**实验报告（第 7 次）**

实验名称 Docker 容器化部署

实验时间 2025.05.06

同组同学 唐璇 小组分工 暂无

**一、实验目的**

1.理解容器化部署的概念：掌握Docker容器化部署的基本原理，了解如何将应用打包成Docker镜像，并通过容器化技术进行部署。

2.掌握Docker的基本操作：能够使用Docker命令进行镜像构建、容器运行和管理，理解Dockerfile的编写和镜像构建流程。

3.提高应用部署效率：通过容器化应用，实现快速部署、隔离运行和跨平台的兼容性，减少开发与生产环境的不一致问题。

4.学习Dockerfile编写：能够根据Go应用需求编写适当的Dockerfile，实现从基础镜像到最终容器的自动化构建。

5.掌握容器化部署的最佳实践：通过实验，理解如何使用Docker进行微服务应用的容器化管理，保证应用的一致性和可移植性。

**二、实验仪器设备或材料**

1.计算机：运行Windows、Linux或macOS操作系统的计算机。

2.Go语言环境：已安装并配置好Go语言环境，包括GOPATH、GOROOT等环境变量的设置。

3.文本编辑器或IDE：如VS Code、GoLand等，用于编写Go代码。

4.Docker环境：已安装并配置好Docker引擎，包括`docker`和`docker-compose`命令。

5.终端工具：如命令行、PowerShell、Terminal等，用于执行Docker命令。

**三、实验原理**

1. Docker容器化原理：Docker容器是一种轻量级的虚拟化技术，可以将应用及其依赖打包成镜像，并通过容器进行运行。容器共享主机操作系统的内核，但彼此隔离，具备独立的文件系统、网络和进程空间。

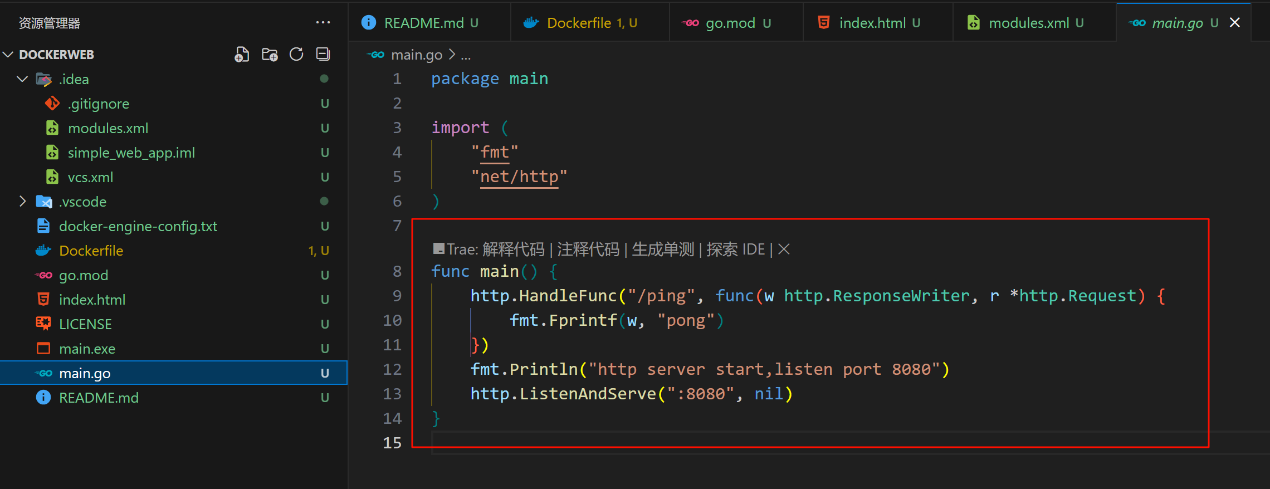
2. Dockerfile的作用：Dockerfile是用于自动化构建Docker镜像的脚本文件。通过指定基础镜像、安装依赖、复制源代码和配置环境等步骤，构建出可执行的应用镜像。

3. 构建与运行容器：构建镜像后，可以通过Docker命令创建和运行容器，容器会根据镜像的配置执行指定的程序。容器的生命周期与镜像分离，可以随时启动、停止或删除容器实例。

