Adriaan de Jonge

ADDISON-WESLEY

Google App Engine

Tworzenie wydajnych aplikacji w Javie

Od projektu do wdrożenia!

Helion

Tytuł oryginału: Essential App Engine:
Building High-Performance Java Apps with Google App Engine

Tłumaczenie: Justyna Walkowska

ISBN: 978-83-246-4689-0

Authorized translation from the English language edition, entitled: ESSENTIAL APP ENGINE: BUILDING HIGH-PERFORMANCE JAVA APPS WITH GOOGLE APP ENGINE; ISBN 032174263X; by Adriaan De Jonge; published by Pearson Education, Inc, publishing as Addison Wesley.

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

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

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

Printed in Poland.

Spis treści

3

Anatomia aplikacji Google App Engine

W tym rozdziale objaśniam strukturę typowej aplikacji App Engine, przedstawiam sposób organizacji plików przed ich wdrożeniem oraz wskazuję miejsce, w którym pliki są trzymane po wdrożeniu. Początek rozdziału to przegląd serwerów, na których działa aplikacja. W następnej kolejności krótko przedstawiam pliki wchodzące w skład pakietu wdrożeniowego ze szczególnym uwzględnieniem każdego z plików konfiguracyjnych. Rozdział kończy się opisem parametrów konfiguracyjnych ustawianych w panelu administracyjnym.

Przesyłanie plików w celu wykonania dynamicznego wdrożenia

Kiedy wysyłasz aplikację na serwery Google, nie jest ona od razu wdrażana jako działająca instancja aplikacji. Zamiast tego aplikacja jest zapisywana na serwerze, z którego może zostać dynamicznie wdrożona, kiedy zajdzie taka potrzeba.

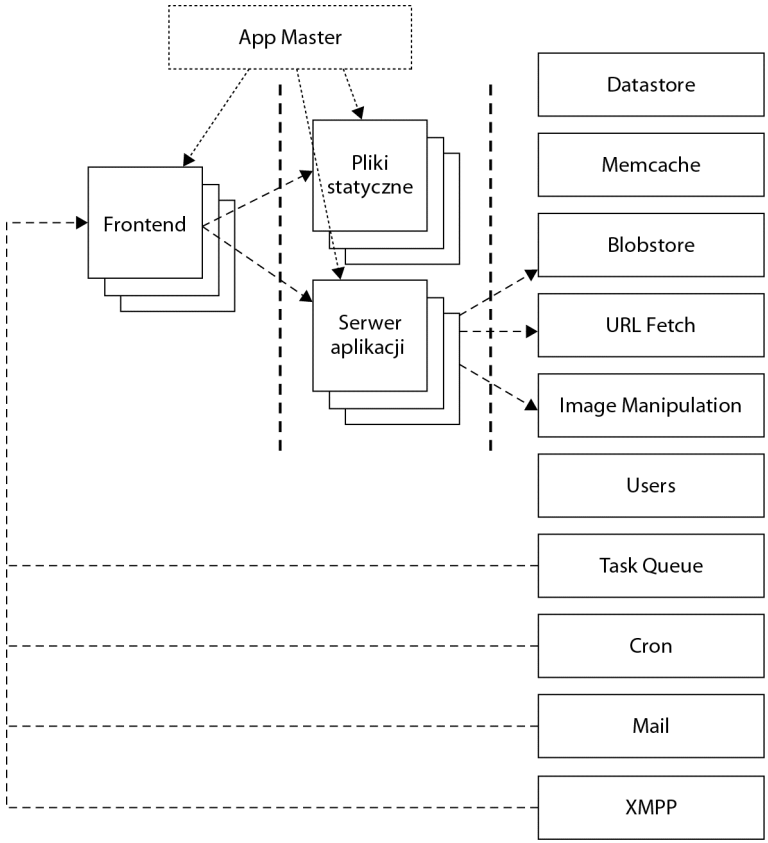
Potrzeba wdrożenia zachodzi wtedy, gdy jeden z serwerów typu *frontend* odbiera żądanie. W tym kontekście frontend to serwer WWW odbierający żądania przed ich przekazaniem do instancji serwera aplikacji.

Na podstawie danych zapisanych w App Masterze frontend podejmuje decyzję, czy żądanie należy przekazać do serwera statycznych plików, czy do jednego z serwerów aplikacji. Serwery uczestniczące w tym procesie zostały przedstawione na rysunku 3.1.

Google skaluje aplikacje, dynamicznie uruchamiając i zatrzymując serwery aplikacji w zależności od aktualnej listy żądań. Aplikacje reagują nie tylko na ilość obciążenia, ale także na jego pochodzenie.

