O'REILLY®

Wydanie III

Ansible w praktyce

Automatyzacja konfiguracji i proste instalowanie systemów

Bas Meijer
Lorin Hochstein
René Moser

Helion

Tytuł oryginału: Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way, 3rd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-8322-152-6

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/ansip3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/ansip3.zip

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 13.

Ansible i kontenery

Platforma Docker od 2013 r., w którym ją udostępniono, szturmem zdobyła świat IT. Trudno jest znaleźć inną technologię, która przyjęłaby się równie szybko. W tym rozdziale opisano, jak się ma Ansible do obrazów kontenerów.

Co to jest kontener?

Wirtualizacja sprzętu polega na tym, że program zwany **hiperwizorem** wirtualizuje cały fizyczny komputer, włącznie z procesorem, pamięcią i urządzeniami, np. dyskami i interfejsami sieciowymi. Dzięki temu wirtualizacja jest bardzo elastyczna. W systemie gościa można uruchomić inny system operacyjny niż w systemie gospodarza (na przykład system Windows Server 2016 może działać na hoście z systemem Red Hat Enterprise Linux). Ponadto maszynę wirtualną można zawiesić lub wznowić tak samo jak fizyczny komputer. Ta elastyczność wprowadza jednak dodatkowe obciążenie.

Konteneryzację czasami określa się mianem **wirtualizacji systemu operacyjnego**, aby odróżnić ją od **wirtualizacji sprzętu**. W zwirtualizowanym systemie operacyjnym (kontenerze) procesy gościa są odizolowane od procesów gospodarza, ale wykorzystują jądro jego systemu operacyjnego. Izolację zapewnia system operacyjny gospodarza.

Konteneryzacja jest formą wirtualizacji. Procesy uruchamiane w systemie operacyjnym gościa nie odwołują się do systemu operacyjnego gospodarza, który działa na fizycznym sprzęcie. Nie mają też bezpośredniego dostępu do zasobów fizycznych, nawet jeżeli zostaną uruchomione z uprawnieniami administratora.

W kontenerze opartym na systemie Linux, np. Docker, procesy muszą być programami linuksowymi. Jednak całkowite obciążenie jest znacznie mniejsze niż w wirtualizacji sprzętowej, ponieważ jest uruchamiany tylko jeden system operacyjny. Oprócz tego procesy wewnątrz kontenera uruchamiają się znacznie szybciej niż w maszynie wirtualnej.

Firma Docker, Inc. (dopisek „Inc.” ma na celu odróżnienie nazwy firmy od nazwy jej produktu) stworzyła nie tylko kontenery. Jej dziełem jest również platforma, której elementami konstrukcyjnymi są kontenery. Używając analogii, kontenery są dla platformy Docker tym, czym maszyny wirtualne dla hiperwizora takiego jak VMware lub VirtualBox. Inne dwie ważne innowacje wprowadzone przez firmę Docker, Inc. to własny format obrazów i interfejs API.

Przeanalizujmy różnice między obrazem kontenera a obrazem maszyny wirtualnej. Obraz kontenera zawiera system plików właściwy dla danego systemu operacyjnego oraz metadane. Od obrazu maszyny wirtualnej różni się przede wszystkim tym, że ma strukturę warstwową. Aby utworzyć nowy obraz kontenera, należy dostosować istniejący obraz, przez odpowiednie dodanie, zmianę lub usunięcie plików. Nowy obraz zawiera odniesienia do oryginalnego obrazu oraz różnice w systemie plików. Dzięki swej warstwowej strukturze jest mniejszy niż obraz maszyny wirtualnej, dzięki czemu przesyłanie go przez internet trwa krócej. Na serwerze projektu Docker (*https://hub.docker.com*) znajduje się rejestr (repozytorium) publicznie dostępnych obrazów.

Platforma Docker udostępnia również interfejs API, umożliwiający zarządzanie nią za pomocą zewnętrznych narzędzi. Wykorzystują go m.in. moduły Ansible o nazwach zaczynających się od `docker_`. Można ich używać do zarządzania kontenerami, ich cyklami życia, oprogramowaniem, systemami operacyjnymi, środowiskiem uruchomieniowym — krótko mówiąc, wszystkimi komponentami.

