Docker for Data Science Projects: A Beginner-Friendly Introduction

Elevate Your Data Science Workflow: Harness Docker's Power for Seamless Project Management

YOUSSEF HOSNI
MAR 03, 2025 · PAID

31 3 Share

When shipping your machine learning code to the engineering team, encounteri compatibility issues with different operating systems and library versions can be frustrating.

Docker can solve compatibility issues between operating systems and library ver when shipping machine learning code to engineering teams, making code execu seamless regardless of its underlying setup.

In this comprehensive tutorial, we will introduce Docker's essential concepts, gu you through installation, demonstrate its practical use with examples, uncover industry best practices, and answer any related queries along the way—so say go to compatibility woes and streamline machine learning workflow with Docker!

Docker for Data Science Projects: A Beginner-Friendly Introduction / Image by Author

Table of Contents:

1. Introduction to Docker

1.1. Docker vs Containers vs Docker Images

Docker is a commercial containerization platform and runtime that helps develo build, deploy, and run containers. It uses a client-server architecture with simple commands and automation through a single API.

With Docker, developers can create containerized applications by writing a Docl which is essentially a recipe for building a container image. Docker then provide

of tools to build and manage these container images, making it easier for develop
package and deploy their applications in a consistent and reproducible way.

A container is a lightweight and portable executable software package that inclu
everything an application needs to run, including code, libraries, system tools, ar
settings.

Containers are created from images that define the contents and configuration o
container, and they are isolated from the host operating system and other contai
on the same system.

This isolation is made possible by the use of virtualization and process isolation
technologies, which enable containers to share the resources of a single instance
host operating system while providing a secure and predictable environment for
running applications.

A **Docker Image** is a read-only file that contains all the necessary instructions fo
creating a container. They are used to create and start new containers at runtime

1.2. Importance of Docker for Data Scientists

Docker lets developers access these native containerization capabilities using sir
commands, and automate them through a work-saving application programming
interface (API). Docker offers:

- **Improved and seamless container portability**: Docker containers run withou
modification across any desktop, data center, or cloud environment.
- **Even lighter weight and more granular updates:** Multiple processes can be
combined within a single container. This makes it possible to build an applic
that can continue running while one of its parts is taken down for an update
repair.
- **Automated container creation**: Docker can automatically build a container l
on application source code.

- **Container versioning**: Docker can track versions of a container image, roll b previous versions, and trace who built a version and how. It can even upload the deltas between an existing version and a new one.
- **Container reuse**: Existing containers can be used as base images—essential templates for building new containers.
- **Shared container libraries**: Developers can access an open-source registry containing thousands of user-contributed containers.

2. Getting Started with Docker

Now after introducing Dockers let's see how we can use it for our data science projects. Let's first start with installing Docker on your local machine and after t we will introduce basic Docker commands.

2.1. Installing Docker on Your Machine

Installing Docker on your machine is fairly easy. You can follow the instructions available on the official documentation:

- Instructions to install Docker for Linux.
- Instructions to install Docker for Windows.
- Instructions to install Docker for Mac.

It is important to note that if you like to create your own images and push them Docker Hub, you must create an account on Docker Hub. Think of Docker Hub central place where developers can store and share their Docker images.

2.2. 10 Docker Basic Commands

Now after you have installed Docker on your machine. Let's explore some of the docker commands that you should be familiar with.

1. **docker run:** The "docker run" command is used to create and start a new container based on a Docker image. Here's the basic syntax for running a

container:

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

- **OPTIONS**: Additional options that can be used to customize the container's behavior, such as specifying ports, volumes, environment variables, etc.
- **IMAGE:** The name of the Docker image to use for creating the container.
- **COMMAND:** (Optional) The command to be executed inside the container.
- **ARG**: (Optional) Arguments passed to the command inside the container.

For example, to run a container based on the "ubuntu" image and execute the "ls command inside the container, you would use the following command:

```
docker run ubuntu ls
```

This will create a new container using the "ubuntu" image and run the "ls" comr which lists the files and directories inside the container's file system. Note that i specified image is not available locally, Docker will automatically pull it from a I registry before creating the container.

