

O'REILLY®

Wydanie II



Ansible w praktyce

AUTOMATYZACJA KONFIGURACJI
I PROSTE INSTALOWANIE SYSTEMÓW

Helion 

Lorin Hochstein, René Moser

Tytuł oryginału: Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way, 2nd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-4171-5

© 2018 Helion S.A.

Authorized Polish translation of the English edition of Ansible: Up and Running, 2E
ISBN 9781491979808 © 2017 Lorin Hochstein and René Moser.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/ansipr>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	11
Wstęp do drugiego wydania	13
Wstęp do pierwszego wydania	15
1. Wprowadzenie	19
Uwaga do wersji	20
Do czego nadaje się Ansible?	20
Jak działa Ansible?	21
Na czym polega wielkość Ansible?	22
Czy Ansible nie jest zbyt proste?	26
Co muszę wiedzieć?	26
Czego tu nie znajdziesz?	27
Instalacja Ansible	27
Konfiguracja serwera testowego	28
Co dalej?	35
2. Scenariusze: pierwsze kroki	37
Wstępne wymagania	37
Bardzo prosty scenariusz	38
Uruchomienie scenariusza	41
Scenariusz to plik YAML	42
Anatomia scenariusza	45
Czy coś się zmieniło? Śledzenie stanu serwera	49
Coś ciekawszego: szyfrowanie TLS	49
3. Ewidencja: opisywanie serwerów	57
Plik ewidencyjny	57
Wstępne wymagania: kilka maszyn Vagrant	58
Funkcjonalne parametry ewidencji	60
Grupy, grupy i jeszcze raz grupy	62

Zmienne serwerowe i grupowe w pliku ewidencyjnym	66
Zmienne serwerowe i grupowe w osobnych plikach	67
Dynamiczna ewidencja	69
Podział ewidencji na kilka plików	73
Dodawanie wpisów w trakcie działania scenariusza za pomocą modułów <code>add_host</code> i <code>group_by</code>	74
4. Zmienne i fakty	77
Definiowanie zmiennych w scenariuszu	77
Wyświetlanie wartości zmiennych	78
Rejestrowanie zmiennych	78
Fakty	81
Definiowanie nowej zmiennej za pomocą modułu <code>set_fact</code>	84
Wbudowane zmienne	85
Definiowanie zmiennych w wierszu poleceń	87
Priorytety	88
5. Mezzanine: nasza testowa aplikacja	89
Dlaczego wdrażanie aplikacji produkcyjnych jest skomplikowane?	89
6. Instalacja Mezzanine za pomocą Ansible	95
Wyświetlanie zadań scenariusza	95
Układ zainstalowanych plików	96
Zmienne jawne i poufne	96
Instalowanie wielu pakietów za pomocą pętli (<code>with_items</code>)	98
Instrukcja <code>become</code> w zadaniu	99
Aktualizacja rejestru <code>apt</code>	99
Sprawdzenie projektu za pomocą modułu <code>git</code>	100
Instalacja Mezzanine i innych pakietów w środowisku wirtualnym	102
Krótką dygresja: skomplikowane argumenty w zadaniach	104
Konfiguracja bazy danych	106
Tworzenie pliku <code>local_settings.py</code> na podstawie szablonu	107
Polecenia <code>django-manage</code>	110
Uruchamianie własnych skryptów Pythona w kontekście aplikacji	111
Utworzenie plików konfiguracyjnych usług	113
Aktywacja konfiguracji serwera <code>Nginx</code>	115
Instalacja certyfikatów <code>TLS</code>	115
Instalacja zadania <code>Twitter</code> w harmonogramie <code>cron</code>	116
Cały scenariusz	117
Uruchomienie scenariusza na maszynie wirtualnej <code>Vagrant</code>	120
Diagnostyka	120
Instalacja Mezzanine na wielu serwerach	121

7. Skalowanie scenariuszy: role	123
Podstawowa struktura roli	123
Przykład: role database i mezzanine	124
Stosowanie ról w scenariuszach	124
Zadania wstępne i końcowe	125
Rola database instalująca bazę danych	126
Rola mezzanine instalująca aplikację Mezzanine	128
Tworzenie plików i katalogów ról za pomocą narzędzia ansible-galaxy	132
Role zależne	132
Repozytorium Ansible Galaxy	133
8. Zaawansowane scenariusze	135
Obsługa błędnie działających poleceń: instrukcje changed_when i failed_when	135
Filtry	138
Wyszukiwarki	140
Zaawansowane pętle	146
Sterowanie pętlami	149
Dołączanie plików	151
Bloki	153
Obsługa błędów za pomocą bloków	154
Szyfrowanie poufnych danych	156
9. Dostosowywanie serwerów, przebiegów i procedur	159
Wzorce specyfikowania serwerów	159
Określanie grupy serwerów	160
Wykonywanie zadania na komputerze sterującym	160
Wykonywanie zadania na innym komputerze niż serwer	161
Wykonywanie zadania na kolejnych serwerach	161
Wykonywanie zadania w grupie serwerów	162
Jednokrotne wykonanie zadania	163
Strategie przebiegów	163
Zaawansowane procedury	166
Jawne gromadzenie faktów	173
Odczytywanie adresu IP serwera	174
10. Wtyczki zwrotne	175
Wtyczki stdout	175
Inne wtyczki	178

11. Przyspieszanie Ansible	185
Zwielokrotnienie sesji SSH (opcja ControlPersist)	185
Potokowanie	188
Zapamiętywanie faktów	189
Równoległe połączenia	192
Równoległe wykonywanie zadań za pomocą instrukcji async	193
12. Własne moduły	195
Przykład: sprawdzenie, czy zewnętrzny serwer jest dostępny	195
Użycie modułu script zamiast tworzenia własnego modułu	195
Skrypt can_reach jako moduł	196
Gdzie umieszczać własne moduły?	196
Jak Ansible uruchamia moduły?	196
Oczekiwane wyniki	198
Tworzenie modułów w języku Python	199
Dokumentowanie modułu	209
Diagnozowanie modułu	210
Implementowanie modułu jako skryptu Bash	211
Określanie alternatywnego położenia powłoki Bash	212
Przykładowe moduły	213
13. Vagrant	215
Przydatne opcje konfiguracyjne środowiska Vagrant	215
Prowizjoner Ansible	217
Kiedy jest uruchamiany prowizjoner?	217
Plik ewidencyjny tworzony przez środowisko Vagrant	217
Równoległe prowizjonowanie maszyn	218
Definiowanie grup maszyn wirtualnych	219
Lokalny prowizjoner Ansible	220
14. Amazon EC2	221
Terminologia	222
Poświadczenia	224
Warunek: instancja biblioteki Python Boto	225
Dynamiczna ewidencja	225
Definiowanie dynamicznych grup instancji na podstawie tagów	228
Chmury EC2-VPC i EC2-Classic	230
Przygotowanie pliku ansible.cfg do korzystania z chmury EC2	231
Uruchamianie nowych instancji	231
Pary kluczy EC2	233
Grupy bezpieczeństwa	234

