**第20讲 docker综合实训**

**一、实验目的**

复习常用docker命令，达到综合应用的目标

**二、实验描述**

1. 利用之前知识进行综合应用，熟练掌握docker的常用命令
2. 实验题目：
   1. 创建一个简单的Web应用程序容器化
   2. 构建自定义镜像
   3. 构建一个运行 Java 程序的 Docker 容器

**三、实验环境**

* 系统版本：centos 7.6
* docker版本： 20.06.1-ce

**四、实验知识点**

* linux系统基础配置
* docker基础命令

**五、综合应用题**

**实验题目一：创建一个简单的Web应用程序容器化**

**参考步骤：**

1. 创建一个名为 "web-app" 的目录，并在其中创建一个名为 "Dockerfile" 的文件。

2. 编辑 Dockerfile 文件，以构建 Docker 镜像。以下是一个示例 Dockerfile：

```Dockerfile

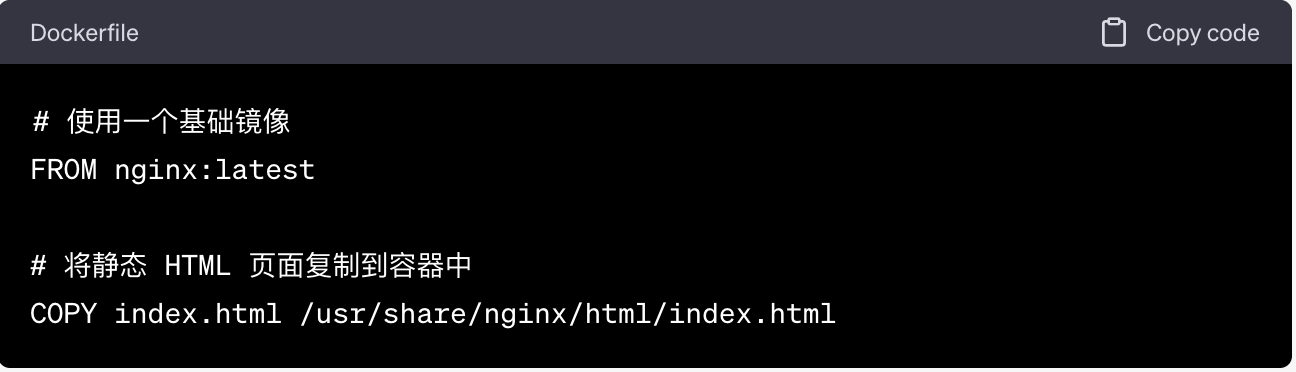
# 使用一个基础镜像

FROM nginx:latest

# 将静态 HTML 页面复制到容器中

COPY index.html /usr/share/nginx/html/index.html

```



3. 在同一目录中创建一个名为 "index.html" 的静态 HTML 文件，内容如下：

```html

<!DOCTYPE html>

<html>

<head>

<title>Hello, Docker!</title>

</head>

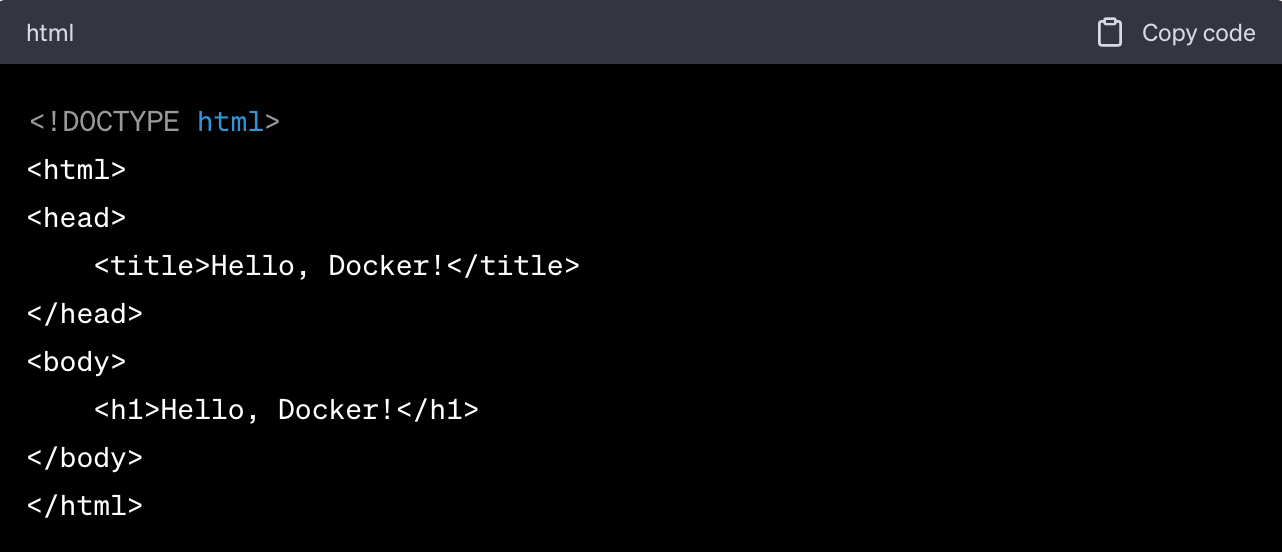
<body>

<h1>Hello, Docker!</h1>

</body>

</html>

```



4. 使用以下命令构建 Docker 镜像：

```bash

docker build -t web-app .

