Kurs Puppeta 4

Łukasz Koszyk

Gigaset, Wrocław, Kompilacja: 24 maja 2017, 12:35

Spis treści

I Wstęp 5

II Listy zadań 7

1. Lista 1 **8**
- 1.1. Potrzebne narzędzia . . . 8
- 1.2. Plan działania . . . 8
  - 1.2.1. Obsługa vagranta . . . 8
  - 1.2.2. Puppet apply . . . 8
  - 1.2.3. Co to jest site.pp . . . 8
  - 1.2.4. Pierwsze zmiany - tworzenie plików i katalogów . . . 8
- 1.3. Ćwiczenia do samodzielnego wykonania . . . 9

2. Lista 2 **10**
- 2.1. Plan działania . . . 10
  - 2.1.1. Domyślne typy zdefiniowane . . . 10
  - 2.1.2. Kolizje nazw w typach zdefiniowanych . . . 10
  - 2.1.3. Kolejność wykonywania kodu . . . 10
  - 2.1.4. Cykle w zależnościach . . . 10
- 2.2. Ćwiczenia na zajęcia . . . 10
  - 2.2.1. Popraw błąd w kodzie . . . 10
  - 2.2.2. Kolejność . . . 11
- 2.3. Ćwiczenia do samodzielnego wykonania . . . 11

3. Lista 3 **12**
- 3.1. Plan działania . . . 12
  - 3.1.1. Odwołania do obiektów . . . 12
  - 3.1.2. Kolejność z użyciem metaparametrów . . . 12
  - 3.1.3. Kolejność z użyciem odwołań do obiektów . . . 12
- 3.2. Ćwiczenia na zajęcia . . . 12
  - 3.2.1. Napraw błędy . . . 12
  - 3.2.2. Zadanie . . . 13
- 3.3. Ćwiczenia do samodzielnego wykonania . . . 13

4. Lista 4 **15**
- 4.1. Plan działania . . . 15
  - 4.1.1. Zmienne . . . 15
  - 4.1.2. Typy zmiennych cz. 1 . . . 15
- 4.2. Ćwiczenia na zajęcia . . . 15
- 4.3. Ćwiczenia do samodzielnego wykonania . . . 15

5. Lista 5 **16**
- 5.1. Plan działania . . . 16
  - 5.1.1. Typy zmiennych cz. 2 . . . 16
  - 5.1.2. Proteza printf . . . 16
- 5.2. Ćwiczenia na zajęcia . . . 16
- 5.3. Ćwiczenia do samodzielnego wykonania . . . 16

6. Lista 6 **19**
6.1. Plan działania . . . 19
6.1.1. Facter . . . 19
6.1.2. Własne fakty . . . 19
6.2. Ćwiczenia na zajęcia . . . 19
6.3. Ćwiczenia do samodzielnego wykonania . . . 20

7. Lista 7 **21**
7.1. Plan działania . . . 21
7.1.1. Instrukcje warunkowe . . . 21
7.1.2. Operatory i wyrażenia . . . 21
7.2. Ćwiczenia na zajęcia . . . 21
7.3. Ćwiczenia do samodzielnego wykonania . . . 21

8. Lista 8 **22**
8.1. Plan działania . . . 22
8.1.1. Pętle . . . 22
8.2. Ćwiczenia na zajęcia . . . 22
8.3. Ćwiczenia do samodzielnego wykonania . . . 22

9. Lista 9 **24**
9.1. Plan działania . . . 24
9.1.1. Pętle - ciąg dalszy . . . 24
9.1.2. Lambdy . . . 24
9.2. Ćwiczenia na zajęcia . . . 24
9.3. Ćwiczenia do samodzielnego wykonania . . . 24

10. Lista 10 **44**
10.1. Plan działania . . . 44
10.1.1. Deduplikacja kodu - pierwsze podejście . . . 44
10.1.2. Zastrzeżone wyrażenia . . . 44
10.1.3. Komentarze w kodzie . . . 44
10.1.4. Więcej niż jedna maszyna . . . 44
10.2. Ćwiczenia na zajęcia . . . 44
10.3. Ćwiczenia do samodzielnego wykonania . . . 44

11. Lista 11 **46**
11.1. Plan działania . . . 46
11.1.1. Własne klasy - wprowadzenie . . . 46
11.1.2. Jak nazwać klasę i gdzie jej szukać . . . 46
11.1.3. A gdzie szukać klasy która nie ma dwukropków? . . . 46
11.1.4. Jak uruchomić swoją klasę . . . 46
11.1.5. Moja klasa ma parametry nagłówkowe lub nie lubię słowa include . . . 46
11.1.6. Czy z poziomu mojej klasy mogę uruchomić inną moją klasę? . . . 47
11.1.7. Dokumentacja . . . 47
11.2. Ćwiczenia na zajęcia . . . 47
11.3. Ćwiczenia do samodzielnego wykonania . . . 47

12. Lista 12 **48**
12.1. Plan działania . . . 48
12.1.1. Własne klasy - ciąg dalszy . . . 48
12.1.2. Dziedziczenie parametrów . . . 48
12.1.3. Dziedziczenie po wielu klasach . . . 48
12.1.4. A co jeśli zmienne będą niedostępne? . . . 48
12.1.5. A co jeśli ktoś w klasie nadrzędnej lub potomnej nadpisze zmienną? . . . 48
12.1.6. Czy można jeszcze jakieś cuda z tymi zmiennymi wyprawiać? . . . 48
12.1.7. Wymuszanie kolejności wykonania kodu na klasach . . . 49

12.1.8. Kolejność wykonywania w puppecie - ostatnia metoda 49
12.1.9. Kilka słów o standardach w kodzie 50
12.1.10. Dokumentacja 50
12.2. Ćwiczenia na zajęcia 50
12.3. Ćwiczenia do samodzielnego wykonania 50

13. Lista 13 **51**
13.1. Plan działania 51
13.1.1. Więcej o typach zdefiniowanych 51
13.1.2. Własny typ zdefiniowany 51
13.1.3. Różnice między klasą a typem 51
13.1.4. Wymuszanie kolejności na własnych typach zdefiniowanych 51
13.1.5. Kolejne kilka słów o standardach w kodzie 52
13.1.6. Typ zdefiniowany a iteracja 52
13.1.7. Dokumentacja 52
13.2. Ćwiczenia na zajęcia 52
13.3. Ćwiczenia do samodzielnego wykonania 52

14. Lista 14 **53**
14.1. Plan działania 53
14.1.1. Dodawanie plików 53
14.1.2. Co do czego 54
14.1.3. Sztuczki 54
14.2. Ćwiczenia na zajęcia 54
14.3. Ćwiczenia do samodzielnego wykonania 54

15. Lista 15 **56**
15.1. Plan działania 56
15.1.1. Hiera 56
15.1.2. Konfiguracja hiery 56
15.1.3. Załączanie klasy w hierze 56
15.1.4. Zmienne w hierze 56
15.1.5. O hierarchii słów kilka 57
15.1.6. Własne moduły 57
15.1.7. Dobre praktyki przy pisaniu modułów 57
15.2. Ćwiczenia na zajęcia 58
15.3. Ćwiczenia do samodzielnego wykonania 58

16. Lista 16 **59**
16.1. Plan działania 59
16.1.1. Własne funkcje w rubym 59
16.1.2. O faktach puppetowych raz jeszcze 59
16.1.3. Cudze moduły puppetowe 59
16.1.4. Zmiana metody dostarczania kodu 59
16.2. Ćwiczenia na zajęcia 59
16.3. Ćwiczenia do samodzielnego wykonania 59

17. Lista 17 **60**
17.1. Plan działania 60
17.1.1. Powrót do dziedziczenia parametrów - klasy params.pp 60
17.1.2. Struktura kodu puppetowego 60
17.2. Lint 60
17.3. Ćwiczenia na zajęcia 60
17.4. Ćwiczenia do samodzielnego wykonania 62

18.Lista 18 **63**
18.1. Plan działania . . . 63
18.1.1. Kod puppetowy pod wiele systemów . . . 63
18.1.2. Jak to zrobić? . . . 63
18.1.3. A skąd puppet ma wiedzieć jaki to system operacyjny? . . . 63
18.1.4. Wielomaszynowy vagrant . . . 63
18.2. Ćwiczenia na zajęcia . . . 63
18.3. Ćwiczenia do samodzielnego wykonania . . . 63

19.Lista 19 **64**
19.1. Plan działania . . . 64
19.1.1. Filebucket . . . 64
19.2. Ćwiczenia na zajęcia . . . 64
19.3. Ćwiczenia do samodzielnego wykonania . . . 64

20.Lista 20 **65**
20.1. Plan działania . . . 65
20.1.1. Pitu pitu o puppecie . . . 65
20.1.2. Jak pisać moduły . . . 65
20.1.3. Jak czytać moduły . . . 65
20.2. Ćwiczenia na zajęcia . . . 66
20.3. Ćwiczenia do samodzielnego wykonania . . . 66

21.Lista 21 **67**
21.1. Plan działania . . . 67
21.1.1. Narzędzia do puppeta . . . 67
21.1.2. Sztuczki puppetowe . . . 67
21.1.3. Jak automatyzować puppetem . . . 68
21.1.4. Testowanie kodu puppetowego . . . 68
21.2. Ćwiczenia na zajęcia . . . 68
21.3. Ćwiczenia do samodzielnego wykonania . . . 68

Część I
Wstęp

Niniejszy poradnik to zbiór list zadań pomocnych przy nauce Puppeta. Nie podejmuję tutaj tematu: dlaczego akurat Puppet, po co mi Puppet itd. Kwestię motywacji pozostawiam czytelnikowi.

Poradnik nie ma charakteru rozprawy naukowej, jest w zasadzie zbiorem różnych porad i luźnych spostrzeżeń, do jakich doszliśmy w dziale operacyjnym w Gigasecie. Może także zawierać drobne błędy. W celu ułatwienia zapoznania się z narzędziem podzielony jest na listy zadań.

Minimalne wymagania aby przejść ten kurs, to motywacja do nauki, znajomość Basha, Linuksa, podstawowych Linuksowych usług.

Potrzebny będzie także vagrant (podstawowy centos 7) z zainstalowanym puppet agentem (poradnik był tworzony przy użyciu wersji 4.10) oraz skrypt bootstrap:

```
1  #!/bin/bash
2
3  echo "sudo su -" >> /home/vagrant/.bashrc
4  echo "exit" >> /home/vagrant/.bashrc
5
6  echo "alias puppet-apply='puppet apply --hiera_config=/vagrant/hieradata/
       hiera.yaml --modulepath=/vagrant/modules --verbose /vagrant/manifests/
       site.pp --show_diff'" >> /root/.bashrc
7  echo "alias puppet-apply-noop='puppet apply --hiera_config=/vagrant/
       hieradata/hiera.yaml --modulepath=/vagrant/modules --verbose /vagrant/
       manifests/site.pp --show_diff --noop'" >> /root/.bashrc
8  echo "alias puppet-check='puppet-lint --no-80chars-check /vagrant; echo;
       echo; echo; find /vagrant/hiera* -type f -name \"*.yaml\" | xargs yaml-
       lint; echo '" >> /root/.bashrc
9
10 rm -rf /etc/puppetlabs/code/environments/production
11 ln -s /vagrant /etc/puppetlabs/code/environments/production
```

Listing 1: bootstrap.sh

oraz przykładowy Vagrantfile:

```
1  VAGRANTFILE_API_VERSION = "2"
2  Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
3      config.vm.define "host_centos" do |host_centos|
4          host_centos.vm.box = "<centos7 image>.box"
5          host_centos.vm.provider "virtualbox" do |v|
6                  v.memory = 2048
7                  v.cpus = 2
8                  end
9          host_centos.ssh.pty = true
10         host_centos.vm.network "private_network", ip: "192.168.13.11"
11         host_centos.vm.provision :shell, :path => "bootstrap.sh", :args =>
       ""
12     end
13 end
```

Listing 2: Vagrantfile

I na koniec metoda tworzenia własnych obrazów vagrantów (wielokrotnie jej używałem): `https://blog.engineyard.com/2014/building-a-vagrant-box`

a tutaj co zainstalować, żeby virtualbox działał na systemach z rodziny RedHat (Centos/Fedora): `https://www.if-not-true-then-false.com/2010/install-virtualbox-with-yum-on-fedora-centos-red-hat-rhel`

Miłej nauki

Część II
Listy zadań

1. Lista 1

1.1. Potrzebne narzędzia

1. Virtualbox `https://www.virtualbox.org/wiki/Linux_Downloads`
2. Vagrant `https://www.vagrantup.com/downloads.html`
3. Repo: ¡redacted¿

1.2. Plan działania

1.2.1. Obsługa vagranta

W katalogu kurs-puppeta:

1. vagrant up - załączenie VM
2. vagrant ssh - wejście do maszyny
3. puppet-apply - wprowadzanie zmian puppetem na maszynie wirtualnej
4. puppet-apply-noop - podgląd, co puppet chce zmienić
5. vagrant suspend - uśpienie maszyny
6. vagrant destroy - usunięcie maszyny

1.2.2. Puppet apply

Jest to uruchomienie lokalnie kompilacji i aplikacji kodu na maszynie.
Dokumentacja: `https://docs.puppet.com/puppet/latest/man/apply.html`

```
1 type puppet-apply
2 type puppet-apply-noop
```

Listing 3: Aliasy do komend puppetowych

1.2.3. Co to jest site.pp

Plik ma dość szerokie zastosowanie. Tymczasowo będzie pełnił rolę prostego miejsca do nauki składni puppeta. Jego właściwe wykorzystanie zostanie wprowadzone na kolejnych spotkaniach.

1.2.4. Pierwsze zmiany - tworzenie plików i katalogów

W puppecie jest do tego odpowiedni typ:
Dokomunetacja: `https://docs.puppet.com/puppet/latest/types/file.html`

```
1 file{'/tmp/plik':
2   ensure => file
3 }
```

Listing 4: Tworzenie pustego pliku

```
1 file{'/tmp/plik':
2   ensure  => file,
3   content => 'zzz',
4   mode    => '0700',
5 }
```

Listing 5: Plik z zawartością

1.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch `k1_imie_nazwisko` i do niego wysłać rozwiązanie poniższych zadań w jednym piku `site.pp`

1. Utwórz katalog `/tmp/cos`, ustaw prawa na użytkownika i grupę root, prawa dostępu na 700
2. Utwórz katalog `/tmp/cos1/cos2/cos3` z pomocą jednego wywołania typu file
3. Zapewnij że nie będzie pliku `/tmp/plik1`
4. Utwórz dowiązanie symboliczne z `/tmp/cos` do `/tmp/cos_link`

2. Lista 2

2.1. Plan działania

2.1.1. Domyślne typy zdefiniowane

Doc: `https://docs.puppet.com/puppet/latest/type.html`
W puppecie istnieją domyślne typy zdefiniowane, jeden z nich - file - poznaliśmy na poprzednim spotkaniu. Czas na kolejne. Nie będziemy ich szczegółowo omawiać jeden po drugim, bo nie ma sensu powtarzać dokumentacji. Polecam porządnie ją przejrzeć, i przyjrzeć się parametrom. Pisząc kod puppetowy lepiej wyrobić sobie nawyk korzystania z dokumentacji, bo jak widać jest tych parametrów bardzo dużo i nie ma sensu ich zapamiętywać.

2.1.2. Kolizje nazw w typach zdefiniowanych

Dwukrotne stworzenie obiektu puppetowego odwołującego się do tego samego w systemie operacyjnym jest niedozwolone (w prost). Na przykład: dwukrotne tworzenie tego samego pliku; dwukrotne załączanie tej samej usługi, dwukrotne wywołanie typu zdefiniowanego z tym samym tytułem.
Doc: `https://docs.puppet.com/puppet/latest/lang_defined_types.html#resource-uniqueness`
Uciążliwe, ale ma sens - wymusza jednoznaczność.

2.1.3. Kolejność wykonywania kodu

Domyślnie nie ma w puppecie czegoś takiego jak kolejność - zmiany są aplikowane w dogodnej dla puppeta kolejności. Stąd korzystając z niego należy odzwyczaić się od myślenia, że kolejność w pliku to kolejność aplikowania. Czasem jednak trzeba wymusić jakąś kolejność - np. najpierw zrobić katalog, a dopiero potem plik w nim; najpierw ustawić konfigurację, a potem uruchomić usługę itd. Ponieważ jest to temat rozległy, będzie on rozbity na kilka zajęć, żeby nie wprowadzać chaosu. Na początek wystarczy nam tzw. chaining arrows:
Doc:
`https://docs.puppet.com/puppet/4.9/lang_relationships.html#syntax-chaining-arrows`
`https://docs.puppet.com/puppet/4.9/lang_relationships.html#reversed-forms`
Najczęściej spotyka się pierwszą notację, gdyż kod czytamy od lewej do prawej, z góry na dół, nie odwrotnie.

2.1.4. Cykle w zależnościach

Obiekty i ich kolejność w puppecie musi mieć strukturę grafu acyklicznego (drzewiastą, k-arną, $k \geqslant 2$), to znaczy że nie może być w nim cykli długości 2 i dłuższej. Nie należy wykluczać także współbieżnego wykonywania kodu. Należy tego silnie unikać pisząc własny kod, gdyż w rozbudowanych zbiorach zależnych od siebie modułów rozwiązywanie takich cykli bywa koszmarem.

