第8章 Docker实战

和前些章节不同，本章将介绍Docker的项目开发流程，重在应用和实例。在开始本节内容之前，首先来了解一下Docker镜像的开发过程，如图8-1所示。

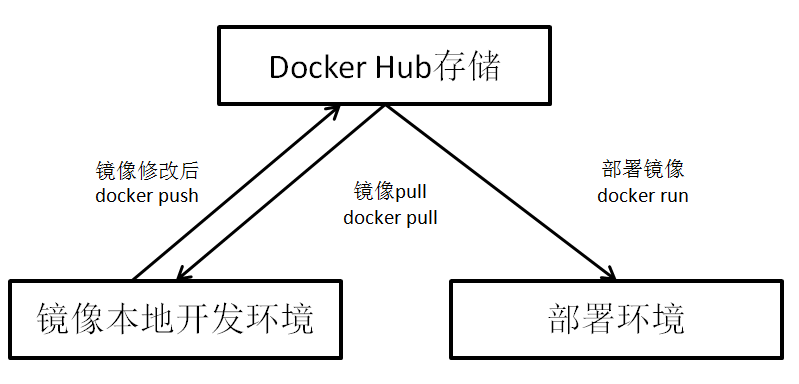


图8-1 Docker镜像开发流程

从前些章节已知道，Docker Hub是用来存储Docker镜像的，Docker本地开发环境是用来做开发镜像的，通常的开发流程是先从Docker Hub中获取到基础镜像，之后在这个镜像的基础上做开发以满足一定的需求或提供某种服务，并由此产生新的镜像，然后就可以push到Docker Hub中，且本地无需保留镜像的备份（相关的源码、Dockerfile还是需要留的），这样开发工作就完成了。部署的时候直接将开发好的镜像pull下来，然后使用docker run命令部署，或者借助其他部署工具（比如docker-compose）来部署。

本章首先会介绍Dockerfile的基本知识和语法，毕竟容器构建是离不开Dockerfile的；而后将创建一个基于Web服务器的工程，并为该Web前端挂上后端务器，后端由几个独立的模块组成。在该工程中，每个模块都会是一个镜像，这些镜像协同工作，通过REST API（HTTP接口）通信，相互独立，共同完成一个Web站点前台和后台工作。该工程使用docker-compose同时管理几个容器，是Docker镜像管理的典型范例。

## Dockerfile简介

接下来将一起来探讨Dockerfile的构成和语法。通过本节的学习，你可以轻松地写出自己需要的Dockerfile。本节由浅入深，由一个简单例子开始，之后会一一介绍Dockerfile的各种命令，不会让你觉得很难入手。

### 8.1.1一个简单的例子

Dockerfile的注释都是以“#”开始的，每一行是一个指令。一般情况下，Dockerfile由4部分组成：基础镜像信息、维护者信息、镜像操作指令和容器启动指令。下面是一个Nginx服务镜像Dockerfile文件的内容：

# This Dockerfile uses the ubuntu image

# VERSION 2 - EDITION 1

# Author: tester

# Command format: Instruction [arguments / command] ..

# Base image to use, this must be set as the first line

**FROM ubuntu**

# Maintainer: tester < tester at email.com> (@docker\_user)

MAINTAINER tester tester@huawei.com

# Commands to update the image

RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/sources.list

RUN apt-get update && apt-get install -y nginx

**RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf**

# Commands when creating a new container

CMD /usr/sbin/nginx

需要说明的是，如果使用Dockerfile来构建镜像，Dockerfile的第一条有效信息（注释除外）必须是基础镜像信息，维护者信息紧随其后。而镜像操作指令则在维护信息之后，因为操作指令的不同，自然就会构建出千差万别的镜像来。最后是镜像启动指令，它被用作设置镜像的默认启动命令。

### 8.1.2 Dockerfile指令

从上面例子中的“FROM ubuntu”，“RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf”等语句不难看出，Dockerfile指令的一般格式为 INSTRUCTION arguments，而这些指令包括 FROM、MAINTAINER、RUN 等。下面将把最为常用的几个指令介绍下：

* FROM指令

**格式为 FROM <image>或FROM <image>:<tag>**

Dockerfile的第一条必须是FROM指令，用来指定要制作的镜像继承自哪个镜像。需要说明的是，可以在Dockerfile中写多个FROM指令来构建复杂的镜像。

