohoto důvodu bylo vytváření souboru zakomponováno do modulu pro automatickou aktualizaci CFEngine repozitářů.

Automatická aktualizace CFEngine repozitářů pro jednotlivé větve z GIT repozitáře byla realizována modulem[25]:

```
commands:

   "$(branches_root)/update.sh master"
       module => "true";

   "$(branches_root)/update.sh testing"
       module => "true";

   "$(branches_root)/update.sh development"
       module => "true";
```

Modul `update.sh` je Bash skript, který spouští příkaz `git pull` ve zvolené větvi a kontroluje jeho výstup:

```
#!/bin/bash

cd /var/cfengine/masterfiles_branches/$1
gitout = `git pull 2>/dev/null`

if [[ $gitout =~ ^.*up-to-date.*$ ]]
   then
      echo "-branch_$1_has_changed"
   else
      echo "+branch_$1_has_changed"
fi
```

Kód skriptu byl zjednodušen pro názornost, kompletní skript viz Příloha 1: Modul aktualizace z GIT repozitáře. Pokud příkaz `git pull` vrací hodnotu odlišnou od regulárního výrazu `^.*up-to-date.*$`, znamená to, že ve větvi byla provedena změna a je nastavena třída `branch_$1_has_changed`, jinak je třída odebrána. Pokud je tato třída nastavena v `update.cf`, pak CFEngine zapíše datum a čas do souboru `validated` v příslušné větvi:

```
files:

   am_policy_hub.branch_master_has_changed::

      "$(branches_root)/master_validated"
            create => "true",
         edit_line => adddatetime($(date_time));

   am_policy_hub.branch_testing_has_changed::

      "$(branches_root)/testing_validated"
            create => "true",
         edit_line => adddatetime($(date_time));

   am_policy_hub.branch_development_has_changed::

      "$(branches_root)/development_validated"
            create => "true",
         edit_line => adddatetime($(date_time));
```

[25] Modul je speciální případ promise typu *command*, viz 2.4.5.10 Commands.

Tím bylo zajištěno automatické vytváření alternativy k souboru `cf_promises_validated` pro každou větev, mohla tak být upravena definice třídy `validated_updates_ready`:

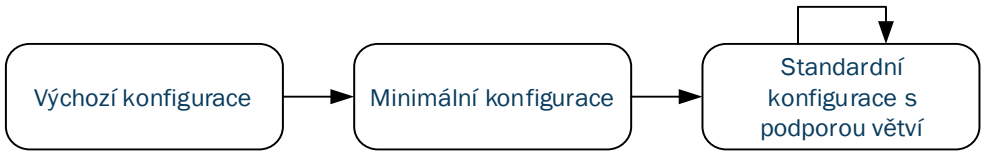
```
files:

   "$(inputs_dir)/$(branch)_validated"
      copy_from => u_rcp("$(branches_root)/$(branch)_validated","$(sys.policy_hub)"),
         action => u_immediate,
        classes => u_if_repaired("validated_updates_ready");
```

Na závěr bylo nutné povolit kopírování souborů z nově vytvořené adresářové struktury v nastavení serveru v souboru `controls/cf_serverd.cf`:

```
access:

   "$(sys.workdir)/masterfiles_branches"
      admit => { ".*$(def.domain)", @(def.acl) };
```

Původní CFEngine adresář `masterfiles`, který sloužil jako repozitář politik pro klienty byl ponechán pouze s minimální verzí konfigurace. Nemá žádnou provozní funkci, ale obsahuje upravený soubor `update.cf`, který zajistí správnou aktualizaci klientů podle verzovacího systému. Tento adresář je kritický pro nově nainstalované klienty spuštěné s parametrem `--bootstrap`. Každý nově nainstalovaný klient, který provádí první aktualizaci politiky, používá vygenerovaný soubor `update.cf`, který vznikne prvním spuštěním klienta bez politiky. Tento soubor nelze ovlivnit, protože je pevně uložen v binárním kódu programu a předpokládá standardní cesty k repozitáři na serveru. Pokud je adresářová struktura změněna, je důležité v původním repozitáři ponechat minimální konfiguraci s upraveným souborem `update.cf`, který již provede následující aktualizaci podle nové struktury.

Životní cyklus souboru `update.cf` je na Obr. 6. Novější soubor `update.cf` je vždy stažen s novější politikou.



Obr. 6

Princip fungování je následující:

1. **Výchozí** `update.cf` aktualizuje politiku z výchozí cesty k repozitáři.
2. **Minimální** `update.cf` aktualizuje politiku z větve *master*.
3. **Standardní** `update.cf` aktualizuje politiku z větve podle přiřazení klienta k větvi.

Výchozí politika není spravována verzovacím systémem GIT, protože nejsou předpokládány žádné úpravy.

3.1.3 Design Center

Design Center je repozitář předpřipravených komponent zvaných *sketch*. Hlavním cílem je umožnit uživatelům používat pokročilé funkce bez nutnosti učit se deklarativní jazyk CFEngine. Vlastní sketche jsou v tomto jazyce vytvořeny. Před instalací je nutné nastavit předem definované proměnné, k tomuto účelu v komunitní verzi slouží sada konzolových nástrojů vytvořených v jazyce Perl. [71]

Design Center není součástí základní instalace, je považován za doplňkový framework, který je veřejně dostupný na serveru Github.com[26] a musí být ručně doinstalován. Do projektu je možné přispívat vlastními úpravami zdrojových kódů nástrojů, také je možné vytvářet vlastní sketche. Obojí podléhá schválení správci projektu.

Každý sketch se skládá minimálně ze dvou částí a to ze:

- souboru JSON s metadaty, např. `sketch_apache.json`;
- obecného souboru politiky, např. `apache.cf`.

Soubor JSON obsahuje metadata ke sketchi, např.:

- název,
- popis,
- verzi,
- licenci,
- jména autorů.

Dále obsahuje interface sketche, především definici proměnných, kterým mají být před nasazením nástrojem `cf-sketch` přiřazeny hodnoty. [72]

V rámci testování byl Design Center nainstalován na server `hucan.zcu.cz`. Protože jsou nástroje vytvořené v jazyce Perl, nebylo třeba stahovat binární balík ani zdrojové kódy překládat. Postačilo pouze naklonovat Git repozitář projektu do kořenového adresáře CFEngine:

```
git clone https://github.com/cfengine/design-center.git /var/cfengine/design-center
```

[26] https://github.com/cfengine/design-center

Nástroj `cf-sketch` je konzole, do které se interaktivně zadávají příkazy. Součástí testování bylo nasazení sketche pro instalaci webového serveru Apache na zvolených klientech:

```
cf-sketch> install Apache::Simple
Sketch Apache::Simple installed under /var/cfengine/masterfiles/sketches.

cf-sketch> define params Apache::Simple
Please enter a name for the new parameter set: apache_params
hostname: hucan2.zcu.cz
port [80]: 8080
ssl_port [443]: 4433
docroot: /var/www
Parameter set apache_params successfully defined.

cf-sketch> define environment -n apache_env hucan2_zcu_cz
Environment 'apache_env' successfully defined.

cf-sketch> activate Apache::Simple apache_params apache_env
Using generated activation ID 'Apache::Simple-2'.
Using existing parameter definition 'apache_params'.
Defining new environment named 'apache_env' for class expression 'apache_env'
Activating sketch Apache::Simple with parameters apache_params.

cf-sketch> deploy
Runfile /var/cfengine/masterfiles/cf-sketch-runfile.cf successfully generated.
```

Předchozí nastavení lze popsat v krocích:

1. Sketch byl z repozitáře nainstalován do `masterfiles` CFEngine.
2. Byly nadefinovány parametry sketche. Každý nastavovaný parametr je definován v souboru JSON a odpovídá některé proměnné v soboru politiky.
3. Bylo definováno prostředí pro spuštění sketche, tím se omezilo jeho spuštění pouze na klienty odpovídající definovaným třídám.
4. Byla vytvořena nová aktivace kombinací sketche, parametrů a prostředí. Ke každému sketchi je možné vytvořit více aktivací různými kombinacemi parametrů a prostředí.
5. Všechny aktivace byly nasazeny. Byl vygenerován soubor `cf-sketch-runfile.cf` do `masterfiles`.

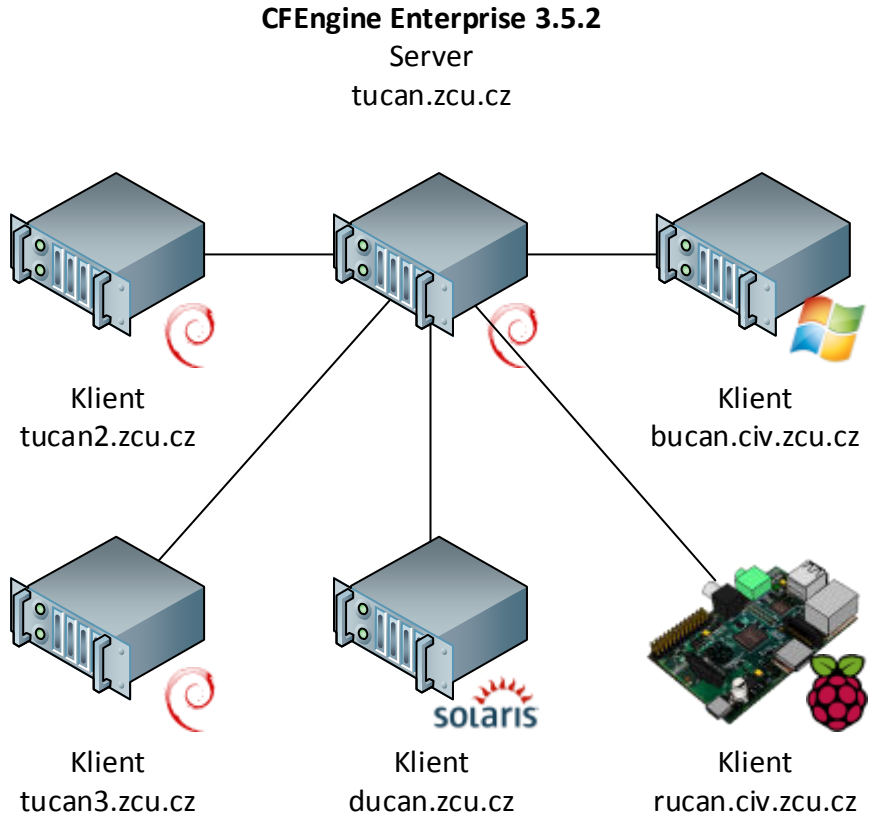
V hlavním souboru politiky `promises.cf` je do `bundlesequence` přidán bundle `cfsketch_run` ze souboru `cf-sketch-runfile.cf`. Pro každou aktivaci je v souboru `cf-sketch-runfile.cf` vygenerován bundle, který je spouštěn jako metoda bundle `cfsketch_run`. Tento bundle obsahuje nastavené proměnné z uživatelského vstupu programu `cf-sketch` a spouští obecný soubor politiky sketche, který používá nastavené proměnné. Dále bundle omezuje spuštění sketche podle definované class expression.

3.2 Laboratorní nasazení 2 - CFEngine 3 Enterprise Edition - Free 25 Node

Pro účely testování *CFEngine 3 Enterprise Edition – Free 25 Node* bylo ve virtualizačním prostředí CIV ZČU vytvořeno pět virtuálních serverů s operačními systémy:

- Debian GNU/Linux 7.1 Wheezy,
- Oracle Solaris 10,
- Microsoft Windows Server 2008 R2.

Dále bylo do testovacího prostředí zařazeno Raspberry Pi s operačním systémem Raspbian Linux (ARMv6).



Obr. 7

3.2.1 Instalace serveru a klientů

Instalace CFEngine 3 Enterprise Edition verze Free 25 Node může být provedena třemi způsoby:

- volně dostupnými skripty pro rychlou instalaci[27],
- stažením a nainstalováním volně dostupných balíků[28],
- stažením a nainstalováním balíků dostupných pouze pro obchodní partnery[29].

[27] http://cfengine.com/docs/3.5/getting-started-installation-installing-enterprise-free.html
[28] https://cfengine.com/enterprise-download
[29] https://cfengine.com/software

Kvůli nejširším možnostem testování byla zvolena poslední varianta instalace. Po kontaktování zástupce firmy CFEngine AS byl získán přístup k nejnovějším balíkům pro všechny vyžádané operační systémy.

Na rozdíl od komunitní verze je rozlišován instalační balík pro server (hub) a balík pro klienty. Server obsahuje mnoho rozšiřujících komponent a bylo by zbytečné takový balík instalovat na klienty. Pro server je balík dostupný pouze pro OS Linux.

Na `tucan.zcu.cz` byl nainstalován balík `cfengine-nova-hub_3.5.2-1_amd64.deb` pro server. Postup instalace byl obdobný jako u Community Edition s rozdílem, že v případě Enterprise Edition bylo nutné balík ručně stáhnout z úložiště na webové stránce CFEngine. Instalace byla následně provedena nástrojem `dpkg`:

```
dpkg -i cfengine-nova-hub_3.5.2-1_amd64.deb
```

Po instalaci balíku byly do masterfiles zkopírovány předem připravené politiky:

```
cp –r /var/cfengine/share/NovaBase/* /var/cfengine/masterfiles/
```

Po instalaci bylo zjištěno, že na serveru zůstaly pozůstatky předchozí klientské instalace CFEngine Community Edition. Databáze původní instalace byly nevhodně integrovány do nové instalace a projevovaly se neočekávanými výstupy ve webovém rozhraní Enterprise Edition. Pro produkční instalace je třeba dbát na dokonalé odstranění souborů předchozích instalací, nejlépe kompletním vymazáním obsahu adresáře `/var/cfengine/`.

