CFEngine3 prakticky

Michal Švamberg
svamberg@civ.zcu.cz

OpenAlt 5.11.2016, Brno

https://goo.gl/tjHoUd

Obsah

- Konfigurační management CFEngine3
- Instalace a první kontakt
- Příklady na začátek
- Příklady pro pokročilé
- Pro náročné
- Cokoliv na přání

Diskuse po celou dobu, dotazy hned

https://goo.gl/tjHoUd

CFEngine3

- Jak to funguje
- Dokumentace
- Úvod do jazyka
- Základní pojmy

Programy pro CFEngine

Důležité:

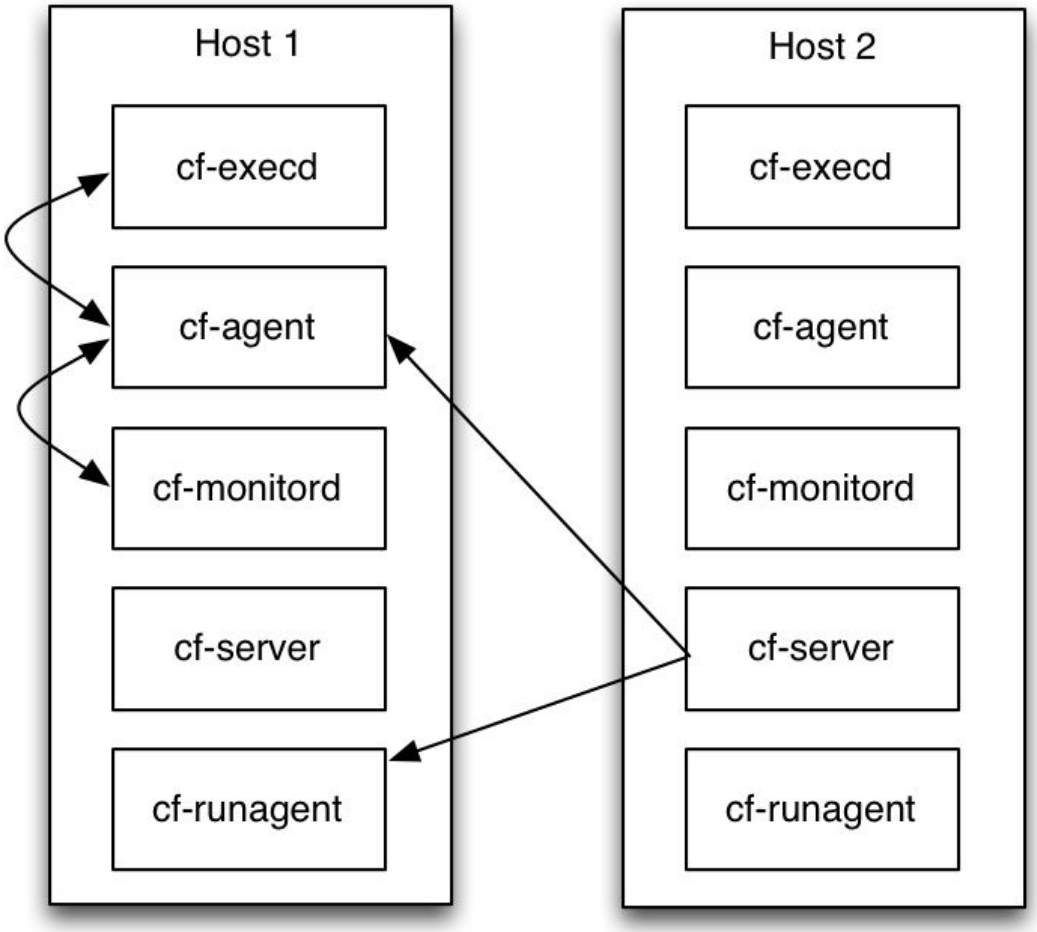
- **cf-promises** - kontrola syntaxe
- **cf-agent** - provedení slibů, vždy se napřed pustí cf-promises
- cf-execd - daemon, který časuje cf-agent (pravidelně jej pouští)
- cf-monitord - poskytuje informace ze systému včetně lehkého monitoringu
- cf-serverd - souborový server pro kopírování

Další: cf-key, cf-runagent, cf-twin, cf-upgrade, ...

