Wydanie II

Helion

Docker

Wydajność i optymalizacja pracy aplikacji

Packt›

Allan Espinosa, Russ McKendrick

Tytuł oryginału: Docker High Performance: Complete your Docker journey by optimizing your application's workflows and performance, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-6232-1

Helion SA
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)

Spis treści

ROZDZIAŁ 5

Wdrażanie kontenerów

W tym rozdziale wykorzystasz działający w kontenerze Dockera serwer Jenkins do przeprowadzania kompilacji aplikacji, przekazywania jej do serwisu Docker Hub, a następnie wdrażania w klastrze Docker Swarm.

W rozdziale zostaną poruszone następujące zagadnienia:

- przygotowanie własnego obrazu z serwerem Jenkins;
- wdrażanie serwera Jenkins w klastrze Docker Swarm;
- konfigurowanie serwera Jenkins do współpracy z klastrem Docker Swarm i kontem w Docker Hub;
- kompilowanie i wdrażanie prostej aplikacji za pomocą trzyetapowego procesu.

Wdrażanie i konfigurowanie serwera Jenkins

Jenkins powstał jako serwer kompilacji o nazwie Hudson (nazwa została zmieniona ze względu na spór z firmą Oracle), którego jedynym przeznaczeniem była kompilacja aplikacji Javy. Od chwili pierwszego wydania w 2004 r. rola serwera była nieustannie rozszerzana. Obecnie jest on uznawany za jedno z najważniejszych rozwiązań w zakresie ciągłej integracji (ang. *continous integration*) i ciągłego dostarczania oprogramowania (ang. *continous delivery*), a nie tylko jako serwer kompilacji dla aplikacji Javy.

Zanim będziesz mógł przystąpić do wdrożenia aplikacji testowej, najpierw musisz zająć się wdrożeniem i skonfigurowaniem serwera Jenkins. Przedstawione tutaj kroki pokazują procedurę wdrożenia i skonfigurowania pełnego serwera Jenkins.

Wdrażanie kontenera Jenkins

Zanim uruchomisz i skonfigurujesz kontener Jenkins w klastrze Docker Swarm, najpierw musisz utworzyć obraz. W trakcie tej operacji jako podstawę wykorzystasz oficjalny obraz Jenkins, a następnie wprowadzisz własne usprawnienia.

1. Pracę zacznij od przygotowania pliku konfiguracyjnego Jenkinsa — powinien on nosić nazwę *init.groovy.d/plugins.groovy*. Ten plik zostanie przekazany do kontenera po jego pierwszym uruchomieniu i nakaże serwerowi Jenkins zainstalowanie kilku wymaganych dodatków. Spójrz na zawartość tego pliku.

```
import jenkins.model.Jenkins;

pm = Jenkins.instance.pluginManager
uc = Jenkins.instance.updateCenter
uc.updateAllSites()

installed = false

["git", "workflow-aggregator"].each {
  if (! pm.getPlugin(it)) {
    deployment = uc.getPlugin(it).deploy(true)
    deployment.get()
    installed = true
  }
}

if (installed) {
  Jenkins.instance.restart()
}
```

2. Mając gotowy plik *init.groovy.d/plugins.groovy*, można przejść do rzeczywistego obrazu Dockera. Następnym krokiem jest utworzenie pliku *Dockerfile*. Jak widać w przedstawionym tutaj fragmencie kodu, zdecydowałem się na wykorzystanie wersji LTS obrazu serwera Jenkins, który znajdziesz na stronie *https://hub.docker.com/r/jenkins/jenkins*.

Kolejnym interesującym aspektem jest zainstalowanie w obrazie samego Dockera. Takie rozwiązanie pozwala zapewnić serwerowi Jenkins możliwość współpracy z klastrem Docker Swarm. Odpowiednią konfigurację przedstawię w dalszej części rozdziału.

```
FROM jenkins/jenkins:lts

ARG docker_version=18.09.5
ARG
docker_tarball=https://download.docker.com/linux/static/stable/x86_
64/docker-$docker_version.tgz

LABEL com.docker.version=$docker_version

USER root
RUN cd /usr/local/bin && \
    curl -s $docker_tarball | tar xz docker/docker --strip-
components=1

USER jenkins
COPY init.groovy.d/plugins.groovy \
     /usr/share/jenkins/ref/init.groovy.d/plugins.groovy
```

3. Gdy niezbędne pliki konfiguracyjne znajdą się na swoich miejscach, można przystąpić do utworzenia obrazu Dockera z serwerem Jenkins i umieszczenia go w Docker Hub, co wymaga wydania przedstawionych tutaj poleceń. Pamiętaj o zastąpieniu `hubuser` własną nazwą użytkownika w Docker Hub.

```
$ docker build -t hubuser/jenkins:latest .
$ docker push hubuser/jenkins:latest
```

4. Zanim będziesz uruchamiać kontenery Jenkins, konieczne jest przygotowanie definicji usługi w pliku typu *Compose*. Utwórz więc plik o nazwie *docker-compose.yml* i umieść w nim przedstawiony niżej fragment kodu. Także tutaj upewnij się o zastąpieniu `hubuser` własną nazwą użytkownika w Docker Hub.

```
version: '3.7'

services:
  jenkins:
    image: hubuser/jenkins:latest
    ports:
      - '8080:8080'
    deploy:
      replicas: 1
```

