第 5-6 课：使用 Docker 部署 Spring Boot 项目

如今 Docker 的使用已经非常普遍，特别是在一线互联网公司，使用 Docker 技术可以帮助企业快速水平扩展服务，从而达到弹性部署业务的能力。在云服务概念兴起之后，Docker 的使用场景和范围进一步发展，如今在微服务架构越来越流行的情况下，微服务 + Docker 的完美组合，更加方便微服务架构运维部署落地。

什么是 Docker

Docker 最初是 dotCloud 公司创始人 Solomon Hykes 在法国期间发起的一个公司内部项目，它是基于 dotCloud 公司多年云服务技术的一次革新，并于 2013 年 3 月以 Apache 2.0 授权协议开源，主要项目代码在 GitHub 上进行维护。Docker 项目后来还加入了 Linux 基金会，并成立推动 开放容器联盟（OCI）。

Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口，它是目前最流行的 Linux 容器解决方案。Docker 将应用程序与该程序的依赖打包在一个文件里面，运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样，有了 Docker，就不用担心环境问题了。

总体来说，Docker 的接口相当简单，用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。

为什么要使用 Docker

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式是运行 10 个不同的应用就要启动 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。

具体说来，Docker 在如下几个方面具有较大的优势。

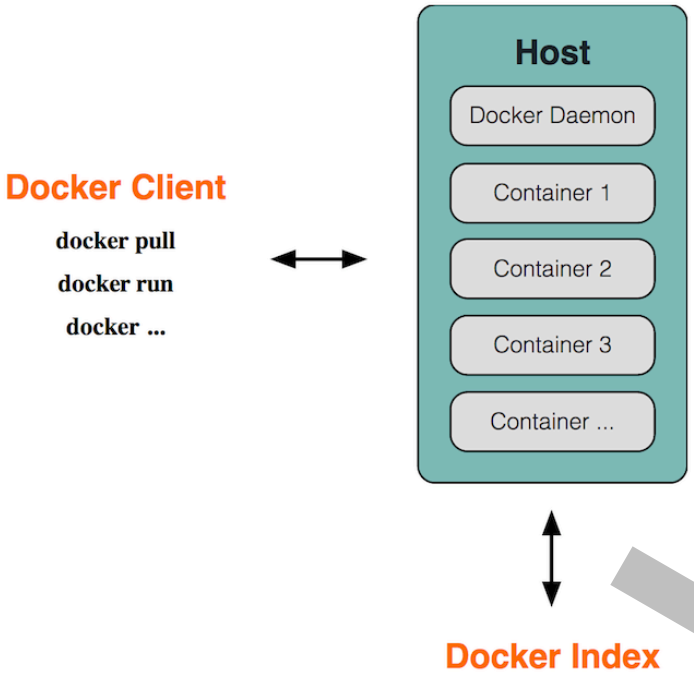
- 更快速的交付和部署，开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。
- 更高效的虚拟化，Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。
- 更轻松的迁移和扩展，Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。
- 更简单的管理，使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。

Docker 相关概念

Docker 是 CS 架构，主要有两个概念，具体如下。

- **Docker daemon**：运行在宿主机上，Docker 守护进程，用户通过 Docker client（Docker 命令）与 Docker daemon 交互。
- **Docker client**：Docker 命令行工具，是用户使用 Docker 的主要方式，Docker client 与 Docker

daemon 通信并将结果返回给用户，Docker client 也可以通过 socket 或者 RESTful API 访问远程的 Docker daemon。



Docker 的组成有三个主要概念，具体如下。

- **Docker image**：镜像是只读的，镜像中包含有需要运行的文件。镜像用来创建 container，一个镜像可以运行多个 container；镜像可以通过 Dockerfile 创建，也可以从 Docker hub/registry 上下载。
- **Docker container**：容器是 Docker 的运行组件，启动一个镜像就是一个容器，容器是一个隔离环境，多个容器之间不会相互影响，保证容器中的程序运行在一个相对安全的环境中。
- **Docker hub/registry**：共享和管理 Docker 镜像，用户可以上传或者下载上面的镜像，官方地址可点击这里查看，也可以搭建自己私有的 Docker registry。

镜像就相当于打包好的版本，镜像启动之后运行在容器中，仓库就是装存储镜像的地方。

接下来创建一个 Spring Boot 项目，然后给项目添加 Docker 支持，最后对项目进行部署。

Spring Boot 项目

在 pom.xml 中，使用 Spring Boot 2.x 相关依赖：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
</parent>
```

添加 web 和测试依赖：

```
<dependencies>
     <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

创建一个 DockerController，在其中有一个 index() 方法，访问时返回：Hello Docker!

```
@RestController
public class DockerController {

    @RequestMapping("/")
    public String index() {
        return "Hello Docker!";
    }
}
```

启动类：

```
@SpringBootApplication
public class DockerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DockerApplication.class, args);
    }
}
```

