2.077 Final Project

Plasticity

Jorge Nin

December 20, 2023

Contents

1 Introduction and Background

This document will help install and setup a docker instance of FEniCSx used for 2.077. We will be using FEniCSx 0.8.0 Dev.

While this is an unstable version right now it is the only version with good Quadrature Element support. If you want the stable version just pull the image:

```
> docker pull jorgenin/dolfinx-2.077:v0.7
```

Listing 1: Docker Pull Command

These instructions are made specifically with Mac OS in mind, but after installing docker, the process should be platform independent.

2 Docker

WARNING: Please backup your data. Data stored in a container may be lost at any time. Use git or a volume to ensure your data is well protected.

2.1 What is Docker

Docker is a platform that uses containerization technology to simplify the deployment and management of software applications. In the context of software development, especially for complex engineering applications like FeniCSx, Docker offers several advantages:

- Containerization:
  - Containers are isolated environments where applications can run. They are lightweight because they share the host system's kernel (but not the OS itself) and are not burdened with the overhead of a virtual machine.
  - For FEniCSx, this means that each instance of the application runs in a consistent environment, irrespective of the underlying hardware or software configurations of the host system.
- Consistency and Reproducibility:
  - Docker ensures that FEniCSX runs identically on every machine, avoiding the common "it works on my machine" problem in software development.
- Ease of Distribution and Version Control:
  - Docker images (blueprints for containers) can be version-controlled and shared through Docker Hub or other registries, ensuring that all students and researchers access the same version of FEniCSX and it's dependencies.

Because of these advantages, we will be using Docker to run FEniCSx.

2.2 Installing Docker

It is recommended that you install docker desktop for first time use. You can find the download here: Docker Desktop and download the appropriate version for your operating system. An account is not necessary for basic install and usage.

After downloading open up the image and drag it into the applications folder. Docker should now be installed.

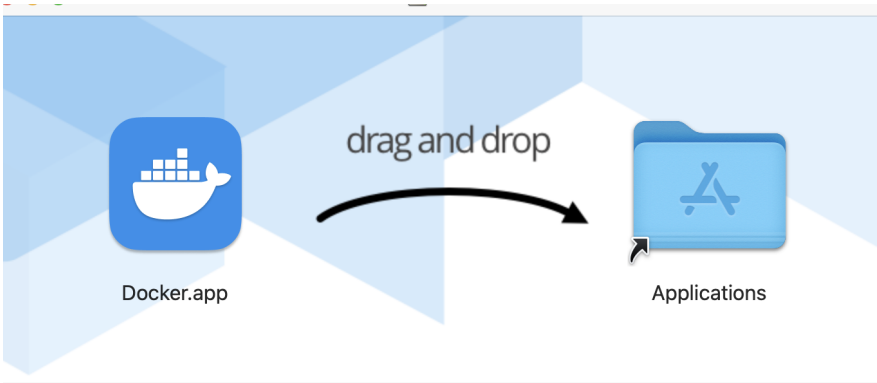
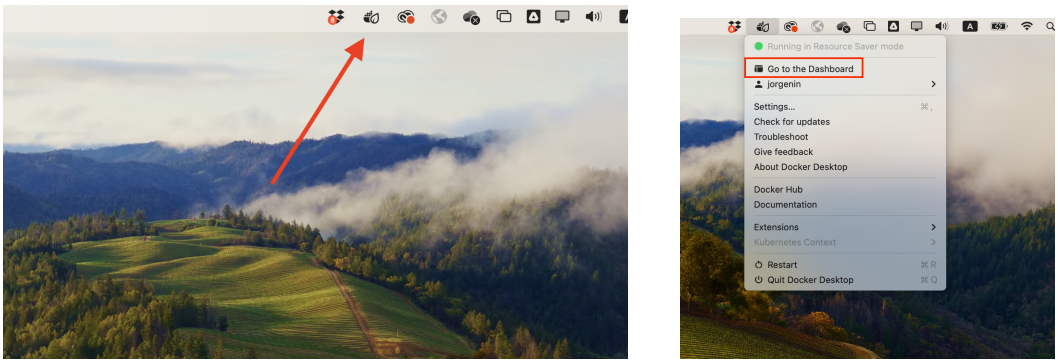


Figure 1: Docker install process

After docker has finished installing open up the docker desktop application.

