docker部署

1.基本概念

1.0核心理念及应用场景

核心理念：“**构建一次，随处运行**”

> Docker的核心价值在于将**应用及其所有依赖**封装为一个标准化的单元（容器），实现环境一致性、可重复部署和跨平台移植性。

适用场景：1.**跨架构**构建环境难以解决依赖的时候；2.需要**集群化**部署相同环境的时候

场景1：当项目需要从x86架构的Windows系统中迁移到ARM架构的Linux系统中时，必须要重新构建环境。这是因为许多包是针对特定操作系统和架构预编译的，操作系统还好说，利用虚拟机虚拟一个相同的操作系统可以解决这个问题，但无法解决宿主机的架构问题，所以必须构建环境。

但如果这个新架构和新操作系统很难配置需要的环境（比如这个Jetson orin nx边缘计算卡，其操作系统内核就是阉割过的，其架构也是ARM与平时的x86不同，需要的pytorch版本找不到），就可以找官方配置好的镜像（比方说nvidia官方专门为这个设备环境配置好的可以用pytorch+cu124一个镜像），在这个镜像的基础上构建的容器，可以解决80%的环境依赖问题，然后再继续配剩下的环境依赖就很容易了。

场景2：像场景1那样配置环境肯定不想在相同的Jetson计算卡上一遍遍配相同的环境，刚刚配好的环境就可以打包为一个镜像，在相同架构相同操作系统内核下就可以重复部署，不用重复造轮子。

参考链接：40分钟的Docker实战攻略，一期视频精通Docker*哔哩哔哩*bilibili

1.1镜像

镜像是**只读的模板**，包含了运行应用所需的所有文件系统、依赖库、环境变量和配置。镜像采用分层存储结构，每一层都是对前一层文件系统的增量修改。

1.2容器

容器是镜像的**运行实例**，在镜像的基础上添加一个可写层（容器层），提供隔离的运行环境。容器共享宿主机的内核，但拥有独立的文件系统、进程空间和网络接口。

1.3镜像仓库

镜像仓库是**集中存储和分发Docker镜像的服务**，类似代码仓库（如GitHub）管理源代码。包含公共仓库（如Docker Hub）和私有仓库（如Harbor、AWS ECR）

2.参考流程

2.1安装docker并添加docker权限

Jetson中自带docker,安装这步可跳过。可采用安装脚本安装docker,用官网安装不上，可以用阿里云镜像

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh --mirror Aliyun
```

docker的权限接近root，默认情况下非root用户没有docker的权限，所以需要配置一下

```
# 将当前用户添加到docker组
sudo usermod -aG docker $USER
# 查看用户是否已添加到docker组
groups $USER  # 应该能看到docker组
# 重要：刷新组权限（否则需要重新登录）
newgrp docker

# 测试：不用sudo运行docker命令
docker version
```

2.2配置镜像仓库

由于docker官方镜像仓库用不了，需要配置一下镜像仓库

```
# 1. 创建配置目录（如果有直接cd）
```

```
sudo mkdir -p /etc/docker

# 2. 创建配置文件
sudo bash -c 'cat > /etc/docker/daemon.json << EOF
{
  "registry-mirrors": [
    "https://registry.hub.docker.com",
    "https://docker.m.daocloud.io",
    "https://dockerproxy.com"
  ]
}
EOF'

# 3. 重启Docker使配置生效
sudo systemctl daemon-reload
sudo systemctl restart docker
```

测试一下能否拉取镜像并运行，这里似乎是由于Jetson系统内核模块的缺失，需要加上参数--network host

```
docker pull hello-world
docker run --network host hello-world
```

2.3拉取基础镜像

```
https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch
https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-pytorch?
version=r35.2.1-pth2.0-py3
```

这两个网站是英伟达官方的构建pytorch的容器镜像仓库，第二个更是针对Jetson的容器镜像仓库。两个都提供arm架构的镜像，可以用来当作基础镜像。

我用的是第一个镜像仓库里的镜像作为我的基础镜像：

```
docker pull nvcr.io/nvidia/pytorch:24.09-py3
```

拉取后运行容器

```
docker run -it --rm --runtime nvidia --network host -v $(pwd):/workspace
nvcr.io/nvidia/pytorch:24.09-py3 bash
```

运行后会**报错**：英伟达驱动版本不匹配，容器基于560.35构建而板卡是540.4.0，但后续证明这是兼容的，**不用理会。**

```
ERROR: This container was built for NVIDIA Driver Release 560.35 or later,
but
       version 540.4.0 was detected and compatibility mode is UNAVAILABLE.
```

2.4在容器中配置剩余的环境依赖

```
#  其余依赖
pip install numpy pandas matplotlib scikit-learn openpyxl xlrd xlwt xlutils
APScheduler joblib speedtest-cli
# 容器要与外界通信，也要安装网络工具，其余工具也同理
apt update && apt install -y iputils-ping net-tools curl wget
```

以上是我项目除了pytorch还需要的包，以及需要的网络工具,安装成功后就能成功运行代码了，环境成功部署

2.5打包该容器为项目环境的镜像

现在容器内的环境已经成功部署，但就此退出的话下次还得重新部署环境，因为**容器是镜像的实例**，所以需要把这个容器打包为新镜像，下次才不需要重新部署环境，也可以方便地在其他设备部署。

1.不要关掉运行中的容器，打开一个新的宿主机终端，**查看目前的容器id**:

```
docker ps

#    结果大致如下
CONTAINER ID   IMAGE                                COMMAND
CREATED         STATUS         PORTS     NAMES
adebd9f50686   nvcr.io/nvidia/pytorch:24.09-py3   "/opt/nvidia/nvidia_…"
5 minutes ago   Up 5 minutes           quirky_kowalevski
```

2.复制容器ID，**构建新的容器镜像**：

```
# docker commit 容器ID 镜像名（全小写）
docker commit adebd9f50686 satinfer_v1.0

# 可以在宿主机看到新的镜像：
nvidia@nvidia-desktop:~$ docker images
                                                        i Info →   U
In Use
IMAGE                          ID             DISK USAGE   CONTENT SIZE
EXTRA
hello-world:latest             d4aaab6242e0       22.5kB         10.2kB
U
nvcr.io/nvidia/pytorch:24.09-py3
                               0603cdfa7c20       26.3GB         8.74GB
U
satinfer_v1.0:latest           faf0b64188fc       26.4GB         8.79GB

ubuntu:22.04                   104ae83764a5        108MB         29.4MB
```

2.6一键部署环境

后续就可以用这个镜像一键部署需要的环境了：

```
nvidia@nvidia-desktop:~/satinfer/SatCom-Infer$ docker run -it --rm --
runtime nvidia --network host -v $(pwd):/workspace satinfer_v1.0:latest
bash
```

> 注意：容器的工作目录是挂载在宿主机下的（-v $(pwd):/workspace），也就是说在同一个目录下挂载两个以上容器的话，两个容器会对同一个目录的文件操作，要注意程序输出不会互相冲突