Effortlessly Build Machine Learning Apps with Hugging Face’s Docker Spaces

This article was fetched from an rss feed(23/03/2023)

The Hugging Face Hub is a platform that enables collaborative open source machine learning (ML). The hub works as a central place where users can explore, experiment, collaborate, and build technology with machine learning. On the hub, you can find more than 140,000 models, 50,000 ML apps (called Spaces), and 20,000 datasets shared by the community.

Using Spaces makes it easy to create and deploy ML-powered applications and demos in minutes. Recently, the Hugging Face team added support for Docker Spaces, enabling users to create any custom app they want by simply writing a Dockerfile.

Another great thing about Spaces is that once you have your app running, you can easily share it with anyone around the world.

This guide will step through the basics of creating a Docker Space, configuring it, and deploying code to it. We’ll show how to build a basic FastAPI app for text generation that will be used to demo the **google/flan-t5-small** model, which can generate text given input text. Models like this are used to power text completion in all sorts of apps. (You can check out a completed version of the app at Hugging Face.)

Prerequisites

To follow along with the steps presented in this article, you’ll need to be signed in to the Hugging Face Hub — you can sign up for free if you don’t have an account already.

Create a new Docker Space

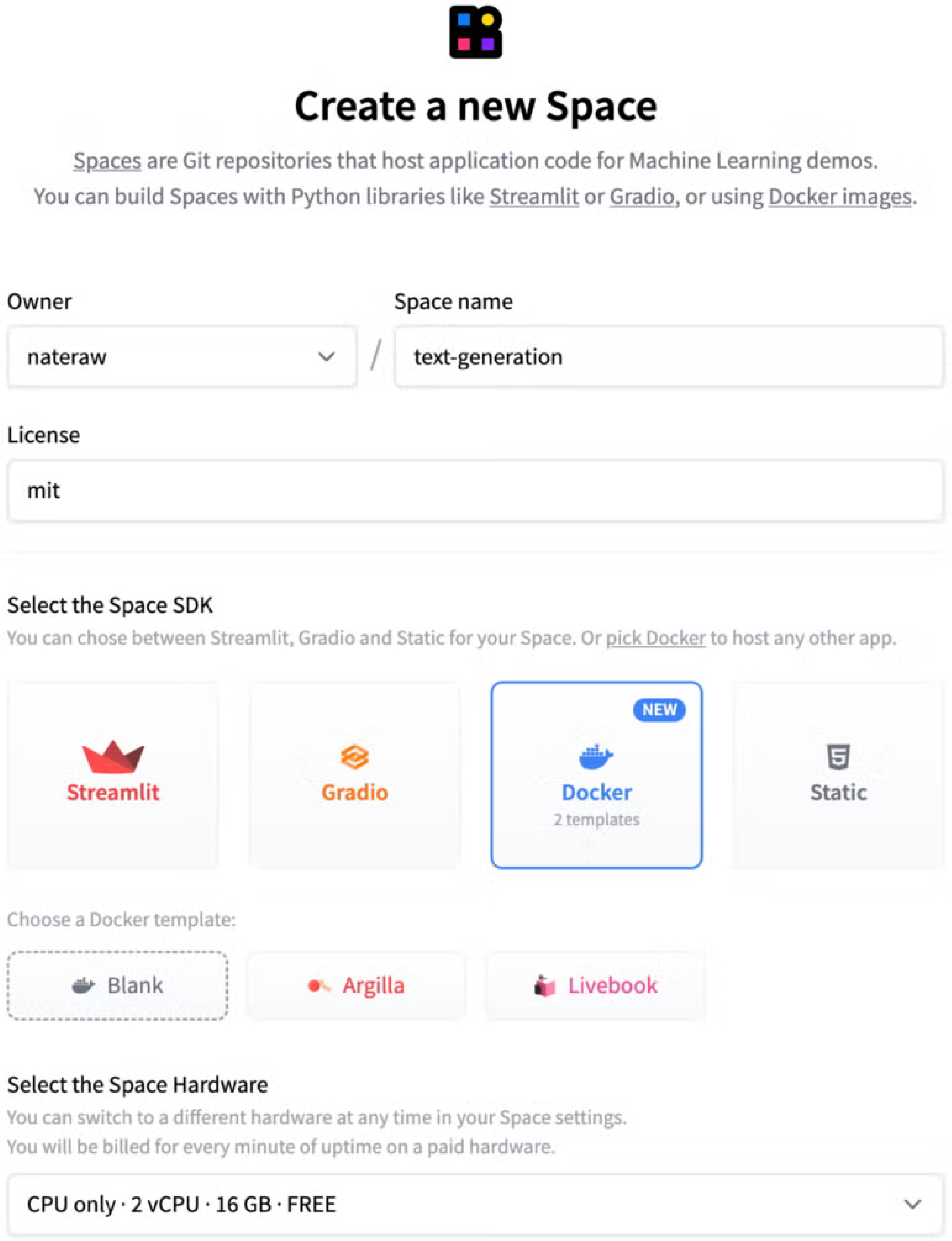
To get started, create a new Space as shown in Figure 1.