5. Teraz powinno być już możliwe wdrożenie serwera Jenkins za pomocą definicji usługi przygotowanej w poprzednim kroku. Wydaj następujące polecenie:

   ```
   $ docker stack deploy jenkins --compose-file docker-compose.yml
   ```

   Ponieważ nakazałeś serwerowi Jenkins przeprowadzenie konfiguracji początkowej na podstawie pliku *init.groovy.d/plugins.groovy* po pierwszym uruchomieniu, to pierwsze uruchomienie może zabrać nieco czasu, w trakcie którego są pobierane dodatkowe wtyczki. Stan operacji zawsze możesz sprawdzić za pomocą przedstawionych tutaj poleceń.

   ```
   $ docker stack ls
   $ docker stack services Jenkins
   ```

Gdy zobaczysz komunikat o replice 1/1, oznacza to, że kontener Jenkins jest gotowy. Wówczas można przystąpić do ukończenia procesu instalacji za pomocą przeglądarki WWW.

Dokończenie konfigurowania serwera Jenkins

W przeglądarce WWW przejdź pod adres *http://dockerhost:8080* (pamiętaj o zastąpieniu `dockerhost` adresem IP hosta Docker Swarm). Zostaniesz powitany stroną podobną do pokazanej na rysunku 5.1.

Rysunek 5.1. Strona początkowa podczas konfiguracji serwera Jenkins

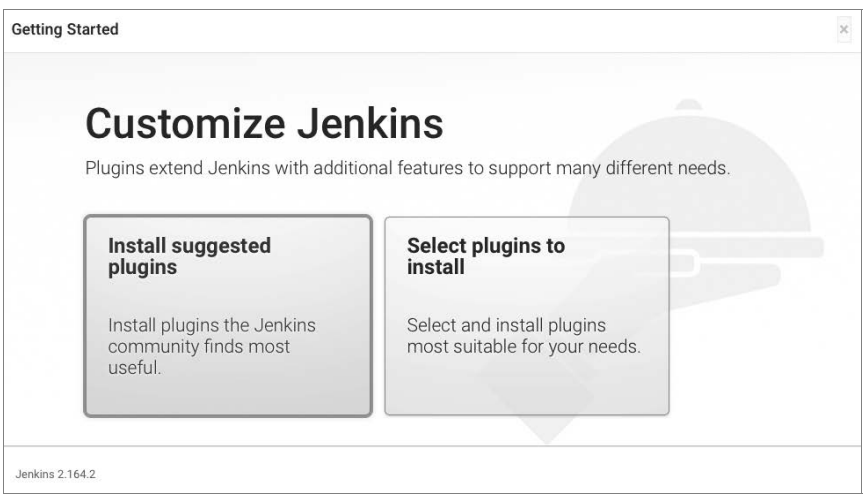
Jak widać, jednym z pierwszych zadań wykonywanych przez serwer Jenkins jest zabezpieczenie się. Istnieje ku temu ważny powód — jeżeli egzemplarz Jenkinsa byłby dostępny publicznie, nie chciałbyś używać domyślnie skonfigurowanej nazwy użytkownika i hasła, ponieważ istnieją boty, których celem jest włamywanie się do nieskonfigurowanych instalacji Jenkinsa.

Aby ustalić hasło administratora, czyli 32-elementowy alfanumeryczny ciąg tekstowy, należy po prostu wydać dwa polecenia.

```
$ container=jenkins_jenkins.1.$(docker service ps jenkins_jenkins -q)
$ docker exec $container cat /var/jenkins_home/secrets/
↪initialAdminPassword
```

Pierwsze z nich powoduje wygenerowanie pełnej nazwy kontenera, natomiast drugie używa polecenia `docker exec` do wyświetlenia na ekranie zawartości pliku */var/jenkins_home/secrets/initialAdminPassword* zawierającego hasło.

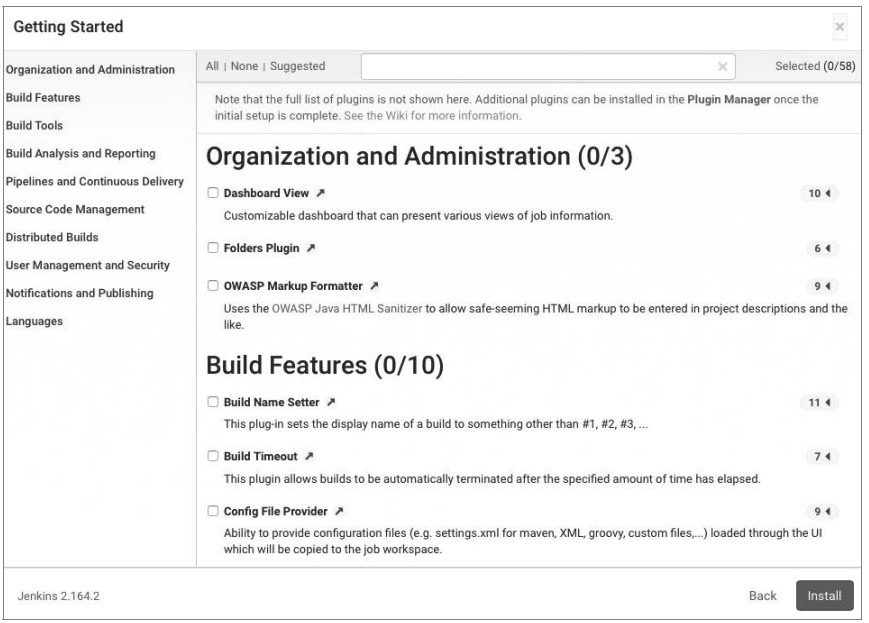
Wprowadź dane uwierzytelniające w odpowiednich polach, a następnie kliknij przycisk *Continue*. Znajdziesz się na kolejnej stronie z dwiema możliwościami (rysunek 5.2).