Uzyskiwanie najnowszego obrazu AMI	236
Dodanie nowej instancji do grupy	237
Oczekiwanie na uruchomienie instancji	239
Idempotentne tworzenie instancji	240
Wszystko razem	241
Definiowanie chmury VPC	242
Tworzenie obrazów AMI	246
Inne moduły	250
15. Docker	253
Przykład użycia programów Docker i Ansible	254
Czas życia aplikacji Docker	254
Przykładowa aplikacja: Ghost	255
Nawiązywanie połączenia z demonem Dockera	255
Uruchomienie kontenera na lokalnym komputerze	256
Tworzenie obrazu za pomocą pliku Dockerfile	257
Orkiestracja kilku kontenerów na lokalnym komputerze	258
Wysyłanie obrazów do rejestru Dockera	259
Odczytywanie informacji o lokalnych obrazach	260
Instalowanie aplikacji zawartych w obrazach	262
Ansible Container	266
16. Diagnostowanie scenariuszy	273
Czytelne komunikaty o błędach	273
Diagnostowanie połączenia SSH	274
Moduł debug	275
Debugger scenariuszy	275
Moduł assert	277
Sprawdzenie scenariusza przed uruchomieniem	279
Wybieranie zadań do wykonania	280
17. Zarządzanie serwerami Windows	283
Połączenie z systemem Windows	283
PowerShell	284
Moduły Windows	286
Nasz pierwszy scenariusz	286
Aktualizacja systemu Windows	287
Tworzenie lokalnych kont użytkowników	288
Podsumowanie	291

18. Ansible i urządzenia sieciowe	293
Obecny stan modułów sieciowych	293
Lista obsługiwanych urządzeń	294
Przygotowanie urządzenia sieciowego	294
Jak funkcjonują moduły?	297
Pierwszy scenariusz	297
Ewidencja i zmienne w modułach sieciowych	298
Korzystanie z plików konfiguracyjnych	302
Szablony, szablony, szablony	305
Gromadzenie faktów	306
Podsumowanie	307
19. Ansible Tower: wersja dla firm	309
Modele subskrypcji	309
Jakie problemy rozwiązuje Ansible Tower?	311
Interfejs REST API	316
Interfejs CLI	316
Co dalej?	320
A. Protokół SSH	321
Natywny klient SSH	321
Agent SSH	321
Uruchomienie agenta SSH	322
Przekazywanie agenta	323
Klucze hosta	326
B. Role IAM i poświadczenia EC2	329
Konsola AWS Management Console	329
Wiersz poleceń	330
Słowniczek	333
Bibliografia	339
Skorowidz	341

Skalowanie scenariuszy: role

Jedną z moich ulubionych cech oprogramowania Ansible jest jego skalowalność w górę i w dół. Nie mam tu na myśli liczby zarządzanych serwerów, tylko złożoność automatyzowanych operacji.

Ansible dobrze skaluje się w dół, ponieważ łatwo implementuje się w nim proste zadania. Dobrze skaluje się w górę, ponieważ oferuje mechanizmy dzielenia skomplikowanych zadań na mniejsze części.

W terminologii Ansible **rola** oznacza podstawowy mechanizm dzielenia scenariusza na kilka plików. Dzięki niemu tworzenie skomplikowanych scenariuszy jest prostsze, a poszczególne pliki można wykorzystywać w różnych scenariuszach. Rolę można wyobrazić sobie jako cechę, którą można przypisać jednemu lub wielu serwerom. Na przykład rolę `database` można przypisać serwerom obsługującym bazy danych.

Podstawowa struktura roli

Rola ma nazwę, na przykład `database`. Informacje o roli zapisywane są w katalogu `roles/database` zawierającym następujące podkatalogi i pliki:

`roles/database/tasks/main.yml`

Plik z zadaniami.

`roles/database/files/`

Katalog z plikami, które będą przesyłane do serwerów.

`roles/database/templates/`

Katalog z szablonami Jinja2.

`roles/database/handlers/main.yml`

Plik z procedurami.

`roles/database/vars/main.yml`

Plik ze zmiennymi, których nie można nadpisywać.

roles/database/defaults/main.yml

Plik z domyślnymi zmiennymi, które można nadpisywać.

roles/database/meta/main.yml

Plik z informacjami o zależnościach roli.

Wszystkie pliki są opcjonalne. Jeżeli rola nie zawiera procedur, wówczas nie trzeba tworzyć pustego pliku *handlers/main.yml*.

Gdzie Ansible szuka plików ról?

Ansible szuka plików ról w podkatalogu *roles* znajdującym się w tym samym katalogu co scenariusz. Ponadto przeszukuje również wspólny katalog */etc/ansible/roles*. Katalog ten można zmienić, modyfikując odpowiednio ustawienie *roles_path* w sekcji *defaults* w pliku *ansible.cfg*, jak pokazuje listing 7.1.

Listing 7.1. Plik ansible.cfg zmieniający domyślny katalog ról

```
[defaults]
roles_path = ~/ansible_roles
```

Katalog można również zmienić, modyfikując zmienną środowiskową *ANSIBLE_ROLES_PATH*.

Przykład: role database i mezzanine

Wróćmy do scenariusza instalującego aplikację Mezzanine i zaimplementujmy w nim role. Moglibyśmy utworzyć jedną rolę o nazwie *mezzanine*, ale do zainstalowania bazy danych PostgreSQL wykorzystamy osobną rolę o nazwie *database*. Dzięki temu łatwiej będzie zainstalować bazę na innym serwerze niż aplikacja Mezzanine.

Stosowanie ról w scenariuszach

Zanim zajmiemy się szczegółami definiowania roli, musisz dowiedzieć się, jak w scenariuszu przypisuje się role do serwerów. Listing 7.2 przedstawia nasz scenariusz instalujący aplikację Mezzanine na pojedynczym serwerze, wykorzystujący rolę *mezzanine*.

Listing 7.2. Plik mezzanine-single-host.yml

```
- name: Instalacja Mezzanine na maszynie Vagrant
  hosts: web
  vars_files:
    - secrets.yml
  roles:
    - role: database
      database_name: "{{ mezzanine_proj_name }}"
      database_user: "{{ mezzanine_proj_name }}"
    - role: mezzanine
      live_hostname: 192.168.33.10.xip.io
  domains:
    - 192.168.33.10.xip.io
    - www.192.168.33.10.xip.io
```

Jeżeli w scenariuszu będą wykorzystywane role, trzeba w nim zdefiniować sekcję `roles`. W sekcji tej znajduje się lista ról. W naszym przykładzie lista zawiera dwie role: `database` i `mezzanine`.

Zwróć uwagę, że w rolach można wykorzystywać zmienne. W tym przypadku są to zmienne `database_name` i `database_user` w roli `database`. Gdy zmienne o takich nazwach zostaną zdefiniowane wcześniej (na przykład w pliku `vars/main.yml` lub `defaults/main.yml`), wtedy zostaną one nadpisane zmiennymi zdefiniowanymi w roli.

Jeżeli w rolach nie są wykorzystywane zmienne, to nazwy ról można zdefiniować jak niżej:

```
roles:
  - database
  - mezzanine
```

Po zdefiniowaniu ról `database` i `mezzanine` utworzenie scenariusza instalującego aplikację WWW i bazę danych na osobnych serwerach jest znacznie prostsze. Listing 7.3 przedstawia scenariusz instalujący bazę danych na serwerze o nazwie `db`, a aplikację WWW na serwerze `web`. Zwróć uwagę, że scenariusz składa się z dwóch osobnych akcji.