Kubernetes

Na platformie Kubernetes kontenerów zazwyczaj nie wdraża się przy użyciu Ansible i hosta sterującego, chociaż można w tym celu skorzystać z modułu `k8s` (*https://docs.ansible.com/ansible/latest/collections/kubernetes/core/k8s_module.html*). Pakiet Kubernetes Operator SDK oferuje trzy technologie do zarządzania zasobami: operatory Go, paczki Helm i operatory Ansible. Największą popularnością cieszą się paczki Helm. Nie będziemy się tutaj zagłębiać w szczegóły Kubernetes i Ansible. Jeżeli interesuje Cię ten temat, sięgnij po książkę *Ansible for Kubernetes* Jeffa Geerlinga, operatory zaś szczegółowo opisali Jason Dobies i Joshua Wood w książce *Kubernetes Operators*.

Jeśli poszukujesz publicznej platformy chmurowej do eksperymentów z kontenerami, skorzystaj z Red Hat OpenShift Online (*https://console.redhat.com/openshift*) lub bezpłatnej, próbnej usługi Google Kubernetes Engine (*https://cloud.google.com/kubernetes-engine*). Obie platformy są otwarte i jeżeli dysponujesz własnym sprzętem, możesz je na nim wdrożyć. Gdybyś chciał użyć innej platformy, przeczytaj wpis na blogu *https://kubernetes.io/blog/2019/03/15/kubernetes-setup-using-ansible-and-vagrant*, o tym, jak skonfigurować oprogramowanie Vagrant. Jeszcze innym rozwiązaniem jest użycie narzędzia Kubespray (*https://kubernetes.io/docs/setup/production-environment/tools/kubespray*).

Jak zapewne wiesz, w dużych systemach produkcyjnych platformę Kubernetes często stosuje się w połączeniu z fizycznymi lub wirtualnymi serwerami do przechowywania danych lub uruchamiania określonego oprogramowania (patrz dokumentacja instalacyjna oprogramowania wireserver na stronie *https://docs.wire.com/how-to/install/index.html*). Za pomocą jednolitego języka Ansible można wszystkie tego rodzaju elementy infrastrukturalne zebrać w jedną całość.

Proces uruchamiania aplikacji kontenerowej

Proces uruchamiania typowej aplikacji kontenerowej jest następujący:

1. Pobranie obrazu kontenera z rejestru bazowego.
2. Przystosowanie obrazu na lokalnym komputerze.

3. Wysłanie obrazu kontenera z lokalnego komputera do rejestru.
4. Pobranie obrazu kontenera z rejestru i zapisanie na zewnętrznym komputerze.
5. Przesłanie informacji konfiguracyjnych do kontenera i uruchomienie go na zewnętrznym komputerze.

Zwykle obraz kontenera tworzy się na lokalnym komputerze lub w systemie ciągłej integracji, np. GitLab lub Jenkins, umożliwiającym wykonywanie tego rodzaju operacji. Nowy obraz zapisuje się w miejscu, z którego będzie go można wygodnie pobierać i umieszczać na zewnętrznych komputerach.

Rejestry

Obrazy kontenerów zwykle umieszcza się w rejestrze. W ramach projektu Docker jest utrzymywany rejestr o nazwie Docker Hub, zawierający zarówno publiczne, jak i prywatne obrazy. Narzędzia terminalowe platformy zawierają wbudowane funkcjonalności służące do wysyłania obrazów i pobierania ich z rejestru. Oprócz tego firma Red Hat prowadzi rejestr o nazwie Quay (*https://quay.io*). Można też założyć własny rejestr, korzystając z usługi firmy Sonatype Nexus (*https://www.sonatype.com/products/nexus-repository*). Tworzenie prywatnych rejestrów oferują również publiczni operatorzy chmurowi.

Po umieszczeniu kontenera w rejestrze należy połączyć się z zewnętrznym hostem, pobrać obraz, a następnie uruchomić kontener. Należy zwrócić uwagę, że przy próbie uruchomienia kontenera, którego obrazu nie ma na hoście, platforma Docker automatycznie pobierze go z rejestru. Nie trzeba więc w tym celu jawnie wydawać specjalnego polecenia.

Ansible i Docker

Proces tworzenia obrazu kontenera i uruchomienia go na zewnętrznym komputerze wygląda następująco:

1. Utworzenie scenariusza tworzenia obrazu.
2. Uruchomienie scenariusza na lokalnym komputerze.
3. Wysłanie obrazu z lokalnego komputera do rejestru.
4. Utworzenie i uruchomienie scenariusza pobierającego obraz na zewnętrzny komputer i przekazującego informacje konfiguracyjne.
5. Uruchomienie scenariusza uruchamiającego kontener.

