WYŻSZA SZKOŁA ZARZĄDZANIA I BANKOWOŚCI W KRAKOWIE

Wydział Zarządzania, Finansów i Informatyki

KIERUNEK: Informatyka

SPECJALNOŚĆ: Systemy i sieci komputerowe

PRACA DYPLOMOWA

Grzegorz Michał Łada

Zastosowanie Ansible w zarządzaniu konfiguracją

PROMOTOR:
mgr inż. Witold Rakoczy

KRAKÓW 2020

Spis treści

1 WSTĘP

Współczesne zarządzanie systemami jest sporym wyzwaniem i wymaga bogatej wiedzy z zakresu administracji systemami, programowania i kilku innych działów IT. Zespoły administratorów oraz inżynierów DevOps stawiają czoła złożonym problemom podczas planowania i budowania infrastruktur opartych na chmurach obliczeniowych i paradygmacie infrastruktury programowalnej IaC (ang. Infrastructure as Code). Zespoły te często są odpowiedzialne również za konfigurowanie i wdrażanie serwisów oraz oprogramowania niezbędnego do działania całego systemu IT. Muszą one zapewnić również wysoką dostępność systemu, jego skalowalność, bezpieczeństwo oraz redundantność.

Procesy konfiguracji i utrzymania systemów z infrastrukturą o wysokiej liczbie serwerów są dużym wyzwaniem dla administratorów. Jednym z zadań, jakiemu muszą oni stawić czoła jest ciągłe wdrażanie (ang. continuous deployment) zmian na serwerach o analogicznej konfiguracji. Dla przykładu, takie działanie jest konieczne, gdy infrastruktura w chmurze składa się z kilku podobnych serwerów bazodanowych w celu zagwarantowania wydajności i równomiernego rozłożenia obciążenia (ang. load balancing) na nich. Manualne konfigurowanie kilku identycznych serwerów byłoby żmudnym zadaniem, dlatego korzysta się z możliwości optymalizacji tego procesu, w efekcie skracając czas poświęcany na prace utrzymaniowe wykonywane przez administratorów. Częstym rozwiązaniem jest pisanie własnych skryptów Bash, które wymagają zaawansowanej wiedzy z systemów Unix. W przypadku, gdy pracuje nad nimi zespół administratorów, skrypty te są problematyczne w tworzeniu i dalszym rozwijaniu, ponieważ nie istnieje ściśle zdefiniowana konwencja pisania skryptów w Bash. Ich struktura różni się w zależności od preferencji i przyzwyczajeń autora. Dodatkowo, Bash ma ograniczoną możliwość zbierania informacji o stanie systemów. Przykładowo, niemożliwe jest sprawdzenie czy pliki konfiguracyjne na serwerze zostały zaktualizowane bez naocznego sprawdzenia ich zawartości. W większych firmach częstą praktyką przy rozwijaniu dużych projektów jest rozgałęzienie środowisk na kilka poziomów, np. deweloperskie, testowe i produkcyjne. Konieczne jest wtedy utworzenie procedur do śledzenia, aktualizowania i cofania zmian w każdym z tych środowisk. Obszar działania inżynierii systemów IT, który zajmuje się rozwiązywaniem opisanych problemów to zarządzanie konfiguracją. Istnieje kilka rozwiązań o otwartym kodzie źródłowym, służących do zarządzania konfiguracją. Niniejsza praca poświęca uwagę

rozwiązaniom, które są w powszechnym użyciu przez administratorów systemów. Zespoły używające rozwiązań do zarządzania konfiguracją są w stanie optymalizować czas poświęcany na śledzenie, monitorowanie i tworzenie kopii zapasowych istniejących już serwerów. Dodatkowo, mogą szybko uruchamiać nowe serwisy, gdyż posiadają napisane wcześniej kody procedur instalacyjnych. Praca ta ma pięć rozdziałów i demonstruje jak narzędzia do zarządzania konfiguracją ułatwiają pracę administratorom systemów przy wielkoskalowych infrastrukturach. Drugi rozdział zostanie poświęcony przedstawieniu i porównaniu kilku najpopularniejszych narzędzi do zarządzania konfiguracją. Trzeci rozdział w całości dotyczy jednego z nich, Ansible, i jego zastosowań wraz z opisanymi przykładami użytkowania. Czwarty rozdział natomiast zaprezentuje zastosowanie Ansible w zarządzaniu siecią typu IoT (ang. Internet of Things) urządzeń pomiarowych i czujników.

2 PRZEGLĄD NARZĘDZI DO ZARZĄDZANIA KONFIGURACJĄ

Niniejszy rozdział ma na celu zaprezentowanie najpopularniejszych narzędzi do zarządzania konfiguracją. Na początku zostanie zdefiniowany zakres tematyki zarządzania konfiguracją oraz kryteria doboru narzędzi. Następnie zostaną opisane wybrane narzędzia wykorzystywane w tej dziedzinie oraz dokonane zostanie ich porównanie według zdefiniowanych kryteriów.

2.1 Wprowadzenie do zarządzania konfiguracją

Zarządzanie konfiguracją jest procesem w inżynierii systemów i oprogramowania, polegającym na ciągłym śledzeniu zmian podczas rozwoju produktu końcowego. Ma on na celu zapewnienie i zachowanie zgodności poszczególnych elementów całego systemu w trakcie procesu jego rozwoju. Poza informatyką, ma on również zastosowanie w innych dziedzinach takich jak przemysł militarny, inżynieria czy produkcja. W informatyce istnieją dwa rodzaje zarządzania konfiguracją: systemu i oprogramowania.

Zarządzanie konfiguracją oprogramowania (ang. Software Configuration Management – SCM) jest używane do śledzenia i wprowadzania zmian w tworzonej aplikacji, dzięki czemu twórcy aplikacji mają pewność, że wytworzone przez nich oprogramowanie spełnia wymagania klienta.

Zarządzanie konfiguracją systemów (ang. Configuration Management – CM) z kolei polega na wdrażaniu, śledzeniu oraz utrzymaniu serwisów i aplikacji na serwerze. Proces ten może być dodatkowo wspierany przez tworzenie i zapisywanie zrzutów czy kopii zapasowych stanu konfiguracji systemu w celu przywrócenia go w razie niespodziewanej awarii. Narzędzia służące do tego celu będą dalej nazywane narzędziami CM.

Niniejsze opracowanie skupia się na zarządzaniu konfiguracją systemów i narzędziach do tego używanych.

Istnieje obecnie wiele rozwiązań programistycznych wykorzystywanych do zarządzania konfiguracją systemów, charakteryzujących się różnymi właściwościami. Poniżej zostanie zdefiniowanych kilka kryteriów, według których zostaną one porównane w dalszej części tego rozdziału.

Poniżej przedstawiono kryteria, według których administratorzy systemów mogą dokonać wyboru narzędzia do zarządzania konfiguracją, optymalnego dla swojego środowiska pracy czy danego zadania:

1. **Możliwość wdrażania, aktualizacji i usuwania plików konfiguracyjnych i serwisów**

   Podstawowe operacje, które każde narzędzie CM powinno wspierać.

2. **Czytelność kodu**

   Wielkoskalowe systemy są stawiane i utrzymywane przez kilkuosobowe zespoły, dlatego dobre narzędzie CM powinno mieć wysoce czytelną przez człowieka składnię kodu i plików konfiguracyjnych.

3. **Możliwość wykonywania pojedynczych komend**

   Czasami administrator systemu musi wykonać szybko drobną zmianę w konfiguracji serwera. Powinien móc to wykonać bez potrzeby tworzenia zaawansowanego projektu w narzędziu CM. Dla przykładu, powinien mieć możliwość aktualizacji pamięci cache repozytoriów dla wszystkich zarządzanych urządzeń, bez potrzeby tworzenia osobnego projektu dla pojedynczego zadania.

4. **Skalowalność**

   Dodawanie nowych usług i konfiguracji powinno być łatwe i wykluczać konieczność większych modyfikacji kodu. Dzięki temu narzędzie CM może być używane przy małych i dużych rozmiarów infrastrukturze.

5. **Wsparcie systemów kontroli wersji**

   Narzędzie do zarządzania konfiguracją powinno wspierać integrację z usługami takimi jak Git[1] – usługa do przechowywania i wersjonowania kodu, aby umożliwić śledzenie zmian w konfiguracji i przywracanie systemu do poprzedniej wersji w razie potrzeby.

[1] Git. https://git-scm.com/book/pl/v2/Pierwsze-kroki-Podstawy-Git

6. **Wieloplatformowość**
   Dobre narzędzie CM powinno pozwalać na wykonywanie zmian na większości systemów z rodziny Unix

7. **Bezagentowość**
   Brak konieczności instalowania dodatkowego oprogramowania na zarządzanych urządzeniach ułatwia wstępną konfigurację i użytkowanie narzędzia CM.

8. **Zewnętrzne wsparcie**
   Możliwość uzyskania wsparcia dużej i aktywnej społeczności oraz dostęp do obszernej i dobrze napisanej dokumentacji narzędzia CM ułatwia jego użytkowanie, zwłaszcza nowicjuszom.

9. **Cena i otwarty kod źródłowy**
   Dostępne są darmowe wersje, wprawdzie ograniczone czasowo, których zaletą jest możliwość przetestowania funkcjonalności oraz otwarty kod źródłowy narzędzia. Dla wielkoskalowych systemów zalecane są komercyjna licencja i kontraktowane wsparcie techniczne.

Dalsza część pracy zostanie poświęcona przeglądowi wybranych dostępnych narzędzi CM, po czym zostaną one porównane według powyżej zdefiniowanych kryteriów.

2.2 Chef

W oparciu o dziewięć kryteriów wyboru narzędzia do zarządzania konfiguracją, charakterystyka narzędzia Chef jest następująca**:**

- ☑ Wdrażanie, aktualizacja i usuwanie plików konfiguracyjnych i serwisów na serwerze
- ☐ Czytelność kodu
- ☐ Wykonywanie pojedynczych komend
- ☑ Skalowalność
- ☑ Wsparcie systemów kontroli wersji
- ☑ Wieloplatformowość

☐ Bezagentowość

☑ Zewnętrzne wsparcie

☑ Cena i otwarty kod źródłowy

Chef jest wolnoźródłowym narzędziem CM, napisanym w języku Ruby. Jego składnia i sposób użytkowania czynią go lepszym narzędziem dla osób znających specyfikę pracy programisty tego języka. Sposób, w jaki pliki konfiguracyjne, serwisy i aplikacje będą umieszczane na serwerach jest opisywany w języku programowania Ruby, którego znajomość jest wymagana już od początku używania Chefa. Chef bazuje na modelu master-client, ale z jedną zmianą – serwery mogą być zarządzane nie tylko z master-node, ale również z poziomu stacji roboczej użytkownika. Model ten poniżej zaprezentowany schemat (Diagram 2.1).