4. Go应用的容器化：Go应用通过Dockerfile中的`COPY`命令将应用源代码复制到镜像中，并通过`RUN`命令编译应用，然后使用`CMD`命令设置容器启动时的执行命令。

**四、实验内容与步骤**

1.创建Go应用代码：创建一个名为`simple\_web\_app`的Go项目，并编写一个http简单响应的接口。



2.编写Dockerfile：

在项目根目录下创建一个名为`Dockerfile`的文件，内容如下：

# 分阶段构建

#源镜像

FROM golang:1.24.3 AS builder

WORKDIR /app

ENV GOPROXY=https://goproxy.cn,https://goproxy.io,direct \

GO111MODULE=on \

CGO\_ENABLED=1

COPY . .

RUN go mod download

#go构建可执行文件

RUN go build -o simple\_web\_app

FROM apline:latest

#作者

LABEL maintainer="test"

LABEL version="1.0"

LABEL description="This is a docker image for my web application."

# MAINTAINER Razil "test"

## 在docker的根目录下创建相应的使用目录

RUN mkdir -p /app

## 设置工作目录

WORKDIR /app

## 把当前（宿主机上）目录下的文件都复制到docker上刚创建的目录下

#应用端口

EXPOSE 8080

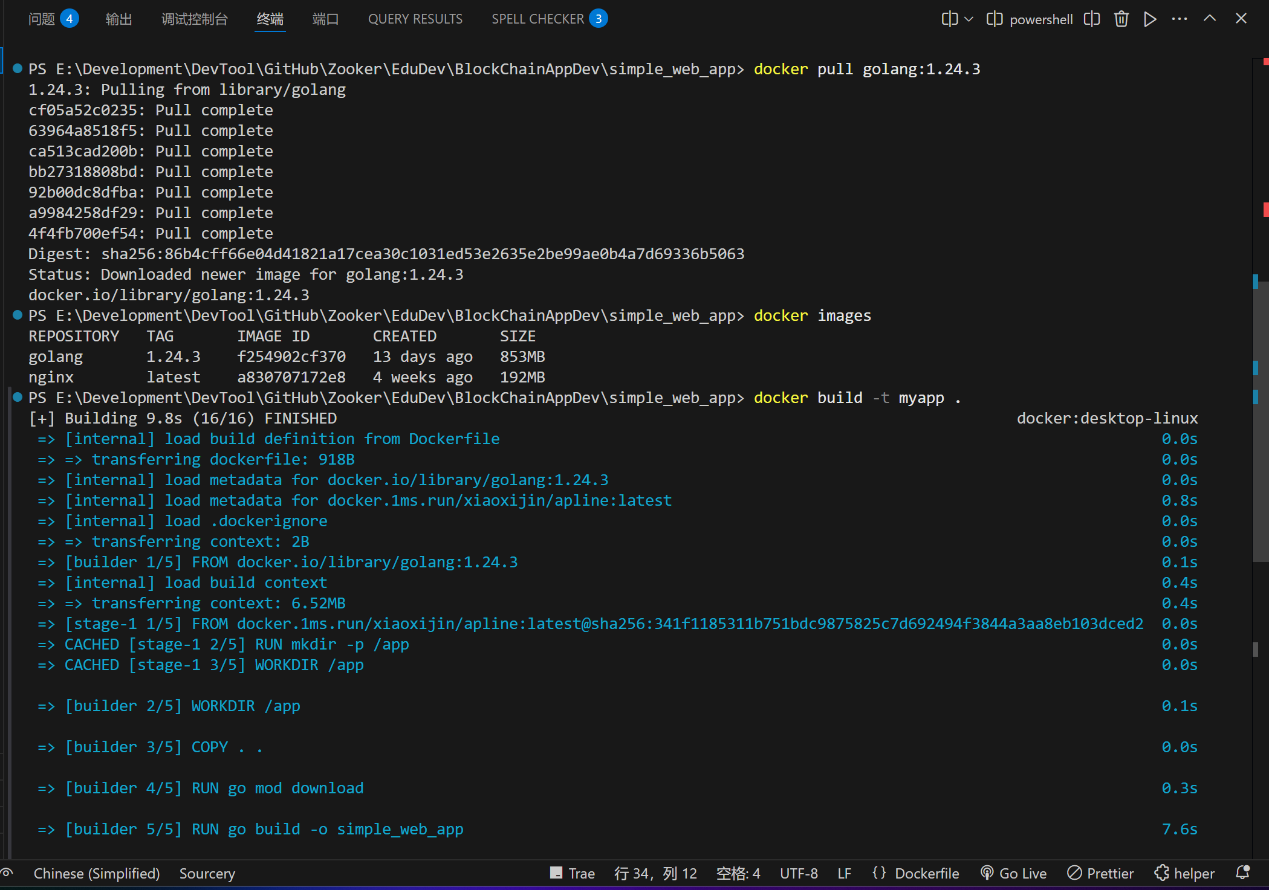
RUN chmod +x simple\_web\_app

ENTRYPOINT ["./simple\_web\_app"]

3.构建Docker镜像：

打开终端，进入到`simple\_web\_app`项目目录，执行以下命令构建Docker镜像：

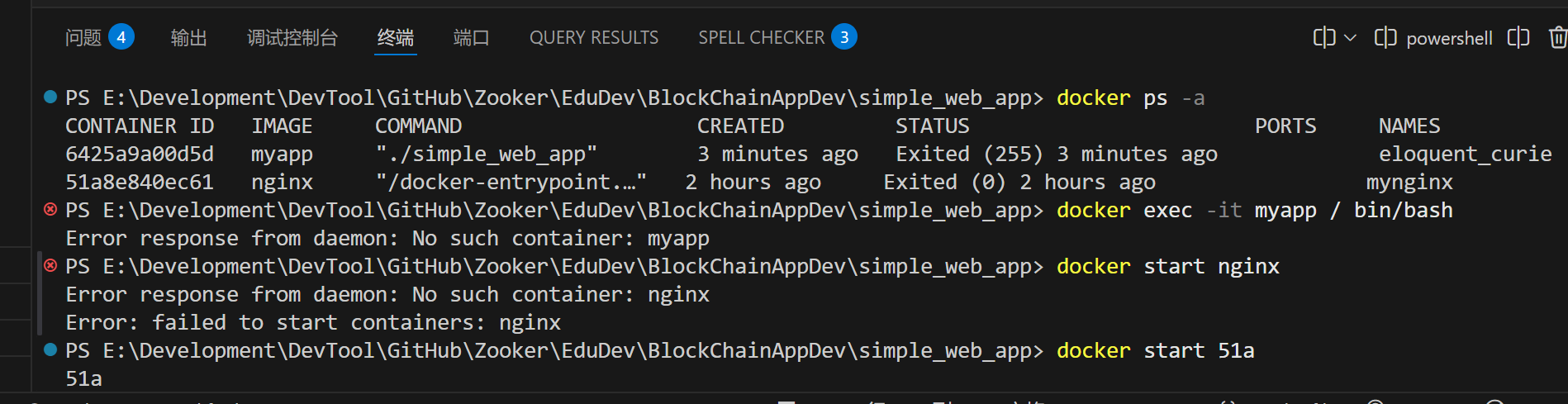
docker build -t myapp .(这里我改成了myapp镜像文件)