2.2. Ćwiczenia na zajęcia

2.2.1. Popraw błąd w kodzie

```
1  file{'/tmp/plik1':
2    ensure => file,
3  }
4
5  file{'pipipi lalala':
6    ensure => file,
7    name   => '/tmp/plik1',
8  }
9
10 file{'pipipi lalala':
11   ensure => file,
```

```
12   name   => '/tmp/plik2',
13 }
```

Listing 6: błąd1

```
1  file{'/tmp/plik1'
2    ensure => file,
3  }
4
5  file ('/tmp/plik2':
6    ensure => file,
7  )
8
9  file{'/tmp/plik3'
10   ensure => file
11   mode   => 0700
12 }
```

Listing 7: błąd2

```
1  file {'/tmp/sss/zzz':
2    ensure => file,
3  }->
4  file{'/tmp/sss':
5    ensure => directory,
6  }
```

Listing 8: błąd3

2.2.2. Kolejność

Utwórz trzy pliki z dowolnymi nazwami. Wymuś kolejność alfabetyczną.

2.3. Ćwiczenia do samodzielnego wykonania

1. Utwórz plik `/tmp/katalog/plik`, z zawartością: 123456789, ustaw prawa na użytkownika i grupę root, prawa dostępu na 700. Upewnij się że katalog istnieje
2. Znajdź sobie jakąś usługę chodzącą w systemie (postfix, sshd itp.) i w puppecie:
   - upewnij się że jest podniesiona i uruchomi się przy starcie systemu, jako provider wybierz systemd
   - utwórz plik /tmp/plik2 z zawartością 012345, który przy zmianie swojej zawartości zrestartuje w/w usługę
   - upewnij się że istnieje jakiś pakiet (np. git) i analogicznie - jeśli nie ma, to instaluje i restartuje ulubioną usługę

3. Lista 3

3.1. Plan działania

3.1.1. Odwołania do obiektów

Załóżmy że mamy obiekt:

```
1 file{'/tmp/plik':
2   ensure => file,
3 }
```

Odwołaniem do powyższego obiektu będzie obiekt:

```
1 File['/tmp/plik']
```

Weźmy kolejny przykład, obiekt:

```
1 service{'pipipi lalala':
2   name   => 'postfix',
3   ensure => 'running',
4   enable => true,
5 }
```

i odwołanie do niego:

```
1 Service['pipipi lalala']
```

Doc: `https://docs.puppet.com/puppet/4.9/lang_data_resource_reference.html` Chwilowo zignorujcie kawałek dotyczący klas - jeszcze ich nie wprowadzaliśmy.

3.1.2. Kolejność z użyciem metaparametrów

Doc:
`https://docs.puppet.com/puppet/4.9/lang_relationships.html#syntax-relationship-metaparameters`
Przyszedł czas na metaparamretry, które służą do ustawiania kolejności. Są to: before, require, notify i subscribe.

3.1.3. Kolejność z użyciem odwołań do obiektów

Ostatnią metodą jest użycie odwołania do obiektów i strzałek, na przykład:

```
1 Package['httpd']~>Service['httpd']
2 File['/etc/httpd/httpd.conf']~>Service['httpd']
3 Package['httpd']->File['/etc/httpd/httpd.conf']
```

3.2. Ćwiczenia na zajęcia

3.2.1. Napraw błędy

```
1 file{'/tmp/f1':
2   ensure  => file,
3   require => [
4     File['/tmp/f2'],
5     Service['sshd'],
6   ],
7 }
8
9 file{'/tmp/f2':
```

```
10    ensure => file,
11    require  => [
12      Service['sshd'],
13      File['/tmp/f3'],
14    ],
15  }
16
17  file{'/tmp/f3':
18    ensure => file,
19    require => [
20      File['/tmp/f1'],
21      Service['sshd'],
22    ],
23  }
24
25  service{'sshd':
26    ensure    => running,
27    subscribe => Package['ruby-devel'],
28    before    => File['/tmp/f3']
29  }
30
31  package{'ruby-devel':
32    ensure => present,
33  }
```

Listing 9: Przerwij pętlę zależności

```
1  service{'sshd':
2    ensure => running,
3  }
4
5  file{'/tmp/sshd':
6    ensure  => file,
7    content => '0000',
8  }
```

Listing 10: Popraw kod tak aby zmiana w pliku powodowała restart usługi

3.2.2. Zadanie

Zainstaluj pakiet httpd, upewnij się że po instalacji znajduje się katalog `/etc/httpd` z prawami 0777. Przy zmianie praw zrestartuj usługę httpd. Niech restart httpd wymusza restart usługi postfix.

3.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch i do niego wysłać rozwiązanie poniższych zadań w jednym piku `site.pp`

1. Upewnij się że w systemie znajduje się pakiet git. W przypadku jego braku zainstaluj i wywołaj poniższe polecenie:

   ```
   1  echo $(date) >> /tmp/instalacja_git
   ```

   W przypadku gdy pakiet znajduje się w systemie, niech puppet nie podejmuje żadnej akcji. Wskazówka: do wywołania polecenia użyć typu `exec`.

2. Utwórz plik `/tmp/plik` z zawartością 1234. Tylko wtedy, kiedy puppet wykonuje zmianę w jego zawartości, niech wywołuje poniższe polecenie:

```
1 echo $(date) >> /tmp/zmiana_w_pliku
```

w przeciwnym wypadku niech nie robi nic.

4. Lista 4

4.1. Plan działania

4.1.1. Zmienne

`https://docs.puppet.com/puppet/4.9/lang_variables.html`

4.1.2. Typy zmiennych cz. 1

Puppet pozwala operować na wielu typach zmiennych. Na początek omówimy kilka podstawowych: ciągi znaków, liczby (float i integer), zmienne boolowskie i tablice (jako proste tablice bez zagnieżdżania obiektów)

`https://docs.puppet.com/puppet/4.9/lang_data.html`
`https://docs.puppet.com/puppet/4.9/lang_data_string.html`
`https://docs.puppet.com/puppet/4.9/lang_data_number.html`
`https://docs.puppet.com/puppet/4.9/lang_data_boolean.html`
`https://docs.puppet.com/puppet/4.9/lang_data_array.html`

4.2. Ćwiczenia na zajęcia

1. Utwórz rekursywnie katalogi: `/tmp/p1/p2/p3` wykorzystując typ file i tablicę w tytule
2. Utwórz rekursywnie katalogi: `/tmp/p1/p2/p3` wykorzystując tylko typ exec
3. Jednym wywołaniem typu file, z tablicą w tytule, utwórz pliki: `/tmp/f1 ~ /tmp/f2 ~ /tmp/f3`

4.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k4_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. W katalogu `/tmp/kat` utwórz trzy pliki `/tmp/kat/f1 ~ /tmp/kat/f2 ~ /tmp/kat/f3`, każdy z zawartością `0123`. Niech puppet usuwa z katalogu `/tmp/kat` pliki, których nie ma zdefiniowanych w kodzie (np. ktoś ręcznie tworzy jakiś plik, to puppet go usunie). Wskazówka: w definicji katalogu należy użyć odpowiedniej/odpowiednich opcji

5. Lista 5

5.1. Plan działania

5.1.1. Typy zmiennych cz. 2

Poprzednio poznaliśmy: ciągi znaków, liczby (float i integer), zmienne boolowskie i tablice (jako proste tablice bez zagnieżdżania obiektów). Czas na ciąg dalszy. Tym razem interesują nas tablice z zagnieżdżonymi obiektami, hashe, sensitive, undef i wyrażenia regularne (jako ciekawostka)

`https://docs.puppet.com/puppet/4.9/lang_data_array.html`
`https://docs.puppet.com/puppet/4.9/lang_data_hash.html`
`https://docs.puppet.com/puppet/4.9/lang_data_sensitive.html`
`https://docs.puppet.com/puppet/4.9/lang_data_undef.html`
`https://docs.puppet.com/puppet/4.9/lang_data_regexp.html`

5.1.2. Proteza printf

W celu bieda-debugowania kodu można użyć funkcji puppetowych takich jak `fail` oraz `notice`. Umożliwiają proste podejrzenie wartości zmiennych. Wywołuje się je następująco:

```
1 fail("Waluo")
2 notice("pozor")
```

5.2. Ćwiczenia na zajęcia

1. Sprawdź jak działają funkcje `fail` oraz `notice`
2. Zdefiniuj hasha z minimum trzema elementami i wypisz go na ekran używając jednej z w/w funkcji. Wypisz jego dowolny element.
3. Zdefiniuj tablicę minimum trzech hashy, zawierających minimum trzy elementy i wypisz ją na ekran używając jednej z w/w funkcji. Wypisz dowolny element drugiego hasha w tablicy

5.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k5_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Wypisz na ekran wartości kluczy `faa fods weece fafa` oraz drugi element tablicy pod kluczem `fla` Pozor: to ma działać nawet jak się zmieni wartości z `sii gem susu soso mumu` na dowolne (czyli nie można na skróty)

```
1  $balagan = [
2    {
3      zooo => {
4        wee => {
5          faa => 'sii',
6        }
7      },
8      fii => {
9        sem => [
10         {
11           fods => 'gem',
12         },
13       ],
14     },
15   },
16   {
```

```
    fro => [
      [
        [
          {
            weece => 'susu',
          }
        ],
        [
          {
            fafa => 'soso',
          },
        ]
      ]
    ],
    fla => [
      'sii',
      'mumu',
    ],
  },
]
```

Listing 11: Bałagan

2. Popraw błędy w poniższej taaaablicy:

```
$wiekszy_balagan = [
  {
    fla => [
      'sii',
      'mumu',
      {
        zooo => {
          wee => {
            faa => 'sii,
          }
        },
        fii => {
          sem => {
            {
              fods => {
                fro > [
                  [
                    [
                      {
                        weece => 'susu',
                      }
                    ],
                    [
                      {
                        fafa => 'soso',
                      },
                    ]
                  ]
                ],
              },
            },
          ],
```

```
        },
      },
    ],
  },
]
```

Listing 12: Większy bałagan

6. Lista 6

6.1. Plan działania

6.1.1. Facter

Puppet ma domyślnie dostępny zbiór różnych zmiennych określających stan i konfigurację systemu operacyjnego dostarczanych przez narzędzie o nazwie facter. Z tych zmiennych można domyślnie korzystać w kodzie puppetowym. Działa to w ten sposób, że przy każdym uruchomieniu puppeta odpytywany jest facter na hoście, a nie na puppet serwerze (stąd zawartość zmiennych pochodzi z hosta a nie z serwera puppetowego).
Doc: `https://docs.puppet.com/facter/3.6/core_facts.html`

6.1.2. Własne fakty

Jest możliwość dostarczania własnych faktów jako. W tej chwili skupimy się na jednej z możlwości - wyjście z prostych skryptów napisanych w dowolnym języku, zlokalizowanych lokalnie na hoście.
Doc: `https://docs.puppet.com/facter/3.6/custom_facts.html#executable-facts-----unix`

Do pliku `/etc/facter/facts.d/fakt.sh` wstaw:

```
1 #!/bin/bash
2 echo "wlasny_fakt=pipipi"
```

Listing 13: Własny fakt

nadaj prawa wykonywania dla wszystkich i odpytaj facter o Twój fakt:

```
1 facter wlasny_fakt
```

Listing 14: Zapytanie o własny fakt

6.2. Ćwiczenia na zajęcia

1. Napisz własny fakt w ulubionym języku, podający liczbę plików w katalogu `/etc` i jego podkatalogach pod zmienną `etc_pliki`. Niech fakt podaje także ile miejsca na dysku w kilobajtach zajmuje katalog `/etc`, wartość tę zapisz w zmiennej `etc_rozmiar`

2. Używając factera utwórz plik `/tmp/systeminfo` o poniższej zawartości:

```
1 Os name: <os name>
2 Architecture: <arch>
3 etc stat: <number of files> files in /etc which uses <etc size> kB
      space
4 Size of partition /dev/mapper/sys-root is: <size>, filesystem: <
      filesystem>
5 My computer has <number> physical processor(s) with average <number>
      cores each
6 Main sda disk have <number> of 512B sectors
```

6.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k6_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Napisz fakt z JSONie lub YAMLu, który dostarczy w postaci hasha wyjście z polecenia

```
1 du -d1 -h /etc
```

Listing 15: Polecenie w bashu

2. Używając factera i korzystając z faktu z zad. 1, wypisz na ekran rozmiar katalogu `/etc/security`

3. Nie używając typu exec i systemowego polecenia sha1 wpisz do pliku `/tmp/sha1` sumę sha1 Twojego faktu. Wartość w `/tmp/sha1` powinna się pojawić najpóźniej w trzecim wywołaniu puppet-apply.

4. Dany jest hash:

```
1  $hashyk = [
2    {
3      zooo => {
4        wee => {
5          faa => [
6            {
7              miii => 'zeee',
8            },
9          ],
10       }
11     },
12   },
13 ]
```

Listing 16: Przykładowy hash

Wypisz na ekran wartość `miii` nie używając poniższego sposobu odwołania:

```
1 $hashyk[0]['zooo'] etc.
```

Nie wolno zakładać, że `miii` ma zawsze wartość `zeee`.
Wskazówka: należy użyć odpowiedniej funkcji.

7. Lista 7

7.1. Plan działania

7.1.1. Instrukcje warunkowe

Krótko i na temat: dostępne są if(-elsif)-else, unless, case, selector. Najczęściej używa się pierwszego typu, pozostałe pojawiają się sporadycznie, ale dobrze je znać i wiedzieć co oznaczają.
Doc `https://docs.puppet.com/puppet/latest/lang_conditional.html`

7.1.2. Operatory i wyrażenia

Większość operatorów działa tak jak w pozostałych językach programowania, ale warto sobie usystematyzować składnię. Warto tutaj spojrzeć do dokumentacji:
Doc: `https://docs.puppet.com/puppet/latest/lang_expressions.html`

7.2. Ćwiczenia na zajęcia

1. Niech będzie dana zmienna numb, typu Integer. Skonstruuj hash o nazwie *test_hash* zawierający poniższe klucze:
   - mod2 = $numb(mod2)$
   - mod3 = $numb(mod3)$
   - mod4 = $numb(mod4)$
   - square = $numb^2$
2. Dla zadanej zmiennej `numb` z zad. 1. utwórz tablicę składającą się z poniższych elementów:
   (a) $numb - 1$
   (b) $numb$
   (c) $numb + 1$

   Dołącz tę tablicę do hasha `test_hash` z zad. 1. pod kluczem `numb_amb`
3. Do tablicy z zad. 2 pod kluczem `$test_hash[numb_amb]` dołącz string ze swoim imieniem. Wypisz go na ekran używając funkcji notice. Wstaw je do pliku `/tmp/imie` używając odwołania do hasha.

7.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k7_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Napisz fakt podający liczę zainstalowanych pakietów rpm w systemie operacyjnym (`rpm~-qa~|~wc~-l`). Oblicz jej resztę z dzielenia przez 3 i w zależności od wyniku:
   (a) jeśli reszta = 0, zainstaluj pakiet git
   (b) jeśli reszta = 1, utwórz plik /tmp/rpmki z liczbą pakietów jako zawartość
   (c) jeśli reszta = 2, wymuś restart usługi postfix z użyciem typu service (puppet nie może wtedy tworzyć żadnego innego obiektu w systemie operacyjnym - pliku, instalacji pakietu itd.)
2. Załóżmy że pod zmienną `$adres_ip` mamy jakiś adres IPv4. Wypisz na ekran informację czy jest wewnętrzny czy zewnętrzny.
3. Załóżmy że mamy zmienną `$adres_cidr` będący połączeniem zmiennej `$adres_ip` oraz łańcucha znaków `/32`.

8. Lista 8

8.1. Plan działania

8.1.1. Pętle

Puppet 4 wprowadza możliwość użycia pętli w kodzie manifestów:
`https://docs.puppet.com/puppet/4.9/lang_iteration.html`
Zajmiemy się podstawową metodą - each:
`https://docs.puppet.com/puppet/4.9/function.html#each`

8.2. Ćwiczenia na zajęcia

1. Używając pętli utwórz 4 pliki: `/tmp/p1` do `/tmp/p4`.
2. Dany jest hash:

```
1 $hash1 = {
2   avkj => 'adcfoj',
3   vskj => 'edfijv',
4   rucs => 'wwncuw',
5 }
```

Listing 17: hash

Używając pętli wypisz jego zawartość w postaci:

```
1 klucz: <klucz>, wartosc: <wartosc>
```

8.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k8_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Napisz fakt podający liczbę znaków w `/etc/fstab`, niech wynik będzie dostępny w **tablicy** `$fstab_chars` (powinna to być tablica jednoelementowa). Fakt powinien być dostarczany przez puppeta. Oblicz resztę z dzielenia przez 3 tej liczby i w zależności od wyniku:
   (a) reszta = 0: wpisz `$fstab_chars` do /tmp/chars
   (b) reszta = 1: wypisz na ekran liczbę `$fstab_chars`
   (c) reszta = 2: nic nie rób

   Żaden z przebiegów puppeta nie może zakończyć się błędem.
2. Wypisz na ekran zawartość hasha:

```
1  $hash2 = {
2    avkj => 'adcfoj',
3    vskj => 'edfijv',
4    rucs => [
5      'daflj',
6      'adflj',
7    ],
8    dscd => {
9      sdu => 'wsdew',
10     esa => 'gbgrc'
11   },
12 }
```

Listing 18: hash

w postaci:

```
klucz: avkj
  string: adcfoj
klucz: vskj
  string: edfijv
klucz: rucs
  tablica:
    daflj
    adflj
klucz: dscd
  hash:
    klucz: sdu, wartosc: wsdew
    klucz: esa, wartosc: gbgrc
```

Należy przyjąć że struktura hasha się nie zmieni (tzn tablice pozostaną tablicami, hashe hashami itd.), jedynie nazwy kluczy i wartości mogą się zmienić.

