ChatGPT

Progetto BadAI – Step 2: Creazione di un Container Docker GPU per ComfyUI

In questo step realizzeremo un container Docker Linux autosufficiente (da usare con Docker Desktop su Windows) che include **ComfyUI** installato da zero, con supporto GPU NVIDIA (CUDA) e con i modelli di AI interni (ad es. Stable Diffusion XL *SDXL* e *Flux 1*). Il container scaricherà automaticamente i modelli di base e gli asset necessari, e verificheremo infine il corretto funzionamento (accesso all'interfaccia web di ComfyUI e generazione di un'immagine di test). Di seguito la guida passo passo in italiano, con anche alcuni suggerimenti per scalare il sistema in futuro.

Prerequisiti e contesto

- **Docker Desktop con supporto GPU NVIDIA:** Assumiamo che dallo Step 1 Docker Desktop sia già installato su Windows con il backend WSL2 e che il supporto GPU sia attivato (ovvero i driver NVIDIA e il *NVIDIA Container Toolkit* sono configurati correttamente). Un test rapido come `docker run --gpus all nvidia/cuda:12.2.0-base nvidia-smi` dovrebbe già funzionare, mostrando le GPU disponibili. In caso contrario, assicurarsi di aver seguito le guide ufficiali per installare driver e toolkit NVIDIA [1].

- **Risorse Hardware:** ComfyUI utilizza l'accelerazione GPU per generare immagini tramite modelli di diffusione (Stable Diffusion e varianti). È quindi necessaria una GPU NVIDIA adeguata con driver aggiornati. Modelli avanzati come SDXL e Flux 1 richiedono GPU con molta VRAM (idealmente 12GB o più, e per *Flux* anche 24GB+ di VRAM o l'uso di ottimizzazioni FP8). Inoltre, questi modelli sono di grandi dimensioni su disco (nell'ordine di decine di GB), per cui assicurarsi di avere spazio sufficiente.

- **Account HuggingFace (opzionale):** Alcuni modelli come Stable Diffusion XL potrebbero richiedere di accettare una licenza su HuggingFace. Per automatizzare il download nel Dockerfile (con `wget`), potrebbe essere utile avere un accesso già configurato o un token di accesso se necessario. In alternativa, si daranno comandi `wget` diretti a file pubblici (in questo caso SDXL Base e alcuni modelli Flux FP8 sono disponibili pubblicamente, ma verificare eventuali restrizioni di licenza prima di procedere).

Nota: Daremo per completato lo Step 1 (installazione di Docker Desktop con WSL2 + test `nvidia-smi` in container). Passiamo quindi allo Step 2.

1. Configurazione dell'ambiente Docker con supporto GPU

Prima di costruire l'immagine Docker, verifichiamo la configurazione Docker per l'uso della GPU:

- **NVIDIA Container Toolkit:** Assicuriamoci che il runtime NVIDIA sia disponibile. Su Docker Desktop questo è integrato: basta abilitare *"Use GPU Support"* nelle impostazioni e utilizzare il flag `--gpus all` nei comandi `docker run` per abilitare l'accesso a tutte le GPU [2]. Su sistemi Linux standard, ciò richiede l'installazione manuale di *nvidia-docker2* o del *NVIDIA Container Toolkit* [1], ma su Windows+WSL2 è sufficiente la configurazione fatta in Step 1.

- **Verifica GPU nel container:** Esegui un rapido test, se non già fatto, per verificare che un container possa vedere la GPU. Ad esempio:

```
docker run --rm --gpus all nvidia/cuda:12.2.0-base nvidia-smi
```

Questo comando dovrebbe mostrare l'output di `nvidia-smi` con i dettagli della tua GPU. Se funziona, sei pronto per proseguire.

- **Pianificazione porte:** ComfyUI di default utilizza una porta web (8188 per la versione web/UI, o 8000 per la versione desktop) [3]. Useremo la porta predefinita 8188, mappandola poi su localhost per l'accesso dal browser. Assicurati che la porta 8188 non sia bloccata sul tuo sistema.

2. Creazione dell'immagine Docker con ComfyUI e modelli inclusi

Ora creiamo un Dockerfile per la nostra immagine. Questa immagine conterrà tutto il necessario: ambiente CUDA, ComfyUI, dipendenze Python, e i modelli di diffusione SDXL e Flux. Non useremo volumi esterni, quindi l'immagine finale sarà autosufficiente (anche se piuttosto grande, dato il peso dei modelli).