Klient pro **GNU/Linux Debian** byl nainstalován na `tucan2.zcu.cz` a `tucan3.zcu.cz` obdobným způsobem:

```
dpkg -i cfengine-nova_3.5.2-1_x86_64.deb
```

Bootstrap klienta Enterprise Edition je shodný s Community Edition:

```
cf-agent --bootstrap --policy-server 147.228.52.142
```

Klient pro **Oracle Solaris** byl nainstalován ze Solaris balíku `cfengine-nova-3.5.3-i386-solaris.pkg` na `ducan.zcu.cz`:

```
su
pkgadd -d ./CFEcfengine-nova-3.5.3-i386-solaris.pkg
./cf-agent --bootstrap 147.228.52.142
```

Na závěr byl nainstalován klient pro **Microsoft Windows** z instalačního balíku `cfengine-nova-3.5.3.0-i686.msi` na `bucan.civ.zcu.cz`:

```
msiexec /i cfengine-nova-3.5.3.0-i686.msi
```

Zajímavé je provedení bootstrapu v prostředí Microsoft Windows, které nejsou POSIX kompatibilní a cesta k binárním souborům CFEngine je odlišná:

```
C:\Program Files\Cfengine\bin\cf-key.exe
C:\Program Files\Cfengine\bin\cf-agent.exe --bootstrap 147.228.52.142
```

Instalátor pro Microsoft Windows na rozdíl od ostatních OS nespustí generování klíčů klienta automaticky, je nutné je před bootstrapem vygenerovat ručně nástrojem `cf-key.exe`. Pokud klíče nejsou vygenerovány, bootstrap skončí selháním.

Instalace CFEngine na Raspberry Pi byla provedena odlišným způsobem, protože pro Raspberry neexistuje balík CFEngine Enterprise Edition ani Community Edition. Instalace je detailně popsána v 3.2.7 Raspberry Pi.

3.2.2 Zpětná vazba

Významným rozšířením CFEngine 3 Enterprise Edition oproti Community Edition je možnost automatického informování o stavu systému. CFEngine shromažďuje historii, stav a data o změnách jednotlivých klientů a slučuje je do jednoho celku. Strategií CFEngine je nahradit konvenční CMDB (Configuration Management Database) více rozšiřitelným a flexibilním přístupem k získávání informací o systému. [73]

V Community Edition se zpětná vazba omezuje pouze na uživatelsky generované zprávy vytvářené promise typu `reports`. Tímto způsobem je možné administrátorovi systému zasílat e-maily z každého klienta samostatně, přičemž zprávy musí být vhodně zakomponovány do kódu bundle. Žádné další prostředky pro zpětnou vazbu nejsou do Community Edition zakomponovány.

Naproti tomu Enterprise Edition přináší významné rozšíření možností zpětné vazby. Lokální stavové databáze klientů jsou periodicky nahrávány zpět na server bez zásahu administrátora. Na serveru jsou klientské databáze sloučeny do jednotné databáze reprezentující celkový stav systému. V této databázi jsou uložena data o zpracování politik např.:

- které bundle byly spuštěny,
- které promise byly dodrženy nebo opraveny,
- jaké třídy byly nastaveny.

Dále jsou v této databázi uchovávána data získaná při průzkumu klientského systému tj. v okamžiku zjišťování podkladů pro nastavení hard classes. Taková data mohou být např.:

- název a verze operačního systému;
- varianta a verze CFEngine klienta;
- počet, typ a zatížení procesorů;
- volné místo na pevných discích;
- síťová rozhraní, jejich IP adresy a průměrné vytížení;
- otevřené TCP porty;
- nainstalované balíky a jejich možné aktualizace.

Z toho vyplývá, že centrální databáze neobsahuje jen výsledky zpracování politik, ale všeobecné informace o každém klientovi. Ty slouží jako dynamická dokumentace, která je automaticky aktualizována bez zásahu administrátora, přesně podle základních principů CFEngine. Dokumentace je tak vždy aktuální a administrátor se může soustředit na vytváření nových řešení.

Zpětná vazba je realizována démonem `cf-hub`, který běží pouze na serveru. Je spuštěn nástrojem `cf-agent`, který zároveň dohlíží na jeho běh. Démon `cf-hub` se periodicky připojuje k démonu `cf-serverd` každého klienta a stáhne lokální data. Ve výchozí konfiguraci je každých 5 minut stažen tzv. delta report, který obsahuje pouze rozdílová data proti minulému stažení. Každých 6 hodin je potom stažena kompletní databáze. Data z klientských databází jsou následně uložena do centrální MongoDB databáze, která je součástí instalace Enterprise Edition. [74]

Uživatelské zobrazení dat a vytváření reportů bylo ve verzích před 3.5 prováděno nástrojem `cf-report`. Tento nástroj byl prohlášen za zastaralý a ve verzích od 3.5 není přítomen. Nově je součástí instalace Enterprise Edition webový server Apache. Ten poskytuje dvě rozhraní pro získávání informací z databáze:

- webové uživatelské rozhraní Mission Portal,
- programátorské rozhraní Enterprise API.

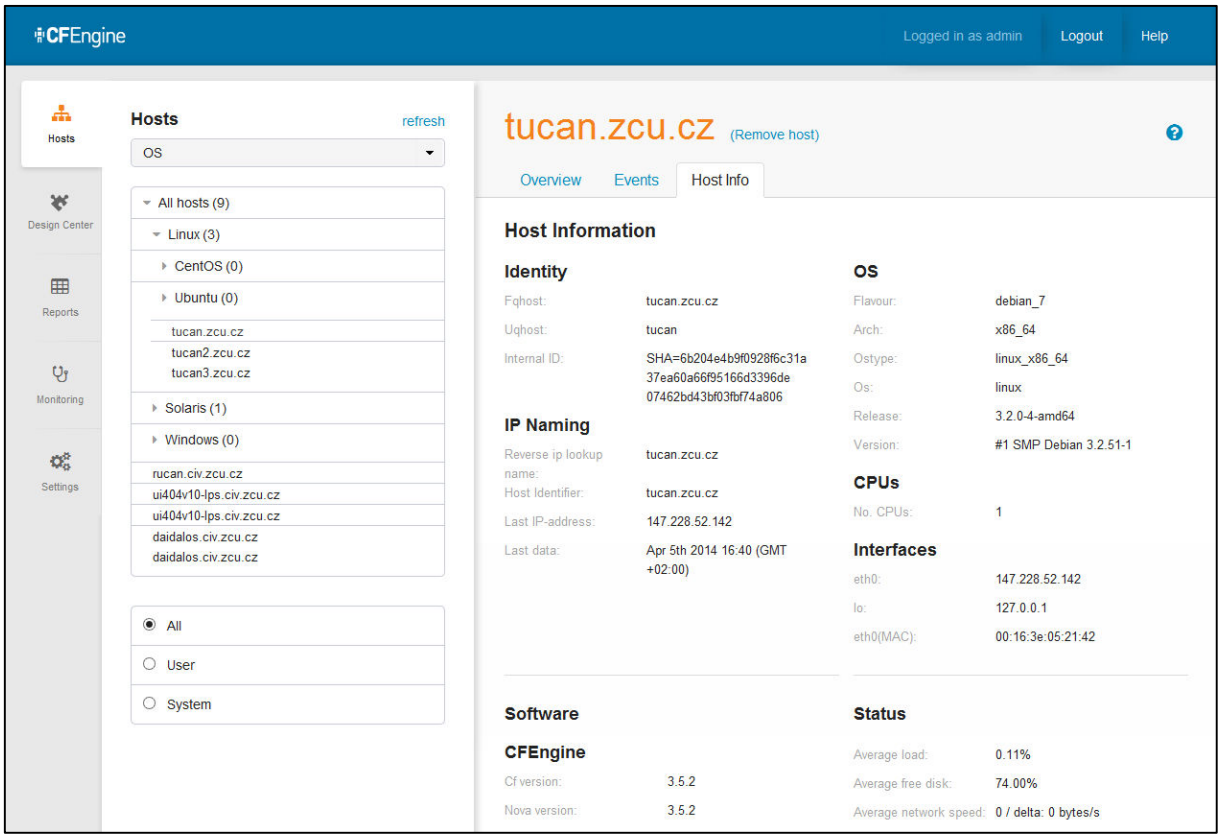
3.2.3 Mission Portal

Mission Portal je modulární webové uživatelské rozhraní, které se skládá z aplikací. Ve verzi CFEngine Enterprise Edition 3.5.2 jich obsahuje těchto pět:

- Hosts,
- Reports,
- Monitoring,
- Settings,
- Design Center GUI (volitelná instalace).

Aplikace *Hosts* (na Obr. 8) obsahuje seznam všech klientů setříděný podle operačních systémů. Ke každému klientu lze:

- zobrazit základní souhrn informací,
- zobrazit rozšířené informace (viz 3.2.2 Zpětná vazba),
- nastavit tzv. *tracker* (hlídání stavu slibu nebo třídy).



Obr. 8

Aplikace *Reports* obsahuje 29 předdefinovaných reportů s možností vytváření vlastních. Vlastní reporty jsou vytvářeny jako SQL dotaz do centrální databáze, přičemž dotaz je možné přímo zapsat nebo je možné jej postupně vytvořit s pomocí interaktivního formuláře. Výsledek je vždy tabulka hodnot. Předdefinované reporty nelze měnit, ani zobrazit jejich kód.

Aplikace *Monitoring* zobrazuje grafy využití procesoru, paměti, diskového prostoru a dalších hodnot jako např. počet přihlášených uživatelů apod. Sledovaných veličin je celkem 31, další nové veličiny nelze do sledování přidávat ani z něj odebírat. Taktéž není možné jakkoliv měnit nastavení předdefinovaných grafů.

Aplikace *Settings* umožňuje provádět základní nastavení Mission Portalu. Je možné nastavovat:

- uživatele a uživatelské role,
- možnosti autentizace (interní, LDAP, Active Directory),
- zobrazení aplikací,
- repozitář systému správy změn GIT,
- výchozí identifikátor klienta pro použití v portálu.

Aplikace *Design Center GUI* slouží jako webové rozhraní pro nástroj Design Center.

Provoz v testovacím prostředí ukázal, že Mission Portal je v současné verzi vhodný především pro získání informace, kolik a kterých klientů server používá a kdy naposledy provedli aktualizaci. Dále je vhodný k získání informace o úspěšnosti zpracování nově přidaného bundle resp. o plnění nově zavedeného slibu. Obě tyto funkce jsou kritické pro udržení konzistentního stavu konfigurace u rozsáhlých systémů. Přestože CFEngine sám o sobě dbá na dodržování konzistentního stavu, nově vytvořená politika může obsahovat chybu, která se v heterogenním prostředí může projevit pouze na některých klientech, aniž by se předtím objevila v testovacím prostředí. Při použití kvalitního nástroje pro zpětnou vazbu lze takovou chybu rychle odhalit a přijmout opatření k jejímu odstranění. Zbývající funkce Mission Portalu jsou neúplné. Funkčnost bude zřejmě v budoucích verzích rozšířena, o syrovosti portálu vypovídá např. absolutní absence jakékoliv dokumentace (kromě Design Center GUI).

3.2.4 Design Center GUI

Aplikace Design Center GUI slouží jako webové rozhraní pro nástroj Design Center. Jejím účelem je zjednodušení ovládání nástrojů Design Center a přidání rozšiřujících funkcí. GUI umožňuje:

- nasazovat sketche kompatibilní s verzí Enterprise bez přístupu ke konzoli OS,
- sledovat průběh nasazení sketchů na klienty,
- podávat zprávy o souladu s nasazenou politikou.

Sketche kompatibilní s Design Center GUI jsou podmnožinou sketchů obecně, z celkově dostupných 80 sketchů je s GUI kompatibilních pouze 7. Předpokladem pro používání GUI je instalace verzovacího systému GIT a jeho nastavení v Mission Portalu.

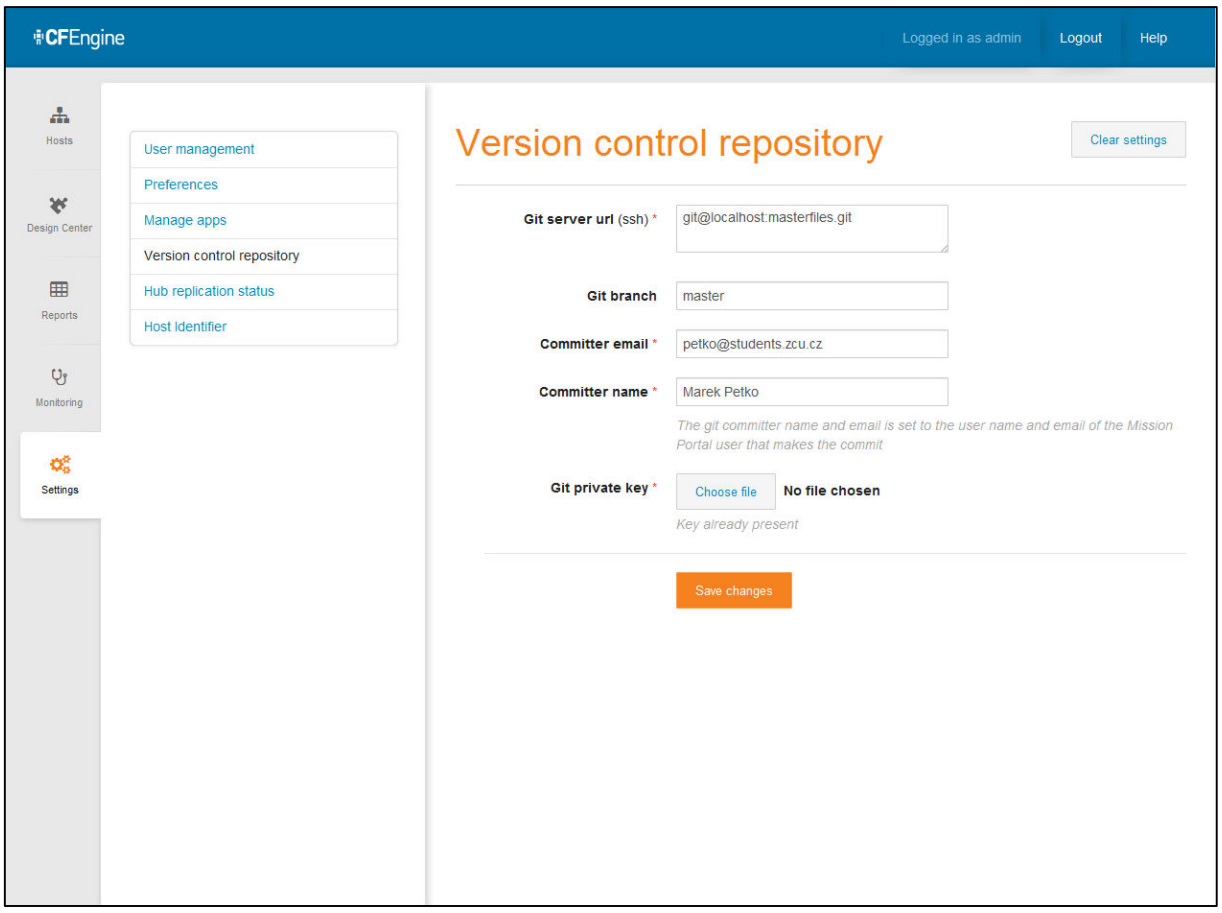
Instalace systému GIT byla na testovacím serveru `tucan.zcu.cz` provedena podle CFEngine manuálu[30]. V porovnání s řešením, které bylo navrženo v 3.1.2 Správa verzí politiky, je tato instalace výrazně zjednodušená, používá pouze jednu vývojovou větev a nepodporuje tudíž žádná testovací prostředí.