9. Lista 9

9.1. Plan działania

9.1.1. Pętle - ciąg dalszy

Kontynuujemy temat pętli w Puppecie 4. Tym razem omówimy pozostałe metody: slice, filter, map, reduce, with. `https://docs.puppet.com/puppet/4.9/lang_iteration.html` `https://docs.puppet.com/puppet/4.9/function.html#slice` `https://docs.puppet.com/puppet/4.9/function.html#filter` `https://docs.puppet.com/puppet/4.9/function.html#map` `https://docs.puppet.com/puppet/4.9/function.html#reduce` `https://docs.puppet.com/puppet/4.9/function.html#with`

Pewnie ktoś z was zastanawia się: *a na co mnie to*. A do onaczenia zmiennych - może se wziąć i se przekształcić różne struktury danych bez używania zewnętrznych funkcji, które musiałby sobie sam napisać w Rubym.

9.1.2. Lambdy

Ogólnie bloki kodu przekazywane do funkcji (jak dotąd do pętli) nazywamy lambdami. Należy je traktować jako mikro-funkcje które coś robią i zwracają jakiś wynik
`https://docs.puppet.com/puppet/latest/lang_lambdas.html`

9.2. Ćwiczenia na zajęcia

1. Używając funkcji slice podziel tablicę `[0, ..., 999]` na więcej tablic, po 20 liczb każda. Tablice muszą być ciągami rozłącznymi.
2. Używając funkcji slice podziel słowo *szklanka* na pojedyncze litery.
3. Dana jest tablica `[0, ..., 999]`, wypisz na ekran wszystkie liczby zawierające 0
4. Dany jest hash:

```
1 $hash1 = {
2   avkj => 'adcfoj',
3   vskj => 'edfijv',
4   rucs => 'wwncuw',
5 }
```

Listing 19: hash

   Używając funkcji map wypisz wartości jego kluczy.
5. Dana jest tablica `[0, ..., 999]`, używając funkcji reduce wypisz na ekran sumę liczb w tablicy.
6. Dana jest tablica `[0, ..., 999]`, używając funkcji with dodaj do tablicy słowo `aaa`

