Szkoła Główna Gospodarstwa Wiejskiego w Warszawie
Wydział Zastosowań Informatyki i Matematyki

inż. Piotr Perzyna – 161195

Analiza porównawcza narzędzi dla DevOps

Comparative analysis of tools for DevOps

Praca magisterska
na kierunku – Informatyka

Praca wykonana pod kierunkiem
dr inż. Jarosław Kurek
Katedra Informatyki

Warszawa, 2017

Oświadczenie promotora pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data ..... Podpis promotora pracy .....

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przez zemnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data ..... Podpis autora pracy .....

STRESZCZENIE

Praca została zainspirowana narzędziami automatyzacji konfiguracji wirtualnych maszyn. Głównym jej celem była analiza porównawcza narzędzi pod kątem wymagań stawianych przez zemnie między innymi takich jak: otwartość kodu źródłowego, skalowalność. Na początku mojej pracy dyplomowej przybliżyłem narzędzia – Ansible, Chef, Puppet i SaltStack. Drugą kwestią, była analiza wybranych narzędzi oraz praktyczna przykładowa implementacja do automatyzacji instalacji usług. Na końcu przedstawiłem korzyści jakie mogą płynąć z zastosowań automatyzacji.

Słowa kluczowe – automatyzacja, konfiguracja, środowisko wirtualne, devops

SUMMARY

The work was inspired by tools automate the configuration of virtual environments. Its main purpose was to benchmark the tools in terms of the requirements that are inherent in me, such as open source code, scalability. At the beginning of my thesis I drew tools - Ansible, Chef, Puppet and SaltStack. The second issue was the analysis of selected tools and the practical implementation of Ansible for automation of service installations. At the end I presented the benefits that may flow from use of automation.

Keywords – automation, configuration, virtual environment, devops

SPIS TREŚCI

Wstęp 9

1 Wprowadzenie 11

1.1 DevOps 11

1.2 Zarządzanie konfiguracją 12

1.3 Wymagania narzędzia 13

2 Narzędzia 14

2.1 Ansible 14

2.1.1 Opis 14

2.1.2 Elementy systemu 15

2.1.3 Zasada działania 20

2.1.4 Wymagania 21

2.1.5 Instalacja 21

2.2 SaltStack 22

2.2.1 Opis 22

2.2.2 Zasada działania 22

2.2.3 Wymagania 24

2.2.4 Instalacja 24

2.3 Puppet 26

2.3.1 Opis 26

2.3.2 Elementy systemu 27

2.3.3 Zasada działania 30

2.3.4 Wymagania 31

2.3.5 Instalacja 32

2.4 Chef 33

2.4.1 Opis 33

2.4.2 Wymagania 34

2.4.3 Instalacja 34

3 Analiza 35

3.1 Własna 35

3.2 Analiza popularności 35

4 Implementacja 37

4.1 Icinga 37

4.2 Środowisko 37

4.3 Playbook 38

4.3.1 Master 38
4.3.2 Klient 38
4.4 Testowanie 38
4.5 Wyniki 40
5 Podsumowanie 41
6 Bibliografia 42
7 Spis rysunków 43
8 Spis tabel 43

Wstęp

W dzisiejszym świecie możemy zaobserwować gwałtowny wzrost popularności usług sieciowych. Przykładową taką usługą są aplikacje webowe, gdzie dynamiczny rozwój zmusił firmy zajmujące się tworzeniem oprogramowania do częstych wydań (*ang. release*) nowych wersji aplikacji. Przełożyło się to na znaczący wzrost obowiązków administratorów systemów komputerowych, którzy oprócz prac wykonywanych do tej pory takich jak [1]: opieka nad serwerową i sieciową infrastrukturą, wdrażanie nowych rozwiązań informatycznych, diagnoza usterek i awarii systemów komputerowych oraz prac administracyjnych muszą sprostać nowym trudnością, które stawia przed nimi proces wydawania nowej wersji oprogramowania *(ang. workflow)*.

W obecnych czasach odnotowujemy również duże zdekoncentrowanie lokalizacji serwerów przez wykorzystanie rozproszonej chmury obliczeniowej (*ang. cloud computing*), które jest spowodowane między innymi [2]: możliwością dużej elastyczności zasobów w ramach zapotrzebowania i chęcią zminimalizowania długości drogi do użytkowników. Druga wymieniona chmury znacznie usprawnia szybkość i stabilność komunikacji. W wielu organizacjach zauważono, że pomimo geograficznych różnic, serwery zlokalizowane w zdalnych centrach danych posiadają identyczne konfiguracje z zainstalowanymi takimi samymi aplikacjami i usługami. Ręczne przygotowanie każdej instancji lub wdrożenie nowego wydania oprogramowania przez jednego człowieka było by bardzo czasochłonne, a czasami niemożliwe. Dodatkowo administracja lub zmiany serwera w ramach usług aplikacji, musiały by być wykonywane wykonane ręcznie w jednym czasie, co w dużej organizacji wymagałoby grupy przeszkolonego personelu. Ponadto możliwość pomyłki podczas ręcznego tworzenia lub wprowadzania zmian w konfiguracji byłaby dość wysoka.

Przez wzgląd na powyższe trudności powstały zespoły wspomagające prace programistów, które zaczęto nazywać DevOps, których podstawowym zadaniem jest dostarczanie wydań oprogramowania szybciej i częściej [3]. Jednak rodzi to kolejny problem z jakim musza stawić czoła administratorzy. W cyklu wydania, oprogramowanie zwykle przechodzi przez trzy etapy: rozwój, testy i produkcja [4]. Dostarczenie nowej wersji oprogramowania z odpowiednimi parametrami na każde z środowisk przypisancyh do etapów jest procesem inżynierii systemów zwanym Zarządzanie Konfiguracją (ang. configuration management).

Dziś odchodzi się od ręcznych konfiguracji, na rzecz zautomatyzowanych procesów. Na rynku powstało wiele narzędzi wspomagających pracę w metodologii DevOps. W swoich badaniach szukałem narzędzi wydawanych na zasadach wolnego oprogramowania, które umożliwiają zautomatyzowane przygotowanie środowisk oraz dostarczanie zmian w sposób ciągły.

Na początku pracy przedstawiłem zagadnienie DevOps i proces Zarządzania Konfiguracją. W kolejnym rozdziale skupiłem się na szczegółowym przedstawieniu najbardziej popularnych narzędzi. Zawarłem informacje takie jak: przyjęta koncepcje, wygląd technologii i architektury oraz opisałem ich zasadę działania. Trzeci rozdział poświęciłem analizie narzędzi pod kątem postawionych przeze mnie wymagać jakie powinno spełniać dobre rozwiązanie. Dokonałem również badania popularności każdego rozwiązania, aby zobrazować wielkość społeczności angażującej się w rozwój narzędzia. W przed ostatnim rozdziale zająłem się praktyczną implementacja aktualnie najpopularniejszego rozwiązania. Zautomatyzowałem proces ciągłego dostarczania nowej wersji aplikacji. W ostatnim paragrafie zawarłem podsumowanie i sugestie dotyczące pracy.

1 Wprowadzenie

1.1 DevOps

W celu zapewnienia jakości dostarczanych wydań oprogramowania w 2009 r powstał termin DevOps sformułowany przez Patricka Dubois [5], który nawiązuje do zespolenia zespołu rozwoju (*ang. development*) i operacji (*ang. operations*). Koncepcja ta wywodzi się od osób z obszaru zarządzania systemami przedsiębiorstwa (*ang. Enterprise Systems Management*) i programowania zwinnego (*ang. Agile software development*) []. Jest najbardziej odpowiednia dla firm, w których częstotliwość wydań kolejnych wersji oprogramowania jest duża. Dla przykładu serwis Flickr rozwinął zdolność przeprowadzania dziesięciu wdrożeń dziennie []. Cykl pracy zespołu DevOps został przedstawiony na rysunku 1, gdzie ciemniejszym kolorem zostały przedstawione obowiązki rozwoju, a jaśniejszym operacji.