[30] https://cfengine.com/docs/3.5/manuals-design-center-integrating-mission-portal-with-git.html

Na rozdíl od navrženého řešení, které pro přístup k repozitáři používá AFS, Mission Portal vyžaduje připojení přes SSH. Z toho důvodu bylo nutné vygenerovat SSH klíče pro uživatele `git`:

```
ssh-keygen -C 'Mission Portal' -N '' -f mission_portal_id_rsa
cat mission_portal_id_rsa.pub >> ~/.ssh/authorized_keys

chown git:git ~/.ssh/authorized_keys
chmod 700 ~/.ssh/authorized_keys
```

Integrace systému GIT do Mission Portalu byla provedena v aplikaci Settings, v záložce *Version control repository*, viz Obr. 9. Pro úplnou funkčnost bylo třeba do Mission Portalu nahrát soukromý SSH klíč vygenerovaný v předchozím kroku.



Obr. 9

Po úspěšné integraci systému GIT do portálu byly v Design Center GUI na vybrané testovací klienty nainstalovány dva sketche:

- `Database::Install::MySQL::Simple`,
- `Database::Install::PostgreSQL::Simple`.

V případě MySQL serveru se z pohledu Mission Portalu instalace jevila jako úspěšná. Při kontrole na testovacích serverech se ovšem ukázalo, že instalace balíku neproběhla ani v jednom případě. V případě PostgreSQL serveru nebyla instalace sketche vůbec dokončena,

podle Mission Portalu zůstala aktivace ve stavu *in progress* a ani po několika dnech se tento stav nezměnil. Příčinu tohoto selhání se nepodařilo odhalit. Částečně proto, že Enterprise Edition obsahuje proprietární rozšíření s uzavřeným zdrojovým kódem. Tato rozšíření jsou popsána pouze strohým uživatelským manuálem, bez jakékoliv technické dokumentace. Podle vyjádření zástupce firmy CFEngine AS je Design Center GUI stále ve fázi intenzivního vývoje a proto může k takovým selháním docházet.

Výsledky testování tohoto rozšíření jsou převážně negativní. Cílem Design Center je umožnit začínajícím uživatelům CFEngine nasazovat účelové politiky bez hlubší znalosti jazyka CFEngine. GUI má nasazování těchto politik ještě více zjednodušit. Pokud ovšem přihlédneme k faktu, že samotná instalace GUI vyžaduje ruční instalaci systému GIT a netriviální zásah do politik pro aktualizaci, pak lze tento doplněk jen stěží považovat za jednoduchý pro začátečníky. Pokud bychom připustili, že v prostředí Mission Portalu může pracovat více uživatelů, tedy alespoň jeden zkušený administrátor a jeden začátečník, pak stejně nelze GUI pro ZČU doporučit, kvůli vysoké nespolehlivosti při nasazování sketchů. Pokud by byl tento nedostatek v budoucích verzích odstraněn, pak by GUI bylo vhodným nástrojem, který by umožnil ostatním administrátorům na ZČU jednoduše využívat výhod hromadné správy.

3.2.5 Enterprise API

Enterprise API je REST API, které umožňuje HTTP klientům získávat informace z centrální databáze serveru. API umožňuje uživateli:

- získat základní informace o API;
- spravovat uživatele, uživatelské role a nastavení;
- procházet informace o klientech;
- pokládat SQL dotazy na data v centrální databázi;
- plánovat připravené reporty.

API bylo otestováno v základním rozsahu pro ověření funkčnosti a seznámení se s jeho použitím.

Ověření funkčnosti bylo provedeno s pomocí nástroje CURL[31]:

```
curl -k --user admin:password http://tucan.zcu.cz/api/
```

[31] http://curl.haxx.se/

Dále bylo otestováno položení SQL dotazu přes REST API:

```
curl -k --user admin:password http://tucan.zcu.cz/api/query -X POST -d \
   '{ "query": "SELECT Hosts.Hostname FROM Hosts" }'
```

SQL dotaz v příkladu vrátí datovou strukturu obsahující doménová jména všech známých klientů.

Kompletní reference příkazů API[32] a SQL schématu[33] je uvedena v manuálu CFEngine.

3.2.6 Windows a Solaris

Součástí Enterprise testovacího prostředí byly počítače s operačními systémy Microsoft Windows a Oracle Solaris. Cílem bylo otestovat fungování CFEngine v heterogenním prostředí. Instalace na obou platformách je popsána v 3.2.1 Instalace serveru a klientů.

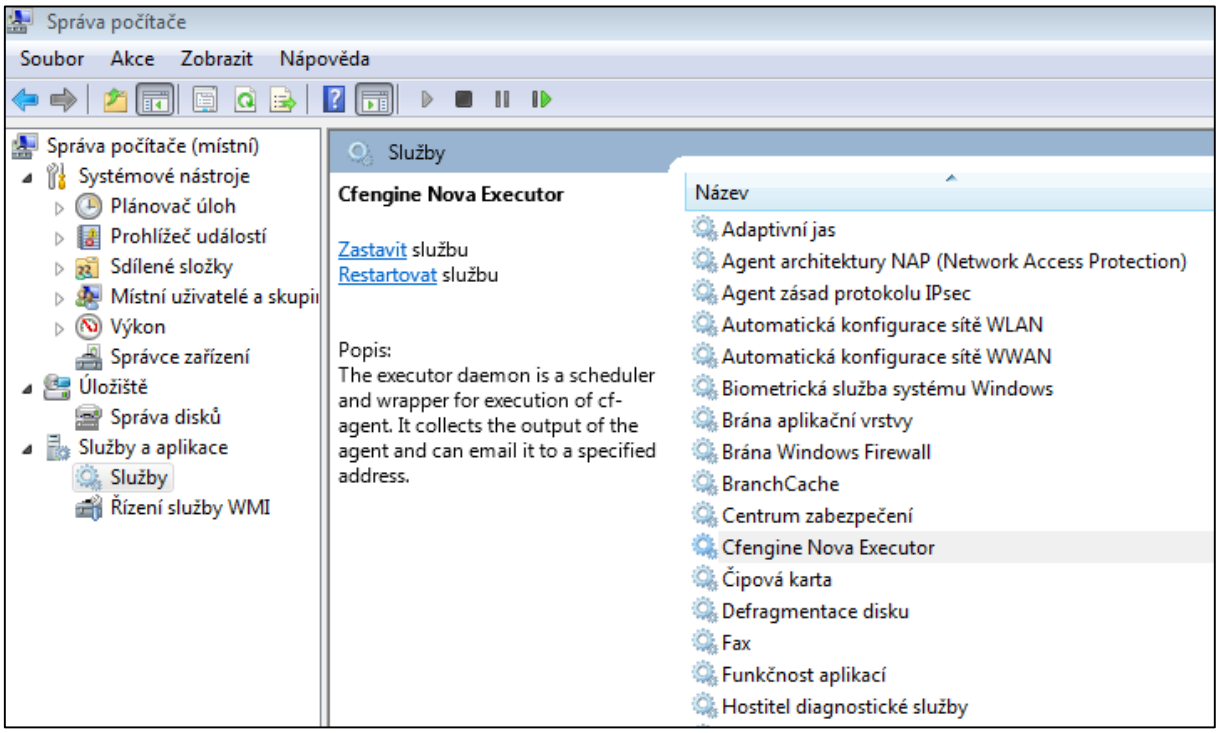
Pro účely otestování multiplatformních politik byl vytvořen testovací bundle `os_test`:

```
bundle agent os_test
{
   files:

      solaris::
         "/tmp/it_works_on_solaris"
            create => "true";

      windows::
         "C:\temp\it_works_on_windows"
            create => "true";

      linux::
         "/tmp/it_works_on_linux"
            create => "true";
}
```

Bundle vytvoří na každém klientovi soubor s názvem odpovídajícím běžícímu operačnímu systému. Po nasazení bundle se na všech připojených klientech do několika minut objevily správné soubory. Cesty jsou v tomto případě uvedeny absolutně, alternativně je možné použít funkci `translatepath()`, která cestu přeloží z obvyklého POSIX formátu do nativního formátu pro OS klienta.

[32] https://cfengine.com/docs/3.5/reference-enterprise-api-uri-resources.html
[33] https://cfengine.com/docs/3.5/reference-enterprise-api-sql-schema.html

V OS Solaris běží `cf-execd` jako démon podobně jako na OS Linux. Na Windows tento nástroj běží jako služba systému, viz Obr. 10.



Obr. 10

Pro Solaris CFEngine neposkytuje žádná speciální rozšíření, pro Windows je možné navíc spravovat klíče v registrech a služby. Praktická funkce, která v CFEngine pro Windows chybí, je správa aktualizací.

3.2.7 Raspberry Pi

Raspberry Pi je počítač velikosti kreditní karty, který pracuje na architektuře ARMv6. Celý počítač je proveden jako System on Chip, tzn., že procesor, grafická karta a další periferie jsou integrovány do jednoho čipu. [75] V laboratorním nasazení bylo pro testování použito Raspberry s nainstalovaným operačním systémem Raspbian, což je neoficiální port GNU/Linux Debian pro architekturu ARMv6. Testování na této platformě bylo zajímavé nejen z důvodů otestování možností CFEngine, ale taktéž proto, že v budoucnu budou některé servery poháněny právě touto architekturou. Společnost AMD začátkem roku 2014 vydala první ARM procesory určené pro servery. Jejich výhodou v porovnání s platformou x86 je levnější a ekologičtější provoz. [5]

Pro operační systém Raspbian ani pro architekturu ARMv6 neexistuje žádný instalační balík CFEngine. Jedinou možností, jak CFEngine otestovat bylo stáhnout zdrojový kód Community Edition[34] a ten přímo na Raspberry přeložit. Zdrojový kód CFEngine je napsán v jazyce C99, tudíž z principu by měl být multiplatformní, na druhou stranu záleží na vývojářích, jestli kód vytvářejí tak, aby skutečně byl.

[34] http://cfengine.com/source-code

Stažení, přeložení a instalace byly provedeny následovně:

```
apt-get install libtokyocabinet-dev openssl libssl-dev libpcre3 libpcre3-dev
wget -O cfengine-3.5.3.tar.gz \
http://cfengine.com/source-code/download?file=cfengine-3.5.3.tar.gz
tar -xf cfengine-3.5.3.tar.gz
cd cfengine-3.5.3/
./configure
make
make install
cd /var/cfengine/bin/
./cf-key
./cf-agent --bootstrap 147.228.52.142
```

Do operačního systému bylo nejdříve nutné nainstalovat knihovny, které byly nutnou podmínkou pro přeložení. Potom byla provedena konfigurace a přeložení nástrojem *make*. Po instalaci byly vygenerovány klíče nástrojem `cf-key` a na závěr byl proveden bootstrap.

Přeložení i bootstrap CFEngine klienta proběhl bez potíží, zdrojový kód CFEngine Community Edition je tedy vysoce přenositelný. Počítač s nainstalovaným Community klientem může pracovat pod Enterprise serverem, jen nezpracovává sliby dostupné pouze v Enterprise Edition a neposkytuje zpětnou vazbu serveru.

Pro otestování funkčnosti provádění politik byl vytvořen jednoduchý scénář: Raspberry Pi bude zastávat roli webserveru na kterém bude nainstalován démon `lighttpd`, který bude prezentovat webové stránky připravené v repozitáři na CFEngine serveru.

Zdrojový kód bundle následuje:

```
bundle agent rpi
{
   packages:

      rucan_civ_zcu_cz::
         "lighttpd"
            package_policy => "add",
            package_method => apt;

   files:

      rucan_civ_zcu_cz::
         "/var/www/index.html"
            copy_from => remote_cp("/var/cfengine/masterfiles/files/index.html",
                                   "$(sys.policy_hub)"),
            perms => m("644");

         "/var/www/index.lighttpd.html"
            delete => tidy;
}
```

V sekci `packages` je definován slib vyžadující přítomnost balíku `lighttpd`. Pokud na serveru není, má být nainstalován balíčkovacím systémem `apt`. V sekci `files` je slib, který dohlíží na přítomnost souboru `index.html` v kořenovém adresáři pro webové dokumenty.

Pokud na serveru není, bude stažen z repozitáře na CFEngine serveru. Poslední slib dohlíží na odstranění výchozího souboru `index.lighttpd.html` z kořenového adresáře.

Vytvořená politika byla klientem na Raspberry úspěšně zpracována a během několika minut už Raspberry odpovídalo na požadavky webového prohlížeče, viz Obr. 11. Tím lze testování na Raspberry Pi zhodnotit jako velice úspěšné, připomeneme-li fakt, že Raspberry není oficiálně podporovaná platforma.

