附录 A 在桌面上运行云

本章内容

- 列出运行本书中代码所需的软件
- 从每个章节的 GitHub 下载源代码
- 使用 Maven 编译和打包源代码
- 构建和配置每章使用的 Docker 镜像
- 使用 Docker Compose 启动由该构建编译的 Docker 镜像

在编写本书中的代码示例并选择部署代码所需的运行时技术时，我有两个目标。第一个目标是确保代码示例易于使用且易于设置。请记住，一个微服务应用程序有多个动态部分，如果没有一些预见的话，以最小的工作量设置这些部分完整地运行对于读者来说可能是困难的。

第二个目标是让每一章都是完全独立的，这样就可以选择本书中的任何章节，并且有一个完整的运行时环境，它封装了在章节中运行代码示例所需的所有服务和软件，而不依赖于其他章节。

为此，你将在本书的每一章中看到以下使用的技术和模式：

- 所有项目都使用 Apache Maven(http://maven.apache.org)作为章节的构建工具。每个服务都是使用 Maven 项目结构构建的，每个服务结构都是一章一章的放置。
- 本章中开发的所有服务都编译为 Docker(http://docker.io)容器镜像。Docker 是一个非常出色的运行时虚拟化引擎，运行在 Windows，OS X 和 Linux 上。 使用 Docker，我可以在桌面上构建一个完整的运行时环境，包括应用程序服务和支持

服务所需的所有基础设施。此外，与更多专有虚拟化技术不同，Docker 可轻松跨多个云提供商移植。

我使用 Spotify 的 Docker Maven 插件(https://github.com/spotify/docker-maven-plugin)来将 Docker 容器的构建与 Maven 构建过程集成在一起。

- 为了在编译成 Docker 镜像之后启动服务，我使用 Docker Compose 来启动一组服务。我有意避免使用更复杂的 Docker 编排工具，例如 Kubernetes (https://github.com/kubernetes/kubernetes)或 Mesos(http://mesos.apache.org/)，以保持章节示例的直观性和可移植性。Docker 镜像的所有配置都使用简单的 shell 脚本完成。

A.1 所需的软件

要为所有章节构建软件，你需要在桌面上安装以下软件。请注意，这些是我为这本书所用的软件的版本。该软件可能与其他版本一起工作，但是这是我用下面软件构建的代码：

- *Apache Maven (http://apache.maven.org)*：我使用是 3.3.9 版本的 Maven。我选择了 Maven，因为当其他构建工具（如 Gradle）非常流行时，Maven 仍然是Java生态系统中使用的主要构建工具。本书中的所有代码示例都是使用Java 1.8版本进行编译的。
- *Docker (http://docker.com)*：我使用 Docker V1.12 构建了本书中的代码示例。本书中的代码示例将与早期版本的 Docker 一起使用，但是如果你想要在早期版本的 Docker 中使用此代码，则可能必须切换到版本 1 的 docker-compose 链接格式。
- *Git Client (http://git-scm.com)* 本书的所有源代码都存储在 GitHub 存储库中。

对于本书，我使用了 Git Client 的 2.8.4 版本。

我不打算介绍如何安装这些组件。列表中列出的每个软件包都具有简单的安装说明，应该可以用最小的工作量安装。

A.2 从 GitHub 下载项目

本书的所有源代码都在我的 GitHub 存储库中(http://github.com/carnellj)。书中的每一章都有自己的源代码库。以下是本书中使用的所有 GitHub 存储库的列表：

- 第 1 章 欢迎来到 Cloud 和 Spring：http://github.com/carnellj/spmia-chapter1
- 第 2 章 微服务介绍：http://github.com/carnellj/spmiachapter2
- 第 3 章 Spring Cloud Config：http://github.com/carnellj/spmiachapter3 和 http://github.com/carnellj/config-repo
- 第 4 章 Spring Cloud/Eureka：http://github.com/carnellj/spmiachapter4
- 第 5 章 Spring Cloud/Hystrix：http://github.com/carnellj/spmiachapter5
- 第 6 章 Spring Cloud/Zuul：http://github.com/carnellj/spmia-chapter6
- 第 7 章 Spring Cloud/Oauth2：http://github.com/carnellj/spmiachapter7
- 第 8 章 Spring Cloud Stream：http://github.com/carnellj/spmia-chapter8
- 第 9 章 Spring Cloud Sleuth：http://github.com/carnellj/spmia-chapter9
- 第 10 章 部署：http://github.com/carnellj/spmia-chapter10 和 http://github.com/carnellj/chapter-10-platform-tests