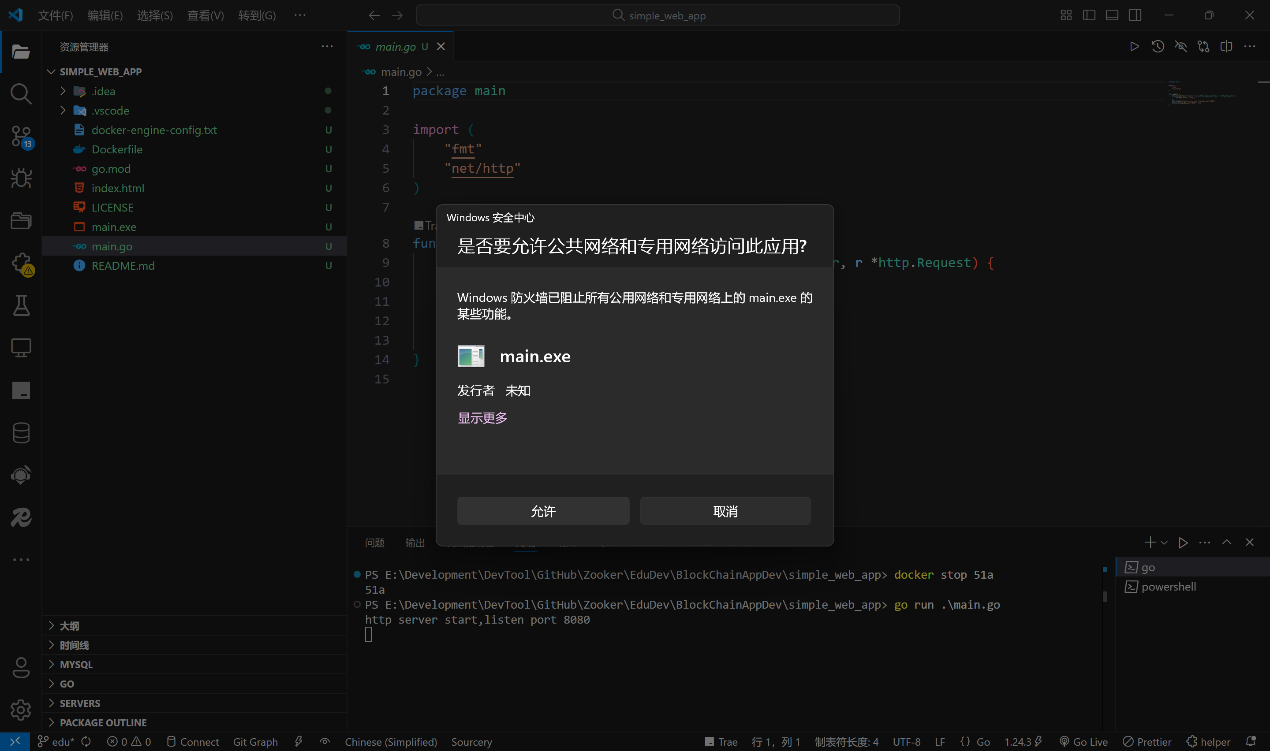
该命令会使用当前目录中的`Dockerfile`文件，构建一个名为`simple\_web\_app`的镜像。