9.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k9_imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Dany jest hash (zawartość można znaleźć na stronie):

```
1 $dict = {
2   'ieSha5ae' => 1,
3   'ez2Ahkie' => 2,
4   'uTahciw3' => 3,
```

```
  'eet0Ka4y' => 4,
  'Ohtoh9Ai' => 5,
  'ew3Aeveu' => 6,
  'jaech5Ie' => 7,
  'xaiJ2Cig' => 8,
  'ABOus5ac' => 9,
  'aMaeDOos' => 10,
  'rie8ooSh' => 11,
  'Eo3au4Sa' => 12,
  'Ia3gohm0' => 13,
  'eeXah6eu' => 14,
  'iCODohdu' => 15,
  'wuiSh7ae' => 16,
  'Mee3uo9i' => 17,
  'ohL8doh8' => 18,
  'Wo6hee7i' => 19,
  'Eiv1zuta' => 20,
  'huOSoiro' => 21,
  'aakee2Ow' => 22,
  'Jae3oobe' => 23,
  'itie7Voh' => 24,
  'faikahX7' => 25,
  'Oof4Uibe' => 26,
  'ael8Su2R' => 27,
  'aB4dah9o' => 28,
  'Ku2iu1em' => 29,
  'nooCeiy2' => 30,
  'eeNgoh7H' => 31,
  'Eevo2AeS' => 32,
  'eiNg7voh' => 33,
  'EeGho5yo' => 34,
  'zaj4Zaew' => 35,
  'va9peiCh' => 36,
  'eiWei8in' => 37,
  'uzuOKaid' => 38,
  'doht9Iew' => 39,
  'uifeZ9ah' => 40,
  'Al1chavo' => 41,
  'deep7AeS' => 42,
  'wi1Zohgh' => 43,
  'Osheew8z' => 44,
  'ge1eaTi8' => 45,
  'AugaiCi1' => 46,
  'Ahh7ceef' => 47,
  'aif4oZe8' => 48,
  'thooOMoh' => 49,
  'xo1Ingoh' => 50,
  'Eili1aeF' => 51,
  'Chuu5uji' => 52,
  'Nah9pho6' => 53,
  'Heuchai0' => 54,
  'QuohTh6Z' => 55,
  'Shaihu8y' => 56,
  'fozu2euW' => 57,
  'eeGh1Ech' => 58,
  'Ishae7si' => 59,
```

```
61  'Reil2Aur' => 60,
62  'ea2kohGu' => 61,
63  'vah9Ail4' => 62,
64  'Seiwoo1z' => 63,
65  'maeM4ael' => 64,
66  'Ba2eenoo' => 65,
67  'aeGi9joh' => 66,
68  'Go2yog7i' => 67,
69  'Oheeda3j' => 68,
70  'cihuaR8u' => 69,
71  'Pa3Kaavi' => 70,
72  'eeh7ji8L' => 71,
73  'Yae1kexe' => 72,
74  'Shohpoh7' => 73,
75  'noiXo6ph' => 74,
76  'moShe6ch' => 75,
77  'xie4AZo8' => 76,
78  'yiboo9Ye' => 77,
79  'oj1Aim7p' => 78,
80  'Tooghoh9' => 79,
81  'eeruTh2a' => 80,
82  'EeD3ephi' => 81,
83  'OoCei3al' => 82,
84  'okieSh7i' => 83,
85  'noo9Za5g' => 84,
86  'ao3Eepoc' => 85,
87  'CeiB1Soh' => 86,
88  'va8Eenga' => 87,
89  'eira2Hoo' => 88,
90  'oc9aeG3v' => 89,
91  'du6ohZae' => 90,
92  'vozei7Oo' => 91,
93  'laiy3Car' => 92,
94  'ob3Io7ai' => 93,
95  'Thip8owu' => 94,
96  'raih3Xui' => 95,
97  'Ujeeba1p' => 96,
98  'puV5phu1' => 97,
99  'Jeleim1u' => 98,
100 'na6ieFoo' => 99,
101 'gietaiJ9' => 100,
102 'PooZ7euM' => 101,
103 'zeeM1deP' => 102,
104 'ahNeiHi9' => 103,
105 'Iejoa0xo' => 104,
106 'aagh4Jia' => 105,
107 'eedi4Oow' => 106,
108 'eiChah8O' => 107,
109 'eRu0Ohna' => 108,
110 'laiKooc6' => 109,
111 'ooRirun3' => 110,
112 'ophai7Mu' => 111,
113 'ieweP0ee' => 112,
114 'If1aid0a' => 113,
115 'Ii6fahee' => 114,
116 'aew5aiSh' => 115,
```

```
117   'CaeV8afa' => 116,
118   'gaicoo3W' => 117,
119   'Ahda5cah' => 118,
120   'ieP7cheL' => 119,
121   'chee2miG' => 120,
122   'Mae7eid8' => 121,
123   'ieSaigh9' => 122,
124   'Iwu6ahd5' => 123,
125   'oi6OoSh1' => 124,
126   'yu1EXeew' => 125,
127   'saCh0chu' => 126,
128   'yi5Thooz' => 127,
129   'Ixae1iek' => 128,
130   'ooh1fu4U' => 129,
131   'ci6Laipi' => 130,
132   'aiVeeP3O' => 131,
133   'OhZ9Mie0' => 132,
134   'aFaik6Oh' => 133,
135   'ieP5eeng' => 134,
136   'yiegh3Ik' => 135,
137   'Soo7gaev' => 136,
138   'oNgoo7co' => 137,
139   'Eiheebe3' => 138,
140   'wai3Ooze' => 139,
141   'Aed7xeeb' => 140,
142   'faeZ4tee' => 141,
143   'co1Phohp' => 142,
144   'aer2ui1D' => 143,
145   'Saeku8To' => 144,
146   'Iu9tham9' => 145,
147   'xahPhie3' => 146,
148   'AGa3hoTh' => 147,
149   'quah5Eil' => 148,
150   'mohGh2pe' => 149,
151   'OHooR1id' => 150,
152   'ui0eu1Sa' => 151,
153   'cieGhio3' => 152,
154   'Eichem0O' => 153,
155   'Oom1ooch' => 154,
156   'eiphai1X' => 155,
157   'afoJ6sai' => 156,
158   'huihooY8' => 157,
159   'ieNiCoh6' => 158,
160   'thaiph1Z' => 159,
161   'Od7Aengo' => 160,
162   'FowuCe6u' => 161,
163   'ooLev4tu' => 162,
164   'ongaec7J' => 163,
165   'ooGhi9ye' => 164,
166   'noo8aePh' => 165,
167   'IeK9eesh' => 166,
168   'lo3mohCo' => 167,
169   'Uki8saeP' => 168,
170   'Dohlie8m' => 169,
171   'wahcee8U' => 170,
172   'leo7ahSh' => 171,
```

```
173   'phoB1ak2' => 172,
174   'Ec2mohvo' => 173,
175   'meiba0Ee' => 174,
176   'JaiK0moo' => 175,
177   'TaoTh2pi' => 176,
178   'Aejae4oh' => 177,
179   'fuuqu2Oh' => 178,
180   'Moph2Azi' => 179,
181   'phaeY6ae' => 180,
182   'Sei8jash' => 181,
183   'Shaitoo1' => 182,
184   'Yaechum8' => 183,
185   'Beox4Cex' => 184,
186   'UshaiSh6' => 185,
187   'thiXee6o' => 186,
188   'Ohjae4ux' => 187,
189   'que1Leeb' => 188,
190   'roo2Aezu' => 189,
191   'ooCe6ATu' => 190,
192   'tiezueX1' => 191,
193   'xaeCaef3' => 192,
194   'oejoo9Qu' => 193,
195   'lahX7Boo' => 194,
196   'la7Eevok' => 195,
197   'xei9Aquo' => 196,
198   'Uphei1Oz' => 197,
199   'Roo7ooNu' => 198,
200   'AeNoh5ov' => 199,
201   'Ieloo3ae' => 200,
202   'ohThee0u' => 201,
203   'aebaiTh5' => 202,
204   'Eoghai7n' => 203,
205   'eebae9Au' => 204,
206   'vauTu4et' => 205,
207   'NeiYie9z' => 206,
208   'Vohk4Ba0' => 207,
209   'rieY9que' => 208,
210   'aepoo3Bu' => 209,
211   'Quo4bioc' => 210,
212   'noch4aeG' => 211,
213   'noa6ohCh' => 212,
214   'Cee8equu' => 213,
215   'piey5uuS' => 214,
216   'AhmeeW5i' => 215,
217   'Aim8ooVu' => 216,
218   'iiHo9sha' => 217,
219   'Se3iwahf' => 218,
220   'Aiv3ohj7' => 219,
221   'AeFohTo4' => 220,
222   'Ang4xooX' => 221,
223   'aY8paht4' => 222,
224   'ohquus4A' => 223,
225   'kibi3Fey' => 224,
226   'ac2Vo9ah' => 225,
227   'uSoothu0' => 226,
228   'eiNaiz1A' => 227,
```

```
229   'deoPh6Ie' => 228,
230   'iL4quoon' => 229,
231   'eequ1Quo' => 230,
232   'Fa6maegu' => 231,
233   'eeQu5bao' => 232,
234   'eemii0Iz' => 233,
235   'Ki3paefo' => 234,
236   'loo6CouL' => 235,
237   'ahNai2qu' => 236,
238   'eesh3Otu' => 237,
239   'eedah9Bu' => 238,
240   'Heengei5' => 239,
241   'FuOnee3p' => 240,
242   'thee2Eef' => 241,
243   'aiw8Ooph' => 242,
244   'nauXu2fe' => 243,
245   'Deeze3or' => 244,
246   'yeMoh5ei' => 245,
247   'ieJ9ier1' => 246,
248   'uKooMu8h' => 247,
249   'XiMohGh2' => 248,
250   'eir6Chee' => 249,
251   'quoiW3No' => 250,
252   'pi6ePhah' => 251,
253   'Ohpash8g' => 252,
254   'Eif1Xael' => 253,
255   'geifOTie' => 254,
256   'du9iW4oo' => 255,
257   'ip2eiZ8G' => 256,
258   'Xai5Weeb' => 257,
259   'Eeg6ahCh' => 258,
260   'dou4Ohxe' => 259,
261   'Phaeb6ko' => 260,
262   'huPhi4Ri' => 261,
263   'Ei6aemax' => 262,
264   'aeRuo3Pi' => 263,
265   'ooxohL7A' => 264,
266   'equ9Phoo' => 265,
267   'aeb8Eeki' => 266,
268   'po8Alah9' => 267,
269   'Ush3aeng' => 268,
270   'xoadie5G' => 269,
271   'ietu6OuG' => 270,
272   'ush1guiY' => 271,
273   'yahNg5ee' => 272,
274   'eiX5osaB' => 273,
275   'Thiozee3' => 274,
276   'eeve6eiJ' => 275,
277   'Uiciesi0' => 276,
278   'deiSai1J' => 277,
279   'Oiph1iel' => 278,
280   'wooV3cei' => 279,
281   'rahT7ahl' => 280,
282   'vaiv4RoS' => 281,
283   'iab7Ooji' => 282,
284   'voh5Pai9' => 283,
```

```
285   'iedoo1Yi' => 284,
286   'ohgh4aeL' => 285,
287   'So7uinot' => 286,
288   'thutie2A' => 287,
289   'lai4mo3E' => 288,
290   'Ohm4xuar' => 289,
291   'Sheih3ae' => 290,
292   'ye0Ootei' => 291,
293   'eipoo9Xu' => 292,
294   'vaL3AhFo' => 293,
295   'wohHe8ka' => 294,
296   'Ed8ieTh6' => 295,
297   'Oong3ogh' => 296,
298   'en1Nuaja' => 297,
299   'Pa9Phai9' => 298,
300   'uju7Seen' => 299,
301   'Saa8heet' => 300,
302   'Sei4Ques' => 301,
303   'yeN8new0' => 302,
304   'pie8Ooth' => 303,
305   'Ieso8yuC' => 304,
306   'ieshee3Y' => 305,
307   'Joaf6chu' => 306,
308   'einiGh6e' => 307,
309   'Eis6Ais6' => 308,
310   'ohkai7Jo' => 309,
311   'eeY9or4i' => 310,
312   'eMooz1ek' => 311,
313   'to3yooPi' => 312,
314   'Xo4oth1u' => 313,
315   'ba5ieTho' => 314,
316   'AiQuie2p' => 315,
317   'oo7dae1I' => 316,
318   'Voo4jaer' => 317,
319   'ooSh1ohm' => 318,
320   'Li7Queig' => 319,
321   'uig7ahR7' => 320,
322   'Hai1eeto' => 321,
323   'caKioch7' => 322,
324   'Oowei7un' => 323,
325   'ae4Cheem' => 324,
326   'Viebae6h' => 325,
327   'rei8Gahz' => 326,
328   'zahk8IuK' => 327,
329   'ooy5hiNe' => 328,
330   'Wuqu2iex' => 329,
331   'eMei2riv' => 330,
332   'gaihaiR2' => 331,
333   'aeSoh5ie' => 332,
334   'Ue0aJei5' => 333,
335   'iekiQu1e' => 334,
336   'Cua1aipi' => 335,
337   'paev1aTh' => 336,
338   'udo3quoX' => 337,
339   'Eexoo7zu' => 338,
340   'Aesh3pha' => 339,
```

```
341   'uifie7Ai' => 340,
342   'Reaph2ah' => 341,
343   'jaeci4Cu' => 342,
344   'Pheet0su' => 343,
345   'vahKeir8' => 344,
346   'Tae3mah9' => 345,
347   'wae1Iu9a' => 346,
348   'eVeiBam9' => 347,
349   'ahghah0I' => 348,
350   'Aikae4nu' => 349,
351   'asie5Eiy' => 350,
352   'Eer6Ejae' => 351,
353   'phahb3La' => 352,
354   'Ahxo5lee' => 353,
355   'ieho6aiG' => 354,
356   'meeb8Och' => 355,
357   'xoo5IeSo' => 356,
358   'YahMei2i' => 357,
359   'ru8Ooquu' => 358,
360   'Yahtee8u' => 359,
361   'Aph4loof' => 360,
362   'Xeejio9E' => 361,
363   'ni5Uifee' => 362,
364   'Fae5aenu' => 363,
365   'vaah6Kae' => 364,
366   'Ohgoh0Mo' => 365,
367   'Kei8yohn' => 366,
368   'rohX2aak' => 367,
369   'etohMah6' => 368,
370   'aicheM0e' => 369,
371   'Eizeph7u' => 370,
372   'kaideeY5' => 371,
373   'oasoo8uK' => 372,
374   'Utusoaf2' => 373,
375   'ohngoh3P' => 374,
376   'yi1Choox' => 375,
377   'Ahl6aiqu' => 376,
378   'Eeth6ohl' => 377,
379   'Ail6aehu' => 378,
380   'woev5Ahs' => 379,
381   'li1Ael4w' => 380,
382   'Eeral1qu' => 381,
383   'eeThou1f' => 382,
384   'Pae2lej8' => 383,
385   'eex0Avah' => 384,
386   'eefaiCo1' => 385,
387   'jo0Chiex' => 386,
388   'Zua6FeiC' => 387,
389   'IGhi7Ohd' => 388,
390   'Ohn9Pahr' => 389,
391   'eeVahch4' => 390,
392   'aer3xooM' => 391,
393   'yah6kahT' => 392,
394   'uif4Akai' => 393,
395   'la7ed3Ch' => 394,
396   'be0Sah5u' => 395,
```

```
397   'hoh3Aixu' => 396,
398   'aeQuao5z' => 397,
399   'ab3Joo2c' => 398,
400   'uu0aeG5g' => 399,
401   'ohohJ2ee' => 400,
402   'Viet8ogh' => 401,
403   'Ough1moh' => 402,
404   'oot5Wa8p' => 403,
405   'eix9ooV7' => 404,
406   'bod4Eico' => 405,
407   'Bie3pahv' => 406,
408   'Ka8ohriu' => 407,
409   'fu6EiFed' => 408,
410   'guid6Yoh' => 409,
411   'OF5haiqu' => 410,
412   'eiX9aimi' => 411,
413   'Kae5Ooqu' => 412,
414   'Fooy2Aig' => 413,
415   'AhH7shei' => 414,
416   'Mu9ahqu4' => 415,
417   'Ta5Phah2' => 416,
418   'av9Bohv6' => 417,
419   'oGonoox6' => 418,
420   'UNie9bie' => 419,
421   'Ahngieh4' => 420,
422   'daeM4que' => 421,
423   'eihohK9B' => 422,
424   'Iuc8seix' => 423,
425   'ciex9Wei' => 424,
426   'tohNi9He' => 425,
427   'aeZohsh3' => 426,
428   'dahthu0O' => 427,
429   'Ahs6iedo' => 428,
430   'ekoh9Joh' => 429,
431   'Oogae0sh' => 430,
432   'ooR8Eith' => 431,
433   'Ahdoogh7' => 432,
434   'IeCh0ohw' => 433,
435   'Auqu0bu5' => 434,
436   'esuNg1oh' => 435,
437   'AeB3ata7' => 436,
438   'Loo8yuch' => 437,
439   'Iew3Miex' => 438,
440   'Go0Ahhae' => 439,
441   'Ia3heuxe' => 440,
442   'ooc8eLol' => 441,
443   'Deidaa6e' => 442,
444   'uuNooch8' => 443,
445   'ooG9eich' => 444,
446   'raiGoo5a' => 445,
447   'pohc5Aes' => 446,
448   'CiaTh5ch' => 447,
449   'Vaeque7h' => 448,
450   'izeiv0Jo' => 449,
451   'AilieB7i' => 450,
452   'Eiz7KieH' => 451,
```

```
453   'ipeiRoh2' => 452,
454   'juaMoo8b' => 453,
455   'ohK8li9G' => 454,
456   'chej2Oez' => 455,
457   'pheipu3O' => 456,
458   'zash8YuF' => 457,
459   'eo6ahJ6i' => 458,
460   'Ugho1iej' => 459,
461   'Uyer4ean' => 460,
462   'FeeR4ohX' => 461,
463   'asaip2Co' => 462,
464   'Shahloo7' => 463,
465   'choFOeil' => 464,
466   'cai7ux1O' => 465,
467   'Pahxah1e' => 466,
468   'hooqu9Oh' => 467,
469   'ohMah8eu' => 468,
470   'be8eil6E' => 469,
471   'Zeigoo7e' => 470,
472   'oothie2D' => 471,
473   'Lai1aeth' => 472,
474   'Mee1ahfe' => 473,
475   'AN7Tohga' => 474,
476   'eex7UHoh' => 475,
477   'Eifec4ei' => 476,
478   'phifuf4K' => 477,
479   'Cei6ya5F' => 478,
480   'vah2uw9E' => 479,
481   'oongoT4C' => 480,
482   'jighuL2E' => 481,
483   'Noo6iequ' => 482,
484   'jeiR9ahS' => 483,
485   'oke7ICoo' => 484,
486   'HohQu7ph' => 485,
487   'Tae1ethe' => 486,
488   'Ulie2eox' => 487,
489   'ooChie7o' => 488,
490   'ohNgu3lu' => 489,
491   'Othoang9' => 490,
492   'eetaeWi2' => 491,
493   'ub2AhPei' => 492,
494   'So1aiJei' => 493,
495   'Oosakae9' => 494,
496   'GajOeuqu' => 495,
497   'phumuw7O' => 496,
498   'eng1Ishe' => 497,
499   'ahieOGai' => 498,
500   'ohz1aeNg' => 499,
501   'uuyemi4E' => 500,
502   'Oosh1yah' => 501,
503   'Wooke3ie' => 502,
504   'Phiudae9' => 503,
505   'thai4Osh' => 504,
506   'eu9ugh9Y' => 505,
507   'IYohOhoh' => 506,
508   'iethai8W' => 507,
```

```
509   'kiQu2ji1' => 508,
510   'Ea4av7ch' => 509,
511   'Izuph0ee' => 510,
512   'mahLee4e' => 511,
513   'Ahpihu2h' => 512,
514   'thaY5eib' => 513,
515   'chahLoh4' => 514,
516   'uTa2dooY' => 515,
517   'Aey6iiya' => 516,
518   'zaeJee9p' => 517,
519   'ahsh1Yah' => 518,
520   'cho0ePh0' => 519,
521   'Ood4Uy6O' => 520,
522   'moshai4E' => 521,
523   'ai9cus9J' => 522,
524   'gohth3Ai' => 523,
525   'Oovae6qu' => 524,
526   'yai6iTh0' => 525,
527   'choXae4u' => 526,
528   'ohM0leey' => 527,
529   'taiB3sho' => 528,
530   'Oopho2ah' => 529,
531   'iyievi9E' => 530,
532   'Od6ohsia' => 531,
533   'shohR9xi' => 532,
534   'eigheeY4' => 533,
535   'pohB8boh' => 534,
536   'Eexee0af' => 535,
537   'ahfai6Ra' => 536,
538   'eihePha7' => 537,
539   'Sahc0evo' => 538,
540   'eif0Eaxi' => 539,
541   'Leech8qu' => 540,
542   'Uo1ahgha' => 541,
543   'veD3ne4e' => 542,
544   'Ahd1eez2' => 543,
545   'Aush6thu' => 544,
546   'ox2Aic9m' => 545,
547   'Eing5Ie0' => 546,
548   'Aihip3Ur' => 547,
549   'eVei0Tha' => 548,
550   'kahThii7' => 549,
551   'Miekoo1p' => 550,
552   'ahvie7Ta' => 551,
553   'aesiel1E' => 552,
554   'igh8ieC2' => 553,
555   'ji9Icai6' => 554,
556   'Oosh5Pha' => 555,
557   'No8yai4i' => 556,
558   'eedeeG8o' => 557,
559   'Aegu2sha' => 558,
560   'vaen9eeW' => 559,
561   'YozieCh7' => 560,
562   'Hied0Ohb' => 561,
563   'rairu0Ae' => 562,
564   'goom8Aew' => 563,
```

```
565   'Eelohko8' => 564,
566   'Ame3aich' => 565,
567   'eeSohT8e' => 566,
568   'ooM9cook' => 567,
569   'Ceo6iech' => 568,
570   'Johfai2W' => 569,
571   'Eb6aShi2' => 570,
572   'EiF3Ra9p' => 571,
573   'Sho0fier' => 572,
574   'Yu0xeiHa' => 573,
575   'eiQuoef9' => 574,
576   'aetiv9Ph' => 575,
577   'Ais9oGhe' => 576,
578   'aqua6Yae' => 577,
579   'Eith6too' => 578,
580   'av9shaiL' => 579,
581   'pahgaeN2' => 580,
582   'Iex1Coh1' => 581,
583   'iefier2Z' => 582,
584   'Ohjieg9A' => 583,
585   'aseex5Ei' => 584,
586   'nu5Aehe1' => 585,
587   'xei9Heiw' => 586,
588   'phoiV1um' => 587,
589   'OoT5shuk' => 588,
590   'vaiK2ain' => 589,
591   'te9AhP0V' => 590,
592   'jaiChos0' => 591,
593   'RieWao9a' => 592,
594   'Dei7icha' => 593,
595   'EiWie2ch' => 594,
596   'oomo2ohY' => 595,
597   'hie2io9I' => 596,
598   'ahF6iep3' => 597,
599   'eithoo7H' => 598,
600   'aphuuC4h' => 599,
601   'aFe1Pu3t' => 600,
602   'xex1leeK' => 601,
603   'Tai9do8a' => 602,
604   'xohCah1u' => 603,
605   'iefohL0o' => 604,
606   'eR2eeshi' => 605,
607   'IePhei4G' => 606,
608   'sohhohL5' => 607,
609   'xiewo5Ph' => 608,
610   'AShaiye3' => 609,
611   'veX5aeKo' => 610,
612   'aho5eiMi' => 611,
613   'oosee9Ei' => 612,
614   'aeGhaj6e' => 613,
615   'gier6ueF' => 614,
616   'ohqu6Cae' => 615,
617   'uzo1Jaej' => 616,
618   'eiN2Shie' => 617,
619   'aikig8Ie' => 618,
620   'Aengohn1' => 619,
```

```
621   'avahB7ye' => 620,
622   'feiZ1ahH' => 621,
623   'ei7Ajei1' => 622,
624   'euSo2iLu' => 623,
625   'aiTeb6oh' => 624,
626   'yies2Rah' => 625,
627   'Bel6ohch' => 626,
628   'nei1yieT' => 627,
629   'uliGai5a' => 628,
630   'OShail8o' => 629,
631   'yoh8EeCe' => 630,
632   'Zahko2ie' => 631,
633   'po1ohVai' => 632,
634   'Gaes5saa' => 633,
635   'IenieR5x' => 634,
636   'ahghuN4u' => 635,
637   'heePh2oV' => 636,
638   'ieSh6Ahz' => 637,
639   'eeyoh0Nu' => 638,
640   'gieC5Eel' => 639,
641   'nee5Thie' => 640,
642   'too7UG8s' => 641,
643   'Cai8peiG' => 642,
644   'saimiw3E' => 643,
645   'eig2eDai' => 644,
646   'Eet7yahb' => 645,
647   'yorah5Eg' => 646,
648   'oTh1Waey' => 647,
649   'haeN8xou' => 648,
650   'Dish8koo' => 649,
651   'eegheiM3' => 650,
652   'Thup7ohf' => 651,
653   'eibim3Xo' => 652,
654   'cee9Ooy6' => 653,
655   'iu6AhGee' => 654,
656   'Zai7aere' => 655,
657   'Mae7fi6y' => 656,
658   'phir7aeP' => 657,
659   'voo6Ootu' => 658,
660   'aiQua5ah' => 659,
661   'aipu7Ohk' => 660,
662   'kahQu0ai' => 661,
663   'iju6Iehi' => 662,
664   'os2Sheet' => 663,
665   'kieTh6ha' => 664,
666   'Xeenie9y' => 665,
667   'eigh5Oom' => 666,
668   'Moy6Aemi' => 667,
669   'ieWahz9j' => 668,
670   'Hohngoo3' => 669,
671   'Phid7rah' => 670,
672   'Zaph9phu' => 671,
673   'theob4Wa' => 672,
674   'quaul7uP' => 673,
675   'Sai5oth1' => 674,
676   'ieHuxu7h' => 675,
```

```
677   'Aid9ies7' => 676,
678   'Ief6phia' => 677,
679   'Ohmu2iew' => 678,
680   'Eiqu2phi' => 679,
681   'Vaighe9i' => 680,
682   'Kea2Ieko' => 681,
683   'ieGh4toh' => 682,
684   'thoch3Ae' => 683,
685   'oozieTu5' => 684,
686   'Ag5zaTh4' => 685,
687   'Uvoo8Mi2' => 686,
688   'eetha3Zi' => 687,
689   'oom3woDi' => 688,
690   'Bai0The7' => 689,
691   'ob8ji4Xi' => 690,
692   'aesait0O' => 691,
693   'taiZoo6w' => 692,
694   'oozav7Ei' => 693,
695   'aochie0J' => 694,
696   'Phi7ahnu' => 695,
697   'aGhooc3h' => 696,
698   'uu9ibuaM' => 697,
699   'chaewa9J' => 698,
700   'saT8bohw' => 699,
701   'aK3zomoo' => 700,
702   'ruN2ubai' => 701,
703   'oP4eeShu' => 702,
704   'Vood0eil' => 703,
705   'ras4ohGe' => 704,
706   'Pheex9Ju' => 705,
707   'Ohneish7' => 706,
708   'Ep0sai0w' => 707,
709   'doo2oiTh' => 708,
710   'Ahqu4coa' => 709,
711   'ooJ4uong' => 710,
712   'shooK2ob' => 711,
713   'aciJeiw3' => 712,
714   'egeiC1be' => 713,
715   'biem8ahF' => 714,
716   'ezeiThu7' => 715,
717   'ahDae5al' => 716,
718   'bahj5aeF' => 717,
719   'Thahn7ae' => 718,
720   'eeQui5ea' => 719,
721   'Ohtonu0I' => 720,
722   'geP4niro' => 721,
723   'iXei4moo' => 722,
724   'ohhu8EoN' => 723,
725   'Gees3gun' => 724,
726   'toh7IeCh' => 725,
727   'Zahgh9fo' => 726,
728   'aip3Eech' => 727,
729   'AiQuam9e' => 728,
730   'quoos3Eo' => 729,
731   'Uing2eZu' => 730,
732   'haeGh4xe' => 731,
```

```
733   'JikooZ6d' => 732,
734   'Jaesh8ai' => 733,
735   'iquOOofo' => 734,
736   'Aerohs2u' => 735,
737   'Ek3iz1vu' => 736,
738   'ohmiboP4' => 737,
739   'Eecie9ko' => 738,
740   'kiLei2uo' => 739,
741   'eiKoOiov' => 740,
742   'haiJ8Koy' => 741,
743   'vie0Ew2m' => 742,
744   'Laeh3cah' => 743,
745   'ri4aeCh7' => 744,
746   'Paebahx3' => 745,
747   'Phaa6vuy' => 746,
748   'eeV6ohsh' => 747,
749   'Aexu8eeg' => 748,
750   'eiGi1phi' => 749,
751   'kah5ia9O' => 750,
752   'quie2eR7' => 751,
753   'Bahph9ph' => 752,
754   'chohj8Oa' => 753,
755   'SiojeYi3' => 754,
756   'exayai8C' => 755,
757   'aich4Oi0' => 756,
758   'thur7yuH' => 757,
759   'uh2eij5B' => 758,
760   'pau2Wede' => 759,
761   'ieke2aeC' => 760,
762   'XohSh8es' => 761,
763   'Oghie3ji' => 762,
764   'lieLooc7' => 763,
765   'sei8ahK8' => 764,
766   'iuTheev2' => 765,
767   'Xai2keet' => 766,
768   'Quai8xee' => 767,
769   'Ooph5aic' => 768,
770   'Ieshee4e' => 769,
771   'noe7eR5O' => 770,
772   'goofo9Tu' => 771,
773   'ahci5Ixa' => 772,
774   'ahJoh2Ah' => 773,
775   'eechaeG3' => 774,
776   'quah3Nai' => 775,
777   'de4ahWee' => 776,
778   'Yien5ozo' => 777,
779   'huOOquu1' => 778,
780   'ohc7eeCh' => 779,
781   'eeyah4Iv' => 780,
782   'pu4Iutuh' => 781,
783   'Ier1et0I' => 782,
784   'Ueshoaib' => 783,
785   'iuf4Sooc' => 784,
786   'die8na5D' => 785,
787   'Booz7Phi' => 786,
788   'Aiquae6a' => 787,
```

```
789   'aiGoof8y' => 788,
790   'woM4rith' => 789,
791   'Ahgoolo2' => 790,
792   'Doo6Jooh' => 791,
793   'chaer2Za' => 792,
794   'huo2Quah' => 793,
795   'ohrook1J' => 794,
796   'uk2thooZ' => 795,
797   'ien9OhPh' => 796,
798   'ia4ixiiP' => 797,
799   'saeFah3i' => 798,
800   'phaish4E' => 799,
801   'iCoo9Chi' => 800,
802   'Geed3Le7' => 801,
803   'Seil6teC' => 802,
804   'uphi6oiF' => 803,
805   'phoo2Iey' => 804,
806   'Ongoh9da' => 805,
807   'Jaiphe0d' => 806,
808   'ohngOoTh' => 807,
809   'aiG6aich' => 808,
810   'eBeeth4u' => 809,
811   'sa7Tooch' => 810,
812   'EiGi5sif' => 811,
813   'IVieM5ah' => 812,
814   'kuHOahza' => 813,
815   'aeR5Edeu' => 814,
816   'paoJei1i' => 815,
817   'uqu5Woom' => 816,
818   'Die3utho' => 817,
819   'tuNg5shi' => 818,
820   'IeVoo9gu' => 819,
821   'siego6Wa' => 820,
822   'isohV4ae' => 821,
823   'Shei8AiW' => 822,
824   'Aihae4ie' => 823,
825   'ishoo8Qu' => 824,
826   'Shee4aes' => 825,
827   'Pu8eeh1l' => 826,
828   'ahmOahBi' => 827,
829   'cei8IJ7i' => 828,
830   'Bia9Ohzi' => 829,
831   'ohVahh2m' => 830,
832   'eedoo6Bo' => 831,
833   'Faiy3ieM' => 832,
834   'eZahh3Ee' => 833,
835   'Eeb2ohx2' => 834,
836   'ees6jaiZ' => 835,
837   'Za8queed' => 836,
838   'Chai4AhW' => 837,
839   'eish2Aip' => 838,
840   'cee7EiCh' => 839,
841   'Eeth7eJa' => 840,
842   'vai4oeCh' => 841,
843   'aeb6Unah' => 842,
844   'ceiHae1Y' => 843,
```

```
845   'bai7oCh0' => 844,
846   'wahd7iPe' => 845,
847   'aehahL7u' => 846,
848   'SahNgoh0' => 847,
849   'OhNg3tho' => 848,
850   'ma3ahn1O' => 849,
851   'moNgoh3k' => 850,
852   'Oog9noc3' => 851,
853   'Ahdia8Ov' => 852,
854   'auCi5ueT' => 853,
855   'ahQui6oo' => 854,
856   'aiko0Yoh' => 855,
857   'aez2OhGi' => 856,
858   'Eequah7O' => 857,
859   'asom6iCh' => 858,
860   'ahsh8ahB' => 859,
861   'Iepa7xie' => 860,
862   'ieZ5Teey' => 861,
863   'aigh2Vie' => 862,
864   'ix3Aevoo' => 863,
865   'Eegh3kej' => 864,
866   'Quie1AhD' => 865,
867   'ieMoop5c' => 866,
868   'teiKoob5' => 867,
869   'Fae3Iwoo' => 868,
870   'ZeeFeij5' => 869,
871   'ooneiK7X' => 870,
872   'jenooy4A' => 871,
873   'Iex1ceiw' => 872,
874   'Ney7feer' => 873,
875   'Tez5puef' => 874,
876   'biej7uQu' => 875,
877   'Ethai7tu' => 876,
878   'US6yahgh' => 877,
879   'ZieP8eap' => 878,
880   'pah5ooQu' => 879,
881   'taGhoo2a' => 880,
882   'FoiT6ahh' => 881,
883   'ieRo6cho' => 882,
884   'OoSheif7' => 883,
885   'uw5Tolo6' => 884,
886   'ohJiaYo3' => 885,
887   'pohPhuu8' => 886,
888   'ivipeiR2' => 887,
889   'Thethei8' => 888,
890   'Jux3aPho' => 889,
891   'Leef5aiw' => 890,
892   'Raiph1yo' => 891,
893   'Hiegh4ok' => 892,
894   'ahB5Queb' => 893,
895   'oHoogh6W' => 894,
896   'Xei8eo2t' => 895,
897   'EiG3xai8' => 896,
898   'Hoo0Upea' => 897,
899   'Ood9Juku' => 898,
900   'kej4Auqu' => 899,
```

```
901   'miCh4AhR' => 900,
902   'Toow5deS' => 901,
903   'xa9Mae6A' => 902,
904   'OhL5poh3' => 903,
905   'CaoWi1ud' => 904,
906   'Theech2s' => 905,
907   'ahxuZ4oo' => 906,
908   'NeNg2equ' => 907,
909   'Zah0aege' => 908,
910   'EiLoop2a' => 909,
911   'oF2amixu' => 910,
912   'Ohzoo4oh' => 911,
913   'Xoh2ieCa' => 912,
914   'yuYol6Li' => 913,
915   'ooZeen4a' => 914,
916   'ohw7nohP' => 915,
917   'Maivu4Yi' => 916,
918   'Bu5eezoh' => 917,
919   'mei3ahX5' => 918,
920   'ohL8Ahz6' => 919,
921   'roNo9ahP' => 920,
922   'hoo5ooCi' => 921,
923   'eeYet2qu' => 922,
924   'wee4Aeti' => 923,
925   'KooRaek0' => 924,
926   'AizuXox9' => 925,
927   'Ku2kaeve' => 926,
928   'Ceeng0re' => 927,
929   'tuo8Ieth' => 928,
930   'faum3aiM' => 929,
931   'ugh4Phao' => 930,
932   'Ohth9eeB' => 931,
933   'awu8Lee0' => 932,
934   'eiXee5ae' => 933,
935   'ua3AhMoo' => 934,
936   'iema4Mah' => 935,
937   'yoa8Ae0F' => 936,
938   'Vee7yaiR' => 937,
939   'eiB0gahf' => 938,
940   'woof4Coo' => 939,
941   'ooPo0Eis' => 940,
942   'Ahso9ucu' => 941,
943   'ooNgahf6' => 942,
944   'Iraow9ye' => 943,
945   'Uag1quei' => 944,
946   'noh3Eiti' => 945,
947   'Tia5ue4s' => 946,
948   'ZouHoe0Z' => 947,
949   'Ai9eeque' => 948,
950   'ooZoo2ri' => 949,
951   'meeHohD5' => 950,
952   'Ho4Ahka7' => 951,
953   'phei1Bex' => 952,
954   'teFuw9da' => 953,
955   'ae4thiaB' => 954,
956   'aPheon2w' => 955,
```

```
  'ree0Zere' => 956,
  'Nezaang1' => 957,
  'VaeY0mei' => 958,
  'eih8feiG' => 959,
  'eeZ1eiNe' => 960,
  'phah1EiG' => 961,
  'she2Iahi' => 962,
  'Hee2zead' => 963,
  'be3viH5B' => 964,
  'eQu1ev6v' => 965,
  'queeLah0' => 966,
  'ohCh7ir7' => 967,
  'juof6Noh' => 968,
  'ahH3ahpu' => 969,
  'rea3AiVu' => 970,
  'uv2Ieshe' => 971,
  'koiKao8s' => 972,
  'fui0Jahs' => 973,
  'cuF0Iepa' => 974,
  'OhWee5oh' => 975,
  'ao0Faino' => 976,
  'oong0Rai' => 977,
  'nech4Rie' => 978,
  'sie7eQuu' => 979,
  'Mok8oiYu' => 980,
  'Teiph3hu' => 981,
  'Thohy6oh' => 982,
  'Ahpa2uip' => 983,
  'yahm4ueK' => 984,
  'Axiebae8' => 985,
  'iereeF8g' => 986,
  'eixei2Ee' => 987,
  'Ur8ahf9c' => 988,
  'eeb9Ii6t' => 989,
  'eulair8R' => 990,
  'Iyaixi9a' => 991,
  'Fiefee1d' => 992,
  'sheij4Ro' => 993,
  'oP0AiThu' => 994,
  'waeCh1Oh' => 995,
  'Yaecait3' => 996,
  'aL0ouCei' => 997,
  'eiTh7Que' => 998,
  'ahTai0xu' => 999,
}
```