* MAINTAINER指令

**格式为 MAINTAINER <name>**

用来指定维护者信息。

* RUN 指令

**格式为 RUN <command> 或 RUN ["executable", "param1", "param2"…]**

该指令是用来执行shell命令的，当解析Dockerfile时，遇到RUN指令，Docker会将该指令翻译为“/bin/sh –c “xxx””，其中xxx为RUN后边的shell命令。

* EXPOSE 指令

**格式为 EXPOSE <port> [<port>...]**

该指令用来将容器中的端口号暴露出来，只有暴露出来的端口号，才可以在容器启动时，通过“docker run –p”命令实现和服务器端口的映射。

* CMD指令

有三种格式：

CMD ["executable","param1","param2"] 使用 exec 执行，推荐方式；

CMD command param1 param2 在 /bin/sh 中执行，提供给需要交互的应用；

CMD ["param1","param2"] 提供给 ENTRYPOINT 的默认参数。

指定启动容器时执行的命令，每个Dockerfile只能有一条CMD 指令。如果指定了多条CMD指令，只有最后一条会被执行。值得说明的是，如果用户启动容器时指定了运行的命令，则会覆盖掉 CMD 指定的命令。可以和 ENTRYPOINT指令对比。

* ENTRYPOINT指令

**两种格式：**

**ENTRYPOINT ["executable", "param1", "param2"]**

**ENTRYPOINT command param1 param2（shell中执行）。**

每个 Dockerfile 中只能有一个 ENTRYPOINT，当指定多个时，只有最后一个起效。

注意 ENTRYPOINT指令是用来配置容器启动后要执行的命令的，并且不可被docker run提供的参数覆盖。所以建议使用CMD指令，而不是ENTRYPOINT指令。

* VOLUME 指令

**格式为 VOLUME ["/data"]。**

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库或需要永久保存的数据。如果和host共享目录，Dockerfile中必须先创建一个挂载点，然后在启动容器的时候通过“docker run –v $HOSTPATH:$CONTAINERPATH”来挂载，其中CONTAINERPATH就是创建的挂载点。

* ENV指令

**格式为 ENV <key> <value>。**

指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。

* ADD指令

**格式为 ADD <src> <dest>。**

该指令将复制指定的 <src> 到容器中的 <dest>。 其中 <src> 可以是Dockerfile所在目录的一个相对路径；也可以是一个 URL；还可以是一个 tar 文件（自动解压为目录）。

* **COPY指令**

**格式为 COPY <src> <dest>。**

复制本地主机的 <src>（为 Dockerfile 所在目录的相对路径）到容器中的 <dest>。当使用本地目录为源目录时，推荐使用 COPY。

### 8.1.3 再谈Docker镜像制作

使用docker build命令并给定一个Dockerfile可以制作一个镜像（详见docker build命令），那么Docker是怎么把这个镜像制作出来的呢？因为本书的读者可能并不是开发Docker或者阅读Docker源码的人，所以本节就抛开源码以实例的方式将Docker制作镜像的过程简单地呈现出来。

首先要准备一个基础镜像比如busybox:latest，这个镜像可以从官方pull下来。然后写一个简单的Dockerfile，如下：

# This Dockerfile uses the busybox image

# VERSION 1 - EDITION 1

# Author: tester

FROM busybox:latest

RUN date;sleep 100;date

RUN echo “abc” > /mytest

RUN date;sleep 100;date

CMD /bin/sh

现在用build命令去制作镜像，名称为busybox:v1，如下：

# docker build busybox:v1 .

当Docker正在制作这个镜像时，我们可以通过docker ps看下到底发生了什么：

# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c5db3d7ca9b9 d57d38e51fb32b790a1e65c32e639f1515a528d3f568cdae05067d057acb8817 "/bin/sh -c 'date;sl 9 seconds ago Up 8 seconds mad\_yalow