Jeśli na przykład aplikacja zyska popularność w Japonii, Google uruchomi maszyny w (lub niedaleko od) Japonii. Jeśli większość żądań pochodzi z Niemiec, zostaną uruchomione tamtejsze serwery.

Myśląc o wydajności aplikacji App Engine, pamiętaj, że większość usług Google działa na zdalnych serwerach. Wywołanie tych usług zawsze obejmuje pewien narzut komunikacyjny.



Rysunek 3.1. Przegląd serwerów i usług App Engine

Na podstawie rysunku 3.1 łatwo zauważyć, że architektura App Engine została oparta o filozofię, zgodnie z którą *wszystko jest żądaniem HTTP*. Wszystkie żądania do instancji App Engine są przesyłane za pośrednictwem HTTP.

Simple Mail Transfer Protocol (SMTP), Extensible Messaging and Presence Protocol (XMPP), rozkazy Task Queue i zadania cron — wszystkie te elementy są tłumaczone na żądania HTTP przekierowywane przez frontend. Wykorzystanie HTTP ułatwia zarówno obsługę żądań, jak i skalowanie aplikacji App Engine. Skalowanie pod kątem dużej liczby przychodzących wiadomości e-mail przebiega dokładnie tak samo jak skalowanie pod kątem dużej liczby odwiedzin.

Struktura katalogów

Jeśli Twoja aplikacja ma zostać zainstalowana na serwerach App Engine, musisz jej nadać strukturę przypominającą strukturę archiwum WAR, stosowanego podczas wdrażania na takich serwerach aplikacji jak Tomcat czy Jetty. Pamiętaj jednak, że App Engine nie przyjmuje plików WAR, a poza tym archiwum musi zawierać pewne dodatkowe pliki konfiguracyjne.

Zamiast pliku WAR narzędzie wdrożeniowe App Engine oczekuje folderu z rozszerzoną wersją pliku WAR. Po kompilacji typowa aplikacja ma strukturę wyglądającą mniej więcej tak:

```
/moja-aplikacja/
--/wszelkie-bezpośrednio-dostępne-pliki
--/WEB-INF
----/appengine-generated/
------/datastore-indexes-auto.xml
----/classes/ (*)
------/najwyższy-poziom-twoich-pakietów/
--------/itd./
----/lib/ (*)
------/wymagany-jar-jeden.jar
------/wymagany-jar-dwa.jar
------/appengine-api-1.5.1.jar
------/appengine-api-labs-1.5.1.jar
----/wszelkie-ukryte-pliki
----/appengine-web.xml (*)
----/cron.xml
----/datastore-indexes.xml
----/dos.xml
----/logging.properties
----/queue.xml
----/web.xml (*)
```

Pliki oznaczone gwiazdką (*) są niezbędne do działania aplikacji. **Pogrubienie** oznacza, że dany plik w rzeczywistej aplikacji ma dokładnie taką samą nazwę jak na tym listingu. Jeśli nazwa pliku nie została pogrubiona, oznacza to, że została użyta jedynie jako przykład.

Uwaga

W rzeczywistości katalogi */classes/* i */lib/* nie są konieczne do uruchomienia aplikacji. Mógłbyś na przykład stworzyć aplikację internetową zawierającą pojedynczy plik JSP niekorzystający z żadnych usług Google. Jednak w większości standardowych aplikacji katalogi te są wymagane.

Mógłbyś także spakować pliki do własnego archiwum JAR i umieścić je w katalogu */lib/*. Istnienie osobnych katalogów */classes/* i */lib/* pozwala jednak na eleganckie rozdzielenie właściwego kodu programu od zewnętrznych bibliotek.

Kiedy porównasz listę bibliotek widoczną w zestawieniu z listą standardowo dołączaną przez narzędzia programistyczne Google, może Ci się ona wydać dosyć krótka. Być może zdążyłeś się już przyzwyczaić do tego, że lista zawiera biblioteki Java Data Objects (JDO), Java Persistence API (JPA), DataNucleus, Java Specification Request (JSR) 107 i być może parę innych. Jeśli jednak nie korzystasz z tych bibliotek, nie musisz dołączać ich do aplikacji. W tej książce nie zachęcam do używania tych dodatkowych bibliotek, dlatego nie będą one dołączane do aplikacji.

Gdy korzystasz z narzędzi programistycznych Google, takich jak plug-in do Eclipse, pozbycie się wspomnianych bibliotek może być nie lada wyzwaniem. Możesz je skasować, ale one i tak co jakiś czas będą do Ciebie wracały. Trwałym rozwiązaniem tego problemu jest rezygnacja z plug-inu i korzystanie z narzędzi konsolowych. Proste narzędzia konsolowe pozwalają na wykonanie każdego zadania realizowalnego z użyciem plug-inu, o ile wiesz, jak skompilować aplikację internetową w języku Java.