Wypisać na ekran w postaci tablicy nazwy jego wszystkich kluczy zawierających cyfrę 2, których wartości dzielą się przez 42. Nie wolno używać operatora dzielenia. Również niedozwolone jest wypisywanie ręcznie żądanych elementów oraz powoływanie się na krotności liczby 42.

2. Używając odpowiednich funkcji przekształć tablicę:

```
$tablica = [
  ['au7eCeep', 'wae1aSho'],
  ['thi7Riek', 'aeKie3ah'],
  ['paH1Aimo', 'Aihae5Ah'],
  ['teewaf9X', 'Shei6eit'],
  ['Loo6igh0', 'Aig1UShe'],
  ['Ieme2IeY', 'voLohsh8'],
  ['eevaiMa1', 'Gu7aphie'],
  ['ik4Cheip', 'Iegah8ye'],
  ['adaig7Ei', 'Eihawa7a'],
  ['Quiquie5', 'wohvaCh7'],
]
```

na hash (zawartość można znaleźć na stronie):

```
{
  'au7eCeep' => 'wae1aSho',
  'thi7Riek' => 'aeKie3ah',
  etc.
}
```

Listing 20: hash

3. Dana jest tablica:

```
[
  '2aaa',
  'b2bb',
  'cc2c',
]
```

Zastąp liczbę 2 literą X we wszystkich elementach tablicy. Wskazówka: użyć odpowiedniej funkcji.

10. Lista 10

10.1. Plan działania

10.1.1. Deduplikacja kodu - pierwsze podejście

Sporą bolączką rozbudowanych projektów jest duplikacja kodu i to niezależnie od języka. Puppet, jako narzędzie służące do konfigurowania serwerów, jest na to szczególnie podatny. Można się przed tych chronić rozmaitymi metodami: jedne są bez wpływu na czytelność kodu lub wręcz go poprawiają, inne zaś z niewielkim uszczerbkiem na czytelności. Na początek poznamy metodę prostego skracania kodu. Wywołując wiele razy te same typy z tymi samymi parametrami (lub większością), zmniejsza się czytelność kodu. Rozwiązaniem jest na to tzw. resource default statements:
`https://docs.puppet.com/puppet/latest/lang_defaults.html`

10.1.2. Zastrzeżone wyrażenia

Jak każdy język, puppet ma swoje zastrzeżone wyrażenia. Szczegóły w dokumentacji:
`https://docs.puppet.com/puppet/lates/lang_reserved.html`

10.1.3. Komentarze w kodzie

Krótko: `https://docs.puppet.com/puppet/4.9/lang_comments.html`

10.1.4. Więcej niż jedna maszyna

Dotąd pisaliśmy kod - powiedzmy - dla jednej maszyny. A co jeśli chcemy mieć więcej maszyn i każda z inną konfiguracją? Puppet ma tutaj wiele do zaoferowania, ale zaczniemy od najprostszego (nieużywanego już dzisiaj) mechanizmu:
`https://docs.puppet.com/puppet/4.9/lang_node_definitions.html`
Jest to zaszłość z czasów, kiedy puppet nie był tak rozbudowany w swojej funkcjonalności. Nie będziemy tego wykorzystywać na środowiskach, ale z racji swojej prostoty ułatwi to zrozumienie istoty sprawy.

10.2. Ćwiczenia na zajęcia

1. Załóżmy, że mamy 3 hosty: `a.com, b.com, c.com`. Na pierwszym ma być zainstalowany i uruchomiony named, na drugim zainstalowany i uruchomiony nginx, na trzecim httpd. Konfigurację tutaj pomijamy.
2. Utwórz 10 plików z nazwami od `/tmp/p1` do `/tmp/p10`. Każdy niech jest pusty i ma prawa root/root, 744. Wyjątki:
   - niech `/tmp/p4` ma zawartość 4
   - niech `/tmp/p7` ma prawa 777
   - niech `/tmp/p9` jest linkiem do /etc/fstab

   Nie duplikować kodu.

10.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k*10_*imie_nazwisko* i do niego wysłać rozwiązanie poniższych zadań w jednym piku *site.pp*

1. Napisz fakt podający liczbę znaków w `/etc/fstab`, niech będzie dostępny pod zmienną `$fstab_chars`. Analogicznie zrób dla liczby linii. Utwórz plik `/tmp/fstab_stat` z zawartością:

```
1 Fstab stat:
2 num of lines: <number>
3 num of chars: <number>
```

Fakt powinien być dostarczany przez puppeta. Czy da się to wykonać przy jednym przebiegu puppet-apply? Uzasadnij. Czy obecność faktu w systemie przed uruchomieniem puppeta zmieniałaby sytuację? (odpowiedź wpisz w formie komentarza w site.pp)

2. Dla niedużej liczby całkowitej nieujemnej (do kilku tysięcy) sprawdź czy jest liczbą pierwszą czy złożoną, czy ani taką, ani taką. Wypisz na ekran odpowiednią informację (pierwsza, złożona, ani pierwsza, ani złożona).
   Wskazówka: użyć konstrukcji omówionych na liście 9.
   Utrudnienie: badana liczba ma być wczytywana ze zmiennej `$wejscie`. Upewnij się, że pod tą zmienną na pewno jest obiekt, jakiego się spodziewasz.

11. Lista 11

11.1. Plan działania

11.1.1. Własne klasy - wprowadzenie

Dotąd umieszczaliśmy kod w jednym pliku - manifests/site.pp, który bezpośrednio był kompilowany przez puppeta. Takie podejście jest przyjemne gdy się ma jeden host, na którym trzeba zrobić kilka rzeczy. Przy kilkuset hostach i wielu usługach plik kod byłby nieutrzymywalny. Dlatego wprowadzimy klasy. Na początek w ograniczonej formie nadal korzystając z site.pp.
Składnia jest następująca:

```
1 class nazwa_klasy_wraz_ze_sciezka(
2   $lista,
3   $parametrow,
4   $naglowkowych,
5 ){
6   Cialo klasy
7 }
```

Parametry nagłówkowe mogą mieć przypisaną wartość (wtedy można je opcjonalnie nadpisać), ale nie muszą (wtedy są wymagane). A może też ich nie być (w tym przypadku nie wolno im ustawiać wartości). W ciele klasy umieszczamy cały kod - czyli to co do tej pory w site.pp. Swoje klasy będziemy umieszczali w katalogu modules (utworzonym w głównym katalogu w repo, tuż obok manifests).

11.1.2. Jak nazwać klasę i gdzie jej szukać

Klasa nie może mieć dowolnej nazwy. Poza zabronionymi znakami, musi mieć odpowiednio ustawioną ścieżkę. Klasa w pliku:
`modules/modulik/manifests/moja_klasa.pp`
będzie miała nazwę
`modulik::moja_klasa`
klasa w
`modules/srutu/manifests/stara/pipi/kuku.pp`
będzie miała nazwę
`srutu::stara::pipi::kuku` etc.
Czyli pełna ścieżka z dwukropkami zamiast ukośników, z pominięciem manifests, bo wiadomo, że skoro to klasa, to na pewno jest to manifest puppetowy. Stąd też nie da się umieścić klasy w innym katalogu np. `modules/srr/moje_manifesty/klasa.pp`.

11.1.3. A gdzie szukać klasy która nie ma dwukropków?

Klasy postaci `klasa1`, `moja_klasa` itd. to pliki `modules/klasa1/manifests/init.pp` oraz `modules/moja_klasa/manifests/init.pp` odpowiednio. Na początku wydaje się to absurdalne. Jednak przeczytaj dokładnie poprzedni rozdział, i okaże się, że jednak ma to sens.

11.1.4. Jak uruchomić swoją klasę

Jest to bardzo proste - w pliku manifests/site.pp wpisujemy:

```
1 include srutu::stara::pipi::kuku
```

To w przypadku jak klasa nie ma żadnych wymaganych parametrów nagłówkowych.

11.1.5. Moja klasa ma parametry nagłówkowe lub nie lubię słowa include

Wtedy wołając klasę trzeba użyć takiej notacji:

```
1 class{'srutu::stara::pipi::kuku':
2 }
```

To też pozwala załączyć klasę mającą wymagane parametry:

```
1 class{'srutu::stara::pipi::kuku':
2   wymagany_parametr => 500,
3 }
```

Daje się go bez dolara na początku.

11.1.6. Czy z poziomu mojej klasy mogę uruchomić inną moją klasę?

Tak, dokładnie tak jak powyżej.

11.1.7. Dokumentacja

Na początek tylko te sekcje:
`https://docs.puppet.com/puppet/4.9/lang_classes.html#syntax`
`https://docs.puppet.com/puppet/4.9/lang_classes.html#location`
`https://docs.puppet.com/puppet/4.9/lang_classes.html#using-include`
`https://docs.puppet.com/puppet/4.9/lang_classes.html#using-require`

11.2. Ćwiczenia na zajęcia

1. Utworzyć trzy klasy:
   - `lista10::cw1::httpd`
   - `lista10::cw1::postfix`
   - `lista10::cw1::sshd`

   Każda z tych klas niech instaluje odpowieni pakiet i uruchamia odpowiednią usługę. Załączyć je wszystkie w site.pp.

11.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k11_imie_nazwisko* (na podstawie brancha master), i do niego wysłać rozwiązanie poniższych zadań

1. Utwórz klasę `moje_zadania`, załącz ją w site.pp.
2. Utwórz klasy `moje_zadania::lista_<numer>::zadanie_<numer>` wybierając 7 swoich dowolnych zaliczonych zadań domowych.
3. Załącz te klasy w `moje_zadania`.

12. Lista 12

12.1. Plan działania

12.1.1. Własne klasy - ciąg dalszy

Poprzednio nauczyliśmy się pisać i nazywać podstawowe klasy. Na tym możliwości puppeta się nie kończą. Nauczymy się pracować ze zmiennymi w sposób nieco bardziej zaawansowany oraz te zmienne pożyczać z innych klas.