从中可以看到，在镜像制作的时候，Docker会启动一个容器，并在该容器里制作这个镜像。接下来通过docker exec命令，去容器中看看到底发生了什么：

# docker exec -ti c5db3d7ca9b9 sh

root@c5db3d7ca9b9:/ # ps -ef

PID USER COMMAND

1 root /bin/sh -c date;sleep 100;date

6 root sleep 100

16 root sh

21 root ps -ef

从上边的命令中可以看到，Docker制作镜像的RUN命令都是在容器中执行的。如果我们等上100秒，不停的ps，就会按顺序抓到RUN的每一条命令。

从上面我们可以看出，Docker在制作镜像的时候，其操作顺序为：

* **解析Dockerfile，并找到基础镜像（可能会从Docker Hub上下载）；**
* **以基础镜像为基础创建一个容器；**
* **在容器中顺序执行Dockerfile中的命令；**
* **如果不是RUN命令，比如ENV命令，记录下来以便启动的时候执行；**
* **属性命令记录在Image的属性中；**
* **所有指令执行完后，commit该容器为新的镜像。**

当然这里说的几个步骤，只是简单地阐述了镜像制作的过程，其实中间涉及很多复杂操作，比如分层存储等。有兴趣的读者，可以去阅读相关源码。

## 基于Docker的Web应用和发布

到这里，大家已经学习了很多Docker知识，本节将基于Docker部署一个HTTPS的Web服务器，以此为例展示Docker的工作方式。我们将把一个Web的工程项目部署为一个以Tomcat为服务器的HTTPS的Web站点。其原理和在普通服务器上部署类似，只需要将自己开发的软件包放到Tomcat工程目录下即可。不同的是，在把自己开发的东西放到镜像中时，Tomcat服务器要启在容器中。

### 8.2.1 选择基础镜像

部署一个Web站点，其实有很多选择，比如使用Nginx、Apache等，在这里以Tomcat为例。首先需要从镜像库中选择Tomcat的基础镜像，如果没有自己的镜像库，可以从Docker Hub中获取。Docker Hub为我们提供了丰富的版本，包括不同的JRE版本：

# docker images | grep tomcat

tomcat 8-jre7 71093fb71661 6 days ago 347.7 MB

tomcat 7-jre8 4dcef5c50d60 6 days ago 494.3 MB

tomcat 7.0.63-jre8 4dcef5c50d60 6 days ago 494.3 MB

tomcat 7.0-jre8 dcef5c50d60 6 days ago 494.3 MB

tomcat 7.0-jre7 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 7 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 7.0.63 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 7-jre7 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 7.0.63-jre7 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 7.0 f1fb45bb5af9 6 days ago 348.2 MB

tomcat 6.0.44-jre8 221207f53791 6 days ago 491.9 MB

tomcat 6.0-jre8 21207f53791 6 days ago 491.9 MB

tomcat 6-jre8 221207f53791 6 days ago 491.9 MB

tomcat 6-jre7 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 6.0.44-jre7 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 6.0.44 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 6.0 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 6.0-jre7 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 6 8c4e1a4ca737 6 days ago 345.8 MB

tomcat 7.0.62-jre8 55f2cc1815fa 4 weeks ago 494.5 MB

tomcat 7.0.62 611a2cc9d3d0 4 weeks ago 348.4 MB

tomcat 7.0.62-jre7 611a2cc9d3d0 4 weeks ago 348.4 MB

tomcat 7.0.61-jre7 1e8d1fcedf1f 11 weeks ago 349.7 MB

tomcat 7.0.61 1e8d1fcedf1f 11 weeks ago 349.7 MB

tomcat 6.0.43-jre7 1e36cc8da5d2 11 weeks ago 347.3 MB

tomcat 6.0.43 1e36cc8da5d2 11 weeks ago 347.3 MB

tomcat 7.0.61-jre8 9e11bd76affb 11 weeks ago 496.4 MB

tomcat 6.0.43-jre8 b8d56384a231 11 weeks ago 494 MB

tomcat 7.0.59-jre8 c54c272df5dc 3 months ago 493.1 MB

tomcat 7.0.59 93f4bcb2ca91 3 months ago 346.1 MB

tomcat 7.0.59-jre7 93f4bcb2ca91 3 months ago 346.1 MB

这里选择Tomcat 7.0-jre8.版本，并在本地将这个镜像Pull下来：

# docker pull tomcat:7.0-jre8

或者

# docker pull –a tomcat

提示 后者会把Tomcat的所有版本都pull下来，笔者就是通过pull –a 参数把所以tomcat镜像拉到本地的。

### 8.2.2 制作HTTPS服务器镜像

默认的基础镜像是提供的HTTP服务器，我们需要部署的是一个HTTPS Web站点，所以第一步需要先让HTTP的Tomcat支持HTTPS。我们知道，HTTPS是需要证书的，因为本书不是介绍HTTPS或者Web安全，所以这里只是简单给出生成证书的方法，然后会详细介绍在容器中如何使用这些证书。