2. **docker ps:** The "docker ps" command is used to list the running containers on Docker host. It provides information such as the container ID, the image used, tl command being executed, status, and port mappings. Here's the basic syntax:

```
docker ps [OPTIONS]
```

The "docker ps" command is used to list the running containers on your Docker It provides information such as the container ID, the image used, the command l executed, status, and port mappings. Here's the basic syntax:

```
docker ps [OPTIONS]
```

By default, “docker ps” only shows the running containers. If you want to see all containers, including those that are stopped or exited, you can use the “-a” optio

```
docker ps -a
```

3. docker stop: The “docker stop” command is used to stop one or more running containers. It sends a signal to the container’s main process, requesting it to stop gracefully. Here’s the basic syntax:

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

- **OPTIONS:** Additional options that can be used to customize the stop behav For example, you can specify a timeout period with the “—time” or “-t” opti allow the container more time to stop gracefully before forcefully terminatir
- **CONTAINER**: The name or ID of the container(s) to stop. You can specify multiple containers separated by spaces.

For example, to stop a container with the name “my-container”, you would use tl following command:

```
docker stop my-container
```

4. docker rm: The “docker rm” command is used to remove one or more stopped containers from your Docker host. It permanently deletes the specified containe and frees up the associated resources. Here’s the basic syntax:

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

- **OPTIONS**: Additional options that can be used to customize the removal behavior. For example, you can use the "-f" or "—force" option to force the removal of a running container.
- **CONTAINER**: The name or ID of the container(s) to remove. You can specif multiple containers separated by spaces.

```
docker rm my-container
```

If you want to remove multiple containers, you can list their names or IDs separa by spaces:

```
docker rm container1 container2 container3
```

5. docker images: The "docker images" command is used to list the Docker imag that are available on your Docker host. It displays information about the images, as the repository, tag, image ID, creation date, and size. Here's the basic syntax:

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

- OPTIONS: Additional options that can be used to customize the output or fi the images. For example, you can use the "—format" option to specify a forr template for the output or the "-a" or "—all" option to show all images, incl intermediate image layers.
- REPOSITORY: (Optional) The repository name of the image.
- TAG: (Optional) The tag of the image.

By default, the "docker images" command lists all images available on your Dock
host. For example:

```
docker images
```

6. docker rmi: The "**docker rmi**" command is used to remove one or more Docke
images from your Docker host. It permanently deletes the specified image(s) fror
local image cache. Here's the basic syntax:

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

- **OPTIONS**: Additional options that can be used to customize the removal behavior. For example, you can use the "-f" or "—force" option to force the removal of an image, even if it's being used by running containers.
- **IMAGE:** The name or ID of the image(s) to remove. You can specify multiple images separated by spaces.

For example, to remove an image with the name "my-image:latest", you would us
following command:

```
docker rmi my-image:latest
```

If you want to remove multiple images, you can list their names or IDs separated
spaces:

```
docker rmi image1 image2 image3
```

7. docker build: The "**docker build**" command is used to build a Docker image fr
Dockerfile. It allows you to define the instructions and dependencies required to

create a customized image. Here's the basic syntax:

```
docker build [OPTIONS] PATH | URL | -
```

- **OPTIONS**: Additional options that can be used to customize the build proce Some commonly used options include "-t" or "—tag" to specify the name an optional tag for the image, "-f" or "—file" to specify the Dockerfile's locatio "—build-arg" to pass build-time variables to the Dockerfile.
- PATH | URL | -: The path to the directory containing the Dockerfile, a URL Git repository, or "-" to build from the standard input.

For example, to build an image using a Dockerfile located in the current director tag it as "my-image:latest", you would use the following command:

```
docker build -t my-image:latest .
```

The "." indicates that the Dockerfile is in the current directory.

8. docker exec: The "docker exec" command is used to execute a command insid running Docker container. It allows you to run commands interactively or in a detached mode. Here's the basic syntax:

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

- **OPTIONS**: Additional options that can be used to customize the execution behavior. Some commonly used options include "-i" or "—interactive" to ke STDIN open for interactive commands, "-t" or "—tty" to allocate a pseudo- and "-d" or "—detach" to run the command in the background.
- **CONTAINER**: The name or ID of the container where the command should executed.

- **COMMAND**: The command to be executed inside the container.
- **ARG**: (Optional) Arguments passed to the command inside the container.

For example, to execute the “ls” command inside a container named “my-contair
you would use the following command:

```
docker exec my-container ls
```

This will run the “ls” command inside the specified container and display the lis
files and directories.