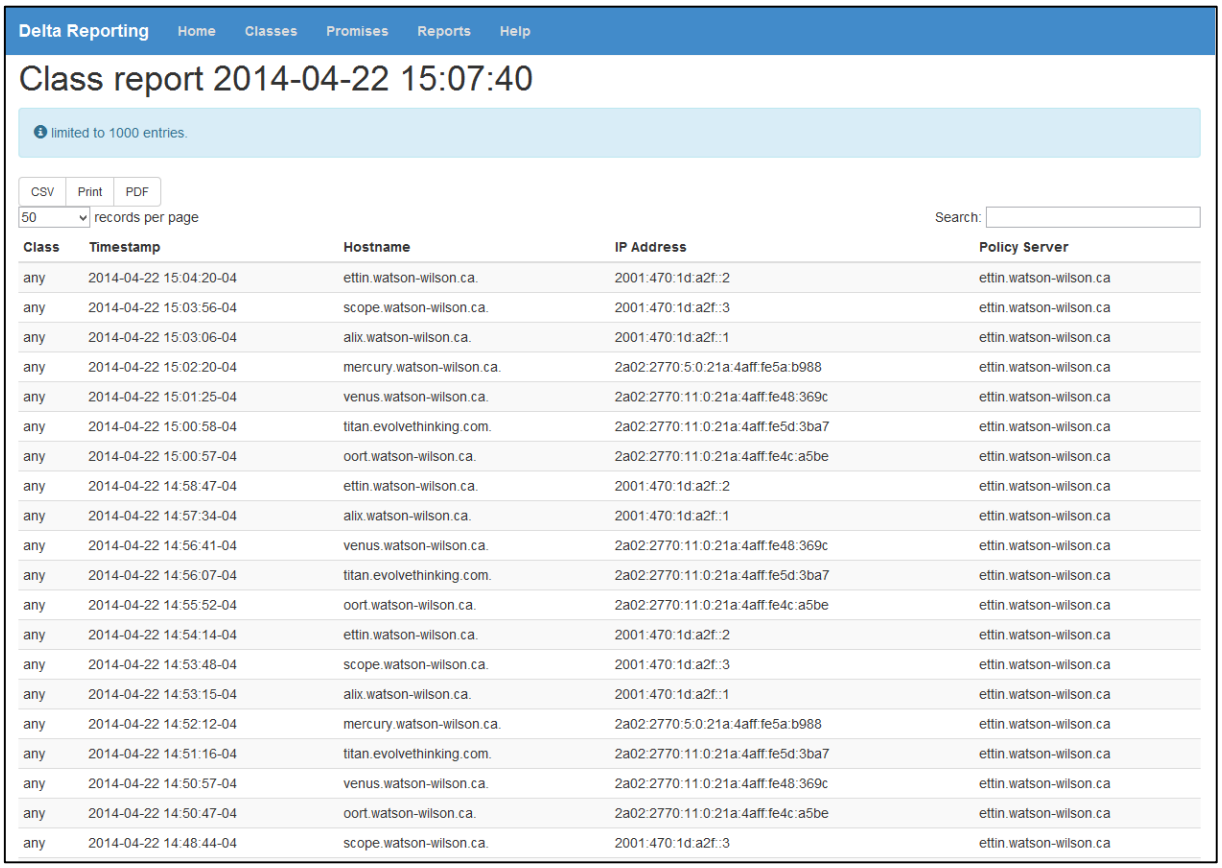
Obr. 11

3.2.8 Nahrazení Enterprise funkcí

Na závěr testování Enterprise Edition byla zvažována otázka, jestli je možné nějakým způsobem, byť v minimálním rozsahu, nahradit reportování klientů zpět na server v Community Edition. Každý klient má ve výchozí konfiguraci spuštěný démon `cf-serverd`, pokud by tedy bylo možné nalézt nebo vygenerovat ladicí výstup, mohla by být vytvořena politika, na základě které by se server připojil ke klientům a tento výstup stáhl do centrálního úložiště.

Průzkum výstupů CFEngine bohužel ukázal, že firma CFEngine AS se snaží udělat vše pro to, aby nástroje v Community Edition generovaly co nejmenší nebo pokud možno žádný výstup do logovacích souborů. Většina souborů je prázdných nebo neposkytuje zajímavé informace. Vzhledem k tomu, že funkce pro zpětnou vazbu jsou zásadním rozšířením Enterprise Edition, dá se předpokládat, že jde o záměrnou ochranu před řešeními třetí strany. Tento předpoklad podporuje i odstranění nástroje `cf-report` z Community Edition od verze 3.5.

Těsně před dokončením tohoto dokumentu byl publikován otevřený nástroj Delta Reporting[35] firmy Evolve Thinking Ltd., která se specializuje na poradenství v oblasti konfiguračního managementu s CFEngine. Delta Reporting je webové rozhraní, které používá CFEngine politiku pro vygenerování reportů o nastavených třídách a stavech slibů na klientech. Politika na serveru reporty kopíruje zpět na server a vkládá je do centrální PostgreSQL databáze. Obsah databáze je možné prohlížet ve webovém rozhraní, viz Obr. 12. Nevýhodou tohoto nástroje je, že bezpodmínečně funguje pouze reportování nastavených tříd, kde je k získání seznamu používána funkce `classesmatching()`. Reportování stavu jednotlivých slibů je řešeno atributem `action`, který používá body z knihovny Evolve Free Library. Tento atribut musí být uveden u každého slibu, jinak není jeho stav zaznamenán.



Class	Timestamp	Hostname	IP Address	Policy Server
any	2014-04-22 15:04:20-04	ettin.watson-wilson.ca.	2001:470:1d:a2f::2	ettin.watson-wilson.ca
any	2014-04-22 15:03:56-04	scope.watson-wilson.ca.	2001:470:1d:a2f::3	ettin.watson-wilson.ca
any	2014-04-22 15:03:06-04	alix.watson-wilson.ca.	2001:470:1d:a2f::1	ettin.watson-wilson.ca
any	2014-04-22 15:02:20-04	mercury.watson-wilson.ca.	2a02:2770:5:0:21a:4aff:fe5a:b988	ettin.watson-wilson.ca
any	2014-04-22 15:01:25-04	venus.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe48:369c	ettin.watson-wilson.ca
any	2014-04-22 15:00:58-04	titan.evolvethinking.com.	2a02:2770:11:0:21a:4aff:fe5d:3ba7	ettin.watson-wilson.ca
any	2014-04-22 15:00:57-04	oort.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe4c:a5be	ettin.watson-wilson.ca
any	2014-04-22 14:58:47-04	ettin.watson-wilson.ca.	2001:470:1d:a2f::2	ettin.watson-wilson.ca
any	2014-04-22 14:57:34-04	alix.watson-wilson.ca.	2001:470:1d:a2f::1	ettin.watson-wilson.ca
any	2014-04-22 14:56:41-04	venus.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe48:369c	ettin.watson-wilson.ca
any	2014-04-22 14:56:07-04	titan.evolvethinking.com.	2a02:2770:11:0:21a:4aff:fe5d:3ba7	ettin.watson-wilson.ca
any	2014-04-22 14:55:52-04	oort.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe4c:a5be	ettin.watson-wilson.ca
any	2014-04-22 14:54:14-04	ettin.watson-wilson.ca.	2001:470:1d:a2f::2	ettin.watson-wilson.ca
any	2014-04-22 14:53:48-04	scope.watson-wilson.ca.	2001:470:1d:a2f::3	ettin.watson-wilson.ca
any	2014-04-22 14:53:15-04	alix.watson-wilson.ca.	2001:470:1d:a2f::1	ettin.watson-wilson.ca
any	2014-04-22 14:52:12-04	mercury.watson-wilson.ca.	2a02:2770:5:0:21a:4aff:fe5a:b988	ettin.watson-wilson.ca
any	2014-04-22 14:51:16-04	titan.evolvethinking.com.	2a02:2770:11:0:21a:4aff:fe5d:3ba7	ettin.watson-wilson.ca
any	2014-04-22 14:50:57-04	venus.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe48:369c	ettin.watson-wilson.ca
any	2014-04-22 14:50:47-04	oort.watson-wilson.ca.	2a02:2770:11:0:21a:4aff:fe4c:a5be	ettin.watson-wilson.ca
any	2014-04-22 14:48:44-04	scope.watson-wilson.ca.	2001:470:1d:a2f::3	ettin.watson-wilson.ca

Obr. 12

[35] https://github.com/evolvethinking/delta_reporting

3.3 Puppet

Po otestování CFEngine 3 Community Edition a Enterprise Edition byla uspořádána odborná diskuse se správcem serverů projektu MetaCentrum. Součástí diskuse byla praktická ukázka.

MetaCentrum je aktivita sdružení CESNET, jejímž dlouhodobým cílem je provoz a koordinace distribuované výpočetní infrastruktury. MetaCentrum umožňuje využití dostupných výpočetních zdrojů pro řešení velmi náročných výpočetních úloh, jejichž zvládnutí je nad možností samostatného pracoviště v ČR. [76]

V MetaCentru je používáno přibližně 800 serverů s operačním systémem GNU/Linux Debian. Na jejich správu je používán nástroj Puppet ve variantě Open Source.

3.3.1 Funkce Puppetu

Protože Puppet historicky vychází z CFEngine 2, jsou některé principy jeho fungování zachovány. Příkladem je stahování politiky do uzlu, kdy klient stahuje politiku autonomně metodou *pull* a server funguje pouze jako repozitář. Politiku není možné nahrávat ze serveru metodou *push*, tím je zajištěna vysoká bezpečnost, protože potenciální útočník nemůže do uzlu nahrát vlastní škodlivý kód. Aby bylo možné rychle rozšířit vybrané kritické aktualizace politiky, je absence *push* nahrazena tzv. metodou *kick*. Server pouze upozorní klienta na změnu v repozitáři a ten následně metodou *pull* stáhne novou politiku.

Každý klient stahuje zjednodušenou, částečně přeloženou verzi politiky. Za běžných podmínek tedy není možné politiku na uzlu vytvářet nebo upravovat a potom autonomně spouštět.

Konfigurace serveru a politiky pro uzly jsou v Puppetu oddělené a používají každá jiný formát zápisu. Konfigurační soubory serveru nemají žádné rysy programovacího jazyka a používají pouze plochý formát běžný pro většinu statických konfiguračních souborů v operačním systému Linux. Naproti tomu politiky pro uzly jsou vytvářeny v deklarativním programovacím jazyce obdobně jako v CFEngine a jsou nazývány *manifest*. Politika je strukturována do modulů, kde každý modul obsahuje:

- manifesty,
- šablony konfiguračních souborů,
- obecné soubory (např. konfigurační soubory, binární soubory).

Šablony a obecné soubory jsou klientem zkopírovány do uzlu a následně jsou zpracovány politikou v manifestu.

Speciální manifest *nodes* v kořenovém adresáři politik určuje, které moduly budou spuštěny na kterém uzlu. Uzly je možné zvolit jednotlivě nebo skupinově s použitím regulárního výrazu.

Pro vytváření manifestů mohou být použity knihovny vytvářené v jazyce Ruby. Tím je možné výrazně rozšířit funkčnost celého manifestu. Výhodou je použití procedurálního

jazyka pro zápis složitějších algoritmů, jejichž vytvoření v deklarativním jazyce s omezenými funkcemi může být pro běžného administrátora složité[36]. Vytvořené knihovny jsou automaticky kopírovány do všech uzlů.

Zjištění informací o uzlu je prováděno lokálně nástrojem *facter*. Nástroj se spustí automaticky po uplynutí stanoveného intervalu, prozkoumá parametry hardware, operačního systému a vytvoří globální proměnné, které mohou být později použity v manifestech. Tyto proměnné jsou zpravidla používány pro větvení běhu programu. Facter má několik desítek předdefinovaných proměnných, další lze definovat uživatelskou knihovnou v jazyce Ruby.

Manifesty k volnému použití jsou v rámci komunity sdíleny online na webové stránce Puppet Labs[37].

Zpětná vazba je v Open Source variantě řešena nástrojem *Puppet Dashboard*, který je zdarma dostupný na serveru Github.com[38]. Správce má k dispozici webové rozhraní, ve kterém může sledovat průběh nasazení nově přidaného modulu a další. Toto rozhraní je odlehčenou variantou rozhraní ve variantě Enterprise.

Puppet nativně umožňuje vytvářet více prostředí pro běh. Volbou parametru při spuštění klienta na uzlu lze ovlivnit, z kterého prostředí bude politiky stahovat a interpretovat.

3.3.2 Použití v MetaCentru

V MetaCentru je většina serverů konfigurována výhradně nástrojem Puppet bez použití konzole. Puppet byl vybrán kvůli možnosti rychlého nasazení, protože byl pro správce snadno ovladatelný. Spuštění klienta na uzlu a zpracování politiky trvá většinou kolem 10 sekund. Dlouhý běh klienta a vyšší výpočetní náročnost zatím nejsou považovány za problém. Za velkou výhodu jsou považovány knihovny pro manifesty a pro nástroj *facter* a jsou hojně využívány např. pro rozlišení virtuálního serveru od fyzického. Komunitní manifesty z webové stránky Puppet Labs nejsou používány, vše je ručně vytvářeno. Politika je spravována verzovacím nástrojem GIT s jednou spouštěnou větví, nově vytvořené balíky jsou testovány v separátním prostředí pro běh. Puppet Dashboard je využíván v omezené míře pouze pro sledování rozšíření nově nasazeného modulu. Závěry o funkčnosti nově nasazeného modulu jsou odvozovány z monitorovacího systému Nagios.

3.3.3 Srovnání s CFEngine 3

V porovnání s CFEngine 3 je Puppet nástroj, který má spoustu praktických vlastností, které vychází z jeho převážně obchodního zaměření. CFEngine je v tomto ohledu více akademický a některé vlastnosti užitečné pro firemní nasazení zatím postrádá.

Základní princip přenosu politiky ze serveru na klienta je totožný, CFEngine ale přenáší politiku kompletní v původním upravitelném formátu. Politika tak může být testována

[36] Nejrozšířenější programovací jazyky jsou založeny na imperativním paradigmatu, viz http://langpop.com.
[37] http://forge.puppetlabs.com
[38] https://github.com/puppetlabs/puppetlabs-dashboard

a upravována lokálně na libovolném klientovi, dokonce může být na klienta nahrána libovolným jiným způsobem, např. SCP protokolem nebo pomocí USB flash disku. Klient je zcela autonomní a nezávislý na okolních vlivech.

Konfigurace serveru, politiky pro klienty a externí knihovny jsou CFEngine jednotně řešeny výhradně deklarativním jazykem a jsou součástí politiky. To má výhodu v unifikaci přístupu a ve větších možnostech přizpůsobení, včetně možnosti přizpůsobení procesu stahování politik. Na druhou stranu jsou tím kladeny vyšší nároky na znalosti a zkušenosti administrátora, který takovou politiku vytváří. Zejména používání deklarativního jazyka pro složitější úkony např. parsování textu, může být obtížné. V CFEngine 3 je takové úkoly možné řešit modulem.

Základní struktura konfigurace a politik je v Puppetu jasně dána. To umožňuje začínajícímu uživateli se rychle zorientovat a začít vytvářet politiky bez nutnosti uvažovat nad vhodnou strukturou. CFEngine je pravým opakem, struktura zcela závisí na uživateli. Uživatel má naprostou volnost, bohužel výsledné řešení nemusí být optimální. Z tohoto důvodu je od CFEngine verze 3.5 základní politika po čisté instalaci alespoň minimálně strukturována a na webové stránce CFEngine[39] jsou popsané nejlepší praktiky pro strukturování politiky. I přesto zůstává mnoho otázek, které musí uživatel vyřešit sám.