#### 1. 生成HTTPS需要的证书

下面是生成ssl证书的方法，供读者参考。

root@ubuntu:~/work# mkdir ssl

root@ubuntu:~/work# cd ssl/

root@ubuntu:~/work/ssl# keytool -genkey -alias tomcat -keyalg RSA -keystore tomcat.keystore

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]: Tester

What is the name of your organizational unit?

[Unknown]: Huawei

What is the name of your organization?

[Unknown]: Huawei

What is the name of your City or Locality?

[Unknown]: Hangzhou

What is the name of your State or Province?

[Unknown]: Zhejiang

What is the two-letter country code for this unit?

[Unknown]: CN

Is CN=Tester, OU=Huawei, O=Huawei, L=Hangzhou, ST=Zhejiang, C=CN correct?

[no]: yes

Enter key password for <tomcat>

(RETURN if same as keystore password):

Re-enter new password:

root@ubuntu:~/work/ssl# ls

tomcat.keystore

这样，就生成了一个SSL证书，并保存在 tomcat.keystore里。接下来可以将证书导入到镜像中。

#### 2. 把证书导入到镜像中

将一个文件导入到镜像中有很多种方式，这里先介绍一种最简单的，也是最实用的方式（后续章节会介绍其他导入方式），如下。

# docker run -ti -v $(pwd):/tmp tomcat:7.0-jre8 bash

root@887a6ec5aeec:/usr/local/tomcat# ls

LICENSE NOTICE RELEASE-NOTES RUNNING.txt bin conf lib logs temp webapps work

root@887a6ec5aeec:/usr/local/tomcat# ls /tmp/

tomcat.keystore

root@887a6ec5aeec:/usr/local/tomcat# mkdir keys

root@887a6ec5aeec:/usr/local/tomcat# cp /tmp/tomcat.keystore keys/

root@887a6ec5aeec:/usr/local/tomcat#

docker run –v参数可以将host的目录动态挂载到容器中的某一路径上，本例中通过-v将当前目录挂载到容器中的/tmp目录，从而达到目录共享的目的，实现文件从服务器到容器的复制。

#### 3. 修改Tomcat的配置并commit

下面修改端口号为8080的Tomcat配置项，Tomcat的默认配置文件存在/usr/local/tomcat/conf/web.xml里，相信使用过Tomcat的读者对此并不陌生。

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443"

SSLEnabled="true" scheme="https" secure="true" clientAuth="false" keystoreFile="/usr/local/tomcat/keys/tomcat.keystore" keystorePass="test" sslProtocol="TLS"/>

修改完毕后，可以commit这个容器，命令如下：

# docker ps –a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

50f2c2d7e873 official/tomcat: 7.0-jre8 “bash” 10 minutes ago Up 8080/tcp condescending\_brown

# docker commit 50f2c2d7e873 tomcat:https

d798c341b5d01c8e6beb860ac3ccb766d2de98c3c8758c916791b7d8dfee86e5

#

提示 这里只是告诉读者如何对于镜像进行简单的修改，但是对于改造相对较大的情况，不建议使用这种方式，应该使用Dockerfile来构建新的镜像。后边的例子全部都是使用Dockerfile来构建新镜像。

#### 4. 验证基础镜像

要想验证基础镜像是否能正常工作，只需要运行起来，看看Tomcat首页是否能正常打开即可。命令如下：

# docker run –p 80:8080 tomcat:https

在上面的命令中，-p参数表示端口映射，80:8080表示将host主机上的80端口，映射到容器中的8080端口，这里两者之所以不一样是为了让读者区分开这两个端口，实际上，也可以是8080:8080。