Połączenie z demonem Docker

Wszystkie moduły Ansible komunikują się z demonem Docker. W systemach Linux i macOS wszystkie moduły platformy Docker Desktop powinny działać bez argumentów. Natomiast w przypadku platformy Boot2Docker lub Docker Machine dla systemu macOS, jak również w innych

sytuacjach, w których moduł znajduje się na innym komputerze niż demon Docker, wymagane jest podanie dodatkowych argumentów. Tabela 13.1 zawiera listę opcji, które można określić za pomocą argumentów modułów lub zmiennych środowiskowych. Więcej szczegółów można znaleźć w dokumentacji do modułu `docker_container`.

Tabela 13.1. Opcje połączenia z demonem Docker

Opcja	Zmienna środowiskowa	Wartość domyślna
`docker_host`	`DOCKER_HOST`	`unix://var/run/docker.sock`
`tls_hostname`	`DOCKER_TLS_HOSTNAME`	`localhost`
`api_version`	`DOCKER_API_VERSION`	`auto`
`cert_path`	`DOCKER_CERT_PATH`	*Brak*
`ssl_version`	`DOCKER_SSL_VERSION`	*Brak*
`tls`	`DOCKER_TLS`	`no`
`tls_verify`	`DOCKER_TLS_VERIFY`	`no`
`timeout`	`DOCKER_TIMEOUT`	60 (sekund)

Przykładowa aplikacja: Ghost

W tym rozdziale zmienimy naszą przykładową aplikację z Mezzanine na Ghost — otwartą platformę blogową, podobną do WordPress. Użyjemy oficjalnego kontenera Docker, zawierającego tę aplikację.

W dalszej części rozdziału opisane są następujące operacje:

- uruchomienie kontenera Ghost na lokalnym komputerze,
- uruchomienie kontenera NGINX z protokołem TLS jako frontonu dla aplikacji Ghost,
- wysłanie niestandardowego obrazu NGINX do rejestru,
- instalacja kontenerów Ghost i NGINX na zewnętrznym komputerze.

Uruchomienie kontenera Docker na lokalnym komputerze

Moduł `docker_container` służy do uruchamiania i zatrzymywania kontenerów Docker. Implementuje w tym celu niektóre funkcje narzędzia `docker`, m.in. polecenia `run`, `kill` i `rm`.

Załóżmy, że zainstalowałeś platformę Docker na lokalnym komputerze. Poniższe polecenie pobiera z rejestru Docker obraz aplikacji Ghost i uruchamia go. Port nr 2368 kontenera jest wiązany z portem nr 8000 komputera, zatem aplikacja Ghost będzie dostępna pod adresem *http://localhost:8000*.

```
$ ansible localhost -m docker_container -a "name=test-ghost image=ghost \
  ports=8000:2368"
```

Pierwsze wykonanie powyższego polecenia może zająć kilka minut, ponieważ musi zostać pobrany obraz Docker. Po pomyślnym pobraniu użyj polecenia `docker ps`, aby wyświetlić szczegółowe informacje o uruchomionym kontenerze:

```
$ docker ps --format "table {{.ID }} {{.Image}} {{.Ports}}"
CONTAINER ID IMAGE PORTS
ff728315015e ghost 0.0.0.0:8000->2368/tcp
```

Aby zatrzymać i usunąć kontener, wpisz:

```
$ ansible localhost -m docker_container -a "name=test-ghost state=absent"
```

Moduł `docker_container` oferuje wiele opcji. Niemal każdy argument polecenia `docker` ma swój odpowiednik w postaci opcji modułu.

Utworzenie obrazu na podstawie pliku Dockerfile

Podstawowy sposób tworzenia własnego obrazu kontenera polega na napisaniu specjalnego pliku tekstowego o nazwie *Dockerfile*, podobnego do skryptu powłoki. Standardowy obraz aplikacji Ghost działa doskonale, ale jeżeli jest wymagany do niego np. bezpieczny dostęp, trzeba skonfigurować fronton w postaci serwera WWW z protokołem TLS.

W ramach projektu NGINX utworzono standardowy obraz serwera, jednak aby użyć go jako frontonu dla aplikacji Ghost, trzeba aktywować w nim protokół TLS, tak jak to zrobiliśmy w rozdziale 7. dla aplikacji Mezzanine. Listing 13.1 przedstawia przykładowy plik *Dockerfile* służący do tego celu.