Rysunek 5.2. Wybór sposobu instalacji wtyczek

Na tej stronie kliknij przycisk *Select plugins to install*. Na ekranie zostanie wyświetlona następna strona z wieloma opcjami. Ponieważ podczas pierwszego uruchomienia serwera Jenkins zostały zainstalowane niezbędne wtyczki, najpierw

kliknij przycisk *None* na początku strony, a później przycisk *Install*, jak pokazałem na rysunku 5.3.



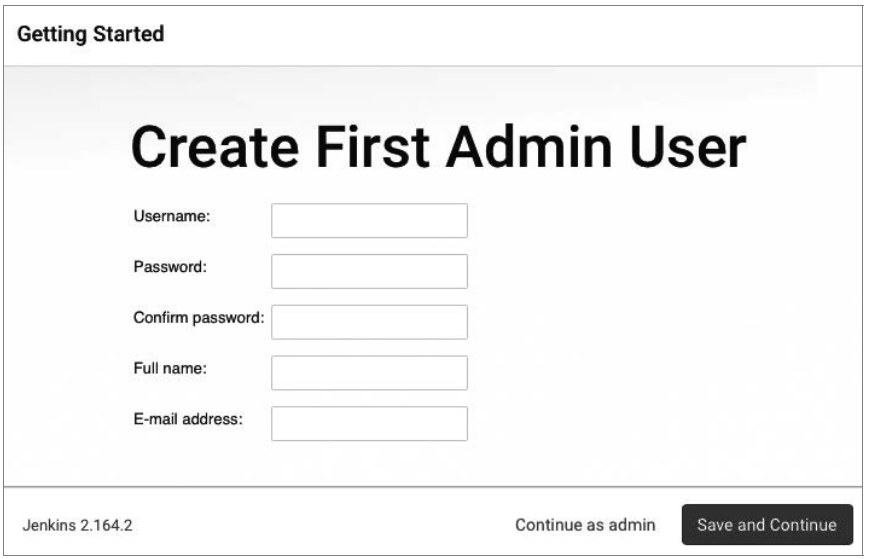
Rysunek 5.3. Wtyczki dostępne do zainstalowania

Jeśli nie zaznaczyłeś niczego do zainstalowania, zostaniesz przeniesiony na stronę pozwalającą utworzyć konto użytkownika z uprawnieniami administratora (rysunek 5.4). Wpisz preferowaną nazwę użytkownika i hasło; ewentualnie możesz kontynuować pracę z nazwą użytkownika i hasłem wygenerowanymi w trakcie pierwszego uruchomienia serwera Jenkins.

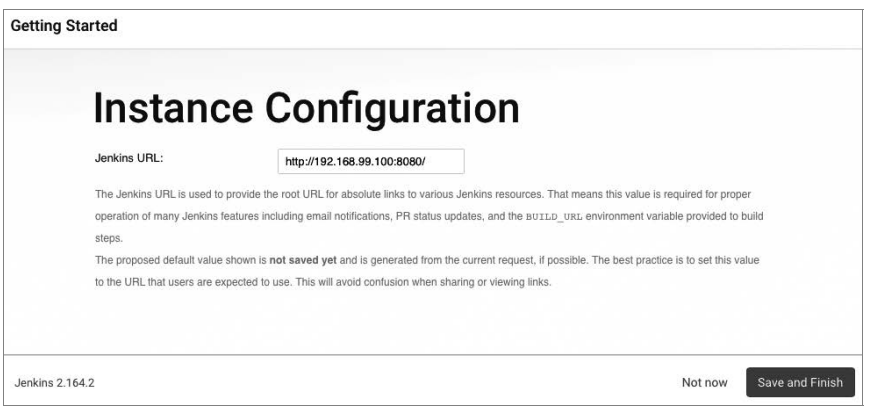
Ostatnim krokiem podczas konfigurowania serwera Jenkins jest potwierdzenie adresu URL serwera. To adres, którego aktualnie użyłeś w celu uzyskania dostępu do serwera Jenkins. Możesz więc kliknąć przycisk *Save and Finish*, jak pokazałem na rysunku 5.5, co spowoduje zakończenie konfiguracji.