当docker run命令不加任何command的时候，默认执行容器的ENTRYPOINT指令，该指令是在容器制作的时候指定的。所以建议在制作镜像的时候，为服务性镜像提供默认的ENTRYPOINT。

另外我们的证书是自己生成的，所以浏览器会提示不安全，此时只需点击继续访问即可，真正部署上线时，可以购买专门的证书。

好了，服务起来了，我们可以通过Web访问了，如图7-2所示，Tomcat服务器已经完全启动成功。

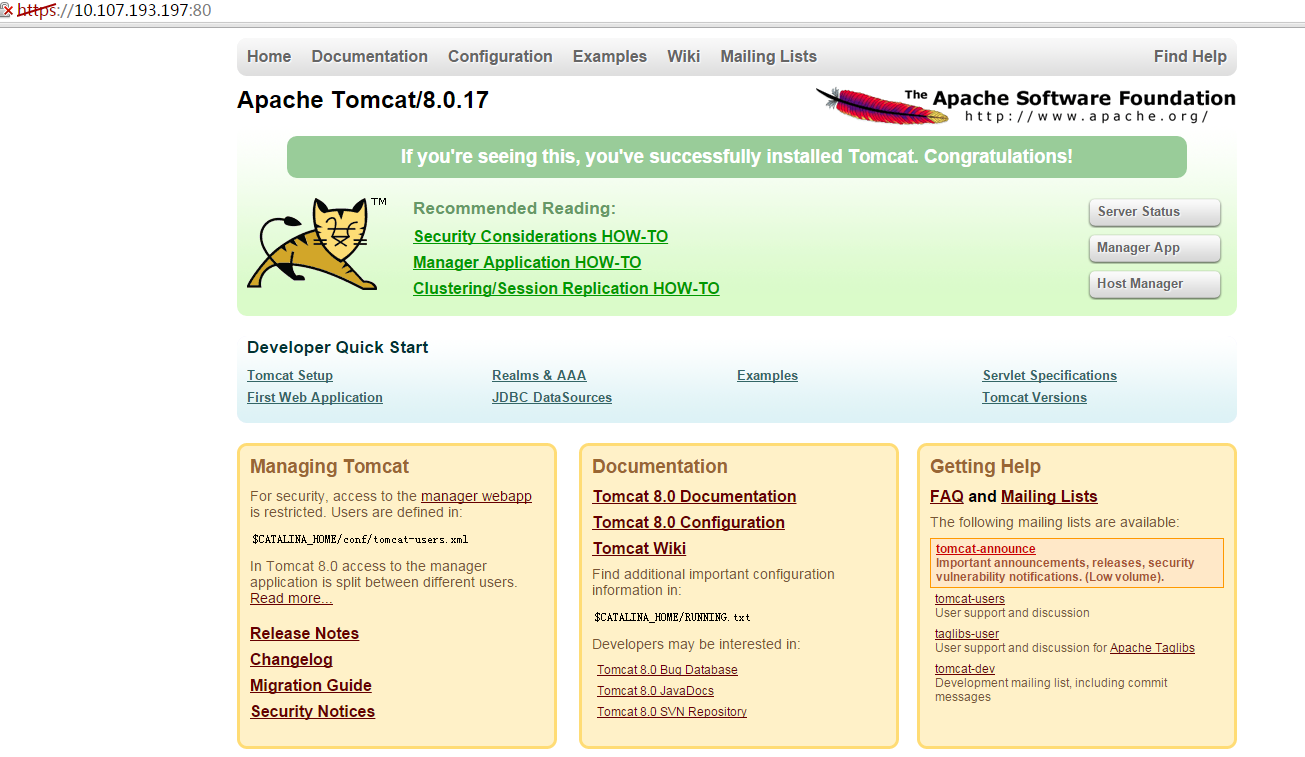


图7-2 Tomcat服务成功部署

这样一个基于HTTPS的Tomcat服务器就可以工作了，是不是很容易？之所以容易，是因为基础镜像为我们做了很多事情：首先我们不需要自己去编译安装一个Tomcat应用，只需要选择一个基础镜像。其次这个镜像默认的启动命令就是启动Tomcat，因此我们无需关心Tomcat细节，当然了，有关Web服务器软件的相关知识，还是需要多少懂一些的。