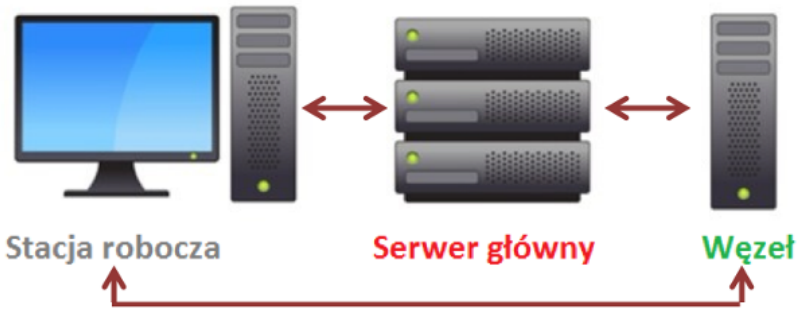


Diagram 2.1. Model komunikacji w Chef

Takie podejście jest bardzo podobne do przebiegu procesu rozwoju programowania w zespołach programistów. Składają się na niego następujące kroki:

1. Napisanie kodu na stacji roboczej:

- Kod jest zorganizowany w *przepisy*, które dalej są grupowane wraz z innymi plikami tworząc procedurę wykonywania, w nomenklaturze Chefa zwaną cookbook, o której więcej będzie w dalszej części pracy. Aby umożliwić komunikowanie się z głównym serwerem, wymagane jest zainstalowanie zestawu narzędzi dla programistów w postaci pakietu SDK oraz stworzenie pliku z główną konfiguracją Chefa o nazwie `knife.rb`. Zawiera on informacje o serwerze głównym i lokalizacje kluczy prywatnych.

2. Przesłanie kodu do serwera głównego:

- Jest to krok analogiczny do procesu przesyłania gotowego kodu do zdalnego repozytorium, w którym przechowywane są pliki cookbooka. Możliwa jest konfiguracja własnego serwera głównego albo użycie tego dostarczanego przez Chefa. Należy zauważyć, że ta druga opcja jest płatna. W większości przypadków, serwer główny jest używany zarówno do komunikacji z klientami, jak i stacją roboczą administratora. Aby umożliwić komunikację między stacją roboczą a serwerem głównym, klucz publiczny węzła klienckiego musi być dodany do serwera głównego.

3. Wykonanie bootstrap[2] albo uruchomienie cookbooków na podłączonych klientach Chefa ze stacji roboczej lub serwera głównego.

- Aby wykonać bootstrap węzła klienckiego, wymagany jest aktywny dostęp do niego poprzez protokół SSH ze stacji roboczej. W następstwie tego możliwe jest aplikowanie cookbooków i ustanowienie połączenia między klientami a serwerem głównym za pomocą następującej komendy (Fragment 2.2):

```
1   $ knife bootstrap ADDRESS --ssh-user USER --sudo
    --identity-file PRIV_KEY --node-name node1 --
2   run-list 'recipe[apache2]'
```

Fragment 2.2. Bootstrap klientów i mastera w Chef

Podczas wykonywania procesu bootstrapu, pakiet chef-client jest instalowany na maszynie klienckiej, następnie jest on wiązany z serwerem głównym, a na końcu pliki wymienione w parametrze `run-list` będą wykonane. Aby zweryfikować czy węzeł kliencki został podłączony poprawnie, należy użyć jednego z poniższych poleceń na węźle master (Fragment 2.3):

[2] Chef. Instalacja Bootstrap w Chef. https://docs.chef.io/install_bootstrap/

```
1    $ knife node list
2    $ knife node show
```

Fragment 2.3. Weryfikacja klientów w Chef

Przy próbie kolejnego wykonania, nie będzie zachodziła konieczność ponownego przesłania plików cookbooka do serwera oraz wiązania z węzłem klienckim. Zaktualizowane cookbooki mogą być uruchamiane za pomocą polecenia `chef-client` wykonywanym na węźle klienckim. Poniżej przedstawiono przykładowe polecenie do wykonania na stacji roboczej[3]:

```
1    $ knife ssh ADDRESS 'sudo chef-client' --manual-list
     --ssh-user USER --identity-file PRIV_KEY
```

Fragment 2.4. Wykonanie cookbooka na kliencie

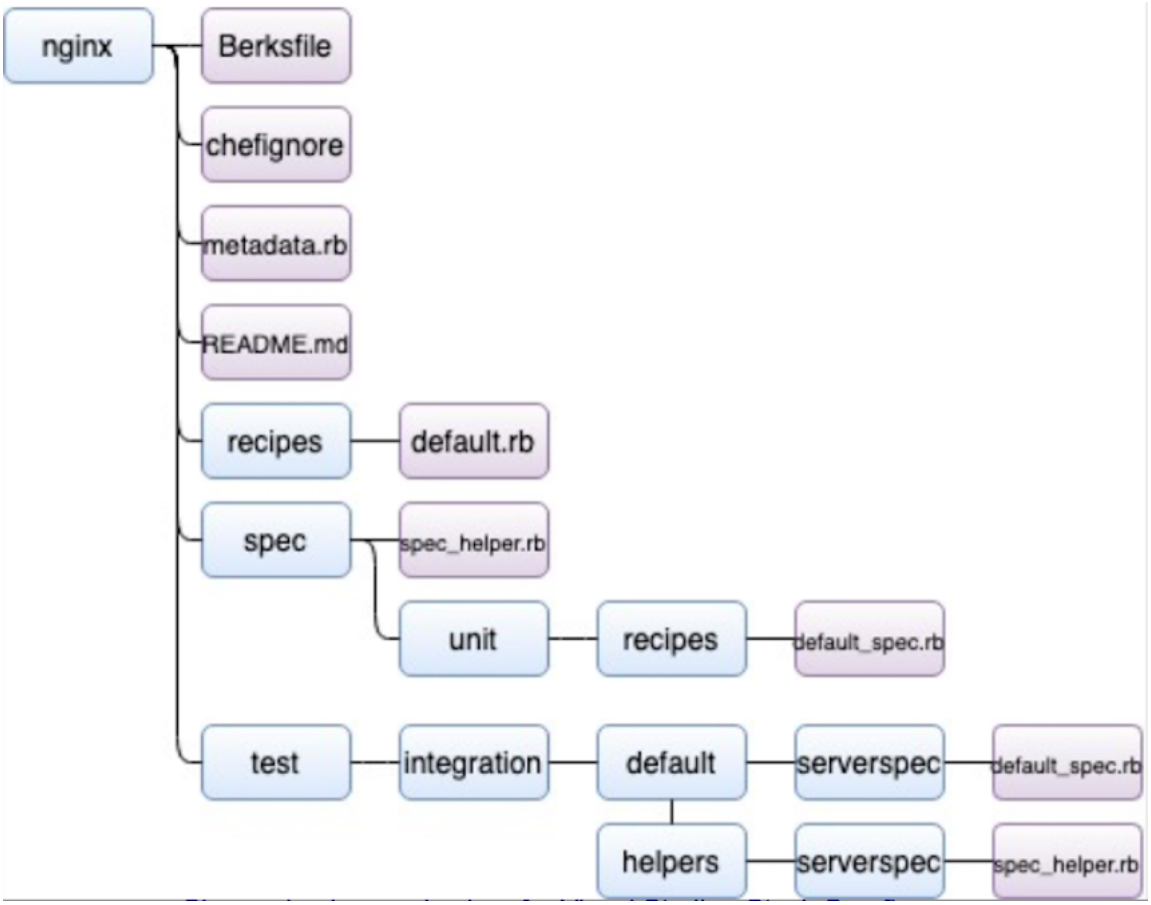
Projekty w Chefie można inicjalizować za pomocą poniższej komendy:

```
1    $ chef generate cookbook nginx
```

Fragment 2.5. Tworzenie szkieletu projektu w Chef

W wyniku wykonania tego polecenia zostanie utworzony `cookbook` o nazwie `nginx` i będzie posiadał następującą strukturę:

[3] Chef. Aktualizacja węzłow w Chef. https://learn.chef.io/manage-a-node/ubuntu/update-your-nodes-configuration/.



Fragment 2.6. Szkielet projektu w Chef

Istotnym katalogiem w strukturze cookbooka jest katalog `recipes`. Przechowywane są w nim pliki zawierające tzw. `resources`. Są to fragmenty kodu języka Ruby, które op isują jak powinien zostać skonfigurowany zarządzany system. Przykładowo, jeśli zadanie polega na zdefiniowaniu przepisu na konfigurację serwera Nginx, to należy stworzyć plik `nginx.rb` w katalogu `nginx/recipes` z następującą zawartością (Fragment 2.7):

```
1  # install nginx package
2  package 'nginx'
3  # ensure that it is enabled and started
4  service 'nginx' do supports :status => true
   action [:enable, :start]
5  end
```

Fragment 2.7. Przepis Chef dla serwera nginx

Tak przygotowany cookbook można przesłać do serwera głównego i uruchomić go na wybranych węzłach klienckich.

Chef jest dobrym wyborem dla zespołu inżynierów DevOps ze względu na zbliżony do programowania sposób pisania cookbooków, szczególnie gdy w zespole kod aplikacji końcowej jest pisany w języku Ruby. Chef posiada doskonałe wsparcie systemów kontroli wersji. W efekcie, stanowi dobry wybór w większych firmach czy korporacjach, które często dokonują zmian na swoich serwerach. Dodatkowo, społeczność skupiona wokół Chefa[4] udostępnia bogaty wybór cookbooków napisanych przez swoich członków. Warto wspomnieć, że Chef wspiera podejście „napisz raz, uruchom wszędzie". Oznacza to, że wcześniej zaimplementowane cookbooki mogą być ponownie wykorzystane na różnych typach infrastruktur. Chef ma również dobrze napisaną dokumentację zawierającą liczne przykłady.

Istnieją również cechy przemawiające przeciwko używaniu Chefa. Jednym z nich jest konieczność znajomości języka Ruby, niezbędna do pisania kodu cookbooków. Dodatkowo, używanie Chefa w dużych zespołach może prowadzić do dezorganizacji w tworzonych projektach, gdy kilku członków zespołu będzie jednocześnie wprowadzać zmiany w kodzie. Jest to sytuacja analogiczna do rozwiązywania konfliktów w Git. Ponadto, Chef nie jest zalecany dla początkujących użytkowników ze względu na programistyczny sposób pracy z tym narzędziem.

2.3 Puppet

Narzędzie Puppet spełnia poniższe kryteria:

- ☑ Wdrażanie, aktualizacja i usuwanie plików konfiguracyjnych i serwisów na serwerze
- ☑ Czytelność kodu
- ☐ Wykonywanie pojedynczych komend
- ☑ Skalowalność
- ☐ Wsparcie systemów kontroli wersji
- ☑ Wieloplatformowość
- ☐ Bezagentowość
- ☑ Zewnętrzne wsparcie
- ☑ Cena i otwarty kod źródłowy

[4] Chef. Społeczność. https://community.chef.io/

Puppet jest oprogramowaniem do zarządzania konfiguracją z otwartym kodem źródłowym. W porównaniu do innych narzędzi zarządzania konfiguracją jest najdłużej, bo aż od 15 lat, istniejącym projektem. Projekt należy do firmy Puppet Labs, przez którą jest rozwijany od 2005 roku. Jej założycielem jest Luke Kanies. Oprogramowanie to napisane jest w języku Ruby, dostępne zarówno w wersji komercyjnej, jak i enterprise. Puppet jest oparty o model master-client i jest używany do testowania, wdrażania oraz śledzenia zmian w plikach konfiguracyjnych na zarządzanych serwerach. Ponadto, Puppet umożliwia tworzenie skalowalnej infrastruktury oraz wdrażanie serwisów za pomocą pojedynczego polecenia.