Komponenty

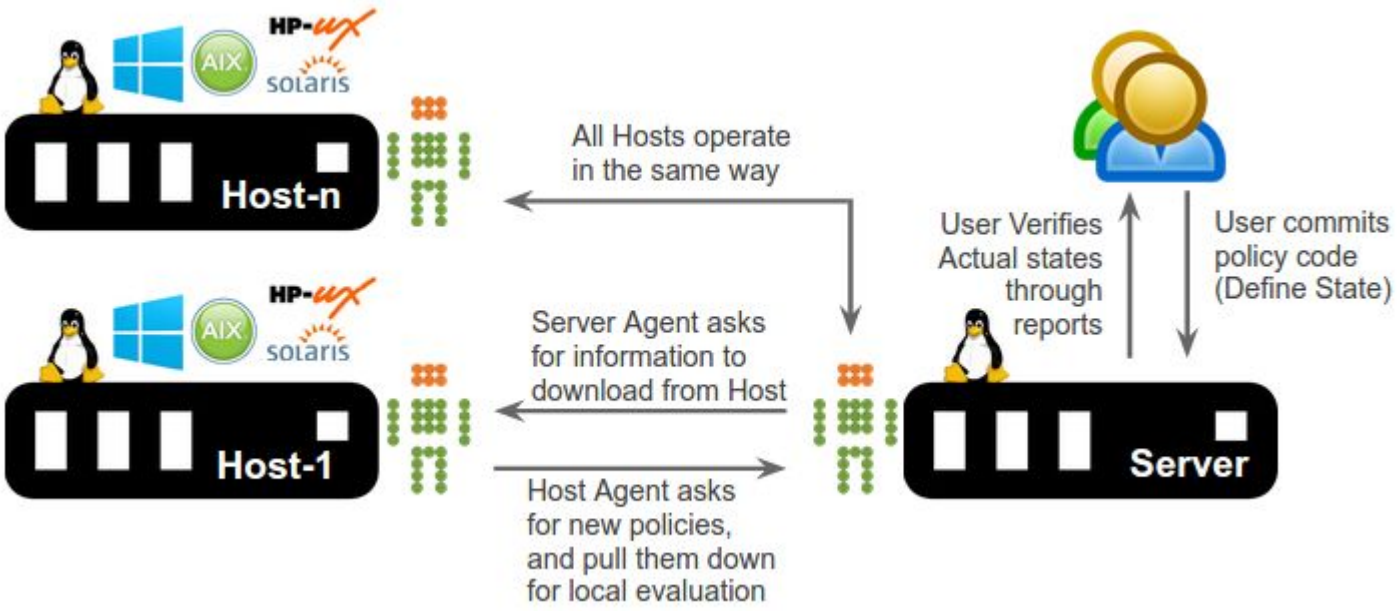
Není rozdíl mezi serverem a klientem, klient může být zároveň serverem.

Policy hub je klient, jehož veřejná (nikoliv loopback) IP adresa se shoduje s IP adresou CFEngine policy serveru (server ze kterého se distribuují pravidla).



Zdroj: cfengine.com

Workflows



Zdroj: cfengine.com

Dokumentace

Referenční dokumentace: https://docs.cfengine.com/lts/reference.html

Jiné architektury (Android, RPi, …) a nástroje: http://www.cfengineers.net/

Emailová skupina: https://groups.google.com/forum/#!forum/help-cfengine

Barvičky pro Vim: https://github.com/neilhwatson/vim_cf3 (čti doc/cf3.txt)

Slib

```
files:
 debian::
  "/etc/default/lbcd"
   edit_line =>
    replace_or_add(
     "^\s*USER=",
     "USER=\"lbcd\""),
    comment =>
     'Oprava chybky init skriptu.',
    classes =>
     if_notkept(
      "edit_fail");
```

Typ slibu - následující sliby se týkají souborů.
Následující sliby jen pokud existuje třída debian.
Ten komu slibuji (promiser), název slibu (zde soubor).
První atribut - budeme dělat řádkové úpravy.
Bundle, který provede úpravy.
První parametr bundle pro řádkovou úpravu.
Druhy parametr bundle pro řádkovou úpravu.
Druhý atribut slibu - komentář.
Prostý parametr atributu, řetězec jako text komentáře.
Třetí atribut - definuj třídu.
Body, nastaví třídu pokud slib není zkontrolován.
Název třídy (parametr body).