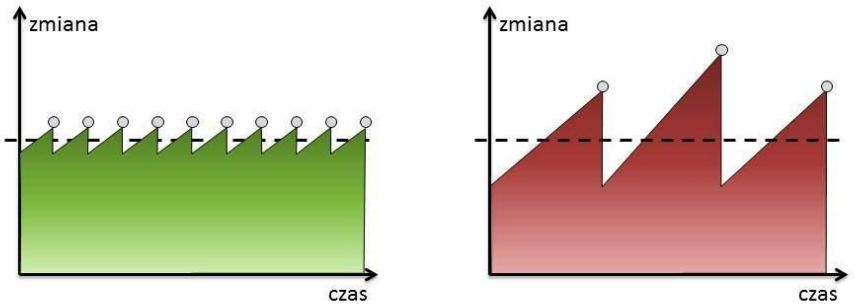
Rysunek SEQ Rysunek * ARABIC 1 – Zadania DevOps

Źródło: https://www.suse.com/

Metodologia DevOps pomaga w zarządzaniu wydaniami (ang. release management), przez standaryzowanie środowisk. To znaczy że wydarzenia zachodzące w cyklu wydania mogą być w łatwy sposób śledzone i udokumentowane. W innym wypadku procesy związane z wydaniami to działania o do dużym stopniu stresu i mogące angażować wiele osób. Dzięki zespołom DevOps wydania nie powodują zwiększonego ryzyka z następujących powodów:

1. redukcji poziomu zmian,
2. zwiększenia ilości wydań,
3. automatyzacji.

Automatyzacja eliminuje czynnik błędu ludzkiego oraz zapewnia powtarzalność działań. Natomiast redukcja poziomu wydań to zmniejszenie ilości zmian które są dostarczane, ale za to zgodnie z drugim punktem zwiększa się ilość wydań. To tworzy wyrównaną liczbę progresywnych zmian w przeciwieństwie do rzadkich wdrożeń o dużym zmianach [rysunek 2].



Rysunek 2 - Różnice wielkości zmian w f. częstotliwości wydań
Źródło: https://www.governica.com/DevOps

Koordynowanie wydań to stosunkowo nowa rola w firmach IT. Można ją porównać do kontrolera lotów, który dokonuje koordynacji działań w realnym czasie, przez różne zasoby, aby osiągnąć wspólny cel grupy (bezpieczne lądowanie i startowanie), używając wspólnych zasobów (lotniska).

Na pomoc zespołom DevOps przychodzą narzędzie pozwalające zautomatyzować cykl wydań. Dostarczają one możliwość zarządzania konfiguracją oraz różne koncepcje jak powinno odbywać się wdrożenie wydania, planowanie infrastruktury oraz przygotowanie środowisk.

1.2 Zarządzanie konfiguracją

Zarządzanie konfiguracją (CM ang. Configuration Management) jest procesem inżynierii systemów, polegającym na administracji: zasobami oraz procesami zachodzącymi w ciągu istnienia danego projektu [11]. Głównym celem zarządzania konfiguracją jest kontrolowane wprowadzanie do projektu zmian dotyczących dokumentacji, kodu programu i innych produktów faz projektowych. Jest to spowodowane chęcią uniknięcia negatywnego wpływu zmian na projekt. W problematyce niniejszej pracy zarządzanie konfiguracją dotyczy się administracji środowiskami, przez badane narzędzie w procesie wydawania nowej wersji oprogramowania.

2 Narzędzia

2.1 Ansible

Rysunek SEQ Rysunek * ARABIC 3 - Logo Ansible

Źródło: https://www.ansible.com/

2.1.1 Opis

Ansible [1] to narzędzie udostępniana na licencji wolnego oprogramowania, które zostało zapoczątkowane w 2012 roku przez Michaela DeHaan, służące do zarządzania w sposób powtarzalny zdalnymi maszynami, takimi jak serwery oraz routery. Oprogramowanie w 2015 roku zostało włączone w dystrybucje systemy linux - Fedory, którym właścicielem jest Red Hat.

Ansible zostało zaprojektowane tak, aby maksymalnie uprościć i ułatwić proces konfiguracji. Narzędzie musi być zainstalowana tylko na maszynie z której używa się do zarządzania infrastrukturą. Nie wymaga instalowania dodatkowego oprogramowana tzw. agentów. Rozwiązanie do komunikacji z zarządzanymi maszynami wykorzystuje bezpieczne połącznie SSH, które zostało szczegółowo przedstawione w rozdziale 2.1.3. Domyślnie Ansible działa w trybie push, to znaczy ze wszystkie zmiany w konfiguracji są inicjalizowane z maszyny zarządcy.

Narzędzie opiera się na języku Python, który jest językiem programowania wysokiego poziomu ogólnego przeznaczenia, a jego interpretery są dostępne na wiele systemów operacyjnych. Dzięki temu Ansible może kontrolować wszystkie główne systemy operacyjne dostępne na rynku takie jak Windows, Unix i GNU/Linux.

2.1.2 Elementy systemu

2.1.2.1 Inventory

Plik z informacją na temat hostów (ang. inventory) jest jednym z ważniejszych plików w projekcie Ansible, w którym w formacie INI-podobnym są przechowywane informacje na temat zarządzanych hostów. Dokument ten przetrzymuje takie dane jak: grupy, nazwa maszyny, IP adres, port połączenia oraz nazwa użytkownika i hasło. Przykładowy plik wygląda następująco:

```
1  mail.example.com
2
3  [webservers]
4  foo.example.om
5  bar.example.com
6  [dbservers]
7  one.example.com
8  two.example.com
9  tree.exameple.com
```

W nawiasach kwadratowy są ujęte nazwy grup, a wszystko co znajduję się pod spodem, aż do następnej grupy to informacje o zarządzanych hosta, które mogą być w wielu grupach jednocześnie. Dzięki temu w łatwy sposób możemy skalować rozwiązanie na duże ilości maszyn. Zasada używania grup jest dość prosta, to znaczy podaję się zamiast nazwy hosta nazwę grupy podczas wykonywania poleceń Ad-Hocs lub PlayBooków.

2.1.2.2 Ad-Hocs i PlayBook

Ansible wyróżnia dwie metody przetwarzania poleceń. Pierwsza metoda to Ad-Hocs, wykonuje pojedyncze polecenia na zarządzanych hostach i zwraca wynik. Przykładowe zapytanie oraz otrzymane odpowiedzi zostały zawarte poniżej.

```
1  ansible webservers -m ping
1  foo.example.com | success >> {
2      „changed”: false,
3      „ping”: „pong”
4  }
```

```
5  bar.example.com | success >> {
6      „changed”: false,
7      „ping”: „pong”
8  }
```

Narzędzie uruchomiło moduł ping i wysłało odpowiednie polecenie na wszystkie hosty zdefiniowane w grupie webservers. Moduł ten może być używany do sprawdzania dostępności kontrolowanych maszyn. Ansible na chwilę obecną udostępnia około 1035 wbudowanych modułów.

Moduły wysłane do instancji zarządzanych można łączyć w PlayBooki, jest to drugą metoda przetwarzania. PlayBooki są to powtarzalne schematy wdrożeń zdefiniowane w języku YAML, którego składnia bazuje na języku angielskim. YAML przeznaczony został do reprezentowania danych w sposób ustrukturyzowany. Dzięki czemu PlayBooki są łatwo czytelne dla człowieka oraz maszyny. Przykładowy PlayBook został przedstawiony poniżej.

```
1  ---
2  - hosts: webservers
3    become: yes
4    become_user: root
5
6    tasks:
7
8     - name: Instalacja Apache2
9       apt: name=apache2 state=latest update_cache=yes
10      notify:
11       - restart_apache
12
13   handlers:
14
15    - name: Restart Apache2
16      service: name=apache2 state=restarted
17      listen: „restart_apache”
```

Przedstawiony PlayBook zostanie wykonany na wszystkich maszynach w grupie webrserwers. Schemat wykona się z konta użytkownika root i uruchomi moduł apt, który reprezentuje menadżera pakietów apt-get w systemach Linux z rodziny Debiana. Spowoduje to automatyczną instalacje (zgodnie z parametrem state=latest) pakietu apache2, który jest serwerem WWW. Parametr update_cache=yes, pozwoli menadżerowi apt-get na aktualizacje swojego repozytorium. Sekcja „notify", określa dodatkowe elementy jakie mają zostać przetworzone na końcu PlayBooka, tylko jeśli wykonywanie aktualnego zadania zmodyfikuje stan systemu i zakończy się sukcesem. W naszym przykładzie jeśli pakiet apache2 nie był wcześniej zainstalowany na maszynie i zostanie poprawnie zainstalowany to zainicjalizuję się wyzwalacz „restart_apache", który jest zdefiniowany w sekcji handlers i powoduje restart serwera WWW.

