

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

© 2023 Helion S.A.

Authorized Polish translation of the English edition of *Ansible: Up and Running*, 3E ISBN 9781098109158
© 2022 Bas Meijer.

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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wstęp do trzeciego wydania	15
1. Wprowadzenie	19
Uwaga do wersji	20
Do czego nadaje się Ansible?	20
Jak działa Ansible?	21
Na czym polega wielkość Ansible?	23
Prostota	23
Użyteczność	25
Bezpieczeństwo	28
Czy Ansible nie jest zbyt proste?	30
Co musisz wiedzieć?	30
Czego tu nie znajdziesz?	31
Co dalej?	31
2. Instalacja i konfiguracja	32
Instalacja Ansible	32
Luźne zależności	33
Uruchomienie Ansible w kontenerze	33
Rozwijanie Ansible	33
Konfiguracja serwera testowego	34
Konfiguracja serwera testowego za pomocą narzędzia Vagrant	34
Wprowadzanie do Ansible informacji o serwerze	36
Ułatwienia dzięki plikowi ansible.cfg	38
Nie miej litości	41
Przydatne opcje konfiguracyjne środowiska Vagrant	41
Przekazywanie portów i prywatne adresy IP	41
Włączanie przekazywania agentów	43
Prowizjoner Docker	43
Lokalny prowizjoner Ansible	43
Kiedy uruchamiany jest prowizjoner?	44

Wtyczki Vagrant	44
Hostmanager	44
VBGuest	45
Dostosowywanie maszyny wirtualnej VirtualBox	45
Plik Vagrantfile to kod Ruby	45
Konfiguracja produkcyjna	48
Podsumowanie	48
3. Scenariusze — pierwsze kroki	49
Wstępne wymagania	49
Bardzo prosty scenariusz	50
Tworzenie pliku konfiguracyjnego Nginx	51
Tworzenie strony WWW	51
Definiowanie grupy serwerów WWW	52
Uruchomienie scenariusza	53
Scenariusz to plik YAML	53
Początek pliku	54
Koniec pliku	54
Komentarze	55
Wcięcia i białe znaki	55
Ciągi znaków	55
Wartości logiczne	55
Listy	56
Słowniki	57
Dzielenie wierszy	57
Czysty YAML zamiast argumentów tekstowych	58
Anatomia scenariusza	58
Akcje	59
Zadania	60
Moduły	61
Korzystanie z dokumentacji Ansible	61
Wszystko razem	62
Czy coś się zmieniło? Śledzenie stanu serwera	62
Coś ciekawszego: szyfrowanie TLS	63
Tworzenie certyfikatu TLS	63
Zmienne	63
Cudzysłowy w ciągach znaków	64
Tworzenie szablonu konfiguracyjnego Nginx	65
Pętle	66
Procedury	67
Kilka cech procedur, o których należy pamiętać	67
Testy	68

Weryfikacja	68
Scenariusz	69
Uruchomienie scenariusza	70
Podsumowanie	72
4. Ewidencja: opisywanie serwerów	73
Plik ewidencyjny	74
Wstępne wymagania: kilka maszyn Vagrant	74
Funkcjonalne parametry ewidencji	77
Zmienianie domyślnych wartości parametrów funkcjonalnych	78
Grupy, grupy i jeszcze raz grupy	78
Przykład: instalacja aplikacji Django	79
Aliasy i porty	82
Grupy grup	82
Serwery numerowane (zwierzaki kontra stado)	82
Zmienne serwerowe i grupowe w pliku ewidencyjnym	83
Zmienne serwerowe i grupowe w osobnych plikach	85
Dynamiczna ewidencja	86
Wtyczki ewidencyjne	87
Amazon EC2	87
Azure Resource Manager	87
Interfejs skryptu dynamicznej ewidencji	88
Tworzenie skryptu dynamicznej ewidencji	89
Podział ewidencji na kilka plików	92
Dodawanie wpisów w trakcie działania scenariusza	
za pomocą modułów <code>add_host</code> i <code>group_by</code>	92
Moduł <code>add_host</code>	92
Moduł <code>group_by</code>	94
Podsumowanie	95
5. Zmienne i fakty	96
Definiowanie zmiennych w scenariuszu	96
Definiowanie zmiennych w oddzielnych plikach	96
Układ katalogów	97
Wyświetlanie wartości zmiennych	97
Interpolacja zmiennych	97
Rejestrowanie zmiennych	98
Fakty	101
Wyświetlanie wszystkich faktów skojarzonych z serwerem	102
Wyświetlanie podzbioru faktów	103
Fakty i informacje może zwracać każdy moduł	103
Fakty lokalne	104
Definiowanie nowej zmiennej za pomocą modułu <code>set_fact</code>	105