2.1 Scelta dell'immagine base (CUDA + Linux)

Partiamo da un'immagine base NVIDIA CUDA ufficiale che includa i driver CUDA e cuDNN appropriati per il deep learning. Ad esempio, useremo **Ubuntu 22.04 con CUDA 12 e cuDNN8** in versione runtime o devel. Questo garantisce compatibilità e alte prestazioni per il nostro ambiente [4]. Nel Dockerfile useremo:

```
FROM nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04
```

> *Motivazione:* come indicato in una guida, l'immagine base scelta (CUDA 12.1.1 + cuDNN8 su Ubuntu22.04) **assicura compatibilità e performance** per progetti di deep learning [4]. In generale, puoi usare una versione di CUDA supportata dalla tua GPU (es. CUDA 12.x per GPU Ampere/Ada/nuove, oppure CUDA 11.8 se hai compatibilità da RTX 3000 in giù). L'importante è che la versione di PyTorch che installeremo sia compatibile con la versione CUDA scelta.

2.2 Installazione delle dipendenze di sistema

Nel Dockerfile, aggiungiamo i comandi per installare le utility e librerie di sistema necessarie. Serviranno:

- **Python e pip:** Se l'immagine base non li include, li installiamo (python3, pip, ecc.).
- **Git e Git LFS:** Necessari per clonare il repository ComfyUI e, eventualmente, scaricare modelli di grandi dimensioni da HuggingFace (che usa Git LFS/Xet per file >10GB).
- **Compilatori e librerie base:** per assicurare che eventuali dipendenze possano essere costruite (ad esempio `build-essential`, librerie SSL, zlib, etc. – talvolta necessarie per pacchetti Python scientifici).
- **Librerie grafiche di base:** ComfyUI potrebbe richiedere alcune librerie grafiche/codec per gestire immagini e video (ad es. ffmpeg, libgl1, etc.). Inseriamo ffmpeg e Mesa GL se necessario.

Ad esempio, nel Dockerfile:

```
# Installiamo utilità di base, Python e librerie necessarie
RUN apt-get update && apt-get install -y \
    python3 python3-pip git git-lfs build-essential cmake \
    libssl-dev zlib1g-dev libjpeg-dev libffi-dev \
    ffmpeg libsm6 libxext6 libgl1-mesa-glx \
    && rm -rf /var/lib/apt/lists/* \
    && git lfs install
```

> *Spiegazione:* questo comando aggiorna i pacchetti e installa Python 3, pip, git (con git-lfs), compilatori e varie librerie. Rimuoviamo la cache di *apt* alla fine per mantenere l'immagine più leggera. L'esecuzione di `git lfs install` garantisce che Git LFS sia pronto per eventuali download di file di grandi dimensioni.

2.3 Clonazione di ComfyUI e installazione Python

Ora cloniamo il codice sorgente di **ComfyUI** e installiamo le dipendenze Python richieste. ComfyUI è disponibile su GitHub (repo `comfyanonymous/ComfyUI`). Possiamo clonarlo in una directory nel container, ad esempio `/opt/ComfyUI`. Subito dopo, installeremo le librerie Python necessarie: in particolare **PyTorch** (con supporto CUDA), e altre dipendenze specificate dal progetto.

Clonazione del repository:

```
# Impostiamo la working directory e cloniamo ComfyUI
WORKDIR /opt
RUN git clone https://github.com/comfyanonymous/ComfyUI.git
WORKDIR /opt/ComfyUI
```

Questo copierà tutto il codice di ComfyUI nel container. La struttura di ComfyUI include già una cartella `models/` (con sottocartelle come `checkpoints`, `vae`, `loras` etc.) dove andremo a collocare i file dei modelli [5].

> *Nota:* Il repository ComfyUI non include i file dei modelli (che vanno aggiunti manualmente). Tuttavia, la cartella `ComfyUI/models/` funge da posizione di default dove ComfyUI cercherà i modelli al lancio [5].