Zgodnie z najlepszymi praktykami zamieszonymi na stronie dokumentacji rozwiązania, dla dużych infrastruktur, których konfiguracja składa się w wielu zadań - PlayBook powinien posiadać odpowiednia organizacje folderów, przedstawioną poniżej.

```
1  production – Plik Inventory
2  testing – Plik Inventory
3
4  group_vars/
5      group1 – Plik zmiennych dla grupy „group1"
6      group2 – Plik zmiennych dla grupy „group2"
7
8  host_vars/
9      hostname1 – Plik zmiennych dla maszyny „hostname1"
10     Hostname2 – Plik zmiennych dla maszyny „hostname2"
11
12 library/ – Folder zawiera dodatkowe, własne moduły
13
14 filter_plugins/ – Folder zawiera dodatkowe, własne filtry
15
16 site.yml – Główny plik playbooka
17 webservers.yml – Pomocniczy plik playbooka
18
```

```
19 roles/
20
21     common/ - Rola „Common”
22
23         tasks/ - Folder zawierający zadania
24             main.yml
25
26         handlers/ - Folder zawierający wyzwalacze
27             main.yml
28
29         templates/ – Folder zawierający szablony
30             ntp.conf.j2
31
32         files/ - Folder dla plików statycznych
33
34         vars/
35             main.yml – Zmienne zależne od roli
36
37         defaults/
38             main.yml – Zmienne najniższe w hierarchii ważności
39
40         meta/
41             main.yml – Plik z informacją o zależnościach
42
43         library/
44
45         lookup_plugins/
46
47     monitoring/ - Rola „Monitoring”
```

Jedną z dużych zalet rozwiązania jest możliwość grupowania zadań w role, które mogą być używane w wielu projektach, często nie powiązanych ze sobą. Ansible zyskuje na tym dużą elastyczność. Raz zdefiniowane PlayBooki mogą zostać wykorzystywane ponownie lub udostępnione innym osobą. Autorzy narzędzia udostępnili miejsce w

Internecie pod nazwą Ansible Galaxy, w którym każdy może podzielić się swoim rozwiązaniem. Stworzyło to ogromna bazę gotowych PlayBooków, który w łatwy i prosty sposób można zaimplementować w innych projektach.

2.1.2.3 Parametryzacja i szablony

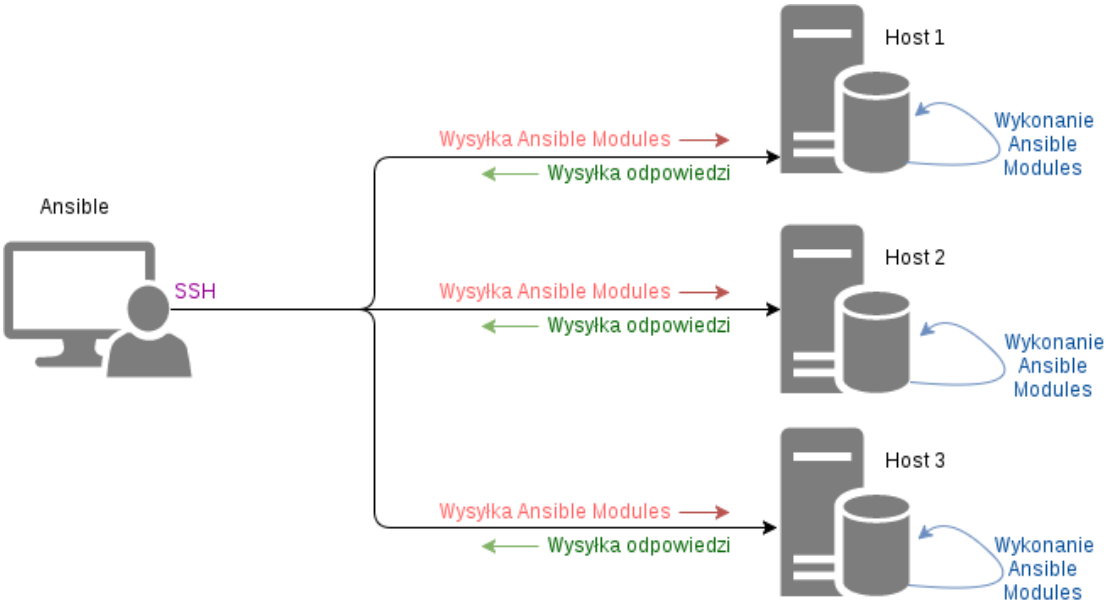
Ważną cechą wpływającą również na elastyczność rozwiązania Ansible jest możliwość parametryzacji kodu oraz tworzenia dynamicznie plików konfiguracji dzięki modułowi template. Przykładowy szablon został przedstawiony poniżej i jest napisane w specjalnie wyznaczonym przez Ansible formacie Jinja2.

```
<VirtualHost *:80>
    ServerName {{ item.domain }}
    ServerAdmin webmaster@{{ item.domain }}
    DocumentRoot {{ item.path }}
    <Directory {{ item.path }}>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Pokazany szablon służy do skonfigurowania wirtualnego hosta w serwerze HTTP, realizowanego przez Apache2. Szablon można wywołać w pętli, definiując różne domeny. Przykładowe zadanie realizujące wdrożenia konfiguracji na maszynę wygląda w następujący sposób.

```
- name: Test Szablonu
  template:
    src=virtualhost-apache.conf.j2
    dest=/etc/apache2/sites-enabled/{{ item.domain }}
.conf
    with_items:
      - { domain: 'example1.com', path: '/opt/www1/' }
      - { domain: 'example2.com', path: '/opt/www2/' }
      - { domain: 'example3.com', path: '/opt/www3/' }
```

2.1.3 Zasada działania



Rysunek 4 - Schemat działania Ansible

Źródło: opracowanie własne

Ansible instaluje się na komputerze zarządcy daną infrastrukturą, a na zarządzanych hostach nie wymaga się dodatkowego oprogramowania. Unika w ten sposób problemu „zarządzania zarządzaniem" z jakim mamy do czynienia w pozostałych narzędziach. Do komunikacji wykorzystuje bezpieczne połączenie SSH (ang. secure shell) w przypadku Linux/Unix i natywnego połączenie z PowerShell, a uwierzytelnienie przebiega zgodnie z wybraną wcześniej metodą: za pomocą nazwy użytkownika i hasła, klucza lub przy wykorzystaniu Kerberosa. Ogólny schemat działania przestawiony został na rysunku nr 4 i można wyróżnić następujące etapy:

1. Ansible łączy się po SSH do hosta.
2. Narzędzie wysyła program zwany „Ansible Modules" do hosta.
3. Na zdalnym hoście wykonywany jest przesłany plik, a rezultat zostaje zwrócony.
4. Ansible usuwa plik z zarządzanej maszyny.

2.1.4 Wymagania

Ansible nie ma wymagań sprzętowych od maszyny na której się je instaluje i która się zarządza. Jedynym wymogiem są zależności oprogramowania. Ansible można

uruchomić jedynie z maszyn z zainstalowanym systemem z rodziny GNU/Linux lub Unix, system Windows nie jest obsługiwany. Do działania narzędzie potrzebuje interpretatora języka Python (w wersji 2.4 lub wyższej). Natomiast kontrolowane maszyny muszą być dodatkowo wyposażone w następujące biblioteki:

- Paramiko,
- PyYAML,
- Jinja2.

Jeśli na zarządzanej maszynie jest włączony SELinux to Ansible potrzebuje również biblioteki libselinux-python.

2.1.5 Instalacja

Instalacja Ansible na maszynie zarządcy w systemie Debian składa się z następujących etapów:

1. Dodania zewnętrznego repozytorium do menadżera pakietów.

```
echo „deb
http://ppa.launchpad.net/ansible/ansible/ubuntu trusty
main” >> /etc/apt/source.list
```

2. Dodania podpisu cyfrowego repozytorium.

```
apt-key adv –keyserver keyserver.ubuntu.com -rev—keys
93c4A3fd7bb9c367
```

3. Instalacji pakietu Ansible przez menadżera pakietów.

