**Docker学习笔记**

1. **Docker介绍：**
2. **Docker 镜像 (Image)**

Docker 镜像 是一个只读的模板，用来创建 Docker 容器。镜像包含了运行应用程序所需的所有内容，包括操作系统、应用程序代码、运行时、库、环境变量和配置文件。镜像是分层的，每一层都基于前一层，并且每一层都代表一次文件系统的变更。

特点：

不可变：镜像是只读的，一旦创建就不能修改。

分层存储：每个镜像由多个层组成，使用联合文件系统（如 AUFS、OverlayFS）进行管理。

共享与复用：相同的镜像层可以在多个镜像间共享，节省存储空间。

常用操作：

**拉取镜像：**从镜像仓库（如 Docker Hub）下载镜像。

docker pull ubuntu:latest

**列出镜像**：查看本地存储的所有镜像。

docker images

**删除镜像**：删除本地存储的镜像。

docker rmi ubuntu:latest

**构建镜像**：使用 Dockerfile 构建新的镜像。

docker build -t myapp:latest .

1. **Docker 容器 (Container)**

Docker 容器 是基于镜像创建的可运行实例。容器包含了应用程序及其依赖项，并且在隔离的环境中运行。容器利用了操作系统级的虚拟化技术，通过共享主机的操作系统内核实现轻量级的隔离。

特点：

轻量级：容器共享主机操作系统内核，启动和运行非常高效。

可移植：容器可以在任何支持 Docker 的环境中运行，实现跨平台的一致性。

独立性：每个容器运行在其独立的环境中，互不干扰。

常用操作：

运行容器：基于镜像创建并启动一个容器。

docker run -d -p 80:80 nginx:latest

**列出运行中的容器**：查看当前正在运行的容器。

docker ps

**停止容器**：停止一个运行中的容器。

docker stop <container\_id>

**删除容器**：删除一个已经停止的容器。

docker rm <container\_id>

1. **Docker的优势**
2. **一致性和可移植性**

环境一致性：Docker 容器包含了应用程序及其所有依赖项，因此无论在开发、测试还是生产环境中，应用程序的运行环境都是一致的。这消除了“在我机器上能跑”的问题。

跨平台可移植性：Docker 容器可以在任何支持 Docker 的平台上运行，包括本地开发环境、测试服务器和云平台，确保应用程序在不同环境中的行为一致。

1. **资源效率和性能**

轻量级：与虚拟机相比，Docker 容器共享主机操作系统内核，启动速度更快，资源开销更小。因此，单台物理机可以运行更多的容器实例。

高效利用资源：Docker 容器的资源利用率高，启动和停止容器非常快速，这使得在资源受限的环境中也能高效运行。

1. **简化的依赖管理**

依赖打包：Docker 镜像包含了应用程序运行所需的所有依赖，包括库和配置文件。这意味着你只需要一次性打包所有依赖，然后可以在任何地方运行。

版本控制：Docker 镜像可以版本化，每次构建新的镜像时都可以生成唯一的版本标签，从而确保依赖和应用程序版本的一致性和可追溯性。

1. **隔离性和安全性**

进程隔离：每个容器在一个独立的进程空间中运行，不会影响其他容器。这种隔离性确保了不同应用程序之间的互不干扰。

安全性：Docker 利用 Linux 内核的命名空间和控制组技术提供隔离性，确保容器之间和容器与主机之间的隔离。

1. **持续集成和持续部署（CI/CD）**

简化 CI/CD 流程：Docker 容器易于集成到持续集成和持续部署流水线中，可以自动化构建、测试和部署流程。容器的一致性和可移植性使得在不同阶段使用相同的镜像成为可能。