使用 GitHub，你可以使用 Web UI 将文件作为 zip 文件下载。每个 GitHub 仓库都会有一个下载按钮。图 A.1 显示了下载按钮在第 1 章的 GitHub 仓库中的位置。

如果你是一个命令行用户，你可以安装 git 客户端并克隆项目。例如，如果你想使用

git 客户端从 GitHub 下载第 1 章，则可以打开命令行并执行以下命令：

```
git clone https://github.com/carnellj/spmia-chapter1.git
```

这将把所有的第 1 章项目文件下载到你运行 git 命令的目录中的一个名为 spmiachapter1 的目录中。

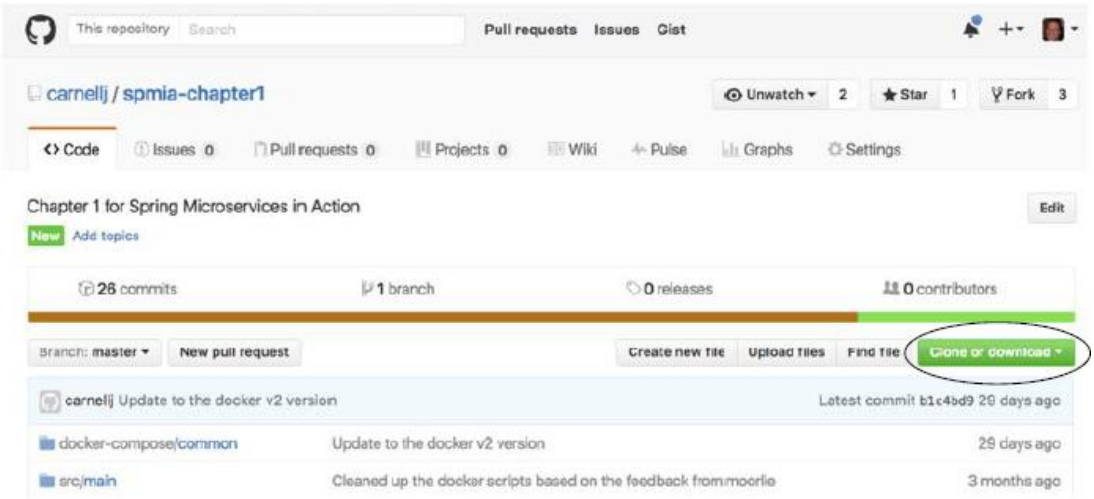

图 A.1 GitHub 用户界面允许你将一个项目作为一个 zip 文件下载。

A.3 每章的剖析

书中的每一章都有一个或多个与之相关的服务。章节中的每个服务都有自己的项目目录。例如，如果你看第 6 章(http://github.com/carnellj/spmia-chapter6)，你会发现里面有七个服务。这些服务是：

- confsvr：Spring Cloud Config server
- eurekasvr：Spring Cloud/with Eureka
- licensing-service：Eagle Eye 许可服务
- organization-service：Eagle 组织服务
- orgservice-new：EagleEye 服务的新测试版本
- specialroutes-service：A/B 路由服务

- zuulsvr：EagleEye Zuul 服务

每章中的每个服务目录都是基于 Maven 的构建项目。每个项目里面都有一个 src/main 目录，其中包含以下子目录：

- java：该目录包含用于构建服务的 Java 源代码。
- docker：这个目录包含两个需要为每个服务构建一个 Docker 镜像的文件。第一个文件将总是被称为 Dockerfile，并包含 Docker 用于构建 Docker 镜像的逐步说明。第二个文件 run.sh 是一个在 Docker 容器中运行的自定义 Bash 脚本。此脚本可确保在某些密钥依赖（数据库已启动并正在运行）可用之前服务不会启动。
- resources：资源目录包含所有服务的 application.yml 文件。虽然应用程序配置存储在 Spring Cloud Config 中，但所有服务都具有本地存储在 application.yml 中的配置。此外，资源目录将包含一个 schema.sql 文件，其中包含用于创建表的所有 SQL 命令以及将这些服务的数据预加载到 Postgres 数据库中。