Proces instalacji mastera i klientów Puppeta nie jest skomplikowany i wygląda następująco (Fragment 2.8):

```
1    $ apt-get install puppet      # On clients (nodes)
2    $ apt-get install puppetmaster    # On server (master)
```

Fragment 2.8. Instalacja Puppeta

Do opisu konfiguracji Puppet wykorzystuje język dziedzinowy (DSL), podobny pod kątem składni do języka JSON, a dane przechowuje w plikach nazywanych **manifestami.** W plikach tych użytkownik może przechowywać tzw. **zasoby**. Zasoby przechowują listę plików, pakietów, paczek systemowych i serwisów, jakie powinny być zainstalowane na zdalnym systemie oraz sposób ich konfiguracji.

Poniżej (Fragment 2.9) zaprezentowano przykładową deklarację zasobu[5], który modyfikuje atrybuty pliku `/etc/passwd`:

```
1    file { '/etc/passwd':
2    ensure => file ,
3    owner => 'root',
4    group => 'root',
5    mode => '0644',
6    }
```

Fragment 2.9. Deklaracja zasobu Puppeta

[5] Puppet. Deklaracja zasobu. https://www.tutorialspoint.com/puppet/puppet_coding_style.htm

Zasoby są grupowane i zawierane w **klasach**, które za pomocą dodatkowych parametrów mogą kontrolować ich zachowanie podczas uruchomienia[6]. Dla przykładu, poniżej została zdefiniowana klasa, w której zawarte są parametry do konfiguracji plików `/etc/passwd` i `/etc/shadow`. Łatwo zauważyć, że klasa ta składa się z dwóch zasobów i używa zmiennej `user`, która definiuje właściciela i jego przynależność do grupy w systemie dla tych dwóch plików, co obrazuje poniższy fragment kodu (Fragment 2.10):

```
1  class linux (String $user = 'root') {
2    file { '/etc/passwd':
3      owner => $user,
4      group => $user,
5      mode => '0644',
6    }
7    file { '/etc/shadow':
8      owner => $user,
9      group => $user,
10     mode => '0440',
11   }
```

Załącznik 2.10. Klasa w Puppet

W oprogramowaniu Puppet istnieją również **moduły**, które umożliwiają rozdzielenie bloków kodu do różnych manifestów (plików z opisem konfiguracji). Istnieje społeczność nazywana **Puppet Forge**, której członkowie udostępniają za darmo stworzone przez siebie moduły.[7]

Proces wprowadzania zmian w konfiguracji za pomocą Puppeta przedstawia poniższy schemat (Diagram 2.11):

[6] Puppet. Klasy. https://docs.puppetlabs.com/puppet/latest/reference/lang_classes.html
[7] Puppet. Puppet Forge. https://forge.puppet.com/

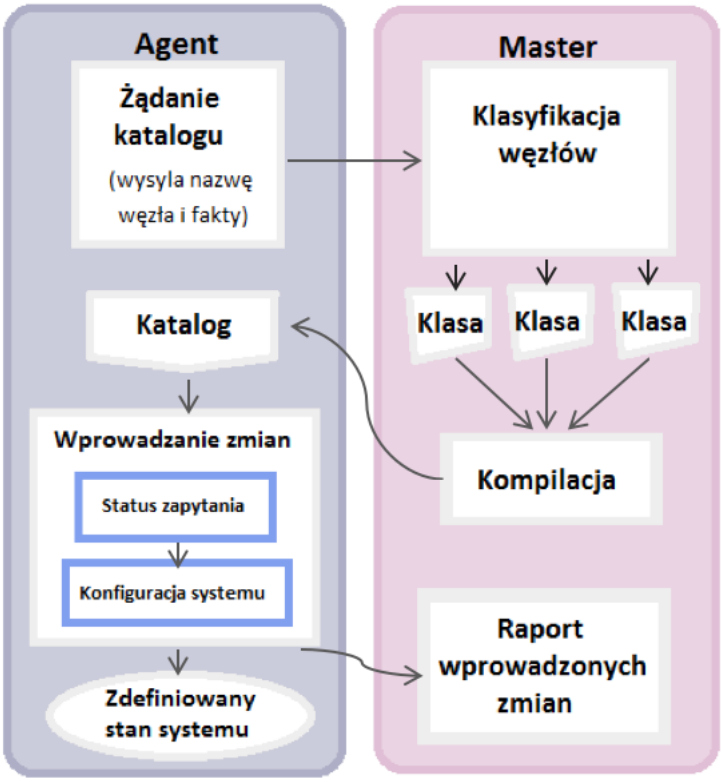


Diagram 2.11. Model komunikacji w Puppet

Składa się on z czterech kroków:

1. **Agent** wysyła odczytane z zastanej konfiguracji systemu fakty i zmienne, a następnie wysyła je do serwera **Master**.
2. **Master** weryfikuje otrzymane fakty i zmienne, a następnie ustala jakie zmiany powinny zostać wprowadzone na danym hoście zarządzanym.
3. **Master** przygotowuje manifesty do uruchomienia, kompiluje je i wysyła do agenta.
4. **Agent** otrzymuje skompilowane manifesty w postaci **katalogów,** wykonuje je na zarządzanym hoście, a następnie wysyła raport z wyniku działania do węzła **Master**.

Podsumowując, Puppet jest dobrym wyborem, gdy najważniejszymi z wymaganych kryterów są stabilność oraz dojrzałość rozwiązania - relatywnie długi czas istnienia na rynku oprogramowania do zarządzania konfiguracją. W porównaniu do innych narzędzi do zarządzania konfiguracją, Puppet sprawdza się lepiej w systemach o dużej skali niż do wykonywania pojedynczych zadań. Posiada on rozwiniętą społeczność Puppet Forge,

która dzieli się licznymi modułami konfiguracyjnymi oraz dobrze napisaną dokumentację. Dodatkowo, interfejs Puppeta jest jednym z najlepiej rozwiniętych w tym segmencie oprogramowania i można go uruchomić na wielu systemach operacyjnych.

Nie jest to narzędzie pozbawione wad, które uwidaczniają się przy wykonywaniu bardziej kompleksowych zadań. Oprócz umiejętności pisania klas, potrzebna jest również znajomość konsoli Puppeta, która jest napisana w języku Ruby. Oznacza to, że w niektórych przypadkach konieczne będzie zrozumienie składni tego języka. Dodatkowo, duża liczba plików-manifestów utrudnia utrzymanie i dalszy rozwój projektu oraz może utrudniać zrozumienie jego kodu przez nowych członków zespołów DevOps.

2.4 SaltStack

SaltStack spełnia poniższe kryteria:

- ☑ Wdrażanie, aktualizacja i usuwanie plików konfiguracyjnych i serwisów na serwerze
- ☑ Czytelność kodu
- ☑ Wykonywanie pojedynczych komend
- ☑ Skalowalność
- ☐ Wsparcie systemów kontroli wersji
- ☑ Wieloplatformowość
- ☑ Bezagentowość
- ☐ Zewnętrzne wsparcie
- ☑ Cena i otwarty kod źródłowy

SaltStack jest narzędziem do zarządzania konfiguracją o otwartym kodzie źródłowym, który pozwala na wykonywanie pojedynczych poleceń. Jest dość popularny wśród developerów chmur obliczeniowych, ze względu na jego skalowalność i elastyczność. Projekt ten rozpoczął Thomas Hatch w 2011 roku. Został on napisany w języku Python. SaltStack ma modularną budowę, co pozwala użytkownikom na podłączanie i wyłączanie z wykonania różnych pakietów. Można go wykorzystywać nie tylko do konfigurowania serwerów, ale również do wykonywania pojedynczych poleceń i monitorowania pracy systemów. Większość plików

konfiguracyjnych w SaltStack jest pisana w języku formalnym YAML, który jest bardzo łatwy w odczycie przez człowieka. Do implementowania bardziej zaawansowanych plików konfiguracyjnych wykorzystywany jest język szablonowania Jinja2, który oferuje więcej możliwości, jednakże jest mniej czytelny podczas analizy kodu. SaltStack bazuje na architekturze master-client i składa się z następujących komponentów[8]:

- **Salt-Master** – system odpowiedzialny za wysyłanie komend i plików konfiguracyjnych do tzw. salt-minionów. Spełnia rolę analogiczną do serwera głównego w klastrze.
- **Salt-Miniony** – systemy klienckie, które otrzymują komendy i konfiguracje z salt-mastera. Pełnią podobna rolę do węzłów podległych w klastrze.
- **Moduły** - komendy które są wykonywane z salt-mastera na salt-minionach. Używane są głównie do wprowadzania szybkich poprawek i monitoringu w czasie rzeczywistym. Przykładowo, wykonując poniższe polecenie (Fragment 2.12)

```
1    $ salt-ping '*'
```

Fragment 2.12. Pingowanie salt-minionów w SaltStack

nastąpi próba pingu na wszystkich salt-minionach (wyszczególnione przez `'*'`) podłączonych do salt-mastera, z których wykonano tę komendę.

Ścieżkę przepływu danych między salt-masterem, a podlegającymi mu salt-minionami prezentuje poniższy schemat (Diagram 2.13):

[8] SaltStack. Architektura SaltStack. https://docs.saltstack.com/en/getstarted/overview.html

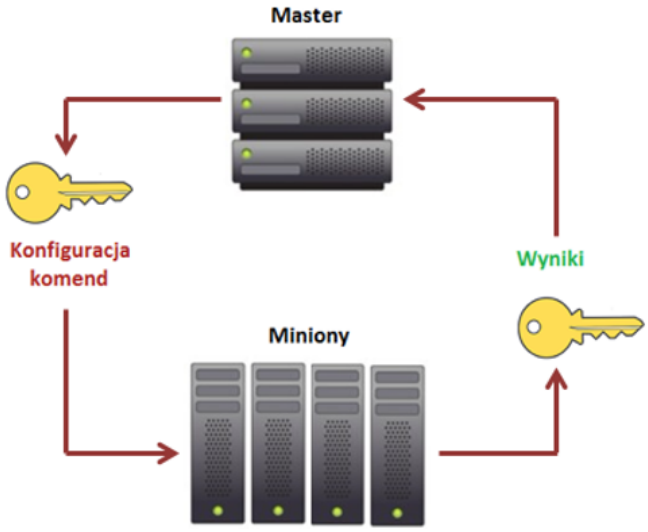


Diagram 2.13. Komunikacja między masterem a minionami w SaltStack

- **Formuły** – opisują stan konfiguracji systemu. Przykładowo, poniższy przykład komendy (Fragment 2.14)

```
1    $ salt '*' state.sls redis
```

Fragment 2.14. Sprawdzenie stanu serwisu Redis

zwróci w wyniku działania status serwisu Redis na wszystkich salt-minionach i zwróci w wyniku następującą odpowiedź (Fragment 2.15):

```
1    redis-server: pkg.installed:
2    service:
3    - name: redis -server - running
```

Fragment 2.15. Sprawdzanie stanu serwisu Redis - odpowiedź

- **Grains** – zmienne systemowe, które przechowują informacje o systemie operacyjnym, pamięci i innych właściwościach zarządzanego systemu;
- **Pillars** – zmienne zdefiniowane przez użytkownika, które zawierają informacje o portach, ścieżkach plików, parametrach konfiguracji oraz hasła uwierzytelniające. Są one zdefiniowane w sposób niejawny na salt-masterze i powiązane z jednym lub więcej salt-minionami. Dla przykładu, można definiować zmienne w zależności od systemu operacyjnego miniona, używając do tego struktury nazywanej w nomenklaturze

SaltStack pillar. Przykład takiej struktury prezentuje poniższy fragment kodu (Fragment 2.16):

```
1  {% if grains['os'] == 'RedHat' %}
2  apache: httpd
3  {% elif grains['os'] == 'Debian' %}
4  apache: apache2
5  {% endif %}
```

Fragment 2.16. Ustalanie nazwy apache w SaltStack

Jeśli komenda zostanie wykonana w systemie z rodziny `RedHat`, to zmienna `apache` przyjmie wartość `httpd`. Jeśli będzie to system operacyjny z rodziny `Debian`, to zmienna `apache` przyjmie wartość `apache2`.