12.1.2. Dziedziczenie parametrów

Nieco naiwnie tłumacząc: każda klasa może pożyczyć parametry wraz z wartościami od innej klasy, w taki sposób, że ustawienie wartości zmiennej w klasie(-matce) powoduje, że zmienna pod tą samą nazwą i z tą samą wartością dostępna jest w klasie(-dziecku), która te parametry pożycza. Na dodatek nie trzeba tych zmiennych w żaden sposób inicjować w klasie(-dziecku) - po prostu dolar i nazwa, i mamy wartość. Opisując to bardziej technicznie: klasa potomna może dziedziczyć parametry po nadrzędnej klasie bez nadmiarowego inicjowania. U osób, które się dopiero uczą puppeta wprowadza to czasem zamęt, bo nagle pojawia się zmienna znikąd, i co gorsza ma ustaloną prawidłową wartość! Zapisujemy to w ten sposób:

```
1 class sisi::klasa1(
2 ) inherits sisi::klasa0 {
3   notice($zmienna)
4 }
```

```
1 class sisi::klasa0(
2 ) {
3   $zmienna = 700
4 }
```

12.1.3. Dziedziczenie po wielu klasach

Jedna klasa może dziedziczyć parametry tylko po jednej innej klasie - czyli nie da się napisać dwóch „inheritów”. Ale jest na to sposób - dziedziczenie kaskadowe: jest klasa0 z parametrami, klasa1 dzidziczy po klasie0, klasa2 dziedziczy po klasie1. W ten sposób klasa0 ma tylko swoje parametry, klasa1 ma parametry swoje + parametry od klasy0, klasa2 ma parametry swoje i pozostałych dwóch klas.

12.1.4. A co jeśli zmienne będą niedostępne?

Tak się nie zdarzy (z prawdopodobieństwem $> 99\%$ - mogą być bugi w kodzie). Puppet tak układa wykonanie klas, żeby zmienne były zadeklarowane zanim wykonają się klasy potomne.

12.1.5. A co jeśli ktoś w klasie nadrzędnej lub potomnej nadpisze zmienną?

Puppet na to nie pozwoli, bo zmiennych raz ustawionych nie wolno nadpisywać (pomijamy tutaj lambdy, w których - jak wiecie z poprzednich list - można to robić).

12.1.6. Czy można jeszcze jakieś cuda z tymi zmiennymi wyprawiać?

Można, z użyciem hiery, ale to wprowadzimy kiedy indziej.

12.1.7. Wymuszanie kolejności wykonania kodu na klasach

Oczywiście da się wymusić kolejność wykonania na klasach (tylko trzeba uważać na dziedziczenie - mogą powstawać cykle). Załóżmy, że mamy klasy:

```
class sisi::klasa1(
) {
  file{'/tmp/plik':
    ensure => file,
  }
}
```

```
class sisi::klasa0(
) {
  service{'postfix':
    ensure => running,
  }
}
```

I chcemy, żeby klasa1 wykonała się przed klasą0. Wówczas możemy to zrobić na kilka sposobów:

1. W klasie nadrzędnej „includującej"(pośrednio lub bezpośrednio) obie te klasy, lub w site.pp jeśli są one tam „zaincludowane":

   ```
   Class['sisi::klasa1'] ~>
   Class['sisi::klasa0']
   ```

2. Jeśli nie mamy klasy nadrzędnej (w przypadku użycia hiery) lub jeśli nie chcemy powyższego robić z różnych powodów: do klasy0 dodajemy linijkę

   ```
   require sisi::klasa1
   ```

   Wadą tego rozwiązania jest brak powiadomień na obiektach - czyli de facto mamy odpowiednik `->` a nie `~>`. Jeśli chcemy jednak mieć i includa, i powiadomienia, wtedy trzeba się o to ręcznie zatroszczyć dodając w klasie0 linijkę:

   ```
   Class['sisi::klasa1'] ~>
   Service['postfix']
   ```

   To po co to require - w tym konkretnym przypadku jest kompletnie nadmiarowy. Stosuje się go jeśli chcemy żeby obiekty z klasy1 się zaaplikowały przed obiektami w klasie0. I tylko tyle. Jeśli potrzebujemy powiadomień na klasach lub między klasami a typami - trzeba użyć powyższej notacji.
   Słabszym warunkiem od require jest include - pozwala na współbieżne wykonanie klas.

3. Kolejną metodą jest użycie notify/subscribe na konkretnych obiektach (omawialiśmy to na poprzednich listach) + include (jeśli klasa nie jest nigdzie indziej załączona). Require też można użyć, ale nic nam nie da, a tylko stracimy na wydajności.

4. jest jeszcze jedna metoda, ale jest ona na tyle istotna, że zasługuje na osobny podrozdział.

12.1.8. Kolejność wykonywania w puppecie - ostatnia metoda

Są to tzw. stage. Do nich można przypisać wyłącznie klasy, nie da się tego zrobić na typach zdefiniowanych. Dlatego też nie mogliśmy tego zrobić na etapie pracy wyłącznie w site.pp - nie mieliśmy klasy. Metoda różni się od tych dotychczas wprowadzonych tym, że jest łatwiejsza we wprowadzeniu zależności w dużej ilości kodu i modułów. Pierwszym z brzegu zastosowaniem jest konfiguracja repozytoriów przed instalacją jakiegokolwiek pakietu w jakiejkolwiek klasie (napisanej przez kogokolwiek). Dotąd wszystkie nasze zmiany były aplikowane w domyślnym stage'u main. Możemy sobie natomiast zdefiniować inne, i zdefiniować zależności między stage'ami (za pomocą metod już poznanych). Szczegóły w dokumentacji:
`https://docs.puppet.com/puppet/latest/lang_run_stages.html`

12.1.9. Kilka słów o standardach w kodzie

Dobre praktyki będziemy wprowadzać powoli. Dotąd nauczyliśmy się kończyć każdą linię w hashu i tablicy przecinkiem. Teraz czas na klasy - listy parametrów nagłówkowych też dobrze kończyć za każdym razem przecinkiem (90% błędów to brak przecinka). Druga zasada: jedna klasa - jeden plik, i wszystko prawidłowo nazwane. Da się utworzyć wiele klas w jednym pliku, ale jest to rozwiązanie na tyle kłopotliwe, że jest niezalecane. Czytanie potem takiego kodu to horror.

12.1.10. Dokumentacja

Na początek tylko te sekcje:
`https://docs.puppet.com/puppet/4.9/lang_classes.html`
`https://docs.puppet.com/puppet/latest/lang_run_stages.html`

12.2. Ćwiczenia na zajęcia

1. Upewnij się, że w systemie jest zainstalowany pakiet git. W przypadku jego instalacji, niech puppet restartuje usługę postfix. W celu wymuszenia kolejności użyj stage'y: `st_git` oraz `st_postfix`. Utwórz pusty plik `/tmp/plik`, niech przy zmianie zawartości restartowana jest usługa postfix. Nie przypisuj tego pliku do żadnego stage'a.

12.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k12_imie_nazwisko* na podstawie brancha master, i do niego wysłać rozwiązanie poniższych zadań

1. Dodaj do systemu operacyjnego repozytorium remi:
   `https://rpms.remirepo.net/`.
   Zainstaluj pakiety: php71-php-fpm oraz nginx. Reinstalacja php71-php-fpm niech wymusza restart nginxa. Cała procedura ma być wykonana w **jednym** przebiegu puppeta.

13. Lista 13

13.1. Plan działania

13.1.1. Więcej o typach zdefiniowanych

Na chwilę przerywamy pracę z klasami, ale tylko pozornie. W puppecie istnieją bardzo zbliżone do klas typy obiektów - typy zdefiniowane. Do tej pory używaliśmy ich niemal jak funkcji, tylko ze specyficznie podanymi parametrami - w formie hasha. Tym razem nauczymy się sami pisać takie typy, póki co z dość ograniczoną funkcjonalnością.

13.1.2. Własny typ zdefiniowany

Załóżmy, że chcemy mieć typ o nazwie `moj_typ`, wtedy będzie to następująca konstrukcja:

```
1 define moj_typ(
2 ) {
3   # jakis kod
4 }
```

Czyli zamiast class mamy define, reszta tak samo, nawet nazewnictwo i ścieżki tak samo.

13.1.3. Różnice między klasą a typem

I klasa i typ to wiaderko na kod puppetowy. Dokładnie ten sam kod puppetowy można wpisać w typ i w klasę z paroma różnicami:

- klasa może dziedziczyć, typ nie
- klasę i typ woła się inaczej:

```
1 Class['tak::wolamy::klase']
2 class{'tak::wolamy::klase':
3   z_takim => parametrem,
4 }
5
6 Tak::Wolamy::Typ['z takim tytulem']
7 tak::wolamy::typ{'z takim tytulem':
8   z_takim => parametrem,
9 }
```

  Zauważ, że w odwołaniu do obiektu (na nim np. ustawiamy kolejność) używamy nazw zaczynających się od dużych liter, natomiast wywołanie typu/klasy z małej (na tym z kolei definiujemy ewentualne parametry)

- klasę na jednym hoście można zawołać tylko raz, typ wiele razy, co już stanowi wymuszenie unikalności - typ musi być unikalny ze względu na parę nazwa + tytuł, klasa zaś musi być unikalna w całości.

13.1.4. Wymuszanie kolejności na własnych typach zdefiniowanych

W typie można zaincludować klasę (tylko ostrożnie, bo dwukrotne zawołanie tego typu może spowodować błąd pt. klasa dwa razy zawołana), ale bezpieczniej jest użyć tutaj require (wiele typów i obiektów może wymagać innej klasy)
Z kolei nie da się zrobić „`require~moj_typ`", bo typ wymaga jeszcze tytułu - trzeba użyć dotychczasowych metod zawołać typ z odpowiednimi meta-parametrami (before/require). Drugie podejście to obudowanie takiego typu w klasę (szumna nazwa - po prostu zrobić klasę z tym typem w środku) i wołać wtedy tę klasę.
Jak już ostatnio sobie powiedzieliśmy - mechanizm stage jest dostępny tylko dla klas. Żeby jakiś typ uruchomić w konkretnym stage'u, trzeba obudować go w klasę, i na niej ustawić stage.

13.1.5. Kolejne kilka słów o standardach w kodzie

Należy być bardzo ostrożnym przy używaniu mechanizmu stage - potrafi wygenerować taki cykl w zależnościach, że rozwiązanie trwa często kilka godzin.

13.1.6. Typ zdefiniowany a iteracja

W puppecie w wersji 3 i wcześniejszych nie było czegoś takiego jak... iteracja (sic!). Żeby zrobić pętlę w puppecie trzeba było nierzadko pozrywać wszystkie podłogi i przewrócić kod do góry nogami, albo też pisać własne funkcje w Rubym. Ale jest jeden hack, którym można sobie zrobić taką pętlę. Co prawda przypomina to czesanie włosów mikserem, ale da się, i działa.

13.1.7. Dokumentacja

`https://docs.puppet.com/puppet/4.9/lang_defined_types.html`

13.2. Ćwiczenia na zajęcia

1. Utwórz typ `zainstaluj_i_zaloguj`, który instaluje zadany pakiet i do pliku `/tmp/instalacja.log` zaloguje datę, godzinę i nazwę pakietu. Niech typ za każdym razem upewnia się, że usługa postfix jest podniesiona.

2. Upewnij się, że w systemie jest zainstalowany pakiet git. W przypadku jego instalacji, niech puppet restartuje usługę postfix. W celu wymuszenia kolejności użyj stage'y: `st_git` oraz `st_postfix`. Utwórz pusty plik `/tmp/plik`, niech przy zmianie zawartości restartowana jest usługa postfix. Nie przypisuj tego pliku do żadnego stage'a.

13.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k*13_*imie_nazwisko* na podstawie brancha master, i do niego wysłać rozwiązanie poniższych zadań

1. Dany jest dowolny niezagnieżdżony hash (załóżmy dla uproszczenia, że klucz i wartość są czystymi stringami, bez liczb i znaków specjalnych). Wypisz na ekran wartości wszystkich kluczy w hashu, każdą z osobna, bez wypisywania nazw kluczy.
   Utrudnienie: nie wolno używać żadnej funkcji dostarczającej iteracji, nie wolno konwertować hasha na inne typy danych, do wypisania użyć funkcji notice.
   Ułatwienie: Można sobie wyłuskać listę kluczy w postaci tablicy ulubioną funkcją poznaną na którychkolwiek zajęciach
   Wskazówka: użyć własnego typu zdefiniowanego

14. Lista 14

14.1. Plan działania

14.1.1. Dodawanie plików

Dotąd dodawaliśmy pliki używając typu `file` i opcji `content`. Ewentualnie używając typu `exec`. Wygoda tego rozwiązania jest dość dyskusyjna, prawda? W taki sposób ustawia się zawartość niezwykle krótkich plików. W typowych zastosowaniach robi się to inaczej. W tym celu wprowadzimy sobie kilka funkcjonalności:

- **template** typu **erb** plik umieszczamy w `modules/<modul>/templates/template.erb`. Wtedy taki template używamy:

```
1 file{'/tmp/plik':
2   ensure  => file,
3   content => template('<modul>/template.erb')
4 }
```

  Zwróć uwagę, że tak jak w przypadku manifestów w ścieżce pomijamy `manifests`, tak w templatkach słowo `templates`
  Doc: `https://docs.puppet.com/puppet/4.9/lang_template_erb.html`

- **inline_template** - szczególny przypadek erb. Kod, który umieścilibyśmy normalnie w pliku `.erb`, umieszczamy w manifeście:

```
1 file{'/tmp/plik':
2   ensure  => file,
3   content => inline_template('kod erb'),
4 }
```

  Doc: `https://docs.puppet.com/puppet/latest/function.html#inlinetemplate`

- **template** typu **epp** jak wyżej - plik umieszczamy w `modules/<modul>/templates/template.epp`. Wtedy taki template używamy:

```
1 file{'/tmp/plik':
2   ensure  => file,
3   content => epp('<modul>/template.epp')
4 }
```

  Zwróć uwagę, że tak jak w przypadku manifestów w ścieżce pomijamy `manifests`, tak w templatkach słowo `templates`
  Doc: `https://docs.puppet.com/puppet/4.9/lang_template_epp.html`

- **file** plik umieszczamy w `modules/<modul>/files/plik.txt`. Wtedy taki plik używamy:

```
1 file{'/tmp/plik':
2   ensure => file,
3   source => 'puppet:///modules/<modul>/plik.txt'
4 }
```

  Zwróć uwagę, że tak jak w przypadku manifestów w ścieżce pomijamy `manifests`, tak w plikach pomijamy słowo `files`
  Doc: `https://docs.puppet.com/puppet/latest/type.html#file-attribute-source`

- **file_line** - dla porządku dodaję informację o tym w tym miejscu - metoda której jeszcze nie umiemy użyć. Jest to funkcja z modułu stdlib. Jej użycie polega na tym, że wybieramy sobie plik niedostarczany przez puppeta (przychodzący z rpmki, istniejący domyślnie w systemie itp.) i upewniamy się że istnieje w nim linia.
  Doc: `https://forge.puppet.com/puppetlabs/stdlib#file_line`

14.1.2. Co do czego

- **file** - najprostsza metoda na dostarczenie pliku lub całego katalogu rekursywnie - po prostu dane są kopiowane z kodu puppetowego
- **template typu erb lub epp** - standardowa metoda na dostarczanie konfiguracji w postaci plików, lecz wtedy, gdy potrzebujemy jakiś fragment dostarczać ze zmiennej (lista hostów, port, nazwa domenowa itp.)
- **inline_template** - prostsza metoda na użycie template'a, pomocna dla bardzo krótkich fragmentów
- **file_line** - metoda na dopisanie jakiegoś fragmentu do pliku istniejącego w systemie (np. mechanizm versionlock w YUMie)

14.1.3. Sztuczki

- używając erbów dostajemy za darmo obsługę funkcji i metod rubiowych, w tym iterację. Oczywiście należy unikać pisania rozbudowanych skryptów w templatkach do plików, ale proste rzeczy typu przetwarzanie struktur danych - czemu nie.
- umieszczając katalog w `<modul>/files/katalog` wraz ze strukturą katalogowo-plikową w nim, można rekursywnie przekopiować cały katalog w wybrane miejsce - w typie file wystarczy użyć opcji `recursive` i `source` ustawić na katalog.

14.2. Ćwiczenia na zajęcia

1. Używając pliku erb utwórz plik /tmp/fqdn, w którym będzie wpisana nazwa domenowa hosta, na którym puppet chodzi
2. Zdefiniuj sobie jakiegoś hasha z pięcioma kluczami o wartościach typu string. Używając erba wypisz do pliku hasha w postaci: `<klucz>: <wartosc>`. Wskazówka: użyj funkcji map w pliku erb
3. Zrób jakiś moduł, w nim w katalogu `files` jakąś strukturę katalogów i plików. Używając sztuczki skopiuj puppetem te katalogi do `/tmp`. Dodaj odpowiednią opcję, dzięki której puppet będzie usuwać z katalogu pliki i katalogi wstawione przez kogoś innego.

14.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k14_imie_nazwisko* na podstawie brancha master, i do niego wysłać rozwiązanie poniższych zadań

1. Dla niedużej liczby całkowitej nieujemnej wypisz na ekran czy liczba jest pierwsza, czy złożona, czy ani taka, ani taka. Niech kod będzie umieszczony w klasie o nazwie `lista14_zadanie1`

   Utrudnienie: cały kod ma być w pliku erb.

2. Utwórz typ zdefiniowany `lista14_zadanie2` który:

   - jako tytuł przyjmuje ciąg znaków, który będziemy badać
   - do pliku `/tmp/string_stat` wpisze informację ile w stringu jest samogłosek, ile spółgłosek, ile cyfr, zamieni wszystkie litery na duże, zamieni wszystkie litery na małe (zakładamy, że dostępne literki to te 26 literek + 10 cyfr) w postaci:

   ```
   1 samoglosek:
   2 spolglosek:
   3 cyfr:
   4 znakow w sumie:
   5 string duzymi literami:
   6 string malymi literami:
   ```

3. Utwórz klasę `lista14_zadanie3`, a w niej użyj faktu z listy 6 zadanie domowe 1 (rozmiar katalogów w /etc). Wpisz zawartość tego faktu do pliku `/tmp/fakt_dump` w postaci:

```
<katalog>: <rozmiar>
```

   Należy tu użyć erba.