The application may not instantly open. Insead it may be in the background, look at figure 2 to see if you're docker is in the background.



(a) Docker Open in the background

(b) Clicking on the Docker Symbol

Figure 2: How to check if docker is open in the background. Click on the "Go to dashboard" to open up docker.

2.3 Docker Layout

Once docker is open you should see a window similar to this. This is the dashboard.

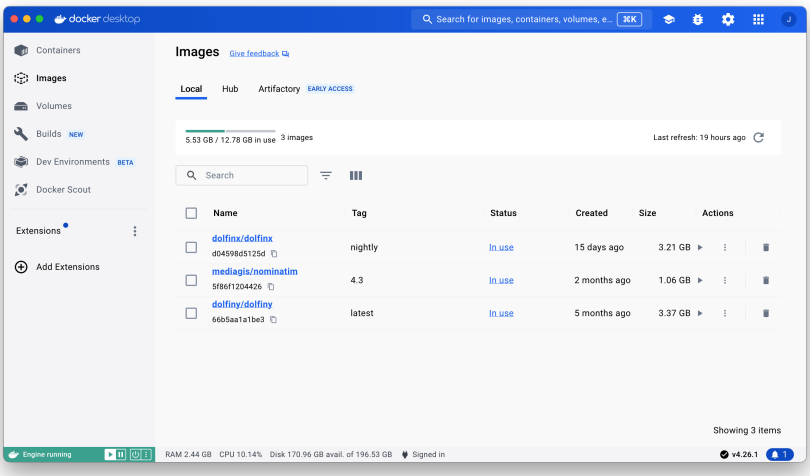


Figure 3: The Docker Dashboard

There are multiple tabs but we will only be convering two of them. The Containers tab, and the images tab.

2.3.1 Images

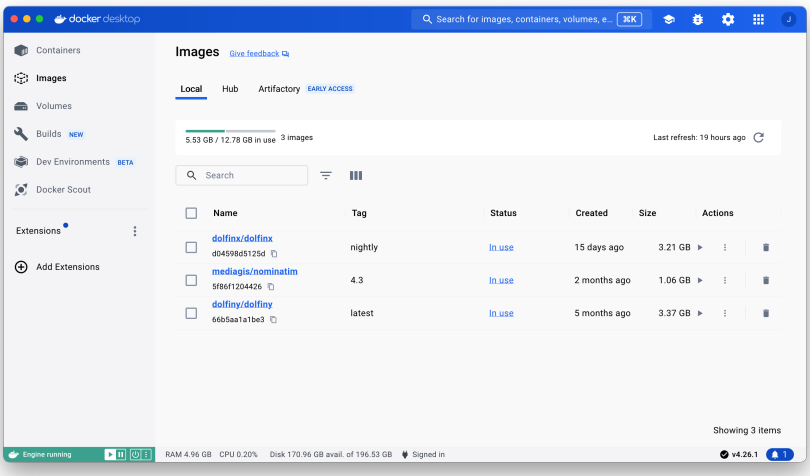


Figure 4: The Docker Images Tab

The docker images tab contains all of the Images you currently have installed. Image are essentially different operating sytems you can have. If you want to run multiple version of FEniCSx for example you could have multiple images of it.

2.3.2 Containers

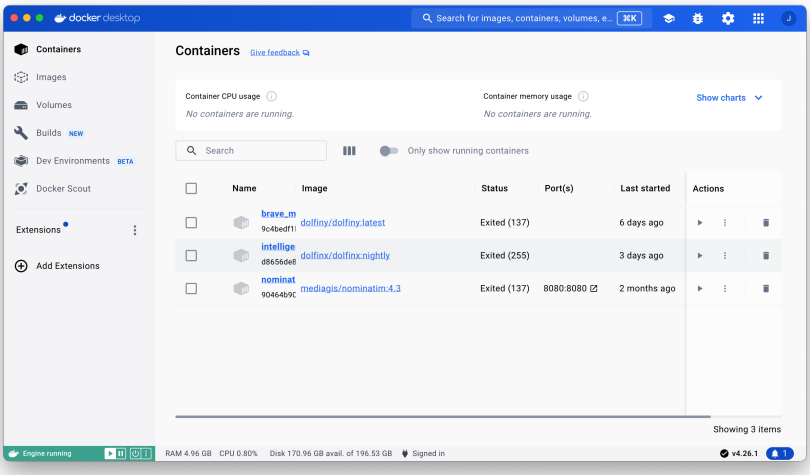


Figure 5: The Docker Containers Tab

The docker containers tab has all of the container you have setup. Each container is essentially a different instance of a specific image. It is normally recommended that for every project you have a different seperate container made. Though those containers could use all the same image.