Listing 7.3. Plik `mezzanine-across-hosts.yml`

```
- name: Instalacja PostgreSQL na maszynie Vagrant
  hosts: db
  vars_files:
    - secrets.yml
  roles:
    - role: database
      database_name: "{{ mezzanine_proj_name }}"
      database_user: "{{ mezzanine_proj_name }}"
- name: Instalacja Mezzanine na maszynie Vagrant
  hosts: web
  vars_files:
    - secrets.yml
  roles:
    - role: mezzanine
      database_host: "{{ hostvars.db.ansible_eth1.ipv4.address }}"
      live_hostname: 192.168.33.10.xip.io
      domains:
        - 192.168.33.10.xip.io
        - www.192.168.33.10.xip.io
```

Zadania wstępne i końcowe

Czasami pojawia się potrzeba wykonania pewnych zadań przed wywołaniem roli lub po jej wywołaniu. Załóżmy, że przed zainstalowaniem aplikacji Mezzanine trzeba zaktualizować rejestr `apt`, a po instalacji wysłać powiadomienie do kanału Slack.

W Ansible, w sekcji `pre_tasks`, można zdefiniować listę zadań do wykonania przed wywołaniem roli, a w sekcji `post_tasks` — listę zadań do wykonania po wywołaniu roli. Listing 7.4 przedstawia zastosowanie takich zadań.

Listing 7.4. Stosowanie sekcji `pre-tasks` i `post-tasks`

```
- name: Instalacja Mezzanine na maszynie Vagrant
  hosts: web
  vars_files:
    - secrets.yml
```

```

pre_tasks:
- name: Aktualizacja rejestru apt
  apt: update_cache=yes
roles:
- role: mezzanine
  database_host: "{{ hostvars.db.ansible_eth1.ipv4.address }}"
  live_hostname: 192.168.33.10.xip.io
  domains:
    - 192.168.33.10.xip.io
    - www.192.168.33.10.xip.io
post_tasks:
- name: Wysłanie powiadomienia do Slack o zaktualizowaniu serwerów
  local_action: >
    slack
    domain=acme.slack.com
    token={{ slack_token }}
    msg="Server WWW {{ inventory_hostname }} skonfigurowany"

```

Na razie dość o stosowaniu ról, zajmijmy się ich tworzeniem.

Rola database instalująca bazę danych

Celem roli database jest zainstalowanie oprogramowania PostgreSQL, utworzenie bazy danych i zdefiniowanie konta jej użytkownika.

Rola jest zdefiniowana za pomocą następujących plików:

- *roles/database/tasks/main.yml*,
- *roles/database/defaults/main.yml*,
- *roles/database/handlers/main.yml*,
- *roles/database/files/pg_hba.conf*,
- *roles/database/files/postgresql.conf*.

Rola obejmuje dwa dostosowane pliki konfiguracyjne oprogramowania PostgreSQL:

postgresql.conf

Plik zawierający zmienioną opcję `listen_addresses` (adresy nasłuchujące), dzięki której z bazą będzie można łączyć się za pomocą dowolnego interfejsu sieciowego. Domyślnie baza PostgreSQL dopuszcza nawiązywanie połączeń tylko z lokalnego komputera, co w naszym przypadku jest nie do przyjęcia, ponieważ baza będzie znajdowała się na innym serwerze niż aplikacja WWW.

pg_hba.conf

Plik z informacjami o uwierzytelnianiu połączeń z bazą za pomocą loginu i hasła.



Wyżej wymienione pliki nie są tu zaprezentowane, ponieważ są bardzo duże. Znajdziesz je w załączonych do książki plikach w katalogu *r08*.

Listing 7.5 przedstawia zadania instalujące bazę PostgreSQL.

Listing 7.5. Plik roles/database/tasks/main.yml

```
- name: Instalacja pakietów apt
  apt: pkg={{ item }} update_cache=yes cache_valid_time=3600
  become: True
  with_items:
    - libpq-dev
    - postgresql
    - python-psycopg2
- name: Kopiowanie pliku konfiguracyjnego
  copy: >
    src=postgresql.conf dest=/etc/postgresql/9.3/main/postgresql.conf
    owner=postgres group=postgres mode=0644
  become: True
  notify: Restart bazy PostgreSQL
- name: Kopiowanie pliku konfiguracji uwierzytelniania klienta
  copy: >
    src=pg_hba.conf dest=/etc/postgresql/9.3/main/pg_hba.conf
    owner=postgres group=postgres mode=0640
  become: True
  notify: Restart bazy PostgreSQL
- name: Ustawienia regionalne
  locale_gen: name={{ locale }}
  become: True
- name: Zdefiniowanie użytkownika
  postgresql_user:
    name: "{{ database_user }}"
    password: "{{ db_pass }}"
  become: True
  become_user: postgres
- name: Utworzenie bazy danych
  postgresql_db:
    name: "{{ database_name }}"
    owner: "{{ database_user }}"
    encoding: UTF8
    lc_ctype: "{{ locale }}"
    lc_collate: "{{ locale }}"
    template: template0
  become: True
  become_user: postgres
```

Listing 7.6 przedstawia plik procedur.

Listing 7.6. Plik roles/database/handlers/main.yml

```
- name: Restart bazy PostgreSQL
  service: name=postgresql state=restarted
  become: True
```

Jedyna domyślna zmienna, którą zdefiniujemy, będzie zawierała numer portu TCP wykorzystywanego przez bazę danych (patrz listing 7.7).

Listing 7.7. Plik roles/database/defaults/main.yml

```
database_port: 5432
```

Zwróć uwagę, że w zadaniach wykorzystywane są następujące zmienne, których nigdzie nie zdefiniowaliśmy:

- database_name (nazwa bazy danych),
- database_user (użytkownik bazy danych),

- `db_pass` (hasło bazy danych),
- `locale` (ustawienia regionalne).

W rolach zdefiniowanych w listingach 7.2 i 7.3 wykorzystane są zmienne `database_name` i `database_user`. Przyjąłem założenie, że zmienna `db_pass` jest zdefiniowana w pliku `secrets.yml` wskazanym w sekcji `vars_files`. Zmienna `locale` będzie prawdopodobnie zawierała taką samą wartość dla każdego serwera, a ponadto będzie można ją wykorzystywać w różnych rolach i scenariuszach, dlatego zdefiniowałem ją w załączonym do książki pliku `group_vars/all`.

Dlaczego zmienne w rolach definiuje się na dwa sposoby?

Gdy w Ansible zostały wprowadzone role, wykorzystywane w nich zmienne można było definiować tylko w jednym miejscu, tj. w pliku `vars/main.yml`. Takie zmienne mają wyższy priorytet niż zmienne zdefiniowane w sekcji `vars` akcji. Oznacza to, że nie można nadpisać tych zmiennych, chyba że zostaną jawnie użyte jako argumenty roli.

W kolejnych wersjach Ansible domyślne zmienne roli można definiować w pliku `defaults/main.yml`. Zmienne te mają niższy priorytet, można więc je nadpisywać innymi zdefiniowanymi w scenariuszu zmiennymi o takich samych nazwach.

Jeżeli przewidujesz, że wartość zmiennej w roli będzie modyfikowana, stosuj domyślną zmienną. W przeciwnym razie użyj zwykłej zmiennej.

Rola mezzanine instalująca aplikację Mezzanine

Przeznaczeniem roli `mezzanine` będzie instalacja aplikacji Mezzanine, odwrotnego serwera proxy Nginx i monitora procesów Supervisor. Rola ta składa się z następujących plików:

- `roles/mezzanine/defaults/main.yml`,
- `roles/mezzanine/handlers/main.yml`,
- `roles/mezzanine/tasks/django.yml`,
- `roles/mezzanine/tasks/main.yml`,
- `roles/mezzanine/tasks/nginx.yml`,
- `roles/mezzanine/templates/gunicorn.conf.py.j2`,
- `roles/mezzanine/templates/local_settings.py.filters.j2`,
- `roles/mezzanine/templates/local_settings.py.j2`,
- `roles/mezzanine/templates/nginx.conf.j2`,
- `roles/mezzanine/templates/supervisor.conf.j2`,
- `roles/mezzanine/vars/main.yml`.