添加完毕后启动项目，启动成功后浏览器访问网址：http://localhost:8080/，页面返回：Hello Docker!，说明 Spring Boot 项目配置正常。

项目添加 Docker 支持

在 pom.xml 中添加 Docker 镜像名称：

```
<properties>
    <docker.image.prefix>springboot</docker.image.prefix>
</properties>
```

plugins 中添加 Docker 构建插件：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <!-- Docker maven plugin -->
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.0.0</version>
            <configuration>
                <imageName>${docker.image.prefix}/${project.artifactId}</imageName
>
                <dockerDirectory>src/main/docker</dockerDirectory>
                <resources>
                    <resource>
                        <targetPath>/</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.jar</include>
                    </resource>
                </resources>
            </configuration>
        </plugin>
        <!-- Docker maven plugin -->
    </plugins>
</build>
```

- ${docker.image.prefix}，自定义的镜像名称
- ${project.artifactId}，项目的 artifactId
- ${project.build.directory}，构建目录，缺省为 target
- ${project.build.finalName}，产出物名称，缺省为 ${project.artifactId}-${project.version}

在目录 src/main/docker 下创建 Dockerfile 文件，Dockerfile 文件用来说明如何来构建镜像。

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD spring-boot-docker-1.0.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

这个 Dockerfile 文件很简单，构建 JDK 基础环境，添加 Spring Boot Jar 到镜像中，简单解释一下。

- FROM，表示使用 JDK 8 环境为基础镜像，如果镜像不是本地的将会从 DockerHub 进行下载。
- VOLUME，VOLUME 指向了一个 /tmp 的目录，由于 Spring Boot 使用内置的 Tomcat 容器，Tomcat 默认使用 /tmp 作为工作目录，这个命令的效果是：在宿主机的 /var/lib/docker 目录下创建一个临时文件并把它链接到容器中的 /tmp 目录。

- ADD，复制文件并且重命名。
- ENTRYPOINT，为了缩短 Tomcat 的启动时间，添加 java.security.egd 的系统属性指向 /dev/urandom 作为 ENTRYPOINT。

这样 Spring Boot 项目添加 Docker 依赖就完成了，下面解释一下什么是 Dockerfile?

Dockerfile 介绍

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

镜像的定制实际上就是定制每一层所添加的配置、文件。如果可以把每一层修改、安装、构建、操作的命令都写入一个脚本，用这个脚本来构建、定制镜像，那么之前提及的无法重复的问题、镜像构建透明性的问题、体积的问题就都会解决，这个脚本就是 Dockerfile。

Dockerfile 是一个文本文件，其内包含了一条条的指令（Instruction），每一条指令构建一层，因此每一条指令的内容，就是描述该层应当如何构建。有了 Dockerfile，当需要定制自己额外的需求时，只需在 Dockerfile 上添加或者修改指令，重新生成 image 即可，省去了敲命令的麻烦。

Dockerfile 文件格式如下：

```
##Dockerfile 文件格式

# This dockerfile uses the ubuntu image
# VERSION 2 - EDITION 1
# Author: docker_user
# Command format: Instruction [arguments / command] ..

#（1）第一行必须指定基础镜像信息
FROM ubuntu

#（2）维护者信息
MAINTAINER docker_user docker_user@email.com

#（3）镜像操作指令
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

#（4）容器启动执行指令
CMD /usr/sbin/nginx
```

Dockerfile 分为四部分：基础镜像信息、维护者信息、镜像操作指令、容器启动执行指令。一开始必须要指明所基于的镜像名称，接下来一般会说明维护者信息；后面则是镜像操作指令，如 RUN 指令。每执行一条

RUN 指令，镜像添加新的一层，并提交；最后是 CMD 指令，来指明运行容器时的操作命令。

Spring Boot 应用 Docker 部署需要使用 Dockerfile 定义部署方案。

构建环境

建议在 Linux 环境下安装 Docker，Window 环境搭建比较复杂且容易出错，使用 Centos7 + yum 来安装 Docker 环境很方便。

Docker 安装

Docker 软件包已经包括在默认的 CentOS-Extras 软件源里，因此想要安装 Docker，只需要运行下面的 yum 命令：

```
yum install docker
```

安装完成后，使用下面的命令来启动 docker 服务，并将其设置为开机启动：

```
service docker start
chkconfig docker on
```

> LCTT 译注，此处采用了旧式的 sysv 语法，如采用 CentOS 7 中支持的新式 systemd 语法，如下：

```
systemctl start docker.service
systemctl enable docker.service
```

使用‘Docker 中国’加速器：

```
vi  /etc/docker/daemon.json

#添加后：
{
    "registry-mirrors": ["https://registry.docker-cn.com"],
    "live-restore": true
}
```

重新启动：