Wbudowane zmienne	105
hostvars	106
inventory_hostname	107
groups	107
Definiowanie zmiennych w wierszu poleceń	107
Priorytety	109
Podsumowanie	110
6. Mezzanine: nasza testowa aplikacja	111
Dlaczego wdrażanie aplikacji produkcyjnych jest skomplikowane?	111
Baza danych: PostgreSQL	113
Serwer aplikacyjny: Gunicorn	114
Serwer WWW: Nginx	114
Menedżer procesów: Supervisor	115
Podsumowanie	115
7. Instalacja Mezzanine za pomocą Ansible	116
Wyświetlanie zadań scenariusza	116
Układ zainstalowanych plików	117
Zmienne jawne i poufne	117
Instalowanie wielu pakietów	119
Instrukcja become w zadaniu	119
Aktualizacja rejestru apt	120
Sprawdzenie projektu za pomocą modułu git	121
Instalacja Mezzanine i innych pakietów w środowisku wirtualnym	122
Krótka dygresja: skomplikowane argumenty w zadaniach	125
Konfiguracja bazy danych	126
Tworzenie pliku local_settings.py na podstawie szablonu	127
Polecenia django-manage	129
Uruchamianie własnych skryptów Pythona w kontekście aplikacji	130
Utworzenie plików konfiguracyjnych usług	133
Aktywacja konfiguracji serwera Nginx	135
Instalacja certyfikatów TLS	136
Instalacja zadania Twitter w harmonogramie cron	137
Cały scenariusz	138
Uruchomienie scenariusza na maszynie wirtualnej Vagrant	142
Diagnostyka	143
Brak dostępu do repozytorium GitHub	143
Brak dostępu do adresu 192.168.33.10.nip.io	143
Komunikat Bad Request (400)	143
Podsumowanie	143

8. Diagnozowanie scenariuszy	144
Czytelne komunikaty o błędach	144
Diagnozowanie połączenia SSH	145
Typowe wyzwania związane z usługą SSH	148
PasswordAuthentication no	148
Połączenie z użyciem innego konta	148
Błąd weryfikacji klucza	149
Sieci prywatne	149
Moduł debug	150
Debugger scenariuszy	150
Moduł assert	152
Sprawdzenie scenariusza przed uruchomieniem	154
Sprawdzenie składni	154
Wyświetlenie listy serwerów	154
Wyświetlenie listy zadań	155
Tryb weryfikacji	155
Różnice (wyświetlenie zmian w plikach)	155
Tagi	156
Limits	157
Podsumowanie	157
9. Skalowanie scenariuszy: role	158
Podstawowa struktura roli	158
Przykład: role database i mezzanine	159
Stosowanie ról w scenariuszach	160
Zadania wstępne i końcowe	161
Rola database instalująca bazę danych	162
Rola mezzanine instalująca aplikację Mezzanine	164
Tworzenie plików i katalogów ról za pomocą narzędzia ansible-galaxy	168
Role zależne	169
Repozytorium Ansible Galaxy	170
Interfejs WWW	170
Interfejs wiersza poleceń	170
Wymagania ról w praktyce	171
Udostępnianie własnej roli	172
Podsumowanie	172
10. Zaawansowane scenariusze	173
Obsługa błędnie działających poleceń	173
Filtry	176
Filtr default	176
Filtry zarejestrowanych zmiennych	176

