راهنمای کامل داکرایز کردن پروژه پایتون

داکر چیست و چگونه از این راهنما استفاده کنیم:

داکر چیست (به زبان ساده)؟

داکر ابزاری است که به شما اجازه میدهد پروژه ی خود را (code + environment + dependencies) داخل یک محفظه بسته بندی کنید (این محفظه همانند جعبه ای سبک و قابل حمل است که به طور ثابت و بدون تغییر روی سایر کامپیوترها قابل اجرا می باشد).

تصور کنید یک نسخه اجرایی قابل حمل از پروژه ی خود ساخته اید که در آن شخص مقابل به جای اجرای مرحله به مرحله ی خط فرمان های زیر:

"Install Python 3.10, ffmpeg, PyTorch, and run these 20 commands…"

با اجرای "Run this container" می تواند پروژه ی شما را روی سیستم خود همانندسازی کند.

چرا از داکر استفاده کنیم؟

- جلوگیری از مسائلی مانند " روی سیستم من که کار میکنه!".
- هر پروژه ای را می توان به راحتی با دیگران به اشتراک گذاشت (سیستم عامل و تنظیمات یکسان).
- تکرارپذیری مراحل ساخت و انجام پروژه.
- محیط های مجزا (از ایجاد اختلال در فضای سیستم شما جلوگیری می کند).
- قرارگیری آسان تر در سرورها، فضای ابری، CI/CD.

نقشه راه :داکرایز کردن پروژه از صفر تا صد

به کمک این راهنما شما می توانید پوشه معمولی حاوی پروژه ی خود را به یک اپلیکیشن کامل داکرایز شده، که قابل اجرا و اشتراک گذاری در هر سیستمی می باشد، تبدیل کنید.

مرحله اول: داشتن پروژه ای که روی سیستم خودتان قابل اجرا می باشد.

پیش از شروع بهتر است موارد زیر را داشته باشید:

- کد مرجع (به عنوان مثال: اسکریپت پایتون).
- داشتن فایل requirement.txt و یا داشتن لیستی از وابستگی ها.
- فولدر نمونه برای فایل های ورودی و خروجی (داشتن این موارد اختیاری است و در عین حال بسیار مفید می باشد).

مرحله دوم: نصب داکر

- Docker Desktop و WSL 2 را نصب کنید (جهت توضیحات بیشتر به بخش ۱ مراجعه کنید).
- از فرمان های زیر جهت بررسی درست بودن نصب استفاده کنید:

```
bash

docker --version
wsl --list --verbose
```

مرحله سوم :سازمان‌دهی ساختار پروژه

ساختار فولدر و فایل های پروژه ی خود را به صورت زیر تنظیم کنید:

```
bash

project-root/
├── Dockerfile        # The instructions for building the Docker image
├── requirements.txt  # Python packages
├── your_script.py    # Your main script (or entry point)
├── input_folder/     # Optional: folder for inputs
└── .dockerignore     # Optional: tells Docker what to exclude
```

این ساختار قابل تغییر است. برای مثال‌ها و توضیحات بیشتر به بخش ۲ مراجعه کنید.

مرحله چهارم: نوشتن داکرفایل (Dockerfile)

نحوه ی ساخت تصویر (image) شما درون داکرفایل (dockerfile) تعیین می شود.

- Image پایه ی خود را انتخاب کنید (e.g., Python, PyTorch, Node.js, etc.)
- نصب وابستگی ها
- از دستور Copy جهت انتقال کد خود استفاده کنید.
- نقطه ی ورود را تنظیم کنید.

این راهنما از یک رویکرد ماژولار با چندین Dockerfile استفاده می‌کند (برای اشکال‌زدایی عالی است). اما شما می‌توانید جهت داکرایز کردن پروژه ی خود فقط از یک Dockerfile استفاده کنید.
(بخش های ۲ و ۳، مطالعه شود).

مرحله ی پنجم: ساخت تصویر داکر

جهت ساخت docker image دستور زیر را اجرا کنید:

```
bash
docker build -f Dockerfile -t myimage:tag .
```

- در این فرمان Dockerfile را با نام فایل خود جایگزین کنید (به عنوان مثال: Dockerfile.final).
- به جای tag نام مورد نظر خود برای image را جایگزین کنید.

برای یادگیری نحوه تست و رفع اشکال تصویر به بخش ۲.۳ مراجعه کنید.

