Data science with JupyterLab

- What is JupyterLab?
- Why use Docker and JupyterLab together?
- Prerequisites
- Run and access a JupyterLab container
- Save and access notebooks
  - Run a JupyterLab container with a volume
  - Save a notebook to the volume
- Customize your JupyterLab environment
  - Define your environment in a Dockerfile
  - Build your environment into an image
- Run your image as a container
- Use Compose to run your container
  - Create a Compose file
  - Run your container using Compose
- Share your work
  - Share your image
  - Share your volume
- Summary

Docker and JupyterLab are two powerful tools that can enhance your data science workflow. In this guide, you will learn how to use them together to create and run reproducible data science environments. This guide is based on Supercharging AI/ML Development with JupyterLab and Docker.

In this guide, you'll learn how to:

- Run a personal Jupyter Server with JupyterLab on your local machine
- Customize your JupyterLab environment
- Share your JupyterLab notebook and environment with other data scientists

What is JupyterLab?

JupyterLab is an open source application built around the concept of a computational notebook document. It enables sharing and executing code, data processing, visualization, and offers a range of interactive features for creating graphs.

Why use Docker and JupyterLab together?

By combining Docker and JupyterLab, you can benefit from the advantages of both tools, such as:

- Containerization ensures a consistent JupyterLab environment across all deployments, eliminating compatibility issues.
- Containerized JupyterLab simplifies sharing and collaboration by removing the need for manual environment setup.
- Containers offer scalability for JupyterLab, supporting workload distribution and efficient resource management with platforms like Kubernetes.

Prerequisites

To follow along with this guide, you must install the latest version of Docker Desktop.

Run and access a JupyterLab container

In a terminal, run the following command to run your JupyterLab container.

```
$ docker run --rm -p 8889:8888 quay.io/jupyter/base-notebook start-notebook.py --
NotebookApp.token='my-token'
```

The following are the notable parts of the command:

- `-p 8889:8888`: Maps port 8889 from the host to port 8888 on the container.

- `start-notebook.py --NotebookApp.token='my-token'`: Sets an access token rather than using a random token.

For more details, see the Jupyter Server Options and the docker run CLI reference.

If this is the first time you are running the image, Docker will download and run it. The amount of time it takes to download the image will vary depending on your network connection.

After the image downloads and runs, you can access the container. To access the container, in a web browser navigate to localhost:8889/lab?token=my-token.

To stop the container, in the terminal press `ctrl`+`c`.

To access an existing notebook on your system, you can use a bind mount. Open a terminal and change directory to where your existing notebook is. Then, run the following command based on your operating system.

{{< tabs >}} {{< tab name="Mac / Linux" >}}

```
$ docker run --rm -p 8889:8888 -v "$(pwd):/home/jovyan/work" quay.io/jupyter/base-notebook
start-notebook.py --NotebookApp.token='my-token'
```

{{< /tab >}} {{< tab name="Windows (Command Prompt)" >}}

```
$ docker run --rm -p 8889:8888 -v "%cd%":/home/jovyan/work quay.io/jupyter/base-notebook
start-notebook.py --NotebookApp.token='my-token'
```

{{< /tab >}} {{< tab name="Windows (PowerShell)" >}}

```
$ docker run --rm -p 8889:8888 -v "$(pwd):/home/jovyan/work" quay.io/jupyter/base-notebook
start-notebook.py --NotebookApp.token='my-token'
```

{{< /tab >}} {{< tab name="Windows (Git Bash)" >}}

```
$ docker run --rm -p 8889:8888 -v "/$(pwd):/home/jovyan/work" quay.io/jupyter/base-notebook
start-notebook.py --NotebookApp.token='my-token'
```

{{< /tab >}} {{< /tabs >}}

The `-v` option tells Docker to mount your current working directory to `/home/jovyan/work` inside the container. By default, the Jupyter image's root directory is `/home/jovyan` and you can only access or save notebooks to that directory in the container.

Now you can access localhost:8889/lab?token=my-token and open notebooks contained in the bind mounted directory.

To stop the container, in the terminal press `ctrl` + `c`.

Docker also has volumes, which are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.

Save and access notebooks

When you remove a container, all data in that container is deleted. To save notebooks outside of the container, you can use a volume.

Run a JupyterLab container with a volume

To start the container with a volume, open a terminal and run the following command

```
$ docker run --rm -p 8889:8888 -v jupyter-data:/home/jovyan/work quay.io/jupyter/base-notebook
start-notebook.py --NotebookApp.token='my-token'
```

The `-v` option tells Docker to create a volume named `jupyter-data` and mount it in the container at `/home/jovyan/work`.

To access the container, in a web browser navigate to localhost:8889/lab?token=my-token. Notebooks can now be saved to the volume and will accessible even when the container is deleted.