- **Runners** – moduły, które są dostępne na salt-masterze, wykorzystywane do wykonywanie różnych operacji na salt-minionach.
- **Returners** – moduły, które mogą być wykonane na salt-minionach do przekazywania danych do zewnętrznych serwisów takich jak np. bazy danych.
- **Reactros** – wyzwalacze, które wykonują swoje zadanie jako reakcja na inne wydarzenie.
- **Salt Cloud** – pozwala na zarządzanie systemami korzystającymi z różnych dostawców chmur, jak np. OpenStack.

SaltStack składa się z dużej liczby komponentów, z których każdy spełnia inną rolę. Warto zauważyć, że w przeciwieństwie do innych narzędzi do zarządzania konfiguracją, salt-master podczas komunikacji z salt-minionami wykorzystuje dwuetapowe uwierzytelnianie i szyfruje cały ruch sieciowy pomiędzy nimi. Dodatkowo, wspiera on również komunikację bezagentową. Oznacza to, że jest możliwe wykonywanie pojedynczych poleceń z salt-mastera, bez konieczności instalacji dodatkowych agentów na zarządzanych salt-minionach. Dużą zaletą SaltStacka jest niezależność od platformy, co ujednolica system i pozwala na wykonywanie identycznych poleceń na systemach Windows i Unix. Dla mniej zaawansowanych użytkowników został stworzony graficzny interfejs webowy.

SaltStack ma też słabsze strony – jego próg wejścia jest stosunkowo wysoki, gdyż wymaga on przygotowania konfiguracji dla wielu komponentów nawet w przypadku nieskomplikowanych zadań. Dodatkowo, dokumentacja jest stosunkowo trudna do zrozumienia, gdyż wymaga posiadania wiedzy z zakresu administracji systemami. Pomimo tego, że jest to narzędzie wspierające wieloplatformowość, istnieją pewne problemy na systemach innych niż Linux. Przykładowo, salt-miniony zainstalowane na systemach Windows są o wiele wolniejsze w responsywności.[9] Dodatkowym problemem jest to, że webowy interfejs graficzny SaltStack nie jest w pełni zoptymalizowany i intuicyjny dla wielu użytkowników.

2.5 Podsumowanie

W rozdziale drugim przedstawiono narzędzia do zarządzania konfiguracją SaltStack, Puppet i Chef. Mimo wielu wspólnych cech, sposoby ich implementacji różnią się. Wymienione narzędzia spełniają od sześciu do siedmiu ze wszystkich dziewięciu kryteriów, podczas gdy Ansible spełni osiem z nich, co uzasadnia poświęcenie mu większej uwagi. Dlatego następny rozdział w całości skupi się na Ansible.

[9] SaltStack. Wolne miniony w Windows. https://github.com/saltstack/salt/issues/48882

3 ANSIBLE

Rozdział ten zostanie w całości poświęcony narzędziu do zarządzania konfiguracją, które okazuje się spełniać najwięcej z obranych kryteriów doboru – Ansible. Po wstępnej analizie zostaną omówione jego poszczególne komponenty i funkcjonalności. W końcowej części tego rozdziału zostaną zestawione zalety i wady tego narzędzia.

Na rynku usług informatycznych uważa się, że Ansible jest optymalnie wyważonym narzędziem do zarządzania konfiguracją, biorąc pod uwagę zaprezentowane kryteria. Praktycznemu wykorzystaniu Ansible i implementacji tzw. playbooków zostanie poświęcony rozdział czwarty niniejszej pracy.

3.1 Wprowadzenie

Ansible spełnia następujące kryteria:

- ☑ Wdrażanie, aktualizacja i usuwanie plików konfiguracyjnych i serwisów na serwerze
- ☑ Czytelność kodu
- ☑ Wykonywanie pojedynczych komend
- ☑ Skalowalność
- ☐ Wsparcie systemów kontroli wersji
- ☑ Wieloplatformowość
- ☑ Bezagentowość
- ☑ Zewnętrzne wsparcie
- ☑ Cena i otwarty kod źródłowy

Ansible jest narzędziem do automatyzacji rozmaitych procesów IT z otwartym kodem źródłowym. Pozwala na automatyczne wdrażanie aplikacji, zarządzanie konfiguracją i orkiestrację bardziej złożonych czynności nie tylko na serwerach, ale również innych urządzeniach, jak np. routery. Pomysłodawcą i twórcą projektu Ansible, po raz pierwszy opublikowanego w 2012 roku, był Michael Dehaan.[10] W 2015 roku prawa do spółki Ansible

[10] Ansible. https://en.wikipedia.org/wiki/Ansible_(software). Data odczytu 14.03.2020 r.

zostały wykupiona przez firmę Red Hat, której przedstawiciele zakup ten skomentowali następująco:[11]

> „Widzimy w Ansible idealne dopasowanie do podstawowych zasad, które kształtują zarządzanie firmą Red Hat, zarówno na poziomie produktowym jak i portfolio."

Ansible nie wymaga instalacji dodatkowego oprogramowania w zarządzanych przez niego systemach, czyli jest bezagentowy. Wymaga jednak obecności na zarządzanych serwerach klienckich uruchomionego serwera OpenSSH i pakietu Python, które są domyślnie dostępne w większości systemów operacyjnych z rodziny Unix. Komunikacja między zarządzającą stacją roboczą a węzłami zarządzanymi odbywa się za pomocą protokołu SSH.[12]

[11] Red Hat. Powody wykupu Ansible. https://www.redhat.com/en/about/blog/why-red-hat-acquired-ansible. Data odczytu 14.03.2020 r.
[12] Ansible. Jak działa Ansible. https://www.ansible.com/overview/how-ansible-works. Data odczytu 15.03.2020 r.

Schemat działania Ansible jest nieskomplikowany. Model komunikacji jest zaprezentowany na poniższym schemacie (Diagram 3.1):

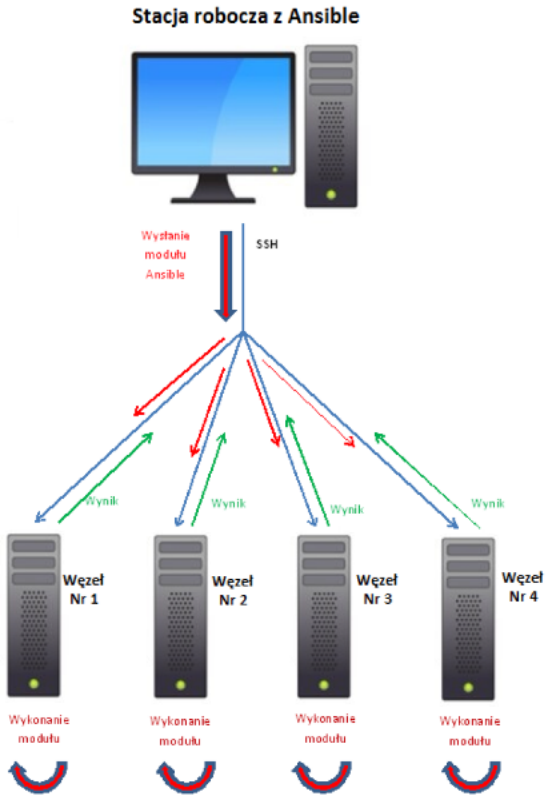


Diagram 3.1. Model komunikacji w Ansible

1. Stacja robocza z Ansible łaczy się za pomocą protokołu SSH ze zdalnym serwerem, a następnie użytkownik jest autoryzowany za pomocą hasła lub klucza.
2. Ansible wysyła programy zwane modułami (modules) do serwera. Moduły zawierają pożądany stan konfiguracji serwera.
3. Moduły są wykonywane przez interpreter Pythona i zwracają wynik działania do terminala maszyny zarządzającej.
4. Pod koniec wykonywania sesji Ansible, moduły są usuwane z zarządzanych węzłów.

Model komunikacji w Ansible przypomina zarządzanie serwerami dwoma metodami: manualną i zautomatyzowaną (skrypty Bash). Administratorzy systemów, którzy dotychczas korzystali z obu metod, mogą w łatwy sposób wykonywać te same zadania używając Ansible.

3.2 Inventory

Inwentarz (ang. Inventory) jest istotnym komponentem w Ansible. Jest to plik typu INI lub YAML, w którym zdefiniowane są informacje o zarządzanych węzłach, takie jak nazwy hostów, adresy IP, loginy, hasła, porty. Inwentarz pozwala również na przypisywanie hostów do różnych grup. Dla przykładu, można przypisać węzły ze względu na typ świadczonej przez nie usługi:

```
1  [database_servers]
2  dbserver1 10.0.200.1
3  dbserver2 10.0.200.2
4  [web_servers]
5  webserver1 10.0.200.1
6  webserver2 10.0.200.2
```

Załącznik 3.2. Grupowanie hostów w Ansible - przykład 1

Grupowanie hostów jest szczególnie przydatne przy zarządzaniu dużymi infrastrukturami z wysoką liczbą serwerów. Częstą praktyką[13] w rozwoju oprogramowania stosowaną zwłaszcza w korporacjach, jest podzielenie procesu i serwerów na trzy gałęzie użytkowe – do rozwoju, testowania i wdrożenia aplikacji:

1. INT (integration) – środowisko integracyjne, które używane jest do scalania i testowania wprowadzanych nowych funkcjonalności aplikacji.
2. STG (staging) – środowisko modelowe, które jest wewnętrznie używaną kopią środowiska produkcyjnego. Ten typ środowiska zawiera wolną od błędów wersję aplikacji i konfigurację serwerów oraz serwisów. Ma ono dać gwarancję, że wprowadzane w oprogramowaniu zmiany nie spowodują niestabilności w środowisku produkcyjnym.
3. PRD (production) – środowisko produkcyjne to środowisko wdrożone do wykorzystania przez użytkowników końcowych. Powinno być wysoce stabilne, dlatego testowanie pliku Inwentarza w Ansible umożliwia zastosowanie tego podejścia i przypisanie hostów do jednej z powyższych grup.