Przechodząc na narzędzia konsolowe, uzyskasz większą kontrolę nad tym, jakie pliki będą dodawane do aplikacji. Jeśli będziesz korzystał z narzędzi konsolowych w sposób pokazany w końcowej części rozdziału 1., „Konfiguracja środowiska", i stworzysz strukturę katalogów analogiczną do tej z ostatniego przykładu, to uzyskasz możliwość wykluczenia z aplikacji każdej biblioteki, której nie potrzebujesz.

Prawdopodobnie zechcesz jednak pozostawić w strukturze katalogów bibliotekę *appengine-api-x.y.z.jar* (gdzie *x.y.z* to numer aktualnej wersji). Bez niej trudno stworzyć jakąkolwiek użyteczną aplikację.

Tworzenie odpowiednich struktur katalogów mogą Ci ułatwić narzędzia automatyzacji budowy, takie jak Maven albo Ant, być może wraz z plug-inem Ivy przeznaczonym dla Ant. Pozwalają one na skompilowanie kodu do żądanej postaci za pomocą polecenia wydawanego z linii komend, niezależnie od tego, jakie środowisko było używane podczas pisania kodu. Budowanie kodu odbywa się wówczas niezależnie od środowiska programistycznego, co przy okazji ułatwia programiście przechodzenie pomiędzy Eclipse, NetBeans i IntelliJ IDEA, kiedy tylko tego zapragnie.

Ustawianie parametrów wdrożenia

Za pomocą pliku *appengine-web.xml* możesz określić parametry konfiguracyjne związane konkretnie z Google App Engine. Większość parametrów definiowanych w tym pliku jest opcjonalna. Jeśli ich nie podasz, App Engine użyje wartości domyślnych.

Identyfikacja aplikacji

Pierwsze dwa elementy, `application` i `version`, są obowiązkowe. Google potrzebuje ich do zidentyfikowania wdrażanej aplikacji. Przykładowe wartości tych parametrów widać na listingu 3.1. Podczas wdrażania zostaniesz poproszony o podanie adresu e-mail oraz hasła, co pozwoli narzędziu wdrożeniowemu na sprawdzenie, czy masz odpowiednie prawa pozwalające na aktualizację danej aplikacji. Kwestia praw zostanie poruszona na końcu tego rozdziału.

Listing 3.1 Przykładowa treść pliku appengine-web.xml

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>blo-gae</application>
  <version>1</version>

  <static-files>
    <include path="/**.jpg" expiration="6h"/>
    <exclude path="/resources/**.xml"/>
  </static-files>

  <resource-files>
    <include path="/**.xml"/>
    <exclude path="/static/**.jpg"/>
  </resource-files>