```



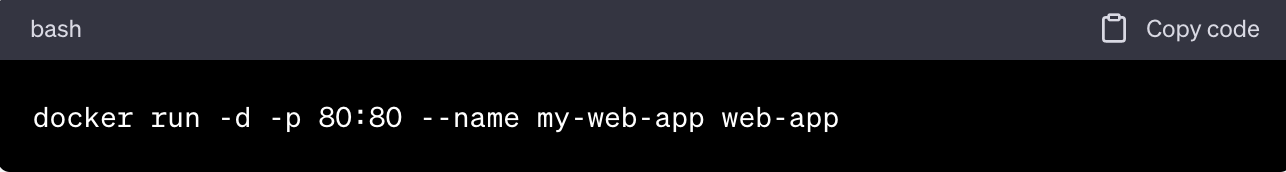
5. 一旦镜像构建完成，使用以下命令运行容器：

```bash

docker run -d -p 8081:80 --name my-web-app web-app

```

注意：这里的ip需要更改，不可以重复，建议使用学号，如8011，8012



6. 现在，你可以在浏览器中访问 http://服务器ip：上述改动端口 来查看 "Hello, Docker!" 的消息。

7. 当你完成测试后，可以停止容器并删除它：

```bash

docker stop my-web-app

docker rm my-web-app

```



注意：

容器名称建议后面也加上学号！

**实训题目二：构建自定义镜像**

**参考步骤：**

1、首先，创建一个简单的 Node.js 应用程序，将其保存为 `app.js` 文件：

```javascript

// app.js

const http = require('http');

const server = http.createServer((req, res) => {

res.end('Hello, Docker!\n');

});

server.listen(3000, () => {

console.log('Server is running on port 3000');

});

```



接下来，创建一个名为 "my-node-app" 的 Dockerfile：

```Dockerfile

# 使用 Node.js 14 作为基础镜像

FROM node:14

# 设置工作目录

WORKDIR /app

# 复制应用程序文件到容器中

COPY app.js .

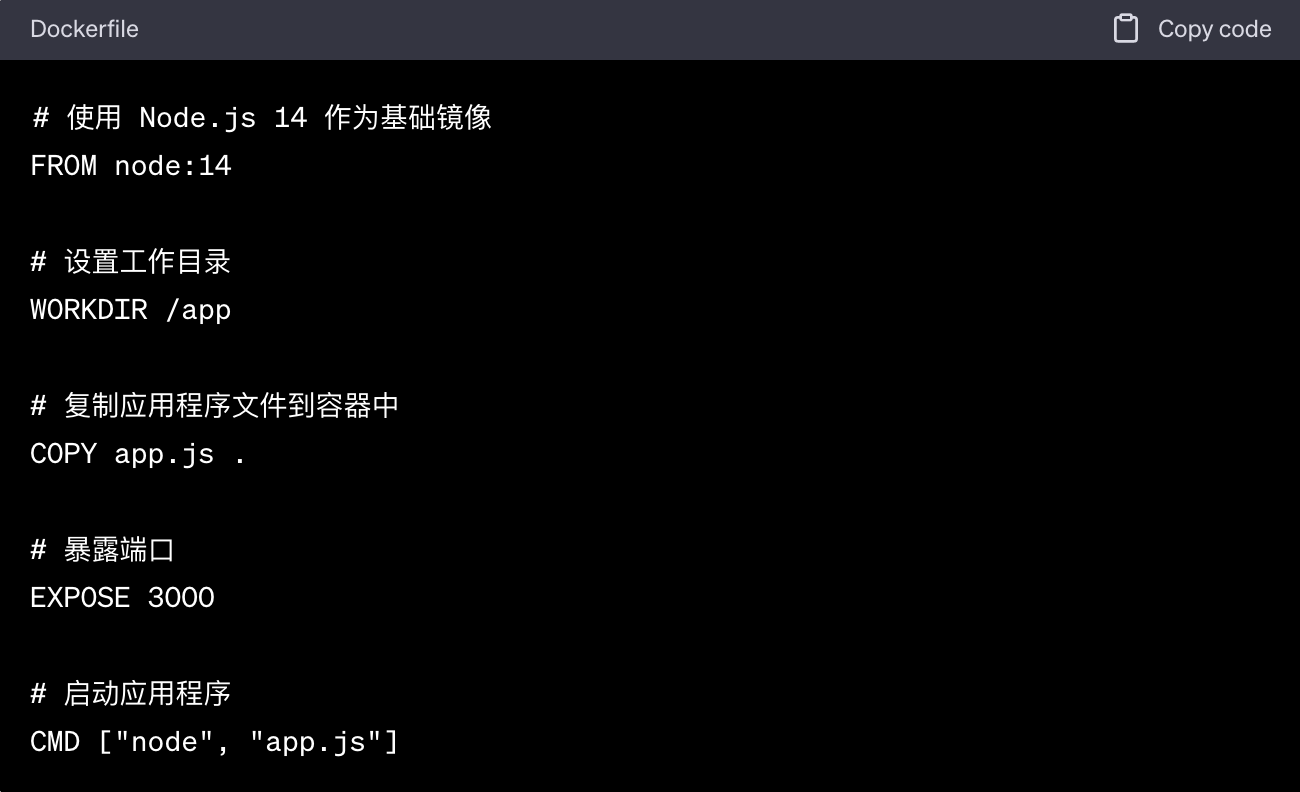
# 暴露端口

EXPOSE 3000

# 启动应用程序

CMD ["node", "app.js"]