مرحله ی ششم: اجرای کانتینر (container)

جهت اجرای کانتینر خود می توانید از فرمان زیر استفاده کنید:

```
bash

docker run --rm \
  -v "${PWD}/input_folder:/app/input_folder" \
  myimage:tag argument1 argument2
```

این دستور اسکریپت شما را در محیط کانتیتر و با استفاده از دیتای خودتان اجرا می کند.
جهت مشاهده ی مثال های بیشتر به بخش ۴,۱ مراجعه کنید.

مرحله ی هفتم: اشتراک‌گذاری، انتقال یا تهیه نسخه پشتیبان

جهت انتقال پروژه ی داکرایزشده ی خود به سیستم دیگر ابتدا آن را به صورت فایل با پسوند **tar.** فشرده کنید:

```
bash

docker save -o myimage.tar myimage:tag
```

جهت بارگزاری مجدد روی سیستم دیگر:

```
bash

docker load -i myimage.tar
docker run ...
```

برای جزئیات بیشتر به بخش ۵ مراجعه کنید.

این قسمت از سند نقشه راه کلی داکرایز کردن پروژه ی پایتون بود و در ادامه جزئیات بیشتر مربوط به هر مرحله آمده است.

بخش ۱ :نصب داکر و اولین قدم‌ها

۱-۱ موارد نصب‌شده

- **Docker Desktop for Windows** (رابط کاربری گرافیکی/مدیر اصلی سرویس برای داکر در ویندوز).
- **WSL 2 backend** (داکر در ویندوز از WSL 2 برای سازگاری با لینوکس استفاده می کند).
- بررسی وضعیت نصب با فرمان های زیر:

```
bash

docker --version          # Check Docker installation
wsl --list --verbose      # Verify WSL distros & version
```

۱-۲ چالش های احتمالی اولیه در زمان نصب داکر

- مشکلات مربوط به Permissions/Admin: داکر به عنوان admin اجرا نمی شود، Apply و Restart کار نمی کنند.
- مشکلات مربوط به WSL: غیرفعال بودن اوبونتو، WSL نسخه ی پیش فرض نیست.

۱-۳ راه‌حل‌ها و رفع مشکلات

- Docker Desktop فقط با WSL2سازگار است و انجام مراحل زیر ضرورت دارد:
- **فعال سازی ادغام WSL**:
- Open **Docker Desktop** → go to **Settings** → **Resources** → **WSL Integration**
- دسترسی داکر را برای توضیع WSL نصب شده ی خود فعال کنید (به عنوان مثال: Ubuntu-20.04).
- این امر به داکر اجازه می‌دهد تا از طریق WSL 2 به طور یکپارچه با محیط‌های مبتنی بر لینوکس در ویندوز کار کند.
- **Ran Docker Desktop as Administrator:**
- Right-click the Docker Desktop icon → "Run as Administrator"
- برخی از تنظیمات و سرویس‌های پس‌زمینه (مانند شبکه) ممکن است برای عملکرد صحیح به مجوزهای سطح بالا نیاز داشته باشند.
- **تنظیم مجدد تنظیمات داکر (عیب‌یابی پیشرفته):**
- اگر داکر رفتار عجیبی داشت یا شروع به کار نکرد، می‌توانید پیکربندی آن را ویرایش کنید:

```
makefile

C:\Users\<YourUser>\AppData\Roaming\Docker\settings.json
```

«هشدار: فقط در صورتی این کار را انجام دهید که سایر راه‌حل‌ها جواب ندادند. قبل از ویرایش این فایل، از آن نسخه پشتیبان تهیه کنید. تنظیم مجدد می‌تواند مشکلات مربوط به تنظیمات خراب یا وضعیت بد را حل کند.

بخش ۲: ساختار و طراحی پروژه

۱-۲ ساختار پروژه

```
text

project-root/
├── Dockerfile.base        # Base OS setup
├── Dockerfile.sysdeps     # System libraries
├── Dockerfile.pip         # Python dependencies
├── Dockerfile.final       # Final image + project code
├── requirements.txt       # Python packages
├── Raya_Pipeline.py       # Main Python script
├── RealAudio/             # Input audio files
└── .dockerignore          # Exclude files from Docker build context
```

هر داکرفایل یک تصویر جداگانه می‌سازد. می‌توانید آنها را جداگانه برچسب‌گذاری و استفاده کنید یا برای اشکال‌زدایی بهتر، آنها را به تدریج بسازید. هر داکرفایل مربوط به یک لایه در ساخت نهایی است که جداسازی اشکالات، استفاده مجدد از مراحل ساخت و بازسازی فقط بخش‌هایی که تغییر می‌کنند را، آسان‌تر می‌کند.