![Screenshot of Hugging Face Spaces, showing "Create new Space" button in upper right.](https://www.docker.com/wp-content/uploads/2023/03/create-new-hugging-face-space-1110x612.png "Screenshot of Hugging Face Spaces, showing "Create new Space" button in upper right. - create new hugging face space")

Figure 1: Create a new Space.

Next, you can choose any name you prefer for your project, select a license, and use Docker as the software development kit (SDK) as shown in Figure 2.

Spaces provides pre-built Docker templates like Argilla and Livebook that let you quickly start your ML projects using open source tools. If you choose the “Blank” option, that means you want to create your Dockerfile manually. Don’t worry, though; we’ll provide a Dockerfile to copy and paste later. 😅



Create a new Space

Spaces are Git repositories that host application code for Machine Learning demos.
You can build Spaces with Python libraries like Streamlit or Gradio, or using Docker images.

Owner: nateraw / Space name: text-generation

License: mit

Select the Space SDK

You can chose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Streamlit | Gradio | Docker (NEW, 2 templates) | Static

Choose a Docker template:

Blank | Argilla | Livebook

Select the Space Hardware

You can switch to a different hardware at any time in your Space settings.
You will be billed for every minute of uptime on a paid hardware.

CPU only · 2 vCPU · 16 GB · FREE

Figure 2: Adding details for the new Space.

When you finish filling out the form and click on the **Create Space** button, a new repository will be created in your Spaces account. This repository will be associated with the new space that you have created.

Note: If you're new to the Hugging Face Hub , check out Getting Started with Repositories for a nice primer on repositories on the hub.

Writing the app

Ok, now that you have an empty space repository, it's time to write some code. 😎

The sample app will consist of the following three files:

- `requirements.txt` — Lists the dependencies of a Python project or application
- `app.py` — A Python script where we will write our FastAPI app
- `Dockerfile` — Sets up our environment, installs `requirements.txt`, then launches `app.py`

To follow along, create each file shown below via the web interface. To do that, navigate to your Space's **Files and versions** tab, then choose **Add file** → **Create a new file** (Figure 3). Note that, if you prefer, you can also utilize Git.

![Screenshot showing selection of "Create a new file" under "Add file" dropdown menu.](https://www.docker.com/wp-content/uploads/2023/03/hugging-face-space-files-versions-1110x400.png "Screenshot showing selection of "Create a new file" under "Add file" dropdown menu. - hugging face space files versions")

Figure 3: Creating new files.

Make sure that you name each file exactly as we have done here. Then, copy the contents of each file from here and paste them into the corresponding file in the editor. After you have created and populated all the necessary files, commit each new file to your repository by clicking on the **Commit new file to main** button.

Listing the Python dependencies

It's time to list all the Python packages and their specific versions that are required for the project to function properly. The contents of the `requirements.txt` file typically include the name of the package and its version number, which can be specified in a variety of formats such as exact version numbers, version ranges, or compatible versions. The file lists `FastAPI`, `requests`, and `uvicorn` for the API along with `sentencepiece`, `torch`, and `transformers` for the text-generation model.

fastapi==0.74.* requests==2.27.* uvicorn[standard]==0.17.* sentencepiece==0.1.* torch==1.11.* transformers==4.*

Defining the FastAPI web application

The following code defines a FastAPI web application that uses the transformers library to generate text based on user input. The app itself is a simple single-endpoint API. The `/generate` endpoint takes in text and uses a transformers pipeline to generate a completion, which it then returns as a response.