2.4 Recommended Settings

Because we will be running FEA software which will be want to use most of our computers resources, there are a few changes that should be made to optimize performance.

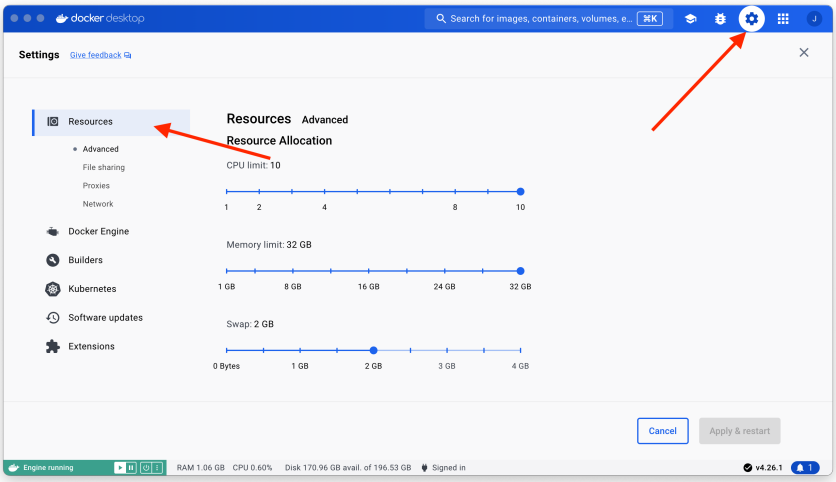


Figure 6: The Docker Settings tab

We will be going to the docker settings tab, and then resources as seen in Figure 6. In here we will be maximizing the number of CPU's available and the memory limit.

3 FEniCSx Container Setup

In this section we will be setting up the FEniCSx image and container so that it runs properly. We will also explain how to mount a folder in the volume so that it is shared between the host and docker container.

3.1 Pulling the Image

First we will need to pull the image from the docker hub. To do this we will need to open up the terminal.
On a mac we can press ctrl+space and then type in terminal to open it up.

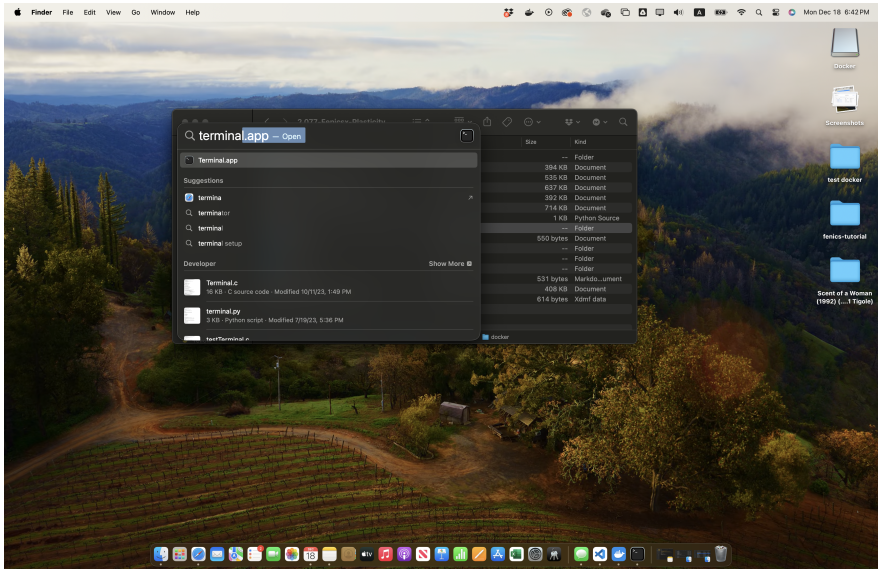


Figure 7: The Docker Settings tab

Once the terminal is open we will need to type in the following command:

```
> docker pull jorgenin/dolfinx-2.077:v0.8
```

Listing 2: Docker Pull Command

Which will download the image and prepare it on your computer.
Figure 8 shows what the output will be.

