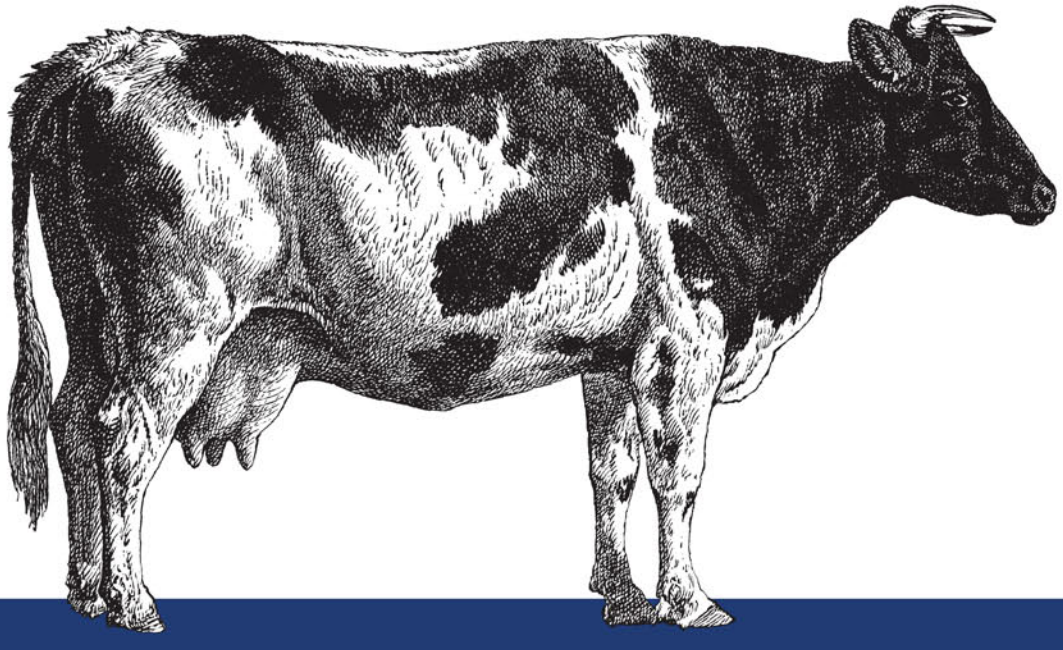
O'REILLY®

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

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.

Spis treści

ROZDZIAŁ 7.

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
    - psycopg2
    - 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 NTP[1]. 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. Załóżmy, że trzeba utworzyć rolę `ntp` konfigurującą synchronizację zegara serwera za pomocą protokołu NTP. Rola zależna może mieć parametry, zatem załóżmy, 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 poradzili 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

Załóżmy, 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.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

Z

O'REILLY®

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!

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

Lorin Hochstein – jest starszym inżynierem oprogramowania w Netfliksie, 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.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-4171-5
9 788328 341715

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 59,00 zł