ChatGPT

Dockerfile completo per il progetto BadAi (ComfyUI)

Per soddisfare i requisiti, predisponiamo un **Dockerfile** che parte dall'immagine base NVIDIA CUDA 12.1.1 su Ubuntu 22.04 e configura un ambiente completo per ComfyUI. Il Dockerfile installerà tutte le dipendenze di sistema e Python, clonerà il repository ComfyUI, installerà PyTorch (con supporto CUDA 12.1), torchvision, xFormers e le altre librerie Python richieste, quindi scaricherà i modelli indicati all'interno del container. Infine esporrà la porta **8188** e avvierà ComfyUI in modalità server su tale porta. Tutto avviene **internamente al container**, senza bisogno di mount esterni, rendendo l'immagine compatibile con Docker Desktop (Windows) con supporto GPU NVIDIA [1]. Di seguito spieghiamo le sezioni principali del Dockerfile in italiano, con riferimenti alle fonti ufficiali, e presentiamo il file completo.

Base image e dipendenze di sistema

Usiamo come base l'immagine `nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04`, che include CUDA 12.1 e cuDNN 8 su Ubuntu 22.04, garantendo compatibilità e prestazioni elevate per operazioni di deep learning su GPU [1]. Nella prima istruzione `RUN` del Dockerfile, aggiorniamo i pacchetti e installiamo **Python 3**, **pip**, **git**, **git-lfs**, **ffmpeg** e altri pacchetti di sistema necessari per ComfyUI. In particolare includiamo strumenti di sviluppo (come `build-essential` e `cmake`) e librerie come `libgl1-mesa-glx`, `libsm6`, `libxext6` e `libglib2.0-0` per assicurare il corretto funzionamento di dipendenze Python che potrebbero richiedere componenti grafici o compilazione nativa [2]. Dopo l'installazione, eseguiamo `git lfs install` per abilitare **Git LFS**, dato che i modelli di grandi dimensioni su HuggingFace usano Git LFS. Ecco un estratto della sezione iniziale:

```
# Immagine base con CUDA 12.1, cuDNN8 su Ubuntu 22.04
FROM nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04

# Installazione pacchetti di sistema: Python, pip, git, git-lfs, ffmpeg, ecc.
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y \
    python3 python3-pip git git-lfs ffmpeg wget curl build-essential cmake \
    libgl1-mesa-glx libglib2.0-0 libsm6 libxext6 \
 && rm -rf /var/lib/apt/lists/* \
 && git lfs install
```

Nell'esempio sopra abbiamo impostato `DEBIAN_FRONTEND=noninteractive` per evitare prompt interattivi durante l'installazione. Abbiamo incluso **wget** e **curl** poiché saranno utilizzati per scaricare i modelli più avanti [3]. Le librerie aggiuntive (`libgl1-mesa-glx`, `libsm6`, etc.) sono aggiunte a scopo precauzionale per soddisfare possibili dipendenze di librerie Python (ad esempio per gestire immagini o GUI) comunemente utilizzate in ambienti di visione artificiale.

Clonazione del repository ComfyUI

Dopo aver installato le dipendenze di sistema, cloniamo il **repository ufficiale ComfyUI** direttamente all'interno del container. Usiamo il comando `git clone` per scaricare il progetto ComfyUI da GitHub. Successivamente impostiamo la directory di lavoro corrente (`WORKDIR`) sulla cartella del progetto clonata, in modo da poter installare facilmente le dipendenze Python e avviare l'applicazione. Tutto questo avviene nel contesto dell'immagine Docker, senza necessità di montare directory dal host [4]. Ecco come appare questa parte:

```
# Clona il repository ufficiale ComfyUI
WORKDIR /opt
RUN git clone https://github.com/comfyanonymous/ComfyUI.git
WORKDIR /opt/ComfyUI
```

La cartella `/opt/ComfyUI` ora contiene il codice sorgente di ComfyUI. Secondo la documentazione, il Dockerfile clona il repository e installa le dipendenze Python necessarie utilizzando pip [4], cosa che faremo nel passo successivo.