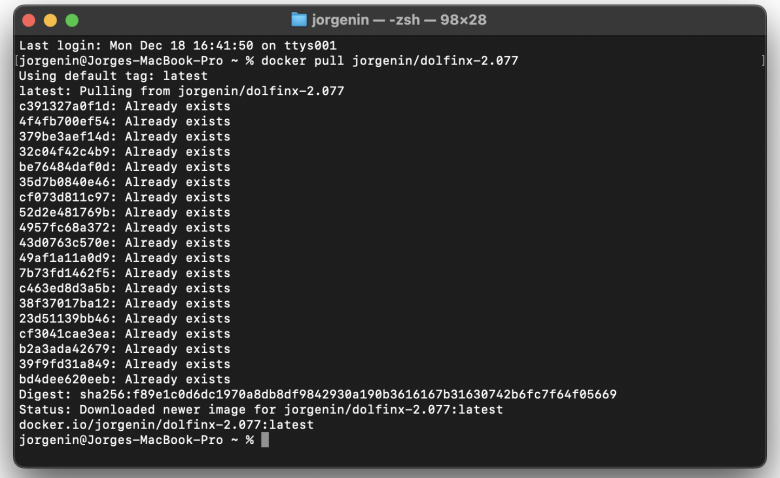


Figure 8: Output from the terminal

3.2 Creating the Container

Now that we have the image downloaded we can create a container from it.

```
> docker run -it jorgenin/dolfinx-2.077:v0.8
```

Listing 3: Docker Pull Command

This command will create the container and run it in interactive mode. Explaining it in more detail: `docker run` will create a container from the image we specify. `-it` will run the container in interactive mode. `jorgenin/dolfinx-2.077` is the image we want to use. `:v0.8` is the tag. We could also use for example `:latest` to get the latest version of the image or `:v0.7` to get the 0.7 version of the image.

Upon executing this command we will be inside of the container and activly running the image.

If we look at the docker desktop application we will see that a container has been created as showing in Figure 9

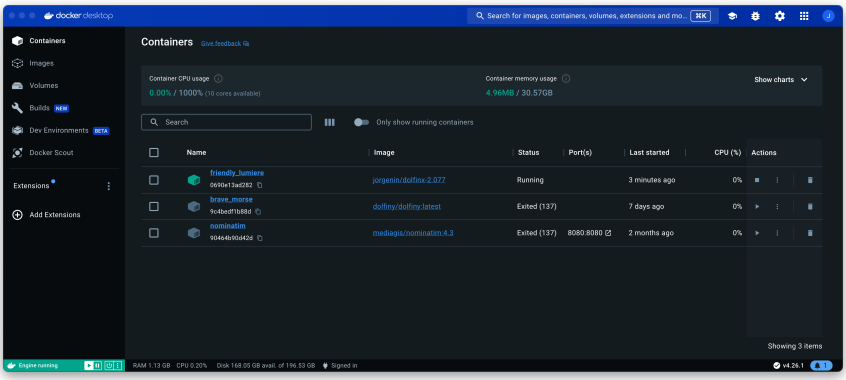


Figure 9: And image from the docker desktop application showing the docker container running.

We can exit the container by typing `exit`.

3.3 Connecting to a Container that was created

If we run the past section multiple times we'll create multiple independent containers. As such when developing we'll normally want to just connect to a conatiner that has already been created.

We can do that through either the terminal or the docker desktop application.

3.3.1 Connecting through a terminal

We can run the following command to run the container interactivly (connecitng to the container immediatly)

```
> docker start -i "container name":"container tag"
```

If we look at Figure 9 we can see the container we created is called `friendly_lumiere`. So in our case the command would be:

```
> docker start -i friendly_lumiere
```

If we just want the command to run the background we can just omit the `-i`. This has the advantage that if we close the terminal the container will continue to run.

```
> docker start friendly_lumiere
```

We could then connect to it at any point by running:

```
> docker attach friendly_lumiere
```

3.3.2 Connecting through the Docker Desktop Application

The docker desktop application is more intuitive (though it does have limitations). To start a container we can just click the play button next to the container we want to run as shown in Figure 10.