- اشکال‌زدایی و جداسازی آسان‌تر
- بازسازی سریع‌تر (ذخیره‌سازی لایه‌ها)
- طراحی ماژولار
- مسئولیت‌های واضح لایه‌ها
- الهام گرفته از:
- **https://www.divio.com/blog/guide-using-multiple-dockerfiles/**

گزینه اختیاری و جایگزین : استفاده از ساخت چندمرحله‌ای برای کاهش حجم تصویر:

اگر می‌خواهید تصویر نهایی داکر خود را مینیمال نگه دارید و ابزارها، لاگ‌ها یا تولیدات میانی غیرضروری را حذف کنید، استفاده از ساخت‌های چندمرحله‌ای را در نظر بگیرید.

به عنوان مثال:

Dockerfile

```
FROM pytorch/pytorch:2.2.2-cuda12.1-cudnn8-runtime as builder
RUN apt-get update && apt-get install -y ffmpeg
RUN pip install -r requirements.txt

FROM pytorch/pytorch:2.2.2-cuda12.1-cudnn8-runtime
COPY --from=builder /usr/local /usr/local
COPY . /app
ENTRYPOINT ["python", "Raya_Pipeline.py"]
```

ساخت‌های چندمرحله‌ای امکان جداسازی منطق زمان ساخت (build-time) از زمان لازم برای اجرای (runtime) تصویر نهایی را فراهم می‌کنند و اندازه و پیچیدگی را کاهش می‌دهند. این به شما امکان می‌دهد مراحل نصب سنگین را در یک مرحله ساخت موقت نگه دارید و فقط بخش‌های لازم را در تصویر نهایی کپی کنید (آن را تمیز و کوچک‌تر نگه دارید).

۲-۳ انتخاب ایمیج پایه مناسب

text

```
pytorch/pytorch:2.2.2-cuda12.1-cudnn8-runtime
```

- دلیل انتخاب این ایمیج پایه برای این پروژه:
- cuda12.1 برای شتاب‌دهی با GPU مورد نیاز است.
- cudnn8 توسط مدل‌های یادگیری عمیق استفاده می‌شود.
- از پیش نصب شده با: PyTorch + CUDA
- مبتنی بر اوبونتو = نصب آسان بسته‌های سیستمی

«شما می‌توانید سازگاری بین نسخه‌های PyTorch، CUDA و درایورها را با استفاده از ماتریس سازگاری PyTorch بررسی کنید.»

بخش ۳ :ساخت و تست تصویرهای داکر در این پروژه

۳-۱ استراتژی ساخت لایه‌ای

- **Dockerfile.base**: راه‌اندازی (Linux base + ابزارهای کاربردی)
- **Dockerfile.sysdeps**: نصب کتابخانه‌های سیستم (مثلاً`ffmpeg` ، `libsndfile1`)
- **Dockerfile.pip**: نصب بسته‌های پایتون (`requirements.txt`،`constraints.txt`)
- **Dockerfile.final:** کپی فایل‌های پروژه و تنظیم نقطه ورود

یکی از داکرفایل ها به عنوان نمونه:

Dockerfile

```
# Dockerfile.pip
FROM myimage:sysdeps

COPY requirements.txt constraints.txt /app/
WORKDIR /app

RUN pip install --upgrade pip setuptools wheel
RUN pip install --use-pep517 -r requirements.txt -c constraints.txt
```

جهت تکرارپذیری می توانید از یک فایل constraints.txt برای قفل کردن نسخه‌های بسته استفاده کنید.

