(deployment-docker)=

Using Docker

(deployment-docker-pre-built-image)=

Use vLLM's Official Docker Image

vLLM offers an official Docker image for deployment.

The image can be used to run OpenAI compatible server and is available on Docker Hub as [vllm/vllm-openai](https://hub.docker.com/r/vllm/vllm-openai/tags).

```console
$ docker run --runtime nvidia --gpus all \
    -v ~/.cache/huggingface:/root/.cache/huggingface \
    --env "HUGGING_FACE_HUB_TOKEN=<secret>" \
    -p 8000:8000 \
    --ipc=host \
    vllm/vllm-openai:latest \
    --model mistralai/Mistral-7B-v0.1
```

You can add any other <project:#engine-args> you need after the image tag (`vllm/vllm-openai:latest`).

:::{note}

You can either use the `ipc=host` flag or `--shm-size` flag to allow the

container to access the host's shared memory. vLLM uses PyTorch, which uses shared memory to share data between processes under the hood, particularly for tensor parallel inference.
:::

(deployment-docker-build-image-from-source)=

Building vLLM's Docker Image from Source

You can build and run vLLM from source via the provided <gh-file:Dockerfile>. To build vLLM:

```console
# optionally specifies: --build-arg max_jobs=8 --build-arg nvcc_threads=2
DOCKER_BUILDKIT=1 docker build . --target vllm-openai --tag vllm/vllm-openai
```

:::{note}
By default vLLM will build for all GPU types for widest distribution. If you are just building for the current GPU type the machine is running on, you can add the argument `--build-arg torch_cuda_arch_list=""`
for vLLM to find the current GPU type and build for that.

If you are using Podman instead of Docker, you might need to disable SELinux labeling by adding `--security-opt label=disable` when running `podman build` command to avoid certain [existing issues](https://github.com/containers/buildah/discussions/4184).
:::

Building for Arm64/aarch64

A docker container can be built for aarch64 systems such as the Nvidia Grace-Hopper. At time of this writing, this requires the use

of PyTorch Nightly and should be considered **experimental**. Using the flag `--platform "linux/arm64"` will attempt to build for arm64.

:::{note}

Multiple modules must be compiled, so this process can take a while. Recommend using `--build-arg max_jobs=` & `--build-arg nvcc_threads=`

flags to speed up build process. However, ensure your `max_jobs` is substantially larger than `nvcc_threads` to get the most benefits.

Keep an eye on memory usage with parallel jobs as it can be substantial (see example below).

:::

```console
# Example of building on Nvidia GH200 server. (Memory usage: ~15GB, Build time: ~1475s / ~25 min, Image size: 6.93GB)
$ python3 use_existing_torch.py
$ DOCKER_BUILDKIT=1 docker build . \
  --target vllm-openai \
  --platform "linux/arm64" \
  -t vllm/vllm-gh200-openai:latest \
  --build-arg max_jobs=66 \
  --build-arg nvcc_threads=2 \
  --build-arg torch_cuda_arch_list="9.0+PTX" \
  --build-arg vllm_fa_cmake_gpu_arches="90-real"
```

Use the custom-built vLLM Docker image

To run vLLM with the custom-built Docker image:

```console
$ docker run --runtime nvidia --gpus all \
    -v ~/.cache/huggingface:/root/.cache/huggingface \
    -p 8000:8000 \
    --env "HUGGING_FACE_HUB_TOKEN=<secret>" \
    vllm/vllm-openai <args...>
```

The argument `vllm/vllm-openai` specifies the image to run, and should be replaced with the name of the custom-built image (the `-t` tag from the build command).

:::{note}
For version 0.4.1 and 0.4.2 only - the vLLM docker images under these versions are supposed to be run under the root user since a library under the root user's home directory, i.e. `/root/.config/vllm/nccl/cu12/libnccl.so.2.18.1` is required to be loaded during runtime. If you are running the container under a different user, you may need to first change the permissions of the library (and all the parent directories) to allow the user to access it, then run vLLM with environment variable `VLLM_NCCL_SO_PATH=/root/.config/vllm/nccl/cu12/libnccl.so.2.18.1` .
:::