Listing 7.8 przedstawia zdefiniowane w tej roli zmienne. Zwróć uwagę, że nazwy wszystkich zmiennych rozpoczynają się od `mezzanine`. Jest to dobra praktyka nazywania zmiennych, ponieważ w Ansible nie można definiować przestrzeni nazw dla ról. Oznacza to, że zmienne zdefiniowane

w różnych rolach lub w dowolnym miejscu scenariusza są dostępne w całym scenariuszu. Jeżeli przypadkowo użyjesz w różnych rolach zmiennej o takiej samej nazwie, możesz uzyskać nieprzewidziane efekty.

Listing 7.8. Plik `roles/mezzanine/vars/main.yml`

```
# Plik zmiennych w roli mezzanine.
mezzanine_user: "{{ ansible_user }}"
mezzanine_venv_home: "{{ ansible_env.HOME }}"
mezzanine_venv_path: "{{ mezzanine_venv_home }}/{{ mezzanine_proj_name }}"
mezzanine_repo_url: git@github.com:lorin/mezzanine-example.git
mezzanine_proj_dirname: project
mezzanine_proj_path: "{{ mezzanine_venv_path }}/{{ mezzanine_proj_dirname }}"
mezzanine_reqs_path: requirements.txt
mezzanine_conf_path: /etc/nginx/conf
mezzanine_python: "{{ mezzanine_venv_path }}/bin/python"
mezzanine_manage: "{{ mezzanine_python }} {{ mezzanine_proj_path }}/manage.py"
mezzanine_gunicorn_port: 8000
```

Listing 7.9 przedstawia domyślną zmienną zdefiniowaną w roli mezzanine. W tym przypadku jest to tylko jedna zmienna. Definiując domyślne zmienne, nie stosuję specjalnych prefiksów w nazwach, ponieważ w innych miejscach scenariusza zamierzam nadpisywać te zmienne.

Listing 7.9. Plik `roles/mezzanine/defaults/main.yml`

```
tls_enabled: True
```

Ponieważ opisywane zadanie jest dość długie, podzieliłem je na kilka plików. Listing 7.10 przedstawia najważniejsze zadanie w roli mezzanine, instalujące pakiety apt. W zadaniu tym za pomocą instrukcji `include` dołączane są dwa inne pliki zadań, przedstawione w listingach 7.11 i 7.12, zapisane w tym samym katalogu co główne zadanie.

Listing 7.10. Plik `roles/mezzanine/tasks/main.yml`

```
- name: Instalacja pakietów apt
  apt: pkg={{ item }} update_cache=yes cache_valid_time=3600
  become: True
  with_items:
    - git
    - libjpeg-dev
    - libpq-dev
    - memcached
    - nginx
    - python-dev
    - python-pip
    - python-psycopg2
    - python-setuptools
    - python-virtualenv
    - supervisor
    - include: django.yml
    - include: nginx.yml
```

Listing 7.11. Plik `roles/mezzanine/tasks/django.yml`

```
- name: Utworzenie katalogu logs
  file: path="{{ ansible_env.HOME }}/logs" state=directory
- name: Sprawdzenie repozytorium na serwerze
  git:
    repo: "{{ mezzanine_repo_url }}"
```

```

    dest: "{{ mezzanine_proj_path }}"
    accept_hostkey: yes
- name: Globalna instalacja pakietów Pythona za pomocą pip
  pip: name={{ item }} state=latest
  with_items:
    - pip
    - virtualenv
    - virtualenvwrapper
- name: Instalacja wymaganych pakietów Pythona
  pip: name={{ item }} virtualenv={{ mezzanine_venv_path }}
  with_items:
    - gunicorn
    - setproctitle
    - psycopy2
    - django-compressor
    - python-memcached
- name: Instalacja pakietów z pliku requirements.txt
  pip: >
    requirements={{ mezzanine_proj_path }}/{{ mezzanine_reqs_path }}
    virtualenv={{ mezzanine_venv_path }}
- name: Utworzenie pliku ustawień
  template: src=local_settings.py.j2 dest={{ mezzanine_proj_path }}/local_settings.py
- name: Migracja w celu utworzenia bazy danych, zebranie statycznej treści
  django_manage:
    command: "{{ item }}"
    app_path: "{{ mezzanine_proj_path }}"
    virtualenv: "{{ mezzanine_venv_path }}"
  with_items:
    - migrate
    - collectstatic
- name: Nadanie identyfikatora strony
  script: scripts/setsite.py
  environment:
    PATH: "{{ mezzanine_venv_path }}/bin"
    PROJECT_DIR: "{{ mezzanine_proj_path }}"
    PROJECT_APP: "{{ mezzanine_proj_app }}"
    WEBSITE_DOMAIN: "{{ live_hostname }}"
- name: Zdefiniowanie hasła administratora
  script: scripts/setadmin.py
  environment:
    PATH: "{{ mezzanine_venv_path }}/bin"
    PROJECT_DIR: "{{ mezzanine_proj_path }}"
    PROJECT_APP: "{{ mezzanine_proj_app }}"
    ADMIN_PASSWORD: "{{ admin_pass }}"
- name: Utworzenie pliku konfiguracyjnego Gunicorn
  template: src=gunicorn.conf.py.j2 dest={{ mezzanine_proj_path }}/gunicorn.conf.py
- name: Utworzenie pliku konfiguracyjnego Supervisor
  template: src=supervisor.conf.j2 dest=/etc/supervisor/conf.d/mezzanine.conf
  become: True
  notify: Restart menedżera Supervisor
- name: Sprawdzenie, czy istnieje ścieżka konfiguracyjna
  file: path={{ mezzanine_conf_path }} state=directory
  become: True
  when: tls_enabled
- name: Instalacja zadania cron odczytującego Twitter
  cron: >
    name="poll twitter" minute="*/5" user={{ mezzanine_user }}
    job="{{ mezzanine_manage }} poll_twitter"

```


Listing 7.12. Plik `roles/mezzanine/tasks/nginx.yml`

```
- name: Utworzenie pliku konfiguracyjnego Nginx
  template: src=nginx.conf.j2 dest=/etc/nginx/sites-available/mezzanine.conf
  notify: Restart serwera Nginx
  become: True
- name: Aktywacja pliku konfiguracyjnego Nginx
  file:
    src: /etc/nginx/sites-available/mezzanine.conf
    dest: /etc/nginx/sites-enabled/mezzanine.conf
    state: link
  notify: Restart serwera Nginx
  become: True
- name: Usunięcie domyślnego pliku konfiguracyjnego Nginx
  file: path=/etc/nginx/sites-enabled/default state=absent
  notify: Restart serwera Nginx
  become: True
- name: Utworzenie certyfikatów TLS
  command: >
    openssl req -new -x509 -nodes -out {{ mezzanine_proj_name }}.crt
    -keyout {{ mezzanine_proj_name }}.key -subj '/CN={{ domains[0] }}' -days 3650
    chdir={{ mezzanine_conf_path }}
    creates={{ mezzanine_conf_path }}/{{ mezzanine_proj_name }}.crt
  become: True
  when: tls_enabled
  notify: Restart serwera Nginx
```

Pomiędzy zadaniami zdefiniowanymi w roli a zadaniami zdefiniowanymi w głównym scenariuszu jest jedna istotna różnica, ujawniająca się podczas korzystania z modułów `copy` i `template`.