۳-۲ تست Dockerfile ها به‌صورت مرحله‌به‌مرحله

- **دسترسی تأیید شده به GPU داخل کانتینر:**

پس از اجرای تعاملی یک کانتینر، می‌توانید با اجرای دستور زیر تأیید کنید که آیا PyTorch پردازنده گرافیکی (GPU) شما را شناسایی می‌کند یا خیر (این دستور بررسی می‌کند که آیا GPU قابل دسترسی است و CUDA به درستی درون کانتینر پیکربندی شده است یا خیر):

```
bash
python -c "import torch; print(torch.cuda.get_device_name(0))"
```

اگر با خطایی مانند "cuda در دسترس نیست مواجه شدید":

- مطمئن شوید که از پرچم (flag) gpus all-- استفاده شده است.
- بررسی کنید که NVIDIA Container Toolkit روی سیستم میزبان شما نصب شده باشد.

نمونه جایگزاری gpus all– در فرمان اجرا:

```
bash
docker build -f Dockerfile.base -t myimage:base .
docker run --rm --gpus all -it myimage:base bash
```

- اگر کاربران گزارش‌های کامل ساخت را می‌خواهند، از progress=plain-- استفاده کنید.
- در صورت استفاده، برای ساخت‌های چند مرحله‌ای از target-- استفاده کنید.

۳-۳ استراتژی اشکال‌زدایی (Debugging):

- اجرای کانتینرها به صورت تعاملی (bash)
- نصب دستی بسته‌های پایتون
- بررسی pip، torch و cuda درون کانتینر
- حافظه پنهان (builder cashe) پاکسازی شده:

```bash
# Clean only build cache (safer)
docker builder prune --force

# Clean everything (use with caution)
docker system prune -a --volumes
```

بخش ۴: اجرای کانتینر

۴-۱ ENTRYPOINT در برابر CMD

ENTRYPOINT مانند بخش ثابت دستور است. CMD پیش‌فرض اختیاری است.

مثال اجرایی:

```Dockerfile
ENTRYPOINT ["python", "Raya_Pipeline.py"]   # Always runs the script
CMD ["arg1", "arg2"]                        # Default args (can override)
```

در این نمونه به دلیل وجود ENTRYPOINT، input.wav را به عنوان اولین آرگومان به Raya_Pipeline.py ارسال می‌کند.

یک نمونه استاندارد اجرایی با ENTRYPOINT:

```bash
docker run --rm --gpus all \
  -v "${PWD}/RealAudio:/app/RealAudio" \
  myimage:full \
  /app/RealAudio/input.wav
```

${PWD} به معنی «دایرکتوری کاری فعلی» است - پوشه‌ای که این برنامه را از آن اجرا می‌کنید. می‌توانید آن را با یک مسیر مطلق مانند C:/Users/JohnDoe/Project در ویندوز جایگزین کنید. ${PWD}/RealAudio (پوشه میزبان) در /app/RealAudio/ (پوشه کانتینر) نصب شده است.

با توجه به اینکه داکرفایل حاوی این فرمان است:

```
Dockerfile
ENTRYPOINT ["python", "Raya_Pipeline.py"]
```

بقیه خط (/app/RealAudio/input.wav) به عنوان یک آرگومان به اسکریپت ارسال می‌شود، مانند:

```
bash
python Raya_Pipeline.py /app/RealAudio/input.wav
```

نادیده گرفتنENTRYPOINT (استفاده پیشرفته):

```
bash

docker run --rm --gpus all \
  myimage:full \
  python /app/Raya_Pipeline.py /app/RealAudio/input.wav
```

- این دستور صراحتاً اسکریپت پایتون را فراخوانی می‌کند و ENTRYPOINT را نادیده می‌گیرد.
- هنگام آزمایش یا اگر می‌خواهید کنترل بیشتری روی آنچه اجرا می‌شود داشته باشید، مفید است.

۲-۴ نصب (mount) کردن یک ولوم (volume) برای ورودی و خروجی

ولوم (volume) های داکر به شما امکان می‌دهند پوشه‌ها را بین سیستم میزبان و کانتینر به اشتراک بگذارید، بنابراین داده‌ها پس از توقف کانتینر همچنان باقی می‌مانند.

نمونه اجرایی با پوشه های ورودی و خروجی:

```
bash

docker run --rm \
  -v "${PWD}/RealAudio:/app/RealAudio" \
  -v "${PWD}/outputs:/app/outputs" \
  myimage:full input.wav
```

توضیح این خط فرمان:

- **-v "${PWD}/RealAudio:/app/RealAudio"**
  - پوشه محلی RealAudio را در کانتینر در مسیر app/RealAudio/ نصب می‌کند.
  - کانتینر می‌تواند فایل‌های ورودی را از این دایرکتوری بخواند.
- **-v "${PWD}/outputs:/app/outputs"**
  - پوشه محلی outputs را در app/outputs/ نصب می‌کند.
  - کانتینر می‌تواند نتایج خروجی (مانند فایل‌های txt. یا wav.) را در اینجا بنویسد.

