Capítulo 2

TUDO COMEÇA EM PRODUÇÃO

Ao contrário do que muitos processos de desenvolvimento sugerem, o ciclo de vida do software só deveria começar quando usuários começam a usá-lo. O problema da última milha, apresentado no capítulo *Introdução*, deveria ser a primeira milha, pois nenhum software entrega valor antes de entrar em produção.

Por isso, o objetivo deste capítulo é subir um ambiente de produção completo, começando do zero, e instalar uma aplicação Java relativamente complexa que será usada como ponto de partida para os conceitos e práticas de DevOps apresentados no livro.

Ao final deste capítulo, teremos uma loja virtual web completa rodando — suportada por um banco de dados — que permitirá que usuários se cadastrem, efetuem compras, além de fornecer ferramentas de administração para gerenciamento do catálogo de produtos, das promoções e do conteúdo disponibilizado na loja virtual.

De que adianta adicionar novas funcionalidades, melhorar o desempenho, corrigir defeitos e deixar as telas da loja virtual mais

bonitas, se isso tudo não for para produção? Vamos começar fazendo exatamente essa parte difícil, essa tal de última milha: colocar o software em produção.

2.1 NOSSA APLICAÇÃO DE EXEMPLO: A LOJA VIRTUAL

Como o foco do livro não é no processo de desenvolvimento em si, mas sim nas práticas de DevOps que auxiliam o *build*, *deploy* e operação de uma aplicação em produção, usaremos uma aplicação fictícia baseada em um projeto de código aberto. A loja virtual é uma aplicação web escrita em Java sobre a plataforma *Broadleaf Commerce* (http://www.broadleafcommerce.org/).

O código utilizado neste capítulo e no restante do livro foi escrito com base no site de demonstração oferecido pela Broadleaf e pode ser acessado no seguinte repositório do GitHub: https://github.com/dtsato/loja-virtual-devops/.

O *Broadleaf Commerce* é uma plataforma flexível que expõe pontos de configuração e de extensão que podem ser customizados para implementar funcionalidades específicas. No entanto, boa parte das funcionalidades padrão de um site de compras online, como o Submarino ou a Amazon, já estão disponíveis, incluindo:

- Navegação e busca no catálogo de produtos;
- Páginas de produto com nome, descrição, preço, fotos e produtos relacionados;
- Carrinho de compras;
- Processo de *checkout* customizável, incluindo códigos promocionais, dados de cobrança e de entrega;

- Cadastro de usuário;
- Histórico de compras;
- Ferramentas de administração para: catálogo de produtos, promoções, preços, taxas de frete e páginas com conteúdo customizado.

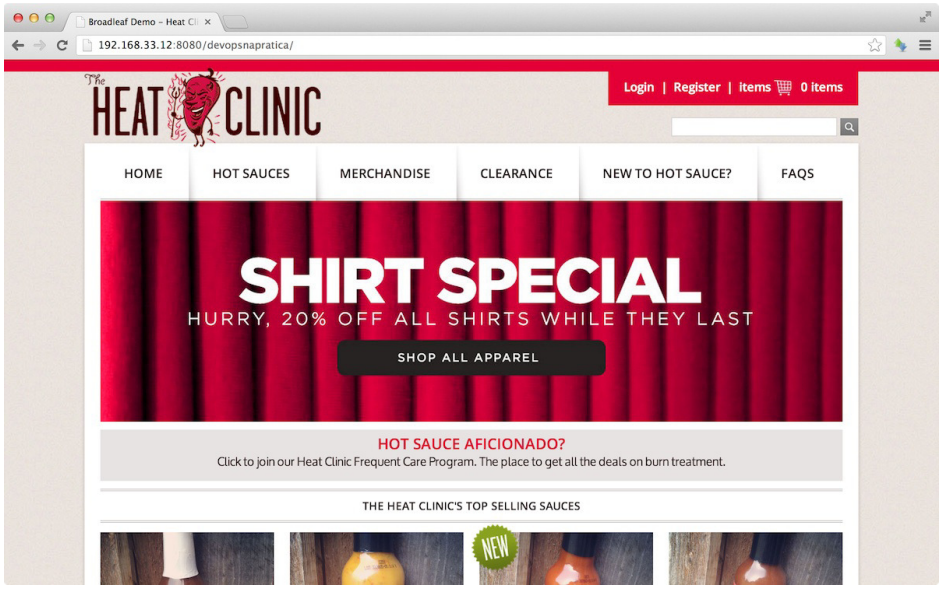

Figura 2.1: Uma prévia da página inicial da loja virtual

Além disso, o *Broadleaf Commerce* utiliza diversos frameworks bem estabelecidos na comunidade Java, tornando o exemplo mais interessante do ponto de vista de DevOps, pois representa bem a complexidade envolvida no processo de *build* e *deploy* de uma aplicação Java no mundo real. Os detalhes de implementação vão além do escopo deste livro, porém é importante conhecer um pouco das principais tecnologias e frameworks utilizados nesta aplicação:

- **Java**: a aplicação é escrita em Java (http://java.oracle.com/), compatível com a versão Java SE 6 e superiores.

- **Spring**: o Spring (http://www.springframework.org/) é um

framework de desenvolvimento de aplicações corporativas, popular na comunidade Java. Ele oferece diversos componentes como: injeção de dependências, gerenciamento de transações, segurança, um framework de MVC, dentre outros.

- **JPA e Hibernate**: o JPA é a API de persistência do Java e o Hibernate (http://www.hibernate.org/) é a implementação mais famosa da JPA na comunidade Java, oferecendo recursos para realizar o mapeamento objeto-relacional (*object-relational mapping* ou ORM) entre objetos Java e as tabelas no banco de dados.