Pro zjišťování informací o systému je v Puppetu použit nástroj facter. Obdobou u CFEngine je program `cf-monitord`, který monitoruje stav systému a vytváří proměnné, které jsou interpretovány jako hard classes v programu `cf-agent`. Místo knihoven mohou být v CFEngine použity politiky, které vytvářejí soft classes, od verze 3.6 nazývané *inventory*.

Obdobou sdílených manifestů Puppetu jsou v CFEngine *sketches*. CFEngine tento koncept rozšiřuje a tyto předem připravené politiky jsou interaktivně konfigurovatelné a nasaditelné s pomocí nástroje design-center, bez nutnosti zasahovat do zdrojového kódu, viz 3.1.3 Design Center.

Zpětná vazba je v Puppet Open Source dostupná po doinstalování Puppet Dashboard, případně jiného nástroje třetí strany. V CFEngine 3 Community Edition taková možnost není a alternativní řešení třetí strany je jen obtížně realizovatelné, viz 3.2.8 Nahrazení Enterprise funkcí. Úplná zpětná vazba v CFEngine 3 je tedy možná pouze u varianty Enterprise Edition.

Různá prostředí pro běh nejsou v CFEngine nativně podporována, ale díky konfigurovatelnosti procesu stahování je možné takové chování docílit, viz 3.1.2 Správa verzí politiky.

[39] https://cfengine.com/docs/3.5/manuals-writing-policy.html

Ze srovnání je zřejmé, že oba nástroje pokrývají přibližně stejnou funkcionalitu, ale každý jiným způsobem. Puppet se vyvinul na základech CFEngine 2 a je zaměřen především na jednoduchost použití a funkce potřebné pro praktické nasazení. CFEngine 3 naproti tomu stojí na pevném teoretickém základu a klade důraz na jednotnost, efektivitu a robustnost. S komercializací CFEngine jsou postupně i do Community Edition doplňovány funkce pro účinné praktické nasazení.

3.4 Srovnání variant

Výsledkem testování CFEngine Community Edition a Enterprise Edition je závěr, že za současných podmínek je nejvhodnější do produkčního prostředí CIV ZČU nasadit CFEngine Community Edition.

Varianta Enterprise Edition sice přináší užitečné funkce zpětné vazby pro administrátora, ale za cenu velké investice. V počáteční fázi nasazování lze tyto funkce oželet a omezit se pouze na základní informování o plnění slibů. V okamžiku, kdy bude počet klientů takový, že stávající model zpětné vazby nebude dostačovat, lze na Enterprise Edition plynule přejít. Enterprise Edition server může pracovat s Community Edition klienty, migrace by měla být bez komplikací.

Nejzajímavější komponentou možné budoucí instalace Enterprise Edition je webové rozhraní Mission Portal. Vzhledem k rychlému vývoji placené varianty lze předpokládat, že webové rozhraní bude rozšiřováno o nové funkce a budoucí investice se vyplatí více než by se vyplatila nyní.

Pokud by v budoucnu měl být spravován větší počet počítačů s operačním systémem MS Windows, pak by se Enterprise Edition taktéž vyplatila. V Community Edition mohou být konfigurovány pouze počítače s klienty v prostředí Cygwin a to jen s omezenými funkcemi.

3.5 Produkční nasazení

V prostředí CIV ZČU pracuje 250 serverů s operačním systémem GNU/Linux Debian, 25 serverů s operačním systémem MS Windows a 11 serverů s operačním systémem Oracle Solaris. Dále je spravováno přibližně 1300 klientských počítačů s MS Windows. Pro správu koncových stanic je již vytvořeno vlastní řešení správy konfigurace. Pro správu serverů žádné takové řešení neexistuje, proto bylo nasazení CFEngine zaměřené především na produkční servery. Na vybrané z nich byl CFEngine postupně nasazován.

V produkční instalaci systému CFEngine byly aplikovány všechny poznatky z testování otevřené i placené varianty CFEngine. Produkční server byl nasazen na server `daidalos.civ.zcu.cz`, kterému byl také přiřazen doménový alias `cf.civ.zcu.cz`.

Instalace CFEngine klienta byla zařazena do systému pro automatickou instalaci serverů FAI[40]. Všechny nové Debian servery nainstalované tímto nástrojem CFEngine klienta obsahují. Klient po instalaci provede bootstrap se serverem a je tak automaticky zařazen do systému hromadného konfiguračního managementu.

3.5.1 Správa verzí politiky

V produkčním prostředí byl aplikován systém správy verzí nástrojem GIT vyvinutý v prvním laboratorním nasazení, viz 3.1.2 Správa verzí politiky.

Repozitář GITu byl umístěn na AFS, tím byla zajištěna správa přístupu a zároveň tím bylo vyřešeno zálohování.

Pro produkční prostředí byl systém omezen pouze na dvě větve:

- *master* – testovací větev,
- *production* – produkční větev.

Vývojová větev byla vypuštěna, protože vývoj probíhá na lokálním počítači administrátora, kde může být politika během vývoje lokálně testována. Po otestování je proveden push na server, před kterým je automaticky zkontrolována syntaxe politiky. Pokud politika není syntakticky správně, je odmítnuta a push do repozitáře na serveru není proveden.

Větev *master* je poněkud nezvykle použita jako testovací (na rozdíl od laboratorního nasazení). Cílem tohoto je zabránit náhodnému commitu kódu do produkční větve. Po provedení příkazu `git clone` na počítači administrátora je jako výchozí přístupná hlavní vývojová větev *master* a mohlo by dojít k situaci, kdy administrátor omylem rozšíří nevhodný kód na desítky produkčních klientů.

Administrátoři CIVu byli s použitím systému správy verzí seznámeni na veřejné prezentaci.

3.5.2 Organizace politiky

Struktura politiky není v CFEngine nijak pevně definována. CFEngine se tímto od konkurenčních produktů odlišuje, protože vše je chápáno jako politika, včetně konfigurace serveru nebo procesu aktualizace klientů. V kódu politiky jsou definovány pouze dva soubory, které slouží jako vstupní body pro nástroj `cf-agent`:

- `promises.cf`, vstupní soubor do výkonné politiky;
- `update.cf`, vstupní soubor do politiky aktualizace.

V každém z těchto souborů je `body common control`, které obsahuje atribut `bundlesequence`. Tomuto atributu je přiřazen `slist` s názvy bundle v pořadí, v jakém mají být spuštěny. Jak je politika dále strukturována, závisí zcela na administrátorovi, jako v běžných programovacích jazycích.

V testovacím prostředí na ZČU bylo ověřeno, že spouštění jednotlivých výkonných bundle v ploché struktuře za sebou odporuje vytváření složitějších celků a znovupoužitelných

[40] http://fai-project.org/

bundle. Taktéž vazba mezi počítačem a bundle byla těžko definovatelná a celý kód nebyl vhodný pro víceuživatelské úpravy. Proto byl pro produkční použití na ZČU vytvořen návrhový vzor pro strukturování politik, který popsané nevýhody řeší.

Základním prvkem navržené struktury je služba. Služba je jeden nebo více bundle, které zajišťují všechny činnosti nutné k poskytnutí služby zákazníkovi (ten může být i interní). Příkladem budiž služba jednoduchého webserveru. Pro zajištění takové služby je potřeba:

- nainstalovat a udržovat balík webserveru např. `apache2`;
- nainstalovat a udržovat podpůrné balíky např. `libapache2-mod-php5`;
- vytvořit nebo zkopírovat a udržovat nastavení webserveru;
- zkopírovat a udržovat prezentovaný obsah tj. PHP skripty, HTML šablony, obrázky;
- spustit službu webserveru a dohlížet nad jejím chodem.

Všechny tyto činnosti jsou shrnuty do jednoho nebo více bundle ve stromové struktuře, přičemž kořenový bundle reprezentuje službu jako celek.

Službu webserveru z příkladu by bylo možné realizovat s pomocí dvou bundle v souborech:

- `apache.cf`,
- `content.cf`.

Bundle `apache` by byl odpovědný za instalaci, nastavení a běh webserveru. Bundle `content` by byl odpovědný za aktuální a správný obsah.

Struktura bundle `apache` by byla následující:

```
bundle agent apache
{
   packages:

      "apache2"...;
      "libapache2-mod-php5"...;

   files:

      "/etc/apache2/sites-available/mujweb" copy_from =>...;
      "/etc/apache2/sites-enabled/mujweb" link_from =>...;

   services:

      "apache2"...;
}
```

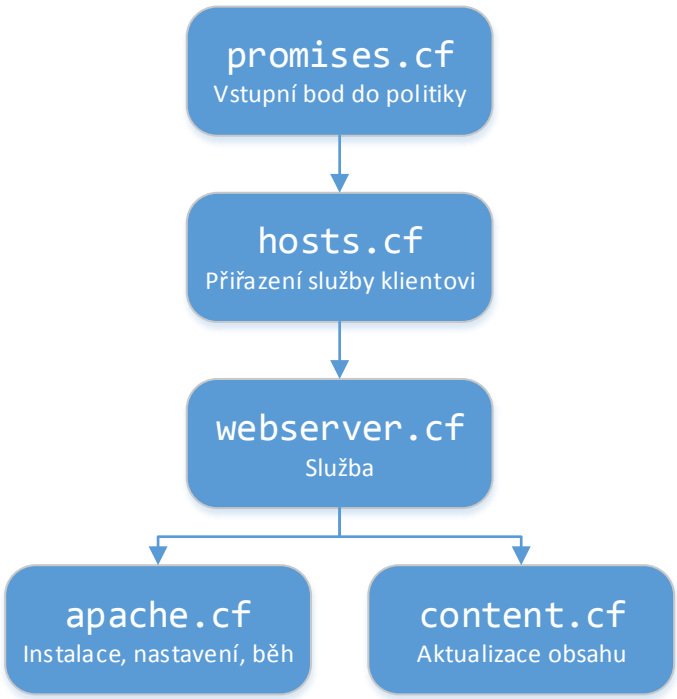
Struktura bundle `content` by byla následující:

```
bundle agent content
{
   files:
      "/var/www" copy_from =>...;
}
```

Přiřazení služby nebo služeb počítačům je prováděno ve speciálním bundle `hosts` v souboru `hosts.cf`. Tím je zajištěna jasná identifikace, který z počítačů jakou službu poskytuje a zároveň je umožněno méně zkušeným administrátorům služby počítačům přidávat nebo odebírat. Navržený soubor hosts.cf má pak zcela záměrně rysy konfiguračního souboru:

```
bundle agent hosts
{
   methods:

      any::
         "any" usebundle => ntp, comment => "Služba NTP klienta";

      webserver_zcu_cz::
         "any" usebundle => webserver, comment => "Služba webserveru";

      ftpserver_zcu_cz::
         "any" usebundle => ftpserver, comment => "Služba FTP serveru";
}
```

Bundle služeb jsou volány s pomocí promise typu `methods`, viz 2.4.5.15 Methods. Strom volání bundle pro službu webserver z příkladu je znázorněn na Obr. 13.



Obr. 13

Takto navržená struktura byla vložena do počátečního commitu do produkčního GIT repozitáře a je používána pro všechny nově nasazované služby. Administrátoři CIVu byli s použitím návrhového vzoru seznámeni na veřejné prezentaci.

3.5.3 Příklady použití

V produkčním výpočetním prostředí ZČU bylo vytipováno několik praktických problémů, na kterých bylo možné otestovat možnosti CFEngine v reálném provozu. Cílem bylo vyřešit stanovený problém automatizací tak, aby nebyl nutný ruční zásah administrátora.

3.5.3.1 NTP klient

Synchronizace času napříč síťovým prostředím je jedním z klíčových činitelů pro správnou funkci informačního systému a pro odhalování vzniklých chyb. Běžné hodiny reálného času počítače jsou schopné udržovat vysokou přesnost, která má být dostačující pro běžného uživatele, není však dostatečná pro činnost některých síťových programů. Příkladem takového software je autentizační software Kerberos, který je v prostředí ZČU využíván. [77] Synchronizace času je také velice důležitá pro analýzu logů, kde bez přesného času nelze jednoznačně spárovat události na různých zařízeních.

V serverovém prostředí CIV se pro synchronizaci času na serverech s OS Debian GNU/Linux využívá démon `ntpd`, který pracuje jako klient pro NTP protokol. Na některých serverech bylo zaznamenáno náhodné ukončení klienta v důsledku neznámé chyby. Interval ukončení se pohyboval v řádech měsíců. Možným řešením by bylo vytvořit modul pro monitorovací systém Nagios a při výpadku informovat administrátora, který by následně odchylku od požadovaného stavu ručně opravil. Výhodnějším řešením bylo použít CFEngine. Byla vytvořena jednoduchá politika s promise typu `services`, která zajišťuje dohled nad během služby NTP. Pokud byl démon z libovolného důvodu ukončen, např. z důvodu chyby nebo ručním zásahem uživatele, CFEngine při svém nejbližším běhu odchylku od definovaného stavu detekuje a NTP démon opět spustí.

```
bundle agent ntp_run
{
   services:

      any::
         "ntpd"
            service_policy => "start";
}
```

Předchozí kód demonstruje jednoduchost deklarativního jazyka pro základní operace. Obdobný kód může být použit pro zajištění běhu libovolné služby. Ve standardní knihovně CFEngine verze 3.6 je definováno 191 běžně používaných služeb, pro které je možné použít předchozí konstrukci. Pokud služba není definována, je nutné si vytvořit vlastní body, viz 2.4.5.12 Services.

3.5.3.2 Firewall

Důležitost ochrany informačních systémů před vnějšími útočníky není třeba detailně popisovat. Většina ze současných systémů je nějakým způsobem připojena do internetu a může se stát obětí lidských útočníků nebo častěji automatizovaných robotů, kteří sami provádí útok, případně vyhledávají zranitelná zařízení. Úroveň potřebné ochrany na koncových zařízeních je zpravidla nepřímo úměrná úrovni ochrany na hranici organizace.

V prostředí CIV ZČU sahá veřejný internet hluboko do vnitřní sítě, aniž by byl uplatňován v komerční sféře oblíbený koncept demilitarizované zóny. Nedůvěryhodný vnější síťový provoz není omezován žádným hraničním firewallem a je bez omezení propouštěn do vnitřní sítě. To výrazně zvyšuje nároky na lokální zabezpečení serverů. Dalším hlediskem pro lokální zabezpečení je nebezpečí výskytu vnitřního útočníka. Ten se může v prostředí ZČU vyskytnout ve dvou variantách:

- Některý ze serverů může být napaden vnějším útočníkem, který nad ním získá kontrolu a může ho zneužít k napadení dalších zařízení uvnitř sítě.
- Studenti technických oborů, kteří se nacházejí uvnitř sítě, mají dostatečné znalosti k provedení nejrůznějších síťových útoků.