Przed wywołaniem modułu `copy` w zadaniu zdefiniowanym w roli oprogramowanie Ansible sprawdza, czy w katalogu `rolename/files` znajduje się plik do skopiowania. Analogicznie przed wywołaniem modułu `template` w zadaniu zdefiniowanym w roli oprogramowanie Ansible sprawdza, czy w katalogu `rolename/templates` znajduje się plik szablonu, który ma być użyty. Oznacza to, że poniższe zadanie zdefiniowane w głównym scenariuszu:

```
- name: Utworzenie pliku konfiguracyjnego Nginx
  template: src=templates/nginx.conf.j2 \
  dest=/etc/nginx/sites-available/mezzanine.conf
```

po przeniesieniu do roli musi mieć następującą postać (zwróć uwagę na parametr `src`):

```
- name: Utworzenie pliku konfiguracyjnego Nginx
  template: src=nginx.conf.j2 dest=/etc/nginx/sites-available/mezzanine.conf
  notify: Restart serwera Nginx
```

Listing 7.13 przedstawia zawartość pliku procedur.

Listing 7.13. Plik `roles/mezzanine/handlers/main.yml`

```
- name: Restart menedżera Supervisor
  supervisorctl: name=gunicorn_mezzanine state=restarted
  become: True
- name: Restart serwera Nginx
  service: name=nginx state=restarted
  become: True
```

Nie prezentuję tu plików szablonów, ponieważ są one bardzo podobne do plików przedstawionych w poprzednim rozdziale (niektóre zmienne mają inne nazwy). Szczegóły znajdziesz w załączonych do książki plikach.

Tworzenie plików i katalogów ról za pomocą narzędzia `ansible-galaxy`

Razem z oprogramowaniem Ansible dostarczane jest narzędzie, o którym jeszcze nie pisaliśmy — *ansible-galaxy*. Służy ono przede wszystkim do pobierania z internetu ról zdefiniowanych przez społeczność użytkowników Ansible (więcej na ten temat piszemy w dalszej części rozdziału). Narzędzia tego można również używać do tworzenia **szkieletu** roli, czyli początkowego zestawu plików i katalogów. W tym celu wpisz następujące polecenie:

```
$ ansible-galaxy init -p playbooks/roles web
```

Argument `-p` służy do wskazania katalogu z rolami. Jeżeli zostanie pominięty, pliki zostaną utworzone w bieżącym katalogu.

Wymienione powyżej polecenie tworzy następujące pliki i katalogi:

```
playbooks
  roles
  web
  README.md
  defaults
    main.yml
  files
  handlers
    main.yml
  meta
    main.yml
  tasks
    main.yml
  templates
  tests
    inventory
    test.yml
  vars
    main.yml
```

Role zależne

Wyobraźmy sobie, że mamy dwie role, `web` i `database`, i obie wymagają, aby na serwerze był skonfigurowany protokół NTP¹. W obu rolach można zawrzeć konfigurację tego protokołu, ale w ten sposób powieli się operacje. Zamiast tego można utworzyć osobną rolę `ntp`, ale trzeba będzie wtedy pamiętać, aby oprócz roli `web` lub `database` stosować również powyższą rolę. Ten sposób pozwoli wprawdzie uniknąć duplikacji operacji, ale będzie podatny na błędy, ponieważ łatwo będzie można zapomnieć o zastosowaniu nowej roli. Potrzebne jest rozwiązanie, w którym rola `ntp` będzie zawsze stosowana razem z rolą `web` lub `database`.

W Ansible dostępne są tzw. **role zależne**, umożliwiające osiągnięcie powyższego celu. W definicji roli można wskazać jedną lub kilka innych ról. Role zależne są stosowane w pierwszej kolejności.

¹ Skrót *NTP* (ang. *Network Time Protocol* — protokół czasu sieciowego) oznacza protokół synchronizowania zegarów komputerów.

Wróćmy do naszego przykładu. Założmy, że trzeba utworzyć rolę `ntp` konfigurującą synchronizację zegara serwera za pomocą protokołu NTP. Rola zależna może mieć parametry, zatem założmy, że w tym przypadku takim parametrem będzie adres serwera NTP.

Zależność roli `web` od roli `ntp` skonfigurujemy w ten sposób, że utworzymy plik `roles/web/meta/main.yml` zawierający nazwę roli `ntp` i jej parametr, jak w listingu 7.14.

Listing 7.14. Plik `roles/web/meta/main.yml`

```
dependencies:
- { role: ntp, ntp_server=ntp.ubuntu.com }
```

Można definiować wiele ról zależnych. Gdybyśmy na przykład mieli rolę `django` konfigurującą serwer WWW Django, zależną od ról `nginx` i `memcached`, to powyższy plik wyglądałby jak w listingu 7.15.

Listing 7.15. Plik `roles/web/meta/main.yml`

```
dependencies:
- { role: web }
- { role: memcached }
```

Szczegółowe informacje o przetwarzaniu zależności pomiędzy rolami znajdziesz w oficjalnej dokumentacji do Ansible (<http://bit.ly/1F6tH9a>).

Repozytorium Ansible Galaxy

Jeżeli będziesz musiał zainstalować na serwerze jakieś otwarte oprogramowanie, to będzie duże prawdopodobieństwo, że inny użytkownik Ansible utworzył już odpowiednią rolę. Choć dzięki Ansible tworzenie skryptów instalacyjnych jest prostsze, to jednak wdrażanie niektórych systemów jest skomplikowane.

Gdy będziesz chciał wykorzystać napisaną przez innego użytkownika rolę lub sprawdzić, jak inni poradzi sobie z problemem, nad którym pracujesz, pomocne może być *Ansible Galaxy*. Jest to otwarte repozytorium ról opracowanych przez społeczność użytkowników Ansible. Role zapisane są w serwisie GitHub.

Interfejs WWW

Role dostępne są na stronie Ansible Galaxy (galaxy.ansible.com). Można je przeglądać według autora lub kategorii, dostępna jest również funkcja wyszukiwania.

Interfejs wiersza poleceń

Za pomocą polecenia `ansible-galaxy` można pobierać role z repozytorium Ansible Galaxy.

Instalacja roli

Założmy, że chcesz zainstalować zapisaną w serwisie GitHub rolę `ntp` utworzoną przez użytkownika *bennojoy*. Rola ta służy do konfigurowania synchronizacji zegara serwera za pomocą protokołu NTP.

Zainstaluj tę rolę przy użyciu poniższego polecenia:

```
$ ansible-galaxy install -p ./roles bennojoy.ntp
```

Domyślnie polecenie `ansible-galaxy` instaluje rolę w ogólnodostępnym katalogu (patrz ramka „Gdzie Ansible szuka plików ról?” wcześniej w tym rozdziale). W powyższym poleceniu za pomocą parametru `-p` wskazany jest inny katalog.

Wynik polecenia jest następujący:

```
downloading role 'ntp', owned by bennojoy
no version specified, installing master
- downloading role from https://github.com/bennojoy/ntp/archive/master.tar.gz
- extracting bennojoy.ntp to ./roles/bennojoy.ntp
write_galaxy_install_info!
bennojoy.ntp was installed successfully
```

Polecenie `ansible-galaxy` zapisało pliki roli w katalogu `./roles/bennojoy.ntp`. W pliku `./roles/bennojoy.ntp/meta/galaxy_install_info` zostały zapisane pewne dodatkowe dane dotyczące instalacji.