  <!-- Konfiguracja java.util.logging -->
```

```
17   <system-properties>
18     <property name="blog.production-version" value="true"/>
19     <property name="java.util.logging.config.file"
20         value="WEB-INF/logging.properties"/>
21   </system-properties>
22
23   <env-variables>
24     <env-var name="LANGUAGE" value="PL"/>
25   </env-variables>
26
27   <ssl-enabled>false</ssl-enabled>
28
29   <sessions-enabled>true</sessions-enabled>
30
31   <user-permissions>
32     <permission class="com.unknown.CustomPermission"
33       name="custom-name" actions="read"/>
34   </user-permissions>
35
36   <public-root>/my</public-root>
37
38   <inbound-services>
39     <service>mail</service>
40   </inbound-services>
41
42   <precompilation-enabled>true</precompilation-enabled>
43
44</appengine-web-app>
```

Oddzielenie plików statycznych od zasobów

Elementy `static-files` i `resource-files` zajmujące linie od 6. do 14. są związane z wysokopoziomową strukturą katalogów, omówioną na początku tego rozdziału. Jeśli nie wyspecyfikujesz tych elementów, wszystkie pliki z wyjątkiem tych z */WEB-INF/** zostaną skopiowane zarówno na serwery statycznych plików, jak i na serwery aplikacji. Oznacza to, że zajmą dwa razy więcej miejsca, niż powinny, przez co możesz szybciej przekroczyć wyznaczony limit darmowego miejsca na dysku (quotę).

Katalog */WEB-INF/* i wszystko poniżej jest traktowane jako zasób. Dla każdego innego pliku powinieneś jawnie określić, czy będzie on odczytywany programistycznie po stronie serwera. Jeśli nie, powinieneś usunąć plik z listy zasobów i dodać go do plików statycznych.

Zasada ta działa także w drugą stronę. Jeśli któryś z plików poza */WEB-INF/* nigdy nie jest odczytywany bezpośrednio przez przeglądarkę i służy jedynie jako wejście dla programu działającego po stronie serwera, rozważ usunięcie go z listy plików statycznych. Może nawet najlepszym rozwiązaniem byłoby przesunięcie takich plików do folderu */WEB-INF/*, dzięki czemu zyskasz pewność, że nie zostaną one odczytane nigdzie indziej.

Konfiguracja właściwości systemowych i plików z logami

W pliku *appengine-web.xml* możesz ustawiać właściwości systemowe. Dość ciekawym zastosowaniem tych właściwości jest określenie, gdzie znajduje się plik *logging.properties*. Za jego pomocą możesz określić minimalny poziom błędów, o których komunikaty mają trafić do logów.

Za pomocą zmiennej `env-variables` określasz zmienne środowiskowe. Nie mają one żadnego dodatkowego zastosowania w App Engine. Zmienne środowiskowe można pobrać wewnątrz programu, wywołując metodę `System.getProperty()`.

Sesje

Domyślnie aplikacje Google App Engine są całkowicie bezstanowe i nie wspierają sesji. Jeśli jednak przełączysz element `sessions-enabled` na `true`, będziesz mógł korzystać z klasy `HttpSession` z interfejsu `Servlet`.

Decydując się na wykorzystanie sesji, miej na uwadze, że ich implementacja korzysta zarówno z magazynu Datastore, jak i z memcache. Ceną jest ilość miejsca na dysku, a potencjalnie także gorsza wydajność związana z odczytem i zapisem danych w cache.

Oznacza to, że musisz rozważyć zarówno koszty komunikacji z zewnętrznymi usługami, jak i koszty serializacji obiektów. Wszystkie obiekty dodawane do sesji powinny implementować `java.lang.Serializable`. Garść ostrzeżeń dotyczących stosowania tego interfejsu znajdziesz w rozdziale 14., „Optymalizacja wydajności za pomocą pamięci podręcznej cache".

Bezpieczeństwo aplikacji

W chwili pisania tego tekstu Google App Engine wspiera połączenia Secure Sockets Layer (SSL) tylko dla aplikacji działających w domyślnej domenie *appspot*. Jeśli aplikacja jest uruchomiona we własnej domenie użytkownika, wykorzystanie SSL nie jest (na razie) możliwe. Trwają prace nad implementacją tego wymagania. Aktualny stan i plany rozwoju App Engine możesz podejrzeć tutaj: *http://code.google.com/intl/pl/appengine/docs/roadmap.html*.

W tej chwili, zakładając, że Twoja aplikacja działa w domenie *appspot*, możesz zażądać zabezpieczenia URL za pomocą HTTPS, korzystając z elementu `ssl-enabled`. Decyzja dotyczy wszystkich adresów URL: albo wszystkie są zabezpieczone, albo żaden. Nie ma możliwości wskazania podgrupy, która ma zostać zabezpieczona.

Na potrzeby bibliotek języka Java dostarczających zewnętrzne klasy uprawnień możesz skonfigurować uprawnienia za pomocą elementu `user-permissions`. W przypadku niektórych frameworków jego zastosowanie może być obowiązkowe, jednak w przykładach w tej książce nie będziemy z niego korzystać.

Poza możliwością dodawania i usuwania plików z serwera plików statycznych możesz użyć jeszcze jednego elementu — `public-root` — w celu uniemożliwienia dostępu do plików statycznych. Skorzystanie z tego elementu nie powoduje zmiany lokalizacji plików statycznych ani odpowiadających im adresów URL. Oznacza on tyle, że nie można odwołać się do żadnego katalogu na serwerze plików statycznych poza tym wskazanym jako `public-root`. Jeśli jako `public-root` wskażemy */static*, nie będziemy mieli dostępu do plików na ścieżce */inne*. Dostępne będą natomiast pliki zagnieżdżone głębiej we wskazanym katalogu, na przykład */static/podpoziom/jeszcze-inne*.

Konfiguracja usług

Rysunek 3.1 przedstawia cztery usługi mające dostęp do frontendu w celu wywoływania akcji na serwerach aplikacji: Task Queue, cron, Mail i XMPP. Jedynie Mail i XMPP trzeba jawnie zadeklarować jako włączone w pliku *appengine-web.xml*.

Różnica pomiędzy XMPP i Mail a resztą jest taka, że są one **usługami przyjmującymi**, które odpowiadają na przychodzące z zewnątrz żądania korzystające z protokołów innych niż HTTP. Możesz je włączyć, przypisując wartości `mail` lub `xmpp_message` elementowi `inbound-services`, jak w przykładzie z listingu 3.1 (linia 39.).

> **Wyłączenie prekompilacji**
>
> Z powodów optymalizacyjnych Google App Engine prekompiluje bajtkod przed uruchomieniem aplikacji. W większości przypadków operacja ta nie jest zauważalna. Istnieje jednak co najmniej jeden scenariusz, w którym warto rozważyć jej wyłączenie: jeśli musisz korzystać z podpisanych bibliotek JAR, prekompilacja zepsuje podpis. W takiej sytuacji ustaw wartość elementu `precompilation-enabled` na `false`.

Uruchamianie zadań okresowych

Możesz wykorzystać plik *cron.xml* do planowania wykonania zadań. Usługa cron uruchamia zadania, wysyłając żądania HTTP do serwerów frontend. Przykładowa treść pliku konfiguracyjnego *cron.xml* została przedstawiona na listingu 3.2. Więcej informacji na temat tej usługi i jej konfiguracji znajdziesz w rozdziale 12., „Wykonywanie zadań w tle za pomocą Task Queue API oraz cron".

Listing 3.2. Przykładowa zawartość pliku cron.xml

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <cronentries>
03   <cron>
04     <url>/tasks/cron/mail</url>
05     <description>Wyślij spamerski e-mail…</description>
06     <schedule>every 2 minutes</schedule>
07   </cron>
08   <cron>
09     <url>/tasks/cron/read-rss</url>
10     <description>Odczytaj strumień RSS…</description>
11     <schedule>every day 22:00</schedule>
12     <timezone>Africa/Johannesburg</timezone>
13   </cron>
14 </cronentries>
```