```
apt-get update
apt-get install ansible
```

2.2 SaltStack

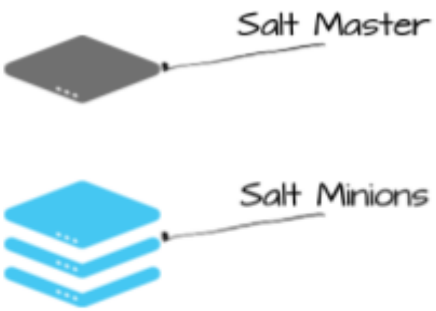
Rysunek 5 - Logo SaltStack

Źródło: https://saltstack.com/

2.2.1 Opis

Autorem projektu SaltStack [2] udostępnionego na licencji Apache 2, który został zapoczątkowany w 2011 roku jest Thomas S. Hatch. Rozwiązanie to podobnie jak Ansible bazuje na języku Python oraz na plikach YAML. Jest narzędziem do zarządzania konfiguracją na zdalnych maszynach, które w rozumieniu SlatStack nazywane są Minion'kami (ang. Minion).

2.2.2 Zasada działania

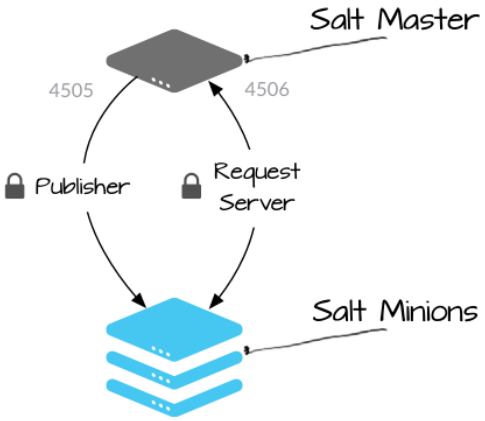


Rysunek SEQ Rysunek * ARABIC 6 - Architektura SaltStack

Źródło: https://docs.saltstack.com/en/getstarted/system/communication.html

W architekturze SaltStack (master-client) pokazanej na rysunku 6 do poprawnego działania wymagane są minimum dwa komponenty. Pierwszym elementem jest Salt Master, który pełni rolę zarządcy. Jest odpowiedzialny za odpowiednie przekazywanie poleceń do drugiego komponentu - agenta, który zlokalizowany jest na maszynie Salt Minion. Komunikacja jest prowadzona na zasadach Publish-Subscribe Pattern pokazanych na rysunku 7. Charakteryzuję się to tym, że łączność zawsze inicjalizowana jest przez Minion'ka do Salt Master na portach 4505 i 4506. Dzięki takiej koncepcji

zwiększa się bezpieczeństwo Minon'ków, ponieważ nie muszą one otwierać portów na przychodzącą komunikacje.

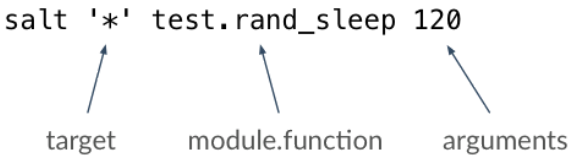


Rysunek 7 - Komunikacja SaltStack

Źródło: https://docs.saltstack.com/en/getstarted/system/communication.html

Dodatkowo zarządca agreguje wyniki wykonywanych poleceń i wyświetla je administratorowi za pomocą webowego interfejsu.

SaltStack został zaprojektowany od początku tak żeby uruchamiać na zdalnych maszynach polecenia, dostarczane przez interfejsy agentów. Wykonywane komendy zostały dostosowane do działania na wielu systemach i wielu platformach. Zwracają zawsze ustrukturyzowane wyniki, które są przechowywane w bazie danych i w łatwy sposób mogą być przeglądane.



Rysunek 8 - Komenda SaltStack

Źródło: https://docs.saltstack.com/en/getstarted/system/execution.html

Przykładową komendę, którą można wykonać na Salt Master pokazano na rysunku nr 5. Wywołanie tej komendy powoduje, że zostanie wysłane polecenie do wszystkich podłączonych Minion'ków do danego zarządcy. Na zdalnych agentach zostanie

wyszukany i uruchomiony moduł ***test*** z funkcją ***rand_sleep*** i argumentami wywołania ***120***, a zarządca otrzyma odpowiedz zwrotną.

Dużą grupę podstawowych modułów i funkcji do kontrolowania wirtualnych środowisk dostarczana jest domyślnie z rozwiązaniem SaltStack. W przypadku braku potrzebnych rozwiązań programiści narzędzia dali możliwość dopisania własnych modułów i funkcji.

2.2.2.1 SaltState

SaltStack oferuje również możliwość tworzenia SaltState. czyli schematów konfiguracji. Wzorce te są zapisywane jako pliki tekstowe przy użyciu języka YAML. Przykład wraz z opisem poszczególnych elementów został przedstawiony poniżej:

```
1  ID:
2    module.function:
3      - name: name
4      - argument: value
5      - argument:
6        - value1
7        - value2
```

Dodatkowo w rozwiązaniu SaltStack wprowadzono SaltPillar, jest to zbiór SaltState z możliwością parametryzacji poszczególnych schematów. Daje to możliwość stworzenia powtarzalnych schematów automatyzacji podobnych do PlayBoków w rozwiązaniu Ansible.

2.2.3 Wymagania

SaltStack nie ma wymagań sprzętowych, ale w zależności od przeznaczenia maszyny ma inne wymagania programowe. Zarówno Salt Master jak i Salt Minion wymagają instalacji odpowiednich paczek, które są dostępne w udostępnianym repozytorium i które zostały przedstawione w rozdziale 2.2.4.

2.2.4 Instalacja

Instalacja SaltStack w systemie Debian składa się z następujących etapów:

1. Dodania zewnętrznego repozytorium do menadżera pakietów.

```
echo „deb
http://repo.saltstack.com/apt/debian/8/amd64/latest
jessie main” >> /etc/apt/sources.list
```

2. Dodania podpisu cyfrowego repozytorium.

```
wget -o –
https://repo.saltstack.com/apt/debian/8/amd64/latest/SA
LTSTACK-GPG-KEY.pub|apt-key add -
```

3. Instalacji pakietu SaltStack w zależności od przeznaczenia maszyny:

a) Salt Master

```
apt-get install salt-master
```

b) Salt Minion

```
apt-get install salt-minion
```

2.3 Chef

Rysunek 9 - Logo Chef

Źródło: https://www.chef.io/

2.3.1 Opis

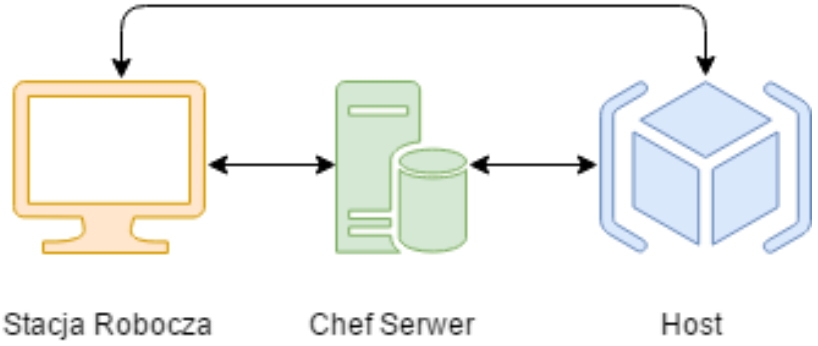
Chef [4] (pol. „szef kuchni") jest kolejnym open-source narzędziem. Jego pierwsze wydanie ukazało się w 2009 roku w firmie Chef Inc., która zaczynała swoja działalność pod nazwą HJK Solutions i zajmowała się automatyzowaniem konfiguracji serwerów dla korporacji. HJK Solution zdała sobie sprawę, że niemożliwa do tej pory w pełni zautomatyzowana infrastruktura staje się osiągalna nawet dla firm, które nie posiadały w wykwalifikowanego zespołu administratorów. Nie był to jedyny czynnik dla którego rozpoczęto prace nad Chef. Głównymi powodami stworzenia oprogramowania były zaobserwowane problemy w istniejących rozwiązaniach, takie jak:

- **brak orientacji na infrastrukturę** – Dotychczasowe rozwiązania nie przetwarzały informacji o infrastrukturze zarządzanych hostów, tylko o zainstalowanych usługach wewnątrz instancji.
- **brak elastyczności kodu** – Dotychczasowe rozwiązania nie umożliwiały współdzielenie kodu pomiędzy administratorów, ani powtarzalności.

Rozwiązanie zostało przygotowane bardziej z myślą o programistach, niż o administratorach systemów. Konfiguracja narzędzia wymaga znajomości języka Ruby, ze względu na to, że pliki konfiguracyjne powinny być stworzone w tym języku.

2.3.2 Elementy systemu

W zastosowanej przez Chef koncepcji komunikacji klient-serwer możemy wyróżnić trzy różne typy instancji:



Rysunek 10 - Schemat komunikacji Chef

Źródło: opracowanie własne

- Stacją roboczą jest komputer, z którego piszą administratorzy piszą CookBook (pol. książki kucharskie). Jest to zazwyczaj maszyna nadzorcy z dowolnym systemem operacyjnym.
- Serwer Chef działa jako centralne repozytorium książek kucharskich, a także przetrzymuję informacje o każdym zarządzanym hoście.
- Host jest dowolną fizyczna lub wirtualną maszyną w sieci obsługiwaną przez serwer Chef. Do działania wymaga zainstalowanego klienta Chef.

2.3.2.1 CookBook

CookBook to ustrukturyzowany zestaw katalogów przechowywujących informacje na temat jakie zadania, mają zostać wykonane na zarządzanych instancjach. Chef udostępnia również narzędzia do tworzenia CookBooków. Przykładowy komplet folderów i plików może zostać przygotowany przez polecenie