W ten sposób zakończyłeś konfigurację serwera Jenkins, co potwierdza komunikat wyświetlony na ekranie (rysunek 5.6). Możesz już używać serwera Jenkins.

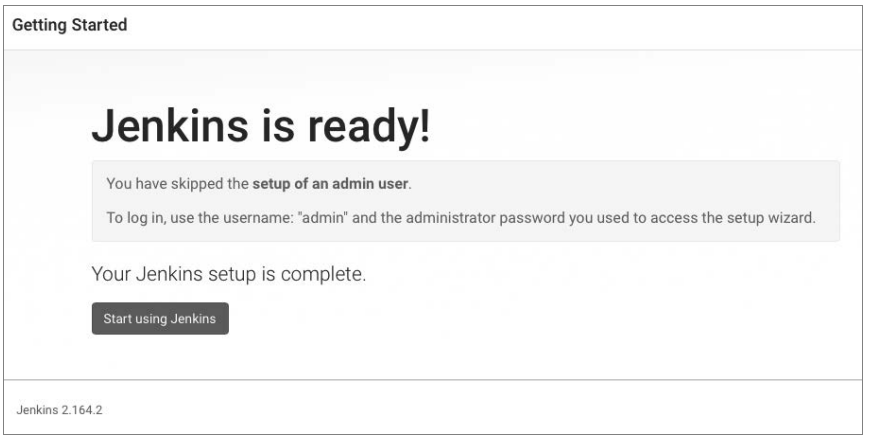
Skoro Jenkins jest dostępny, należy o tym fakcie poinformować jeszcze klaster Docker Swarm.



Rysunek 5.4. Strona pozwalająca na utworzenie konta użytkownika z uprawnieniami administratora



Rysunek 5.5. Adres URL serwera Jenkins



Rysunek 5.6. Komunikat potwierdzający zakończenie konfigurowania serwera Jenkins

Definiowanie danych uwierzytelniających Dockera za pomocą serwera Jenkins

Istnieje kilka różnych kroków zapewniających dostęp do zarówno naszego klastra Docker Swarm, jak i konta serwisu Docker Hub, więc wdrożony wcześniej kontener Jenkins może tworzyć, przekazywać i wdrażać kontenery.

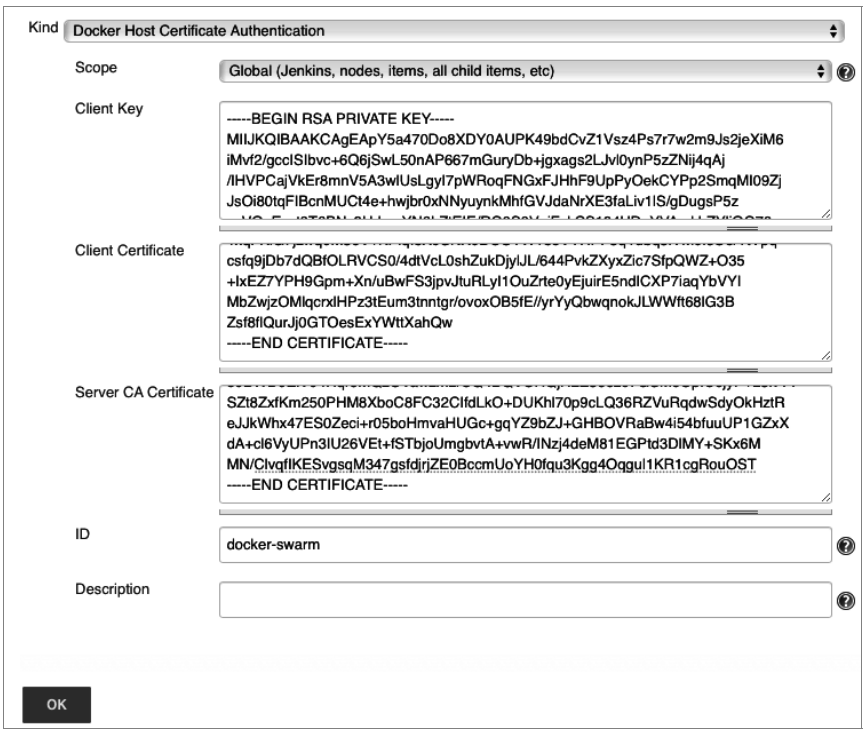
Należy zacząć od dostarczenia certyfikatów wymaganych w celu uzyskania dostępu do klastra Docker Swarm. Wprowadzenie tych danych uwierzytelniających jest możliwe przez przejście pod adres *http://dockerhost:8080/credentials/store/system/domain/_/newCredentials* (nie zapomnij o uaktualnieniu adresu URL zgodnie z potrzebą) bądź też przez kliknięcie *Credentials/System/Global credentials (unrestricted)* i później kliknięcie *Add Credentials*.

W wyświetlonym formularzu z rozwijanego menu *Kind* wybierz opcję *Docker Host Certificate Authentication*, co spowoduje zmianę formularza pozwalającego teraz na podanie kolejnych szczegółów.

1. **Client Key.** W tym polu wklej zawartość pliku *~/.docker/key.pem* z hosta Dockera.
2. **Client Certificate.** W tym polu wklej zawartość pliku *~/.docker/cert.pem* z hosta Dockera.