Listing 13.1. Plik Dockerfile

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY ghost.conf /etc/nginx/conf.d/ghost.conf
```

Listing 13.2 przedstawia konfigurację serwera NGINX jako frontonu dla aplikacji Ghost. Od pliku dla Mezzanine różni się przede wszystkim tym, że tutaj serwer NGINX nie wykorzystuje do komunikacji z aplikacją gniazda domeny Unix, tylko port TCP 2368. Ponadto ścieżka do pliku z certyfikatem TLS ma nazwę */certs*.

Listing 13.2. Plik ghost.conf

```
server {
   listen 80 default_server;
   listen [::]:80 default_server;
   server_name _;
   return 301 https://$host$request_uri;
}
server {
   listen 443 ssl;
   client_max_body_size 10M;
   keepalive_timeout 15;
   ssl_certificate      /certs/nginx.crt;
   ssl_certificate_key  /certs/nginx.key;
   ssl_session_cache    shared:SSL:10m;
   ssl_session_timeout  10m;
   ssl_protocols TLSv1.3;
   ssl_ciphers EECDH+AESGCM:EDH+AESGCM;
   ssl_prefer_server_ciphers on;
   location / {
      proxy_pass          http://ghost:2368;
      proxy_set_header    X-Real-IP $remote_addr;
```

```
        proxy_set_header    Host $http_host;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

W powyższej konfiguracji przyjęto założenie, że kontener NGINX może wykorzystywać do komunikacji z aplikacją Ghost nazwę hosta `ghost`. Sprawdź to, wdrażając kontenery. W przeciwnym razie kontener NGINX nie będzie miał dostępu do kontenera Ghost.

Zakładając, że pliki *Dockerfile* i *nginx.conf* są umieszczone w katalogu *nginx*, zostanie utworzony nowy obraz o nazwie *ansiblebook/nginx-ghost*. Prefiks *ansiblebook/* oznacza, że obraz będzie umieszczony w repozytorium Docker Hub o nazwie *ansiblebook/nginx-ghost*. Możesz jednak użyć innego prefiksu, zawierającego nazwę Twojego konta w serwisie Docker (*https://hub.docker.com)*.

```
- name: Utworzenie obrazu NGINX
  docker_image:
    build:
      path: ./nginx
    source: build
    name: ansiblebook/nginx-ghost
    state: present
    force_source: "{{ force_source | default(false) }}"
    tag: "{{ tag | default('latest') }}"
```

Sprawdź zawartość repozytorium za pomocą polecenia `docker images`:

```
$ docker images
REPOSITORY                TAG      IMAGE ID       CREATED         SIZE
ansiblebook/nginx-ghost   latest   e8d39f3e9e57   6 minutes ago   133MB
ghost                     latest   e8bc5f42fe28   3 days ago      450MB
nginx                     latest   87a94228f133   3 weeks ago     133MB
```

Zwróć uwagę, że użycie modułu `docker_image` z nazwą istniejącego obrazu nie przyniesie żadnego efektu, nawet jeżeli plik *Dockerfile* zostanie zmieniony. W celu wymuszenia utworzenia obrazu należy użyć dodatkowego argumentu, `force_source=true`, jak niżej:

```
$ ansible-playbook build.yml -e force_source=true
```

Dobrą praktyką jest stosowanie argumentu `tag` (etykieta) z numerem wersji, zwiększanym przy każdej kompilacji. Zaktualizowany obraz można wtedy utworzyć za pomocą modułu `docker_image` bez konieczności stosowania wymuszającego argumentu. Domyślnie jest używana etykieta `default`, niezbyt przydatna przy oznaczaniu wersji.

```
$ ansible-playbook build.yml -e tag=v2
```

Wysłanie obrazu do rejestru

Do wysłania obrazu do rejestru Docker Hub użyj osobnego scenariusza, pokazanego w listingu 13.3. Pamiętaj, że scenariusz musi się najpierw zalogować do rejestru za pomocą modułu `docker_login`. Domyślnie moduły `docker_login` i `docker_image` odwołują się do rejestru Docker Hub.

Listing 13.3. Plik publish.yml

```
---
- name: Wysłanie obrazu do rejestru Docker Hub
  hosts: localhost
  gather_facts: false

  vars_prompt:
    - name: username
      prompt: Podaj nazwę konta w rejestrze Docker Hub
    - name: password
      prompt: Podaj hasło
      private: true

  tasks:
    - name: Uwierzytelnianie w rejestrze
      docker_login:
        username: "{{ username }}"
        password: "{{ password }}"
      tags:
        - login

    - name: Wysłanie obrazu
      docker_image:
        name: "ansiblebook/nginx-ghost"
        push: true
        source: local
        state: present
      tags:
        - push
```

Gdybyś chciał użyć innego rejestru, umieść w sekcji `docker_login` parametr `registry_url`. Jeżeli rejestr wykorzystuje niestandardowy port HTTP/HTTPS, wpisz go po nazwie hosta. Listing 13.4 przedstawia zmieniony scenariusz, w którym rejestr ma adres *http://reg.example.com*.