Save a notebook to the volume

For this example, you'll use the Iris Dataset example from scikit-learn.

1. Open a web browser and access your JupyterLab container at localhost:8889/lab?token=my-token.
2. In the **Launcher**, under **Notebook**, select **Python 3**.
3. In the notebook, specify the following to install the necessary packages.

   ```
   !pip install matplotlib scikit-learn
   ```

4. Select the play button to run the code.

5. In the notebook, specify the following code.

```
from sklearn import datasets


iris = datasets.load_iris()
import matplotlib.pyplot as plt


_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
_ = ax.legend(
   scatter.legend_elements()[0], iris.target_names, loc="lower right",
title="Classes"
)
```

6. Select the play button to run the code. You should see a scatter plot of the Iris dataset.
7. In the top menu, select **File** and then **Save Notebook**.
8. Specify a name in the `work` directory to save the notebook to the volume. For example, `work/mynotebook.ipynb`.
9. Select **Rename** to save the notebook.

The notebook is now saved in the volume.

In the terminal, press `ctrl` + `c` to stop the container.

Now, any time you run a Jupyter container with the volume, you'll have access to the saved notebook.

When you do run a new container, and then run the data plot code again, it'll need to run `!pip install matplotlib scikit-learn` and download the packages. You can avoid reinstalling packages every time you run a new container by creating your own image with the packages already installed.

Customize your JupyterLab environment

You can create your own JupyterLab environment and build it into an image using Docker. By building your own image, you can customize your JupyterLab environment with the packages and tools you need, and ensure that it's consistent and reproducible across different deployments. Building your own image also makes it easier to share your JupyterLab environment with others, or to use it as a base for further development.

Define your environment in a Dockerfile

In the previous Iris Dataset example from Save a notebook to the volume, you had to install the dependencies, `matplotlib` and `scikit-learn`, every time you ran a new container. While the dependencies in that small example download and install quickly, it may become a problem as your list of dependencies grow. There may also be other tools, packages, or files that you always want in your environment.

In this case, you can install the dependencies as part of the environment in the image. Then, every time you run your container, the dependencies will always be installed.

You can define your environment in a Dockerfile. A Dockerfile is a text file that instructs Docker how to create an image of your JupyterLab environment. An image contains everything you want and need when running JupyterLab, such as files, packages, and tools.

In a directory of your choice, create a new text file named `Dockerfile`. Open the `Dockerfile` in an IDE or text editor and then add the following contents.

```
# syntax=docker/dockerfile:1


FROM quay.io/jupyter/base-notebook
RUN pip install --no-cache-dir matplotlib scikit-learn
```

This Dockerfile uses the `quay.io/jupyter/base-notebook` image as the base, and then runs `pip` to install the dependencies. For more details about the instructions in the Dockerfile, see the Dockerfile reference.

Before you proceed, save your changes to the `Dockerfile`.

Build your environment into an image

After you have a `Dockerfile` to define your environment, you can use `docker build` to build an image using your `Dockerfile`.

Open a terminal, change directory to the directory where your `Dockerfile` is located, and then run the following command.

```
$ docker build -t my-jupyter-image .
```

The command builds a Docker image from your `Dockerfile` and a context. The `-t` option specifies the name and tag of the image, in this case `my-jupyter-image`. The `.` indicates that the current directory is the context, which means that the files in that directory can be used in the image

creation process.

You can verify that the image was built by viewing the `Images` view in Docker Desktop, or by running the `docker image ls` command in a terminal. You should see an image named `my-jupyter-image`.

Run your image as a container

To run your image as a container, you use the `docker run` command. In the `docker run` command, you'll specify your own image name.

```
$ docker run --rm -p 8889:8888 my-jupyter-image start-notebook.py --NotebookApp.token='my-token'
```

To access the container, in a web browser navigate to localhost:8889/lab?token=my-token.

You can now use the packages without having to install them in your notebook.

1. In the **Launcher**, under **Notebook**, select **Python 3**.
2. In the notebook, specify the following code.

   ```
   from sklearn import datasets

   iris = datasets.load_iris()
   import matplotlib.pyplot as plt

   _, ax = plt.subplots()
   scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
   ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
   _ = ax.legend(
      scatter.legend_elements()[0], iris.target_names, loc="lower right",
   title="Classes"
   )
   ```

3. Select the play button to run the code. You should see a scatter plot of the Iris dataset.

In the terminal, press `ctrl` + `c` to stop the container.

Use Compose to run your container

Docker Compose is a tool for defining and running multi-container applications. In this case, the application isn't a multi-container application, but Docker Compose can make it easier to run by defining all the `docker run` options in a file.