3. **Server CA Certificate.** W tym polu wklej zawartość pliku *~/.docker/ca.pem* z hosta Dockera.
4. **ID.** Wpisz `docker-swarm`.
5. Kliknij przycisk *OK*.

Po wprowadzeniu niezbędnych informacji formularz powinien wyglądać jak na rysunku 5.7.



Rysunek 5.7. Wypełniony formularz definiowania danych uwierzytelniających w serwerze Jenkins

Zachowaj odwołanie do tych danych uwierzytelniających `docker-swarm`, ponieważ będą jeszcze wykorzystywane. Następnie serwerowi Jenkins trzeba przekazać dane uwierzytelniające do konta Docker Hub. W tym celu ponownie kliknij *Add Credentials*. Tym razem w rozwijanym menu *Kind* pozostaw opcję domyślną, *Username with Password*.

1. **Username.** W tym miejscu podaj nazwę użytkownika w Docker Hub, np. `hubuser`.
2. **Password.** W tym miejscu podaj hasło do konta Docker Hub.
3. **Id.** Wpisz `dockerhub` — do tej nazwy będziesz się później odwoływać.

Po dołączeniu do instalacji Jenkinsa danych uwierzytelniających można przygotować nowy potok Jenkinsa, który wykorzysta te dane do utworzenia aplikacji testowej.

Utworzenie i wdrożenie kontenera

Po skonfigurowaniu serwera Jenkins trzeba zdecydować się na aplikację, która zostanie skonfigurowana i wdrożona.

Przygotowanie aplikacji

Przypomnij sobie, jak w rozdziale 4. utworzyłeś prostą aplikację w języku Go, której zadaniem było wyświetlenie komunikatu `Witaj, świecie!` w przeglądarce WWW. Plik *Dockerfile* definiujący wieloetapowy proces tworzenia obrazu przedstawiał się następująco:

```
FROM golang:1.11-stretch

ADD hello.go hello.go
RUN go build hello.go

FROM busybox
COPY --from=0 /go/hello /app/hello
COPY --from=0 /lib/x86_64-linux-gnu/libpthread.so.0 \
              /lib/x86_64-linux-gnu/libpthread.so.0
COPY --from=0 /lib/x86_64-linux-gnu/libc.so.6 /lib/x86_64-linux-
gnu/libc.so.6
COPY --from=0 /lib64/ld-linux-x86-64.so.2 /lib64/ld-linux-x86-64.so.2
WORKDIR /app
EXPOSE 8080
ENTRYPOINT ["./hello"]
```

Z kolei plik *hello.go* zawierał przedstawiony tutaj fragment kodu.

```
package main

import (
  "fmt"
  "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "Witaj, świecie!")
}

func main() {
  http.HandleFunc("/", handler)
  http.ListenAndServe(":8080", nil)
}
```

Poza dwoma wymienionymi plikami, konieczne jest dodanie także kolejnego o nazwie *Jenkinsfile*. Na podstawie jego nazwy można się domyślić, że jest podobny do *Dockerfile* i definiuje sposób, w jaki serwer Jenkins powinien skompilować aplikację. Plik *Jenkinsfile* dla naszej aplikacji określa trzy etapy jej kompilowania.

1. **Utworzenie obrazu Dockera.** Na tym etapie następuje utworzenie obrazu lokalnie na podstawie pliku *Dockerfile* w aplikacji.
2. **Przekazanie obrazu do Docker Hub.** W trakcie tego etapu następuje przekazanie do usługi Docker Hub obrazu utworzonego na poprzednim etapie.
3. **Wdrożenie obrazu w środowisku produkcyjnym.** Ostatni etap polega na wdrożeniu aplikacji za pomocą obrazu przekazanego do Docker Hub.

```
node {
    def app
    stage('Utworzenie obrazu Dockera') {
        checkout scm
        docker.withServer('tcp://dockerhost:2376', 'docker-swarm')
{
            app = docker.build('hubuser/jenkins-app:latest')
        }
    }
    stage('Przekazanie obrazu do Docker Hub') {
        docker.withServer('tcp://dockerhost:2376', 'docker-swarm')
{
            docker.withRegistry("https://index.docker.io/v1/",
"dockerhub") {
```

```
                app.push('latest')
            }
        }
    }
    stage('Wdrożenie obrazu w środowisku produkcyjnym') {
        docker.withServer('tcp://dockerhost:2376', 'docker-swarm')
{
            sh "docker service update app --force --image
${app.id}"
        }
    }
}
```

Do rozpowszechniania konfiguracji aplikacji zostanie użyty system kontroli wersji Git. Ta konfiguracja składa się z trzech plików: *Dockerfile*, *hello.go* i *Jenkinsfile*. Zachęcam do sklonowania istniejącego repozytorium, które znajduje się w serwisie GitHub na stronie *https://github.com/dockerhp/jenkins-app* i jest dostępne publicznie.