Promises & Bundles

Já, cf-agent slibuji, že do proměnné (vars:) s názvem sites vložím seznam (slist) jmen souboru, které dodá funkce lsdir() z adresáře /etc/apache2/sites-available odpovídají regulárnímu výrazu ^[^\.].* včetně plné cesty (true). Já, cf-agent slibuji, že vytisknu (reports:) všechny hodnoty proměnné sites.

```
bundle agent apache {
vars:
  "sites"
    slist => lsdir("/etc/apache2/sites-available", "^[^\.].*", "true"),
    comment => "Soubory vyjma tech zacinajici teckou + '.' a '..'" ;
reports:
  "Nalezen soubor $(sites)";
}
```

Typy slibů a jejich pořadí

- bundle se provádí 3x, opakovaně
- postupně podle typu slibu
- stejné typy slibů pak dle pořadí
- snaha napřed mazat a pak tvořit
- sliby se podle typu liší svými možnostmi a množstvím použitelných funkcí
- jedná se o deklarativní jazyk, tzn. na pořadí slibů (příkazů) nezáleží
- slibem definujete jak má vypadat výsledek, nikoliv jak se to má udělat, to už je věc CF3
- snažte se držet data odděleně od kódu

bundle agent:
meta
vars
defaults
classes
users
files
packages
guest_environments
methods
processes
services
commands
storage
databases
reports

bundle edit_line:
meta
vars
defaults
classes
delete_lines
field_edits
insert_lines
replace_patterns
reports

Body - skupina atributů

Skupina atributů slibu, tak aby se nemusely stále vypisovat. Mohou a nemusejí mít parametry. Nejlepší materiál v masterfiles: /var/cfengine/masterfiles/lib/

```
files:
 "/usr/local/bin/l"
  copy_from =>
   backup_local_dcp(
    "$(zcu.dir_templates)/usr/local/bin/l"),
  comment => "misto aliasu ls -l vlastni skript",
  create => "true",
  perms => mog("0755", "root", "root");
```

```
body perms mog(m,u,g){
 mode => { "$(m)" };
 owners => { "$(u)" };
 groups => { "$(g)" };
}
```

Třídy

Jsou globální (definované v `bundle common` nebo v `body classes`) a je nutné s tím počítat při pojmenování. Ostatní třídy, nejčastěji vzniklé v `classes:` jsou lokální a po opuštění bundle zmizí.

Třídy je možné kombinovat logickými operátory:

```
reports:
 !(Saturday|Sunday).Night::
   "Je pracovní den a noc, v továrně by se mělo svítit.";
```

Třídy lze mít i dynamické na základě proměnných, ale o tom později.

Pokud není třída udána, předpokládá se třída `any`.

Proměnné

Jsou vždy vázány na konkrétní bundle (neexistuje vlastně globální proměnná) lze se na jakoukoliv dotázat přes jméno bundle. Existuje celá řada předdefinovaných proměnných:

```
sys.host
this.bundle
mon.listening_ports
```

Proměnná může být různého typu:

- skalární (`string`, `int`, `real`)
- seznam skalárních proměnných (`slist`, `ilist`, `rlist`)
- datový kontejner (`data`), umí pracovat s JSON, YAML a CSV formátem
- asociativni pole, nyní zastaralé, má se použít datový kontejner

Implicitní cyklus

Pokud se má zpracovat proměnná typu seznam, pak se operace provede automaticky nad všemy prvky seznamu. Pokud se současně zpracovává více seznamů, pak to budou navíc všechny jejich kombinace:

```
bundle agent test {
vars:
 "x" slist => { "a", "b" };
 "y" slist => { "1", "2" };
reports:
 "Hodnota x=$(x) a y=$(y)";
}
```

```
# cf-agent -Kf ./tst.cf -b test
R: Hodnota x=a a y=1
R: Hodnota x=b a y=1
R: Hodnota x=a a y=2
R: Hodnota x=b a y=2
```

Startujeme

- Instalovat nejlépe z balíčku od CFEngine
- Základní ovládání (cf-promises, cf-agent)
- Inicializace a struktura masterfiles
- Životní cyklus (failsafe, update, promises)

Instalace

Nejlépe LTS z balíčku na https://cfengine.com/product/community/

Debian verzi (balík cfengine3) začátečníkům nedoporučuji:

- jinak cesty: `/var/cfengine` vs. `/var/lib/cfengine`
- úvodní konfigurace není správně odladěná
- potíže s prvotní inicializací
- rozdílné proměnné v `/etc/default/cfengine3`

Verze 3.7.x a novější jsou na tom mnohem lépe.

```
# dpkg -i cfengine-community_3.7.4-1_amd64.deb
```

Bootstrap

Vytvoříme si policy hub (cfengine server), tím se automaticky spustí i daemoni:

```
# ip addr | grep global | awk '{print $2}'
147.228.1.200/24
# cf-agent -B 192.168.1.1
R: Bootstrapping from host '192.168.1.211' via built-in policy
'/var/cfengine/inputs/failsafe.cf'
R: This host assumes the role of policy server
R: Updated local policy from policy server
R: Started the server
R: Started the scheduler
  notice: Bootstrap to '192.168.1.211' completed successfully!
# ps xfa | grep bin/cf | grep -v grep
12570 ?      Ss   0:00 /var/cfengine/bin/cf-execd
12576 ?      Ss   0:00 /var/cfengine/bin/cf-serverd
12587 ?      Ss   0:00 /var/cfengine/bin/cf-monitord
```

Co se vlastně stalo?