Installazione di PyTorch e altre dipendenze Python: ComfyUI ha un file `requirements.txt` con molte librerie Python (es. Flask, PIL, numpy, etc.), **ma potrebbe NON includere PyTorch**, lasciando all'utente l'installazione della versione corretta per la propria GPU. Per sicurezza installeremo manualmente PyTorch compatibile con CUDA 12 (o 11.x a seconda della base scelta) e qualche libreria opzionale come *xformers* per ottimizzare la memoria:

```
# Installiamo Pytorch (CUDA) e dipendenze Python di ComfyUI
RUN pip3 install --upgrade pip setuptools wheel && \
    pip3 install torch==2.0.1+cu121 torchvision==0.15.2+cu121 --extra-index-
url https://download.pytorch.org/whl/cu121 && \
```

```
    pip3 install xformers==0.0.20 && \
    pip3 install -r requirements.txt
```

Spiegazione rapida: - Aggiorniamo *pip* e strumenti base. - Installiamo PyTorch e Torchvision versione compatibile con CUDA 12.1 (`+cu121`), ad esempio PyTorch 2.0.1/2.1. (Se avessimo scelto CUDA 11.8, avremmo usato le build cu118, ecc.). Usiamo l'indice extra di PyTorch per scaricare le wheel precompilate [6] [7]. - Installiamo *xformers* (una libreria che accelera l'attenzione nei modelli di diffusione) – facoltativa ma utile per performance. - Infine installiamo tutte le altre dipendenze elencate nel `requirements.txt` di ComfyUI.

> **Nota:** In alternativa, invece di gestire manualmente PyTorch, si potrebbe usare un ambiente Python gestito (come in alcune guide che usano *pyenv* [8]), ma per semplicità usiamo pip direttamente. Assicuriamoci solo di scegliere la versione corretta di PyTorch per la nostra versione CUDA.

2.4 Download e inclusione dei modelli SDXL e Flux nel container

Adesso, l'aspetto chiave: **aggiungere i modelli di AI all'interno dell'immagine**. Come anticipato, inseriremo ad esempio il modello base di Stable Diffusion XL 1.0 (*SDXL Base*) e il modello *Flux 1*. Entrambi andranno copiati nella directory `ComfyUI/models/` appropriata nel container prima del runtime, così che ComfyUI li rilevi automaticamente all'avvio.

ComfyUI organizza i modelli in sottocartelle sotto `ComfyUI/models/` [5]: - *checkpoints/* – per modelli "checkpoint" all-in-one (tipicamente file `.safetensors` o `.ckpt` contenenti l'intera rete di diffusione + VAE + encoder). - *vae/* – per eventuali file di VAE esterni. - *text_encoders/*, *loras/*, *controlnet/*, ecc. – per altri tipi di modelli (testo, LoRA, reti di controllo).
SDXL e Flux, nel formato semplificato che useremo, saranno inseriti come **checkpoint singoli** nella cartella *checkpoints/*, così da poterli caricare con il nodo "Load Checkpoint" in ComfyUI [9].

Scaricamento dei modelli: Utilizzeremo comandi `wget` nel Dockerfile per scaricare i file dei modelli da HuggingFace (o altra fonte). Ad esempio:

- **SDXL Base 1.0:** disponibile su HuggingFace nel repository `stabilityai/stable-diffusion-xl-base-1.0`. Il file pesa circa 7GB (fp16). Usiamo la URL di resolve diretta:

  ```
  RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors -P ./models/checkpoints/
  ```

- **SDXL Refiner 1.0 (opzionale):** SDXL ha anche un modello *refiner* per migliorare i dettagli. Possiamo scaricarlo similmente:

  ```
  #RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-refiner-1.0/resolve/main/sd_xl_refiner_1.0.safetensors -P ./models/checkpoints/
  ```

  (Qui è commentato come opzionale: il refiner è utile ma non obbligatorio per generare immagini base.)

- **Flux 1 (Flux-*Dev*):** Il modello Flux di Black Forest Labs. Possiamo usare la versione "FP8 checkpoint" combinata, resa disponibile per ComfyUI [9]. Su HuggingFace, la comunità **Comfy-Org** ha rilasciato *flux1-dev-fp8.safetensors* (ca. 17GB). Lo scarichiamo:

  ```
  RUN wget -c https://huggingface.co/Comfy-Org/flux1-dev/resolve/main/
  flux1-dev-fp8.safetensors -P ./models/checkpoints/
  ```