Filtry ścieżek plików	177
Tworzenie własnych filtrów	178
Wyszukiwarki	179
file	181
pipe	181
env	182
password	182
template	182
csvfile	182
dig	183
redis	184
Utworzenie własnej wyszukiwarki	185
Zaawansowane pętle	185
Wyszukiwarki with	186
with_lines	186
with_fileglob	187
with_dict	188
Wyszukiwarki jako pętle	188
Sterowanie pętlami	189
Określanie nazwy zmiennej iteracyjnej	189
Umieszczanie etykiet w wynikach	190
Importowanie i dołączanie plików	191
Dynamiczne dołączanie plików	192
Dołączanie ról	193
Sterowanie realizacją roli	193
Bloki	194
Obsługa błędów za pomocą bloków	194
Szyfrowanie poufnych danych	197
Kilka sejfów z różnymi hasłami	199
Podsumowanie	199
11. Dostosowywanie serwerów, przebiegów i procedur	200
Wzorce specyfikowania serwerów	200
Określanie grupy serwerów	201
Wykonywanie zadania na komputerze sterującym	201
Jawne gromadzenie faktów	202
Odczytywanie adresu IP serwera	202
Wykonywanie zadania na innym komputerze niż serwer	203
Wykonywanie zadania na kolejnych serwerach	204
Wykonywanie zadania w grupie serwerów	205
Jednokrotne wykonanie zadania	206

Selektywne wykonywanie zadań	206
step	206
start-at-task	207
Tagi wykonywane	207
Tagi pomijane	208
Strategie przebiegów	208
Strategia linear	208
Strategia free	209
Zaawansowane procedury	211
Procedury w zadaniach wstępnych i końcowych	211
Procedury natychmiastowe	212
Metapolecenia	213
Procedury powiadamiające inne procedury	213
Procedury nasłuchujące	214
Procedury nasłuchujące: konfiguracja certyfikatów SSL	214
Podsumowanie	219
12. Zarządzanie serwerami Windows	220
Połączenie z systemem Windows	220
PowerShell	221
Moduły Windows	223
Nasza maszyna programistyczna Java	224
Tworzenie lokalnych kont użytkowników	225
Funkcje Windows	226
Instalacja oprogramowania za pomocą menedżera Chocolatey	226
Konfiguracja środowiska Java	227
Aktualizacja systemu Windows	228
Podsumowanie	228
13. Ansible i kontenery	229
Kubernetes	230
Proces uruchamiania aplikacji kontenerowej	230
Rejestry	231
Ansible i Docker	231
Połączenie z demonem Docker	231
Przykładowa aplikacja: Ghost	232
Uruchomienie kontenera Docker na lokalnym komputerze	232
Utworzenie obrazu na podstawie pliku Dockerfile	233
Wysyłanie obrazu do rejestru	234
Konfigurowanie kontenerów na lokalnym komputerze	236
Uzyskiwanie informacji o lokalnym obrazie	237
Wdrożenie aplikacji kontenerowej	238
Utworzenie maszyny MySQL	238

Wdrożenie bazy danych dla aplikacji Ghost	239
Fronton	240
Fronton: Ghost	240
Fronton: NGINX	241
Usunięcie kontenerów	242
Podsumowanie	242
14. Kontrola jakości przy użyciu platformy Molecule	243
Instalacja i konfiguracja	243
Konfigurowanie sterowników Molecule	244
Utworzenie roli Ansible	245
Scenariusze Molecule	246
Żądany stan	246
Konfigurowanie scenariusza Molecule	246
Zarządzanie maszynami wirtualnymi	247
Zarządzanie kontenerami	248
Polecenia Molecule	249
Lintowanie	250
YAMLLint	250
ansible-lint	251
ansible-later	252
Weryfikatory	252
Ansible	253
Goss	253
TestInfra	254
Podsumowanie	255
15. Kolekcje	256
Instalacja kolekcji	257
Wyświetlenie listy kolekcji	258
Stosowanie kolekcji w scenariuszu	258
Tworzenie kolekcji	259
Podsumowanie	260
16. Tworzenie obrazów	261
Tworzenie obrazów za pomocą narzędzia Packer	261
Tworzenie maszyny wirtualnej w środowisku Vagrant VirtualBox	261
Połączenie narzędzi Packer i Vagrant	264
Obrazy chmurowe	265
Google Cloud Platform	265
Azure	267
Amazon EC2	268
Scenariusz	269