提示 如果公司内部有Registry服务器，可以将这些基础镜像积累下来，方便日后重复利用。当然也可以注册Docker官方的DockerHub，将其上传之。

### 8.2.3 将Web源码导入到Tomcat镜像中

在上一小节，通过commit的命令已经成功地将证书导入到镜像中，实际上，将文件导入到镜像中，有很多方式，本节将介绍更通用、更便于开发的Dockerfile方式来制作镜像。

总体上讲，将源码导入到镜像中有两种思路，一种是静态导入，就是将host主机上的源码拷贝一份到镜像中；还有一种可以动态将host主机上的源码目录挂载到镜像中。下面分别看下这两种方式。

#### 1. 静态导入

假如源码放在工程目录的websrc下，并在当前目录创建了一个Dockerfile：

# this Dockerfile is used to build HTTP Web Image.

FROM tomcat:https

MAINTAINER Wentao Zhang <[zhangwentao234@huawei.com](mailto:zhangwentao234@huawei.com)>

COPY ./websrc /usr/local/tomcat/webapps/myproj/

基础镜像将Tomcat安装到了/usr/local/tomcat下，所以我们需要把源码拷贝到webapps下面，然后制作镜像：

root@ubuntu:~/work# docker build myweb:v1 .

下面检测镜像制作完成情况：

# docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

myweb v1 4986bf8c1536 7 mimutes ago 509.8 MB

#### 2. 动态挂载

# This Dockerfile will use dynamic VOLUME to mount the Host SRC code

# to container.

FROM tomcat:https

MAINTAINER Wentao <[wentao@huawei.com](mailto:wentao@huawei.com)>

RUN mkdir –p /usr/local/tomcat/webapps/myproj

VOLUME /usr/local/tomcat/webapps/myproj

动态导入实际上在容器里只创建了一个用来放源代码的数据卷，只有在执行docker run命令时，源码才会动态地挂载到容器中。

# docker run -ti –v $(pwd)/websrc:/usr/local/tomcat/webapps/myproj myweb:v1

这种方式也可以将web部署成功，读者可以自行检验。

总结一下，静态导入和动态挂载各有优缺点。在制作镜像的时候，静态导入法会直接将源码导入到镜像中，干净利索，镜像独立，无论在什么地方只要有Docker环境就可以运行部署成功。但是如果在开发阶段，需要频繁的修改源码，就很不方便；而在动态挂载的方式中，host和容器会共享一份源码，在开发阶段，可以使用这种方式，若发现问题，可随时修改，修改后立即生效，省去镜像制作的过程，提高调试效率。

提示 可以在工程目录下创建两个Dockerfile，一个用于发布版本，此时采用静态导入的方式；另一个用于调试，这里使用动态挂载方式。docker build是可以指定Dockerfile的（-f选项）。

### 8.2.4 部署与验证

镜像制作成功后，就可以部署这个容器应用了。

docker run -ti –p 80:8080 myweb:v1

当然如果是动态挂载的，还需要加上-v参数。如果以上步骤部署成功，则以通过Chrome等浏览器访问该站点。

提示 Tomcat中server.xml的配置里，这里配的是8080端口，如果读者想用其他端口，可自行修改。

## 为Web站点添加后台服务

在前面的章节中，已经学习过docker-compose，使用docker-compose可以轻松地管理多个容器，各个容器可完成不同的任务和功能。还是以我们的Web站点为例，一个Web站点是不可能没有后台服务的。假如我们的Web站点挂了三个后台服务：

