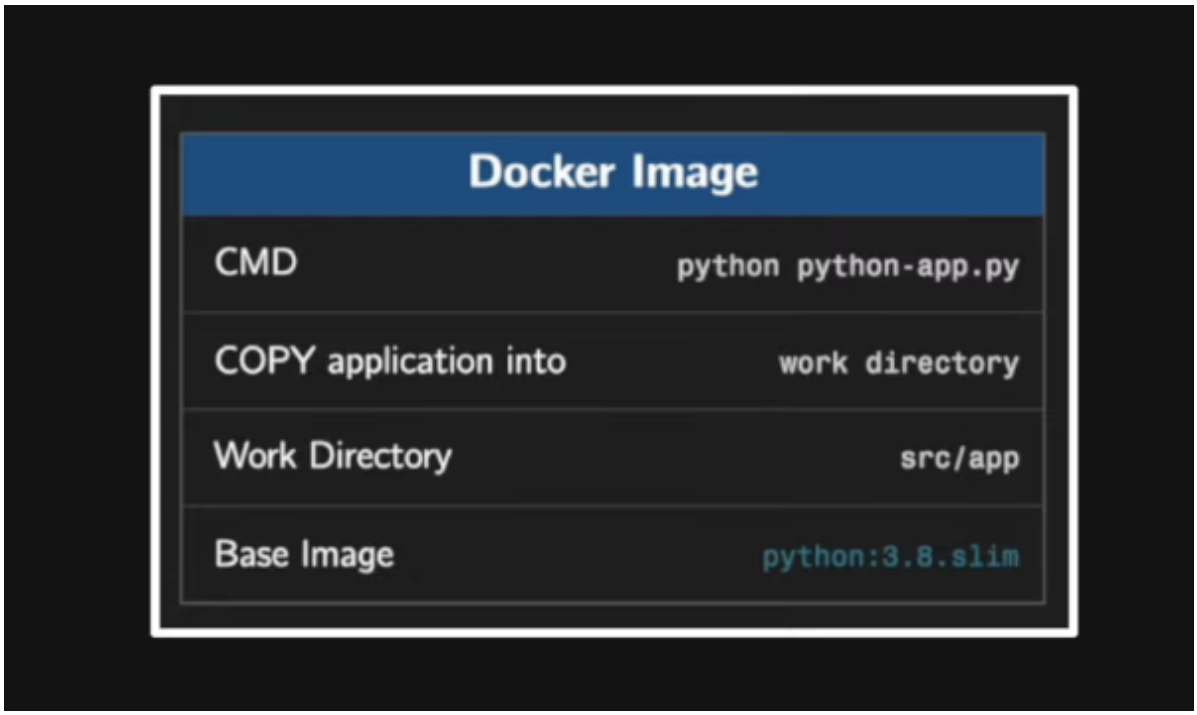
docker

1. 运行命令
    - docker run -v "/Users/huangjiayi/Documents/003UCI/learn/docker-course-remastered-main/lesson-starter-projects/01-starter-code:/src/app" python:3.8-slim python /src/app/python-app.py
    - 将本地的文件目录 `/Users/huangjiayi/Documents/003UCI/learn/docker-course-remastered-main/lesson-starter-projects/01-starter-code` 映射到容器内的 `/src/app` 目录。然后，它使用了 `python:3.8-slim` 镜像作为容器的基础镜像，并在容器内执行了 `python /src/app/python-app.py` 命令，这是在容器内运行名为 `python-app.py` 的 Python 脚本
2. docker ps - 列出正在运行的 Docker 容器

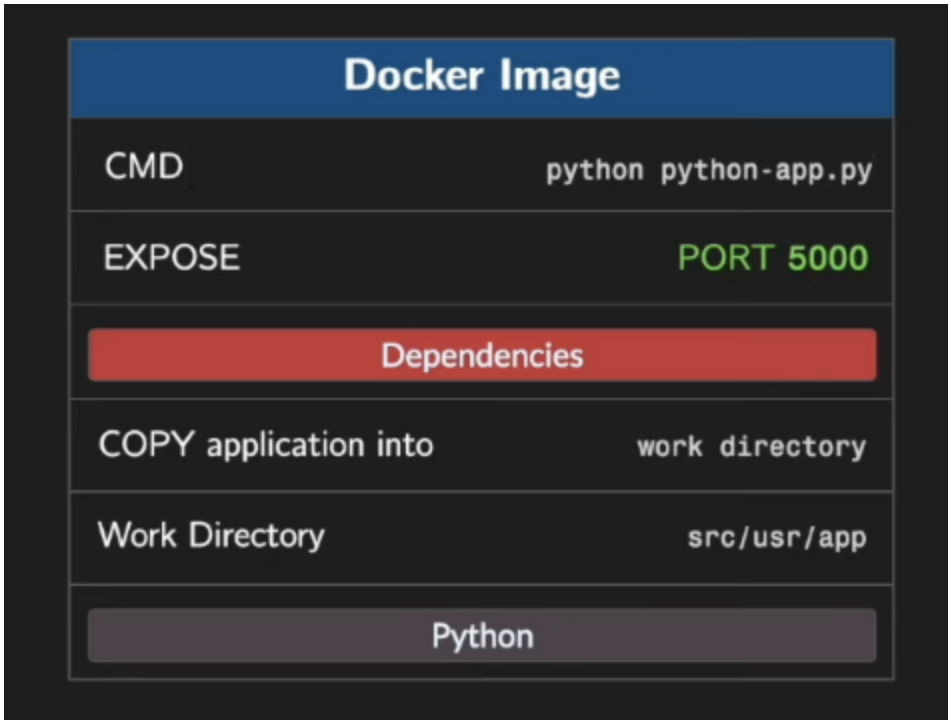
   docker ps -a 列出所有的 Docker 容器，包括正在运行的和已经停止的容器
3. docker rm {containerID} - 删除容器
4. docker stop {containerID} - 停止容器
5. python执行完后，只是停止，并没有删除
6. docker images - 查看镜像
7. docker image rm {imagesID} - 删除镜像
8. 创建执行python的镜像




```
//Dockerfile
## 1. Which base image do you want to use?
## 指定了基础镜像，即构建新镜像所使用的原始镜像。
##使用了 Python 3.8 版本的 slim 版本作为基础镜像
FROM python:3.8-slim

## 2. Set the working directory.
## 设置容器内的工作目录为 /src/app。在容器中执行的命令将在这个目录下运
WORKDIR /src/app

## 3. Copy your source code file to the working directory.
##将本地的 python-app.py 文件复制到容器的工作目录 (/src/app) 内
COPY python-app.py .

## 4. Define the command to run when the container starts.
## 定义容器启动时要运行的默认命令。
## 它指定了运行 Python 解释器，并传递 python-app.py 文件作为参数，从
CMD ["python","/src/app/python-app.py"]
```