Installazione di PyTorch (CUDA 12.1), torchvision, xFormers e altre librerie Python

All'interno della directory di ComfyUI, procediamo a installare le librerie Python richieste. In particolare:

- **PyTorch con supporto CUDA 12.1**: utilizziamo pip indicando l'**extra index** ufficiale di PyTorch per CUDA 12.1 (URL `https://download.pytorch.org/whl/cu121`) in modo da installare la versione GPU corretta di `torch`, insieme a `torchvision` e `torchaudio` [5]. Questo garantisce che PyTorch utilizzi CUDA 12.1 presente nell'immagine (in alternativa si potrebbe specificare una versione precisa compatibile, ma usando l'indice extra PyPI di PyTorch si ottiene automaticamente una build compatibile con cu121).
- **xFormers**: installiamo anche la libreria `xformers`, utile per ottimizzazioni di memoria/velocità nella diffusione stabile. La includiamo esplicitamente poiché non sempre è presente di default nei requirements. (Notiamo che negli esempi viene consigliato di evitare la versione 0.0.18 di xFormers; installeremo quindi l'ultima versione disponibile diversa da quella se necessario [5]).
- **Requirements di ComfyUI**: il file `requirements.txt` del repository contiene le altre dipendenze Python necessarie (ad esempio `numpy`, `transformers`, `safetensors`, ecc.). Eseguiamo `pip install -r requirements.txt` per installarle tutte [4].

Possiamo combinare queste installazioni in un'unica istruzione `RUN` per ottimizzare la build. Ad esempio, sfruttando pip possiamo installare **torch, torchvision, torchaudio** e **xformers** assieme alle dipendenze, specificando l'indice extra di PyTorch. Un esempio di comando combinato, ispirato a un Dockerfile esistente, è il seguente [5]:

```
# Installa PyTorch (CUDA 12.1), torchvision, torchaudio, xFormers e le altre
dipendenze Python
RUN pip3 install --no-cache-dir torch torchvision torchaudio xformers \
    --extra-index-url https://download.pytorch.org/whl/cu121 \
 && pip3 install --no-cache-dir -r requirements.txt
```

In questo modo garantiamo che `torch` e i relativi pacchetti vengano presi dal repository di PyTorch (compilati per CUDA 12.1) e poi installiamo tutte le altre librerie richieste da ComfyUI. La flag `--no-cache-dir` è opzionale, ma aiuta a contenere la dimensione dell'immagine evitando di salvare cache temporanee di pip.

Nota: In alternativa, avremmo potuto eseguire prima `pip install -r requirements.txt` e poi installare PyTorch separatamente. Tuttavia, includendo `torch` e `torchvision` nella lista, ci assicuriamo che pip non installi accidentalmente una versione CPU-only di torch (dal indice PyPI standard) ma usi direttamente quella GPU grazie a `--extra-index-url` [5]. Questo approccio è conforme alle indicazioni ufficiali per installare PyTorch su GPU NVIDIA [6].

Download interno dei modelli pre-addestrati

Una volta configurato l'ambiente e le librerie, il Dockerfile provvede a **scaricare i modelli richiesti** direttamente all'interno dell'immagine, posizionandoli nelle directory appropriate. Questo soddisfa la richiesta di avere i modelli disponibili nel container **senza montaggi esterni**. Utilizziamo `wget` con l'opzione `-c` (resume, utile in caso di ricostruzioni parziali) e `-P` per specificare la cartella di destinazione [3]. Di seguito i modelli da scaricare e la rispettiva destinazione:

- **Stable Diffusion XL Base 1.0** – file `sd_xl_base_1.0.safetensors` da collocare in `models/checkpoints/`. Questo è il modello base SDXL 1.0 rilasciato da Stability AI. Usiamo l'URL ufficiale di HuggingFace per SDXL Base 1.0 [7].
- **Flux 1.0 (Dev FP8)** – file `flux1-dev-fp8.safetensors` da collocare in `models/checkpoints/`. Questo modello "Flux" (formato FP8) è fornito dal team ComfyUI e disponibile su HuggingFace (repository `Comfy-Org/flux1-dev`) [8]. È un modello di esempio per pipeline text-to-image avanzate (in un unico file include UNet e text encoder per Flux).
- **VAE fine-tunata MSE (840000)** – file `vae-ft-mse-840000-ema-pruned.safetensors` da collocare in `models/vae/`. Si tratta di una **VAE** ottimizzata rilasciata da Stability AI, spesso utilizzata per migliorare la qualità delle immagini generate (derivata dal modello Stable Diffusion 1.5) [9].