Listing 13.4. Plik publish.yml z adresem niestandardowego rejestru

```
tasks:
  - name: Uwierzytelnianie w rejestrze
    docker_login:
      registry_url: https://reg.example.com
      username: "{{ username }}"
      password: "{{ password }}"
    tags:
      - login

  - name: Wysłanie obrazu
    docker_image:
      name: reg.example.com/ansiblebook/nginx-ghost
      push: true
      source: local
      state: present
    tags:
      - push
```

Zwróć uwagę, że w powyższym scenariuszu odpowiednio zmieniono również nazwę tworzonego obrazu: `reg.example.com/ansiblebook/nginx-ghost`.

Konfigurowanie kontenerów na lokalnym komputerze

Często uruchamia się wiele kontenerów Docker i łączy je ze sobą. Podczas tworzenia aplikacji wszystkie kontenery są zazwyczaj uruchamiane na lokalnym komputerze, natomiast w środowisku produkcyjnym są zwykle hostowane na różnych maszynach. Aplikacje często wdraża się w klastrze Kubernetes, natomiast bazy danych — na osobnych serwerach.

Na potrzeby lokalnego programowania, gdy wszystkie kontenery są uruchamiane na jednym komputerze, opracowano narzędzie Docker Compose, ułatwiające przenoszenie kontenerów i łączenie ich ze sobą. Można nim sterować, tj. uruchamiać i zatrzymywać usługi, za pomocą modułu Ansible `docker_compose`.

Listing 13.5 przedstawia scenariusz *docker-compose.yml* uruchamiający kontenery NGINX i Ghost. Przyjęto w nim założenie, że w jednym z katalogów znajduje się plik z certyfikatem TLS.

Listing 13.5. Plik docker-compose.yml

```
version: '2'
services:
  nginx:
  image: ansiblebook/nginx-ghost
  ports:
    - "8000:80"
    - "8443:443"
  volumes:
    - ${PWD}/certs:/certs
  links:
    - ghost
  ghost:
    image: ghost
```

Listing 13.6 przedstawia scenariusz tworzenia niestandardowego obrazu NGINX z samopodpisanym certyfikatem, a następnie uruchomienia usług zdefiniowanych w listingu 13.5.

Listing 13.6. Plik ghost.yml

```
---
- name: Uruchomienie kontenera Ghost na lokalnym komputerze
  hosts: localhost
  gather_facts: false
  tasks:

    - name: Utworzenie obrazu kontenera NGINX
      docker_image:
        build:
          path: ./nginx
        source: build
        name: ansiblebook/nginx-ghost
        state: present
        force_source: "{{ force_source | default(false) }}"
        tag: "{{ tag | default('v1') }}"

    - name: Utworzenie certyfikatu
      command: >
        openssl req -new -x509 -nodes
        -out certs/nginx.crt -keyout certs/nginx.key
```

```
        -subj '/CN=localhost' -days 365
      args:
        creates: certs/nginx.crt

    - name: Uruchomienie usług
      docker_compose:
        project_src: .
        state: present
...
```

Moduł `docker_compose` jest ciekawym komponentem dla programistów aplikacji. Gdy aplikacja dojrzeje do wdrożenia w środowisku produkcyjnym, wymagania stawiane środowisku wykonawczemu często prowadzą do wybrania platformy Kubernetes.

Uzyskiwanie informacji o lokalnym obrazie

Moduł `docker_image_info` dostarcza metadanych o obrazie przechowywanym na lokalnym komputerze. Listing 13.7 przedstawia przykładowy scenariusz, w którym ten moduł jest użyty do odczytania informacji o porcie i woluminach dyskowych wykorzystywanych przez kontener Ghost.