A.4 构建和编译项目

因为本书中的所有章节都遵循相同的结构，并使用 Maven 作为构建工具，所以构建源代码变得非常简单。每一章都在目录的根目录中有一个 pom.xml，作为所有子章节的父 pom。如果要编译源代码并为单个章节中的所有项目构建 Docker 镜像，则需要在本章的根目录下运行以下代码：

```
mvn clean package docker:build
```

这将在每个服务目录中执行 Maven pom.xml 文件。它也将在本地构建 Docker 镜像。

如果要在本章中构建单个服务，则可以切换到特定的服务目录并运行 mvn clean package docker:build 命令。

A.5 构建 Docker 镜像

在构建过程中，本书中的所有服务都打包为 Docker 镜像。这个过程由 Spotify Maven 插件执行。有关此插件的实例，您可以查看第 3 章许可服务的 pom.xml 文件（chapter3/licensing-service)。以下清单显示了在每个服务的 pom.xml 文件中配置此插件的 XML 片段。

清单 A.1 Spotify Docker Maven 插件用于创建 Docker 镜像

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.4.10</version>
  <configuration>
    <imageName>
      ${docker.image.name}:[ca]${docker.image.tag}
    </imageName>
    <dockerDirectory>
      ${basedir}/target/dockerfile
    </dockerDirectory>
    <resources>
      <resource>
        <targetPath>/</targetPath>
        <directory>${project.build.directory}</directory>
        <include>${project.build.finalName}.jar</include>
      </resource>
    </resources>
  </configuration>
</plugin>
```

①每个创建的 Docker 镜像都会有一个与之关联的标记。Spotify 插件将使用 ${docker.image.tag} 标签中定义的任何名称命名创建的镜像。

②在本书中创建的所有 Docker 镜像都是使用 Dockerfile。Dockerfile 用于逐步说明如何配置 Docker 镜像。

③当 Spotify 插件被执行时，它会将服务的可执行 jar 复制到 Docker 镜像。

XML 片段做了三件事：

- 它将服务的可执行 jar 以及 src/main/docker 目录的内容复制到 target/docker。
- 它执行 target/docker 目录中定义的 Dockerfile。Dockerfile 是每当为该服务设置新的 Docker 镜像时执行的命令列表。
- 它将 Docker 镜像推送到安装 Docker 时安装的本地 Docker 镜像库。

以下清单显示了你的许可服务中的 Dockerfile 的内容。

清单 A.2 Dockerfile 准备 Docker 镜像

①这是你要在 Docker 运行时使用的 Linux Docker 映像。此安装针对 Java 应用程序进行了优化。

②你可以安装 nc（netcat），你将使用这个实用程序 ping 相关服务以查看是否已启动。

③Docker ADD 命令将可执行 JAR 从本地文件系统复制到 Docker 镜像。

④你可以添加一个自定义的 BASH shell 脚本来监控服务依赖，然后启动实际的服务。