Określenie indeksów w magazynie danych

W strukturze plików na początku rozdziału zauważysz dwa pliki, których nazwy zaczynają się od *datastore-indexes*: są to pliki *datastore-indexes.xml* oraz *datastore-indexes-auto.xml*.

Plik *datastore-indexes.xml* służy do wskazania, które właściwości których encji powinny zostać zindeksowane w celu optymalizacji czasu zapytań do magazynu danych.

Plik *datastore-indexes-auto.xml* w katalogu */WEB-INF/appengine-generated* jest tworzony przez serwer roboczy. Może on współpracować z *datastore-indexes.xml*, o ile wartość atrybutu `autoGenerate` jest ustawiona na `true`. W przeciwnym razie jest używany jedynie plik *datastore-indexes.xml*, a *datastore-indexes-auto.xml* jest ignorowany. Listing 3.3 przedstawia treść pliku *datastore-indexes.xml*, który pozwala serwerowi roboczemu na automatyczne wprowadzanie zmian w pliku *datastore-indexes-auto.xml*.

Listing 3.3. Przykładowa zawartość pliku datastore-indexes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<datastore-indexes
  autoGenerate="true">
  <datastore-index kind="BlogPost" ancestor="false">
    <property name="title" direction="asc" />
    <property name="entry-date" direction="desc" />
  </datastore-index>

  <datastore-index kind="User" ancestor="false">
    <property name="name" direction="asc" />
  </datastore-index>
</datastore-indexes>
```

Więcej informacji na temat tworzenia indeksów i korzystania z nich znajdziesz w rozdziale 4., „Modelowanie danych na potrzeby magazynu Google App Engine Datastore".

Blokowanie zakresów IP

W celu ochrony aplikacji przed nadużyciami możesz chcieć ograniczyć dostęp do niej za pomocą zakresu adresów IP. Możesz wpisać adres IP lub zakres adresów na „czarną listę" przy użyciu pliku *dos.xml* (nazwa to skrót od *denial of service* — odmowa obsługi).

Listing 3.4 przedstawia przykład zablokowania jednego konkretnego adresu IP z zakresu IPv4 oraz jednej podsieci. Podsieć została zdefiniowana za pomocą notacji Classless Inter-Domain Routing (CIDR). Pełne objaśnienie CIDR wykracza poza zakres tej książki, jednak z łatwością znajdziesz w internecie zasoby opisujące ten koncept.

Listing 3.4. Przykładowa treść pliku dos.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blacklistentries>
  <blacklist>
    <subnet>170.224.225.27</subnet>
    <description>Jeden konkretny adres</description>
  </blacklist>
  <blacklist>
    <subnet>170.224.225.0/24</subnet>
    <description>Cała podsieć</description>
  </blacklist>
</blacklistentries>
```

Konfiguracja poziomów logowania

Listing 3.1 zawiera odwołanie do pliku *logging.properties*. Za pomocą tego pliku możesz określić poziom logowania, wskazując najmniej ważny poziom komunikatów, które mają już zostać uwzględnione w plikach z logami. Przykład z listingu 3.5 definiuje logowanie wszystkich komunikatów o poziomie co najmniej `INFO`, niezależnie od pakietu czy klasy.

Listing 3.5. Definicja poziomu logowania w pliku logging.properties

```
01 #logging.properties
02 # Ustaw domyślny poziom logowania dla wszystkich loggerów na INFO
03 .level = INFO
```

Inne poziomy to `trace`, `debug`, `warning`, `error` i `fatal`. Możesz wprowadzić różne ustawienia dla różnych części aplikacji, wskazując nazwy pakietów lub pełne nazwy klas przed ciągiem `.level`. Jest to standardowy mechanizm konfiguracji `java.util.logging`.