Przykład tak pogrupowanych grup serwerów zaimplementowany przy użyciu pliku inwentarza Ansible może wyglądać następująco (Fragment 3.3):

[13] Środowiska programistyczne. https://dltj.org/article/software-development-practice/. Data odczytu 15.03.2020r.

```
[int_servers]
int_server1 10.0.1.101
int_server2 10.0.1.102
[stg_servers]
stg_server1 10.0.2.101
stg_server2 10.0.2.102
[prd_servers]
prd_server1 10.0.3.101
prd_server2 10.0.3.102
```

Załącznik 3.3. Grupowanie serwerów w Ansible - przykład 2

Powyższe podejście umożliwia w wygodny sposób dostarczanie systemów dużej skali, charakteryzujących się wysoką liczbą hostów, które są dodatkowo rozmieszczone w różnych środowiskach. Ewidencjonowane w ten sposób grupy są obsługiwane przez Ansible poprzez wykonywanie komend ad-hoc i playbooków. Te ostatnie stanowią w Ansible język używany do opisu żądanej konfiguracji i zestawu poleceń do wykonania na zarządzanym serwerze.[14]

3.3 Komendy ad-hoc i playbooki

W Ansible istnieją dwa sposoby wykonywania poleceń na zarządzanych węzłach. Pierwszą z nich jest interaktywna metoda ad-hoc. Pozwala ona na wykonywanie pojedynczych komend lub modułów i otrzymanie wyników w ciągu chwili. Dla przykładu, jeśli zaistnieje potrzeba sprawdzenia osiągalności zarządzanych hostów w sieci można wykorzystać moduł `ping`, co przedstawia poniższy przykład (Fragment 3.4):

```
$ ansible int_servers -i hosts -u devops -m ping
int_server1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
int_server2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Fragment 3.4. Sprawdzanie osiągalności hostów w Ansible

Jako efekt wykonania powyższego polecenia, Ansible użyje modułu `ping` do sprawdzenia możliwości logowania poprzez SSH jako użytkownik `devops` na każdym serwerze z grupy

[14] Ansible. Playbooki w Ansible. https://docs.ansible.com/ansible/latest/user_guide/playbooks.html. Data odczytu 26.06.2020 r.

`int_servers`, ewidencjonowanym w pliku `hosts`. Metoda ad-hoc może być używana do szybkiego wprowadzania drobnych zmian lub aplikowaniu łatek (ang. patch) na wielu serwerach jednocześnie. Obecnie istnieje ponad 700 modułów o rozmaitych zastosowaniach, a ich liczba wzrasta z każdą nową wersją Ansible.

Drugim sposobem opisywania w jaki sposób systemy będą skonfigurowane przy użyciu Ansible są playbooki. Są to skrypty pisane w języku YAML[15]. Poniżej zaprezentowano prosty przykład takiego skryptu (Fragment 3.5):

```
1   ---
2   - hosts: int_servers
3     remote_user: grzegorz
4     become: true
5     become_method: sudo
6     become_user: root
7
8     tasks:
9       - name: install nginx server
10        apt: name=nginx state=latest update_cache=yes
11        when: ansible_os_family == "Debian"
12
```

Fragment 3.5. Ansible - instalacja serwera nginx

Po uruchomieniu tego playbooka Ansible nawiąże połączenie SSH jako użytkownik `grzegorz` z każdym hostem z grupy `int_servers`. W następnym kroku zgromadzi informacje o zarządzanych hostach, które w nomenklaturze Ansible istnieją jako tzw. fakty (ang. facts). Następnie zostaną wykonane zadania zdefiniowane w sekcji `tasks` po przelogowaniu się na użytkownika `root`. W powyższym przykładzie Ansible spróbuje zainstalować serwer Nginx na hostach z system operacyjnym `Debian`. Sprawdzi on czy pakiet ten jest obecnie zainstalowany i zwróci odpowiednią informację. Tylko wtedy, gdy Nginx nie jest obecny w systemie, zaktualizuje on repozytoria i dokona instalacji. Ustawienie parametru `state` na wartość `latest` nakazuje Ansible by sprawdzić, czy paczka Nginx jest dostępny w najnowszej wersji w systemie.

Po wykonaniu aktualizacji konfiguracji serwera Nginx, wymaga on restartu - w Ansible można wykorzystać do tego tzw. procedury (ang. handlers)[16]. Odpowiada ona za sprawdzanie stanu

[15] YAML. https://yaml.org. Data odczytu 16.03.2020 r.
[16] Ansible. Procedury w Ansible. https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro#handlers-running-operations-on-change. Data odczytu 16.03.2020 r.

serwisów i zmiany ich stanu na inny. Uruchamiane są one tylko w przypadku, gdy zadanie, które je wywołuje dokonało zmian na serwerze. Procedury są uruchamiane na samym końcu wykonywania poleceń z playbooka.

Przykładowo, w poniższym playbooku (Fragment 3.6) za każdym razem, gdy pakiet Nginx zostanie zainstalowany lub zaktualizowany, procedura `restart_nginx` zostanie uruchomiona na samym końcu wykonywania playbooka. Procedury są wykonywane na samym końcu działania skryptu, ponieważ zbędnym jest kilkukrotne restartowanie jednego serwisu podczas wykonywania tego samego playbooka.

```
1  ---
2  - hosts: servers
3    remote_user: grzegorz
4    become: true
5    become_method: sudo
6    become_user: root
7
8    tasks:
9      - name: ensure nginx is at the latest version
10       apt: name=nginx state=latest update_cache=yes
11       when: ansible_os_family == "Debian"
12       notify:
13         - restart_nginx
14
15   handlers:
16   - name: restart_nginx
17     service: name=nginx state=restarted
18
```

Fragment 3.6. Przykład użycia procedur w Ansible

Ponadto, możliwe jest rozdzielenie wszystkich zadań i procedur do mniejszych playbooków i zaimportowanie ich do głównego playbooka. Ansible wspiera również uruchamianie playbooków w trybie próbnym (ang. dry-run), który sprawdzi jakie zmiany zostaną faktycznie dokonane przy rzeczywistym uruchomieniu tego playbooka.

Dla kontrastu, instalowanie serwera Nginx za pomocą skryptu Bash wygląda następująco (Fragment 3.7):

```
#!/bin/bash

if [[ "$OSTYPE" == "linux-gnu" ]]; then
    if [[ -x "$(command -v nginx)" ]]; then
        echo "*** nginx already installed ***"
        exit 0;
    else
        echo "*** Installing nginx ***"
        sudo apt update 2>&1 >/dev/null
        sudo apt -y install nginx 2>&1 >/dev/null
    fi
fi
```

Fragment 3.7. Instalacja Nginx za pomocą skryptu Bash

Można zauważyć, że kod pisany napisany w YAML jest o wiele łatwiejszy w interpretacji przez człowieka niż ten napisany w Bashu. Zważywszy na ściśle określoną konwencję pisania kodu w YAML, nie jest widoczne, który administrator napisał poszczególny fragment kodu. Skryptów pisanych w Bashu nie ogranicza żadna konwencja czy zasady, a styl ich pisania zazwyczaj różni się pomiędzy administratorami systemów. Stanowi to wyzwanie w przypadku, gdy do zespołu dołączy nowy członek – zaznajomienie się ze skryptem konfiguracyjnym i wprowadzanie w nim zmian zajmie wtedy więcej czasu niż gdyby był to skrypt YAML.

Jedną z najciekawszych opcji w Ansible jest możliwość grupowania playbooków w tzw. role (ang. roles), a następnie implementowanie tych ról w innych projektach.[17] Dla przykładu, można zdefiniować rolę służącą do instalacji serwera Nginx, a następnie wykorzystać ją w playbooku wdrażającym aplikację napisaną we frameworku Flask. Role mogą być udostępnione w społeczności Ansible Galaxy.[18] Można również pobierać role za darmo z dostępnego tam zbioru i wykorzystywać je we własnych playbookach.

3.4 Szablony

Szablony (ang. templates) w Ansible są modułem, który pozwala na definiowanie dynamicznych plików konfiguracyjnych.[19] Szablony te używają specjalnych zmiennych i są definiowane w systemie szablonowania Jinja2.[20]

[17] Ansible. Role w Ansible. http://docs.ansible.com/ansible/playbooks_roles.html. Data odczytu 18.03.2020 r.
[18] Ansible. Ansible Galaxy. https://galaxy.ansible.com. Data odczytu 21.03.2020 r.
[19] Ansible. Szablony Ansible. https://docs.ansible.com/ansible/latest/modules/template_module.html. Data odczytu 21.03.2020 r.
[20] Jinja2. https://jinja.palletsprojects.com/en/master/. Data odczytu 21.03.2020 r.

```
# {{ ansible_managed }}
NameVirtualHost *:80

<VirtualHost *:80>
    ServerName {{ servername }}
    DocumentRoot {{ documentroot }}
    ServerAdmin {{ serveradmin }}
    <Directory "{{ documentroot }}">
        AllowOverride All
        Options -Indexes FollowSymLinks
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

Załącznik 3.8. Przykład szablonu w Ansible

Załącznik 3.8 przedstawia szablon do automatycznego konfigurowania wirtualnych hostów w serwerze webowym Apache2. Pozwala on na uruchomienie kilku aplikacji jako osobnych serwisów, używając do tego jednej domeny i adresu IP. Zmienne w szablonie, które są zdefiniowane w podwójnych nawiasach klamrowych, konwertowane są do faktycznych wartości podczas wykonywania playbooka. Ich wartości mogą zostać ustawione bezpośrednio w playbooku lub w osobnych plikach. Pierwsza linia szablonu `# {{ ansible managed }}` jest używana jako instrukcja dla Ansible, że ma do czynienia z szablonem, a nie zwyczajnym plikiem. Po zdefiniowaniu szablonu, można go umieścić na zarządzanych serwerach za pomocą poniższego zadania (Fragment 3.9):

```
- name: Add virtual hosts config
  template: src=files/{{ item }}/default dest={{ item }}
  /{{ project }} mode=0644
  with_items:
  - /etc/apache2/sites-available
  notify:
  - apache2_restart
```

Fragment 3.9. Użycie szablonów w Ansible

Powyższe zadanie prześle plik konfiguracyjny ze stacji roboczej, biorąc jako źródło szablon ze ścieżki `files/etc/apache2/sites-available`, wyśle go na zarządzane serwery do ścieżki `/etc/apache2/sites-available/app_name`, gdzie wartość `app_name` pochodzi ze zmiennej `{{ project }}`. Dla przesłanego pliku konfiguracyjnego zostaną ustawione również uprawnienia 644. Jeśli szablon na danym zdalnym hoście już istnieje, a jego zawartość jest identyczna jak w szablonie, wtedy polecenie nie zostanie wykonane. Jeśli

szablon istnieje, ale z innymi parametrami, zostanie on nadpisany. Dodatkowo procedura `apache2_restart` zostanie wykonana na samym końcu, restartując serwer Apache2
Tego typu zadania są często wykonywane przy pracy z Ansible.

3.5 Podsumowanie

Ansible jest dobrze wyważonym narzędziem do zarządzania konfiguracją i infrastrukturą, gdzie wymagana jest wysoka dostępność serwerów. Używany przez niego plik inwentarza pozwala na łatwe dodawanie, usuwanie i edytowanie informacji o serwerach. Dodatkowo, Ansible wspiera wieloplatformowość, a jego moduły mogą być wykonywane w wielu różnych systemach operacyjnych. Ansible posiada zdolność identyfikacji różnych parametrów systemu operacyjnego poprzez zbieranie faktów o hostach, którymi zarządza. Ponadto, playbooki są łatwe do odczytu, i nawet użytkownicy nieznający tego narzędzia są w stanie zrozumieć zastosowanie danego playbooka. Dlatego też Ansible jest dobrym narzędziem do poznania i zrozumienia zasad zarządzania konfiguracją. Playbooki napisane pod konkretne zadanie mogą być łatwo użyte w przyszłości bez potrzeby tworzenia nowych od zera dla podobnie konfigurowanych serwerów. Ansible wspiera również tryb testowy, który jest przydatny do testowania nowo zaimplementowanych playbooków i sprawdzania, jakie zmiany w konfiguracji serwerów zostały wprowadzone.