4. Utwórz klasę `lista14_zadanie4` która przyjmuje parametr liczba typu Integer. Następnie w erbie sprawdza czy jest podzielna przez 7, jeśli nie, to używając funkcji notice wypisuje na ekran tę liczbę, jeśli tak, wywołuje funkcję fail z tą liczbą w treści.

15. Lista 15

15.1. Plan działania

15.1.1. Hiera

Do tej pory załączaliśmy klasy w site.pp, tam też uczyliśmy się przypisywać klasy do poszczególnych hostów. Oczywiście obecnie się tego nie robi w ten sposób. W tym celu używa się tzw. hiery. Jest to narzędzie do definiowania wartości zmiennych, przypisywania klas do hostów itd. Będziemy używali hiery z plikami yaml, które są dużo prostsze w edycji, niż JSONy.
Doc: `https://docs.puppet.com/hiera/latest/`

15.1.2. Konfiguracja hiery

Na początek, żeby użyć hiery, należy ją skonfigurować. Robi się to w prostym pliku hiera.yaml. Nasz aliast `puppet-apply` ma tak ustawione miejsce konfiguracji hiery: `--hiera_config=/vagrant/hieradata/hiera.yaml`. I w tym pliku należy umieścić konfigurację jak niżej:

```
1  :backends:
2    - yaml
3
4  :hierarchy:
5    - host/%{fqdn}
6    - domain/%{domain}
7    - common
8
9  :yaml:
10    :datadir: /vagrant/hieradata
11
12 :merge_behavior: deeper
```

Listing 21: hiera.yaml

Chwilowo taki konfig nam wystarczy na potrzeby nauki. Co on mówi? Ano tyle że spodziewa się plików `host/<fqdn>.yaml` `domain/<domain>.yaml` oraz `common.yaml`, przy czym `domain` oraz `fqdn` to fakty z factera. Resztę parametrów można sobie doczytać w dokumentacji.

Ostatnią rzeczą, jaką musimy zrobić, to w site.pp załączyć użycie hiery wstawiając linijkę:

```
1 hiera_include(classes)
```

Od tej pory podczas wykonywania `site.pp` będzie odpytywana hiera o tablicę `classes`, i tam będzie się spodziewać nazw klas.

15.1.3. Załączanie klasy w hierze

Załóżmy że mamy klasę `moja_klasa`, czyli plik `modules/moja_klasa/manifests/init.pp`. Umiemy tę klasę załączać w innej klasie lub w `site.pp`. W hierze załączamy ją w odpowiednim pliku yaml wpisując:

```
1 classes:
2   - moja_klasa
```

ewentualnie jeśli w pliku jest już tablica classes, to dopisując do tej tablicy.

15.1.4. Zmienne w hierze

Wiemy, że typ zdefiniowany i klasa mogą przyjmować różne parametry. Można je ustawić domyślnie w klasie, podczas wołania klasy w innej klasie, a trzecią metodą jest ustawienie ich w hierze. Załóżmy, że mamy klasę:

```
1 class moja_klasa::kat1::kat2::klasa1(
2   $parametr1 = '12345',
3   $parametr2,
4 ){
5
6 }
```

Jak załączymy tę klasę w hierze używając:

```
1 classes:
2   - moja_klasa::kat1::kat2::klasa1
```

Wówczas puppet krzyknie, że brakuje mu wartości dla zmiennej `parametr2`. Wystarczy dodać w tym samym pliku yaml:

```
1 moja_klasa::kat1::kat2::klasa1::parametr2: 'zxcvbnm'
```

i problem rozwiązany. Zwróć uwagę na to, że nazwa parametru to pełna ścieżka: nazwa klasy + :: + nazwa parametru. Taką samą metodą możemy nadpisać `parametr1` jeśli jego wartość nam się nie podoba.

15.1.5. O hierarchii słów kilka

Hiera jest strukturą hierarchiczną - bazując na przykładzie naszego konfigu, załóżmy że mamy poniższą strukturę plików:

```
1  hieradata/common.yaml
2  hieradata/domain/env.yaml
3  hieradata/domain/localdomain.yaml
4  hieradata/domain/com.yaml
5  hieradata/host/host1.env.yaml
6  hieradata/host/host1.env.yaml
7  hieradata/host/localhost.localdomain.yaml
8  hieradata/host/localhost.localdomain.yaml
9  hieradata/host/mooo.com.yaml
10 hieradata/host/pipi.com.yaml
```

Zgodnie z tym, co mamy napisane w hiera.yaml, to najpierw przeszukane zostaną pliki w `hieradata/host/*.yaml`, potem w `hieradata/domain/*.yaml`, a na końcu plik `hieradata/common.yaml`. Dzięki temu możemy ustawić/nadpisać jakiś parametr lub załączyć klasę per host, per domena lub wszystkim hostom. W szczególności jeśli wartość parametru jest ustawiona w każdym z tych plików, to brana jest wartość z pierwszego pliku w kolejności listy w hiera.yaml (czyli absolutne pierwszeństwo mają definicje per host, a najniższe w `common.yaml`)
Doc: `https://docs.puppet.com/hiera/3.3/hierarchy.html`

15.1.6. Własne moduły

Kolejną rzeczą, jaka nas interesuje to własne moduły, czyli samodzielny zespół kodu puppetowego, który robi jakąś konkretną rzecz, na przykład: moduł do instalacji i konfiguracji httpd, moduł do instalacji i konfiguracji nginxa itp. Jest to nic innego jak zespół klas + konfiguracja w templatkach lub plikach.

15.1.7. Dobre praktyki przy pisaniu modułów

Zasadniczą praktyką jest rozbijanie procesu na trzy etapy: instalacja pakietów, konfiguracja, obsługa usługi. Załóżmy że mamy moduł `moj_modul` i ma on coś skonfigurować. Wówczas potrzebujemy czterech klas: `moj_modul::package`, `moj_modul::config`, `moj_modul::service` i `moj_modul` gdzie wołamy trzy pierwsze klasy ustawiając przy tym kolejność i powiadomienia. W pliku `moj_modul/manifests/init.pp` zasadniczo powinna być tylko logika, żadnych obiektów wprowadzających bezpośrednio zmiany na hostach - od tego są pozostałe trzy klasy. W szczególności

można rozdzielić każdą z nich na podklasy, gdy uznamy że na przykład klasa z konfiguracją jest zbyt opasła i da się to podzielić na kilka fragmentów mających ze sobą coś wspólnego.

15.2. Ćwiczenia na zajęcia

1. Utwórz 3 klasy: `klasa1`, `klasa2::podklasa1`, `klasa2::podklasa2`. W hierze załącz pierwszą klasę na wszystkich hostach, drugą tylko na hostach w domenie `zozo`, trzecią tylko na hoście `host1.zozo`. Przetestuj czy każde z ustawień działa.

2. Załóżmy że mamy powyższe klasy, każda załączona na wszystkich hostach, w każdej jest parametr o nazwie `par` ustawiony na wartość `qwertyuiop`. W każdej z klas jest wywołanie funkcji notice wypisujące na ekran tę zmienną. Używając hiery zmień wartość parametru na `111` dla hosta `pipi.zozo`, na `222` dla wszystkich hostów w domenie `zozo` i na `999` dla pozostałych.

15.3. Ćwiczenia do samodzielnego wykonania

Utworzyć branch *k*15_*imie_nazwisko* na podstawie brancha master, i do niego wysłać rozwiązanie poniższych zadań

1. napisz moduł `uni_install` instalujący i uruchamiający dokładnie jedną z poniższych usług (przyjmujemy że to jest lista nazw tych usług):

   - postfix
   - nginx
   - bind
   - redis
   - autofs

   Przyjmujemy, że domyślna konfiguracja każdej z tych usług jest dla nas wystarczająca. Nazwa usługi do zainstalowania powinna być dostępna pod zmienną `usluga` w klasie `uni_install /manifests/init.pp`. Moduł powinien tworzyć plik `/tmp/uni_install.info` o zawartości:

   ```
   = Modul uni_install =
   nazwa uslugi: <nazwa>
   zainstalowany pakiet: <nazwa>
   uruchomiono usluge: <nazwa>
   ```

   Utrudnienie: moduł ma być gotowy na instalację każdej z tych usług.
   Utrudnienie + wskazówka: w klasie uni_install należy zdefiniować hasha mapującego nazwę z listy na nazwę pakietu oraz nazwę usługi dla każdego przypadku.
   Utrudnienie: moduł powinien wyświetlić błąd na ekran w przypadku ustawienia zmiennej `usluga` na jakąkolwiek wartość spoza listy. W kodzie błędu powinna znajdować się dozwolona lista usług.
   Utrudnienie: moduł musi być gotowy na dopisanie dodatkowych usług do hasha
   Ułatwienie: nie należy się przejmować tym, że przy działającym apaczu, nginix się nie chce podnieść i puppet rzuca błędem. Zakładamy że klasa jest uruchamiana na czystym systemie.
   Utrudnienie: całość ma się wykonać w jednym przebiegu puppeta

2. napisz moduł `uni_install_multi` o parametrach jak wyżej, z jedną zasadniczą zmianą: moduł ma przyjmować listę usług do instalacji, będącą podzbiorem listy zdefiniowanej w zadaniu poprzednim. W szczególności ma instalować wszystkie.
   Utrudnienie: nie wolno używać funkcji dostarczających iteracji

3. załóżmy że mamy moduł `uni_install_multi` o parametrach j.w. Czy możliwa jest taka jego implementacja, że `uni_install_multi::package`, `uni_install_multi::config` oraz `uni_install_multi::service` są klasami? Czy usunięcie warunku o nieużywaniu funkcji dostarczających iteracji coś w tym względzie zmieni?

16. Lista 16

16.1. Plan działania

16.1.1. Własne funkcje w rubym

Do tej pory używaliśmy domyślnych funkcji puppetowych. Ale można pisać też własne. Wystarczy się tylko nauczyć Rubiego :)
Doc: `https://docs.puppet.com/guides/custom_functions.html`

16.1.2. O faktach puppetowych raz jeszcze

Żeby nie było tak prosto, fakty można dostarczać w równie przyjemny sposób co funkcje puppetowe . Okazuje się, że ma to jedną dodatkową zaletę - fakt puppetowy skopiuje się na hosta, wykona i zapisze swoje wyniki w facterze, i to w jednym przebiegu. Są też inne metody omówione w dokumentacji.
Doc: `https://docs.puppet.com/facter/3.6/custom_facts.html#loading-custom-facts`

16.1.3. Cudze moduły puppetowe

Nikt przy zdrowych zmysłach nie pisze wszystkich modułów puppetowych sam. Co prawda są tacy ludzie, ale nie spełniają warunku z poprzedniego zdania. Istnieje ogólnie dostępna biblioteka open-sourcowych modułów pod adresem `http://forge.puppetlabs.com`, gdzie można znaleźć moduły do bardzo wielu rzeczy. To tylko z pozoru wygląda różowo - praktyka bywa taka, że czasem moduł niekoniecznie dostarcza tego, co chcemy, albo nie w takiej formie jakiej chcemy. Lub też jest kiepskiej jakości. Wtedy rozsądnym podejściem jest dopisać własny kawałek kodu do modułu i zgłosić pull requesta, lub dostosować swoje wizje konfiguracji do możliwości modułu. Innym podejściem jest skorzystać z fragmentu modułu i dopisać własne ópakowanie". Absolutną ostatecznością powinno być pisanie swoich modułów.

16.1.4. Zmiana metody dostarczania kodu

Każdy powinien sobie stworzyć na podstawie brancha master swojego brancha o nazwie: `kod_<imie>_<nazwisko>`. Od tej pory cały kod, który będziecie pisać ma się znajdować w tym jednym branchu.

16.2. Ćwiczenia na zajęcia

1. Przygotuj sobie branch `kod_<imie>_<nazwisko>`. Skonfiguruj na nim hierę i utwórz klasę która będzie zawsze aktualizować puppet-agenta do najnowszej wersji. Załącz tę klasę w hierze dla wszystkich hostów. Na początek napisz własny prosty moduł.
2. Używając modułu z forge'a skonfiguruj sobie strefę czasową (również dla każdego hosta)
3. Na głównej stronie forge'a znajdują się informacje jak skonfigurować sobie ulubiony edytor do kodu puppetowego, żeby ładnie kolorował składnię puppeta. Zalecany jest vim. W przypadku problemów z obsługą vima - przejść kurs o nazwie vim tutor.

16.3. Ćwiczenia do samodzielnego wykonania

Zadania nie będą sprawdzane - należy sobie przygotować środowisko do kolejnych zajęć.

1. Zainstalować z forge'a moduł stdlib. Zapoznać się z dokumentacją.
2. Zainstalować z forge'a moduł concat. Zapoznać się z dokumentacją.
3. Zainstalować z forge'a moduł do nginxa (puppet/nginx). Przejrzeć dokumentację.
4. Zainstalować z forge'a moduł do apacza (puppetlabs/apache). Przejrzeć dokumentację.
5. Napisz swój fakt w rubym, który pod zmienną 'moje_imie' będzie przechowywać Twoje imię. Fakt niech będzie dostępny w module k_15_zad

17. Lista 17

17.1. Plan działania

17.1.1. Powrót do dziedziczenia parametrów - klasy params.pp

Przyjmuje się, że w modułach klasy params.pp służą do definiowania wartości parametrów per system operacyjny itp. Na przykład w module ustawiającego Binda klasa pod zmienną `$package` zwraca dla jednego systemu operacyjnego `bind9`, dla drugiego `named`, dla trzeciego `named-server` itd. Dzięki temu unikamy rozbudowanej logiki na parametrach w klasach, gdzie wywołuje się moduły, typy, klasy itd.

17.1.2. Struktura kodu puppetowego

Podstawową strukturą kodu puppetowego jest:

- moduły niskiego poziomu
- moduły wysokiego poziomu
- profile
- role
- hiera

Co to oznacza w praktyce? Moduły niskiego poziomu to zwykłe moduły konfigurujące usługi. Moduły wyższego poziomu opakowują moduły niskiego poziomu w dodatkową ustandaryzowaną funkcjonalność - zakładają pewne globalne parametry, które definiuje się w hierze itp. Moduły niskiego i wysokiego poziomu zbiera się w profile, np. profil ELK instaluje Elasticsearch, Logstasha i Kibanę, z odpowiednią konfiguracją. Profil accounts będzie ustawiać użytkowników i hasła, które przyjdą z hiery. Profil web_frontend instaluje apacha, z odpowiednią konfiguracją, dodaje ustawienia selinuxa itd.
Jeszcze wyżej są role - np. host o roli gitlab będzie zawierał profile z instalacją gitlaba, ustawieniem kont i kluczy ssh, ustawieniem firewalla, redisa, reverse proxy, backupów, monitoringu. Albo rola cloud_reverse_proxy to host, na którym skonfigurowany ma być nginx, haproxy, klucze ssh, selinux, firewall, backupy, monitoring, pomiary.
Role i profile podłączamy w hierze per host/grupa hostów/domena.
Doc: `https://docs.puppet.com/pe/2017.1/r_n_p_full_example.html`

17.2. Lint

Słowo, którego nie lubi żadna osoba pisząca kod w puppecie, ale doskonale wie, że bez niego zginie. Jest to narzędzie pilnujące czystości i czytelności kodu. Do vagranta dołączony został alias
`puppet-check`
który wykonuje sprawdzenie kodu. Odtąd macie obowiązek na kursie go używać i lint nie może zgłaszać żadnych uwag do kodu
Uzupełnieniem puppet-linta jest yaml-lint, który sprawdza składnię plików yaml.
Doc: `http://puppet-lint.com/`

17.3. Ćwiczenia na zajęcia

Ćwiczenia należy wykonać używając swojego brancha `kod_<imie>_<nazwisko>`

1. Napisać sobie moduł instalujący puppet-lint oraz yaml-lint (wymaga paczek ruby i ruby-devel w systemie)
2. Używając modułu z forge'a ustawić w systemie plik /etc/hosts

3. Napisać klasę `l17_zad3::klasa1`, która instaluje pakiet vim. Napisać klasę `l17_zad2::klasa2`, która również instaluje pakiet vim (sic!). Załączyć obie klasy w hierze. Używając odpowiedniej funkcji z stdliba rozwiązać otrzymany błąd omijając ograniczenie unikalności wywołanych typów.

4. Załóżmy, że mamy usługę, która w katalogu /etc/pipi spodziewa się pliku pipi.conf. Katalog ma zawierać tylko ten plik, puppet ma usuwać inne pliki. Plik ten składa się z trzech części:

   (a) początek:

   ```
   1 Service pipi
   2 Load module pi
   3 Load module ci
   ```

   (b) konfiguracja listy hostów

   ```
   1 Hosts begin ->
   2   Host: a.blue
   3   Mode: default
   4   Encryption: no
   5
   6   Host: b.blue
   7   Mode: default
   8   Encryption: no
   9 Hosts end <-
   ```

   (c) oraz końcówki:

   ```
   1 On stop notify hosts @@hosts_list
   ```

   Zadanie:

   (a) napisać moduł instalujący pakiet (jako że pipi nie ma, to weźmy git), następnie dostarczyć konfigurację, i na koniec zrestartować usługę (tutaj też zamiast pipi weźmiemy np. postfix)

   (b) Przy zmianie konfiguracji, usługa ma być restartowana

   (c) Moduł ma działać tak, żeby użytkownik sobie zdefiniował taką konfigurację dla takiej liczby hostów, jaką chce.