Konfiguracja kolejek zadań

Jeśli nie stworzysz pliku *queues.xml*, będzie istniała jedna domyślna kolejka zadań o przepustowości pięciu zadań na sekundę. Jeśli chcesz wprowadzić dodatkowe kolejki o różnej przepustowości, możesz dodać plik *queues.xml*, taki jak na listingu 3.6. Więcej informacji na ten temat znajdziesz w rozdziale 12., „Wykonywanie zadań w tle za pomocą Task Queue API oraz cron".

Listing 3.6. Przykładowa treść pliku queues.xml

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <queue-entries>
03   <queue>
04     <name>mail-queue</name>
05     <rate>2000/d</rate>
06   </queue>
07   <queue>
08     <name>second-mail-queue</name>
09     <rate>8/m</rate>
10   </queue>
11 </queue-entries>
```

Zabezpieczanie URL

Większość plików konfiguracyjnych, które się pojawiły w tym rozdziale, była związana ze specyfiką App Engine i nie miałaby sensu przy wdrożeniu aplikacji w innym kontenerze. Jednak istnieje część konfiguracji, która działa tak samo we wszystkich standardowych kontenerach serwletów. Chodzi tu o jedyny pominięty do tej pory plik konfiguracyjny — *web.xml*. Nie wymaga on zbyt obszernych wyjaśnień.

Warto jednak wspomnieć przynajmniej jedną z części tego pliku — ograniczanie dostępu do poszczególnych adresów URL, tak by tylko zalogowani użytkownicy mogli się do nich odwołać. Fragment pliku *web.xml* opisujący adres URL o ograniczonym dostępie przedstawia listing 3.7.

Listing 3.7. Fragment pliku web.xml

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03     xmlns="http://java.sun.com/xml/ns/javaee"
04     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
05     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
06     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
07     version="2.5">
08
09   <!-- [...] -->
10
11   <servlet>
12     <servlet-name>ReceiveMailServlet</servlet-name>
13     <servlet-class>com.appspot.mail.[…]</servlet-class>
14   </servlet>
15   <servlet-mapping>
16     <servlet-name>ReceiveMailServlet</servlet-name>
17     <url-pattern>/_ah/mail/*</url-pattern>
18   </servlet-mapping>
19
20   <!-- [...] -->
21
22   <security-constraint>
23     <web-resource-collection>
24       <web-resource-name>receive-mail-url</web-resource-name>
25       <url-pattern>/_ah/mail/*</url-pattern>
26     </web-resource-collection>
27     <auth-constraint>
28       <role-name>admin</role-name>
29     </auth-constraint>
30   </security-constraint>
31
32   <!-- [...] -->
33
34 </web-app>
```

W linii 28. pojawia się nazwa roli `admin`. Możliwe są tutaj tylko dwie wartości: * lub `admin`. Wartość `admin` oznacza, że tylko zarejestrowani programiści aplikacji lub systemu mogą uzyskać dostęp do danego URL. Z kolei wartość * wskazuje, że dostęp może uzyskać każdy zalogowany użytkownik.

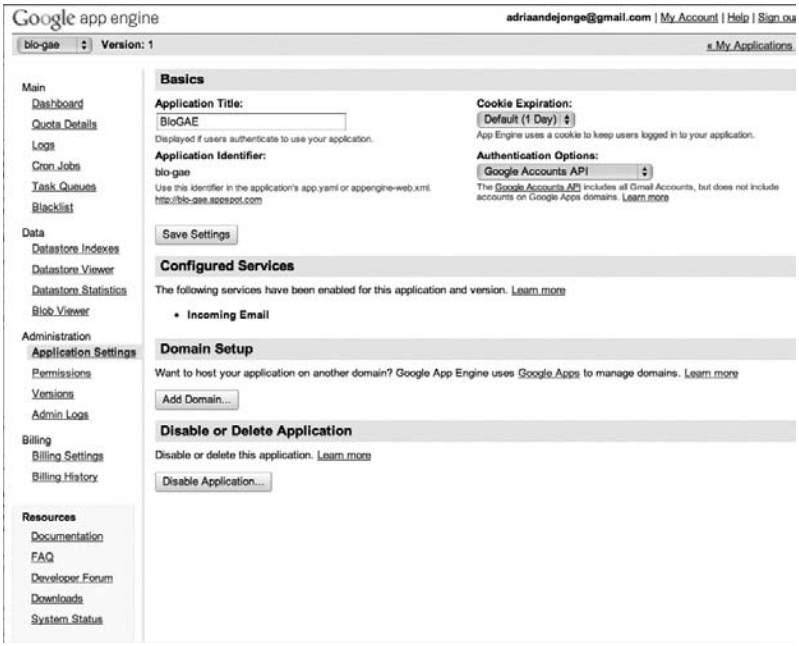
Konfiguracja panelu administracyjnego

Większość ustawień konfiguracyjnych Google App Engine definiuje się w plikach konfiguracyjnych dostarczanych wraz z aplikacją. Jednak pewna niewielka część konfiguracji odbywa się za pośrednictwem panelu administracyjnego. W przypadku większości obecnych tam opcji oczywiste