Inseriamo quindi tre comandi `RUN wget` consecutivi nel Dockerfile per ottenere questi file. Ecco come appare la sezione relativa:

```
# Scarica i modelli Stable Diffusion XL Base, Flux1-dev FP8, e VAE MSE e posizionali nelle cartelle opportune
RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors -P ./models/checkpoints/ \
 && wget -c https://huggingface.co/Comfy-Org/flux1-dev/resolve/main/flux1-dev-fp8.safetensors -P ./models/checkpoints/ \
 && wget -c https://huggingface.co/stabilityai/sd-vae-ft-mse-original/resolve/main/vae-ft-mse-840000-ema-pruned.safetensors -P ./models/vae/
```

Come consigliato nella documentazione, stiamo inserendo i file **.safetensors** direttamente nelle directory previste da ComfyUI (ad es. *checkpoints* per i modelli principali e *vae* per i modelli VAE) [10]. L'uso di URL diretti di HuggingFace con `resolve/main` garantisce che scarichiamo il file effettivo. Questi URL sono quelli suggeriti dalle fonti ufficiali (ad esempio la guida di ComfyUI e i repository HuggingFace dei rispettivi modelli) [9] [8].

Nota: Il download di questi file è voluminoso (diversi GB, in particolare ~6.9GB per SDXL Base e ~11.9GB per Flux1 Dev). Assicurarsi di avere una connessione affidabile durante la build. Grazie all'opzione `-c`, se la build Docker viene interrotta durante un download, alla prossima esecuzione tenterà di riprendere anziché ricominciare da capo.

Esposizione della porta 8188 e avvio di ComfyUI

Infine, configuriamo il container perché all'avvio esegua ComfyUI in modalità server, in ascolto sulla porta **8188** su **tutti gli indirizzi** (0.0.0.0). Inseriamo nel Dockerfile:

- **EXPOSE 8188:** per dichiarare la porta su cui il container accetterà connessioni (utile per documentazione e tooling, anche se per l'effettiva esposizione all'host bisognerà avviare il container con il port mapping).
- **CMD** o **ENTRYPOINT:** per definire il comando di default che l'immagine eseguirà quando parte il container. In questo caso utilizziamo Python per lanciare ComfyUI. Dalla directory di ComfyUI, il comando tipico è `python3 main.py`. Aggiungiamo le opzioni `--listen 0.0.0.0` (per fare in modo che il servizio sia raggiungibile dall'esterno del container, e non solo su localhost interno [11]) e `--port 8188` per impostare la porta desiderata.

La documentazione conferma che per rendere ComfyUI accessibile in rete bisogna passare `--listen 0.0.0.0` agli argomenti di lancio [11]. Pertanto, la parte finale del Dockerfile sarà:

```
# Espone la porta 8188 per l'interfaccia web di ComfyUI
EXPOSE 8188

# Comando di avvio: lancia ComfyUI in ascolto su 0.0.0.0:8188
CMD ["python3", "main.py", "--listen", "0.0.0.0", "--port", "8188"]
```

Con questa configurazione, quando il container viene eseguito (ad esempio con `docker run --gpus all -p 8188:8188 nome/immagine`), ComfyUI partirà automaticamente e sarà raggiungibile dall'host tramite il browser all'indirizzo `http://localhost:8188`. L'opzione `--gpus all` in `docker run` è necessaria per abilitare l'accesso alla GPU sul host (Docker Desktop con WSL2 supporta questa funzionalità se la GPU NVIDIA è configurata correttamente).