Wadą Ansible jest brak natywnego wsparcia dla systemów kontroli wersji, co utrudnia śledzenie zmian, gdy playbook jest współtworzony przez kilka osób. Dodatkowo, możliwa jest sytuacja, w której wersja danego playbooka różni się pomiędzy stacjami roboczymi – wykonanie go może prowadzić do nieścisłości w konfiguracji na zarządzanych serwerach. W dużych firmach problem ten jest rozwiązany poprzez przechowywanie projektów tworzonych w Ansible w repozytorium, a zmiany w konfiguracji wprowadzane są używając metody scalania kodu zwanej pull request. Przy zastosowaniu tego podejścia, dobrą praktyką jest nieumieszczanie wrażliwych danych (loginy, hasła, klucze) w systemie kontroli wersji, a przechowywanie ich w osobnych plikach. W efekcie, używanie Ansible w połączeniu z systemami kontroli wersji zaciera różnice między tworzeniem kodu, a administracją systemami, gdyż tworzenie aplikacji końcowej i projektów w Ansible wymagają podobnych działań.

4 IMPLEMENTACJA ZARZĄDZANIA KONFIGURACJĄ

4.1 Cel

W tym rozdziale zostanie zaprezentowany system automatyzujący procesy zarządzania konfiguracją, aktualizowania oprogramowania i raportowania urządzeń w sieci IoT, składającej się z jednostek RSRU oraz podłączonych do nich czujników w projekcie Smart Oil Plug (SOP). RSRU (ang. Remote Sensing Research Unit) jest to wyspecjalizowany do zbierania i procesowania danych komputer jednopłytkowy. W projekcie SOP wykorzystywana jest płytka Atomic Pi zbudowana w 64-bitowej architekturze i wyposażona w czterordzeniowy procesor Intel Atom x5-Z8350 o bazowym taktowaniu 1.44 GHz i 2GB pamięci RAM.
Urządzenia RSRU wyposażone są w system operacyjny Linux, w jednej z dwóch wersji: Debian lub Clear Linux. Celem przechowywania danych, na płytce drukowanej urządzeń RSRU montowana jest pamięć eMMC o pojemności 20GB oraz karta pamięci SHDC o pojemności 5GB. Urządzenia RSRU wyposażone są również w gniazdo na kartę SIM.
Implementacja systemu automatyzującego zarządzanie projektem (dalej nazywana SOP_Updater) została oparta o platformę do zarządzania konfiguracją Ansible.
Zważywszy na niestabilne środowisko pracy jednostek RSRU cała komunikacja odbywa się bezprzewodowo, przy użyciu sieci Bluetooth i WiFi. Model komunikacji między komponentami sieci w projekcie SOP przedstawia poniższy schemat (Diagram 4.1):

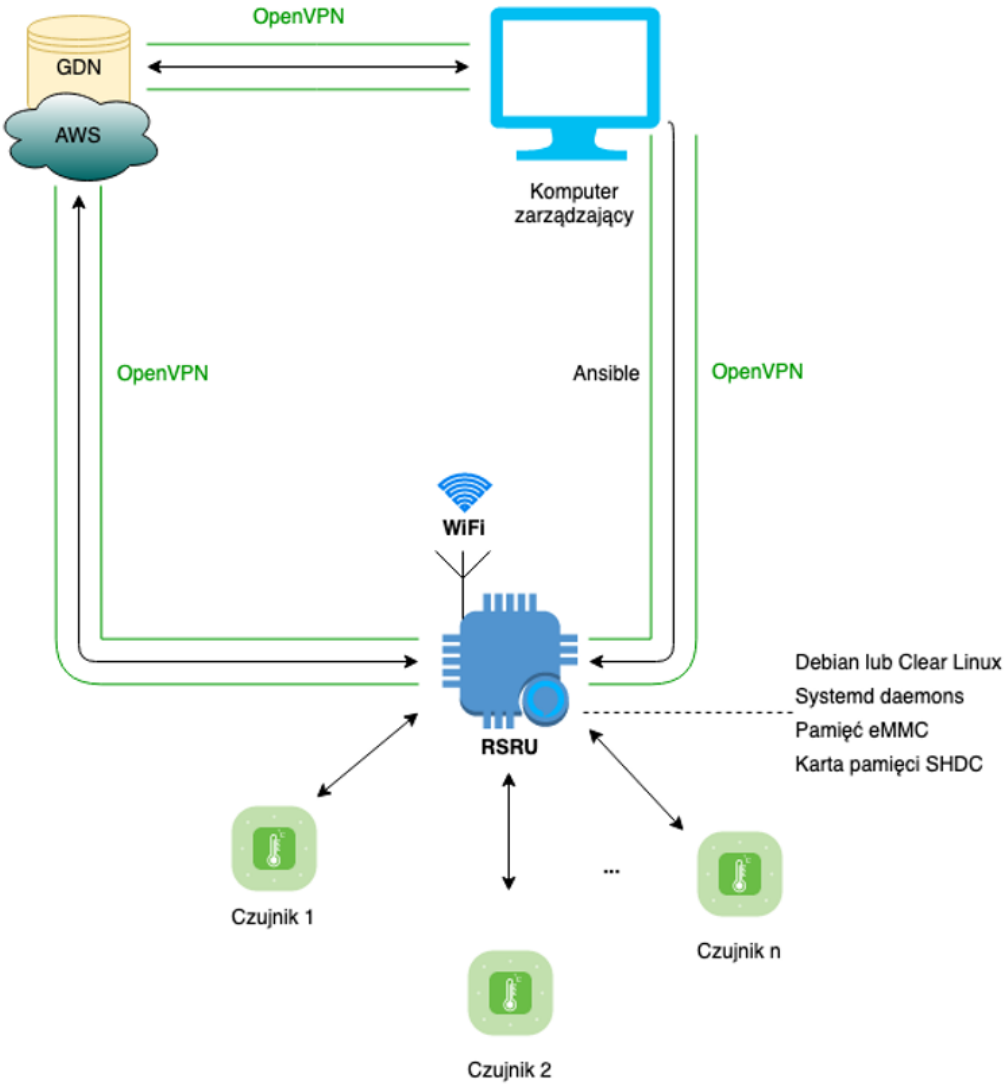


Diagram 4.1. Schemat komunikacji w sieci SOP

Każde urządzenie RSRU instalowane w miejscu docelowym jest wyposażone w antenę patykową dookólną, która rozgłasza sieć Wifi używając do tego pasma o częstotliwości 5 GHz. Za pomocą tej sieci do urządzenia RSRU podłączone są urządzenia peryferyjne (tzw. Plugs). Sa to mikroczujniki zamontowane np. w misie olejowej przekładni skrzyni biegów czy tensometrze mierzącym naprężenie metalowych elementów. Ich celem jest monitorowanie i gwarancja prawidłowych parametrów pracy różnych komponentów pociągów, okrętów lub schodów ruchomych w zależności od miejsca ich montażu.

Część danych z pracy RSRU i czujników jest przechowywana w bazie danych Microsoft SQL Server, nazywanej roboczo GDN. Jest ona hostowana w chmurze obliczeniowej AWS. RSRU nawiązuje połączenie z bazą danych GDN celem zapisu zebranych danych. Część modułów

Ansible wymaga wcześniejszego pobrania danych z bazy GDN. Na ich podstawie obliczane i ustawiane są parametry dla tych modułów.

Komunikacja między RSRU, a komputerem zarządzającym i bazą danych GDN odbywa się przy wykorzystaniu sieci GSM.

Moduły i polecenia Ansible są wysyłane z komputera zarządzającego poprzez tunel OpenVPN. Podobnie dzieje się w przypadku komunikacji między jednostkami RSRU, a bazą GDN.

4.2 Struktura projektu SOP_Updater

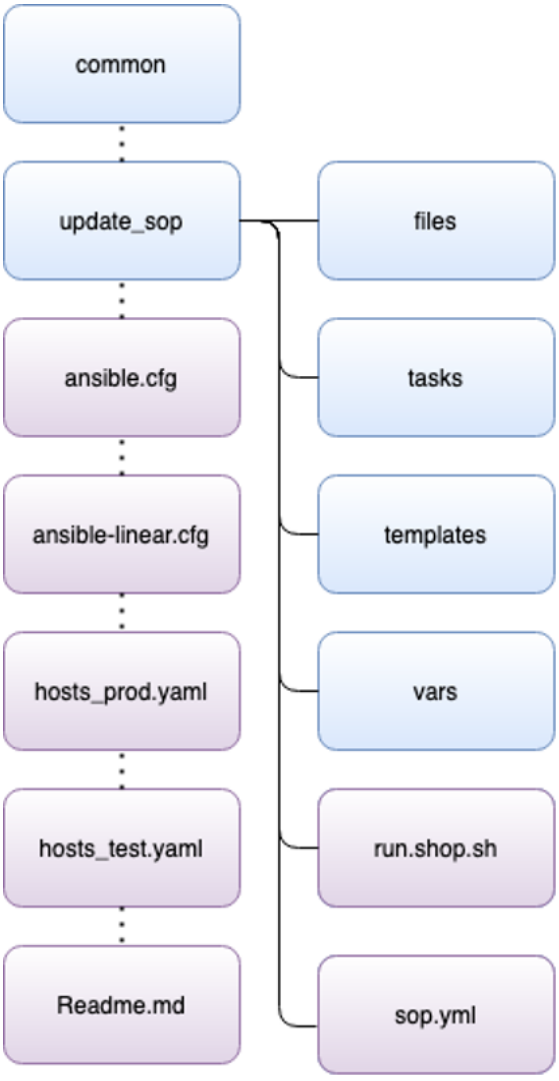


Diagram 4.2. Struktura projektu SOP_Updater

Powyżej (Diagram 4.2) zaprezentowano pełną strukturę systemu Sop_Updater Do uruchomienia projektu został przygotowany skrypt `run_sop.sh` działający w systemach

macOS i Linux. Obsługuje on 5 trybów działania systemu SOP_Updater, które są szerzej opisane w rozdziale 4.1.3.

Głównym playbookiem systemu SOP_Updater jest `sop.yml`. Z niego w sposób proceduralny uruchamiane są inne playbooki z katalogu `tasks` wykorzystując do tego celu moduł Ansible `include_tasks`. Zaimplementowane zostało sterowanie uruchamiania trybów systemu za pomocą klauzuli `when`. Dla przykładu (Fragment 4.3), playbook `sop_config_only.yml` zostanie uruchomiony, tylko w przypadku gdy trybem działania systemu będzie `config-only`.

```
59      - name: Perform peripheral config update only
60        include_tasks: tasks/sop_config_only.yml
61        when: mode == "config-only"
```

Fragment 4.3. SOP w trybie config-only

Do uruchomienia projektu SOP_Updater konieczne jest wskazanie pliku inwentarza Ansible. W przypadku SOP_Updater jest to plik `hosts_prod.yaml` przechowywany w głównym katalogu projektu SOP_Updater. Przechowywana jest w nim lista zarządzanych urządzeń RSRU, umieszczona w grupie hostów sop_prod. Każdy host ma przypisaną również listę słowników o nazwie `peripherals`, w której umieszczone są zmienne przechowujące konfigurację czujników. Są to identyfikatory (`id`), adresy fizyczne MAC (`addr`), numery seryjne (`sn`), flagi dla skryptu dfu (`dfu`) oraz konfiguracja dla bazy mongodb (`cfg`). W poniższej (Fragment 4.4) części pliku inwentarza zaprezentowano przykładową konfigurację dla jednego z urządzeń RSRU.

```
6       hosts:
7         10.8.6.121:
8           peripherals:
9             - { id: e45ca22ac70a, addr: "E4:5C:A2:2A:C7:0A", sn: 194101, dfu: { run: yes,
              new_hdw: no }, cfg: { zThreshold: 30, yThreshold: 200, xThreshold: 200,
              accAcqDivider: 0, measSamples: 32000, checkSamples: 3200, advertInterval: 300 } }
10            - { id: de33a8a01d98, addr: "DE:33:A8:A0:1D:98", sn: 194102, dfu: { run: yes,
              new_hdw: no }, cfg: { zThreshold: 30, yThreshold: 200, xThreshold: 200,
              accAcqDivider: 0, measSamples: 32000, checkSamples: 3200, advertInterval: 300 } }
```

Fragment 4.4. Konfiguracja parametrów RSRU w inwentarzu

4.3 Tryby działania i sposób uruchomienia

Projekt SOP_Updater rozpoczyna swoje działanie od uruchomienia pliku `run_sop.sh`, który jest dostępny w głównym katalogu SOP_Updater. Zaimplementowane zostało pięć trybów działania projektu SOP:

- `full-update` – Tryb ten przygotowuje i wykonuje pełną aktualizację firmware'u urządzeń RSRU.