- **Google Web Toolkit**: o GWT (http://developers.google.com/web-toolkit/) é um framework desenvolvido pelo Google para facilitar a criação de interfaces ricas que rodam no browser. O GWT permite que o desenvolvedor escreva código Java que é então compilado para JavaScript. A loja virtual utiliza o GWT para implementar a interface gráfica das ferramentas de administração.

- **Apache Solr**: o Solr (http://lucene.apache.org/solr/) é um servidor de pesquisa que permite a indexação do catálogo de produtos da loja virtual e oferece uma API eficiente e flexível para efetuar consultas de texto em todo o catálogo.

- **Tomcat**: o Tomcat (http://tomcat.apache.org/) é um servidor que implementa as tecnologias web (Java Servlet e JavaServer Pages) do Java EE. Apesar do *Broadleaf Commerce* rodar em servidores de aplicação alternativos (como Jetty, GlassFish ou JBoss), usaremos o Tomcat por

ser uma escolha comum em diversas empresas rodando aplicações web Java.

- **MySQL**: o MySQL (http://www.mysql.com/) é um servidor de banco de dados relacional. O JPA e o Hibernate permitem que a aplicação rode em diversos outros servidores de banco de dados (como Oracle, PostgreSQL ou SQL Server), porém também utilizaremos o MySQL por ser uma escolha popular e por disponibilizar uma versão de código aberto.

Não é uma aplicação pequena. Melhor ainda: ela utiliza bibliotecas que são frequentemente encontradas na grande maioria das aplicações Java no mercado. Como falamos, vamos usá-la como nosso exemplo, mas você também pode seguir o processo com a sua própria aplicação, ou ainda escolher outro software, independente de linguagem, para aprender as técnicas de DevOps que serão apresentadas e discutidas durante o livro.

O próximo objetivo é colocar a loja virtual no ar, porém, antes de fazer o primeiro *deploy*, é preciso ter um ambiente de produção pronto, com servidores em que o código possa rodar. O ambiente de produção em nosso exemplo será inicialmente composto de 2 servidores, conforme mostra a figura a seguir:

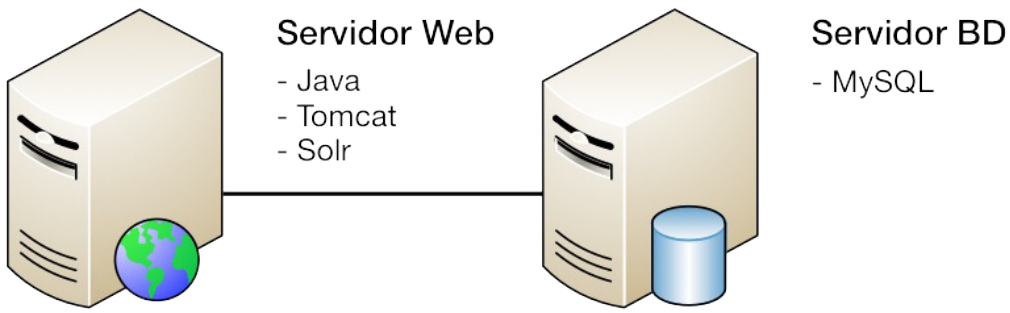

Figura 2.2: Ambiente de produção da loja virtual

Usuários acessarão a loja virtual, que rodará em uma instância do Tomcat no servidor web. O servidor web rodará todas as bibliotecas e frameworks Java utilizados pela loja virtual, inclusive uma instância embutida do Solr. Por fim, a aplicação web utilizará o MySQL, rodando em um servidor de banco de dados separado.

Esta é uma arquitetura web de duas camadas, comumente usada por diversas aplicações no mundo real. No capítulo *Tópicos avançados*, discutiremos com mais detalhes os fatores que influenciam a escolha da arquitetura física para sua aplicação, mas por enquanto vamos usar algo comum para simplificar o exemplo e colocar a loja virtual no ar o quanto antes.

2.2 INSTALANDO O AMBIENTE DE PRODUÇÃO

Comprar servidores e hardware para a loja virtual custaria muito dinheiro, então inicialmente vamos usar máquinas virtuais. Isso permitirá criar todo o ambiente de produção na nossa máquina. Existem diversas ferramentas para rodar máquinas virtuais (como VMware ou Parallels), porém algumas são pagas e a maioria usa uma interface gráfica para configuração, o que tornaria este capítulo uma coleção de *screenshots* difíceis de acompanhar.

Para resolver esses problemas, usaremos duas ferramentas que facilitam o uso e configuração de máquinas virtuais: o **Vagrant** (http://www.vagrantup.com) e o **VirtualBox** (http://www.virtualbox.org). O VirtualBox é uma ferramenta da Oracle que permite configurar e rodar máquinas virtuais nas principais plataformas: Windows, Linux, Mac OS X e Solaris.

O VirtualBox possui uma ferramenta de linha de comando para executar e configurar as máquinas virtuais. Entretanto, para facilitar ainda mais o trabalho, usaremos o Vagrant que fornece uma DSL em Ruby para definir, gerenciar e configurar ambientes virtuais. O Vagrant possui também uma interface de linha de comando simples para subir e interagir com esses ambientes virtuais.

Caso você já possua o Vagrant e o VirtualBox instalados e funcionando, você pode pular para a subseção *Declarando e Subindo os Servidores*. Caso contrário, o processo de instalação dessas ferramentas é simples.

Instalando o VirtualBox

Para instalar o VirtualBox, acesse a página de *download*, em http://www.virtualbox.org/wiki/Downloads, e escolha a versão mais atual. No momento da escrita do livro, a versão mais nova é o VirtualBox 4.3.8 e será a versão utilizada no decorrer do livro.