W moim przypadku plik ten miał następującą zawartość:

```
{install_date: 'Sun Nov 25 15:38:21 2017', version: master}
```



Rola `bennojoy.ntp` nie ma określonego numeru wersji. Jest po prostu oznaczona słowem *master*. Inne role mogą mieć numery wersji, na przykład 1.2.

Wyświetlanie zainstalowanych ról

Zainstalowane role możesz wyświetlić za pomocą następującego polecenia:

```
$ ansible-galaxy list
```

Wynik polecenia wygląda jak niżej:

```
bennojoy.ntp, master
```

Dezinstalacja roli

Rolę usuwa się przy użyciu poniższego polecenia:

```
$ ansible-galaxy remove bennojoy.ntp
```

Udostępnianie własnej roli

Szczegółowe informacje na temat udostępniania roli użytkownikom Ansible znajdziesz na stronie galaxy.ansible.com/intro w sekcji *Create and Share* (tworzenie i udostępnianie). Ponieważ role zapisywane są w serwisie GitHub, musisz posiadać w nim własne konto.

Teraz wiesz już, jak korzystać z ról, tworzyć własne role i pobierać role utworzone przez innych użytkowników. Role są doskonałym sposobem porządkowania scenariusza. Używam ich nieustannie i również Tobie bardzo je polecam.

A

- Active Directory, 288
- adres IP serwera, 174
- adresy IP
 - dozwolone, 235
 - prywatne, 215
- agent SSH, 321
- akcje, 46
- aktualizacja
 - rejestr apt, 99
 - systemu Windows, 287
- aliasy, 65
- Amazon EC2, 221
- AMI, 236
- analiza argumentów, 200
- anatomia scenariusza, 45
- Ansible, 20
- Ansible Tower, 309
- aplikacja
 - Django, 63
 - Ghost, 255
 - Mezzanine, 89
- argument
 - start-at-task, 281
 - step, 281
- argumenty
 - analiza, 200
 - odczytywanie, 201
 - opcja
 - aliases, 202
 - choices, 202
 - default, 202
 - required, 202
 - type, 203
 - w zadaniach, 104
- asercje, 277

B

- baza danych
 - etcd, 146
 - PostgreSQL, 92, 262
- bazy danych
 - konfiguracja, 106
 - rola database, 126
- bezpieczeństwo, 234, 287
- biblioteka Boto, 225
- bloki, 153
- błędy, 154, 273

C

- certyfikat TLS, 50, 115
- chmura, 221
 - EC2, 221
 - biblioteka Boto, 225
 - dynamiczna ewidencja, 225
 - instancja, 223
 - pary kluczy, 233
 - pliki konfiguracyjne, 225
 - role IAM, 329
 - tagi, 223
 - EC2-Classic, 230
 - EC2-VPC, 230
 - IaaS, 221
 - poświadczenia, 224
 - VPC, 242
- ciągi znaków, 43
- cudzysłowy, 51
- czas życia aplikacji, 254

D

- debuger scenariuszy, 275
- definiowanie
 - dynamicznych grup instancji, 228
 - grupy serwerów WWW, 41
 - uprawnień dostępu, 311
 - zmiennych, 77, 84
- demon, 94
- dezinstalacja roli, 134
- diagnostyka, 120
- diagnozowanie
 - modułu, 210
 - połączenia SSH, 274
 - scenariuszy, 273
- DNS, Domain Name System, 144
- Docker, 253
 - czas życia aplikacji, 254
 - połączenie z demonem, 255
 - tworzenie obrazów, 257, 267
 - wysyłanie obrazów do rejestru, 259
- dodawanie instancji, 237
- dokumentacja
 - Ansible, 48
 - modułu, 209
- dołączanie
 - dynamiczne, 152
 - plików, 151
 - ról, 152
- dostęp
 - do adresu, 120
 - do instancji EC2-Classic, 231
- dynamiczna ewidencja, 69, 225
- dynamiczne
 - dołączanie plików, 152
 - grupy instancji, 228
- dzielenie wierszy, 44

E

- EBS, Elastic Block Store, 236
- etykiety, 150
- ewidencja, 57, 298
 - dynamiczna, 69, 225
 - parametry funkcjonalne, 60
 - podział na pliki, 73
 - serwerów, 313

F

- fakt `ansible_eth0`, 174
- fakty, 81, 173
 - lokalne, 83
 - skojarzone z serwerem, 82
 - wyświetlanie podzbioru, 82
 - zapamiętywanie, 189
 - zwracanie, 83
- filtr default, 138
- filtry, 138
 - ścieżek plików, 139
 - własne, 139
 - zarejestrowanych zmiennych, 138
- front aplikacji, 263
- funkcjonalne parametry ewidencji, 60

G

- gniazdo sterujące, 185
- gromadzenie faktów, 306
- grupa, 62
 - bezpieczeństwa, 234
 - porty TCP, 236
- grup, 65
- serwerów WWW, 41
- użytkowników, 288
- Gunicorn, 93

H

- harmonogram cron, 116

I

- IaaS, 221
- idempotentne tworzenie instancji, 240
- implementowanie modułu, 211
- informacje
 - o lokalnych obrazach, 260
 - o serwerze, 31, 70
 - o wykonaniu modułu, 207
- instalacja, 27
 - aplikacji, 262
 - bazy danych, 126
 - certyfikatów TLS, 115
 - Django, 63
 - Mezzanine, 95, 102, 121, 128
 - pakietów Pythona, 102

- pakietów systemowych, 98
- roli, 133
- zadania Twitter, 116

instancja

- biblioteki Boto, 225
- EC2, 241

instrukcja become, 99

instrukcje iteracyjne, 147

interfejs

- CLI, 316
- REST API, 316
- wiersza poleceń, 133
- WWW, 133

interpreter YAML, 39

J

jawne gromadzenie faktów, 173

język

- PowerShell, 284
- YAML, 39, 42
 - ciągi znaków, 43
 - dzielenie wierszy, 44
 - komentarze, 43
 - listy, 43
 - słowniki, 44
 - wartości logiczne, 43
 - zapis scenariusza, 46

K

klasa AnsibleModule, 204

klucze

- EC2, 233
- hosta, 326

komentarze, 43

komunikacja z kontenerami, 265

komunikat Bad Request, 121

komunikaty o błędach, 273

konfiguracja

- aplikacji Ghost, 263
- bazy danych, 106
- certyfiatów SSL, 169
- potokowania, 188
- serwera Nginx, 115
- serwera testowego, 28
- usługi Nginx, 264

konsola AWS Management Console, 222, 329

konta użytkowników, 288, 290

kontener, 253, 256

- Conductor, 266

kontrola wersji kodu, 33

konwergencja, 24

L

lista obsługiwanych urządzeń, 294

listy, 43

M

magazyn EBS, 236

maszyna wirtualna, 219

- Vagrant, 37, 58, 120

menedżer procesów

- Supervisor, 94

Mezzanine, 89

- instalacja, 95, 102, 121, 128
- rola mezzanine, 128
- uruchomienie, 89

modele subskrypcji, 309

moduł, 24, 47

- add_host, 74
- assert, 277
- debug, 275
- docker_container, 256
- ec2, 238
- ec2_ami, 247
- file, 115
- git, 100
- group_by, 75
- script, 111, 195
- set_fact, 84
- setup, 82
- stat, 278

moduły

- diagnozowanie, 210
- dokumentacja, 209
- implementowanie, 211
- informacje o wykonaniu, 207
- jako skrypty Bash, 211
- kopiowanie do serwera, 197
- sieciowe, 293, 298
- uruchamianie, 196
- w języku Python, 199
- Windows, 286
- własne, 195
- wywołanie, 198

monitoring, 286

N

narzędzie

- Ansible Container, 266
 - ansible-galaxy, 132
 - AWS CLI, 330
 - Packer, 247
 - Vagrant, 28
- Nginx, 93
- aktywacja konfiguracji serwera, 115