چرا این روش بهتر از کپی کردن فایل‌ها در تصویر (image) است؟

- نیازی نیست هر بار که ورودی‌ها تغییر می‌کنند، تصویر داکر را از نو بسازید.
- داده‌های ورودی/خروجی را خارج از کانتینر نگه می‌دارد (تمیزتر و انعطاف‌پذیرتر).
- کاربران می‌توانند با نصب پوشه خودشان، از فایل‌های خودشان استفاده کنند!

بخش ۵ :ذخیره و اشتراک‌گذاری

۵-۱ ذخیره ایمیج به‌صورت tar.

```
bash
docker save -o myimage_updated.tar myimage:full
```

یک ایمیج tar. می‌تواند بدون ارسال به داکر هاب، بین دستگاه‌ها به اشتراک گذاشته شود.

۵-۲ بسته بندی (bundle) کردن ایمیج و فایل‌ها

نمونه ی اجرایی در PowerShell:

```
powershell
Compress-Archive -Path myimage_updated.tar, RealAudio -DestinationPath bundle.zip
```

اگر Compress-Archive با خطا مواجه شد، برای فایل‌های بزرگ از 7-Zip یا WinRAR استفاده کنید.

۵-۳ بارگذاری و اجرای تصویر روی سیستم دیگر

قبل از اجرا، مطمئن شوید که در سیستم جدید درایورهای GPU و زمان اجرای CUDA با هم مطابقت دارند.

```
Powershell

# 1. Extract the tar file (if it was zipped or sent as an archive)
tar -xf myimage_updated.tar

# 2. Load the image into Docker
docker load -i myimage_updated.tar

# 3. Run the container with GPU access and mount your input folder
docker run --rm --gpus all `
  -v "${PWD}/RealAudio:/app/RealAudio" `  # Mount input folder
  myimage:full `                          # Use loaded image
  /app/RealAudio/test.wav                 # Input file to process
```

درس‌های آموخته‌شده و نکات

- جهت انعطاف پذیری از نصب فضا (volume mounting) استفاده کنید.
- ENTRYPOINT به طور پیش‌فرض آرگومان‌ها را اضافه می‌کند، نیازی به تکرار نام اسکریپت نیست.
- ”dockerignore.“ مانند gitignore. عمل می‌کند تا از کپی کردن فایل‌های غیرضروری در تصویر جلوگیری کند. بنابراین، به جلوگیری از کپی‌های بزرگ کمک می‌کند (با حذف فایل‌های بزرگ یا نامربوط (مانند/__pycache__ ، RealAudio/outputs/،*.log) در زمان ساخت صرفه‌جویی می‌کند.)
- در صورت عدم موفقیت به‌روزرسانی apt، از جایگزینی آینه‌ای (mirror replacement) استفاده کنید.
- قبل از حذف کردن تصاویر، می توانید از نسخه‌های پشتیبان آن ها را به صورت tar. نگه دارید.
- داکرفایل ها را به صورت جداگانه آزمایش کنید.
- در صورت امکان از تصاویر پایه ی سبک استفاده کنید.
- جهت تکرارپذیری آزمایش تمام مراحل و تغییرات را مکتوب کنید.
- جهت انتقال داده با حجم زیاد از volume ها به جای کپی کردن استفاده کنید.
- جهت پیوستگی نصب ها در ساخت ها مختلف، با استفاده از consraints.txt نسخه های pip را قفل کنید.