jest, dlaczego nie trafiły one do plików konfiguracyjnych. Bez wdawania się w filozoficzne dysputy można stwierdzić, że powody są czysto praktyczne. Panel administracyjny jest dość zwięzły i łatwo się w nim odnaleźć.

Podstawy aplikacji

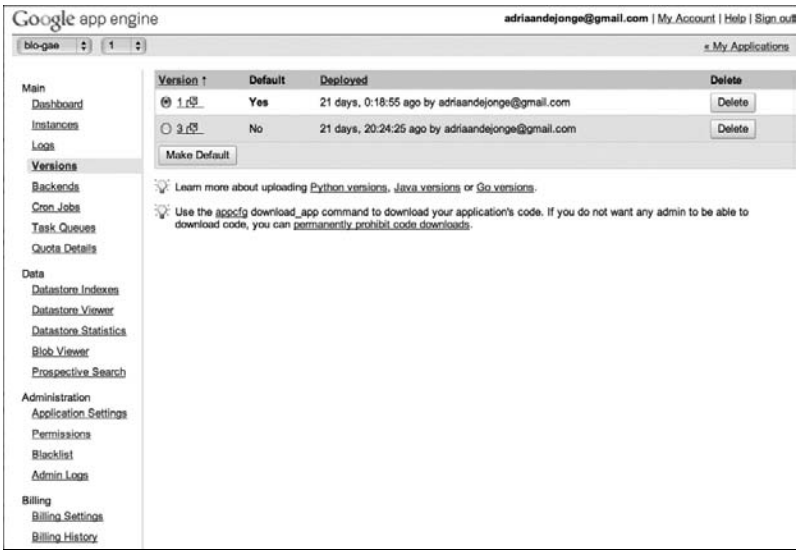
Na ekranie *Application Settings* (ustawienia aplikacji) możesz ustawić kilka podstawowych parametrów, takich jak nazwa aplikacji, czas ważności ciasteczek oraz sposób uwierzytelniania użytkowników. Możesz podać własną nazwę domenową, a także dezaktywować lub skasować aplikację, jeśli nie chcesz już świadczyć usług. Ten ekran panelu administracyjnego został przedstawiony na rysunku 3.2.



Rysunek 3.2. Określenie nazwy aplikacji, dziedziny oraz mechanizmu uwierzytelniania

Aktualna wersja

Listing 3.1 zawierał informację na temat wersji. Za pomocą panelu administracyjnego widocznego na rysunku 3.3 możesz przełączyć wersję aplikacji. Umożliwia to przygotowanie się do aktualizacji i podjęcie decyzji, kiedy nowa wersja stanie się aktywna. Możliwe jest także wycofanie wersji w przypadku nieoczekiwanych problemów. Pamiętaj, że niezależnie od wersji aplikacji uruchomiona jest tylko jedna wersja magazynu danych. Dostęp do danych należy implementować, mając na uwadze kompatybilność wstecz i w przód.