Listing 13.7. Plik image-info.yml

```
---
- name: Odczytanie wykorzystywanych portów i woluminów
  hosts: localhost
  gather_facts: false
  vars:
    image: ghost
  tasks:
    - name: Odczytanie informacji o obrazie
      docker_image_info:
        name: ghost
      register: ghost

    - name: Wyodrębnienie portów
      set_fact:
        ports: "{{ ghost.images[0].Config.ExposedPorts.keys() }}"

    - name: Wykorzystywany może być tylko jeden port
      assert:
        that: "ports|length == 1"

    - name: Wyświetlenie wykorzystywanego portu
      debug:
        msg: "Wykorzystywany port: {{ ports[0] }}"

    - name: Wyodrębnienie woluminów
      set_fact:
        volumes: "{{ ghost.images[0].Config.Volumes.keys() }}"

    - name: Wyświetlenie woluminów
      debug:
        msg: "Wolumin: {{ item }}"
      with_items: "{{ volumes }}"
...
```

Wynik wykonania scenariusza jest następujący:

```
$ ansible-playbook image-info.yml
PLAY [Odczytanie wykorzystywanych portów i woluminów] ********************************
TASK [Odczytanie informacji o obrazie] ************************************************
ok: [localhost]
TASK [Wyodrębnienie portów] ***********************************************************
ok: [localhost]
TASK [Wykorzystywany może być tylko jeden port] ***************************************
ok: [localhost] ==> {
    "changed": false,
    "msg": "All assertions passed"
}
TASK [Wyświetlenie wykorzystywanego portu] ********************************************
ok: [localhost] ==> {
    "msg": "Wykorzystywany port: 2368/tcp"
}
TASK [Wyodrębnienie woluminów] ********************************************************
ok: [localhost]
TASK [Wyświetlenie woluminów] *********************************************************
ok: [localhost] => (item=/var/lib/ghost/content) => {
    "msg": "Wolumin: /var/lib/ghost/content"
}
```

Używaj modułu `docker_image_info` do rejestrowania ważnych informacji o swoich obrazach.

Wdrożenie aplikacji kontenerowej

Domyślnie aplikacja Ghost wykorzystuje bazę danych SQLite, jednak w tym rozdziale użyjemy MySQL. Za pomocą programu Vagrant utworzymy dwie maszyny. Na pierwszej (`ghost`) uruchomimy platformę Docker z kontenerami Ghost i NGINX, a na drugiej (`mysql`) bazę MySQL, w której aplikacja Ghost będzie zapisywała swoje dane.

W tym przykładzie przyjęto założenie, że poniższe zmienne są zdefiniowane w miejscu dostępnym na obu powyższych maszyn, np. w pliku *group_vars/all*:

- `database_name=ghost`,
- `database_user=ghost`,
- `database_password=mysupersecretpassword`.

Utworzenie maszyny MySQL

Aby utworzyć maszynę z bazą MySQL, trzeba zainstalować kilka pakietów. Służy do tego scenariusz przedstawiony w listingu 13.8.

Listing 13.8. Utworzenie maszyny MySQL

```
- name: Utworzenie maszyny z bazą danych
  hosts: mysql
  become: true
  gather_facts: false
  tasks:

    - name: Instalacja pakietów dla bazy MySQL
```

```
    apt:
      update_cache: true
      cache_valid_time: 3600
      name:
        - mysql-server
        - python3-pip
      state: present

  - name: Instalacja zależności
    pip:
      name: PyMySQL
      state: present
      executable: /usr/bin/pip3
```

Wdrożenie bazy danych dla aplikacji Ghost

Aby wdrożyć bazę danych dla aplikacji Ghost, należy utworzyć samą bazę, a w niej konto użytkownika, który będzie łączył się z innej maszyny. Oznacza to, że trzeba ponownie skonfigurować adres sieciowy wykorzystywany przez bazę MySQL, a następnie uchwyt, który będzie ponownie uruchamiał bazę tylko wtedy, gdy zmieni się konfiguracja. Zadania te realizuje scenariusz przedstawiony na listingu 13.9.

Listing 13.9. Scenariusz wdrożenia bazy danych

```
- name: Wdrożenie bazy danych
  hosts: database
  become: true
  gather_facts: false

  handlers:
    - name: Ponowne uruchomienie MySQL
      systemd:
        name: mysql
        state: restarted
  tasks:

    - name: Nasłuch
      lineinfile:
        path: /etc/mysql/mysql.conf.d/mysqld.cnf
        regexp: '^bind-address'
        line: 'bind-address     = 0.0.0.0'
        state: present
      notify: Ponowne uruchomienie MySQL

    - name: Utworzenie bazy danych
      mysql_db:
        name: "{{ database_name }}"
        state: present
        login_unix_socket: /var/run/mysqld/mysqld.sock

    - name: Utworzenie konta użytkownika
      mysql_user:
        name: "{{ database_user }}"
        password: "{{ database_password }}"
        priv: '{{ database_name }}.*:ALL'
        host: '%'
        state: present
        login_unix_socket: /var/run/mysqld/mysqld.sock
```

W tym przykładzie baza wykorzystuje adres 0.0.0.0, co oznacza, że będzie można się z nią połączyć z dowolnego komputera (nie jest to jednak bezpieczna konfiguracja).