```
FROM openjdk:8-jdk-alpine
RUN apk update && apk upgrade && apk add netcat-openbsd
RUN mkdir -p /usr/local/licensingservice
ADD licensing-service-0.0.1-SNAPSHOT.jar /usr/local/licensingservice/
ADD run.sh run.sh
RUN chmod +x run.sh
CMD ./run.sh
```

在此清单的 Dockerfile 中，你将使用 Alpine Linux（https://alpinelinux.org/）提供的实例。Alpine Linux 是一个经常用来构建 Docker 镜像的小型 Linux 发行版。你正在使用的 Alpine Linux 镜像已经安装了 Java JDK。

在配置 Docker 镜像时，你将安装名为 nc 的命令行实用程序。nc 命令用于 ping 一台服务器并查看是否有特定的端口在线。你将在 run.sh 命令脚本中使用它，以确保在启动服务之前，所有依赖服务（例如数据库和 Spring Cloud Config 服务）都已启动。nc 命令通过监视依赖服务监听的端口来执行此操作。nc 的安装是通过 RUN apk update && apk upgrade && apk add netcat-openbsd，使用 Docker Compose 运行服务。

接下来，你的 Dockerfile 将为许可服务的可执行 jar 文件创建一个目录，然后将 jar 文件从本地文件系统复制到在 Docker 镜像上创建的目录。这全部通过 ADD licensing-service-0.0.1-SNAPSHOT.jar/usr/local/licensingservice/完成。

配置过程的下一步是通过 ADD 命令安装 run.sh 脚本。run.sh 脚本是我写的一个自定义脚本，在启动 Docker 镜像时启动目标服务。它使用 nc 命令来监听许可服务所需的任何关键服务依赖的端口，然后阻塞，直到这些依赖启动。

以下清单显示了如何使用 run.sh 来启动许可服务。

清单 A.3 用于启动许可服务的 run.sh 脚本

```
#!/bin/sh
echo "********************************************************"
```

```
echo "Waiting for the configuration server to start on port
      $CONFIGSERVER_PORT"
echo "*********************************************************"
while ! `nc -z configserver $CONFIGSERVER_PORT `;
      [ca]do sleep 3; done
echo ">>>>>>>>>>>> Configuration Server has started"
echo "*********************************************************"
echo "Waiting for the database server to start on port $DATABASESERVER_PORT"
echo "*********************************************************"
while ! `nc -z database $DATABASESERVER_PORT`; do sleep 3; done
echo ">>>>>>>>>>>> Database Server has started"
echo "*********************************************************"
echo "Starting License Server with Configuration Service :
      $CONFIGSERVER_URI";
echo "*********************************************************"
java -Dspring.cloud.config.uri=$CONFIGSERVER_URI \
      -Dspring.profiles.active=$PROFILE \
      -jar /usr/local/licensingservice/licensing-service-0.0.1-SNAPSHOT.jar
```

①在继续尝试启动服务之前，run.sh 脚本会等待依赖服务的端口处于打开状态。

②通过使用 Java 调用 Dockerfile 脚本安装的可执行 jar 来启动许可服务。$<<VARIABLE_NAME>> 表示传递给 Docker 镜像的环境变量。

一旦将 run.sh 命令复制到许可服务 Docker 镜像，CMD ./run.sh Docker 命令用于告知 Docker 在实际镜像启动时执行 run.sh 启动脚本。

> ***注意***：我给你一个关于 Docker 如何提供镜像的高级概述。如果你想深入了解 Docker，我建议你查看 Jeff Nickoloff 的《Docker in Action》（Manning，2016）或 Adrian Mouat 的《Using Docker》（O'Reilly，2016）。这两本书都是优秀的 Docker 资源。

A.6 使用 Docker Compose 启动服务

Maven 构建完成后，现在可以使用 Docker Compose 启动本章的所有服务。Docker Compose 作为 Docker 安装过程的一部分进行安装。这是一个服务编排工具，它允许你将服务定义为一个组，然后作为一个单元一起启动。Docker Compose 包含了同时为每个服务定义环境变量的功能。

Docker Compose 使用 YAML 文件来定义将要启动的服务。本书中的每一章都有一个名为“<<章节>>/docker/common/docker-compose.yml”的文件。该文件包含用于在

本章中启动服务的服务定义。我们来看看第 3 章中使用的 docker-compose.yml 文件。下面的清单显示了这个文件的内容。

清单 A.4 docker-compose.yml 文件定义了要启动的服务