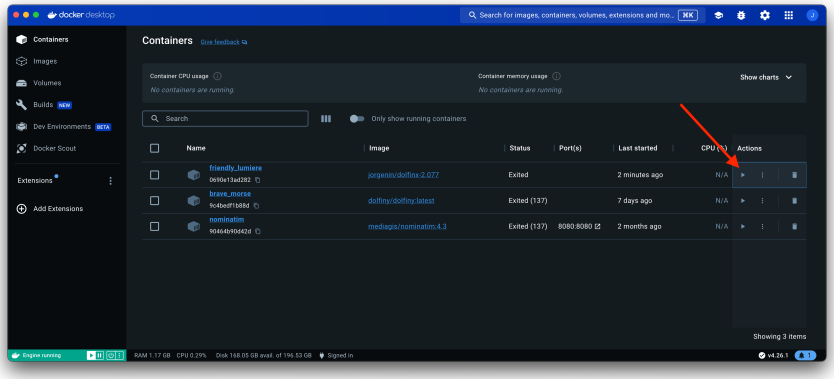


Figure 10: How to start a container in docker desktop

If we want to attach to a running container we can then click on the three dots and then click on `Open In Terminal` as seen in 11

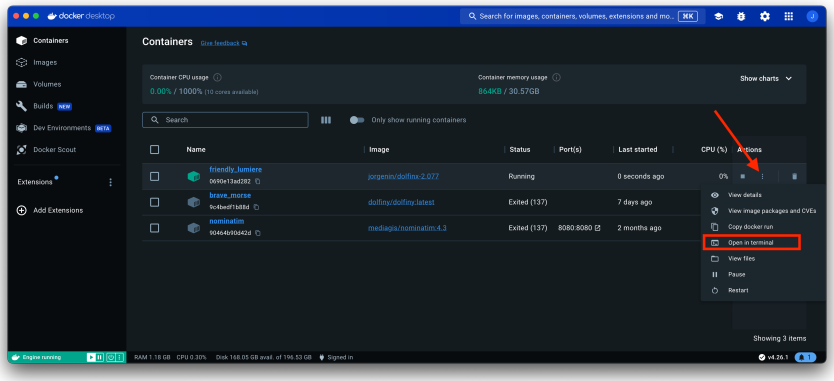


Figure 11: How to Connect to DockerTerminal

3.4 Sharing A Folder

To share (or mount) a folder between the host and the container we will need to use the `-v` flag when creating the container.

```
> docker run -it -v "folder path":"container folder path" jorgenin/dolfinx-2.077:v0.8
```

For example if we wanted to share the folder `/users/jorgenin/Documents/2.077-FEniCSx-Finite-Elasticity` with the container we would run the following command:

```
> docker run -it -v /users/jorgenin/Documents/2.077-FEniCSx-Finite-Elasticity:/home/
  project jorgenin/dolfinx-2.077:v0.8
```

Now any files that are in the folder on the OS will be shown to the container and viceversa.

4 VSCode

VS Code will be our main development software we will be using. It's very flexible and supports a multitude of programming languages. To install it download it from Microsoft here VSCode Download and install it. The process is very similar to installing docker.

4.1 VSCode Extensions

The main idea behind vscode is that can be easily customized to any usecase through the liberal use of extensions.

To simplify the process we have created a custom extension pack with most everything you'll need to get started. To download and install it click on the following link: VSCode FEniCSx Extension Pack and click on install.

Another way to do so (and to install your own extensions) is to click on the extensions tab on the left side of the screen and then search for the extension you want to install. You can see the process in Figure 12.