```
1  chef generate cookbook motd
```

i będzie wyglądał następująco:

```
cookbook
    .kitchen.yml
    README.md
    attributes/
       default/
    chefignore
    files/
       default/
    metadata.rb
       recipes/
         default.rb
    templates/
       default/
```

.kitchen.yml – Plik w formacie YAML używany do testów CookBooka.

README.md – Plik w formacie markdown zawierający dokumentacje CookBooka.

attributes – Folder zawierający pliki ze zmiennymi atrybutami.

chefignore – Plik przetrzymujący listę plików statycznych, które mają nie być wysyłane do Chef Serwera.

files – Folder zawierający pliki statyczne, które mają być wysłane na zarządzane maszyny.

metadata.rb – Plik z informacjami takimi jak: nazwa, zależności, wersja.

recipes – Najważniejszy folder zawierający przepisy, opisane w rozdziale 2.3.2.2

templates – Folder przechowywujący szablony dla plików konfiguracyjnych, które mogą być parametryzowane w zależności od ustawień zarządzanej maszyny.

2.3.2.2 Recipes

Recipes (pol. przepisy) to najważniejsze pliki konfiguracyjne w rozwiązaniu Chef. Zawierają informacje jakie usługi oraz w jaki sposób mają zostać skonfigurowane. Pliki muszą być napisane w języku Ruby. Przykładowy przepis został przedstawiony poniżej.

```
#
# Cookbook Name:: apache
# Recipe:: default
#

package "httpd" do
  action :install
end

service "httpd" do
  action [ :enable, :start ]
end

template "/var/www/html/index.html" do
  source 'index.html '
  mode '0644'
end
```

Na początku przedstawionego przepisu na zdalnej maszynie zostanie zainstalowany i uruchomiony serwer WWW. Następnie do domyślnej lokalizacji dla której serwer httpd szuka plików stron, zostanie wysłany plik index.html z odpowiednimi uprawnieniami.

2.3.3 Zasada działania

Zasada działania rozwiązania Chef może zostać przedstawiona w trzech krokach.

1. Administrator przygotowuje przepisy, które są grupowane w książki kucharskie.
2. Przygotowane przez administratora pliki są wysyłane do Chef Serwera.

   Chef Serwer przetrzymuje i wersjonuje wszystkie pliki, które są mu dostarczane. Następnie jest odpowiedzialny za komunikacje ze wszystkimi swoimi klientami.
3. Chef Serwer udostępnia narzędzie knife (pol. nóż), które dostarcza odpowiednie przepisy z Cookboków do klientów.

2.3.4 Wymagania

Chef nie ma wymagań sprzętowych, ale w zależności od przeznaczenia maszyny ma inne wymagania programowe. Zarówno Chef Serwer jak i Chef Klient wymagają instalacji odpowiednich paczek, które są dostępne w udostępnianym repozytorium i które zostały przedstawione w rozdziale 2.3.5.

2.3.5 Instalacja

Instalacja Chef w systemie Debian składa się z następujących etapów:

1. Dodania zewnętrznego repozytorium do menadżera pakietów.

```
1 echo „deb https://packages.chef.io/repos/apt/stable
  jessie main” >> /etc/apt/sources.list
```

2. Dodania podpisu cyfrowego repozytorium.

```
1 wget -qO - https://packages.chef.io/chef.asc | apt-key
  add -
```

3. Instalacji pakietu Chef w zależności od przeznaczenia maszyny:

a) Chef Serwer

```
1 apt-get install chef chef-server
```

b) Chef Klient

```
1 apt-get install chef chef-client
```

2.4 Puppet

Rysunek 11 - Logo Puppet

Źródło: https://puppet.com/

2.4.1 Opis

Puppet (pol. „kukiełka”) [3] to najstarsze narzędzie z przedstawianej czwórki. Został stworzony przez firmę Reductive Labs (aktualnie Puppet Labs) w 2005 roku, którą założył Luke Kanies. Stwierdził on, że nie jest zadowolony z ówcześnie dostępnych narzędzi, które były niewielkimi projektami akademickimi, a nie komercyjnymi rozwiązaniami. Głównym celem Puppet, było opracowanie wysokopoziomowego, deklaratywnego języka opisu konfiguracji oraz narzędzia, które przez abstrakcyjna warstwę potrafiłoby aplikować ją do maszyn działających pod różnymi systemami operacyjnymi.

Aktualnie rozwijane jako wolne oprogramowanie na licencji Apache oraz dostępna jest również wersja komercyjna zwana „enterprise editions”. Narzędzie obsługuje wszystkie główne systemy dostępne na rynku: Windows, GNU/Linux oraz inne z rodziny Unix i BSD. Opiera się na języku Ruby DSL (ang. Domain-Specific Language) i jest używane do śledzenia, testowania oraz wdrażania konfiguracji na zarządzanych serwerach.

Podstawowym typem danych w specyfikacji narzędzia Puppet są zasoby (np. użytkownicy, paczki programów, pliki itp.), które przy użyciu deklaratywnego języka podobnego do JSON, można grupować w klasy i zapisywać w plikach zwanych manifestami. Ta deklaratywna natura rozwiązania tworzy ważne różnicę pomiędzy przedstawionymi wcześniej narzędziami, a Puppet. Język deklaratywny zawiera instrukcje dotyczące stanu końcowego konfiguracji. To znaczy, że dla administratorów

Puppet nie jest ważny jak został osiągnięty dany stan, tylko jakie usługi w wyniku jego działania zostały zainstalowane i są uruchomione

2.4.2 Elementy systemu

2.4.2.1 Manifesty

Manifesty fizycznie to pliki tekstowe z rozszerzeniem .pp, w których używając języka Puppet zapisuję się końcowy spodziewany stan systemu. Podstawowe elementy manifestu to zasobu, które można grupować w klasy i moduły. Przykładowy plik manifestu, który spowoduje zainstalowanie na zarządzanych hostach edytora tekstu vim został pokazany poniżej.

```
1  package {
2  „vim”: {
3         ensure => installed,
4      }
5  }
```

2.4.2.2 Zasoby

Zasoby (ang. „resource”) w nazewnictwie Puppet to określenie takich obiektów zarządzanego systemu jak: pliki, użytkownicy, pakiety, komendy, serwisy itp. Każdy z zasobów może posiadać różne parametry takie jak np. położenie w systemie, uprawnienia lub zawartość. W poniższym przykładzie został zdefiniowany zasób pliku.