Rysunek 3.3. Wybór wersji aplikacji

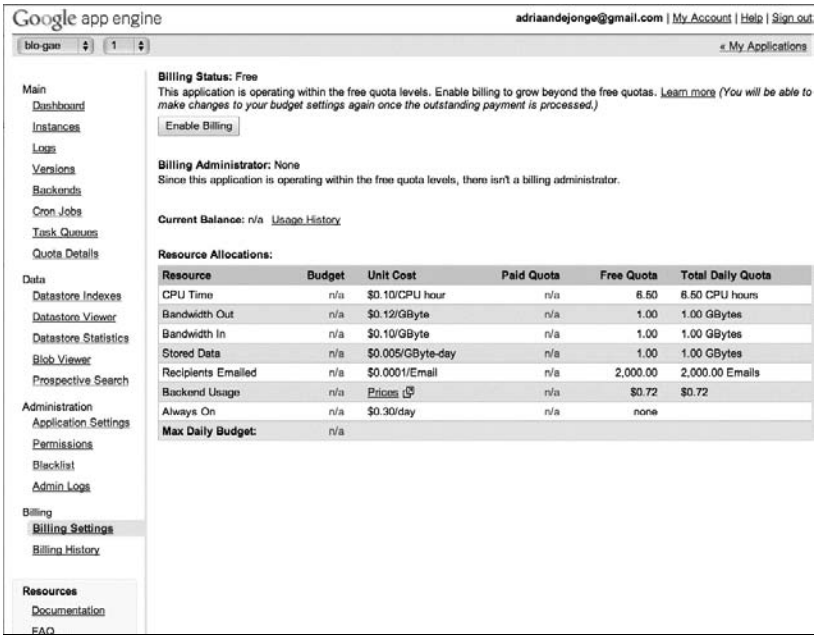
Dodawanie użytkowników

W miarę zdobywania popularności przez aplikację możesz chcieć dodawać do niej zaufane osoby, które będą sprawowały kontrolę nad sytuacją w czasie, gdy jesteś niedostępny. Możesz nadawać innym osobom prawa w oparciu o konta Google. Tacy użytkownicy mogą zarządzać aplikacjami i dodawać nowe wersje. Ekran dodawania użytkownika został przedstawiony na rysunku 3.4.

Rysunek 3.4. Zapraszanie dodatkowych użytkowników do sprawowania kontroli nad systemem

Naliczanie opłat

Gdy aplikacja stanie się popularna, może się okazać, że wyczerpałeś limit darmowych zasobów Google. Jeśli aplikacji towarzyszy przemyślany model biznesowy, nie powinno to stanowić problemu. Możesz wybrać, jaką ilość i które z dostępnych zasobów chcesz opłacać. Rysunek 3.5 przedstawia ustawienia związane z płatnościami.



Rysunek 3.5. Konfiguracja płatności za zasoby przekraczające dzienny darmowy limit

Podsumowanie

Po przeczytaniu tego rozdziału powinieneś mieć całkiem niezłe pojęcie na temat konfiguracji aplikacji App Engine. Na początku rozdziału przedstawiłem ogólną architekturę serwerów App Engine i wskazałem w niej miejsce Twojej aplikacji. Okazuje się, że jest ona podzielona pomiędzy serwery aplikacji i serwery statycznych plików. Lokalizacja plików zależy od zdefiniowanej przez Ciebie konfiguracji. Jeśli nie zachowasz ostrożności, pliki statyczne mogą trafić na oba serwery i zająć dwa razy więcej miejsca, niż powinny, przy czym liczba darmowego miejsca na dysku (quota) jest ograniczona. W dalszej części rozdziału przedstawiłem wszystkie istniejące pliki konfiguracyjne App Engine.

Skorowidz

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Ż

ZAGWARANTUJ NAJLEPSZĄ WYDAJNOŚĆ TWOJEJ APLIKACJI W KAŻDYCH WARUNKACH!

Google App Engine

Tworzenie wydajnych aplikacji w Javie

Google App Engine to idealny przykład usługi PaaS (ang. Platform as a Service). W tym modelu płaci się wyłącznie za wykorzystane zasoby dostawcy. Pozwala to na budowanie niezwykle elastycznych rozwiązań informatycznych. Jednak największą zaletą z perspektywy użytkownika tego rozwiązania jest brak konieczności utrzymywania własnej infrastruktury. Niezależnie od sytuacji będziesz zawsze przygotowany na obsłużenie dowolnie dużego ruchu, a to naprawdę się opłaca!

Dzięki tej książce błyskawicznie rozpoczniesz przygodę z platformą Google App Engine. Autor pokaże Ci, jak szybko tworzyć złożone i wydajne aplikacje w chmurze Google. Zaprezentuje przełomowe techniki pozwalające na skonstruowanie aplikacji, które są w stanie odpowiedzieć na żądanie w ciągu dwóch sekund przy tzw. zimnym uruchomieniu i w ciągu co najwyżej setek milisekund podczas normalnego działania w pozostałej części sesji. W trakcie lektury dowiesz się, jak uniknąć najczęstszych błędów, które dramatycznie pogarszają wydajność i skalowalność aplikacji w chmurze, oraz poznasz najświeższe technologie do tworzenia interfejsów użytkownika. Proces powstawania aplikacji został omówiony od podstaw — od projektowania i modelowania danych, przez bezpieczeństwo i testowanie, aż po wdrożenie.

Po lekturze tej książki:

- swobodnie zainstalujesz aplikację na platformie Google App Engine
- skorzystasz z dodatkowych możliwości platformy
- będziesz przechowywać dane w Datastore'ach
- stworzysz w pełni bezpieczne rozwiązanie
- zagwarantujesz Twojej aplikacji najlepszą infrastrukturę!

helion.pl
księgarnia internetowa

Nr katalogowy: 9273

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ

ISBN 978-83-246-4689-0

9788324646890

Cena 54,90 zł

Informatyka w najlepszym wydaniu