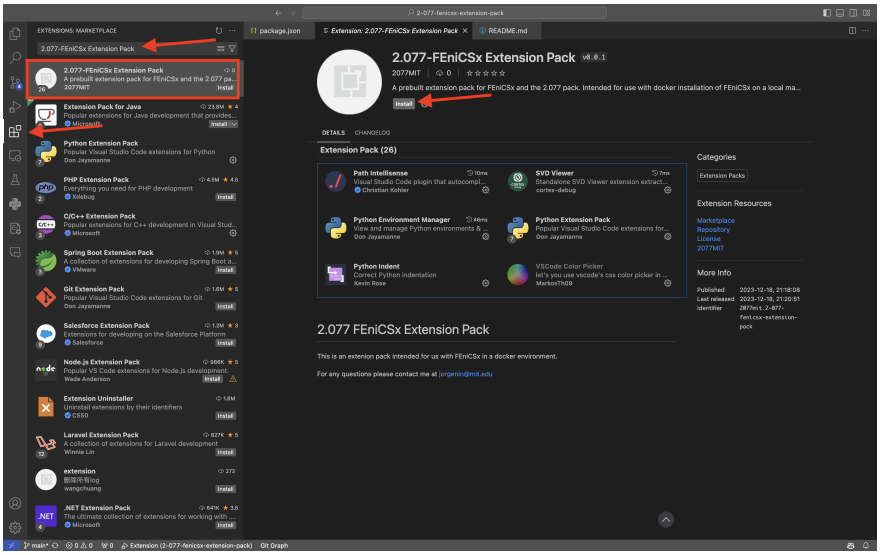


Figure 12: How to install extensions in vscode

5 Running Example Program

With the previous steps completed we should now be prepared to actually run a real example. Here I will walk through step by step from initializing an image to finally running it, and connecting to it through VS Code.

Through this section we will assume you have done all of the basic setup steps previous mentioned.

5.1 Pulling the Image

To pull an image we will open up a terminal and run the command:

```
> docker pull jorgenin/dolfinx-2.077:v0.8
```

Listing 4: Docker Pull Command

We will see the following output from the terminal:

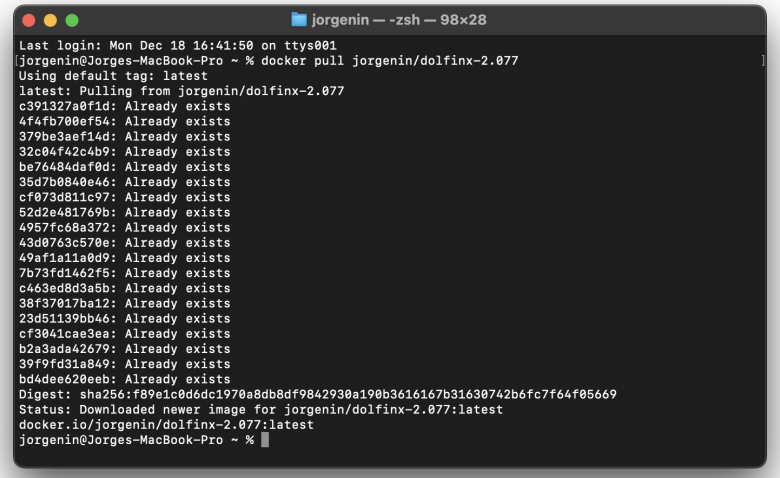


Figure 13: Output from the terminal

5.2 Creating the Container

Here we will be creating the container and mounting a local folder to it from where we will be running all of the results. We will be mounting a folder `/Users/jorgenin/Desktop/fenicsx-tutorial` to the opening folder of the docker instance `/home/project`.

The command is:

```
> docker run -it -v /Users/jorgenin/Desktop/fenicsx-tutorial:/home/project jorgenin/dolfinx
    -2.077:v0.8
```

Listing 5: Docker Pull Command

We can see the results in Figure 14.

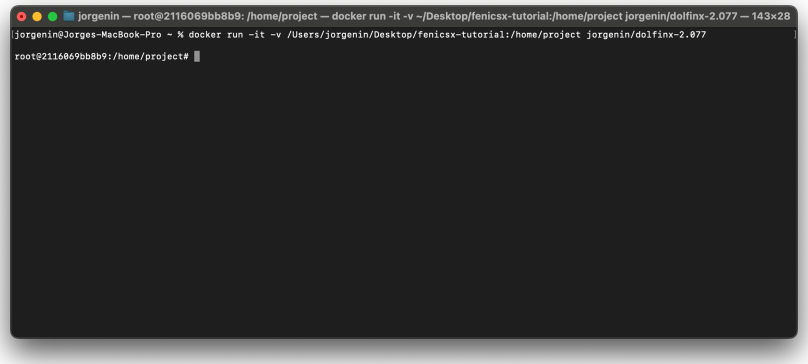


Figure 14: Output from the terminal after mounting image

5.3 Connecting to the Container through VSCode

We now launch vscode and connect to the container we just created. We can go to the docker extension, find the container we just launched right click on it and then click attach `Visual Studio Code`.