Obraz Docker: GCC 11	270
Podsumowanie	272
17. Infrastruktura chmurowa	273
Terminologia	276
Instancja	276
Obraz AMI	277
Etykieta	277
Definiowanie poświadczeń	277
Zmienne środowiskowe	278
Pliki konfiguracyjne	279
Wymagania: biblioteka Python Boto3	279
Dynamiczne ewidencjonowanie instancji	280
Buforowanie ewidencji	281
Inne opcje konfiguracyjne	281
Definiowanie dynamicznych grup zasobów za pomocą etykiet	282
Przypisywanie etykiet do istniejących zasobów	282
Czytelne nazwy grup	283
Wirtualne chmury prywatne	284
Przygotowanie pliku ansible.cfg	284
Uruchamianie nowych instancji	285
Pary kluczy EC2	286
Utworzenie nowego klucza	286
Grupy zabezpieczeń	287
Dozwolone adresy IP	288
Porty w grupach zabezpieczeń	288
Uzyskiwanie najnowszego obrazu AMI	288
Utworzenie nowej instancji i dodanie jej do grupy	289
Oczekiwanie na gotowość instancji	291
Wszystko razem	291
Konfiguracja chmury VPC	294
Dynamiczne ewidencjonowanie i chmura VPC	296
Podsumowanie	297
18. Wtyczki zwrotne	298
Wtyczki standardowego wyjścia	298
ARA	299
debug	300
default	300
dense	300
json	301
minimal	301

null	301
oneline	301
Wtyczki powiadomień i agregacji	301
Moduły Pythona	302
foreman	302
jabber	303
junit	303
log_plays	304
logentries	304
logstash	304
mail	305
profile_roles	305
profile_tasks	305
say	306
slack	306
splunk	306
timer	306
Podsumowanie	307
19. Własne moduły	308
Przykład: sprawdzenie, czy zewnętrzny serwer jest dostępny	308
Użycie modułu script zamiast tworzenia własnego modułu	309
Skrypt can_reach jako moduł	310
Czy trzeba tworzyć własne moduły?	310
Gdzie umieszczać własne moduły?	310
Jak Ansible uruchamia moduły?	311
Utworzenie niezależnego skryptu Pythona z argumentami (tylko Python)	311
Skopiowanie modułu do serwera	311
Utworzenie pliku argumentów na serwerze (inne języki)	311
Wywołanie modułu	312
Oczekiwane wyniki	312
Zmienne wynikowe oczekiwane przez Ansible	313
Tworzenie modułów w języku Python	313
Analiza argumentów	315
Odczytywanie argumentów	316
Import klasy pomocniczej AnsibleModule	316
Opcje argumentów	316
Parametry konstruktora klasy AnsibleModule	319
Zwracanie informacji o pomyślnym lub niepomyślnym wykonaniu modułu	322
Wywoływanie zewnętrznych programów	322
Tryb weryfikacji (suchy przebieg)	324
Dokumentowanie modułu	324