  Nota: Questo file è molto grande (oltre 17 GB) [10] [11]. Assicurarsi di avere una connessione e spazio adeguati. In alternativa, esiste anche *flux1-schnell-fp8.safetensors* (una versione distillata più veloce, anch'essa ~17GB) [12]. Abbiamo scelto flux1-dev FP8 come esempio.
- **Stable Diffusion v1.5 (opzionale):** La guida ufficiale di ComfyUI carica di default un modello SD1.5 se presente [13]. Possiamo inserire anche *v1-5-pruned-emaonly.safetensors* (~4GB) per evitare avvisi di modello mancante. Ad esempio:

  ```
  #RUN wget -c https://huggingface.co/runwayml/stable-diffusion-v1-5/
  resolve/main/v1-5-pruned-emaonly.safetensors -P ./models/checkpoints/
  ```

  (Comando commentato se non necessario. Questo modello è utile se si vuole usare pipeline SD1.x classiche.)

Come si vede, stiamo **aggiungendo i modelli all'immagine Docker durante la build**, usando `wget` per scaricarli nella directory corretta [14]. Questo approccio è menzionato anche in guide: per aggiungere un modello custom al container, basta modificare il Dockerfile con un comando `RUN wget ... -P ./models/checkpoints/` che scarichi il file desiderato [14].

> *Nota licenze:* SDXL e alcuni modelli possono richiedere di aver accettato le licenze su HuggingFace. Se `wget` fallisce per permessi, potrebbe essere necessario ottenere un URL con token di autenticazione oppure effettuare login con `huggingface-cli` prima del download. Per semplicità, supponiamo che i link funzionino (SDXL Base è accessibile dopo accettazione della licenza una volta, mentre Flux FP8 è rilasciato con licenza non commerciale ma scaricabile liberamente).

Possiamo anche includere un modello VAE ottimizzato per SD (ad esempio *vae-ft-mse-840000-ema-pruned.safetensors* da StabilityAI, ~335MB) per migliorare la qualità delle immagini SD1.x e SD2.x. Nel Dockerfile di esempio si può aggiungere nella cartella `models/vae/`:

```
RUN wget -c https://huggingface.co/stabilityai/sd-vae-ft-mse-original/
resolve/main/vae-ft-mse-840000-ema-pruned.safetensors -P ./models/vae/
```

(Inserito come esempio, utile se si useranno modelli SD1.5 che beneficiano di un VAE migliore.)

2.5 Impostare il comando di avvio (entrypoint) del container

Infine, configuriamo il container in modo che all'avvio lanci automaticamente ComfyUI. Nel repository clonato, il file principale è `main.py`. Per avviare il server, normalmente eseguiremmo `python main.py` (con eventuali parametri) [15]. Nel contesto di Docker, aggiungeremo una direttiva **CMD** nel Dockerfile:

```
# Esporre la porta e definire il comando di esecuzione
EXPOSE 8188
CMD ["python3", "main.py", "--listen", "0.0.0.0", "--port", "8188"]
```

Spiegazione: - `EXPOSE 8188` documenta la porta su cui il container ascolta (utile ma informativo; mapperemo comunque la porta manualmente al run). - Il comando `python3 main.py --listen 0.0.0.0 --port 8188` avvia ComfyUI in modalità server, in ascolto su tutte le interfacce (`0.0.0.0`) sulla porta 8188 [16] [3]. Questo è importante perché per default ComfyUI tende ad ascoltare solo su localhost (127.0.0.1) nel container, il che impedirebbe l'accesso dall'esterno. Impostando `--listen 0.0.0.0` permettiamo l'accesso via browser da host. Abbiamo fissato la porta a 8188 (potevamo lasciarla di default, ma esplicitarla è buona pratica; se servisse cambiarla, possiamo modificare qui e nel mapping).