Jak widać, w pliku *Jenkinsfile* znajdują się odwołania do dwóch zbiorów danych uwierzytelniających niezbędnych do współpracy z klastrem Docker Swarm. Dane uwierzytelniające `docker-swarm` są używane w celu uzyskania dostępu do klastra, natomiast `dockerhub` — podczas logowania do konta Docker Hub. Dlatego też choć plik *Jenkinsfile* jest dostępny publicznie, tak naprawdę informacje wrażliwe — np. certyfikaty lub dane pozwalające na zalogowanie się do Docker Hub — nigdy nie są udostępniane, a wszystko jest przechowywane bezpiecznie w instalacji serwera Jenkins.

Zanim przejdziesz dalej, konieczne jest przeprowadzenie edycji pliku *Jenkinsfile* i upewnienie się, że wszystkie odwołania do *tcp://dockerhost:2376* zostały uaktualnione i odzwierciedlają adres IP oraz nazwę hosta Dockera. Nie zapomnij o zastąpieniu `hubuser` odpowiednią nazwą użytkownika Docker Hub. Ponadto upewnij się o przekazaniu pliku *Jenkinsfile* do repozytorium Git.

Po wprowadzeniu niezbędnych uaktualnień można przystąpić do wdrożenia usługi, która ostatecznie będzie odpowiedzialna za hosting aplikacji. Wymaga to wydania następującego polecenia:

```
$ docker service create --name app --publish 80:8080 nginx
```

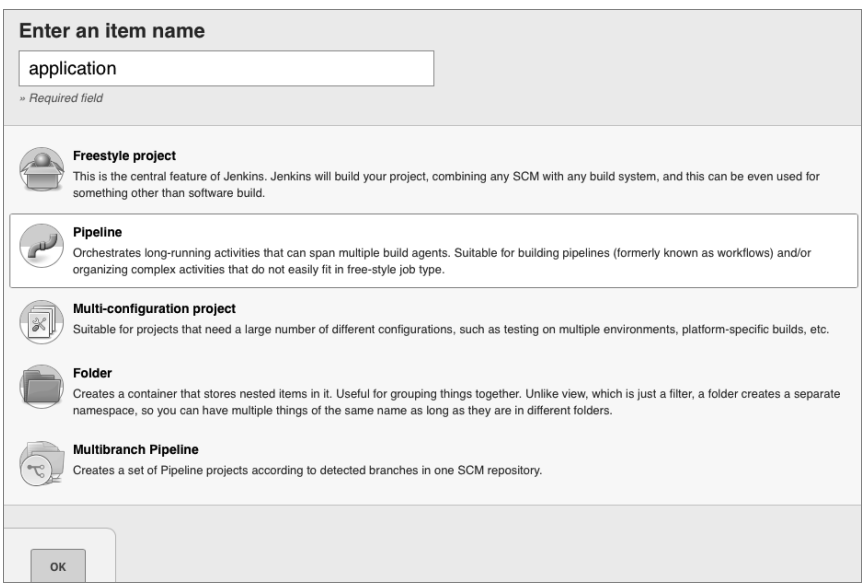
Jak pewnie zauważyłeś, nastąpiło wdrożenie oficjalnego obrazu NGINX, a nie przykładowej aplikacji. Nie przejmuj się tym, ponieważ ostatni etap operacji spowoduje zastąpienie tego obrazu odpowiednim. Wdrożenie można szybko przetestować przez wydanie przedstawionego tutaj polecenia.

```
$ curl http://dockerhost
curl: (7) Failed connect to dockerhost:80; Connection refused
```

Próba zakończyła się niepowodzeniem. Tego można się było spodziewać, ponieważ tak naprawdę jeszcze nie została przeprowadzona odpowiednia konfiguracja NGINX. W następnej sekcji zajmiesz się utworzeniem zadania Jenkinsa, które użyje potoku zdefiniowanego w pliku *Jenkinsfile* do ostatecznego wdrożenia aplikacji.

Utworzenie zadania Jenkinsa

Dotarłeś do ostatniego etapu wdrożenia. Wszystkie przygotowane wcześniej fragmenty zostaną połączone w jedną całość — powstanie potok odpowiedzialny za utworzenie i wdrożenie aplikacji. Zacznij od zdefiniowania nowego zadania — w tym celu wystarczy przejść pod adres *http://dockerhost:8080/newJob*. W przeglądarce WWW zostanie wyświetlonych pięć różnych typów zadań Jenkinsa, jak pokazałem na rysunku 5.8.



Rysunek 5.8. Pięć różnych typów zadań możliwych do utworzenia w serwerze Jenkins

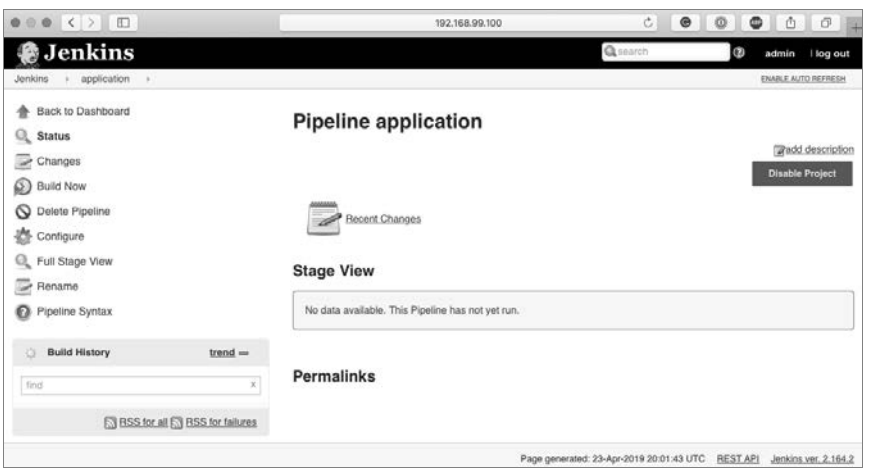
W polu *Enter an item name* wpisz **application**, zaznacz zadanie typu *Pipeline* i kliknij przycisk *OK*.