Všechny tyto důvody vedou k nutnosti použití lokálního firewallu na každém serveru. Většina moderních operačních systémů obsahuje nějaký vestavěný firewall. [78] V případě operačního systému GNU/Linux Debian, který je na serverech CIV ZČU v největším zastoupení, je firewall součástí jádra systému. Pravidla se nastavují nástrojem `iptables`. Nastavení bohužel není persistentní, a proto se používají různé nadstavby, které umožňují dosáhnout persistence a zjednodušují ovládání.

Na GNU/Linux Debian serverech v CIV ZČU se u starších instalací používá skript `iptables`, který je součástí starších verzí Debianu. U novějších instalací se používá nástroj `shorewall`[41]. Dohled nad spuštěním firewallu byl přenechán CFEngine. Byla vytvořena politika, která detekuje, jestli je firewall aktivní a pokud není, tak jej spustí.

[41] http://shorewall.net/

Kód politiky následuje:

```
bundle agent network_firewall
{
   vars:

         "iptables_check_cmd" string => "/sbin/iptables -L INPUT";
         "shorewall_cmd" string => "/etc/init.d/shorewall start";
         "iptables_script_cmd" string => "/etc/init.d/iptables start";

         "iptables_output" string => execresult($(iptables_check_cmd), "noshell");

   classes:

         "firewall_disabled"
            expression => regcmp("(?ms).*^Chain INPUT \(policy ACCEPT\)$.*",
                                 $(iptables_output));

         "shorewall"
            expression => fileexists("/etc/shorewall");

         "iptables_script"
            expression => fileexists("/var/lib/iptables/active");

   commands:

      firewall_disabled.shorewall::
         "$(shorewall_cmd)"
            classes => if_repaired("firewall_enabled");

      firewall_disabled.!shorewall.iptables_script::
         "$(iptables_script_cmd)"
            classes => if_repaired("firewall_enabled");

   reports:

      firewall_disabled.!firewall_enabled::
         "Firewall byl deaktivovan a nepodarilo se jej nastartovat.";
}
```

Politika v ukázce musí překonat dvě základní překážky:

1. Firewall je realizován v jádře systému, tudíž po spuštění skriptu nezůstává na pozadí běžet žádný proces.
2. Servery mohou obsahovat libovolnou z variant firewallu.

První překážka znemožňuje použití promise typu `services`, který byl použit u NTP démona. Proto byla kontrola stavu firewallu oddělena od jeho spuštění. Pro kontrolu stavu je spouštěn nástroj `iptables` s parametrem `-L INPUT`, který vypíše výchozí politiku a seznam pravidel. Pokud je ve výstupu detekována politika `ACCEPT`, je firewall považován za neaktivní. Nastane-li takový stav je nastavena třída `firewall_disabled`, která je podmínkou spuštění příslušného skriptu.

Druhá překážka byla odstraněna nastavením třídy `shorewall` nebo `iptables_script`, podle toho který adresář s konfigurací byl na serveru detekován. Tyto třídy byly následně použity pro rozhodování, který skript má být spuštěn, jestliže došlo k deaktivaci firewallu.

Výhody navrženého řešení jsou zřejmé:

1. Pokud administrátor při své činnosti na serveru deaktivoval firewall a zapomněl ho opět aktivovat, pak CFEngine detekuje odchylku od politiky a firewall spustí.
2. Pokud by se nějakému útočníkovi podařilo firewall deaktivovat, má ve výchozím nastavení maximálně 5 minut na provedení útoku na nechráněný server. Po uplynutí tohoto času CFEngine detekuje odchylku a firewall spustí. Tím se výrazně snižují jeho možnosti.
3. Pokud dojde vinou technické závady k selhání spuštění firewallu, administrátor je o závadě informován e-mailem a může zjednat nápravu.

Body 1. a 2. opět demonstrují základní výhody CFEngine a jejich dopad na bezpečnost. Pokud by byla odchylka od bezpečnostní politiky pouze detekována (např. systémem Nagios) a byl by nutný zásah administrátora, pak by čas odstranění takové odchylky byl značně variabilní v řádech jednotek až desítek hodin. V případě použití CFEngine je odchylka odstraněna ve fixním čase během několika minut.

3.5.3.3 Konfigurace DNS klienta

Služba překladu doménových jmen je neodmyslitelnou součástí jakékoliv sítě. V prostředí ZČU pracuje několik DNS serverů. Páteří celého DNS systému ZČU jsou servery, které sdílí anycastovou IP adresu `147.228.3.3` (`dns.zcu.cz`). Tím je zajištěno, že požadavek klienta je routery směrován na nejbližší páteřní DNS server. Koncové stanice získávají tuto adresu přes DHCP. Servery mají v tomto ohledu větší volnost, mohou používat DNS servery manuálně nakonfigurované administrátorem. Přestože se konfigurace podle lokalit liší (kolejní síť, část města, atd.), měla by odpovídat politice CIVu ZČU, která je popsána na CIV wiki[42].

[42] http://support.zcu.cz/index.php/LPS:CFengine3/Politiky

Na GNU/Linux Debian serverech se konfigurace služby překladu doménových jmen provádí v souboru `/etc/resolv.conf`. Každý administrátor může tento soubor manuálně upravovat, což vede k nejednotnému nastavení serverů. Aby bylo dosaženo standardní konfigurace v souladu s politikou, byla připravena CFEngine politika, která dohlíží na dodržování správné konfigurace:

```
bundle agent resolver
{
   vars:

      "resolv_conf" string => "/etc/resolv.conf";

      "primary_nameserver" string => "147.228.3.3";

      "default_search" string => "zcu.cz civ.zcu.cz";

      "allowed_nameservers" slist => {"127.0.0.1",
                                      "10.0.1.2",
                                      "10.190.0.1",
                                      "$(primary_nameserver)",
                                      "147.228.52.11",
                                      "147.228.10.15",
                                      "147.228.190.15",
                                      "147.228.150.5"};

      "allowed_search" string => "zcu\.cz civ\.zcu\.cz(\s[a-zA-Z]+\.zcu\.cz)?";

      "allowed_nameservers_array[nameserver\s+\Q$(allowed_nameservers)\E\s*]"
         string => "";

      "allowed_nameservers_lines"
         slist => getindices("allowed_nameservers_array");

      "allowed_search_lines"
         slist => {"search\s+$(allowed_search)\s*"};

   files:

      "$(resolv_conf)"
         create => "true",
         edit_line => fixresolv();
}
```

Konfigurace musí být v souladu s politikou CIVu ZČU, ale v různých lokalitách se liší. Proto není možné zkopírovat jeden předem připravený soubor konfigurace. Namísto toho bundle `resolver` obsahuje seznam přípustných DNS serverů a kontrolní regulární výraz pro seznam lokálních domén. V bundle je nad souborem `resolv.conf` spuštěn bundle `fixresolv`, který provádí řádkové úpravy souboru. Zdrojový kód bundle `fixresolv` viz Příloha 2: Bundle fixresolv.

Bundle provádí tyto změny:

1. Prověří záznamy `nameserver` a odstraní všechny, které nejsou uvedeny na seznamu přípustných serverů.
2. Zkontroluje záznam `search`, pokud neodpovídá kontrolnímu regulárnímu výrazu, odstraní ho.
3. Zkontroluje, jestli je mezi záznamy `nameserver` uveden primární DNS server `147.228.3.3`. Pokud záznam není nalezen, přidá jej za záznamy serverů privátních sítí, pokud jsou uvedeny. Pokud je nalezena duplicita, informuje administrátora.
4. Zkontroluje, jestli je v konfiguraci uveden záznam `search` ve standardním formátu. Pokud ne, pak přidá záznam ve výchozím formátu. Pokud je nalezena duplicita, informuje administrátora.

Takto vytvořená politika provádí řádkové operace nad téměř libovolným formátem souboru. Jejím cílem je především odstranit nežádoucí záznamy a po jejich odstranění doplnit výchozí záznamy tak, aby v každém případě byla zachována funkčnost služby.

3.5.3.4 SMART pro pevné disky

Provozní data v informačních systémech CIV ZČU jsou uložena na pevných discích, obdobně jako ve většině současných data center. Disky jsou sdružovány do externích datových polí, v některých případech jsou používány lokální disky uvnitř serverů. Lokální disky jsou používány jako systémové, ostatní data jsou ukládána do diskových polí nebo do AFS. Pevné disky mají dlouhou životnost, ale relativně nízkou spolehlivost, proto jsou kombinovány do redundantních diskových polí RAID. Pro systémové disky je z důvodů vysoké spolehlivosti používán RAID 1 (zrcadlení), pro datové disky je používán RAID 5, případně RAID 6. Tato pole jsou schopna bez ztráty dat fungovat při selhání jedno resp. dvou pevných disků (RAID 6). Takový stav se nazývá degradovaný a v případě selhání dalšího disku dochází ke ztrátě dat. Pole musí být odstaveno a musí být provedena obnova dat ze zálohy, přičemž celá tato operace znamená neplánovanou odstávku služby. Aby bylo možné výpadku předejít, je nutné výměnu selhaného disku provést co nejrychleji, případně se pokusit disk, který hrozí selháním, odhalit a preventivně nahradit.

Predikce selhání disku je komplexní úkol bez jednoduchého řešení. Většina moderních disků má zabudovánu funkci S.M.A.R.T., která zajišťuje monitorování provozních parametrů a chyb čtení na úrovni firmware. Studie společnosti Google Inc. prokázala, že zvýšený počet detekovaných chyb čtení znamená několikanásobně vyšší pravděpodobnost selhání disku v následujících 2 měsících. Na druhou stranu pouze u 57% selhaných disků byly předem detekovány S.M.A.R.T. chyby. Z toho vyplývá, že s pomocí funkce S.M.A.R.T lze odhalit disky u kterých hrozí selhání, stále ale budou existovat disky, u kterých selhání nastane bez varování. [79]

S.M.A.R.T sbírá data zcela automaticky v rámci běžného provozu. Ten nemusí pokrýt celý povrch disku a proto je vhodné jednou za měsíc provést tzv. Long Self-test, který provede kompletní kontrolu. V operačním systému GNU/Linux Debian je na CIV ZČU používán

nástroj `smartctl`, který umožňuje manuálně spustit Long Self-test na pozadí. Test nevypisuje žádné výsledky, pouze aktualizuje S.M.A.R.T. hodnoty. Tyto hodnoty jsou sledovány systémem Nagios, který upozorní odpovědného administrátora, pokud byly překročeny nastavené prahové hodnoty. Stále je ale nutné test pravidelně ručně spouštět. Tuto činnost lze automatizovat s pomocí CFEngine.

Ukázka bundle pro zajištění automatického spouštění smartctl následuje:

```
   bundle agent smarthdd
   {
      vars:

         "partitions_cmd"
            string => "/bin/cat /proc/partitions";

         "partitions_out"
            string => execresult($(partitions_cmd), "noshell");

         "dimension"
            int => parsestringarrayidx("partitions_table",
                                       "$(partitions_out)",
                                       "", "\s+", "1000", "200000");

         "partitions_keys"
            slist => getindices("partitions_table");

         "partition_names_array[$(partitions_keys)]"
            string => "$(partitions_table[$(partitions_keys)][4])";

         "partition_names_list"
            slist => getvalues("partition_names_array");

         "physical_drives_list"
            slist => filter("[sh]d[a-z]", "partition_names_list", "true", "false",
"9999");

      reports:

         "physical drives:$(physical_drives_list)";

      methods:

         "any" usebundle => smartctl("$(physical_drives_list)");
   }

   bundle agent smartctl(drive)
   {
      commands:

         "smartctl -t long $(drive)";
   }
```

Pro spuštění nástroje `smartctl` bylo nejdříve nutné zjistit nainstalované disky. CFEngine žádnou funkci pro získání jejich seznamu nemá, proto byl vytvořen postup pro jeho získání v jazyce CFEngine. Jazyk nemá příliš mnoho prostředků na parsování rozsáhlejších textů,

proto je výsledný zdrojový kód relativně komplikovaný[43]. Pro zjištění seznamu všech diskových oddílů přítomných v systému je použita funkce `execresult()`. Získaný řetězec je rozdělen funkcí `parsestringarrayidx()` do dvourozměrného pole hodnot. Z pole je vybrán 4. sloupec a funkcí `filter()` jsou vybrány pouze hodnoty odpovídající regulárnímu výrazu. Pro každou položku výsledného seznamu fyzických disků je implicitní iterací spuštěn nástroj `smartctl` s příslušným parametrem.

Spouštění bundle bylo v nadřazeném bundle omezeno pouze na fyzické servery jedenkrát za měsíc.

3.5.4 Zpětná vazba

Pro nasazení v produkčním prostředí CIV ZČU byla zvolena varianta CFEngine 3 Community Edition, která, jak bylo ověřeno v testovacím prostředí, nemá příliš mnoho funkcí pro zpětnou vazbu. Hlavním pilířem zpětné vazby produkčního nasazení se staly sliby typu `reports`, které zasílají e-mail administrátorovi. Každý z klientů zasílá e-mail samostatně zvlášť, proto bylo nutné s rostoucím počtem klientů omezit hlášené události. Nejpraktičtějším se ukázalo u klíčových slibů vytvářet třídy, kterými je vypisování hlášení omezeno. Tím se e-maily zasílané administrátorovi omezily pouze na chyby a to pouze takové, které nebyl CFEngine sám schopný opravit.

Rámcový příklad použití následuje:

```
bundle agent sluzba
{
   ...

   reports:

      doslo_k_chybe::
         "Došlo k chybě.";
}
```

Jednou z hlavních nevýhod takového přístupu je, že do výchozí konfigurace lze zapsat pouze jednu e-mailovou adresu, je tedy informována pouze jedna osoba. Nejjednodušším řešením je posílat e-maily na skupinovou adresu. Lepším řešením je využít faktu, že konfigurace CFEngine je politika zapsaná v deklarativním jazyce a je tak možné s pomocí tříd vytvořit různé konfigurace pro různé skupiny klientů.

Druhou z nevýhod je, že jsou detekovány pouze chyby politiky. Administrátor nemůže získat přehled úspěšně zpracovaných slibů apod.

[43] Pro parsování složitějších textů je vhodnější vytvořit externí CFEngine modul, který může být napsán v libovolném programovacím jazyce.