```
1  file {
2     „/root/sample.txt”
3        ensure   => file,
4        mode     => 0644,
5        owner    => „root”,
6        content  => „Hello World”,
7  }
```

2.4.2.3 Klasy

Klasy służą do grupowaniu zasobów. W poniższym przykładzie możemy zobaczyć konfigurację serwera SSH, która składa się z zainstalowanego pakietu ssh, przygotowanego pliku z konfiguracją oraz uruchomionego serwisu:

```
1  class ssh {
2     package {
3        „ssh”:
4           ensure => installed,
5     }
6     file {
7        „sshd_config”:
8           path       => „/etc/ssh/sshd_config”
9           ensure     => file,
10          require    => Package[„ssh”]
11    }
12    service {
13       „sshd”:
14          name       => „sshd”,
15          ensure     => running,
16          enable     => true,
17          subscribe  => File[„sshd_config”]
18    }
19 }
```

Przykład opiera się o statyczną konfiguracje, jednak Puppet pozwala na stosowanie zmiennych w klasach, przez co można tworzyć bardziej uniwersalny kod. W przykładzie zostały również zawarte dwie funkcje języka, które pokazują dodatkowe możliwości:

- **require –** definiuje zależności między zasobami. W tym przypadku plik sshd_config zostanie zaktualizowany lub utworzony jeśli pakiet sshd jest obecny w systemie
- **subscribe –** udostępnia możliwość reagowania na zdarzenia. Usługa sshd będzie uruchomiona ponownie, gdy plik sshd_config zostanie zmieniony.

2.4.2.4 Dostawcy

Dostawca (ang. „provider”) to inaczej klasa, w której zawarta jest logika przetwarzania operacji zawartych w zasobach. Każdy jeden typ zasobu ma przypisanego dostawce. Puppet ma zaimplementowany podstawowy zestaw dostawców oraz posiada możliwość pobierania zewnętrznych dostawców przez wtyczki. Pozwala to tworzyć nowe zasoby lub rozszerzać istniejące. Przykładowo, zasób „package” ma poddostawców dla programu zarządzającego pakietami.

2.4.2.5 Moduły

Moduły maja za zadanie łączenie w grupy powiązanych ze sobą części składowych. Każdy moduł może posiadać manifesty, pliki statyczne, wtyczki (małe programy zewnętrznych dostawców), szablony i testy automatyczne. Głównym celem modułów jest rozszerzenie możliwości Puppeta, tak aby mógł nadzorować wszystkie elementy systemu, a konfiguracja wymagała jedynie dostarczenia odpowiednich parametrów.

Moduł na dysku jest reprezentowany przez drzewo katalogów o z góry ustalonej strukturze. Dodanie nowego modułu odbywa się przez umieszczenie go w odpowiednim katalogu zdefiniowanym w konfiguracji Puppeta. Dzięki, czemu nie trzeba aktualizować bazy ani listy modułów.

Przykładowa struktura modułu o nazwie „hello_word” został pokazany poniżej:

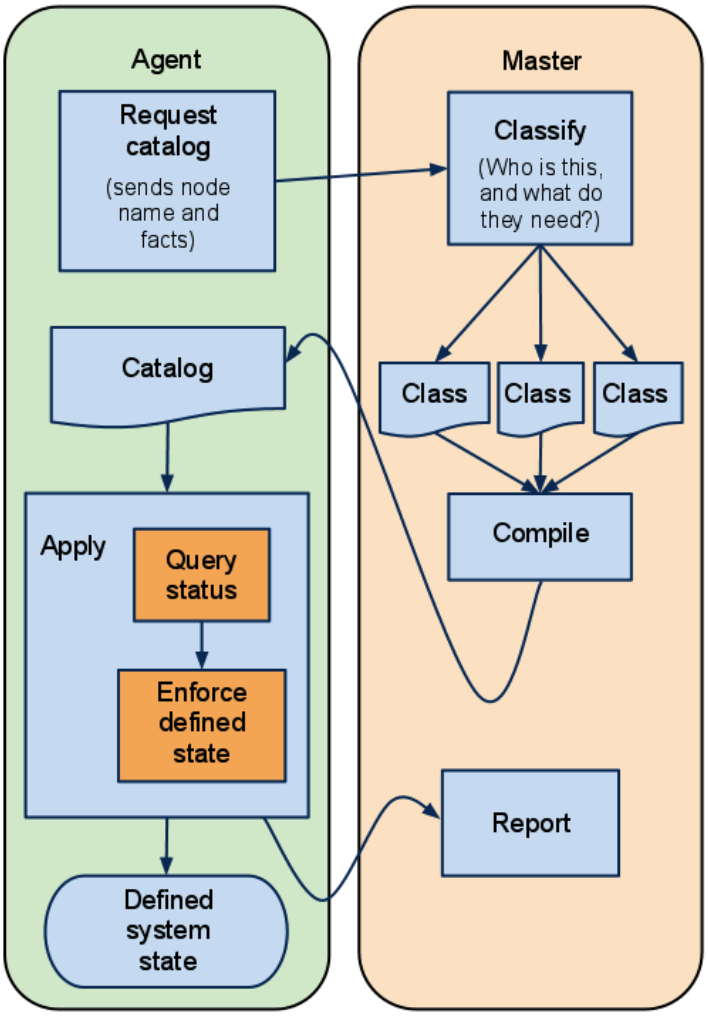
```
1  hello_word/ - katalog modułu
2      manifests/ – zawiera wszystkie manifesty modułu
3          init.pp – podstawowy plik zawierający definicje klasy modułu
4          other_class.pp – zawiera definicje hello_word::other_class
5      files/ - zawiera pliki statyczne
6      lib/ - zawiera wtyczki
7      templates/ – zawiera szablony dla manifestów
8      test/ - zawiera przykłady użycia klas modułu
9          init.pp – przykład użycia głównej klasy modułu
10         other_class.pp – przykład użucia klasy other_class
```

11 `spec/` – zawiera specyfikacje testów dla wtyczek

2.4.3 Zasada działania

Rozwiązanie Puppet może funkcjonować autonomicznie tylko na jednej instancji albo w strukturze klient - serwer (Agent - Master), w z której występuję następująca odpowiedzialność:

- Agent – Klient, który cyklicznie pobiera specyfikacje od Mastera i wdraża zmiany na maszynie na której jest zainstalowany.
- Master – Serwer, który udostępnia specyfikacje Agentom.



Rysunek 12 - Komunikacja w Puppet

Źródło:
https://tr1.cbsistatic.com/hub/i/2015/05/07/13ecf8b9-f499-11e4-940f-14feb5cc3d2a/puppet-agent master.png

Zasada wdrażania konfiguracji na zarządzane hosty została pokazana na rysunku X i składa się z następujących kroków:

1. Agent przy inicjalnym uruchomieniu generuje certyfikat SSL, którym zabezpiecza wiadomości przekazywane do Mastera. Następnie wysyła swoja nazwę oraz fakty czyli informacje na temat maszyny na której jest zainstalowany (takie jak adresy IP, wersja systemy operacyjnego, ilość pamięci RAM itp.) do serwera głównego.
2. Master weryfikuje otrzymane fakty i zapisuje je w manifestach jako zmienne.
3. Później serwer główny sprawdza jak dany host powinien być skonfigurowany i przygotowuje manifesty w zależności od przesłanych parametrów. Dzięki czemu specyfikacja może być przygotowana np. pod konkretną wersje systemu operacyjnego.
4. Następnie Master kompiluje manifesty do katalogu i wysyła do Agenta.
5. Agent wykonuje otrzymany manifest zgodnie z instrukcjami i wysyła raport do Mastera

2.4.4 Wymagania

Minimalne wymagania sprzętowe w monolitycznej strukturze dla serwera głównego zostały przedstawione w tabeli 1 dla różnych ilości kontrolowanych instancji.

ilość instancji	rdzeni procesora	pamięci RAM	przestrzeni dyskowej /var/	przestrzeni dyskowej /opt/
1 - 10	2	6 GB	-	20 GB
10 – 4000	16 +	32 + GB	10 GB	100 GB

Tabela 1 - Wymagania Puppet wersja 4.10

W zależności od wersji oprogramowania Puppet na serwerze głównym mogą być różne wymagania programowe i zależności. Przykładowa instalacja znajduję się w kolejnym rozdziale - 2.3.5.

2.4.5 Instalacja

Instalacja Puppet w systemie Debian w architekturze klient-serwer składa się z następujących etapów:

1. Zainstalowania paczki zawierające zewnętrzne repozytorium

```
1 get
  https://apt.puppetlabs.com/puppetlabs-release-precise.d
  eb
2 dpkg -i puppetlabs-release-precise.deb
```

2. Instalacji pakietu Puppet w zależności od przeznaczenia maszyny:

a) Serwer zarządzający