- `config-only` – Tryb ten ogranicza działanie do wykonania aktualizacji konfiguracji urządzeń RSRU przechowywanej w bazie NoSQL mongoDB.

- `deploy` – Program uruchomiony w tym trybie umieści sam siebie na zdalnym serwerze, zrobi konfigurację i uruchomi potrzebne mu daemony systemd.

- `report` – Tryb ten wykorzystywany jest do pobierania danych i metryk z pracy systemów. Zebrane dane mają swoją reprezentację w pliku JSON, a ich wartości przypisywane są z w trybie report mail.

- `report-mail` – Używany do wygenerowania raportu końcowego z szablonu Jinja2. Wszystkie zmienne urządzeń RSRU czytane są z plików JSON w pętli. Następnie raport jest wysyłany do odbiorców docelowych celem dalszej analizy i weryfikacji pracy urządzeń RSRU.

Aby uruchomić system SOP_Updater należy w terminalu komputera zarządzającego wykonać polecenie według poniżej zaprezentowanego schematu:

```
$ bash SOP_Updater/update_sop/run_sop.sh <Tryb> <Grupa hostów>
```

Na przykład:

```
$ bash SOP_Updater/run_sop.sh full-update sop_prod
```

4.4 Logika działania

W tym rozdziale zostaną zaprezentowane kluczowe moduły, logika działania oraz ich rola w projekcie SOP_Updater.

4.4.1 Zbieranie danych

Projekt SOP uruchomiony w trybie `report` ma na celu weryfikację stanu urządzeń RSRU i wygenerowanie raportu końcowego. Proces weryfikacji rozpoczyna się wraz z uruchomieniem playbooka `sop_report.yml.`

Weryfikacji zostają poddane następujące komponenty i ich parametry:

- Daemon app_transceiver – Daemon zarządzający główną aplikacją dla urządzeń RSRU, która składa się z kilku skryptów.
- Daemon mongodb – Baza danych NoSQL
- Zużycie pamięci na karcie SD
- Zużycie pamięci wbudowanej eMMC
- Dostępność interfejsu webowego aplikacji Transceiver umożliwiającą podgląd parametrów pracy RSRU

Każdy komponent (oprócz interfejsu www) zostaje weryfikowany za pomocą dedykowanemu mu playbookowi, dołączanym za pomocą modułu `include_tasks`. Poniżej (Fragment 4.5) zaprezentowano fragmentu kodu, który uruchamia playbooka `report_daemons.yml` celem sprawdzenia stanu serwisów app_transceiver i mongodb.

update_sop > tasks > ! sop_report.yml

```yaml
- name: Check services
  include_tasks: report_daemons.yml
  loop:
  - app_transceiver
  - mongodb
  loop_control:
    loop_var: service_name
```

Fragment 4.5. Sprawdzanie serwisów w RSRU

Część danych pochodzi z globalnej bazy `GDN`, dostępnej przez cały czas działania projektu. Są to dane prezentujące sumarycznej ilości plików z danymi i logami w ciągu 7 ostatnich dni i timestamp ostatniego połączenia urządzenia RSRU z globalną bazą. Do ich pozyskania używany jest playbook `report_sql.yml`

Parametry dotyczące pracy czujników takie jak wskaźnik mocy odbieranego sygnału (RSSI) są pozyskiwane z tej samej bazy za pomocą playbooka `report_sql_peripheral.yml`

Po zebraniu wszystkich danych są one przechowywane w plikach wynikowych `.json`. Zapis tych danych jest inwokowany przez poniższy (Fragment 4.6) blok kodu playbooka `sop_report.yml`.

```yaml
- name: Save json data
  copy: content="{{rsru|to_nice_json}}" dest=json/{{inventory_hostname}}.json
  delegate_to: localhost
```

Fragment 4.6. Zapis zebranych danych z RSRU do JSON

4.4.2 Obróbka zebranych danych

Do obliczania danych związanych z czasem pobierana jest data za pomocą funkcji `date` w przypadku daemonów lub funkcją `NOW(),` w przypadku bazy danych GDN jak pokazano w poniższym bloku kodu (Fragment 4.7) Potem wykonywana jest konwersja pobranej daty do czasu w formacie Epoch, otrzymując w ten sposób reprezentację w sekundach. Następnie analogiczne kroki są wykonywane dla pobranej z bazy danych np. daty ostatniej aktywności urządzenia peryferyjnego. Następnym krokiem jest porównanie obu wartości, w wyniku którego otrzymywana jest różnica czasów w sekundach. Oprócz tego ustawiane są statusy wraz z odpowiednim kolorem na podstawie zdefiniowanych progów, które zostaną użyte przy generowaniu raportu końcowego.

```yaml
- name: Get location age
  local_action: command mysql {{db_conn}} "SELECT (UNIX_TIMESTAMP(NOW()) - tstamp) FROM
  logs_locations WHERE device = '{{dev.cr_id}}' ORDER BY id DESC LIMIT 1;"
  register: result
- name: Parse location age result
  set_fact: age_secs="{{result.stdout|int}}"
- name: Format time difference
  include_tasks: tasks/time_units.yml
  vars:
    t_in_secs: "{{age_secs}}"
- name: Create location age info
  set_fact:
    loc_age_info:
      name: "Last location age"
      value: "{{t_out_str}}"
      color: "{{'green' if age_secs|int <= 86400 else 'orange' if age_secs|int <= 3*86400 else
      'red'}}"
```

Fragment 4.7. Przykład zbierania danych z MySQL

Na końcu wykonywana jest konwersja otrzymanej różnicy do czasu w minutach, godzinach lub dniach. Wykorzystywany do tego jest dedykowany playbook `time_units.yml,` który na wejściu akceptuje zmienną `t_in_secs`, dokonuje konwersji czasu w sekundach, a następnie zwraca wynik do zmiennej `t_out` (Fragment 4.8)

update_sop > tasks > ! time_units.yml

```yaml
- set_fact:
    t_out: { num: "{{ ((t_in_secs | int) / 60) | round | int }}", unit: "minutes" }
  when: t_in_secs|int < 3600

- set_fact:
    t_out: { num: "{{ ((t_in_secs | int) / 3600) | round | int }}", unit: "hours" }
  when: t_in_secs|int > 3600 and t_in_secs|int <= 86400

- set_fact:
    t_out: { num: "{{ ((t_in_secs | int) / 86400) | round | int }}", unit: "days" }
  when: t_in_secs|int > 86400

- set_fact:
    t_out_str: "{{ t_out.num }} {{ t_out.unit }}"
```

Fragment 4.8. Konwersja czasu w playbooku time_units.yml

4.4.3 Generowanie raportu

Po zebraniu i opracowaniu wszystkich danych, generowany jest raport na podstawie wcześniej przygotowanego statycznego szablonu Jinja2 dostępnego w ścieżce `update_sop/templates/report.j2`

Celem zastosowania zaawansowanego formatowania, został on wbudowany w strukturę HTML. Zastosowanie szablonowania Jinja2 ogranicza ilość kodu koniecznego do generowania pozycji w raporcie dla kolejnych urządzeń RSRU – wystarczy blok z jedną pętlą `for` oraz instrukcje warunkowe `if`. Na tej podstawie generowane są wpisy w raporcie dla kolejnych urządzeń RSRU w sposób rekursywny. Proces generowania tych danych prezentuje poniższy fragment szablonu (Fragment 4.9)

```
{% for rsru in rsrus %}
{% for peripheral in rsru.peripherals %}
{% set info_style = 'class="bg-success"' %}
{% if "red" in peripheral.parameters|map(attribute="color") %}
{% set info_style = 'class="bg-danger"' %}
{% elif "orange" in peripheral.parameters|map(attribute="color") %}
{% set info_style = 'class="bg-warning"' %}
{% endif %}
```

Fragment 4.9. Generowanie wpisów dla raportu

4.4.4 Cykliczne wykonywanie raportu

Projekt SOP wymaga, aby moduł `report` był wykonywany cyklicznie i w pełni samoistnie raz w tygodniu. Aby to osiągnąć, na węźle zarządzającym umieszczane i konfigurowane są dwa serwisy systemd: [21]

[21] Serwisy systemd. https://www.freedesktop.org/software/systemd/man/systemd.service.html

- `ansible_generate_report.service` – serwis uruchamiający skrypty generujące raport
- `ansible_send_mail.service` – serwis odpowiadający za wysyłanie wygenerowanego raportu mailem

Oba serwisy posiadają również `timer`, który zarządza czasem wykonywania serwisów `systemd`. Dla przykładu, poniższa definicja timera serwisu (Fragment 4.10) `ansible_generate_report.timer` zleci wykonanie serwisu `ansible_generate_report.service` w każdy piątek o godz. 8:00:

```
1  [Unit]
2  Description= Timer for creating SOP report every friday
3
4  [Timer]
5  OnCalendar=Fri 08:00:00
6  Unit=ansible_generate_report.service
7
8  [Install]
9  WantedBy=multi-user.target
```

Fragment 4.10. Cykliczne generowanie raportu SOP

4.4.5 Aktualizacja firmware

Tryb działania `full-update` umożliwia automatyczną aktualizację oprogramowania firmware na urządzeniu RSRU. Do tego zadania, został przygotowany playbook `sop_full_update.yml`, który dzieli ten proces na pięć poniższych zadań:

- Zatrzymanie serwisów systemd (task o nazwie `Prepare services`)
- Przygotowanie niezbędnych skryptów i plików instalacyjnych (playbook transfer.yml)
- Aktualizacja oprogramowania firmware RSRU (playbook firmware.yml)
- Aktualizacja konfiguracji w bazie danych urządzenia RSRU (playbook mongo.yml)
- Restartowanie daemonów RSRU (task Restart services)