Il Dockerfile completo, sommando i pezzi visti, risulterà simile al seguente:

```
# Dockerfile per ComfyUI con supporto GPU e modelli interni (SDXL, Flux)
FROM nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04

# Installazione pacchetti di sistema e Python
RUN apt-get update && apt-get install -y \
    python3 python3-pip git git-lfs build-essential cmake \
    libssl-dev zlib1g-dev libjpeg-dev libffi-dev \
    ffmpeg libsm6 libxext6 libgl1-mesa-glx && \
    rm -rf /var/lib/apt/lists/* && \
    git lfs install

# Clonazione ComfyUI
WORKDIR /opt
RUN git clone https://github.com/comfyanonymous/ComfyUI.git
WORKDIR /opt/ComfyUI

# Installazione Pytorch (CUDA) e dipendenze Python
RUN pip3 install --upgrade pip setuptools wheel && \
    pip3 install torch==2.0.1+cu121 torchvision==0.15.2+cu121 --extra-index-
url https://download.pytorch.org/whl/cu121 && \
    pip3 install xformers==0.0.20 && \
    pip3 install -r requirements.txt

# Download modelli AI dentro il container
RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/
resolve/main/sd_xl_base_1.0.safetensors -P ./models/checkpoints/
#RUN wget -c https://huggingface.co/stabilityai/stable-diffusion-xl-
refiner-1.0/resolve/main/sd_xl_refiner_1.0.safetensors -P ./models/
checkpoints/
RUN wget -c https://huggingface.co/Comfy-Org/flux1-dev/resolve/main/flux1-
dev-fp8.safetensors -P ./models/checkpoints/
#RUN wget -c https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/
main/v1-5-pruned-emaonly.safetensors -P ./models/checkpoints/
RUN wget -c https://huggingface.co/stabilityai/sd-vae-ft-mse-original/
```

```
resolve/main/vae-ft-mse-840000-ema-pruned.safetensors -P ./models/vae/

# Esporre porta e definire comando di default
EXPOSE 8188
CMD ["python3", "main.py", "--listen", "0.0.0.0", "--port", "8188"]
```

> **Nota:** I comandi `wget` relativi a modelli opzionali sono commentati (`#`) per indicare che sono facoltativi. Si possono attivare rimuovendo `#` se si vogliono includere anche quei file. Sentiti libero di aggiungere altri modelli modificando analogamente il Dockerfile (ad esempio modelli **ControlNet**, modelli aggiuntivi SD 2.1, modelli LoRA, ecc., scaricandoli nelle rispettive sottocartelle). La struttura a cartelle sotto `/ComfyUI/models/` da seguire è riassunta nella documentazione ComfyUI [5].

A questo punto, abbiamo definito tutto il necessario per costruire l'immagine Docker.

3. Build dell'immagine Docker

Con il Dockerfile pronto (salvato in una directory, ad es. in `C:\progetti\BadAI\Step2\Dockerfile`), eseguiamo il comando di **build**. Apri un terminale nella cartella che contiene il Dockerfile e digita:

```
docker build -t badai/comfyui:step2 .
```

Spiegazione: - `-t badai/comfyui:step2` assegna un nome e tag all'immagine (puoi sceglierne uno a piacere, qui ad esempio *badai/comfyui:step2*). - Il `.` finale indica di usare il Dockerfile nella directory corrente.

Il processo di build richiederà un po' di tempo, soprattutto durante il download dei modelli (potrebbero volerci decine di minuti per decine di GB di modelli). Al termine dovresti avere l'immagine Docker pronta, verificabile con `docker images` (dovrebbe comparire con il nome e tag scelto).

Nota: La dimensione finale dell'immagine sarà piuttosto grande (potenzialmente 20+ GB) a causa dei modelli inclusi. Docker potrebbe mostrare warning sulla dimensione; assicurati di avere spazio su disco sufficiente.

4. Esecuzione del container e verifica di ComfyUI

Adesso eseguiremo un container dall'immagine appena creata e verificheremo che ComfyUI funzioni correttamente, con la GPU attiva e i modelli disponibili.

4.1 Avviare il container con supporto GPU e porta esposta

Usiamo `docker run` con i parametri adeguati:

```
docker run -d --rm --gpus all -p 8188:8188 --name comfyui_gpu badai/
comfyui:step2
```

Vediamo il significato delle opzioni principali: - `-d` avvia il container in **modalità detached** (in background). In alternativa, puoi ometterlo per vedere i log in foreground. - `--rm` fa sì che il container venga rimosso automaticamente al termine (così non rimane stoppato occupando spazio). - `--gpus all` abilita l'uso di tutte le GPU NVIDIA nel container (equivale al vecchio `--runtime=nvidia`). Questa è la chiave per permettere a ComfyUI di usare CUDA [2]. - `-p 8188:8188` mappa la porta 8188 del container sulla porta 8188 dell'host (il PC Windows). Così, collegandosi a `localhost:8188` nel browser accederemo all'UI di ComfyUI che gira nel container. - `--name comfyui_gpu` assegna un nome al container (facoltativo, per riferimento). - Infine l'identificativo dell'immagine da eseguire (nel nostro caso *badai/comfyui:step2* creato prima).