`/var/cfengine/masterfiles` - politika, která se dodává klientům

`/var/cfengine/inputs` - politika, kterou se řídí cf-* programy

`/var/cfengine/outputs` - výstupní logy s chybami/reporty

Hned si něco můžeme zkusit:

```
# cd /var/cfengine/inputs
# cf-agent -KIf ./update.cf
# cf-promises --show-classes
# cf-promises --show-vars
# service cfengine3 stop
# cf-agent -KIf ./update.cf
    info: Executing 'no timeout' ... '"/var/cfengine/bin/cf-execd"'
    info: Command related to promiser ...
    ...
```

Struktura `masterfiles/`

`controls/` - konfigurace daemonu, nastavení emailu, inicializační volby

`inventory/` - prohlídka hw a sw vybavení (vznik tříd a proměnných)

`lib/` - předpřipravené bundly a body, které se opravdu hodí, organizováno podle typu slibů

`services/` - vlastní politiky, které chceme provádět (výkonný kód)

`templates/` - data pro vlastní politiky (oddělení dat od kódu)

`sketches/` - pro použití s cf-sketch (vytváří politiku na základě interakce s administrátorem, není nutná znalost CFEngine jazyka), lze použít pro Design Center (webovém rozhraní).

Tři základní soubory

`failsafe.cf` - použije se, pokud selže zpravování, např. při chybě syntaxe nebo při bootstrapu. Kód je součástí kódu `cf-agent` a je to jednoduchý `update.cf`.

`update.cf` - aktualizuje `/var/cfengine/inputs` z policy serveru

`promises.cf` - vstupní soubor pro programy (cf-agent, cf-serverd, …), lze změnit parametrem. Zde je uložena politika ve formě slibů, která se bude provádět.

cf-promises

`-c` - full check
`-f file` - vstupní soubor
`--show-classes` - zobrazí třídy
`--show-vars` - zobrazí proměnné
`-b bundle` - vstupní bod, def. `bundlesequence` v `body common control`

Typické použití:

```
cf-promises -cf ./promises.cf
```

cf-agent

`-K` - uvolní zámek, který chrání před spuštěním každou minutu
`-B` - bootstrap, inicializace klienta
`-I` - informativní výpisy o změnách, voláních, kopírování, ...
`-f file` - vstupní soubor
`-b bundle` - vstupní bod, def. `bundlesequence` v `body common control`
`-v` - verbose (dost detailní, pro použití to chce trochu praxe)
`-d` - debug (pro odvážné, výpisy až na úroveň syntaktické analýzy)
`-D class` - nadefinuje třídu, např. `DEBUG` pro vlastní reporty

Typické použití:

```
cf-agent -KIf ./promises.cf
```

Rychlokurz

- Můj první slib aneb něco to dělá
- Můj druhý bundle
- Kopírujeme soubor

Hello CFEngine

1. **Vypište text obsahující hostname vašeho stroje.**
   `cf-promises --show-vars`
   nezapomeň, že formát zápisu proměnné je `$(proměnná)`
   `cf-agent -Kf ./hello.cf -b <bundle>`
2. **Přidejte text, který je zavislý na jménu operačního systému.**
   `cf-promises --show-classes | grep -i <jmeno_os>`
3. **Přidejte text, který se bude zobrazovat jen při nastavené třídě DEBUG**
   `cf-agent -Kf ./hello.cf -b <bundle> -D DEBUG`

Kolik paměti?