* 后台service（bkservice）
* 认证服务器（auth）
* Mysql数据库（Mysql）

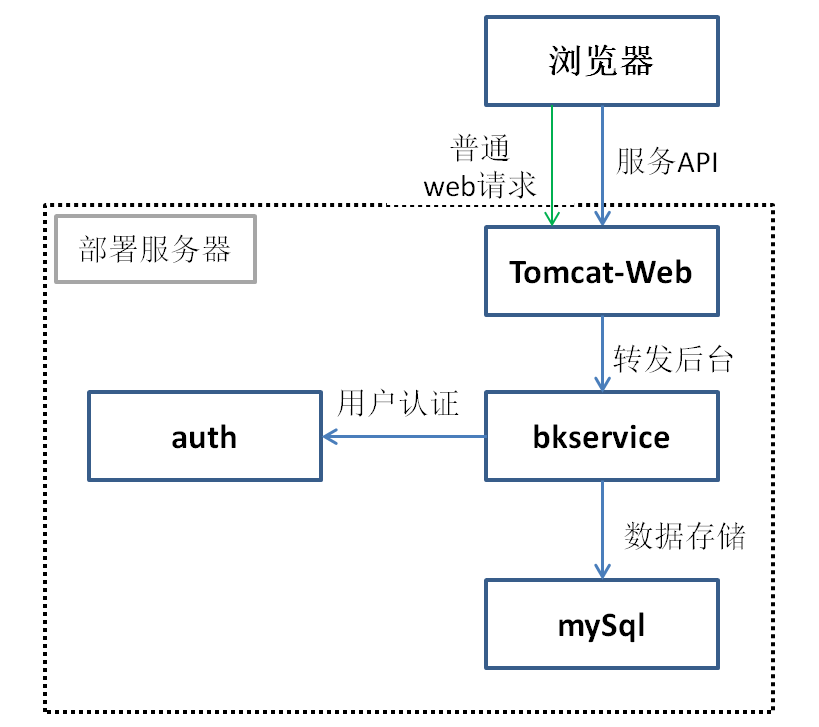
其中后台service是用来为Web站点服务的，具体功能由Web的功能而定。认证服务器是为了解耦用户认证的服务器，两者通过发送HTTP请求API达到认证鉴权的目的。MySQL是数据库，bkservice下挂的数据库。其中每个模块都是独立的，在本工程中，我们会将每个模块都制作成一个镜像。下面看看整体工程框架，如图7-3所示。

图8-3 Docker实战工程模块

浏览器和前端交互，普通的前端请求均由Tomcat服务器响应（比如请求web、显示控件等）。但是需要后台协作的API则会转发到后台，也就是转发给bkservice模块。当然bkservice模块每次收到HTTP请求后，需要发送API到auth模块，从而验证用户的身份以排除非法用户。验证通过后，bkservice需要将某些数据存储到MySQL的数据库中。

因为本书是介绍Docker的，所以Web前端或者后台服务器开发相关的内容不会细讲，且这个项目只是设计了简单的用例场景，并未真正实现具体功能，只是将该框架搭起来而已。

### 8.3.1 代码组织结构

从图7-3可以看到，我们至少有四个功能上非常独立的模块，这些模块可以不用关心对方在什么地方、什么环境、以什么样的方式运行，只要能提供接口API就可以一起工作。从组织结构上讲，它们彼此也是独立的。若按照传统的开发部署模式，须将不同的模块分给不同的组，然后各自为战，开发源码。等部署的时候，各自维护各自的环境，各自部署各自的服务。但这样一来，部署、维护就成本很高。

如果把各个服务做成Docker镜像，并用docker-compose工具来管理工程，就会大大简化工作。首先，将服务发布为镜像，服务所依赖的服务器环境就会被封装在镜像中，host上只需要能跑Docker即可。其次，使用docker-compose工具，可以简单地一次部署多个镜像服务。下面是笔者做的工程目录结构：

# tree.

├── build.sh

├── docker-compose.yml

├── keys

│   ├── server.xml

│   └── tomcat.keystore

├── misc

│   ├── auth

│   │   ├── Dockerfile

│   ├── bkservice

│   │   ├── Dockerfile

│   │   └── src

│   └── webui

│   ├── Dockerfile

│   └── src

├── release.sh

└── scripts

说明：

* build.sh：表示编译使用脚本程序。
* docker-compose.yml： 此文件是docker-compose的配置文件。
* keys ：此文件夹存放用于Https的证书文件，每次制作镜像，将证书拷贝到Tomcat镜像中。
* misc：镜像目录，每个镜像有个单独目录，用于制作镜像。
* auth：认证模块，使用目录下Dockerfile制作auth镜像。
* bkservice：后台服务，源码在src目录下。
* webui：前台UI制作镜像目录，src目录为源码（可选，或者可以在Dockerfile里的git clone下载源码，然后编译源码把输出二进制放在镜像中）。
* release.sh：版本发布时执行的脚本。
* scripts：工程中需要的一些工具脚本。