```
1 apt-get install puppetserver
```

b) Serwer zarządzany

```
1 apt-get install puppet
```

3 Analiza

3.1 Wymagania narzędzia

Dobre narzędzie do automatyzacji wydań oraz zarządzania konfiguracją powinno spełniać następujące wymagania:

- **W1** - Powinno umieć zarządzać zainstalowanymi usługami oraz plikami statycznymi.
- **W2** - Powinno potrafić kontrolować maszyny z różnymi systemami operacyjnymi, takimi jak Windows, Unix i GNU/Linux.
- **W3** - Powinno posiadać kontrolę wersji, tak aby można było śledzić zachodzące zmiany i w razie potrzeby przywrócić poprzednia wersje.
- **W4** - Powinno być wydawanym na zasadach wolnego oprogramowania oraz posiadać płatne wsparcie dla dużych projektów.
- **W5** - Powinno nie posiadać zależności, które muszą zostać zainstalowane na kontrolowanych maszynach.
- **W6** - Powinno pozwolić zarządzać mała infrastrukturą oraz dużym środowiskiem.

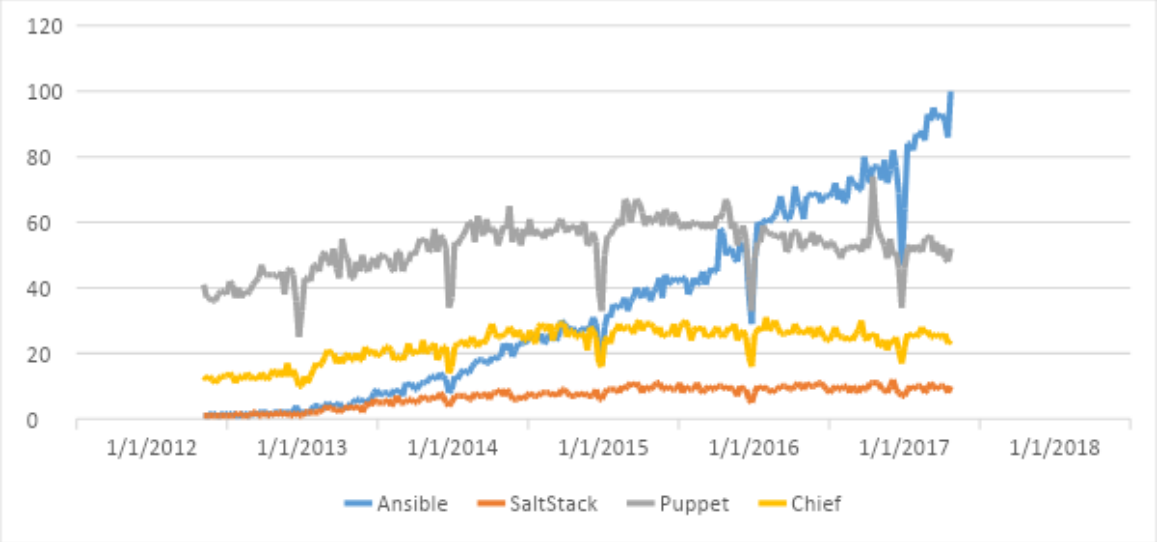
Pod kątem przedstawionych wymagań została stworzona tabela nr 2, która obrazuje jakie narzędzia badane przeze mnie spełniają warunki.

	Ansible	**SaltStack**	**Puppet**	**Chef**
W1	TAK	TAK	TAK	TAK
W3	NIE	NIE	TAK	TAK
W6	TAK	TAK	TAK	TAK
W2	TAK	TAK	TAK	TAK
W5	TAK	NIE	NIE	NIE
W4	TAK	TAK	TAK	TAK

Tabela 2 – Analiza

Źródło: opracowanie własne

3.2 Popularność



Rysunek 13 - Analiza popularności

	Ansible	SaltStack	Puppet	Chef
Contributors	2 702	1 786	442	509
Branches	55	17	11	205
Releases	155	146	303	952
Commits	30 403	81 388	24 834	19 228

Źródło: Google Trends

Tabela 3 - Dane GitHub

Źródło: www.github.com

4 Implementacja

W tym rozdziale opisana została koncepcja wraz z praktyczna implementacją narzędzia Ansible do automatycznego tworzenia i zarządzania środowiskiem: rozwojowym, testowym i produkcyjnym aplikacji internetowej opartej na systemie WordPress.

4.1 WordPress – jako aplikacja

Obecnie najpopularniejsza aplikacja sieciowa wykorzystywana do tworzenia stron to WordPress []. Jest to system zarządzania treścią wydawany jako otwarte oprogramowanie. Został napisany w języku PHP z wykorzystaniem bazę danych MySQL []. Idealnie nadający się dla użytkowników, którzy nie mają doświadczenia w programowaniu i tworzeniu stron internetowych. Jednakże, WordPress posiada liczne rozszerzenie, przez co chętnie używany jest również przez duże firmy.

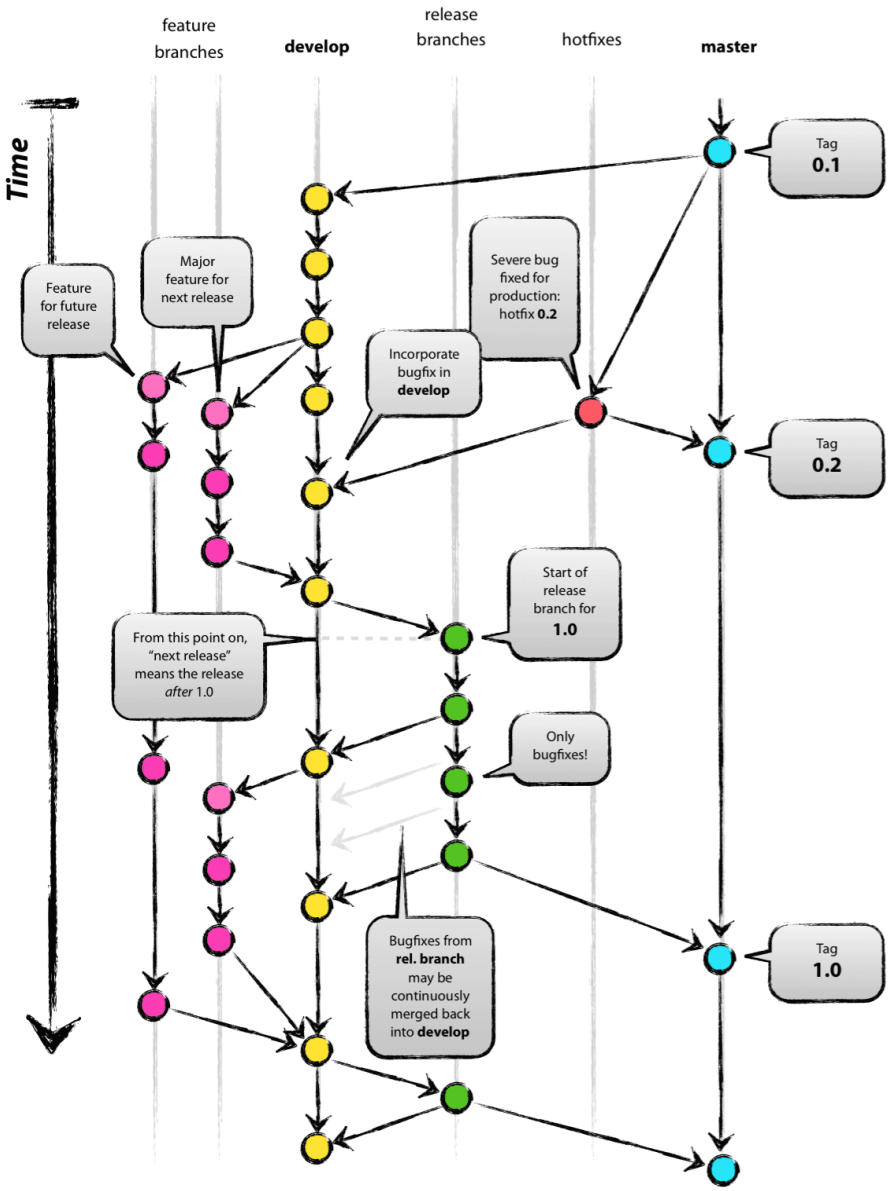
4.2 Git i GitHub - jako repozytorium

GitHub jest serwisem internetowym oferujący hosting przeznaczony dla projektów programistycznych wykorzystujących system kontroli wersji Git. Został stworzony w 2008 roku przy wykorzystaniu freamework-a Ryby on Rails []. Udostępnia darmowy hosting dla programów open source. Dla tego rozdziału tej pracy GitHub został użyty jako główne repozytorium kodu opublikowane pod nazwą: pperzyna/masters-thesis, które zostało podzielone na dwa katalogi:

/ansible – katalog zawierający skrypty Ansible,

/wp – katalog zawierający aplikacje WordPress.

Dzięki wykorzystaniu rozproszonego systemu kontroli wersji Git, każdy z developerów może pracować nie zależnie od siebie. Możemy również podzielić kod na różne branche []. W naszym przypadku konkretne branche będą przetrzymywały kod, który znajduję się na danym środowisku. Jednak istnieje wiele dobrze znanych strategii utrzymywania branchy. Jedna z najbardziej znanych została sformalizowana w 2010 roku przez Vincent Driessen, nazwana Git Flow [ ] i została pokazana na rysunku X. Zastosowana przeze mnie strategia wykorzystania branchy jest uproszczoną wersją Git Flow zgodnie z tabelka nr X.



Rysunek 14 - Git Flow

Źródło: http://nvie.com/img/git-model@2x.png

Branch	Środowisko	Opis
master	production	Środowisko produkcyjne
staging	staging	Środowisko testowe
develop	development	Środowisko rozwojowe

Tabela 4 - Git Flow
Źródło: opracowanie własne

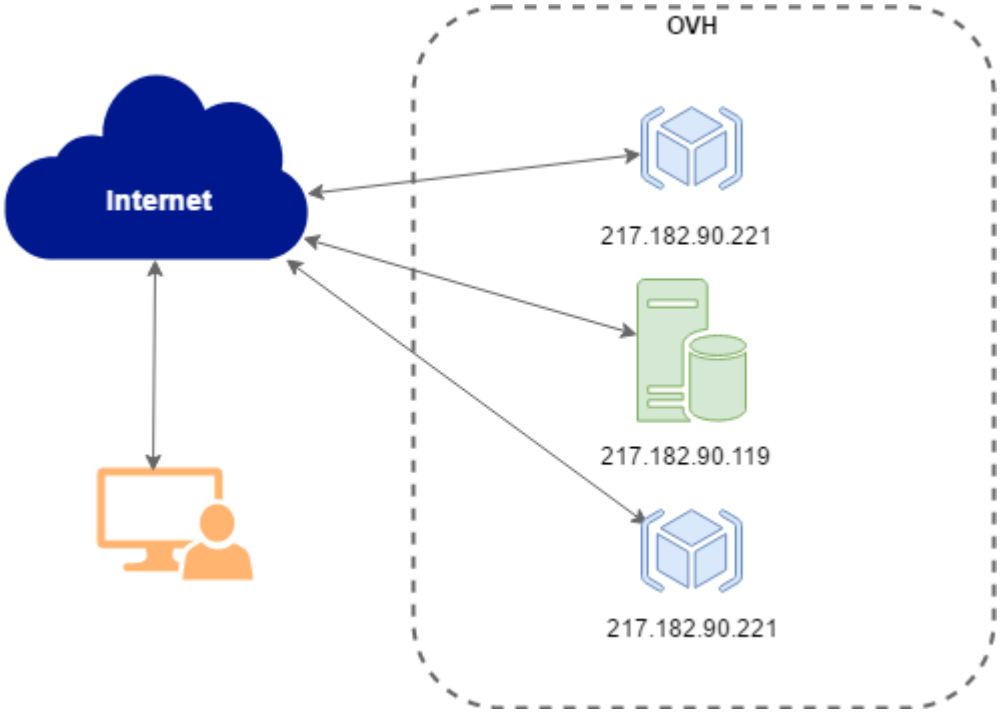
4.3 Środowiska

Środowiska ze względu na

4.4 Playbook

Implementacja Ansible do stworzenia i zarządzania środowiskiem opisanym w rozdziale 4.2 została podzielona na dwa playbooki. Pierwszy z nich ma za zadanie skonfigurowanie elementów serwera głównego - Master. Drugi playbook jest odpowiedzialny za poprawne dodawanie klientów do systemu monitoringu.

4.5 Testowanie



Rysunek 15 - Infrastruktura w OVH

Źródło: opracowanie własne

Działanie przygotowanych playbooków, zostało przetestowane w infrastrukturze firmy OVH, która świadczy usługi hostingowe. Do testów zostały stworzone trzy wirtualne serwery działające przy wykorzystaniu wirtualizacji KVM z zainstalowanym systemem Debian w wersji 9 i parametrami przedstawionymi w tabeli nr 3.

NAZWA	CPU	RAM	HDD

VPS SSD 1	1 x vCore 2,4 GHz	2 GB	10 GB SSD

Tabela 5 - Parametry VPS

Źródło: https://www.ovh.pl/vps/vps-ssd.xml

Wszystkie zamówione VPS podczas instalacji systemu operacyjnego, którą realizowała firma OVH otrzymały zdefiniowany wcześniej klucz publiczny dla użytkownika debian, który zapewniał dostęp przez SSH.

Testowanie playbooków odbyło się w trzech następujących krokach:

1. Stworzenie pliku z hostami o nazwie „hosts”, zgodnie z otrzymanymi adresami IP.