Fronton

Wdrożenie frontonu jest bardziej skomplikowane, ponieważ składa się on z dwóch kontenerów, Ghost i NGINX, które trzeba ze sobą połączyć. Oprócz tego należy w kontenerze Ghost umieścić informacje konfiguracyjne, aby miał dostęp do bazy danych MySQL.

Do połączenia kontenerów Ghost i NGINX wykorzystamy sieć Docker. W tym celu utworzymy niestandardową sieć i dołączymy do niej oba kontenery, które będą odwoływać się do siebie za pomocą nazw hostów.

Utworzenie sieci Docker jest proste:

```
- name: Utworzenie sieci
  docker_network:
    name: "{{ net_name }}"
```

Dobrym rozwiązaniem jest umieszczenie nazwy sieci w zmiennej, ponieważ będzie ona wykorzystywana w każdym dołączanym kontenerze. Listing 13.10 przedstawia fragment scenariusza.

Listing 13.10. Scenariusz wdrożenia kontenera Ghost

```
- name: Wdrożenie kontenera Ghost
  hosts: ghost
  become: true
  gather_facts: false

  vars:
    url: "https://{{ inventory_hostname }}"
    database_host: "{{ groups['database'][0] }}"
    data_dir: /data/ghostdata
    certs_dir: /data/certs
    net_name: ghostnet

  tasks:
    - name: Utworzenie sieci
      docker_network:
        name: "{{ net_name }}"
```

W powyższym scenariuszu przyjęto założenie, że istnieje grupa `database` zawierająca jednego hosta. Ta informacja jest wykorzystana do utworzenia zmiennej `database_host`.

Fronton: Ghost

Kontener Ghost trzeba skonfigurować tak, aby łączył się z bazą danych MySQL, jak również działał w trybie produkcyjnym. W tym celu należy do polecenia `npm start` przesłać parametr `production`. Tę informację można umieścić w kontenerze z wykorzystaniem zmiennej środowiskowej. Oprócz tego kontener musi mieć możliwość trwałego zapisywania generowanych plików w zamontowanym woluminie.

Listing 13.11 przedstawia fragment scenariusza tworzącego katalog, w którym będą zapisywane dane. Dodatkowo scenariusz ten uruchamia kontener podłączony do sieci `ghostnet`.

Listing 13.11. Scenariusz tworzenia kontenera Ghost

```
- name: Utworzenie katalogu ghostdata
  file:
    path: "{{ data_dir }}"
    state: directory
    mode: '0750'

- name: Uruchomienie kontenera ghost
  docker_container:
    name: ghost
    image: ghost
    container_default_behavior: compatibility
    network_mode: "{{ net_name }}"
    networks:
      - name: "{{ net_name }}"
    volumes:
      - "{{ data_dir }}:/var/lib/ghost/content"
    env:
      database__client: mysql
      database__connection__host: "{{ database_host }}"
      database__connection__user: "{{ database_user }}"
      database__connection__password: "{{ database_password }}"
      database__connection__database: "{{ database_name }}"
      url: "https://{{ inventory_hostname }}"
      NODE_ENV: production
```

Zwróć uwagę, że powyższy scenariusz nie konfiguruje żadnych portów, ponieważ z kontenerem Ghost komunikuje się tylko kontener NGINX.

Fronton: NGINX

Konfigurację kontenera NGINX określiliśmy w scenariuszu tworzącym obraz `ansiblebook/nginx--ghost`. Kontener wykorzystuje do komunikacji adres `ghost:2368`. Trzeba jednak skopiować certyfikaty TLS. Tak jak w poprzednich przykładach wystarczy użyć samopodpisanych certyfikatów. Ilustruje to listing 13.12.