Pokud v budoucnu dosáhne počet CFEngine klientů na ZČU více jak 100, bude pravděpodobně potřeba zajistit jiné efektivnější řešení zpětné vazby. Možností je několik:

- zakoupit licence a migrovat na Enterprise Edition,
- otestovat a nasadit nástroj Delta Reporting,
- vyvinout vlastní řešení zpětné vazby.

Migrace na Enterprise Edition by byla nejvhodnějším řešením, vynaložené náklady na zakoupení by ale musely vyvážit úspory získané použitím CFEngine při správě serverů ZČU.

3.6 Nalezené chyby

Při provozu v testovacích prostředích a produkčním prostředí bylo odhaleno několik chyb v CFEngine. Tyto chyby byly nahlášeny, a pokud to bylo možné, byla oprava zdrojového kódu poskytnuta zpět do projektu CFEngine.

3.6.1 Chyba detekce instalovaných balíků

V operačním systému GNU/Linux Debian bylo pozorováno, že v některých případech nebyla korektně provedena instalace balíku. Pokud měl být nainstalován nový balík, který již v systému někdy byl, pak jeho instalace neproběhla.

CFEngine používá výpis programu `dpkg` k získání seznamu nainstalovaných balíků. Odinstalovaný balík ve výpisu `dpkg` zůstává se statusem `rc`, pokud jeho odstranění nebylo provedeno kompletně s parametrem `--purge`. Bohužel regulární výraz ve standardní knihovně detekující příslušné řádky byl zapsán tak, že detekoval i balíky se statusem `rc` jako nainstalované, pokud jejich název obsahoval znak `i`.

Chyba byla nahlášena do bugtrackeru CFEngine[44]. Pro opravu chyby byl vytvořen pull request[45], který byl schválen, a úprava byla zařazena do hlavní vývojové větve projektu[46].

3.6.2 Chyba odesílání e-mailů

Při provozu v produkčním prostředí bylo zjištěno, že od počítačů v doméně `civ.zcu.cz` nepřicházejí žádná hlášení do e-mailové schránky administrátora. Prověření na klientech ukázalo, že lokální mailserver odmítá odesílat e-maily, u kterých odesilatel přísluší do jiné domény než mailserver. Špatně nastavená adresa odesilatele, kterou CFEngine používal, byla programově vytvářena v nastavení serveru:

```
mailfrom   => "root@$(sys.host).$(def.domain)";
```

Hodnota proměnné `$(def.domain)` je ručně nastavena v konfiguraci serveru globálně pro všechny klienty. Dojde-li k situaci, že CFEngine spravuje počítače ve více doménách např. `zcu.cz` a `civ.zcu.cz`, pak je e-mailová adresa správně vytvořena pouze na počítačích v jedné doméně.

[44] https://cfengine.com/dev/issues/3908
[45] https://github.com/cfengine/masterfiles/pull/67
[46] https://github.com/cfengine/masterfiles/commit/66f4accd793cdf8e05cb9d4364e3ffdadef59840

Nalezená chyba byla nahlášena do bugtrackeru[47]. Její oprava byla provedena použitím proměnné `$(sys.fqhost)` k vytvoření e-mailové adresy. Tato proměnná je nastavena automaticky na FQDN počítače.

```
  mailfrom   => "root@$(sys.fqhost)";
```

Pro opravu chyby byl vytvořen pull request[48], který bohužel nebyl schválen. Nebylo navrženo žádné jiné řešení problému a požadavek na opravu chyby zůstává stále nevyřešen.

3.6.3 Chyby dokumentace

Kromě chyb ve zdrojovém kódu byly také odhaleny dvě chyby v on-line dokumentaci. Obě chyby byly nahlášeny do bugtrackeru[49] a byly týmem CFEngine opraveny.

3.7 Prezentace

Výsledky této práce společně se základním úvodem do konfiguračního managementu s CFEngine byly prezentovány na 44. konferenci EurOpen.CZ v Herbertově a na veřejné prezentaci CIV ZČU v Plzni.

Podklady pro prezentaci na konferenci EurOpen.CZ jsou v příloze na CD. Součástí přílohy je příspěvek do sborníku z konference (ISBN 978-80-86583-27-3).

Podklady pro prezentaci na CIV ZČU jsou v příloze na CD. Součástí přílohy je záznam z přednášky.

[47] https://cfengine.com/dev/issues/4851
[48] https://github.com/cfengine/masterfiles/pull/162
[49] https://cfengine.com/dev/issues/3942
https://cfengine.com/dev/issues/5402

4 Závěr

V úvodu práce byla provedena analýza současných řešení v oblasti konfiguračního managementu. Existující řešení poskytují různou funkcionalitu od jednoduchých operací prováděných na základě konfiguračního souboru po plně funkční programovací jazyky. Úroveň decentralizace se pohybuje od plně centralizovaných nástrojů provádějících příkazy vzdáleně po distribuované systémy s autonomními uzly. Nejpokročilejšími distribuovanými nástroji pro konfigurační management jsou ty, ve kterých uzly pracují autonomně, provádějí dohled nad lokálním systémem a veškeré nekonzistentnosti konfigurace samostatně opravují bez zásahu administrátora. Tím lze dosáhnout dlouhodobé udržitelnosti výpočetního systému. Ze srovnávaných nástrojů tyto principy nejlépe implementuje CFEngine.

V laboratorním prostředí byla otestována otevřená a komerční varianta CFEngine. Otevřená varianta byla nasazena v homogenním prostředí se servery GNU/Linux Debian, ve kterém byl později vyvinut automatizovaný systém správy verzí politiky. Komerční varianta byla nasazena v heterogenním prostředí s operačními systémy GNU/Linux Debian, Microsoft Windows a Oracle Solaris. Součástí testování bylo i nasazení na architektuře ARMv6. Pro produkční použití na CIV ZČU byla vybrána otevřená verze s možností budoucího upgrade na verzi komerční.

V produkčním výpočetním prostředí ZČU byl aplikován vytvořený systém správy verzí politiky a byl vytvořen nový návrhový vzor pro její strukturování. Obojí je nutné pro efektivní víceuživatelskou práci se systémem. Pro nejčastější konfigurační úkony bylo vytvořeno několik politik, které zároveň slouží jako referenční implementace navržené struktury. Na závěr byla evaluována zpětná vazba od klientů a bylo navrženo několik řešení pro budoucí rozšiřování systému. Nasazený systém byl prezentován na prezentaci CIV ZČU a následně byl předán k použití administrátorům. Seznam produkčních strojů, kde byl k 1. květnu 2014 nasazen lze nalézt v příloze.

V průběhu testování a produkčního nasazení bylo v CFEngine nalezeno několik chyb, které byly nahlášeny, opraveny a kód opravy byl poskytnut zpět do projektu.

Výsledky práce společně se základním úvodem do konfiguračního managementu s CFEngine byly prezentovány na 44. konferenci EurOpen.CZ v Herbertově.

Všechny body zadání byly splněny.

5 Přehled zkratek

BYOD – Bring Your Own Device
CI – Configuration Items
CLI – Command Line Interface
CMDB – Configuration Management Database
CMS – Configuration Management System
DSL – Domain Specific Language
FQDN – Fully Qualified Domain Name
ITIL – IT Infrastructure Library
ITSM – IT Service Management
LDAP – Lightweight Directory Access Protocol
NAT – Network Address Translation
PCRE – Perl Compatible Regular Expressions
PID – Process ID
POSIX – Portable Operating System Interface
PST – Pacific Standard Time
RAID – Redundant Array of Independent Disks
SLA – Service Level Agreement
SMART – Self-Monitoring, Analysis, and Reporting Technology
SSH – Secure Shell
VPN – Virtual Private Network
YAML – Yet Another Markup Language

6 Literatura

1. **Lorenz, Mirko.** How Many Servers Worldwide? *VISION Cloud.* [Online] 26. 8 2011. [Citace: 4. 5 2014.] http://www.visioncloud.eu/content.php?s=191,324.

2. **CFEngine AS.** CFEngine 3 Tutorial. *CFEngine Documentation Archive.* [Online] CFEngine AS. [Citace: 24. 4 2014.] https://cfengine.com/archive/manuals/cf3-tutorial.

3. **van Bon, Jan, Nugteren, Marianne a Polter, Selma.** *ISO/IEC 20000: A Pocket Guide.* místo neznámé : Van Haren Publishing, 2006. ISBN 978 90 77212 79 0.

4. **Giese, Holger, Seibel, Andreas a Vogel, Thomas.** A Model-Driven Configuration Management System for Advanced IT Service Management. *CEUR Workshop Proceedings.* [Online] 21. 9 2009. [Citace: 4. 1 2014.] http://ceur-ws.org/Vol-509/paper_7.pdf.

5. **Shimpi, Anand Lal.** It Begins: AMD Announces Its First ARM Based Server SoC, 64-bit/8-core Opteron A1100. *AnandTech.* [Online] 28. 1 2014. [Citace: 13. 4 2014.] http://www.anandtech.com/show/7724/it-begins-amd-announces-its-first-arm-based-server-soc-64bit8core-opteron-a1100.

6. **Pingdom AB.** Linux now on 42% of consumer computing devices. *Pingdom.* [Online] Pingdom AB, 28. 12 2012. [Citace: 4. 5 2014.] http://royal.pingdom.com/2012/12/28/happy-birthday-linus-linux/.

7. **Gigaom, Inc.** AWS EC2 usage by operating system. *GIGAOM.* [Online] Gigaom, Inc., 11. 6 2013. [Citace: 4. 5 2014.] http://gigaom.com/2013/06/11/can-free-red-hat-on-aws-make-it-the-de-facto-linux-for-the-cloud-too/aws-ec2-usage-by-operating-system/.

8. **E-Soft Inc.** OS/Linux Distributions using Apache. *Security Space.* [Online] E-Soft Inc., 1. 12 2012. [Citace: 4. 5 2014.] https://secure1.securityspace.com/s_survey/data/man.201211/apacheos.html.

9. **Tsalolikhin, Alekscy.** Automating System Administration with Cfengine 3: An Introduction. *Vertical Sysadmin.* [Online] 21. 12 2009. [Citace: 4. 1 2014.] http://www.verticalsysadmin.com/cfengine3/.

10. **CFEngine AS.** CFEngine Quick Start Guide. *CFEngine Documentation Archive.* [Online] CFEngine AS. [Citace: 26. 4 2014.]

11. —. The History of CFEngine. *CFEngine.* [Online] CFEngine AS. [Citace: 1. 4 2014.] http://cfengine.com/the-history-of-cfengine.

12. —. CFEngine Community Edition. *CFEngine.* [Online] CFEngine AS. [Citace: 4. 1 2014.] http://cfengine.com/community.

13. —. CFEngine 3 Nova Owner's Manual. *CFEngine Documentation Archive.* [Online] 21. 10 2011. [Citace: 4. 1 2014.] https://cfengine.com/manuals_files/NovaOwnersManual.pdf.

14. —. Which CFEngine Edition is Right for Me? *CFEngine.* [Online] [Citace: 4. 1 2014.] http://cfengine.com/cfengine-comparison.

15. **Burgess, Mark, a další.** INSTALL. *Github.com.* [Online] 19. 11 2013. [Citace: 19. 1 2014.] https://github.com/cfengine/core/blob/master/INSTALL.

16. **CFEngine AS.** CFEngine 3 Linux Distributions. *CFEngine.* [Online] CFEngine AS. [Citace: 19. 1 2014.] http://cfengine.com/cfengine-linux-distros.

17. —. Supported Platforms and Versions. *CFEngine.* [Online] CFEngine AS. [Citace: 19. 1 2014.] https://cfengine.com/docs/3.5/getting-started-supported-platforms.html.

18. **Tsalolikhin, Alekscy.** Relative Origin of Cfengine, Puppet and Chef. *Vertical Sysadmin.* [Online] 15. 9 2010. [Citace: 18. 1 2014.] http://verticalsysadmin.com/blog/uncategorized/relative-origins-of-cfengine-chef-and-puppet.

19. **Puppet Labs, Inc.** Puppet Open Source. *Puppet Labs.* [Online] Puppet Labs, Inc. [Citace: 18. 1 2014.] http://puppetlabs.com/puppet/puppet-open-source.

20. —. Enterprise vs. Open Source - Which Is Right For Your Organization? *Puppet Labs.* [Online] Puppet Labs, Inc. [Citace: 18. 1 2014.] http://puppetlabs.com/puppet/enterprise-vs-open-source.

21. —. Support Plans. *Puppet Labs.* [Online] Puppet Labs, Inc. [Citace: 18. 1 2014.] http://puppetlabs.com/services/support-plans.

22. —. Puppet Open Source - Supported Platforms and System Requirements. *Puppet Labs.* [Online] Puppet Labs, Inc. [Citace: 18. 1 2014.] http://docs.puppetlabs.com/guides/platforms.html.

23. —. Puppet Enterprise - System Requirements and Pre-Installation. *Puppet Labs.* [Online] Puppet Labs, Inc. [Citace: 18. 1 2014.] http://docs.puppetlabs.com/pe/latest/install_system_requirements.html.

24. **Chef Software, Inc.** An Overview of Chef. *Chef Documents.* [Online] 2014. [Citace: 26. 4 2014.] http://docs.opscode.com/chef_overview.html.

25. —. Which Chef is Right for You? *Chef.* [Online] 2014. [Citace: 26. 4 2014.] http://www.getchef.com/chef/.

26. —. System Requirements. *Chef Documents.* [Online] [Citace: 26. 4 2014.] http://docs.opscode.com/chef_system_requirements.html.

27. **Ansible, Inc.** About Ansible. *Ansible Documentation.* [Online] Ansible, Inc., 18. 4 2014. [Citace: 27. 4 2014.] http://docs.ansible.com/.

28. **Karban, David.** Konfigurační a orchestrační nástroj Ansible. *Root.cz.* [Online] Internet Info, s.r.o., 27. 8 2013. [Citace: 27. 4 2014.] http://www.root.cz/clanky/konfiguracni-a-orchestracni-nastroj-ansible-uvod/. ISSN 1212-8309.

29. —. Ansible: konfigurace, fakta, role. *Root.cz.* [Online] Internet Info, s.r.o., 5. 9 2013. [Citace: 27. 4 2014.] http://www.root.cz/clanky/ansible-konfigurace-fakta-role/. ISSN 1212-8309.

30. **Ansible, Inc.** Ansible Pricing. *Ansible.* [Online] Ansible, Inc., 2014. [Citace: 27. 4 2014.] http://www.ansible.com/pricing.