Escolha o pacote de instalação de acordo com a sua plataforma: no Windows, o instalador é um arquivo executável `.exe` ; no Mac OS X, o instalador é um pacote `.dmg` ; no Linux, o VirtualBox disponibiliza pacotes `.deb` ou `.rpm` , dependendo da sua distribuição.

Uma vez que você tenha feito o *download*, instale o pacote. No Windows e no Mac OS X, basta dar um duplo clique no arquivo de instalação e seguir as instruções do instalador.

No Linux, caso tenha escolhido o pacote `.deb` , instale-o executando o comando `dpkg -i {arquivo.deb}` , substituindo

`{arquivo.deb}` pelo nome do arquivo baixado, por exemplo `virtualbox-4.3_4.3.8-92456~Ubuntu~raring_i386.deb`. Caso tenha escolhido o pacote `.rpm`, instale-o executando o comando `rpm -i {arquivo.rpm}`, substituindo `{arquivo.rpm}` pelo nome do arquivo baixado, por exemplo `VirtualBox-4.3-4.3.8_92456_el6-1.i686.rpm`.

> **Observação**
>
> No Linux, caso seu usuário não seja `root`, você precisará rodar os comandos anteriores com o comando `sudo` na frente, por exemplo: `sudo dpkg -i {arquivo.deb}` ou `sudo rpm -i {arquivo.rpm}`.

Para testar que o VirtualBox está instalado corretamente, acesse o terminal e execute o comando `VBoxManage -v`. Para abrir o terminal no Windows, você pode usar `Win+R` e digitar `cmd`. Caso tudo esteja correto, o comando retornará algo como "4.3.8r92456", dependendo da versão instalada.

Instalando o Vagrant

Uma vez que o VirtualBox estiver instalado, prossiga com o processo de instalação do Vagrant, que é bem parecido. Acesse a página de *download* do Vagrant (http://www.vagrantup.com/downloads.html) e escolha a versão mais atual. No momento da escrita do livro, a versão mais nova é o Vagrant 1.5.1 e será a versão utilizada.

Faça o *download* do pacote de instalação de acordo com a sua plataforma: no Windows, o instalador é um arquivo `.msi` ; no Mac OS X, o instalador é um pacote `.dmg` ; e no Linux, o Vagrant disponibiliza pacotes `.deb` ou `.rpm` , dependendo da sua distribuição.

Assim que tenha feito o *download* do pacote, instale-o: no Windows e no Mac OS X, basta dar um duplo clique no arquivo de instalação e seguir as instruções do instalador. No Linux, basta seguir os mesmos passos da instalação do VirtualBox no Linux usando o pacote escolhido (`vagrant_1.5.1_i686.deb` ou `vagrant_1.5.1_i686.rpm`).

No Mac OS X e no Windows, o comando `vagrant` já é colocado no `PATH` após a instalação. No Linux, você vai precisar adicionar `/opt/vagrant/bin` no seu `PATH` .

Para testar que o Vagrant está instalado corretamente, acesse o terminal e execute o comando `vagrant -v` . Caso tudo esteja correto, o comando retornará algo como "Vagrant 1.5.1", dependendo da versão instalada.

O último passo necessário é configurar uma imagem inicial que serve de *template* para inicializar cada máquina virtual. Estas imagens, também conhecidas como *box*, servem como ponto de partida, contendo o sistema operacional básico. No nosso caso, usaremos uma *box* oferecida pelo próprio Vagrant, que contém a imagem de um Linux Ubuntu 12.04 LTS de 32 bits. Para baixar e configurar esta *box*, é preciso executar o comando:

```
$ vagrant box add hashicorp/precise32
==> box: Loading metadata for box 'hashicorp/precise32'
    box: URL: https://vagrantcloud.com/hashicorp/precise32
...
```

Esse comando fará o *download* do arquivo de imagem da máquina virtual, que é grande (299MB), portanto ele vai demorar para executar e será necessária uma boa conexão com a internet. Uma lista de outras imagens disponibilizadas pela comunidade Vagrant pode ser encontrada em http://www.vagrantbox.es/ e o processo de inicialização é parecido com o comando anterior. A empresa por trás do Vagrant, a HashiCorp, introduziu recentemente o **Vagrant Cloud** (https://vagrantcloud.com/), uma forma de encontrar e compartilhar *boxes* com a comunidade.

Declarando e subindo os servidores

Uma vez que o Vagrant e o VirtualBox estão funcionando, declarar e gerenciar um ambiente virtual é simples. A declaração de configuração das máquinas virtuais é feita em um arquivo chamado `Vagrantfile`. O comando `vagrant init` cria um arquivo `Vagrantfile` inicial, com comentários que explicam todas as opções de configuração disponíveis.

No nosso caso, começaremos de forma simples e configuraremos apenas o necessário para subir os dois servidores de produção. O conteúdo do `Vagrantfile` que precisamos é:

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise32"

  config.vm.define :db do |db_config|
    db_config.vm.hostname = "db"
    db_config.vm.network :private_network,
                         :ip => "192.168.33.10"
  end

  config.vm.define :web do |web_config|
    web_config.vm.hostname = "web"
```

```
    web_config.vm.network :private_network,
                          :ip => "192.168.33.12"
  end
end
```

Esse arquivo configura duas máquinas virtuais, com apelidos de `db` e `web`. Ambas utilizarão a *box* `hashicorp/precise32` que instalamos anteriormente. Cada uma possui um bloco de configuração que define o *hostname* da máquina e um endereço IP para configurar a conexão com a rede. Os endereços IP `192.168.33.10` e `192.168.33.12` foram escolhidos de forma arbitrária para evitar conflitos com as configurações de rede da sua máquina.

Para subir os servidores, basta executar o comando `vagrant up` no mesmo diretório onde o arquivo `Vagrantfile` foi criado, e o Vagrant tomará conta de subir e configurar as máquinas virtuais para você. Se tudo correr bem, a saída do comando será algo parecido com:

```
$ vagrant up
Bringing machine 'db' up with 'virtualbox' provider...
Bringing machine 'web' up with 'virtualbox' provider...
==> db: Importing base box 'hashicorp/precise32'...
==> db: Matching MAC address for NAT networking...
==> db: Checking if box 'hashicorp/precise32' is up to date...
==> db: Setting the name of the VM: blank_db_1394680068450_7763
==> db: Clearing any previously set network interfaces...
==> db: Preparing network interfaces based on configuration...
    db: Adapter 1: nat
    db: Adapter 2: hostonly
==> db: Forwarding ports...
    db: 22 => 2222 (adapter 1)
==> db: Booting VM...
==> db: Waiting for machine to boot. This may take a few
                                                    minutes...
    db: SSH address: 127.0.0.1:2222
    db: SSH username: vagrant
    db: SSH auth method: private key
```

```
==> db: Machine booted and ready!
==> db: Configuring and enabling network interfaces...
==> db: Mounting shared folders...
    db: /vagrant => /private/tmp/blank
==> web: Importing base box 'hashicorp/precise32'...
==> web: Matching MAC address for NAT networking...
==> web: Checking if box 'hashicorp/precise32' is up to date...
==> web: Setting the name of the VM:
                                    blank_web_1394680087001_3842
==> web: Fixed port collision for 22 => 2222. Now on port 2200.
==> web: Clearing any previously set network interfaces...
==> web: Preparing network interfaces based on configuration...
    web: Adapter 1: nat
    web: Adapter 2: hostonly
==> web: Forwarding ports...
    web: 22 => 2200 (adapter 1)
==> web: Booting VM...
==> web: Waiting for machine to boot. This may take a few
                                                        minutes...
    web: SSH address: 127.0.0.1:2200
    web: SSH username: vagrant
    web: SSH auth method: private key
==> web: Machine booted and ready!
==> web: Configuring and enabling network interfaces...
==> web: Mounting shared folders...
    web: /vagrant => /private/tmp/blank
```

Isso significa que as máquinas virtuais estão funcionando e prontas para serem usadas. Se você ler a saída com cuidado, verá que os passos foram os mesmos para ambas as máquinas `db` e `web`.

Além disso, o Vagrant também mapeou portas de SSH e montou uma partição para acessar os arquivos existentes no diretório da máquina hospedeira. Isso pode soar um pouco confuso, mas demonstra algumas das funcionalidades poderosas que o Vagrant disponibiliza.

2.3 CONFIGURANDO DEPENDÊNCIAS NOS

SERVIDORES DE PRODUÇÃO

Agora que os servidores estão rodando, é preciso instalar os pacotes e dependências para que a loja virtual funcione. Administradores de sistema provavelmente já estão acostumados a trabalhar em diversas máquinas ao mesmo tempo, porém desenvolvedores geralmente trabalham sempre em uma máquina só.

Nas próximas seções, começaremos a interagir com mais de uma máquina ao mesmo tempo e a maior parte dos comandos precisará ser executada no servidor correto. Usaremos uma notação especial para guiar as instruções no restante do livro, representando o usuário e o servidor no *prompt* da linha de comando para servir de lembrete para você verificar se está rodando o comando no servidor certo.

O padrão será `{usuário}@{servidor}$`, no qual o usuário geralmente será `vagrant` e o servidor pode variar entre `db` ou `web`. Desta forma, comandos precedidos de `vagrant@db$` devem ser executados no servidor `db`, e comandos precedidos de `vagrant@web$` devem ser executados no servidor `web`.

Em vez de pular de um servidor para o outro, vamos focar inicialmente em um servidor por vez, começando pelo servidor de banco de dados.

Servidor de banco de dados

Primeiramente, precisamos logar no servidor de banco de dados para instalar o MySQL. Isso é feito através do comando `vagrant ssh db`, que abre uma sessão SSH com o servidor `db`.

Uma vez logado, a instalação do servidor MySQL é feita através do pacote `mysql-server` :

```
vagrant@db$ sudo apt-get update
Ign http://us.archive.ubuntu.com precise InRelease
Ign http://us.archive.ubuntu.com precise-updates InRelease
...
Reading package lists... Done
vagrant@db$ sudo apt-get install mysql-server
Reading package lists... Done
Building dependency tree
...
ldconfig deferred processing now taking place
```

Para entender o que está acontecendo, vamos analisar cada parte desse comando. O comando `apt-get` é usado para gerenciar quais pacotes estão instalados no sistema. A instrução `update` atualiza o índice de pacotes para as últimas versões. A instrução `install` serve para instalar um novo pacote no sistema.

Nesse caso, o pacote que queremos instalar é o pacote `mysql-server` que contém o servidor MySQL. O comando inteiro é precedido pelo comando `sudo` . Em sistemas UNIX, existe um usuário especial com "superpoderes" conhecido como usuário `root` e o comando `sudo` permite executar algo como se o usuário atual fosse `root` .

O processo de instalação do servidor MySQL exige a escolha de uma senha de acesso para o usuário `root` . Você precisará fornecer essa senha duas vezes no processo de instalação. A escolha de uma senha segura é importante para garantir a segurança do sistema, porém é também um tópico de discussão à parte, que abordaremos no capítulo *Tópicos avançados*.

Se alguém descobrir a senha mestre, essa pessoa poderá executar qualquer comando no banco de dados, inclusive apagar todas as tabelas! Na vida real, você não quer que isso aconteça, porém, no exemplo do livro, usaremos uma senha simples para facilitar: `secret`. Se preferir, você pode escolher outra senha, mas, daqui para a frente, tome cuidado para substituir qualquer menção da senha `secret` pela qual você escolheu.

Ao terminar a instalação do pacote, o servidor MySQL já vai estar rodando, porém apenas para acesso local. Como precisamos acessar o banco de dados do servidor web, é preciso configurar o MySQL para permitir conexões externas. Para isto, basta criar o arquivo de configuração `/etc/mysql/conf.d/allow_external.cnf`.

Você pode usar o editor de texto que preferir — `emacs` e `vim` são duas opções comuns —, porém aqui usaremos o `nano`, por ser um editor mais simples de aprender para quem não está acostumado a editar arquivos na linha de comando. O comando seguinte cria o arquivo (como `root` novamente) desejado:

```
vagrant@db$ sudo nano /etc/mysql/conf.d/allow_external.cnf
```

Basta então digitar o conteúdo do arquivo a seguir, usar `Ctrl+O` para salvar e `Ctrl+X` para salvar e sair do editor:

```
[mysqld]
   bind-address = 0.0.0.0
```

Uma vez que o arquivo foi criado, é preciso reiniciar o servidor MySQL para que ele perceba a nova configuração. Isso é feito com o comando:

```
vagrant@db$ sudo service mysql restart
mysql stop/waiting
```

```
mysql start/running, process 6460
```

Neste momento, o servidor MySQL já está pronto e rodando, porém ainda não criamos nenhum banco de dados. É comum que cada aplicação crie seu próprio banco de dados, ou *schema*, para persistir suas tabelas, índices e dados. Precisamos então criar um banco de dados para a loja virtual, que chamaremos de `loja_schema`, através do comando:

```
vagrant@db$ mysqladmin -u root -p create loja_schema
Enter password:
```

Ao executar este comando, a opção `-u root` instrui que a conexão com o MySQL deve ser feita como usuário root, e a opção `-p` vai perguntar a senha, que nesse caso será `secret`. O resto do comando é autoexplicativo. Para verificar que o banco de dados foi criado com sucesso, você pode executar o seguinte comando que lista todos os banco de dados existentes:

```
vagrant@db$ mysql -u root -p -e "SHOW DATABASES"
Enter password:
+--------------------+
| Database           |
+--------------------+
| information_schema |
| loja_schema        |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

Assim como não é recomendável executar comandos no sistema operacional como usuário `root`, não é recomendável executar queries no MySQL como usuário `root`. Uma boa prática é criar usuários específicos para cada aplicação, liberando apenas o acesso necessário para a aplicação rodar.

Além disso, a instalação padrão do servidor MySQL inclui uma conta anônima para acesso ao banco de dados. Antes de criar a conta para a loja virtual, precisamos remover a conta anônima:

```
vagrant@db$ mysql -uroot -p -e "DELETE FROM mysql.user WHERE \
> user=''; FLUSH PRIVILEGES"
Enter password:
```

Uma vez que a conta anônima foi removida, criaremos um usuário chamado `loja`, com uma senha `lojasecret`, com acesso exclusivo ao *schema* `loja_schema`:

```
vagrant@db$ mysql -uroot -p -e "GRANT ALL PRIVILEGES ON \
> loja_schema.* TO 'loja'@'%' IDENTIFIED BY 'lojasecret';"
Enter password:
```

Para testar que o novo usuário foi criado corretamente, podemos executar uma *query* simples para verificar que tudo está OK (a senha deve ser `lojasecret`, pois estamos executando o comando como usuário `loja` em vez de `root`):

```
vagrant@db$ mysql -u loja -p loja_schema -e
  "select database(), user()"
Enter password:
+--------------+----------------+
| database()   | user()         |
+--------------+----------------+
| loja_schema  | loja@localhost |
+--------------+----------------+
```

Neste ponto, o servidor de banco de dados já está configurado e rodando corretamente. Para deslogar da máquina virtual `db`, você pode digitar o comando `logout` ou pressionar `Ctrl+D`. O banco de dados ainda está vazio — nenhuma tabela ou dados foram criados —, porém cuidaremos disto na seção *Build e deploy da aplicação*. Antes disso, precisamos terminar de configurar o resto da infraestrutura: o servidor web.

Servidor web

Vamos instalar o Tomcat e configurar uma fonte de dados para usar o banco de dados criado anteriormente. Para isto, precisamos logar no servidor rodando o comando `vagrant ssh web`, que abre uma sessão SSH com o servidor `web`. Uma vez logado, a instalação do Tomcat é feita através do pacote `tomcat7`. Também precisamos instalar o cliente para conectar com o banco de dados MySQL, disponível no pacote `mysql-client`:

```
vagrant@web$ sudo apt-get update
Ign http://us.archive.ubuntu.com precise InRelease
Ign http://us.archive.ubuntu.com precise-updates InRelease
...
Reading package lists... Done
vagrant@web$ sudo apt-get install tomcat7 mysql-client
Reading package lists... Done
Building dependency tree
...
ldconfig deferred processing now taking place
```

Isto vai instalar o Tomcat 7, o cliente MySQL e todas as suas dependências, incluindo o Java 6. Para verificar que o Tomcat está rodando corretamente, você pode abrir seu navegador e digitar a URL `http://192.168.33.12:8080/` do servidor web — `192.168.33.12` foi o endereço IP configurado no Vagrant e `8080` é a porta onde o Tomcat roda por padrão. Se tudo estiver ok, você deve ver uma página com uma mensagem *"It works!"*.

A loja virtual precisa de uma conexão segura com o servidor para transferir dados pessoais — como senhas ou números de cartão de crédito — de forma criptografada. Para isso, precisamos configurar um conector SSL para que o Tomcat consiga rodar tanto em HTTP quanto HTTPS.