Na wyświetlonej stronie będzie dostępnych wiele opcji. Przewiń sekcje *General*, *Build Triggers* oraz *Advanced Project Options* i przejdź prosto do *Pipeline*.

Domyślnie w definicji jest wybrana opcja *Pipeline script*. Skoro nasz potok został zdefiniowany w pliku *Jenkinsfile* przechowywanym w repozytorium Git, zmień opcję na *Pipeline script from SCM*. Spowoduje to uaktualnienie formularza: w rozwijanym menu *SCM* będzie wybrana opcja *None*. Zmień ją na *Git*.

W polu *Repository URL* wpisz adres HTTPS URL repozytorium Git, np. *https://github.com/dockerhp/jenkins-app.git*. Wszystkim pozostałym polom pozostaw wartości domyślne i kliknij przycisk *Save*.

Powinieneś znaleźć się na stronie pokazanej na rysunku 5.9.



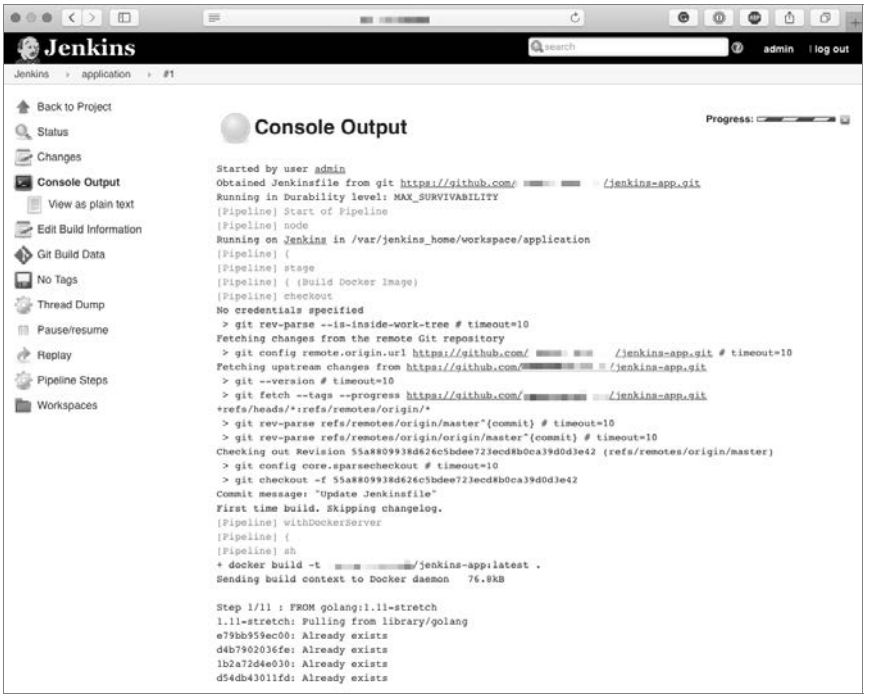
Rysunek 5.9. Tworzenie potoku aplikacji

Pozostało już tylko kliknięcie przycisku *Build Now*.

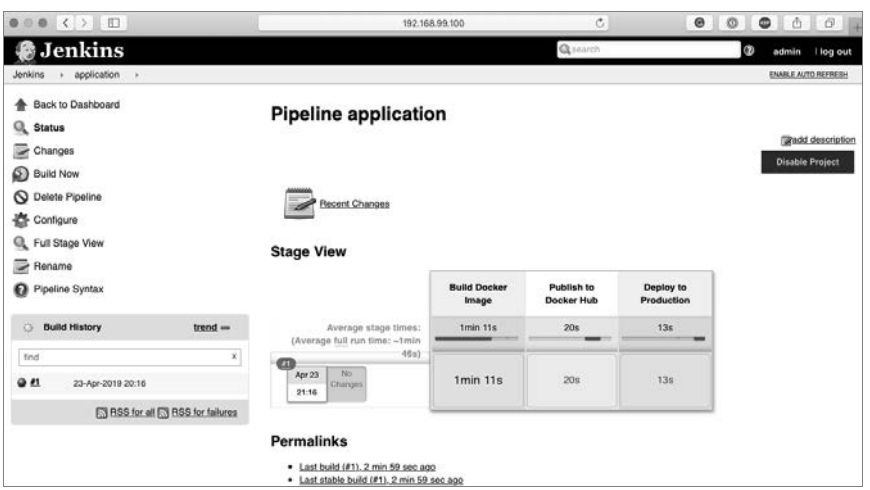
Uruchamianie potoku

Po kliknięciu przycisku *Build Now* powinieneś zobaczyć zadanie wyświetlone na dole menu po lewej stronie. Kliknięcie numeru zadania, a następnie *Console Output* dostarcza generowanych w czasie rzeczywistym informacji na temat wykonywanego zadania. Będą to dane podobne do pokazanych na rysunku 5.10.