快速回滚：由于 Docker 镜像是版本化的，可以轻松回滚到之前的版本，确保在出现问题时快速恢复。

1. **微服务架构支持**

微服务部署：Docker 非常适合微服务架构，每个服务可以独立打包成一个容器，从而实现独立部署、扩展和管理。

容器编排：借助 Kubernetes、Docker Swarm 等容器编排工具，可以轻松管理大规模分布式应用，实现服务发现、负载均衡、自动伸缩等功能。

1. **开发和测试便利性**

快速启动和停止：容器可以在几秒钟内启动和停止，非常适合开发和测试环境，节省时间。

隔离的开发环境：每个开发人员可以在自己的隔离环境中进行开发和测试，避免相互之间的依赖冲突。

1. **社区和生态系统**

丰富的生态系统：Docker 拥有庞大的社区和生态系统，包括 Docker Hub 上丰富的镜像仓库，可以方便地获取和使用各种应用镜像。

活跃的社区支持：有大量的社区资源和支持，可以帮助开发者快速解决问题并获得最佳实践建议。

1. **打包Docker镜像**
2. **编写Dockerfile**

Dockerfile是一个包含一系列指令的文本文件，这些指令定义了Docker镜像的内容和设置。以下是一个简单的示例Dockerfile，用于一个Python应用程序：

# 使用官方的 Python 3.9 版本作为基础镜像

FROM python:3.9.17-slim

# 设置工作目录

WORKDIR /app

# 复制 requirements.txt 并安装依赖

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

# 安装 Tkinter 和 OpenCV 依赖的库

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

libgl1-mesa-glx \

libglib2.0-0 \

python3-tk \

tk-dev \

&& apt-get clean \

&& rm -rf /var/lib/apt/lists/\*

# 复制项目文件到工作目录

COPY . .

# 设置环境变量

# ENV FLASK\_APP=app

# ENV FLASK\_RUN\_HOST=0.0.0.0

# 赋予启动脚本执行权限

# RUN chmod +x startup\_rag.sh

# RUN chmod +x startup.sh

# 暴露应用运行的端口（可选，因为我们会在 docker-compose.yml 中指定端口）

EXPOSE 6999

EXPOSE 16665

# 设置默认命令，可以在 docker-compose.yml 中覆盖

CMD ["python", "main\_wechat.py"]

1. **编写docker-compose.yml文件**

Docker Compose 是一个用于定义和运行多容器 Docker 应用的工具。通过 Compose，你可以使用一个 YAML 文件来配置应用程序的服务。然后，使用一个单独的命令来创建和启动所有服务。

version: '3.8'

services:

itchat:

build: .

ports:

- "6999:6999"

volumes:

- .:/app

command: ["python", "main\_wechat.py"]

rag:

build: .

ports:

- "16665:16665"

volumes:

- .:/app

command: ["python", "main\_wechat.py", "-r"]

1. **构建Docker镜像**

使用docker build命令基于Dockerfile构建镜像。你需要在包含Dockerfile的目录中运行此命令：

docker build -t my\_image:latest .

或docker-compose build

docker images

1. **运行Docker容器**

使用docker run命令基于新镜像运行容器，以验证其功能：

docker run -d --name my\_container -p 8000:8000 my\_image:latest

或docker-compose up server\_name -d

· -d 选项表示以分离模式运行容器。

· --name 选项用于指定容器的名称。

· -p 选项用于将主机的端口映射到容器的端口。

1. **推送镜像到镜像仓库**

首先，使用以下命令登录到Docker Hub：

docker login

然后，使用docker tag命令标记镜像，以匹配仓库格式：

docker tag my\_image:latest your\_dockerhub\_username/my\_image:latest

使用docker push命令将镜像推送到Docker Hub：

docker push your\_dockerhub\_username/my\_image:latest

1. **拉取镜像**

其他用户可以使用docker pull命令从Docker Hub拉取镜像：

docker pull your\_dockerhub\_username/my\_image:latest

1. **终止服务**

构建后启动容器

docker-compose up server\_name -d

停止并删除所有容器

docker-compose down

暂停某服务

docker-compose stop rag

列出所有运行的服务

docker-compose ps

显示所有服务的日志

docker-compose logs

1. **导出镜像**

导出镜像

docker save -o my\_flask\_app\_v1.tar my\_flask\_app:latest

将文件传输到其他 PC（以 SCP 为例）

scp my\_flask\_app\_v1.tar user@other-pc:/path/to/destination

在其他 PC 上导入镜像

docker load -i /path/to/destination/my\_flask\_app\_v1.tar