Dockerfile finale

Di seguito riportiamo il **Dockerfile completo** assemblato, integrando tutte le sezioni illustrate sopra. Questo Dockerfile può essere utilizzato così com'è per costruire l'immagine Docker del progetto **BadAi** con ComfyUI:

```
# Dockerfile per progetto BadAi - Basato su NVIDIA CUDA 12.1.1 (Ubuntu 22.04)
FROM nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04

# Installa Python, pip, git, git-lfs, ffmpeg e librerie di sistema necessarie
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y \
    python3 python3-pip git git-lfs ffmpeg wget curl build-essential cmake \
    libgl1-mesa-glx libglib2.0-0 libsm6 libxext6 \
 && rm -rf /var/lib/apt/lists/* \
```

```
 && git lfs install

# Clona il repository ComfyUI
WORKDIR /opt
RUN git clone https://github.com/comfyanonymous/ComfyUI.git
WORKDIR /opt/ComfyUI

# Installa PyTorch (CUDA 12.1), torchvision, torchaudio, xformers e altre dipendenze Python
RUN pip3 install --no-cache-dir torch torchvision torchaudio xformers \
    --extra-index-url https://download.pytorch.org/whl/cu121 \
 && pip3 install --no-cache-dir -r requirements.txt

# Scarica i modelli (SDXL Base 1.0, Flux1-dev FP8, VAE MSE) nelle rispettive cartelle
RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors -P ./models/checkpoints/ \
 && wget -c https://huggingface.co/Comfy-Org/flux1-dev/resolve/main/flux1-dev-fp8.safetensors -P ./models/checkpoints/ \
 && wget -c https://huggingface.co/stabilityai/sd-vae-ft-mse-original/resolve/main/vae-ft-mse-840000-ema-pruned.safetensors -P ./models/vae/

# Espone la porta 8188 e avvia ComfyUI in modalità server
EXPOSE 8188
CMD ["python3", "main.py", "--listen", "0.0.0.0", "--port", "8188"]
```

Questo Dockerfile soddisfa tutte le richieste specificate:

- **Dipendenze di sistema installate:** Python 3, pip, git, git-lfs, ffmpeg e le librerie ausiliarie sono tutte installate all'interno dell'immagine [2].
- **Clonazione di ComfyUI:** il repository ufficiale viene scaricato nel container, evitando la necessità di volumi condivisi [4].
- **Installazione di PyTorch CUDA 12.1 e librerie Python:** il container installa PyTorch con supporto GPU (CUDA 12.1), torchvision, torchaudio, xformers e tutte le librerie elencate nel `requirements.txt` di ComfyUI [5].
- **Download dei modelli nel container:** i tre modelli richiesti (`sd_xl_base_1.0`, `flux1-dev-fp8` e `vae-ft-mse-840000-ema-pruned`) vengono scaricati e salvati nelle directory appropriate prima dell'esecuzione, rendendoli disponibili immediatamente all'avvio di ComfyUI [9] [8].
- **Porta 8188 esposta e servizio attivo:** la porta 8188 è esposta nell'immagine e ComfyUI viene lanciato con `--listen 0.0.0.0 --port 8188`, permettendo l'accesso all'interfaccia web da host e altre macchine sulla rete [11].

Riepilogo: Buildando questa immagine Docker e avviandola con il runtime NVIDIA, otterrete un container che esegue ComfyUI su GPU, accessibile via browser su porta 8188, senza necessità di configurare manualmente dipendenze o montare modelli dal filesystem locale. Buon utilizzo!

Fonti Utilizzate:

- Configurazione di un Dockerfile ComfyUI con CUDA 12.1 e installazione di dipendenze (esempio da HuggingFace) [2] [5] [7]

- Guida Krasamo – Dockerizzazione di ComfyUI con modelli pre-scaricati [4] [10] [3]
- Repository HuggingFace ufficiali per i modelli SDXL 1.0, Flux e VAE (percorsi di download utilizzati nel Dockerfile) [9] [8]
- Consigli dalla community per avviare ComfyUI in modalità server accessibile (`--listen 0.0.0.0`) [11]

---

[1] [3] [4] [10] Setting Up a Dockerized AI Environment with ComfyUI and NVIDIA CUDA | Krasamo
https://www.krasamo.com/comfyui-docker/

[2] [5] [7] [9] Dockerfile · Nymbo/ComfyUI-Demo at 0cc2ea691ddb45d5f16e9a8acc14f9d413f0a1f5
https://huggingface.co/spaces/Nymbo/ComfyUI-Demo/blame/0cc2ea691ddb45d5f16e9a8acc14f9d413f0a1f5/Dockerfile

[6] GitHub - comfyanonymous/ComfyUI: The most powerful and modular diffusion model GUI, api and backend with a graph/nodes interface.
https://github.com/comfyanonymous/ComfyUI

[8] Flux1 dev produces nothing but dots · Issue #4943 - GitHub
https://github.com/comfyanonymous/ComfyUI/issues/4943

[11] Docker Container with Ubuntu and ComfyUI -- Problem to access ...
https://forums.docker.com/t/docker-container-with-ubuntu-and-comfyui-problem-to-access-webpage-err-empty-response-solution/146947