Po zakończeniu zostanie wyświetlona strona z podsumowaniem, którą pokazałem na rysunku 5.11.



Rysunek 5.10. Informacje generowane podczas wykonywania zadania przez serwer Jenkins



Rysunek 5.11. Strona wyświetlana po zakończeniu wykonywania zadania przez Jenkins

Jak widać w podsumowaniu, wszystkie trzy etapy zakończyły się sukcesem. Jeżeli z jakiegokolwiek powodu operacja zakończy się niepowodzeniem, możesz przejrzeć komunikaty generowane w *Console Output*. Zachęcam do prawidłowego uaktualnienia w pliku *Jenkinsfile* adresu URL Dockera i nazwy użytkownika Docker Hub. Jeżeli zachodzi potrzeba ponownego przeprowadzenia operacji, upewnij się o przekazaniu zmian do repozytorium i później do serwisu GitHub, a następnie ponownie kliknij przycisk *Build Now*.

Po wdrożeniu aplikacji wydanie przedstawionego tutaj polecenia powinno spowodować wyświetlenie komunikatu `Witaj, świecie!`

```
$ curl http://dockerhost
Witaj, świecie!
```

Wprawdzie to naprawdę prosty przykład, ale wyraźnie pokazuje, że takie podejście może być bardzo korzystne. Bez większego wysiłku można skonfigurować w serwerze Jenkins zaczep sieciowy obserwujący repozytorium GitHub. Po wykryciu jakiejkolwiek zmiany nastąpi automatyczne wykonanie potoku, który z kolei utworzy obraz, przekaże go do serwisu Docker Hub, a następnie wdroży kontener w środowisku produkcyjnym.

Podsumowanie

W tym rozdziale pokazałem, jak używać serwera Jenkins do tworzenia, rozprowadzania i wdrażania aplikacji w postaci kontenera. Więcej informacji na temat serwera Jenkins znajdziesz w witrynie internetowej projektu pod adresem *https://jenkins.io/*.

W następnym rozdziale zaczniesz przeprowadzać testy wydajności pozwalające sprawdzić, jak wdrożona aplikacja Dockera zachowuje się pod obciążeniem.


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

Wydajny. Wydajniejszy. Docker.

Docker to technologia, dzięki której można uruchamiać kod w wielu środowiskach i na różnych platformach. Sposób działania tego oprogramowania bardzo upraszcza opracowywanie, testowanie, wdrażanie i skalowanie aplikacji. Docker wciąż jest sukcesywnie rozwijany. Zyskał znakomitą stabilność, a zestaw udostępnianych programistom narzędzi stale rośnie. Twórcy dużych systemów, zwłaszcza rozproszonych, coraz bardziej doceniają jego potencjał. Szczególnie atrakcyjny jest dla tych projektantów, którzy stawiają na konteneryzację i automatyzację przepływu pracy i ciągle poszukują sposobów optymalizowania działania swoich aplikacji.

Ta książka to znakomity przewodnik, dzięki któremu szybko poprawisz wydajność swoich aplikacji Dockera. Wyjaśniono tu zasady dostrajania plików Dockerfile, pokazano praktyczne techniki wdrażania kontenerów Dockera, przedstawiono także informacje o monitorowaniu wydajności kontenerów oraz o pracy z dziennikami zdarzeń hostów za pomocą stosu ELK. Z książki dowiesz się również, w jaki sposób standardowe narzędzia Linuksa umożliwiają diagnozowanie i rozwiązywanie problemów związanych z kontenerami. Nie zabrakło bardzo przydatnych wskazówek odnoszących się do przygotowania aplikacji do wdrożenia w środowiskach produkcyjnych z wykorzystaniem najefektywniejszych technik DevOps.

W tej książce między innymi:

- przygotowanie Dockera i jego konfigurowanie za pomocą programu Chef
- monitorowanie Dockera za pomocą systemu Prometheus
- sprawne wdrażanie aplikacji i testy wydajności
- skalowanie aplikacji Dockera
- debugowanie kontenerów

Allan Espinosa — aktywnie rozwija narzędzia systemów rozproszonych, takie jak Docker i Chef. Zajmuje się obsługą wielu obrazów Dockera wraz z popularnym oprogramowaniem *open source*. Tworzył skalowalne aplikacje na wiele platform. Obecnie pracuje w firmie Bloomberg, gdzie odpowiada za infrastrukturę Hadoop.

Russ McKendrick — jest doświadczonym administratorem systemów. Od pewnego czasu wspiera rozwój narzędzi i systemów *open source* w publicznych i prywatnych chmurach N4Stack. W wolnym czasie zajmuje się pisaniem książek.

ISBN 978-83-283-6232-1
9 788328 362321
Cena: 39,90 zł

Packt›