Create a Compose file

To use Compose, you need a `compose.yaml` file. In the same directory as your `Dockerfile`, create a new file named `compose.yaml`.

Open the `compose.yaml` file in an IDE or text editor and add the following contents.

```
services:
  jupyter:
    build:
      context: .
    ports:
      - 8889:8888
    volumes:
      - jupyter-data:/home/jovyan/work
    command: start-notebook.py --NotebookApp.token='my-token'

volumes:
  jupyter-data:
    name: jupyter-data
```

This Compose file specifies all the options you used in the `docker run` command. For more details about the Compose instructions, see the Compose file reference.

Before you proceed, save your changes to the `compose.yaml` file.

Run your container using Compose

Open a terminal, change directory to where your `compose.yaml` file is located, and then run the following command.

```
$ docker compose up --build
```

This command builds your image and runs it as a container using the instructions specified in the `compose.yaml` file. The `--build` option ensures that your image is rebuilt, which is necessary if you made changes to your `Dockerfile`.

To access the container, in a web browser navigate to localhost:8889/lab?token=my-token.

In the terminal, press `ctrl` + `c` to stop the container.

Share your work

By sharing your image and notebook, you create a portable and replicable research environment that can be easily accessed and used by other data scientists. This process not only facilitates collaboration but also ensures that your work is preserved in an environment where it can be run without compatibility issues.

To share your image and data, you'll use Docker Hub. Docker Hub is a cloud-based registry service that lets you share and distribute container images.

Share your image

1. Sign up or sign in to Docker Hub.
2. Rename your image so that Docker knows which repository to push it to. Open a terminal and run the following `docker tag` command. Replace `YOUR-USER-NAME` with your Docker ID.

   ```
   $ docker tag my-jupyter-image YOUR-USER-NAME/my-jupyter-image
   ```

3. Run the following `docker push` command to push the image to Docker Hub. Replace `YOUR-USER-NAME` with your Docker ID.

   ```
   $ docker push YOUR-USER-NAME/my-jupyter-image
   ```

4. Verify that you pushed the image to Docker Hub.
   1. Go to Docker Hub.
   2. Select **My Hub** > **Repositories**.
   3. View the **Last pushed** time for your repository.

Other users can now download and run your image using the `docker run` command. They need to replace `YOUR-USER-NAME` with your Docker ID.

```
$ docker run --rm -p 8889:8888 YOUR-USER-NAME/my-jupyter-image start-notebook.py --
NotebookApp.token='my-token'
```

Share your volume

This example uses the Docker Desktop graphical user interface. Alternatively, in the command line interface you can back up the volume and then push it using the ORAS CLI.

1. Sign in to Docker Desktop.
2. In the Docker Dashboard, select **Volumes**.
3. Select the **jupyter-data** volume by selecting the name.
4. Select the **Exports** tab.
5. Select **Quick export**.
6. For **Location**, select **Registry**.
7. In the text box under **Registry**, specify your Docker ID, a name for the volume, and a tag. For example, `YOUR-USERNAME/jupyter-data:latest`.
8. Select **Save**.
9. Verify that you exported the volume to Docker Hub.
    1. Go to Docker Hub.
    2. Select **My Hub** > **Repositories**.
    3. View the **Last pushed** time for your repository.

Other users can now download and import your volume. To import the volume and then run it with your image:

1. Sign in to Docker Desktop.
2. In the Docker Dashboard, select **Volumes**.
3. Select **Create** to create a new volume.
4. Specify a name for the new volume. For this example, use `jupyter-data-2`.
5. Select **Create**.
6. In the list of volumes, select the **jupyter-data-2** volume by selecting the name.
7. Select **Import**.
8. For **Location**, select **Registry**.
9. In the text box under **Registry**, specify the same name as the repository that you exported your volume to. For example, `YOUR-USERNAME/jupyter-data:latest`.
10. Select **Import**.
11. In a terminal, run `docker run` to run your image with the imported volume. Replace `YOUR-USER-NAME` with your Docker ID.

```
$ docker run --rm -p 8889:8888 -v jupyter-data-2:/home/jovyan/work YOUR-USER-NAME/my-jupyter-
image start-notebook.py --NotebookApp.token='my-token'
```

Summary

In this guide, you learned how to leverage Docker and JupyterLab to create reproducible data science environments, facilitating the development and sharing of data science projects. This included, running a personal JupyterLab server, customizing the environment with necessary tools and packages, and sharing notebooks and environments with other data scientists.

Related information:

- Dockerfile reference
- Compose file reference
- Docker CLI reference
- Jupyter Docker Stacks docs

□□□ #2
□ □□□ □□□ 2026-10-31 18:45:58 UTC
□ □□□ □□□ 2026-10-31 18:45:59 UTC