خطاهای رایج و روش‌های رفع آن‌ها

- نمی‌توان به سرویس داکر متصل شد: دسکتاپ داکر در حال اجرا نیست یا به دسترسی ادمین نیاز دارد.
- خطاهای ENTRYPOINT: اگر هم اسکریپت و هم مسیر ارسال شوند: خطای آرگومان دوگانه.
- Compress شناسایی نمی شود: از Compress-Archive به جای Compress استفاده کنید.
- خطای400 apt update: از Princeton mirror استفاده کنید.
    - ایجاد یک نسخه پشتیبان از /etc/apt/sources.list قبل از جایگزینی می‌تواند مفید باشد:

```
bash

# Switch Ubuntu apt mirrors to a faster/more reliable one
sed -i 's|archive.ubuntu.com|mirror.math.princeton.edu/pub/ubuntu|g' /etc/apt/sources.list
sed -i 's|security.ubuntu.com|mirror.math.princeton.edu/pub/ubuntu|g' /etc/apt/sources.list
```

- هشدارPkg_resources: قابل قبول است مگر اینکه پس از سال ۲۰۲۵ در PyTorch منسوخ شود.
- GPU در کانتینر وجود ندارد: نصب nvidia-container-runtime را بررسی کنید.
- از دسترسی به app/RealAudio/ جلوگیری می شود: از پرچم user– به دقت استفاده کنید یا تنظیمات دسترسی درون کانتینر را عوض کنید.

نقشه راه آینده

- **`docker-compose`**: برای تنظیمات چندکانتینره، یا در مواردی که از پرچم های GPU استفاده می شود، مسیرهای volume ها، تغییر محیط ها با پیچیدگی کمتر، و خودکارسازی فرایندهای ساخت.
- **Makefile**: خودکارسازی وظایف تکراری مانند build، run، prune، و save.
- **یکپارچه‌سازی و استقرار مداوم** :(CI/CD) پیشنهاد استفاده از GitHub Actions برای ساخت، آزمون و انتشار تصاویرDocker .
- **بهینه‌سازی تصویر:** از ابزارهایی مانند ` Dive ` برای تجزیه و تحلیل لایه‌های تصویر و کاهش اندازه استفاده کنید.

ضمیمه :دستورات کاربردی

```
bash

# List all images
docker images

# List all containers
docker ps -a

# See running containers
docker ps

# Enter running container
docker exec -it <container_id> bash

# Stop all
docker stop $(docker ps -q)

# Clean everything
docker system prune -a –volumes

# Save and load image
docker save -o myimage.tar myimage:tag
docker load -i myimage.tar

# Check GPU inside container
python -c "import torch; print(torch.cuda.get_device_name(0))"

# Show container logs
docker logs <container_id>

# Tag image before pushing
docker tag myimage:full myrepo/myimage:tag

# Push to Docker Hub
docker push myrepo/myimage:tag
```

واژه‌نامه

- **Image (ایمیج)**: یک تصویر از سیستم فایل و نرم‌افزار؛ تصویری لحظه‌ای (snapshot) از وضعیت سیستم فایل که شامل تمامی داده‌ها، وابستگی‌ها و نرم‌افزارهای لازم برای اجرای یک برنامه در محیط ایزوله است.
- **Base image (ایمیج پایه)**: در یک فایل Dockerfile، به‌عنوان پایه‌ای برای ساخت سایر ایمیج‌های کانتینر به‌کار می‌رود. این ایمیج با دستور FROM مشخص می‌شود و سیستم فایل و محیط اولیه‌ای را فراهم می‌کند که لایه‌های بعدی بر آن افزوده می‌شوند.
- **Container (کانتینر)**: نمونه‌ی در حال اجرای یک ایمیج است؛ یعنی محیطی ایزوله و مستقل که بر اساس یک ایمیج ساخته شده و برنامه در آن اجرا می‌شود.
- **Volume (حجم داده / ولوم)**: روشی برای به‌اشتراک‌گذاری فایل‌ها بین سیستم میزبان (host) و کانتینر است. با استفاده از Volume می‌توان داده‌ها را میان کانتینر و میزبان به‌صورت پایدار منتقل و ذخیره کرد.
- **ENTRYPOINT (نقطه ورود)**: دستوری است که مشخص می‌کند پس از راه‌اندازی کانتینر، کدام برنامه به‌عنوان فرمان اصلی اجرا شود. این دستور همواره اجرا می‌شود و باعث می‌گردد کانتینر مانند یک برنامه‌ی اجرایی مستقل رفتار کند.
- **CMD (فرمان پیش‌فرض)**: فرمانی است که در زمان اجرای کانتینر از یک ایمیج، به‌صورت پیش‌فرض اجرا می‌شود. این دستور انعطاف‌پذیری ایجاد می‌کند و به کاربر اجازه می‌دهد تا هنگام اجرای کانتینر، آن را نادیده گرفته و فرمان دلخواه خود را جایگزین کند.