O

obraz

- AMI, 223, 236, 246
- maszyny, 223

obsługa błędów, 154

odczytywanie

- adresu IP, 174
- argumentów, 201

opcja

- ControlPersist, 185
- requiretty, 188

opcje argumentów, 202

opisywanie serwerów, 57

orkiestracja

- kontenerów, 258
- wdrożenia, 20

P

pakiety

- Pythona, 102
- systemowe, 98

parametr

- add_file_common_args, 205
- argument_spec, 204
- bypass_checks, 206
- check_invalid_arguments, 205
- ewidencji
 - ansible*_interpreter, 61
 - ansible_connection, 61
 - ansible_python_interpreter, 61
 - ansible_shell_type, 61
- mutually_exclusive, 205
- no_log, 204
- required_one_of, 205

parametry

- konstruktora klasy AnsibleModule, 204
- metody run_command, 207
- połączenia, 300

pętla, 146

- with_fileglob, 148
- with_lines, 147

platforma

- .NET, 284
- Nginx, 40

plik

- ansible.cfg, 32, 231
- app-upgrade.yml, 155
- container.yml, 268, 270
- docker-compose.yml, 258
- ewidencyjny, 57
- files/nginx.conf, 40
- ghost.conf, 257
- greet.yml, 87
- group_vars/production, 68
- group_vars/production/db, 69
- group_vars/production/rabbitmq, 69
- local_settings.py, 107
- mezzanine-across-hosts.yml, 125
- mezzanine-single-host.yml, 124
- playbooks/templates/index.html.j2, 40
- publish.yml, 260
- requirements.txt, 103
- roles/database/defaults/main.yml, 127
- roles/database/tasks/main.yml, 127
- roles/mezzanine/handlers/main.yml, 131
- roles/mezzanine/tasks/django.yml, 129
- roles/mezzanine/tasks/main.yml, 129
- roles/mezzanine/vars/main.yml, 129
- tasks/main.yml, 267
- templates/gunicorn.conf.py.j2, 113
- templates/nginx.conf.j2, 114
- templates/supervisor.conf.j2, 113
- Vagrantfile, 37, 58
- web-notls.yml, 45
- web-tls.yml, 49
- whoami.yml, 78
- z zadaniami, 152

pliki

- argumentów, 197
- ewidencyjne, 62, 217
- JSON, 191
- konfiguracyjne, 225

- Nginx, 39
- urządzeń sieciowych, 302
- usług, 113
- ról, 124, 132
- YAML, 42
- polecenia
 - debugera, 276
 - django-manage, 110
- polecenie
 - ansible-playbook, 87
 - ansible-vault, 157
 - apt-cache, 99
 - async, 193
 - changed_when, 135
 - failed_when, 135
 - sudo, 324
 - with_dict, 148
- połączenia
 - lokalne, 299
 - równoległe, 192
 - SSH, 274
 - z systemem Windows, 283
- porty, 65
 - TCP, 38, 236
- PostgreSQL, 92
- poświadczenia, 224
 - EC2, 329
- potokowanie, 188
 - konfigurowanie, 188
 - włączenie, 188
- PowerShell, 284
- powłoka Bash, 212
- priorytety zmiennych, 88
- procedury, 53
 - nasłuchujące zdarzeń, 168, 172
 - natychmiastowe, 167
 - restartu usług, 113
 - w zadaniach wstępnych i końcowych, 166
 - zaawansowane, 166
- profil instancji, 330
- program
 - cowsay, 42
 - Docker, 253
- programy zewnętrzne, 207
- projekty, 311
- protokół SSH, 295, 321
- pro wizjoner
 - Ansible, 217
 - Ansible Local, 250

- Ansible Remote, 248
 - lokalny, 220
- pro wizjonowanie, 20
- przebiegi
 - strategie, 163
- przekazywanie
 - agentów, 216, 324
 - portów, 215
- przypisywanie tagów, 229

R

- raport JUnit, 180
- rejestr
 - apt, 99
 - umieszczanie obrazu, 271
 - uwierzytelnianie, 271
- rejestrowanie zmiennych, 78
- rekordy zadań, 315
- repozytorium
 - Ansible Galaxy, 133
 - GitHub, 120
- rola
 - database, 124, 126
 - mezzanine, 124, 128
 - ssl, 169
- role, 123
 - dezinstalacja, 134
 - IAM, 329
 - instalacja, 133
 - udostępnianie, 134
 - zależne, 132
- równoległe
 - połączenia, 192
 - pro wizjonowanie maszyn, 218
 - wykonywanie zadań, 193

S

- scenariusz, 21
 - image-facts.yml, 261
 - instalujący agenta Zabbix, 286
 - mezzanine.yml, 95, 117
 - publish.yml, 259
 - web-ami.yml, 248
 - web-notls.yml, 39
- scenariusze
 - akcje, 46
 - debuger, 275

- scenariusze
 - definiowanie zmiennych, 77
 - diagnozowanie, 273
 - dodawanie wpisów, 74
 - idempotentne, 240
 - moduły, 47
 - sprawdzenie, 279
 - sprawdzenie dostępności serwera, 195
 - stosowanie ról, 123
 - uruchomienie, 41, 54
 - wyświetlanie zadań, 95
 - z obsługą błędów, 156
 - zaawansowane, 135
 - zadania, 47
- serwer
 - aplikacyjny
 - Gunicorn, 93
 - Nginx, 93
 - Windows, 283
 - WWW, 41
- serwery
 - konfigurowanie potokowania, 188
 - kopiowane moduły, 197
 - określanie grupy, 160
 - plik argumentów, 197
 - numerowane, 65
 - specyfikacja, 159
 - sprawdzanie dostępności, 195
 - wykluczone z ewidencji, 314
 - wykonywanie zadania, 161
 - wzorce specyfikowania, 159
- sesje SSH, 185
- skalowalność w dół, 24
- skalowanie scenariuszy, 123
- skrypt
 - can_reach, 196, 199
 - can_reach.sh, 196
 - dynamicznej ewidencji, 69, 71, 228
 - scripts/setadmin.py, 112
 - scripts/setsite.py, 112
 - vagrant.py, 72
- skrypty ewidencyjne, 73
- słownik, 44, 80
 - instancji, 238
- specyfikacja serwera, 159
- sprawdzanie
 - klucza, 328
 - projektu, 100
 - scenariusza, 279
 - składni, 279
- SSH
 - agent, 321
 - natywny klient, 321
 - przekazywanie agenta, 323
 - uruchomienie agenta, 322
- sterowanie pętlami, 149
- strategia
 - free, 165
 - linear, 164
- strategie przebiegów, 163
- strona WWW, 40
- struktura roli, 123
- suchy przebieg, 208
- Supervisor, 94
- synchronizowanie projektów, 312
- system DNS, 144
- szablon, 107, 305
 - konfiguracyjny Nginx, 52
 - web.json, 248
- szablony zadań, 313
- szkielet roli, 132
- szyfrowanie
 - danych, 156
 - SSL, 38
 - TLS, 38, 49

§

- śledzenie stanu serwera, 49
- środowisko
 - produkcyjne
 - wdrażanie kontenerów, 272
 - wirtualne, 102