Se tutto va bene, il container partirà in pochi secondi. Puoi controllare i log con `docker logs -f comfyui_gpu` (specie se non hai usato `-d`). Dovresti vedere messaggi di inizializzazione di ComfyUI, qualcosa tipo *"Running on http://127.0.0.1:8188"* ecc. senza errori.

Possibili verifiche iniziali da log: - ComfyUI dovrebbe rilevare la GPU e stampare il nome della GPU e la memoria disponibile. - Dovrebbe anche elencare i modelli trovati nelle cartelle. Ad esempio, potresti vedere nei log riferimenti a `sd_xl_base_1.0.safetensors` e `flux1-dev-fp8.safetensors` se tutto è corretto (a volte al primo avvio ComfyUI può indicizzare i modelli e potrebbe non stamparli tutti, ma in generale nessun errore "no checkpoints found" dovrebbe apparire visto che li abbiamo inseriti in *models/checkpoints* [17]).

4.2 Accesso all'interfaccia web di ComfyUI

Apri un browser web sul tuo host (Windows) e naviga all'indirizzo: `http://localhost:8188`. Dovresti vedere l'interfaccia grafica di ComfyUI caricata (un layout a nodi su sfondo scuro, con un menu a sinistra e un canvas a destra).

Se l'interfaccia non si carica: - Controlla i log del container (`docker logs`) per eventuali errori. - Assicurati che la porta 8188 sia correttamente esposta e non bloccata dal firewall di Windows. - Verifica di aver usato `--listen 0.0.0.0` nel CMD del Dockerfile (come fatto sopra), altrimenti il servizio potrebbe essere in ascolto solo dentro al container.

4.3 Caricare un workflow e selezionare un modello

Per generare un'immagine di prova, segui questi passi nell'UI ComfyUI (ricorda: ComfyUI funziona a nodi, simile a un flowchart): 1. **Workflow di default:** All'avvio, ComfyUI spesso carica automaticamente un workflow di esempio *"Image Generation"*. In caso non lo facesse, clicca sull'icona *folder* nella barra laterale sinistra, quindi su *"Browse example workflows"* e scegli un template di generazione base. Questo posizionerà dei nodi sul canvas (in particolare un nodo **Load Checkpoint**, alcuni nodi di elaborazione e un nodo **Save Image** in fondo). 2. **Seleziona il modello SDXL:** Il nodo **Load Checkpoint** (solitamente il primo nodo in alto a sinistra) serve a scegliere il modello di diffusione da usare. Cliccando sul nodo, nella sua parte destra comparirà un menu a tendina con i file modello disponibili. Dovresti vedere elencato *sd_xl_base_1.0.safetensors* (e anche *flux1-dev-fp8.safetensors* eventualmente, più eventuali altri modelli inseriti). Seleziona **sd_xl_base_1.0.safetensors** su questo nodo. (Volendo provare Flux, selezioneresti flux1-dev, ma attenzione: Flux richiede anche settaggi particolari, come *CFG=1.0* nel nodo di diffusione, ed è molto pesante; inizialmente testiamo SDXL.) 3. **Imposta un prompt semplice:** Nel nodo di **Text Prompt** (di solito collegato all'ingresso del modello), inserisci un testo descrittivo, ad esempio *"una bellissima illustrazione digitale di un paesaggio al tramonto"*. Nel nodo **Negative Prompt** puoi lasciare vuoto o inserire qualche termine da evitare (opzionale). 4. **Avvia la generazione:** Clicca sul pulsante **"Queue"** in alto (oppure premi Ctrl+Invio) per eseguire il workflow. ComfyUI caricherà il

modello selezionato e inizierà la generazione dell'immagine. Potrai monitorare lo stato nella UI (in basso appare una barra di progresso e il log dei passi). 5. **Verifica del risultato:** Al termine, il nodo **Save Image** avrà l'output. Cliccando sull'anteprima dell'immagine generata, dovresti poterla vedere in maggiore dettaglio e salvarla se vuoi.

Se hai seguito i passi, dovresti ottenere la tua prima immagine generata con ComfyUI nel container. Questo conferma che: - Il container vede correttamente la GPU e la usa per il modello (la generazione di un'immagine 1024x1024 con SDXL dovrebbe impiegare pochi secondi su GPU potenti, e qualche decina di secondi su GPU meno recenti). - I modelli sono stati installati correttamente (infatti li hai potuti selezionare dal menu e usare). In caso contrario, ComfyUI avrebbe segnalato *"No checkpoints found"* o ti avrebbe richiesto di installare un modello [17]. - L'interfaccia web funziona e possiamo interagire dal nostro host.