4.运行Docker容器：运行构建好的镜像，创建并启动一个容器。

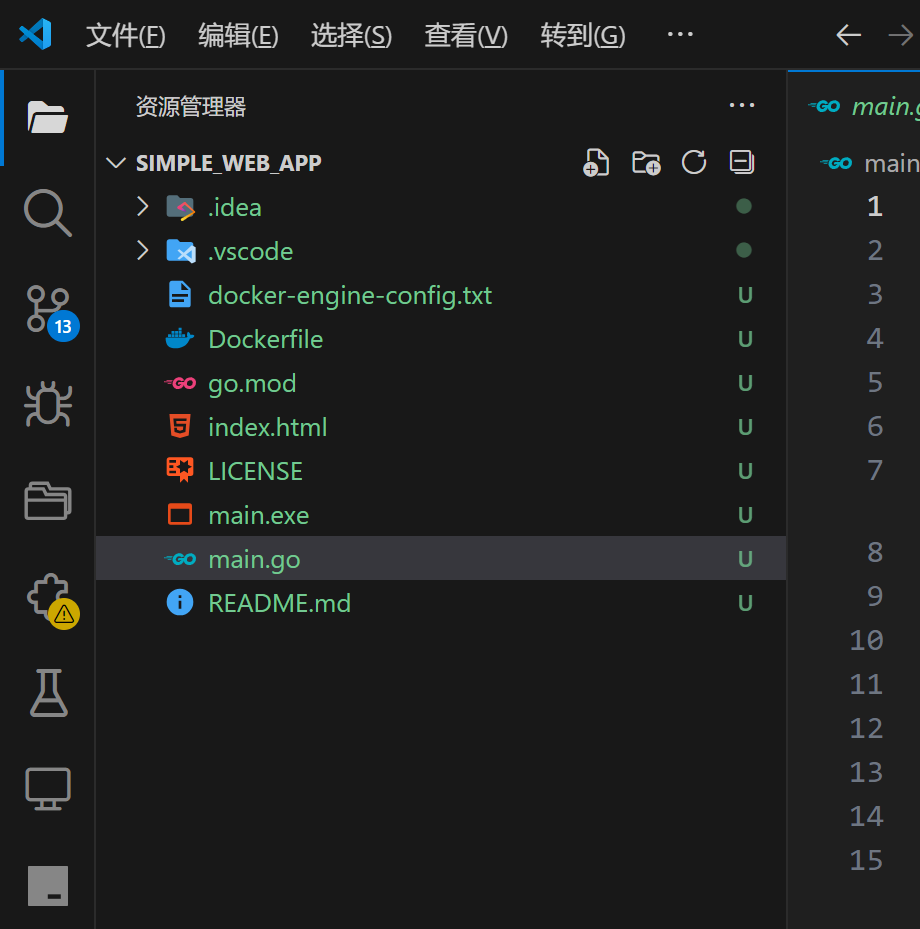


5.验证应用是否正常工作：在浏览器或使用`curl`命令访问`http://localhost:8080/ping`，如果返回`pong`，则说明应用已成功运行。