Listing 13.12. Scenariusz tworzenia kontenera NGINX

```
- name: Utworzenie katalogu certs
  file:
    path: "{{ certs_dir }}"
    state: directory
    mode: '0750'

- name: Utworzenie certyfikatu TLS
  command: >
    openssl req -new -x509 -nodes
    -out "{{ certs_dir }}/nginx.crt"
    -keyout "{{ certs_dir }}/nginx.key"
    -subj "/CN={{ ansible_host }}" -days 3650
  args:
    creates: certs/nginx.crt

- name: Uruchomienie kontenera NGINX
  docker_container:
    name: nginx_ghost
```

```
    image: ansiblebook/nginx-ghost
    container_default_behavior: compatibility
    network_mode: "{{ net_name }}"
    networks:
      - name: "{{ net_name }}"
    pull: true
    ports:
      - "0.0.0.0:80:80"
      - "0.0.0.0:443:443"
    volumes:
      - "{{ certs_dir }}:/certs"
```

Samopodpisane certyfikaty można stosować jedynie tymczasowo, podczas budowania wewnętrznej sieci. Gdy tylko inni użytkownicy zaczną korzystać z usługi, należy użyć certyfikatu podpisanego przez zaufany urząd.

Usunięcie kontenerów

Za pomocą Ansible można łatwo zatrzymywać i usuwać kontenery, co jest przydatną opcją podczas pisania i testowania skryptów wdrożeniowych. Listing 13.13 przedstawia scenariusz usuwający oba kontenery.

Listing 13.13. Usunięcie kontenerów

```
- name: Usunięcie wszystkich kontenerów oraz sieci
  hosts: ghost
  become: true
  gather_facts: false
  tasks:

    - name: Usunięcie kontenerów
      docker_container:
        name: "{{ item }}"
        state: absent
        container_default_behavior: compatibility
      loop:
        - nginx_ghost
        - ghost

    - name: Usunięcie sieci
      docker_network:
        name: ghostnet
        state: absent
```

Moduł `docker_container` ma parametr `cleanup` typu `boolean`, powodujący usunięcie kontenera po zakończeniu działania.

Podsumowanie

Platforma Docker udowodniła swoją potęgę. W tym rozdziale dowiedziałeś się, jak zarządzać obrazami, kontenerami i sieciami za pomocą modułów Ansible.

Minimalne rozmiary, prostota i wyjątkowa skuteczność — poznaj Ansible!

Ansible służy do automatyzacji wdrożeń oprogramowania i zarządzania jego konfiguracjami. Inżynierowie cenią ten framework za minimalne rozmiary, brak konieczności instalowania czegokolwiek na serwerach i prostotę użytkowania. Oferuje on proste i bardzo przydatne funkcje przeznaczone do automatyzacji wielowarstwowych środowisk, przydaje się też do obsługi ciągłej integracji i ciągłego wdrażania oprogramowania (CI/CD) bez żadnego przestoju. Może służyć do różnych celów: przygotowania infrastruktury jako kodu, wdrożeń aplikacji czy automatyzacji codziennych, czasochłonnych zadań administracyjnych.

Ta książka jest przeznaczona dla programistów i administratorów, którzy poszukują wydajnej metody zarządzania systemami. Pokazano w niej, w jaki sposób działa Ansible i jak należy przygotować go do pracy. Omówiono sposoby tworzenia scenariuszy (są to skrypty do zarządzania konfiguracją), zasady zarządzania zewnętrznymi serwerami, a także zaprezentowano najciekawsze funkcjonalności tego oprogramowania: wbudowane deklaratywne moduły. W tym wydaniu uwzględniono zmiany wynikające z dynamicznego rozwoju Ansible, dodano też kilka rozdziałów poświęconych kontenerom, platformie Molecule, kolekcjom Ansible, obrazom i infrastrukturze chmurowej. Wszystkie kody zostały zaktualizowane, a całość wzbogacono o praktyczne wskazówki dotyczące dobrych praktyk programistycznych na platformach do weryfikowania kodu.

W książce:

- zarządzanie konfiguracją i wdrożeniami systemów za pomocą Ansible
- dobre praktyki pracy z Ansible
- formaty kolekcji, moduły i wtyczki
- generowanie obrazów kontenerów i instancji chmurowych
- tworzenie infrastruktury chmurowej
- automatyzacja procesów CI/CD w środowisku programistycznym
- platforma Ansible Automation w metodyce DevOps

Bas Meijer jest niezależnym inżynierem oprogramowania i konsultantem metodyki DevOps. Otwartym oprogramowaniem zajmuje się od wczesnych lat dziewięćdziesiątych.

Lorin Hochstein jest starszym inżynierem oprogramowania w Netfliksie, zajmuje się inżynierią chaosu.

René Moser jest inżynierem systemów, zaangażowanym w projekt ASF CloudStack. Autor integracji CloudStack z Ansible.

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 978-83-8322-152-6

9 788383 221526

Cena: 99,00 zł