> **Nota:** Il workflow di default di ComfyUI originariamente è tarato su SD1.5; se *non* hai inserito un modello SD1.5, inizialmente potrebbe averti mostrato un avviso di modello mancante con opzione di download [13]. Se hai ignorato l'avviso e proseguito selezionando manualmente SDXL come detto sopra, va bene. In alternativa, includendo il modello v1-5 (come da Dockerfile commentato) quel prompt non apparirà. In ogni caso, una volta selezionato un modello disponibile, l'avviso scompare.

5. Suggerimenti sulla scalabilità futura

Fin qui abbiamo ottenuto un container "monolitico" funzionante, ma ci sono alcuni aspetti da tenere a mente per futuri sviluppi o per scenari più complessi:

- **Dimensione dell'immagine & Modelli esterni:** Inserire i modelli direttamente nell'immagine Docker, se da un lato rende il container autosufficiente, dall'altro lo rende molto pesante e meno flessibile. In futuro potresti valutare di **non includere i modelli nel build**, ma di montarli come volume esterno. Ad esempio, potresti mantenere i file dei modelli in una cartella host (es. `C:\model-cache\`) e poi eseguire il container montandola su `/opt/ComfyUI/models/` nel container. In questo modo l'immagine Docker rimane più piccola e i modelli possono essere aggiornati senza ricostruire l'immagine. Inoltre, i modelli (e output generati) persistono anche se ricrei o aggiorni il container [18]. La guida Krasamo suggerisce proprio un'opzione *USE_PERSISTENT_DATA* per tenere output e modelli fuori dal container [19]. Nel nostro caso, basterebbe usare `-v` in `docker run` se volessimo montare modelli esternamente.

- **Aggiornamenti dei modelli:** Se volessi aggiungere nuovi modelli (es. altri checkpoint Stable Diffusion, modelli LoRA, ControlNet, ecc.), nel nostro attuale setup dovresti modificare il Dockerfile e fare rebuild (o entrare nel container e scaricarli manualmente). Un approccio più scalabile è appunto usare volumi o uno storage condiviso. Potresti anche impostare ComfyUI in modo che legga modelli da percorsi aggiuntivi configurati via file `extra_model_paths.yaml` (nel repo c'è un esempio di questo file) per puntare a directory esterne.

- **Orchestrazione (Kubernetes o Docker Compose):** Se il progetto BadAI dovesse evolvere verso un ambiente distribuito o un servizio, considera l'uso di Docker Compose o Kubernetes:

- Con **Docker Compose**, potresti definire un servizio per ComfyUI e potenzialmente altri servizi (es. un database o un frontend separato). Puoi inserire parametri come la porta e volumi nel file `docker-compose.yml`. Per esempio, la guida citata usava *docker-compose up* per esporre l'interfaccia [20].

- Con **Kubernetes**, potresti deployare il container in un cluster (ad es. per offrire generazione di immagini via API internamente). In tal caso, useresti un *Deployment* con risorsa GPU (necessario configurare il device plugin NVIDIA su K8s) e un *Service* per esporre la porta 8188. Anche qui, i modelli sarebbe meglio metterli su un volume PersistenVolumeClaim così che i pod possano condividerli e non replicarli inutilmente.

- Se prevedi di scalare a più istanze (ad esempio per servire più richieste contemporaneamente), avere i modelli su un volume condiviso evita di avere 3 copie da 20GB ciascuna su 3 container; invece tutte le istanze montano lo stesso volume in sola lettura per i modelli.

- **Sicurezza e accesso:** La nostra configurazione esposta su `0.0.0.0:8188` è pensata per ambiente locale/trusted. Se dovessi mettere ComfyUI online o in rete aziendale, considera di aggiungere autenticazione o restrizioni. Esistono progetti come *AI-Dock* che integrano autenticazione a ComfyUI [21]. In Docker/K8s potresti anche mettere un proxy con password. ComfyUI stesso al momento non ha autenticazione integrata, essendo pensato per uso locale.

- **Persistenza degli output:** Al momento, le immagini generate rimangono nel container (in `ComfyUI/output/` credo). Con un container effimero, rischi di perderle quando il container è rimosso. Valuta di montare anche la cartella output come volume esterno, o configurare ComfyUI perché salvi in una cartella condivisa (la guida Krasamo ad esempio monta `/data` per salvare output e file vari [22]).