```
n1 ansible_host=217.182.90.219 ansible_ssh_user=debian
n2 ansible_host=217.182.90.220 ansible_ssh_user=debian
n3 ansible_host=217.182.90.221 ansible_ssh_user=debian

[icinga-client]
n2
n3

[icinga-master]
n1
```

2. Uruchomienie playbooków.

```
ansible-playbook -i hosts icinga-server.yml
ansible-playbook -i hosts icinga-client.yml
```

Do wywołanego polecenia została użyta flaga -i mówiąca o tym gdzie znajduje oraz jak się nazywa plik zawierający definicje hostów. Ansible pozwala również na użycie innych flag ułatwiających testowanie, między innymi takich jak:

- -- check – Sprawdzenie jaki mogą być potencjalne zmiany na serwerach.
- -- diff – Sprawdzenie jakie zaszły zmiany w plikach playbooka.

- -- tags = XXX – Zostaną wykonane tylko zadania z tagiem XXX.
- -- limit = XXX – Zostaną uwzględniona tylko grupa XXX

4.6 Wyniki

5 Podsumowanie

6 Bibliografia

[1] Obowiązki administratora

[2] Zalety wykorzystania chmury

[3] Dev ops in agile

[4] software workflow

[5] http://dev2ops.org/2010/02/what-is-devops/

[6] Ansible - https://www.ansible.com/

[7] Chef - https://www.chef.io/

[8] SaltStack - https://saltstack.com/

[9] SaltStack -

[10] Puppet - https://puppet.com/

[11] HALL J., Mastering SaltStack, Packt Publishing, 2015

[12] UPHILL T., Mastering Puppet, Packt Publishing, 2014

[13] KEATING J., Mastering Ansible, Packt Publishing, 2015

[14] JOSHI M., Mastering Chef, Packt Publishing, 2015

[15] VERONA J., Practical DevOps, Packt Publishing, 2016

[16] ANSI/EIA-649

[17]

7 Spis rysunków

Rysunek 1 – Zadania DevOps 10
Rysunek 2 - Różnice wielkości zmian w f. częstotliwości wydań 11
Rysunek 3 - Logo Ansible 13
Rysunek 4 - Schemat działania Ansible 13
Rysunek 5 - Inventory Ansible 14
Rysunek 6 - Ad-Hocs Ping Ansible 15
Rysunek 7 - Ansible PlayBook Ping 15
Rysunek 8 - Dodanie repozytorium Ansible 16
Rysunek 9 - Dodanie podpisu cyfrowego repozytorium Ansible 16
Rysunek 10 - Instalacja Ansible 16
Rysunek 11 - Logo SaltStack 17
Rysunek 12 - Architektura SaltStack 17
Rysunek 13 - Komunikacja SaltStack 18
Rysunek 14 - Komenda SaltStack 18
Rysunek 15 - Salt State 19
Rysunek 16 - Dodanie repozytorium SaltStack 20
Rysunek 17 - Dodanie podpisu cyfrowego repozytorium SaltStack 20
Rysunek 18 - Instalacja pakietu salt-master 20
Rysunek 19 - Instalacja pakietu salt-minion 20
Rysunek 20 - Logo Puppet 21
Rysunek 21 - Instalacja VIM – Puppet 21
Rysunek 22 - Komunikacja w Puppet 22
Rysunek 23 - Instalacja paczki repozytorium Puppet 23
Rysunek 24 - Instalacja pakietu Puppet na serwerze zarządzającym 23
Rysunek 25 - Instalacja pakietu Puppet na serwerze zarządzanym 24
Rysunek 26 - Logo Chef 25
Rysunek 27 - Schemat komunikacji Chef 25
Rysunek 28 - Analiza popularności 28

8 Spis tabel

Tabela 1 – Analiza 21

Wyrażamy zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW.

.....

(czytelny podpis autora)