- 在dockerfile的路径下执行，创建成功: docker build -t flask-app:0.0.1 .
- docker run {imageName} - 执行自定义镜像

9. docker创建执行web的镜像

10. 


```
## 1. Which base image do you want to use?
FROM python:3.8-slim

## 2. Set the working directory
WORKDIR "/src/app/"

## 3. Copy the project files into the working directory.
## 第一个dot表示和dockerfile同路径的所有文件，第二个dot表示workdir的
COPY . .

## 4. Install the dependencies
##运行在镜像本身内部去安装依赖
RUN pip install -r flask-demo/requirements.txt
```

```
## 5. Document and inform the developer that the application
EXPOSE 5000

## 6. Define the command to run when the container starts.
## 如何执行应用程序
CMD ["python","flask-demo/app.py"]

##teminal: 最后一个点表示寻找路径下的Dockerfile文件
##docker build -t flask-demo:0.0.1 .
```

- #docker build -t flask-demo:0.0.1 . (创建镜像)
- docker run -p 3000:5000 flask-demo:0.0.1 - 运行web docker container

  `3000:5000` 表示将容器内部的端口 `5000` 映射到主机上的端口 `3000`。这意味着可以通过主机的 `3000` 端口访问容器中运行的应用程序。

11. docker inspect {imageID} - 用于获取有关 Docker 对象（如容器、镜像、网络等）的详细信息
12. 往自己的dockerhub上传镜像
    - docker build -t hjyyyyy22/grade-submission:flask-0.0.1 .
    - docker push hjyyyyy22/grade-submission:flask-0.0.1
13. 拉下dockerhub的镜像
    - docker pull hjyyyyy22/grade-submission:flask-0.0.1
    - docker inspect {imageID} (查看端口号)
    - docker run -p 3000:5000 hjyyyyy22/grade-submission:flask-0.0.1 （执行web）
14. docker system prune -a （清理所有无用的资源，包括未被引用的镜像、容器、网络和卷）
15. 有的container需要设置环境变量，如需要mysql的密码

    docker run -p 8000:3000 -e DATABASE_HOST=host.docker.internal -e DATABASE_USER=user -e DATABASE_PASSWORD=password -e DATABASE_NAME=db flask-mysql:latest

16. 直接拉取新的mysql docker container，并创建mysql库
    - docker pull mysql/mysql-server:8.0
    - docker run -e MYSQL_DATABASE=db -e MYSQL_USER=user -e MYSQL_PASSWORD=password mysql/mysql-server:8.0
17. 可以直接使用compose，不需要mysql和web开两个terminal

```
version: '3.3'
services:
  mysql:
    image: mysql/mysql-server:8.0
    environment:
      MYSQL_DATABASE: 'db'
      MYSQL_USER: 'user'
      MYSQL_PASSWORD: 'password'
    ports:
      - '3306:3306'
    volumes:
      - new-db:/var/lib/mysql #数据存在主机里，docker down之后数

  flaskapp:
    image: grade-submission:0.0.1 # your image here
    depends_on:
      - mysql
    environment:
      DATABASE_HOST: 'mysql'
      DATABASE_USER: 'user'
      DATABASE_PASSWORD: 'password'
      DATABASE_NAME: 'db'
    ports:
      - '8080:8080'
volumes:
  new-db:
```

Terminal:

up - docker compose up

down - docker compose down

18. volumes：卷
    - 匿名卷
        - 由 Docker 自动创建的，无需用户明确指定名称
        - 当容器被删除时，相关联的匿名卷也会被自动删除
    - 命名卷
        - 由用户显式地创建并命名
        - 即使容器被删除，命名卷仍然存在
    - commands
        - docker volume ls
        - docker volumn rm {volumeName}
19. 进入mysql的container
    - docker exec -it {containerID} bash
    - mysgl -u root -pdees