31. —. Installation. *Ansible Documentation.* [Online] Ansible, Inc., 18. 4 2014. [Citace: 27. 4 2014.] http://docs.ansible.com/intro_installation.html.

32. **Microsoft Corporation.** System Center 2012 R2 Configuration Manager and Windows Intune Datasheet. *System Center 2012 R2 Configuration Manager.* [Online] 1. 10 2013. [Citace: 4. 1 2014.] http://download.microsoft.com/download/D/0/6/D0625D3D-0B8A-4CCF-B3FF-FF62F28C6D01/System_Center_2012_R2_Configuration_Manager_Datasheet.pdf.

33. **Godes, Kerry.** Say "Yes" to BYOD with Windows Intune. *Technet Blogs.* [Online] Microsoft Corporation, 30. 1 2013. [Citace: 4. 1 2014.] http://blogs.technet.com/b/openness/archive/2013/01/30/byod-windows-intune-systems-center.aspx.

34. **Toland, Ron.** Cfengine vs. Puppet vs. Chef. *SCALE 2013.* [Online] 21. 9 2013. [Citace: 2. 5 2014.] http://www.slideshare.net/mindbat/cfengine-vs-puppet-chef.

35. **Ansible, Inc.** Installation. *Ansible Docs.* [Online] 28. 4 2014. [Citace: 2. 5 2014.] http://docs.ansible.com/intro_installation.html.

36. **Bettag, Franz.** Cfengine, Bcfg2, Chef and Puppet. *Uberblogs.* [Online] 24. 6 2012. [Citace: 2. 5 2014.] http://uberblogs/2012/06/cfengine-bcfg2-chef-and-puppet.

37. **Venezia, Paul.** Review: Puppet vs. Chef vs. Ansible vs. Salt. *InfoWorld.* [Online] Infoworld, Inc., 21. 11 2013. [Citace: 2. 5 2014.] http://www.infoworld.com/d/data-center/review-puppet-vs-chef-vs-ansible-vs-salt-231308.

38. **CFEngine AS.** Frequently Asked Questions. *CFEngine.* [Online] CFEngine AS. [Citace: 2. 5 2014.] https://cfengine.com/techFaq.

39. **Puppet Labs, Inc.** Passenger. *Docs.* [Online] Puppet Labs, Inc., 2014. [Citace: 2. 5 2014.] http://docs.puppetlabs.com/guides/passenger.html.

40. **Chef Software, Inc.** About the chef-client. *Chef Documents.* [Online] Chef Software, Inc. [Citace: 2. 5 2014.] http://docs.opscode.com/essentials_chef_client.html.

41. **Quinn, Corey.** A Taste of Salt: Like Puppet, Except It Doesn't Suck. *SmartBear.* [Online] SmartBear Software, 25. 4 2013. [Citace: 2. 5 2014.] http://blog.smartbear.com/devops/a-taste-of-salt-like-puppet-except-it-doesnt-suck/.

42. **Stenberg, Eystein Maalocy.** Scalability of CFEngine 3.3.5 and Puppet 2.7.19. *Blogcompiler.* [Online] 30. 9 2012. [Citace: 2. 5 2014.] http://www.blogcompiler.com/2012/09/30/scalability-of-cfengine-and-puppet-2/.

43. **Burgess, Mark a Bergstra, Jan.** A static theory of promises. *Computing Research - Oslo University College.* [Online] 18. 10 2008. [Citace: 3. 5 2014.] http://project.iu.hio.no/papers/origin2.pdf.

44. —. Local and Global Trust Based on the Concept of Promises. *Computing Research - Oslo University College*. [Online] 20. 9 2006. [Citace: 3. 5 2014.] http://project.iu.hio.no/papers/trust.pdf.

45. **Burgess, Mark.** Introduction to promise theory. *Promise Theory*. [Online] [Citace: 3. 5 2014.] http://research.iu.hio.no/promises.php.

46. **CFEngine AS.** CFEngine 3 Concept Guide. *CFEngine Documentation Archive*. [Online] CFEngine AS. [Citace: 20. 4 2014.] https://cfengine.com/archive/manuals/cf3-conceptguide.

47. —. The CFEngine Components. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 24. 4 2014.] https://cfengine.com/docs/3.5/manuals-components.html.

48. —. CFEngine 3.5. *CFEngine Documentation*. [Online] CFEngine AS. [Citace: 15. 2 2014.] https://cfengine.com/docs/3.5/index.html.

49. —. Design. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 15. 2 2014.] https://cfengine.com/docs/3.5/manuals-design.html.

50. —. Language Concepts - Promises. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 15. 2 2014.] https://cfengine.com/docs/3.5/manuals-language-concepts-promises.html.

51. —. Language Concepts - Bundles. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 15. 2 2014.] https://cfengine.com/docs/3.5/manuals-language-concepts-bundles.html.

52. —. Language Concepts - Bodies. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 20. 4 2014.] https://cfengine.com/docs/3.5/manuals-language-concepts-bodies.html.

53. —. Language Concepts - Variables. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 19. 2 2014.] https://cfengine.com/docs/3.5/manuals-language-concepts-variables.html.

54. —. Functions - getindices. *CFEngine Reference*. [Online] CFEngine AS. [Citace: 10. 03 2014.] https://cfengine.com/docs/3.5/reference-functions-getindices.html.

55. —. Iteration in CFEngine. *CFEngine Documentation Archive*. [Online] CFEngine AS. [Citace: 10. 3 2014.] https://cfengine.com/archive/manuals/st-iterate.

56. —. Normal Ordering. *CFEngine Manuals*. [Online] CFEngine AS. [Citace: 10. 3 2014.] https://cfengine.com/docs/3.5/manuals-language-concepts-normal-ordering.html.

57. —. commands. *CFEngine Reference*. [Online] CFEngine AS. [Citace: 14. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-commands.html.

58. —. processes. *CFEngine Reference*. [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-processes.html.

59. —. services. *CFEngine Reference*. [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-services.html.

60. —. files. *CFEngine Reference*. [Online] CFEngine AS. [Citace: 22. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-files.html.

61. —. Promising and Editing File Content. *CFEngine Documentation Archive.* [Online] CFEngine AS. [Citace: 22. 4 2014.] https://cfengine.com/archive/manuals/st-editing.

62. —. packages. *CFEngine Reference.* [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-packages.html.

63. —. methods. *CFEngine Reference.* [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-methods.html.

64. —. reports. *CFEngine Reference.* [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/docs/3.5/reference-promise-types-reports.html.

65. —. CFEngine 3 Security. *CFEngine.* [Online] CFEngine AS. [Citace: 19. 4 2014.] https://cfengine.com/security.

66. —. CFEngine Architecture and Security. [Online] 7 2010. [Citace: 19. 4 2014.] http://cfengine.com/manuals_files/SpecialTopic_Security.pdf.

67. —. The Impact of Configuration Management on Security in IT Operations. [Online] 17. 4 2013. [Citace: 19. 4 2014.] http://info.cfengine.com/rs/cfengineab/images/20130417%20-%20Whitepaper%20-%20The%20Impact%20of%20CM%20on%20Security%20in%20IT%20Operations.pdf.

68. **Zamboni, Diego.** CFEngine Tip #004: How to Bootstrap a CFEngine Client. *News about "Learning CFEngine 3" book.* [Online] 25. 9 2012. [Citace: 15. 3 2014.] http://blog.cf-learn.info/cfengine-tip-004-how-to-bootstrap-a-cfengine/.

69. **CFEngine AS.** Version Control. *CFEngine Manuals.* [Online] CFEngine AS. [Citace: 9. 2 2014.] https://cfengine.com/docs/3.5/manuals-writing-policy-version-control.html.

70. **Matuszek, David.** Git A distibuted version control system. *Programming Languages & Techniques I.* [Online] 7. 11 2012. [Citace: 9. 2 2014.] http://www.cis.upenn.edu/~matuszek/cit591-2012/Lectures/git.ppt.

71. **CFEngine AS.** Design Center. *CFEngine Manuals.* [Online] CFEngine AS. [Citace: 7. 1 2014.] https://cfengine.com/docs/3.5/manuals-design-center.html.

72. —. Sketch Structure. *CFEngine Reference.* [Online] CFEngine AS. [Citace: 7. 1 2014.] https://cfengine.com/docs/3.5/reference-design-center-sketch-structure.html.

73. —. Monitoring and Reporting: A CFEngine Special Topics Handbook. *CFEngine Manuals.* [Online] 11. 12 2012. [Citace: 5. 4 2014.] http://cfengine.com/manuals_files/SpecialTopic_Reporting.pdf.

74. —. Reporting Architecture. *CFEngine Manuals.* [Online] [Citace: 5. 4 2014.] https://cfengine.com/docs/3.5/manuals-enterprise-reporting-architecture.html.

75. **Raspberry Pi Foundation.** FAQs. *Raspberry Pi.* [Online] [Citace: 13. 4 2014.] http://www.raspberrypi.org/help/faqs/.

76. **CESNET, z.s.p.o.** MetaCentrum NGI. *MetaCentrum.* [Online] CESNET, z.s.p.o. [Citace: 2. 2 2014.] http://www.metacentrum.cz/cs/.

77. **Brennen, V. Alex.** Kerberos Infrastructure HOWTO. *The Linux Documentation Project.* [Online] 29. 05 2004. [Citace: 10. 03 2014.] http://tldp.org/HOWTO/Kerberos-Infrastructure-HOWTO/time-sync.html.

78. **University of Northern Iowa.** Security - Use a Firewall. *Information Technology Services.* [Online] University of Northern Iowa. [Citace: 8. 4 2014.] http://www.uni.edu/its/support/article/717.

79. **Pinheiro, Eduardo, Weber, Wolf-Dietrich a Barroso, Luiz André.** Google Research. *Failure Trends in a Large Disk Drive Population.* [Online] 19. 12 2006. [Citace: 12. 4 2014.] http://static.googleusercontent.com/media/research.google.com/cs//archive/disk_failures.pdf.

80. **Agarwal, Yuvraj a Hall, Malcolm.** ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing. *University of California.* [Online] 22. 4 2013. [Citace: 4. 1 2014.] http://www.cs.cmu.edu/~yuvraja/docs/Agarwal_MobiSys2013_ProtectMyPrivacy.pdf.

81. **Stanford University.** Mobile Device Management - Android Rooting Information. *Stanford University IT Services.* [Online] Stanford University, 8. 8 2013. [Citace: 1. 4 2014.] https://itservices.stanford.edu/service/mobiledevice/management/manage_android/root.

7 Přílohy

Příloha 1: Modul aktualizace z GIT repozitáře

```
#!/bin/bash

#
#  Update z GITu a overeni navratove hodnoty - Modul pro CFEngine3
#
#  Marek Petko
#

# Cesta k vetvim
branches="/var/cfengine/masterfiles_branches/"

if [[ $# < 1 ]]
   then
      echo "Chyba: Nebyl zadan parametr - nazev vetve."
      exit 1
fi

cd $branches$1

gitout=`git pull 2>/dev/null`
RET=$?

if [ $RET -ne 0 ] ; then
   echo "+brach_$1_git_pull_error"
   exit 1
fi

if [[ $gitout =~ ^.*up-to-date.*$ ]]
   then
      echo "-branch_$1_has_changed"
   else
      echo "+branch_$1_has_changed"
fi

cd ..
exit 0
```

Příloha 2: Bundle fixresolv

```
bundle edit_line fixresolv
{
   vars:
     "primary_nameserver_count"
       int => countlinesmatching("nameserver\s+\Q$(resolver.primary_nameserver)\E\s*",
                                 $(resolver.resolv_conf));

     "propper_search_count"
        int => countlinesmatching("search\s+$(resolver.allowed_search)\s*",
                                  $(resolver.resolv_conf));

   classes:
      "primary_nameserver_found"
         expression => isgreaterthan($(primary_nameserver_count),0);

      "primary_nameserver_too_many_entries"
         expression => isgreaterthan($(primary_nameserver_count),1);

      "propper_search_found"
         expression => isgreaterthan($(propper_search_count),0);

      "propper_search_too_many_entries"
         expression => isgreaterthan($(propper_search_count),1);

   delete_lines:
      ".*nameserver.*"
         delete_select => delete_nameservers;

      ".*search.*"
         delete_select => delete_search;

   insert_lines:
      !primary_nameserver_found::
         "nameserver $(resolver.primary_nameserver)"
            location => after_private;

      !propper_search_found::
         "search $(resolver.default_search)";

   reports:
      primary_nameserver_too_many_entries::
        "Nalezen duplicitni zaznam primarniho nameserveru v $(resolver.resolv_conf)!";

      propper_search_too_many_entries::
        "Nalezen duplicitni zaznam search v $(resolver.resolv_conf)!";
}

body location after_private
{
   select_line_matching =>
   "\s*(search.*)|(nameserver\s+127\.0\.0\.1)|(nameserver\s+10\.\d{1,3}\.\d{1,3}\.\d{1,3})\s*";

   before_after => "after";
   first_last => "last";
}

body delete_select delete_nameservers
{
   delete_if_not_match_from_list => {@(resolver.allowed_nameservers_lines)};
}

body delete_select delete_search
{
   delete_if_not_match_from_list => {@(resolver.allowed_search_lines)};
}
```

Příloha 3: Produkční nasazení

Seznam všech produkčních počítačů spravovaných CFEngine k 1. 5. 2014.

```
allycon-tst.zcu.cz
allycondb-tst.zcu.cz
apate-ch.fek.zcu.cz
apate-hj.zcu.cz
cicomexocitl.civ.zcu.cz
cloud-001.civ.zcu.cz
cloud-101.civ.zcu.cz
collect.civ.zcu.cz
daidalos.civ.zcu.cz
daphne.civ.zcu.cz
daphne2.civ.zcu.cz
dbavu.zcu.cz
dbmvso.zcu.cz
dbvoscb.zcu.cz
dbvsss.zcu.cz
gate-vpn1.zcu.cz
gate-vpn2.zcu.cz
gmontest.zcu.cz
hc.civ.zcu.cz
horus.civ.zcu.cz
kraken.zcu.cz
laila.civ.zcu.cz
lamia.zcu.cz
listek.civ.zcu.cz
listik.zcu.cz
lm64.zcu.cz
metalist.civ.zcu.cz
midas.civ.zcu.cz
netman2.civ.zcu.cz
pdev.civ.zcu.cz
phix-test.zcu.cz
phix.zcu.cz
polyxen.civ.zcu.cz
procesy.zcu.cz
pythia.zcu.cz
xen1.civ.zcu.cz
```