Każdy z tych kroków wykonywany jest jako osobno zdefiniowane zadanie w playbooku `sop_full_update.yml`. Zadania odpowiedzialne za transfer plików, instalacji nowego firmware i wykonania aktualizacji konfiguracji w bazie danych umieszczone są w osobnych playbookach i uruchamiane za pomocą funkcji `include_tasks`. Definicje tych zadań w kodzie Ansible przedstawia poniższy blok kodu (Fragment 4.11):

```yaml
---
- name: Prepare services
  systemd:
    name: "{{ item.name }}"
    state: "{{ item.state }}"
    enabled: yes
  with_items:
    - { name: 'app_transceiver', state: 'stopped' }
    - { name: 'watchdog_kick', state: 'restarted' }

- name: Transfer files
  include_tasks: transfer.yml

- name: Run device firmware update
  include_tasks: firmware.yml

- name: Update mongodb
  include_tasks: mongo.yml

- name: Restart services
  systemd:
    name: "{{ item }}"
    state: restarted
  with_items:
    - 'mongodb'
    - 'io_broker'
    - 'app_transceiver'
    - 'www_run'
```

Fragment 4.11. Aktualizacja firmware RSRU

4.4.6 Aktualizacja konfiguracji

Projekt uruchomiony w trybie `config-only` wykonuje aktualizacje parametrów czujników w bazie mongo bez aktualizacji oprogramowania RSRU. Zmienne te można konfigurować w pliku inwentarza `hosts_prod.yaml`, a następnie użyć przypisane im wartości odwołując się do zmiennej `peripherals` jak przedstawiono w załączniku [4.12].

update_sop > tasks > ! sop_config_only.yml

```yaml
---
- include_tasks: mongo_peripheral.yml
  vars:
    peripheral: "{{item}}"
  with_items: "{{ hostvars[inventory_hostname].peripherals }}"
```

Załącznik 4.12. Przygotowanie nowej konfiguracji mongo

4.4.7 Automatyczne wykrywanie nowych urządzeń

Jednym z wymagań projektu SOP jest automatyczne wykrywanie nowych urządzeń w sieci, bez wykonywania dodatkowych czynności umożliwiających działanie Ansible. Aby to umożliwić, w głównym pliku konfiguracyjnym `ansible.cfg` zostało wyłączone sprawdzanie autentyczności hostów za pomocą kluczy (Fragment 4.13). Nie powoduje to luki w bezpieczeństwie, gdyż cały ruch sieciowy odbywa się poprzez tunel OpenVPN.

ansible-linear.cfg

```
1  [defaults]
2  strategy = linear
3  forks = 30
4  host_key_checking = False # disable host authenticity checking
```

Fragment 4.13. Fragment konfiguracji ansible.cfg

Następnie w każdym nowoodkrytym hoście zostaje umieszczona za pomocą modułu `authorized_key` kopia klucza publicznego SSH, pobierana z katalogu domowego użytkownika z którego uruchamiany jest playbook (Fragment 4.14).

```
38      - name: Set authorized key for root
39        authorized_key:
40          user: root
41          state: present
42          key: "{{ lookup('file', lookup('env','HOME') + '/.ssh/id_rsa.pub') }}"
```

Fragment 4.14. Kopiowanie klucza publicznego ssh na serwer

4.4.8 Praca z niestabilnym połączeniem sieciowym

Częstym scenariuszem w SOP jest, że dane urządzenie RSRU przebywa w strefie bez możliwości połączenia z siecią GSM przez długi czas, rzędu kilku godzin. W takim przypadku Ansible zakończy swoje działanie zwracając wyjątek UNREACHABLE i dla tego hosta nie będą ponawiane próby połączenia i wykonywanie kolejnych modułów Ansible. W celu rozwiązania tego problemu na samym początku playbooka wyłączono zbieranie faktów o hoście oraz ustawiono timeout w wysokości 10800 sekund. Dla pozostałych trybów działania projektu wartości timeoutu są niższe, gdyż podczas ich wykonywania nie następuje połączenie z urządzeniami RSRU (Fragment 4.15):

update_sop > ! sop.yml

```yaml
---
- hosts: sop_prod
  gather_facts: no

  tasks:
    - include_vars:
        dir: vars

    # Concatenate all files and send report
    - name: Send SOP report mail
      include_tasks: tasks/report_mail.yml
      when: mode == "report-mail"
    - meta: end_play
      when: mode == "report-mail"

    - name: Create SQL report (offline)
      include_tasks: ./tasks/report_sql.yml
      vars:
        state: "offline"
      when: mode == "report"

    # Test SQL only - stop processing here
    # - fail: msg="OK"

    - name: device is available
      wait_for_connection:
        connect_timeout: 30
        timeout: "{{ item.secs }}"
      when: "item.when"
      with_items:
      - { secs: 3600, when: "{{ mode == 'report' }}" }
      - { secs: 60, when: "{{ mode == 'report-mail' }}" }
      - { secs: 10800, when: "{{ mode != 'report' and mode != 'report-mail' }}" }

    - name: Gathering facts
```

Fragment 4.15. Obsługa niestabilnych hostów

Ansible posiada kilka strategii wykonywania tasków: `debug`, `free`, host_`pinned`, `linear`.[22] Domyślnie działa używając strategii `linear`, tzn. dopiero po wykonaniu danego taska na wszystkich hostach, przystąpi do wykonania następnego. Ze względu na niestabilne środowisko w jakim pracują urządzenia RSRU, konieczna była zmiana strategii wykonywania playbooka w konfiguracji `ansible.cfg` na strategię `free` (Fragment 4.16):

ansible.cfg

```
[defaults]
strategy = free
```

Fragment 4.16. Strategia free w Ansible

[22] Ansible. Strategie w Ansible. https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html

Dzięki temu cały playbook zostanie wykonany dla każdego hosta niezależnie, a niedostępne akurat hosty nie będą spowalniać pracy Ansible na pozostałych.

4.5 Wynik działania

W wyniku działania trybu `report` zwracany jest raport w formacie HTML dzielący się na dwie części. Pierwsza część w nagłówku (Fragment 4.17) raportu wyświetla nazwę urządzenia RSRU (kolumna RSRU) oraz wszystkie podłączone do niego czujniki (kolumna Plug). Dodatkowo przekazywany jest ogólny stan czujnika w kolumnie State, ułatwiając odbiorcy raportu identyfikację i interpretację ogólnego stanu urządzenia RSRU.

Smart-Oil Plug Report

RSRU	Plug	State
Northern Trial 4	f93325ab606f	
Northern Trial 4	c05c7c2d2cd6	
Northern Trial 3	e45ca22ac70a	
Northern Trial 3	de33a8a01d98	
Northern Trial 10	f4b8108cf15b	
Northern Trial 10	f03cebec73e5	
Northern Trial 5	c23f8fd7c421	
Northern Trial 5	dd3db5cec1f0	
Northern Trial 9	cc0a878c5848	
Northern Trial 9	dbd12e4957bb	
Northern Trial 12	ff21ac3807d6	
Northern Trial 12	dafa7c7179eb	
Northern Trial 8	e4ad02938dde	
Northern Trial 8	c7fb1a612d33	
Northern Trial 6	eeb4cc7b94e7	
Northern Trial 6	d49943f2c88b	
Northern Trial 11	eb251790082c	
Northern Trial 11	c52206a99f36	
Northern Trial 13	e8196e8336b3	
Northern Trial 13	c6304850d90d	

Fragment 4.17. Nagłówek raportu SOP

Dla każdego urządzenia RSRU generowana jest sekcja ze szczegółowymi danymi dotyczącymi pracy urządzenia RSRU. W poniższej sekcji raportu (Fragment 4.18) zaprezentowano dane dla urządzenia RSRU o nazwie Northern Trial 4 do którego podpięte są dwa czujniki. Dla urządzenia RSRU zawarte są dane dotyczące pracy daemonów systemd oraz suma wygenerowanych plików i zapisanych w pamięci eMMC. W drugiej i następujących po niej polach Plug przedstawione są szczegółowe dane z pracy czujników. W razie przekroczenia

dopuszczalnych norm pozycja w raporcie jest oznaczana kolorem czerwonym, informując odbiorcę raportu o konieczności reakcji na ten incydent.

Northern Trial 4

RSRU | 10.8.6.101 | CR-0afef7			
State:	offline		
Last location age:	43 minutes		
Last location age:	1 days		
Total files (7 days):	61		

Plug | f93325ab606f | 194103 | 03 Northern - GB E63440976			
Total files (7 days):	0		
Sample age:	55 days		
Capacitance [pF]:	2.687		
Battery voltage [V]:	3.44		
Temperature [°C]:	14.4		
RSSI [dBm]:	-80		
Packet counter:	2		
Firmware version:	3.08		

Plug | c05c7c2d2cd6 | 194104 | 04 Northern - GB 3441186			
Total files (7 days):	61		
Sample age:	1 days		
Capacitance [pF]:	5.598	x̄: 5.551	[5.512, 5.598]
Battery voltage [V]:	3.77	x̄: 3.77	[3.74, 3.80]
Temperature [°C]:	17.3	x̄: 13.5	[10.4, 17.3]
RSSI [dBm]:	-64	x̄: -66	[-85, -56]
Packet counter:	3		
Firmware version:	3.098		

Fragment 4.18. Wpis RSRU w raporcie SOP

5 PODSUMOWANIE

Praca ta przedstawiła korzyści, jakie płyną ze stosowania narzędzi do zarządzania konfiguracją w pracy administratorów systemów i inżynierów DevOps. Zostały omówione najpopularniejsze rozwiązania w tym segmencie – Chef, Puppet, SaltStack oraz Ansible. Stanowią one dobrą alternatywę do ręcznego instalowania serwisów i dokonywania zmian w ich konfiguracji oraz pisanych do tego skryptów Bash. Dużą zaletą wynikającą z użytkowania tych narzędzi jest standaryzacja konfiguracji serwerów w grupie serwerów. Zapobiega to pojawieniu się prędzej czy później nieścisłości w konfiguracji i potrzebie czasochłonnej analizy oraz prac naprawczych.

Najszerzej zbadane zostało narzędzie do zarządzania konfiguracją Ansible, a jego szerokie możliwości odzwierciedlają przykłady pojedynczych zadań przedstawionych w trzecim rozdziale oraz udokumentowana w czwartym rozdziale implementacja systemu do zarządzania siecią IoT urządzeń badawczych RSRU.
Rosnące praktykowanie paradygmatów DevOps oraz IaC w inżynierii systemów pozwala na jednoczesne zarządzanie setkoma lub tysiącami serwerów przez jedną osobę.
Jest to zdecydowanie kierunek, w którym będzie nadal podążać obszar inżynierii IT jakim jest zarządzanie konfiguracją.

6 BIBLIOGRAFIA

1. L. Hochstein, R. Moser, Ansible w praktyce. Automatyzacja konfiguracji i proste instalowanie systemów. 2018. Wydawnictwo Helion S.A.
2. Daniel Hall. Ansible Configuration Management, Second Edition . 2015. Wydawnictwo Packt Publishing.
3. https://www.usenix.org/system/files/login/articles/105457-Lueninghoener.pdf
4. https://www.admin-magazine.com/Archive/2014/23/Choosing-between-the-leading-open-source-configuration-managers
5. https://docs.ansible.com/
6. https://docs.saltstack.com/en/latest/contents.html
7. https://docs.chef.io/
8. https://puppet.com/docs/puppet/5.5/puppet_index.html
9. https://dev.mysql.com/doc/workbench/en/
10. https://dev.mysql.com/doc/refman/8.0/en/
11. https://docs.mongodb.com/manual/tutorial/query-documents/
12. https://jinja.palletsprojects.com/en/2.11.x/templates/
13. https://docs.python.org/3.8/

7 SPIS ZAŁĄCZNIKÓW