首先看下 docker-compose.yml文件：

# cat docker-compose.yml

dockerui:

build: misc/webui

volumes:

- ./keys:/usr/local/tomcat/keys

links:

- dockerbkservice:dockerbkservice

ports:

- 8080:8080

restart: always

dockerbkservice:

build: misc/dockerconsole

links:

- dockerauth:dockerauth

- dockersql:dockersql

volumes:

- ./config.yml:/usr/local/bkservice/config.yml

ports:

- "9090:9090"

environment:

- MACHINE\_CLIENT\_CERT\_PATH=/root/.docker/machine/certs

dockerauth:

build: misc/auth

environment:

- no\_proxy=127.0.0.1

volumes:

- misc/dockerauth/app.conf:/conf/app.conf

dockersql:

image: mysql:v1.0

以dockerbkservice为例，讲解如下：

* dockerbkservice: 顶级标签，表示一个镜像。
* build: 是指编译构建的时候（docker-compose build）会去哪个目录构建这个镜像。
* links: 和其他容器的link，可以以域名形式访问该容器。
* volumes: 将host主机上的某个路径（文件），挂载到容器中的某个路径（文件）。
* ports: 将host的端口映射到容器中的某个端口。
* environment: 设置一个环境变量。

从上面不难推测到，其实docker-compose工具做的工作相当于解析配置文件，之后按照配置文件的配置项去执行Docker命令而已。这里的四个镜像容器均由docker-compose管理。

提示 如果觉得将不同的模块放在同一个工程下管理不方便，可以将各模块分别管理。比如auth模块，可以作为一个工程，等执行的时候，则在misc/auth/Dockerfile里通过版本控制工具将代码下载下来，并编译运行。以Git为例，在Dockerfile添加”RUN git clone [git@test/auth.git](mailto:git@test/auth.git); RUN cd auth/ ;make”即可。

### 8.3.2 组件镜像制作过程

前一节介绍了整个工程的组织结构，下面来看看每个模块是怎么制作镜像的，因为镜像制作的过程大同小异，这里以UI工程为例，解析下Web制作过程。

首先看下制作Web镜像的Dockerfile，该文件放在<project>/misc/webui/下。

# cat misc/webui/Dockerfile

FROM tomcat:https

MAINTAINER Wentao Zhang <zhangwentao234@huawei.com>

ENV no\_proxy dockerbuild

RUN mkdir -p /usr/local/tomcat/keys

VOLUME /usr/local/tomcat/keys

RUN cd /usr/local/tomcat/webapps; git clone -b master http://xx/myproj.git; cd myproj

逐条解析：

* FROM语句选择了基础镜像，即我们之前生产的tomcat:https镜像。
* MAINTAINER，指定维护者信息。
* ENV no\_proxy， 因为笔者服务器配有代理，所以设置了no\_proxy。
* RUN mkdir，命令在容器中创建了keys文件夹。
* VOLUME指令创建了keys卷，实际上证书是从host动态挂载到容器中的。结合docker-compose的配置文件，是把host当前目录下的keys文件夹挂载到容器中。
* RUN git clone，将远程的git源码clone到容器中，并部署。

提示 因为将证书静态拷贝到镜像中很生硬，所以工程中用了动态挂载方式将证书导入到镜像中，方便实用。之前介绍的commit方式，只是为了帮助大家了解commit的用法而已。

### 8.3.3 整体部署服务

其实有了docker-compose工具，部署就会变得极为简单，首先可以考虑删掉之前的镜像，然后重新build，最后一次性启动所有容器。如下：

# docker-compose rm --force

# docker-compose build

# docker-compose up –d

docker-compose命令的用法参看前几个章节。其中rm --force是用来删除镜像（主要是上次build遗留下来的）；build命令是用来制作这些镜像的；up –d是启动这几个镜像的。

## 小结

目前，Docker已经越来越流行了，将来掌握Docker可能成为不少工作的必备技能。和其他章节不同，本章注重实例和运用，主要以工程实例的方式，介绍了Docker的开发流程。对于Docker的学习，笔者认为，最好的学习方式就是实践，所以读者需要在阅读Docker相关书籍的同时，多动手，多练习，才能很好的驾驭Docker。