1. **Vypiš volnou paměť z `/proc/meminfo`**
   Dobře se to děla funkcí `getfileds()`, viz dokumentace
2. **Přidej informaci o volnosti na swapu**
3. **Vypiš info, zda je více prostoru na swapu nebo v paměti**
   Lze použít `isgreaterthan()` do atributu `if` (nebo `ifvarclass`)
4. **Místo atributu `if` použij vlastní vytvořenou třídu a její negaci.**
   Při vytváření třídy se použije atribut `expression`
5. **Použij jen jeden slib typu report pro zobrazení jestli je více místa na swapu nebo v paměti.**
   ```
   "Vice volne pameti ma $(mem_type)";
   ```
   Po vyhodnocení, kdo je větší, je třeba naplnit `mem_type`, tj. při druhém průchodu cfenginu bundlem. Použij třídu z bodu 4 ve `vars:`

Kopírování

1. **Okopíruj soubor (např. `/etc/motd`) do /tmp/**
   budeš potřebovat vlastní `body copy_from mojecp` které použij v atributu `copy_from`, inspirovat se můžeš v `lib/3.7/files.cf` u `body copy_from local_cp`
2. **Nastav souboru vlastníka `root:nobody` a práva `640`**
   Doplň atribut `perms` a `body perms prava` s předáním parametrů
   Lze opět použít `body perms mog` v souboru `lib/3.7/files.cf`
3. **Smaž cílový soubor /tmp/motd a spusť agenta**
   Použij `-I` aby jsi viděl co dělá.
4. **Změň oprávnění a znova spusť agenta, sleduj co píše**
   Zkus si i parametr `-v` (verbose) nebo `-d` (debug)

Pokročilí

- Autorun
- Šablony

Autorun

1. **Uprav `control/3.7/def.cf` dle "patche":**
   ```
   -       "services_autorun" expression => "!any";
   +       "services_autorun" expression => "any";
   ```
2. **Prohlídni si blok slibů v `bundle common services_autorun` počínaje řádkem 267 v `promises.cf` a odkazovaný `lib/3.7/autorun.cf`**
   Zkus pochopit, jak je autorun realizovaný.
3. **Otestuj, že funguje funguje `services/autorun/hello.cf`**
   Jaký parametr musíš přidat k `cf-agent` aby jsi viděl výpis?
4. **Přidej `services/autorun/openalt.cf` s výpisem a otestuj ji.**
   Slib v meta: je důležitý, právě přes něj se přilepí autorun.
5. **Aktualizuj politiku, pokud se tak již nestalo :-)**
6. **Spusť `cf-agent` bez parametru `-f` a ověř že se provedl `openalt.cf`**

Šablony

Vytvoř jednoduchou šablonu a vytiskni do ní nějakou proměnnou.

Jsou 2 typy šablon (standardní a mustache na dalším slidu). Použíje se atribut `edit_template => "soubor"`

V šabloně je třeba zapisovat proměnné plným jménem. Zkuste si také podmíněný výstup. Reálný kousek ze šablony pro sources.list:

```
[%CFEngine !(debian_6|debian_stretch):: %]
deb $(debian_apt_sources.mirror_site) \
$(debian_apt_sources.codename[stable])-backports main contrib non-free

[%CFEngine debian_6:: %]
deb http://http.debian.net/debian/ \
$(debian_apt_sources.codename[stable])-lts main contrib non-free
```

Na každém stroji vznikne soubor specifický dle používané verze Debianu.

Mustache

1. **prohlídni si `templates/host_info_report.mustache`**
2. **porovnej s `cfe_internal/core/host_info_report.cf`**
3. **spusť agenta a zkontroluj výstup**
   kde je výstup?
   `$(host_info_report.host_info_report_output)`

Mustache šablony mají povinný atribut `template_method => "mustache"`

Mají více možností zpracování (seznamy lze tisknout jako cyklus, …). Stejně jako u standardních šablon je nutné použít plné jméno proměnných (tj. včetně názvu bundle).

Co se nevešlo

- Editace souborů
- JSON jako zdroj pro řízení běhu CF3
- JSON jako zdroj dat pro šablony
- Použití klíčů a vzájemná autentizace
- Instalace balíků (od 3.7 nový způsob)
- Kopírování souborů ze serveru
- Oprávnění pro přístup k cf-serverd
- Rozdíly mezi komerční a community verzí
- Centrální sběr dat z agentů
- Frameworky (ncf, rudder, design-center)
- Sketche
- Inventory
- Procesy
- Uživatelé
- Příkazy
- ...

Přání / dotazy

Vymysli si zadání a zkusíme s tím pohnout...

… nejdéle však do oběda

Kdyby nebyly dotazy

- `antivir/xorddos.cf`
- `zen1# cf-agent -Kf ./promises.cf -D SERVICE_bacula,DEBUG_bacula_fileset`
- úprava `/etc/hosts.allow`
- kontrola, že stroj je monitorován a jsou monitorovány jeho služby
- inventory - virtualizace, fibrechannel
- git - větve stable/testing/production a jejich navázání na cfengine

Děkuji a přeji pevné nervy :-)