Conexões SSL usam um certificado assinado por uma autoridade confiável para identificar o servidor e para criptografar os dados que trafegam entre o seu navegador e o servidor. Usaremos a ferramenta `keytool`, que é parte da distribuição padrão do Java, para gerar um certificado para o nosso servidor web. Durante este processo, você precisará fornecer diversas informações sobre o servidor além de uma senha para proteger o `keystore`, que no nosso caso será novamente `secret` para simplificar. O comando completo, juntamente com exemplos dos dados que precisam ser fornecidos, fica:

```
vagrant@web$ cd /var/lib/tomcat7/conf
vagrant@web$ sudo keytool -genkey -alias tomcat -keyalg RSA \
> -keystore .keystore
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Loja Virtual
What is the name of your organizational unit?
  [Unknown]:  Devops na Prática
What is the name of your organization?
  [Unknown]:  Casa do Código
What is the name of your City or Locality?
  [Unknown]:  São Paulo
What is the name of your State or Province?
  [Unknown]:  SP
What is the two-letter country code for this unit?
  [Unknown]:  BR
Is CN=Loja Virtual, OU=Devops na Prática, ..., C=BR correct?
  [no]:  yes

Enter key password for <tomcat>
  (RETURN if same as keystore password):
```

Explicando o comando passo a passo: a opção `-genkey` gera uma nova chave; a opção `-alias tomcat` define um apelido para o certificado; a opção `-keyalg RSA` especifica o algoritmo usado para gerar a chave, nesse caso RSA; por fim, a opção `-keystore`

`.keystore` define o arquivo que armazenará o `keystore`. Este formato de arquivo permite armazenar diversas chaves/certificados no mesmo `keystore`, por isso é bom escolher um apelido para o certificado gerado para o Tomcat. Além disso, tanto o `keystore` quanto o certificado do Tomcat são protegidos por senha. Nesse caso, usamos `secret` para ambos.

Uma vez que o certificado foi criado e armazenado, precisamos configurar o servidor do Tomcat para habilitar o conector SSL. Isso é feito editando o arquivo `/var/lib/tomcat7/conf/server.xml`:

```
vagrant@web$ sudo nano /var/lib/tomcat7/conf/server.xml
```

Dentro deste arquivo, a definição do conector SSL para a porta 8443 já está predefinida, porém comentada. Você precisa descomentar a definição deste conector e adicionar mais dois atributos que representam o arquivo do `keystore` e a senha de proteção. A parte relevante do arquivo XML que precisamos alterar é:

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  ...
  <Service name="Catalina">
    ...
    <!-- Define a SSL HTTP/1.1 Connector on port 8443
     This connector uses the JSSE configuration, when using
     APR, the connector should be using the OpenSSL style
     configuration described in the APR documentation -->

    <Connector port="8443" protocol="HTTP/1.1"
               SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               keystoreFile="conf/.keystore"
               keystorePass="secret"
               clientAuth="false" sslProtocol="SSLv3" />
    ...
```

```
  </Service>
</Server>
```

A configuração do servidor web está quase pronta aqui. A última alteração que precisamos fazer é aumentar a quantidade de memória que o Tomcat pode usar. A loja virtual exige mais memória do que a instalação padrão do Tomcat define, por isso precisamos alterar uma linha no arquivo `/etc/default/tomcat7` :

```
vagrant@web$ sudo nano /etc/default/tomcat7
```

A linha que define a opção `JAVA_OPTS` precisa ser alterada para permitir que a máquina virtual do Java use até 512Mb de memória em vez da configuração padrão de 128Mb. Após a alteração, a opção `JAVA_OPTS` deve ficar:

```
JAVA_OPTS="-Djava.awt.headless=true -Xmx512M
                                    -XX:+UseConcMarkSweepGC"
```

Por fim, para que todas as configurações sejam aplicadas, precisamos reiniciar o servidor Tomcat:

```
vagrant@web$ sudo service tomcat7 restart
```

Para verificar que tudo está funcionando, você pode voltar no navegador e tentar acessar a página padrão através do conector SSL, acessando a URL `https://192.168.33.12:8443/` . Dependendo de qual navegador você está usando, um aviso surgirá para alertar que o certificado do servidor foi autoassinado.

Em um cenário real, você geralmente compra um certificado SSL assinado por uma autoridade confiável, mas nesse caso vamos prosseguir e aceitar o alerta, pois sabemos que o certificado foi autoassinado intencionalmente. Se tudo der certo, você verá

novamente a página *"It works!"*, dessa vez usando uma conexão segura em HTTPS.

Neste momento, ambos os servidores estão configurados e prontos para a última etapa: fazer o *build* e *deploy* para colocar a loja virtual em produção!

2.4 BUILD E DEPLOY DA APLICAÇÃO

Até agora, a configuração dos servidores de banco de dados e web foi relativamente genérica. Instalamos os pacotes de software básicos para rodar o MySQL e o Tomcat, configuramos conectores para HTTP/HTTPS, criamos um usuário e um banco de dados vazio, que por enquanto não estão diretamente associados à loja virtual, a não ser pelo nome do *schema* e do usuário que escolhemos.

Chegou o momento de baixar e compilar o código, rodar os testes, empacotar a aplicação e colocar a loja virtual no ar! Esse processo de compilação, teste e empacotamento é conhecido como *build*. As etapas do processo de *build* também podem incluir gerenciamento de dependências, rodar ferramentas de análise estática do código, cobertura de código, geração de documentação, dentre outros. A principal função do processo de *build* é gerar um ou mais artefatos, com versões específicas, que se tornam potenciais candidatos, ou *release candidates*, para o *deploy* em produção.

Geralmente, recomenda-se rodar o processo de *build* em um ambiente parecido com o ambiente de produção para evitar incompatibilidades de sistema operacional ou versões de

bibliotecas. Poderíamos até configurar o Vagrant para criar uma nova máquina virtual para utilizarmos como servidor de *build*.

Entretanto, para facilitar a discussão neste capítulo, faremos o *build* no servidor web, que é onde o *deploy* vai acontecer. Isso economizará diversos passos agora, porém vamos revisitar este processo no capítulo *Integração contínua*.

Para realizar o *build*, é preciso instalar mais algumas ferramentas: o **Git** (http://git-scm.com/) para controle de versão, o **Maven** (http://maven.apache.org/) para executar o *build* em si e o **JDK** (*Java Development Kit*) para conseguirmos compilar o código Java:

```
vagrant@web$ sudo apt-get install git maven2 openjdk-6-jdk
Reading package lists... Done
Building dependency tree
...
ldconfig deferred processing now taking place
```

Uma vez que essas ferramentas foram instaladas, precisamos baixar o código que está hospedado no GitHub através do comando `git clone`, e executar o *build* rodando o comando `mvn install`:

```
vagrant@web$ cd
vagrant@web$ git clone \
> https://github.com/dtsato/loja-virtual-devops.git
Cloning into 'loja-virtual-devops'...
remote: Counting objects: 11358, done.
remote: Compressing objects: 100% (3543/3543), done.
remote: Total 11358 (delta 6053), reused 11358 (delta 6053)
Receiving objects: 100% (11358/11358), 55.47 MiB | 1.14 MiB/s,
                                                        done.
Resolving deltas: 100% (6053/6053), done.
vagrant@web$ cd loja-virtual-devops
vagrant@web$ export MAVEN_OPTS=-Xmx256m
vagrant@web$ mvn install
```

```
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   loja-virtual-devops
[INFO]   core
[INFO]   admin
[INFO]   site
[INFO]   combined
[INFO] ------------------------------------------------------
[INFO] Building loja-virtual-devops
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------
...
```

A primeira execução do Maven vai demorar de 20 a 25 minutos, pois diversas coisas acontecem:

- **Subprojetos** — O projeto está organizado em 4 módulos: `core` possui as configurações principais, extensões e customizações da loja virtual; `site` é a aplicação web da loja virtual; `admin` é uma aplicação web restrita para administração e gerenciamento da loja virtual; por fim, `combined` é um módulo que agrega as duas aplicações web e o `core` em um único artefato. Cada módulo atua como um subprojeto que o Maven precisa realizar o *build* separadamente.

- **Resolução de dependências** — O Maven vai resolver todas as dependências do projeto e baixar as bibliotecas e frameworks necessários para compilar e rodar cada módulo da loja virtual.

- **Compilação** — O código Java e GWT precisa ser compilado para execução. Isso toma um bom tempo na primeira vez. O compilador Java é inteligente o suficiente para recompilar apenas o necessário em *builds* subsequentes.

- **Testes automatizados** — Em módulos que definem testes automatizados, o Maven vai executá-los e gerar relatórios de quais testes passaram e quais falharam.

- **Empacotamento de artefatos** — Por fim, o módulo `core` será empacotado como um arquivo `.jar` enquanto os módulos web (`site` , `admin` e `combined`) serão empacotados como um arquivo `.war` . Ambos os arquivos `jar` e `war` são equivalentes a arquivos `zip` que você baixa regularmente da internet, no entanto a estrutura interna dos pacotes é bem definida e padronizada pelo Java: o `jar` contém classes compiladas e serve como biblioteca enquanto o `war` contém classes, bibliotecas, arquivos estáticos e de configuração necessários para rodar uma aplicação web em um *container* Java como o Tomcat.

Ao final da execução do processo de *build*, se tudo correr bem, você deve ver um relatório do Maven parecido com:

```
...
[INFO] ------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------
[INFO] loja-virtual-devops ............... SUCCESS [1:35.191s]
[INFO] core .............................. SUCCESS [6:51.591s]
[INFO] admin ............................. SUCCESS [6:54.377s]
[INFO] site .............................. SUCCESS [4:44.313s]
[INFO] combined .......................... SUCCESS [25.425s]
[INFO] ------------------------------------------------------
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------
[INFO] Total time: 20 minutes 31 seconds
[INFO] Finished at: Sat Mar 15 02:34:24 UTC 2014
[INFO] Final Memory: 179M/452M
[INFO] ------------------------------------------------------
```

Isso significa que o *build* rodou com sucesso e o artefato que nos interessa para *deploy* é o arquivo `devopsnapratica.war` do módulo `combined`, que se encontra no diretório `combined/target`.

Antes de fazer o *deploy* do arquivo `war`, a última configuração necessária no Tomcat é a definição das fontes de dado, ou `DataSource`, que a aplicação vai acessar para conectar com o banco de dados. Se tais definições estivessem dentro do artefato em si, nós precisaríamos realizar um novo *build* toda vez que os parâmetros de conexão com o banco de dados mudassem, ou caso quiséssemos rodar a mesma aplicação em ambientes diferentes.

Em vez disso, a aplicação depende de recursos abstratos através da especificação JNDI (*Java Naming and Directory Interface*). Estes recursos precisam ser configurados e expostos pelo *container*, possibilitando a alteração sem necessidade de um novo *build*.

A loja virtual precisa de três fontes de dados no JNDI, chamadas `jdbc/web`, `jdbc/secure` e `jdbc/storage`. No nosso caso, usaremos o mesmo banco de dados para os três recursos, definidos no arquivo `context.xml`:

```
vagrant@web$ sudo nano /var/lib/tomcat7/conf/context.xml
```

As configurações relevantes são os três elementos `<Resource>` e a diferença entre eles é apenas o atributo `name`:

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
  <WatchedResource>WEB-INF/web.xml</WatchedResource>

  <Resource name="jdbc/web" auth="Container"
      type="javax.sql.DataSource" maxActive="100" maxIdle="30"
      maxWait="10000" username="loja" password="lojasecret"
      driverClassName="com.mysql.jdbc.Driver"
```

```
      url="jdbc:mysql://192.168.33.10:3306/loja_schema"/>

  <Resource name="jdbc/secure" auth="Container"
      type="javax.sql.DataSource" maxActive="100" maxIdle="30"
      maxWait="10000" username="loja" password="lojasecret"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://192.168.33.10:3306/loja_schema"/>

  <Resource name="jdbc/storage" auth="Container"
      type="javax.sql.DataSource" maxActive="100" maxIdle="30"
      maxWait="10000" username="loja" password="lojasecret"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://192.168.33.10:3306/loja_schema"/>
</Context>
```

Para fazer o *deploy* da loja virtual em si, simplesmente precisamos copiar o artefato `war` para o lugar certo e o Tomcat fará o necessário para colocar a loja virtual no ar:

```
vagrant@web$ cd ~/loja-virtual-devops
vagrant@web$ sudo cp combined/target/devopsnapratica.war \
> /var/lib/tomcat7/webapps
```

Assim como o *build* demorou um pouco, o processo de *deploy* também vai levar alguns minutos na primeira vez, pois a aplicação criará as tabelas e populará os dados usando o Hibernate. Você pode acompanhar o processo de *deploy* olhando para os logs do Tomcat. O comando `tail -f` é útil nesse caso, pois ele mostra em tempo real tudo que está acontecendo no log:

```
vagrant@web$ tail -f /var/lib/tomcat7/logs/catalina.out
Mar 15, 2014 2:08:51 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Mar 15, 2014 2:08:51 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8443"]
Mar 15, 2014 2:08:51 AM org.apache.catalina.startup.Catalina
                                                          start
INFO: Server startup in 408 ms
...
```

O *deploy* termina assim que você perceber que as atividades no

log pararam. Para terminar o comando `tail`, você pode usar `Ctrl+C`. Agora é só testar que tudo está funcionando: abra uma nova janela no navegador e digite a URL `http://192.168.33.12:8080/devopsnapratica/`. Parabéns, a loja virtual está em produção!

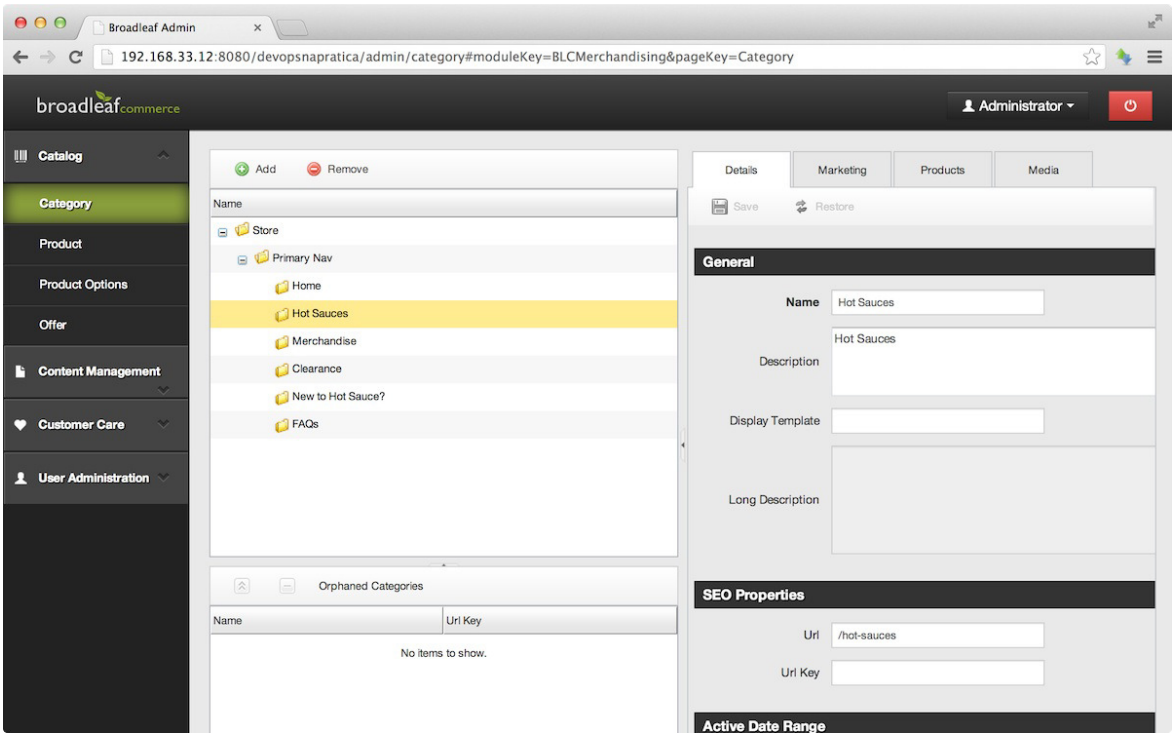

Figura 2.3: Página de administração da loja virtual

Você também pode acessar a página de administração que se encontra na URL `http://192.168.33.12:8080/devopsnapratica/admin/` (a barra no final da URL é importante). Entrando com o usuário `admin` e senha `admin`, você pode navegar pelas páginas de administração e gerenciar o catálogo de produtos, categorias, o conteúdo das páginas da loja virtual, gerenciar pedidos, assim como usuários e permissões.

Agora que estamos em produção e conquistamos a última

milha, podemos focar nos princípios e práticas de DevOps necessários para operar nossa aplicação em produção, ao mesmo tempo permitindo que novas mudanças sejam introduzidas de forma confiável.