```
systemctl restart docker
```

测试：

```
docker version
```

输入上述命令，返回 docker 的版本相关信息，证明 docker 安装成功。

安装 JDK

```
yum -y install java-1.8.0-openjdk*
```

- 配置环境变量
- 打开 vim /etc/profile
- 添加以下内容

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-0.b14.el7_4.x86_64
export PATH=$PATH:$JAVA_HOME/bin
```

修改完成之后，使其生效：

```
source /etc/profile
```

输入 java -version 返回版本信息则安装正常。

安装 MAVEN

下载地址可点击这里查看：

```
## 解压
tar vxf apache-maven-3.5.2-bin.tar.gz
## 移动
mv apache-maven-3.5.2 /usr/local/maven3
```

修改环境变量， 在 /etc/profile 中添加以下几行：

```
MAVEN_HOME=/usr/local/maven3
export MAVEN_HOME
export PATH=${PATH}:${MAVEN_HOME}/bin
```

记得执行 source /etc/profile 使环境变量生效。

输入 mvn -version 返回版本信息则安装正常。

> 这样整个构建环境就配置完成了。

使用 Docker 部署 Spring Boot 项目

将项目 spring-boot-docker 复制到服务器中，进入项目路径下进行打包测试。

```
#打包
mvn clean package
#启动
java -jar target/spring-boot-docker-1.0.jar
```

看到 Spring Boot 的启动日志后表明环境配置没有问题，接下来使用 DockerFile 构建镜像。

```
mvn package docker:build
```

第一次构建可能有点慢，当看到以下内容的时候表明构建成功：

```
...
Step 1 : FROM openjdk:8-jdk-alpine
 ---> 224765a6bdbe
Step 2 : VOLUME /tmp
 ---> Using cache
 ---> b4e86cc8654e
Step 3 : ADD spring-boot-docker-1.0.jar app.jar
 ---> a20fe75963ab
Removing intermediate container 593ee5e1ea51
Step 4 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
 ---> Running in 85d558a10cd4
 ---> 7102f08b5e95
Removing intermediate container 85d558a10cd4
Successfully built 7102f08b5e95
[INFO] Built springboot/spring-boot-docker
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 54.346 s
[INFO] Finished at: 2018-03-13T16:20:15+08:00
[INFO] Final Memory: 42M/182M
[INFO] ------------------------------------------------------------------------
```

使用 docker images 命令查看构建好的镜像：

```
docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED
          SIZE
springboot/spring-boot-docker   latest              99ce9468da74        6 seconds
ago       117.5 MB
```

springboot/spring-boot-docker 就是我们构建好的镜像，下一步就是运行该镜像：

```
docker run -p 8080:8080 -t springboot/spring-boot-docker
```

启动完成之后使用 docker ps 查看正在运行的镜像：

```
docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREAT
ED              STATUS              PORTS                    NAMES
049570da86a9        springboot/spring-boot-docker   "java -Djava.security"   30 se
conds ago       Up 27 seconds       0.0.0.0:8080->8080/tcp   determined_mahavira
```

可以看到构建的容器正在在运行，访问浏览器：http://192.168.0.x:8080/，返回：

```
Hello Docker!
```

> 说明使用 Docker 部署 Spring Boot 项目成功！

常用命令

除过以上使用的 Docker 命令外，还有一些其他常用的命令，通过这些命令可以查看容器的相关状态。

拉取 docker 镜像：

```
docker pull image_name
```

查看宿主机上的镜像，Docker 镜像保存在 /var/lib/docker 目录下：

```
docker images
```

删除镜像：

```
docker rmi  docker.io/tomcat:7.0.77-jre7   或者  docker rmi b39c68b7af30
```

查看当前有哪些容器正在运行：

```
docker ps
```

查看所有容器：

```
docker ps -a
```

启动、停止、重启容器命令：

```
docker start container_name/container_id
docker stop container_name/container_id
docker restart container_name/container_id
```

后台启动一个容器后，如果想进入到这个容器，可以使用 attach 命令：

```
docker attach container_name/container_id
```

删除容器的命令：

```
docker rm container_name/container_id
```

删除所有停止的容器：

```
docker rm $(docker ps -a -q)
```

查看当前系统 Docker 信息：

```
docker info
```

从 Docker hub 上下载某个镜像：

```
docker pull centos:latest
docker pull centos:latest
```

通过这些命令可以发现 Docker 的使用方式非常灵活，可以把项目做成镜像来管理，部署的时候直接下载对应版本的镜像直接运行即可。

总结

Docker 是容器化技术的基础，服务编排、弹性水平扩充都依赖于 Docker 的使用，使用 Docker 部署 Spring Boot 项目，可以把项目当做版本库一样管理，不需要关注环境、迁移等问题。Spring Boot 是微服务开发的基础，Docker 是容器化部署的基础，两者结合使用更加方便微服务架构运维部署落地。

> 点击这里下载源码。