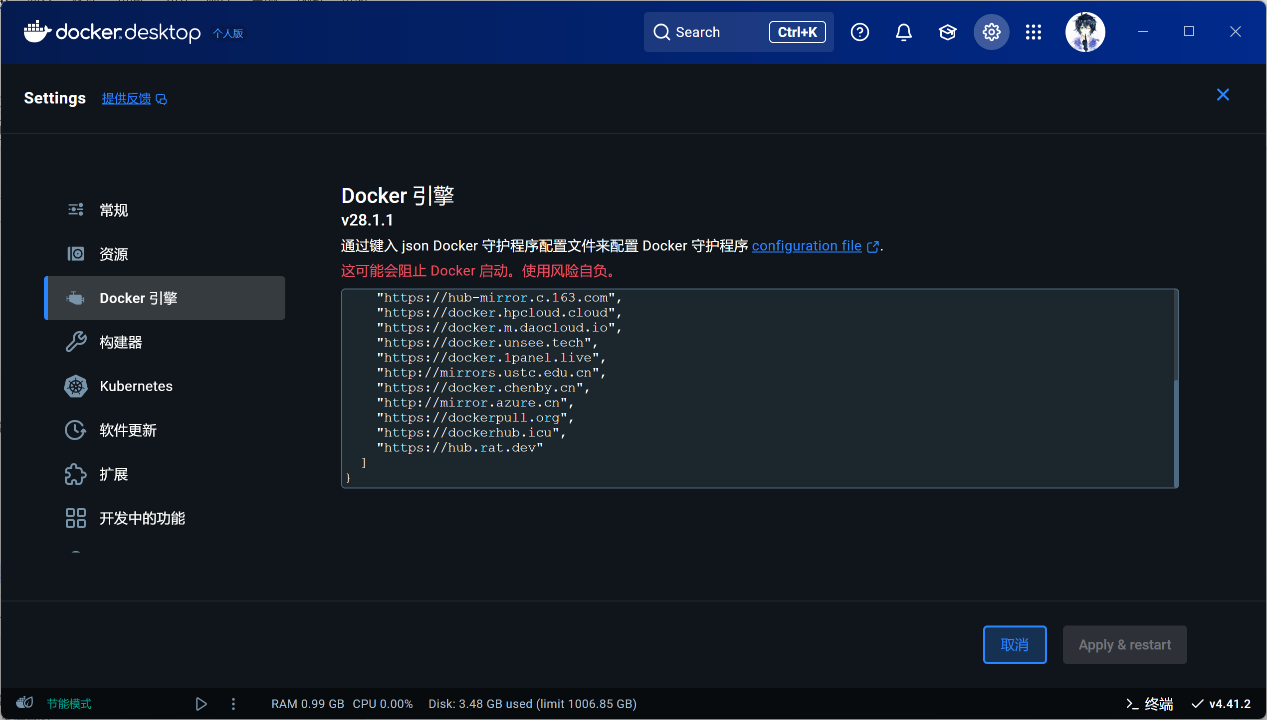


**五、实验结果与分析**

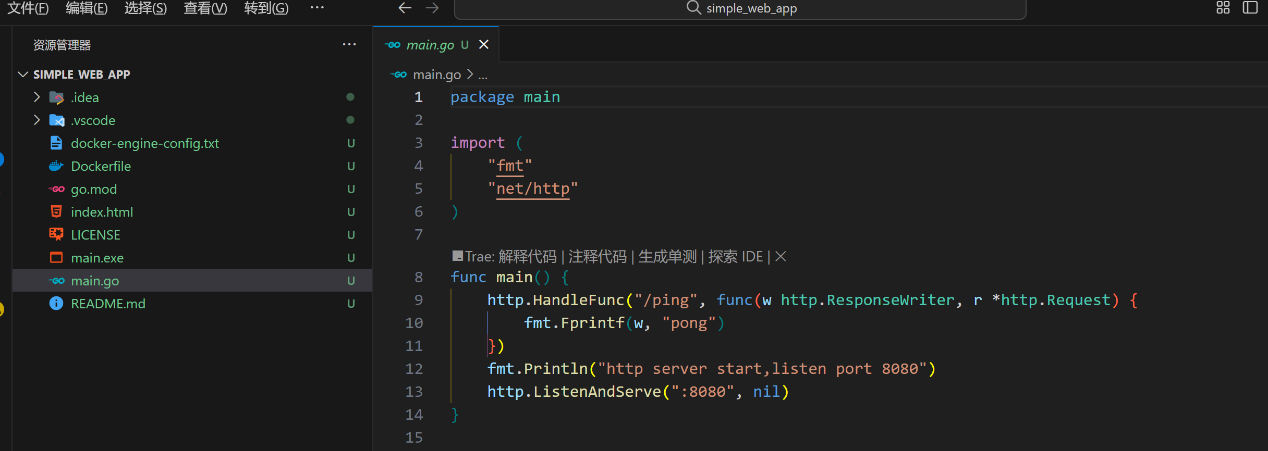
1. 构建项目文件目录及其相关文件（simple\_web\_app）



2.启动docker desktop并添加镜像网址到docker 引擎



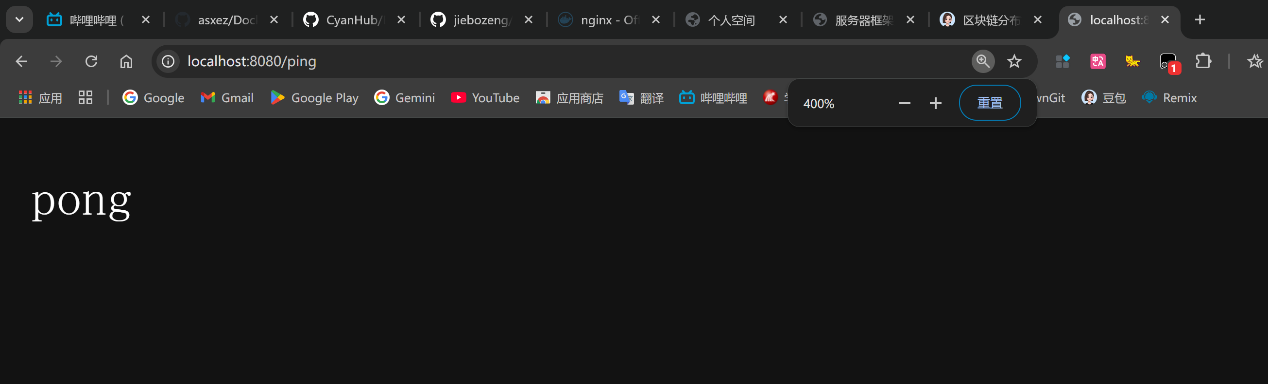
3. 编写main.go文件来实现基础的编写一个http简单响应的接口



4. 查看docker中是否存在相关的镜像文件



5. 跳转至浏览器并访问url查看是否成功（http://localhost:8080/ping）



**六、结论与体会**

通过本次 Docker 容器化部署实验，我系统掌握了从 Go 应用开发到镜像构建、容器运行的全流程。Dockerfile 的分阶段构建（先使用 Golang 镜像编译代码，再基于轻量级 Alpine 镜像运行）有效减小了镜像体积，提升了部署效率。实践中，我学会通过docker build和docker run命令实现应用的快速打包与跨环境运行，深刻体会到容器化在隔离依赖、简化部署中的优势。

过程中遇到容器启动失败问题，经排查发现是端口映射和执行权限配置疏忽导致，通过调整EXPOSE端口和chmod权限后成功运行。这让我意识到 Dockerfile 语法细节和容器生命周期管理的重要性。此次实验为微服务部署奠定了基础，未来可进一步结合 Docker Compose 实现多容器协同，提升复杂系统的可维护性与可移植性。

**七、教师评语**