If you want to run an interactive command, such as starting a shell inside the container, you can use the “-it” options together:

```
docker exec -it my-container bash
```

This will start an interactive shell session inside the container, allowing you to e
multiple commands interactively.

9. docker pull: The “docker pull” command is used to download Docker images f
Docker registry, such as Docker Hub. It retrieves the specified image or images a
saves them to your local image cache. Here’s the basic syntax:

```
docker pull [OPTIONS] IMAGE[:TAG]
```

- **OPTIONS**: Additional options that can be used to customize the pull proces
  Some commonly used options include “—all-tags” to pull all available tags f
  image, “—platform” to specify the platform for which to pull the image, and
  quiet” to suppress the progress output.

- **IMAGE:** The name of the image to pull from the Docker registry. It can be i format "repository/image" or "repository/image:tag". If the tag is not specifie "latest" is used by default.

For example, to pull the latest version of the "ubuntu" image from Docker Hub, y would use the following command:

```
docker pull ubuntu
```

If you want to pull a specific tagged version of the image, you can specify the tag

```
docker pull ubuntu:20.04
```

The specified image will be downloaded from the Docker registry and saved to y local image cache. Once the image is pulled, you can use it to create and run containers on your Docker host.

10. docker push: The "**docker push**" command is used to upload Docker images Docker registry, such as Docker Hub or a private registry. It allows you to share y locally built or modified images with others. Here's the basic syntax:

```
docker push [OPTIONS] NAME[:TAG]
```

- **OPTIONS**: Additional options that can be used to customize the push proce Some commonly used options include "—all-tags" to push all tags for an im: —disable-content-trust" to skip content trust verification, and "—quiet" to suppress the progress output.
- **NAME:** The name of the image to push. It should include the repository and image name. For example, "username/repository:image".

- **TAG:** (Optional) The tag of the image to push. If not specified, the “latest” ta used by default.

Before pushing an image, you need to ensure that you are authenticated to the D registry. You can log in to the registry using the “docker login” command, provic your username, password, and registry URL if necessary.

For example, to push an image named “my-image” with the “latest” tag to Docke Hub, assuming you are logged in to Docker Hub, you would use the following command:

```
docker push username/my-image:latest
```

The specified image will be uploaded to the Docker registry and made available f others to download and use.

3. Dockerizing a Machine Learning Application

To dockerize a machine learning application there are three main steps:

- Create a requirements.txt file
- Write a Dockerfile
- Build the Docker image

Let’s build a simple machine learning application and see a step-by-step guide o to dockerize it. The application below trains a simple classification model (logist regression) on the iris dataset.

```
# Load the libraries

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Load the iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=


# Train a logistic regression model

clf = LogisticRegression()

clf.fit(X_train, y_train)


# Make predictions

y_pred = clf.predict(X_test)


# Print the accuracy of the model

accuracy_score = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy_score}')
```

3.1. Defining the environment

The first step is to define the current environment precisely so that it can be repl in another location. The most effective and straightforward way is to create a requirements.txt file that outlines all the libraries your project is using, includin versions. To create this file you can simply run the following command in the command line:

```
pip3 freeze > requirements.txt  # Python3
```

This will generate a requirements.txt file with all the used packages and librarie the exact version used.

3.2. Write a Dockerfile

The next step is to create a file named Dockerfile that can create the environme executes our application in it.

```
FROM python:3.9

WORKDIR /src

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python","iris_classification.py"]
```

This Dockerfile uses the official Python image as the base image, sets the workir directory, copies the requirements.txt file, installs the dependencies, copies the application code, and runs the python iris_classification.py command to start th application.

3.3. Build the Image

The final step to create a reproducible environment is to create an image (also kr as a template) that can be run to create any number of containers with the same configurations.

You can build the image by running the command `docker build -t <image name> .` in the same directory where the Dockerfile is located.

Looking to start a career in data science and AI, and do not know how. I offer c science mentoring sessions and long-term career mentoring:

- **Mentoring sessions:** https://lnkd.in/dXeg3KPW
- **Long-term mentoring:** https://lnkd.in/dtdUYBrM

31 Likes · 3 Restacks

← Previous Nex

Discussion about this post

Comments Restacks

Write a comment...