- **Ottimizzazioni performance:** In futuro, potresti voler abilitare flag di performance. Ad esempio, avviare ComfyUI con `--use-pytorch-cross-attention` (per usare l'implementazione ottimizzata di PyTorch per l'attention, se supportata) o `--lowvram` / `--half` in caso di poca VRAM. Queste opzioni possono essere passate tramite variabile d'ambiente `CLI_ARGS` come visto in alcune immagini [2], oppure aggiunte al comando in CMD. Nel nostro caso, avendo messo esplicitamente `python3 main.py --listen ... --port ...`, potresti aggiungere altri parametri lì. Ad esempio:

  ```
  CMD ["python3", "main.py", "--listen", "0.0.0.0", "--port", "8188", "--use-pytorch-cross-attention"]
  ```

- **Modularità:** Man mano che aggiungi più componenti (es. nuovi modelli, plugin di ComfyUI, ecc.), tieni traccia di queste modifiche. Documenta magari in un README interno quali modelli versione hai incluso (specialmente per flux, che potrebbe aggiornarsi). Questo aiuta a rigenerare l'ambiente o crearne una versione leggera in futuro.

In sintesi, abbiamo costruito un container che **funziona out-of-the-box**: basta eseguirlo e accedere all'UI per generare immagini. Per futuri utilizzi in produzione o per facilitare aggiornamenti, conviene però valutare approcci più dinamici (volumi per modelli e dati, orchestrazione se serve alta affidabilità, ecc.).

Continua così con il progetto BadAI! Ora che Step 2 è concluso con successo, hai a disposizione un ambiente containerizzato potente per sperimentare con l'AI generativa. Buon divertimento con ComfyUI e la creazione di immagini!

Riferimenti:

- ComfyUI – Documentazione ufficiale e repository GitHub [23] [15]
- Guida Krasamo: *"Setting Up a Dockerized AI Environment with ComfyUI and NVIDIA CUDA"* – esempi di Dockerfile e best practice [24] [25]
- Esempio Dockerfile (HuggingFace Spaces) con download modelli SDXL [26]
- YanWenKun ComfyUI Docker – esempi di run con GPU [2]
- Documentazione ComfyUI – gestione modelli e percorsi [5] [27] (modelli in *ComfyUI/models/*)
- Esempi ComfyUI per Flux e SDXL – come aggiungere i file dei modelli e caricarli [9].
- Suggerimenti su persistenza modelli e dati nei container [18] [14].

---

[1] [4] [14] [18] [19] [20] [22] [24] [25] [27] Setting Up a Dockerized AI Environment with ComfyUI and NVIDIA CUDA | Krasamo
https://www.krasamo.com/comfyui-docker/

[2] GitHub - YanWenKun/ComfyUI-Docker: Dockerfile for ComfyUI. | 容器镜像与启动脚本
https://github.com/YanWenKun/ComfyUI-Docker

[3] [16] Server Config - ComfyUI
https://docs.comfy.org/interface/settings/server-config

[5] [13] [23] Getting Started with AI Image Generation - ComfyUI
https://docs.comfy.org/get_started/first_generation

[6] [7] [8] [26] Dockerfile · SpacesExamples/ComfyUI at main
https://huggingface.co/spaces/SpacesExamples/ComfyUI/blob/main/Dockerfile

[9] Flux Examples | ComfyUI_examples
https://comfyanonymous.github.io/ComfyUI_examples/flux/

[10] [11] flux1-dev-fp8.safetensors · Comfy-Org/flux1-dev at main
https://huggingface.co/Comfy-Org/flux1-dev/blob/main/flux1-dev-fp8.safetensors

[12] flux1-schnell-fp8.safetensors · Comfy-Org/flux1-schnell at main
https://huggingface.co/Comfy-Org/flux1-schnell/blob/main/flux1-schnell-fp8.safetensors

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

[17] Can't Choose/Select a Checkpoint/Model or Lora in ComfyUI - Reddit
https://www.reddit.com/r/StableDiffusion/comments/15k8u7i/cant_chooseselect_a_checkpointmodel_or_lora_in/

[21] AI-Dock + ComfyUI Docker Image - GitHub
https://github.com/ai-dock/comfyui