```
version: '2'
services:
  configserver:
    image: johncarnell/tmx-confsvr:chapter3
    ports:
      - "8888:8888"
    environment:
      ENCRYPT_KEY: "IMSYMMETRIC"
  database:
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "p0stgr@s"
      POSTGRES_DB: "eagle_eye_local"
  licensingservice:
    image: johncarnell/tmx-licensing-service:chapter3
    ports:
      - "8080:8080"
    environment:
      PROFILE: "default"
      CONFIGSERVER_URI: "http://configserver:8888"
      CONFIGSERVER_PORT: "8888"
      DATABASESERVER_PORT: "5432"
      ENCRYPT_KEY: "IMSYMMETRIC"
```

①每个正在启动的服务都有一个应用于它的标签。这将成为 Docker 实例启动时的 DNS 条目，以及其他服务如何访问它。

②Docker Compose 将首先尝试在本地 Docker 存储库中查找要启动的目标镜像。如果找不到，它将检查中央 Docker hub (http://hub.docker.com)。

③这个入口定义了启动的 Docker 容器上的端口号，这个端口号将暴露给外部世界。

④环境标签用于将环境变量传递到正在启动的 Docker 镜像。在这种情况下，ENCRYPT_KEY 环境变量将在正在启动的 Docker 镜像上设置。

⑤这是一个如何在 Docker Compose 文件（configserver）的一部分中定义的服务用作另一个服务中的 DNS 名称的示例。

在清单 A.4 的 docker-compose.yml 中，我们看到三个服务被定义（配置服务器、数据库和许可服务）。每个服务都有一个使用镜像标签定义的 Docker 镜像。当每个服务启动时，它将通过端口标签暴露端口，然后通过环境标签将环境变量传递到正在启动的 Docker 容器。

继续，通过从 GitHub 下拉的 chapter 目录的根目录执行以下命令启动你的 Docker 容器：

```
docker-compose –f docker/common/docker-compose.yml up
```

当这个命令发出时，docker-compose 启动 docker-compose.yml 文件中定义的所有服务。

每个服务都将其标准输出打印到控制台。图 A.2 显示了第 3 章中 docker-compose.yml 文件的输出。

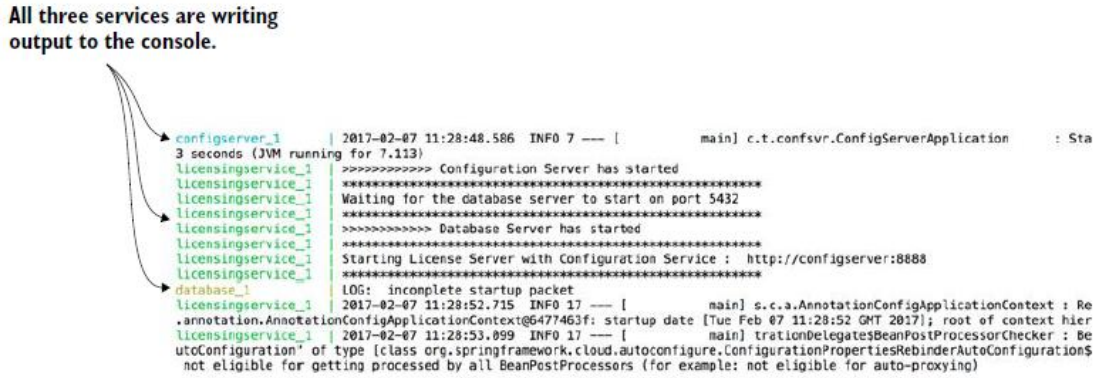

图 A.2 启动的 Docker 容器的所有输出都写入标准输出。

① *All three services are writing output to the console.*

所有这三个服务都将输出写入控制台。

提示：通过使用 Docker Compose 启动的服务写出的每行标准输出都将服务的名称打印到标准输出。当你启动 Docker Compose 业务流程时，发现打印出来的错误可能会很痛苦。如果你想查看基于 Docker 的服务的输出，请使用-d 选项（docker-compose -f docker/common/docker-compose.yml up –d)以 detached 模式启动 docker-compose 命令。然后，你可以通过使用 logs 选项(docker-compose -f docker/common/docker-compose.yml logs -f licensingservice)执行 docker-compose 命令来查看该容器的特定日志。

本书所使用的所有 Docker 容器都是短暂的。它们在启动和停止时不会保留自己的状态。记住这一点，如果你开始使用代码，你看到你的数据在重新启动你的容器后消失。如果你想让你的 Postgres 数据库在容器的启动和停止之间保持不变，我会建议你看看关于 Postgres Docker 的注意事项(https://hub.docker.com/_/postgres/)。