Diagnozowanie modułu	326
Implementowanie modułu jako skryptu Bash	327
Określanie alternatywnego położenia powłoki Bash	328
Podsumowanie	329
20. Przyspieszanie Ansible	330
Zwielokrotnienie sesji SSH (opcja ControlPersist)	330
Ręczne włączenie zwielokrotnienia sesji SSH	331
Opcje zwielokrotniania sesji SSH	332
Dodatkowe strojenie sesji SSH	333
Zalecenia dotyczące algorytmów	334
Potokowanie	335
Włączenie potokowania	335
Konfigurowanie potokowania na serwerze	335
Mitogen dla Ansible	337
Zapamiętywanie faktów	338
Zapamiętywanie faktów w plikach JSON	339
Zapamiętywanie faktów w bazie Redis	340
Zapamiętywanie faktów w bazie Memcached	340
Równoległe połączenia	341
Równoległe wykonywanie zadań za pomocą instrukcji async	341
Podsumowanie	343
21. Sieci i bezpieczeństwo	344
Zarządzanie siecią	344
Obsługiwane urządzenia	344
Komunikacja Ansible z urządzeniami sieciowymi	345
Tryb uprzywilejowany	346
Ewidencja sieci	346
Zastosowania automatyzacji operacji sieciowych	347
Bezpieczeństwo	347
Czy trzeba przestrzegać norm?	348
Zabezpieczony, ale nie bezpieczny	349
Szare IT	352
Jasne IT	352
Zero zaufania	353
Podsumowanie	353
22. Procesy CI/CD i Ansible	355
Ciągła integracja oprogramowania	355
Elementy systemu ciągłej integracji oprogramowania	356
Jenkins i Ansible	360
Uruchomienie procesu CI dla ról Ansible	364

Testy	365
Wtyczka Ansible	366
Wtyczka Ansible Tower	367
Podsumowanie	370
23. Ansible Automation Platform	371
Modele subskrypcyjne	374
Wersja próbna platformy Ansible Automation Platform	375
Do czego służy platforma Ansible Automation?	376
Kontrola dostępu	376
Projekty	376
Zarządzanie ewidencją	378
Uruchamianie zadań według szablonów	379
Interfejs REST API	381
Kolekcja awx.awx	383
Instalacja	383
Zdefiniowanie organizacji	384
Utworzenie ewidencji	384
Uruchamianie scenariusza za pomocą szablonu zadania	385
Uruchamianie Ansible za pomocą kontenerów	386
Tworzenie środowisk wykonawczych	387
Podsumowanie	388
24. Dobre praktyki	389
Prostota, modułowość i kompozycyjność	389
Porządkowanie treści	390
Oddzielenie ewidencji od projektów	390
Oddzielenie ról od kolekcji	390
Scenariusze	391
Styl kodu	391
Oznaczanie i testowanie wszystkiego	391
Żądany stan	391
Ciągłe dostarczanie oprogramowania	392
Bezpieczeństwo	392
Wdrażanie	392
Wskaźniki wydajności	393
Ocenianie skuteczności dobrych praktyk	393
Słowo końcowe	394
Bibliografia	395

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.

Przeanalizujemy 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. Wykorzystując 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 wdroić. 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 wire-server 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
<code>docker_host</code>	<code>DOCKER_HOST</code>	<code>unix://var/run/docker.sock</code>
<code>tls_hostname</code>	<code>DOCKER_TLS_HOSTNAME</code>	<code>localhost</code>
<code>api_version</code>	<code>DOCKER_API_VERSION</code>	<code>auto</code>
<code>cert_path</code>	<code>DOCKER_CERT_PATH</code>	<i>Brak</i>
<code>ssl_version</code>	<code>DOCKER_SSL_VERSION</code>	<i>Brak</i>
<code>tls</code>	<code>DOCKER_TLS</code>	<code>no</code>
<code>tls_verify</code>	<code>DOCKER_TLS_VERIFY</code>	<code>no</code>
<code>timeout</code>	<code>DOCKER_TIMEOUT</code>	<code>60 (sekund)</code>

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 ECDH+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
```

Dobłą 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
```

Wysyłanie 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"
}
```

Użyjaj 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.

PROGRAM PARTNERSKI

— GRUPY 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
Helion 

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 Netflixie, 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. Kościuski 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-8322-152-6



Cena: 99,00 zł