To give folks something to see, we reroute FastAPI's interactive Swagger docs from the default `/docs` endpoint to the root of the app. This way, when someone visits your Space, they can play with it without having to write any code.

from fastapi import FastAPI from transformers import pipeline

Create a new FastAPI app instance

app = FastAPI()

Initialize the text generation pipeline

This function will be able to generate text

given an input.

pipe = pipeline("text2text-generation", model="google/flan-t5-small")

Define a function to handle the GET request at `/generate`

The generate() function is defined as a FastAPI route that takes a

string parameter called text. The function generates text based on the # input using the pipeline() object, and returns a JSON response

containing the generated text under the key "output"

@app.get("/generate") def generate(text: str): """ Using the text2text-generation pipeline from `transformers`, generate text from the given input text. The model used is `google/flan-t5-small`, which can be found [here](https://huggingface.co/google/flan-t5-small). """ # Use the pipeline to generate text from the given input text output = pipe(text)

```
# Return the generated text in a JSON response
return {"output": output\[0\]\["generated\_text"\]}
```

Writing the Dockerfile

In this section, we will write a Dockerfile that sets up a Python 3.9 environment, installs the packages listed in `requirements.txt`, and starts a FastAPI app on port 7860.

Let's go through this process step by step:

FROM python:3.9

The preceding line specifies that we're going to use the official Python 3.9 Docker image as the base image for our container. This image is provided by Docker Hub, and it contains all the necessary files to run Python 3.9.

WORKDIR /code

This line sets the working directory inside the container to `/code`. This is where we'll copy our application code and dependencies later on.

COPY ./requirements.txt /code/requirements.txt

The preceding line copies the `requirements.txt` file from our local directory to the `/code` directory inside the container. This file lists the Python packages that our application depends on

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

This line uses `pip` to install the packages listed in `requirements.txt`. The `--no-cache-dir` flag tells pip to not use any cached packages, the `--upgrade` flag tells `pip` to upgrade any already-installed packages if newer versions are available, and the `-r` flag specifies the requirements file to use.

RUN useradd -m -u 1000 user USER user ENV HOME=/home/user \\ PATH=/home/user/.local/bin:$PATH

These lines create a new `user` named user with a user ID of 1000, switch to that user, and then set the home directory to `/home/user`. The `ENV` command sets the `HOME` and `PATH` environment variables. `PATH` is modified to include the `.local/bin` directory in the user's home directory so that any binaries installed by pip will be available on the command line. Refer the documentation to learn more about the user permission.

WORKDIR $HOME/app

This line sets the working directory inside the container to `$HOME/app`, which is `/home/user/app`.

COPY --chown=user . $HOME/app

The preceding line copies the contents of our local directory into the `/home/user/app` directory inside the container, setting the owner of the files to the user that we created earlier.

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "7860"]

This line specifies the command to run when the container starts. It starts the FastAPI app using `uvicorn` and listens on port 7860. The `--host` flag specifies that the app should listen on all available network interfaces, and the app:app argument tells `uvicorn` to look for the app object in the app module in our code.

Here’s the complete Dockerfile:

# Use the official Python 3.9 image FROM python:3.9

Set the working directory to /code

WORKDIR /code

Copy the current directory contents into the container at /code

COPY ./requirements.txt /code/requirements.txt

Install requirements.txt

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

Set up a new user named "user" with user ID 1000

RUN useradd -m -u 1000 user

Switch to the "user" user

USER user

Set home to the user's home directory

ENV HOME=/home/user \\ PATH=/home/user/.local/bin:$PATH

Set the working directory to the user's home directory

WORKDIR $HOME/app

Copy the current directory contents into the container at $HOME/app setting the owner to the user

COPY --chown=user . $HOME/app

Start the FastAPI app on port 7860, the default port expected by Spaces

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "7860"]

Once you commit this file, your space will switch to **Building**, and you should see the container’s build logs pop up so you can monitor its status.

If you want to double-check the files, you can find all the files at our app Space.

Note: For a more basic introduction on using Docker with FastAPI, you can refer to the official guide from the FastAPI docs.

Using the app

If all goes well, your space should switch to **Running** once it’s done building, and the Swagger docs generated by FastAPI should appear in the **App** tab. Because these docs are interactive, you can try out the endpoint by expanding the details of the `/generate` endpoint and clicking **Try it out!** (Figure 4).

![Screenshot of FastAPI showing "Try it out!" option on the right-hand side.](https://www.docker.com/wp-content/uploads/2023/03/fastapi-swagger-docs-1110x646.png "Screenshot of FastAPI showing "Try it out!" option on the right-hand side. - fastapi swagger docs")

Figure 4: Trying out the app.

Conclusion

This article covered the basics of creating a Docker Space, building and configuring a basic FastAPI app for text generation that uses the google/flan-t5-small model. You can use this guide as a starting point to build more complex and exciting applications that leverage the power of machine learning.

If you’re interested in learning more about Docker templates and seeing curated examples, check out the Docker Examples page. There you’ll find a variety of templates to use as a starting point for your own projects, as well as tips and tricks for getting the most out of Docker templates. Happy coding!