   (d) Należy w module użyć klasy lub typu zdefiniowanego, który przyjmie odpowiednie parametry

   (e) Konfiguracja w odcinku drugim ma być posortowana po hostach w porządku leksykograficznym

   (f) Wskazówka: sprawę ułatwi moduł concat i w miarę potrzeb stdlib

   (g) Zakładamy, że mode: default oraz encryption: no są domyślnymi parametrami, których użytkownik nie musi podawać, ale może nadpisać wartości na dowolny string. Wymaganym parametrem jest wyłącznie host

   (h) Napisać plik README.md ze skrótową dokumentacją

   (i) Napisać dokumentację parametrów w głównej klasie modułu wzorując się na modułach z forge'a

   (j) Do modułu napisać profil z przykładowym wywołaniem, lista hostów wraz z wartościami parametrów ma być podawana z hiery

   (k) Zachować czystość kodu, nie duplikować go

17.4. Ćwiczenia do samodzielnego wykonania

1. Zapoznać się i zainstalować moduły:
   - puppetlabs/firewall
   - puppetlabs/mysql
   - puppetlabs/postgresql
   - puppetlabs/motd
   - ghoneycutt/ssh
   - example42/network
2. Ustawić sobie mota. Ma zawierać polecenia puppet-apply, puppet-apply-noop oraz puppet-check wraz z bardzo skrótowym opisem co robi każde z poleceń
3. Poczytać sobie o iptables w linuksie (zwłaszcza o przepływie pakietów)
4. Wyłączyć puppetem selinuxa w systemie operacyjnym

18. Lista 18

18.1. Plan działania

18.1.1. Kod puppetowy pod wiele systemów

Dotąd pracowaliśmy jednynie na vagrancie z CentOS 7. Od teraz będziemy mieli do dyspozycji dodatkowy obraz - z Debianem 8. Od tej pory cały kod puppetowy musi działać poprawnie na obydwu systemach.

18.1.2. Jak to zrobić?

Pomocna będzie tutaj klasa params.pp (omawiane na jednej z poprzednich list). Tam należy ustawić logikę na parametrach, a moduł po prostu z tych parametrów korzysta.

18.1.3. A skąd puppet ma wiedzieć jaki to system operacyjny?

Z factera - tam znajdziesz wszystko - nazwę systemu, jego główną wersję, numer aktualizacji i takie tam. Cały problem jest w tym, żeby znaleźć różnice między systemami i tak napisać logikę w params.pp, żeby wszystko było przejrzyste i działało. Pewne rzeczy jak na przykład instalacja pakietów z użyciem yum/rpm w centosie oraz apt/dpkg w debianie, są ustawione domyślnie (można je oczywiście nadpisać).

18.1.4. Wielomaszynowy vagrant

Vagrant pozwala uruchomić więcej niż jedną maszynę jednym Vagrantfilem. Na branchu master jest poprawiony Vagrantfile. Zmerdżujcie sobie go do swoich branchy.
Obsługa jest taka sama, z jedną różnicą - trzeba wskazać maszynę, `vagrant status` pokaże listę maszyn, użycie:
`vagrant [up|ssh|destroy|suspend] <maszyna>`

18.2. Ćwiczenia na zajęcia

Ćwiczenia należy wykonać używając swojego brancha `kod_<imie>_<nazwisko>`

1. Popraw swój kod tak, aby działał poprawnie na Debianie (może się zdarzyć, że nie będą potrzebne żadne poprawki). Pamiętaj, że kod musi być maksymalnie prosty i nie zawierać zbędnych duplikacji.

2. Napisz profil ustawiający bazowy firewall. Polityki INPUT DROP, FORWARD DROP, OUTPUT ACCEPT. W łańcuchu FORWARD wszystkie pakiety należące do jakiejkolwiek sieci wewnętrznej mają być ostatecznie odrzucane. W łańcuchu INPUT: pozwól na pingi z sieci wewnętrznych, odrzuć pakiety INVALID, analogicznie do łańcucha FORWARD odrzuć na koniec pakiety z sieci wewnętrznych. Do tego standardowe reguły - pozwól na pakiety na pętli zwrotnej oraz te w połączeniach RELATED i ESTABLISHED. Do tego utwórz łańcuch services, niech każdy nowy pakiet przechodzi przez ten łańcuch. Niech puppet usuwa wszystkie nieutrzymywane łańcuchy oraz nieutrzymywane reguły.

3. skonfiguruj SSH, na firewallu wpuść ruch na 22/TCP bez ograniczeń, ale tylko gdy maszyna jest uruchamiana na Vagrancie, reguła niech będzie wpisywana do łańcucha services.

4. Skonfiguruj ngnxa, wystaw na porcie 80 katalog `/var/www/html`. Otwórz port 80/TCP bez ograniczeń (oczywiście wpisując to do łańcucha services)

5. Zainstaluj postgresql używając modułu z forge'a. Otwórz port 5432 do sieci `192.168.251.0/27`

18.3. Ćwiczenia do samodzielnego wykonania

Ni ma

19. Lista 19

19.1. Plan działania

19.1.1. Filebucket

Czy zastanawiałeś się kiedyś, co się dzieje z plikami usuwanymi przez Puppeta? Nie jest to klasyczny `rm -f` czy przenoszenie do `/dev/null`. Puppet ma wbudowany mechanizm filebucket. Domyślnie jest to katalog - na każdym hoście, na którym jest uruchamiany agent. W momencie gdy puppet usuwa jakiś plik, wypisuje na ekran informację że filebucketed pod jakimś hashem. Używając tego hasha możesz znaleźć plik w koszu puppetowym.
Co ciekawe filebucket nie musi być rozproszony po hostach - można stworzyć centralne miejsce na masterze, do którego puppet będzie wysyłać zbędne pliki.
`https://docs.puppet.com/puppet/latest/types/filebucket.html`

19.2. Ćwiczenia na zajęcia

Ćwiczenia należy wykonać używając swojego brancha `kod_<imie>_<nazwisko>`

1. Wejdź na branch `lista19_cw1` i odpal puppeta na maszynie z centosem
2. Przeczytaj wyjście z puppeta i na tej podstawie określ co zrobił
3. Zerknij do kodu i sprawdź co faktycznie zrobił. Co powiesz o czytelności kodu?
4. Cofnij zmiany wprowadzone przez puppeta. Czy wszystkie da się cofnąć?
5. Wnioski?

19.3. Ćwiczenia do samodzielnego wykonania

Afirmacja na dziś: piszę czytelny kod bez zaciemnień.

20. Lista 20

20.1. Plan działania

20.1.1. Pitu pitu o puppecie

Macie za sobą pierwsze użycia cudzych modułów. Podstawowy problem z jakim się zderzyliście, jest dysonans między tym, że macie swoje wyobrażenie o konfiguracji, jak powinna wyglądać, a tym, że moduł który chcecie użyć, pasuje swoim działaniem jak pięć do nosa. Bo albo nie ma dokumentacji, albo ta dokumentacja jest tak napisana, że chyba tylko autor wie o co w niej chodzi, albo zawiera błędy. Człowiek przekleja 1:1 przykład, i nie działa. I zaczyna się frustracja, i narzekanie, że puppet jest taki a taki.
Największą pracą jaką trzeba wykonać przy korzystaniu z puppeta, to odzwyczaić się od dotychczasowego myślenia opartego na własnych skryptach i swoim widzi-mi-się. Bo tak najczęściej wygląda zarządzanie serwerami w początkowych etapach nauki. Skrypty dają złudne poczucie wolności, bo przecież możecie je sobie napisać po swojemu, wstawić tyle spacji ile chcecie itd.
Otrzeźwienie przychodzi w momencie, gdy tych skryptów jest tyle, że się w nich gubicie. Każdy napisany w inny sposób. Konfiguracja skryptów i same skrypty rozproszone po całym systemie plików. Wtedy odpowiedzenie na pytanie: co ten serwer zawiera, sprowadza się do wielogodzinnych przeszukiwań maszyny.
Sprawa jeszcze bardziej się komplikuje, gdy osób zarządzających infrastrukturą jest kilka. Wtedy chaos jest nieunikniony.
Puppet wprowadza pewną standaryzację, a w zasadzie wprowadza możliwość standaryzacji. Bo da się tak napisać kod puppetowy, że niewiele będzie się różnił od chaosu miliona skrypcików. W efekcie znajdziemy się w punkcie wyjścia. Bo to, że coś jest w puppecie, nie znaczy, że jest z gruntu właściwe i na pewno dzięki temu będzie można odzyskać maszynę bez większych problemów.

20.1.2. Jak pisać moduły

- Przede wszystkim prosto i przejrzyście, kod musi być czytelny przede wszystkim dla innych
- Pisać dobrą dokumentację do modułu lub klas - kilka zdań, ale sensownych
- Używać standardowych metod puppetowych. Jeśli jakaś funkcjonalność jest dostępna w puppecie, choćby w najbardziej upierdliwy sposób, to należy użyć standardowej metody
- Typ `exec` jest bardzo wygodny, ale na tym się kończą jego zalety. Nie ma wsparcia filebucketa, diffów, nie mówiąc o czytelności takiego kodu. Dlatego exec może być stosowany wyłącznie w ostateczności, w możliwie najprostszy sposób
- Dodawać funkcje przyrostowo, gruntownie każdą testując - napisać trochę kodu, przetestować, dopisać odrobinę, przetestować. Najwięcej błędów popełnia się wówczas, gdy ja-wiem-jak-się-pisze-puppeta więc napiszę kilkadziesiąt linii kodu, a potem siedzę godzinę i szukam błędu. Albo co gorsza nie zauważę tego błędu
- Nie ufać swoim oczom i pamięci - zawsze sprawdzać dokumentację podstawowych typów, dłuższe zmienne kopiować, a nie przepisywać, zaś fragmenty kodu porównywać vimdiffem lub podobnymi narzędziami, a nie na oko.
- Przeprowadzać weryfikację typów zmiennych, możliwie najdokładniej.
- Po napisaniu modułu skrócić kod.
- Kod puppetowy musi się poprawnie aplikować na czystej maszynie.

20.1.3. Jak czytać moduły

Chociaż lista zasadniczo dotyczy cudzych modułów, to ma też zastosowanie do własnych, zwłaszcza tych, których po jakimś czasie już nie pamiętamy jak użyć.
Po pierwsze zawsze należy przeczytać całą dokumentację (bardziej skupić się na ostrzeżeniach

autora, obostrzeniach i zalecanych metodach niż opisach parametrów). Korzystając z modułu - patrzeć do dokumentacji, ale i do klasy/typu jaki chce się użyć, i na bieżąco weryfikować obecność i składnię parametrów.
Czasem pojawia się pytanie czy jakaś funkcjonalność jest wykonalna w danym module. To, że nie jest ona w dokumentacji opisana, nie znaczy, że nie istnieje. Przeciwnie - czasem w modułach są „ukryte funkcjonalności", tzn takie, z których istnienia sam autor nie zdaje sobie sprawy (albo zdaje, tylko niezbyt zgrabnie to opisał). Można je znaleźć wyłącznie analizując kod. Bywa też tak, że dokumentacja jest tak kiepska, albo tak rozwlekła, że szybciej popatrzeć do kodu modułu. Dlatego uczyliśmy się tak szczegółowo składni.
Zdarza się, że czasem nie rozumiemy kodu. Wtedy dobrze pododawać sobie trochę notice-ów, zrzucić zmienne do pliku, i popatrzeć co się dzieje. To nie fizyka kwantowa, jakoś to przecież musi działać :)

20.2. Ćwiczenia na zajęcia

Analiza kodu kilku popularnych modułów.

20.3. Ćwiczenia do samodzielnego wykonania

Ni ma

21. Lista 21

21.1. Plan działania

21.1.1. Narzędzia do puppeta

Są graficzne narzędzia współpracujące z puppetem. Proste narzędzie - Puppetboard pozwala na przeglądanie stanu puppeta. Foreman - rozbudowane narzędzie służące do zarządzania (tworzenie, usuwanie itp.) maszyn.

21.1.2. Sztuczki puppetowe

- „Dziedziczenie"parametrów w typach zdefiniowanych - wiem wiem, mówiłem, że typy nie mogą dziedziczyć parametrów, i nadal to prawda - nie mogą. Ale da się te parametry tak zaonaczyć, że będze to wyglądało jak dziedziczenie. Niezbyt elegancka metoda, ale działa, i rzadko używana upraszcza mocno kod. Przyjrzyj się funkcji `getvar` z stdliba: `https://forge.puppet.com/puppetlabs/stdlib#getvar`

- Szybkie Ruby w manifestach - czasem potrzeba użyć jakiejś gotowej funkcji rubiowej, a nie ma jej w puppecie. Pisać własne opakowanie do tej funkcji tak, żeby puppet z tego wprost skorzystał to za dużo roboty. Tym razem z pomocą przychodzi standardowa funkcja `inline_template` `https://docs.puppet.com/puppet/latest/function.html#inlinetemplate`. Jej wadą (a właściwie to jest jej funkcja) jest to, że produkuje zawsze stringa. Ale co za problem potem dokonać prostej konwersji... Rozwiązanie również mało eleganckie, ale czasem wydaje się lepszym wyjściem, niż pisanie zewnętrznej funkcji puppetowej, zwłaszcza gdy będzie to kilkanaście znaków. Argumentem funkcji jest typowa składnia z erba.

- Sprawdzanie czy istnieje plik w kodzie puppetowym - czasem pisząc moduł (na przykład do wrzucania certów) trzeba przypilnować użytkownika, żeby na pewno wstawił pliki w dobrym miejscu w kodzie puppetowym, a jeśli tego nie zrobi, to napisać mu jakiś ładniejszy błąd niż „Could not intern index from PSON" lub innego niedziałabozepsute.
  Tutaj przykład użycia: ¡redacted¿.
  Metoda oczywiście sprawdza, czy plik istnieje na masterze puppetowym.

- Uruchamianie jednej rzeczy - czasem masz ochotę testować na vagrancje tylko jedną klasę (a nie 700 pozostałych załączonych w common.yaml, które mielą się godzinami, i zabierają czas procesora, który mógłbyś wykorzystać do kopania bitkojnów). Robisz po prostu
  `type puppet-apply`
  a następnie zamiast
  `/vagrant/manifests/site.pp`
  wstaw:
  - `-e 'include sru'`
  - `-e 'class { 'sru':}'`
  - `-e '#dowolny fragment kodu manifestu (zamiast nowej linii - spacja)'`

- Noop nie gryzie - używaj tej opcji i czytaj wyjście, wprowadzaj poprawki, dopiero jak jesteś zadowolony z efektu puszczaj bez noopa - to pozwala na uniknięcie niektórych błędów

- Rób jedną rzecz naraz. Najpierw akwarium, potem rybki, a na końcu połącz je wodą :)

- Dostarczaj działający, coś robiący kod, tak szybko jak się da i leć z nim aż do produkcji

- Wdrażaj zmiany małymi porcjami

- Staraj się rozwiązywać problemy samemu (wtedy więcej się uczysz), ale nie bój się poprosić o pomoc w przypadku błędu - dodatkowa para oczu wyłapie literówkę dużo szybciej

- Jeśli robota nie idzie, zajmij się czym innym, oddal się od komputera (np. w kierunku domu). Po powrocie łatwiej pójdzie.

21.1.3. Jak automatyzować puppetem

- `https://puppet.com/resources/infographic/5-tips-for-getting-started-puppet-enterprise`
- Jedno po drugim, zacznij od podstawowych rzeczy
- Automatyzuj pierwowzór na maszynę na boku, przetestuj czy działa, a potem rób przepięcie (w DNSach lub na sieci)

21.1.4. Testowanie kodu puppetowego

Tak jak aplikacje mają swoje testy (end-to-end, unit itd.) tak kod puppetowy można testować, i to na różnych poziomach:

- rspec - testuje moduł, klasa po klasie, patrząc od strony kodu (czy coś się wywołało itd.), testy oczywiście pisze się ręcznie. `http://rspec-puppet.com/`
- serverspec - testuje kod jako całość patrząc od strony serwera `http://serverspec.org/`
- lint - puppet lint, hiera yaml - testuje składnię i stylistykę (to już znamy z poprzednich list)
- kompilacja kodu per host - czy puppet przechodzi na środowisku przed wpuszczeniem merdża (przeprowadzenie kompilacji kodu host po hoście, na jakiejś farmie dockerów, biorąc aktualną listę hostów ze środowiska)
- ręczne testowanie na Vagrancie (nie wszystko wychodzi)
- ręczne przeglądanie MRów przed zaakceptowaniem
- okresowe usuwanie maszyn i stawianie puppetem od nowa

21.2. Ćwiczenia na zajęcia

Omówienie kodu puppetowego używanego w dziale operacyjnym.

21.3. Ćwiczenia do samodzielnego wykonania

Nauczyć się omówionego kodu na pamięć. Przynajmniej pierwszych kilku znaków