T

- tagi, 223, 228, 281
- tryb weryfikacji, 208
- tworzenie
 - bazy danych, 107
 - certyfikatu TLS, 50
 - konta użytkownika, 317
 - kontenerów, 268
 - lokalnych kont użytkowników, 288
 - modułów, 199
 - obrazu, 257
 - AMI, 246
 - Dockera, 267

- pary kluczy, 233
- plików konfiguracyjnych, 113
- ról, 267
- skryptu dynamicznej ewidencji, 71
- strony WWW, 40
- szablonu, 52
- własnych filtrów, 139
- wyszukiwarki, 146

U

- udostępnianie roli, 134
- układ zainstalowanych plików, 96
- uruchamianie
 - agenta SSH, 322
 - instancji, 231, 239
 - kontenera, 256
 - lokalne kontenerów, 270
 - Mezzanine, 89
 - modułu, 196
 - scenariusza, 41, 54, 120
 - własnych skryptów, 111
 - zadań, 313, 318
- urządzenia sieciowe, 293
 - pliki konfiguracyjne, 302
 - włączenie protokołu SSH, 295
- usługa, 94
 - Nginx, 171
 - SSH, 173
 - VPC, 230
- usuwanie kontenerów, 264
- uwierzytelnianie w rejestrze, 271

V

- Vagrant, 37, 58, 215
 - opcje konfiguracyjne, 215
 - plik ewidencyjny, 217
 - pro wizjoner, 217
 - pro wizjoner lokalny, 220
 - równoległe pro wizjonowanie maszyn, 218
 - uruchomienie scenariusza, 120
- VPC, Virtual Private Cloud, 230

W

- warstwa abstrakcji, 25
- wartości logiczne, 43
- wbudowane zmienne, 85

- wdrażanie
 - aplikacji produkcyjnych, 89
 - kontenerów, 272
 - systemów, 20
- weryfikacja plików, 325
- wiersz poleceń, 330
- WinRM, 283
- wirtualny system operacyjny, 253
- włączenie
 - potokowania, 188
 - protokołu SSH, 295
- wtyczka
 - actionable, 176
 - debug, 176
 - dense, 177
 - foreman, 179
 - hipchat, 179
 - jabber, 180
 - json, 177
 - junit, 180
 - log_plays, 181
 - logentries, 181
 - logstash, 181
 - mail, 182
 - minimal, 177
 - online, 178
 - osx_say, 182
 - profile_tasks, 182
 - selective, 178
 - skippy, 178
 - slack, 183
 - stdout, 175
 - timer, 183
- wtyczki zwrotne, 175
- wyszukiwarka, 140
 - csvfile, 143
 - dnstxt, 144
 - env, 142
 - file, 142
 - password, 143
 - pipe, 142
 - redis_kv, 145
 - template, 143
- wyszukiwarki
 - jako pętla, 149
 - własne, 146
- wyświetlanie
 - faktów, 82
 - grup, 70
 - podzbioru faktów, 82

- wartości zmiennych, 78
- zadań scenariusza, 95
- zainstalowanych ról, 134
- listy serwerów, 279
- wyniku zadania, 135

wywołanie

- modułu, 198
- zewnętrznego programu, 207

Z

zadania, 47

- na kolejnych serwerach, 161
- na komputerze sterującym, 160
- równoległe, 193
- skomplikowane argumenty, 104
- szablony, 313
- w grupie serwerów, 162
- wstępne i końcowe, 125
- wykonane jednokrotnie, 163
- z identycznymi argumentami, 151

zapamiętywanie faktów, 189

- w bazie Memcached, 192
- w bazie Redis, 191
- w plikach JSON, 191

zapisanie konfiguracji, 301

zarządzanie

- konfiguracją, 20
- kontami użytkowników, 288
- lokalnymi grupami, 288
- serwerami Windows, 283
- urządzeniami sieciowymi, 293

zdarzenia, 172

zmienna

- changed, 198
- failed, 199
- groups, 86
- hostvars, 85
- inventory_hostname, 86
- msg, 199

zmiennie, 50, 77

- definiowanie, 84
- definiowanie w wierszu poleceń, 87
- grupowe, 66, 67
- iteracyjne, 149
- jawne i poufne, 96
- priorytety, 88
- rejestrowanie, 78
- serwerowe, 66, 67
- środowiskowe, 224
- uwierzytelniające, 301
- w modułach sieciowych, 298
- wbudowane, 85
- wynikowe oczekiwane, 198
- wyświetlanie wartości, 78

zwielokrotnianie sesji SSH, 185

- opcje, 187
- ręczne włączenie, 186

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Ansible: skuteczne narzędzie najlepszych adminów!

Zarządzanie konfiguracją oprogramowania w systemach sieciowych jest niebanalnym zadaniem. Nawet zwykła aktualizacja czy wdrożenie nowego oprogramowania mogą się skończyć katastrofą, zwłaszcza w przypadku serwerów pracujących pod kontrolą systemów Linux lub Unix. Konieczność pilnowania ustawień w wielu różnych plikach konfiguracyjnych, z których każdy służy innemu elementowi, sprawia, że problemy mogą sprawiać nawet zasadniczo nieskomplikowane czynności – chyba że konfiguracja i wdrażanie oprogramowania w systemie zostaną zautomatyzowane za pomocą odpowiedniego narzędzia, na przykład Ansible.

Istnieje wiele narzędzi do zarządzania konfiguracją oprogramowania. Spośród nich Ansible wyróżnia się szczególnymi zaletami: ma minimalne rozmiary, nie wymaga instalowania czegokolwiek na serwerach i jest proste w użyciu. Dzięki tej książce szybko nauczysz się korzystać z najnowszej wersji Ansible do instalowania nowego kodu aplikacji w środowisku produkcyjnym czy też do lepszego i prostszego zarządzania rozbudowanymi systemami. Zapoznasz się między innymi z oprogramowaniem Ansible Tower, a także dowiesz się, jak skutecznie zarządzać komputerami z systemem Windows i sprzętem sieciowym. Ten niezwykle praktyczny podręcznik powinien być stale pod ręką każdego administratora systemu, wdrożeniowca i programisty!

Lorin Hochstein – jest starszym inżynierem oprogramowania w Netflixie, w zespole zajmującym się inżynierią chaosu. Wcześniej pracował jako inżynier w SendGrid Labs, główny architekt usług chmurowych w Nimbis Services. Obronił doktorat w dziedzinie informatyki na Uniwersytecie Maryland.

René Moser – jest inżynierem systemów. Od lat angażuje się w projekty open source, takie jak ASF CloudStack. Od 2016 roku jest członkiem projektu Ansible Core. Ceni proste systemy, które łatwo skalować. Mieszka z rodziną w Szwajcarii.

W tej książce między innymi:

- Ansible a inne narzędzia do zarządzania konfiguracją systemów
- scenariusze w języku YAML
- testowanie i skalowanie scenariuszy
- techniki wdrażania aplikacji w systemie
- automatyzacja konfigurowania urządzeń sieciowych

	<p>Sprawdź nasze szkolenia</p>	<p>KOD KORZYŚCI Sięgnij po więcej! ▶</p> 
 helion.pl	 <p>AKADEMIA IT & BUSINESS</p> <p>WWW.SZKOLENIA.HELION.PL</p>	<p>ISBN 978-83-283-4171-5</p>  <p>9 788328 341715</p>
<p>INFORMATYKA W NAJLEPSZYM WYDANIU</p>		<p>Cena: 59,00 zł</p>