```



然后，构建镜像并运行容器：

```bash

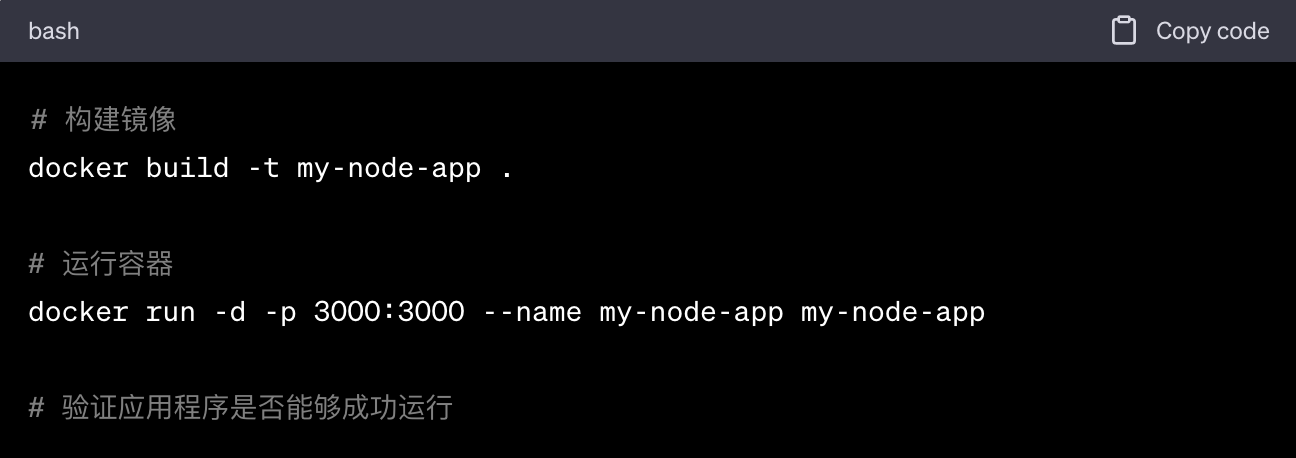
# 构建镜像

docker build -t my-node-app .

# 运行容器

docker run -d -p 3000:3000 --name my-node-app my-node-app

# 验证应用程序是否能够成功运行

```

**实训题目三：构建一个运行 Java 程序的 Docker 容器**

**参考步骤：**

步骤 1：创建 Java 程序

首先，在一个目录中创建一个简单的 Java 程序文件（例如，`HelloWorld.java`）：

```java

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, Docker!");

}

}

```



步骤 2：创建 Dockerfile

在同一目录中创建一个名为 `Dockerfile` 的文件，用于定义 Docker 容器的构建过程：

```Dockerfile

# 使用官方 OpenJDK 11 镜像作为基础镜像

FROM openjdk:11

# 将当前目录的 Java 程序文件复制到容器中的 /app 目录

COPY HelloWorld.java /app/HelloWorld.java

# 设置工作目录为 /app

WORKDIR /app

# 编译 Java 程序

RUN javac HelloWorld.java

# 声明容器需要暴露的端口（如果需要的话）

# EXPOSE 8080

# 运行 Java 程序

CMD ["java", "HelloWorld"]

```



这个 Dockerfile 使用了 OpenJDK 11 镜像作为基础镜像，然后复制了 Java 程序文件到容器中、编译 Java 程序、设置工作目录、声明需要暴露的端口（如果有的话）以及运行 Java 程序。

步骤 3：构建 Docker 镜像

在包含 Dockerfile 的目录中运行以下命令以构建 Docker 镜像：

```bash

docker build -t my-java-app .

```



步骤 4：运行 Docker 容器

使用以下命令运行新创建的 Docker 容器：

```bash

docker run my-java-app

```



你将看到 "Hello, Docker!" 的输出，这表示 Java 程序在容器中成功运行。