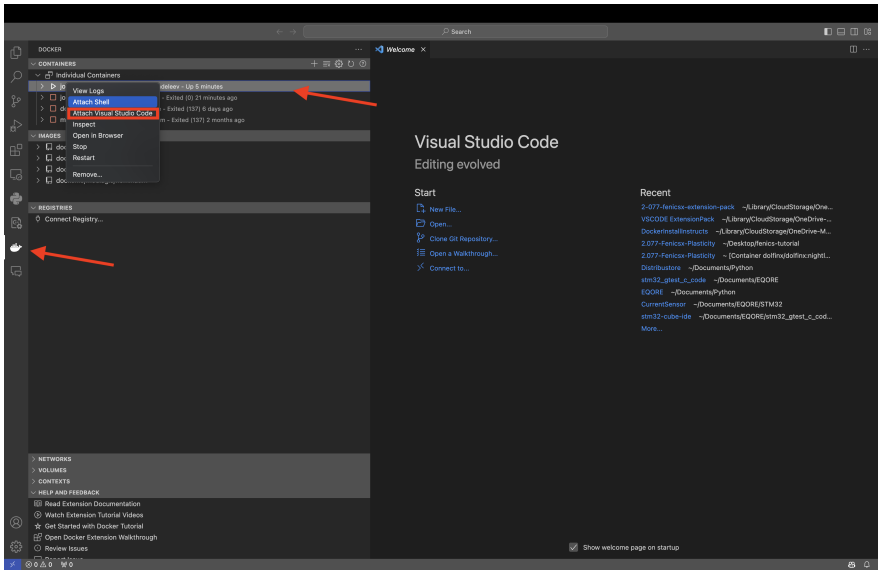


Figure 15: How to open up the docker extension and launch into the file. Right clicking on any of the containers will allow us to attach a VSCode instance.

This will open up a new tab of vscode. We can ensure we are connected by looking at the bottom left corner of the screen.

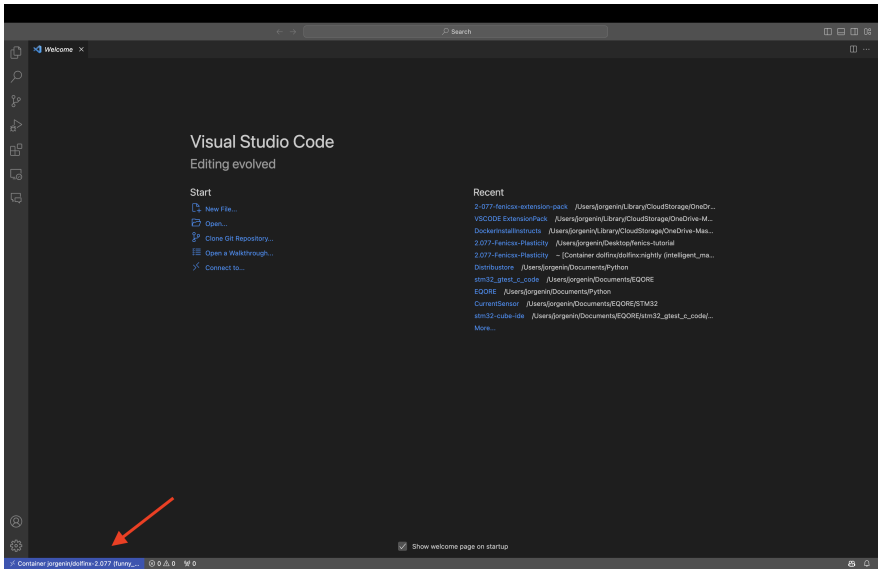


Figure 16: How to open up the docker extension and launch into the file. Right clicking on any of the containers will allow us to attach a VSCode instance.

5.4 Enabling Extensions In Container

By default VSCode extensions are not installed in the container. To enable them we can just go to the exensions, and search for the 2.077 FeniCSx exension pack and insall it.

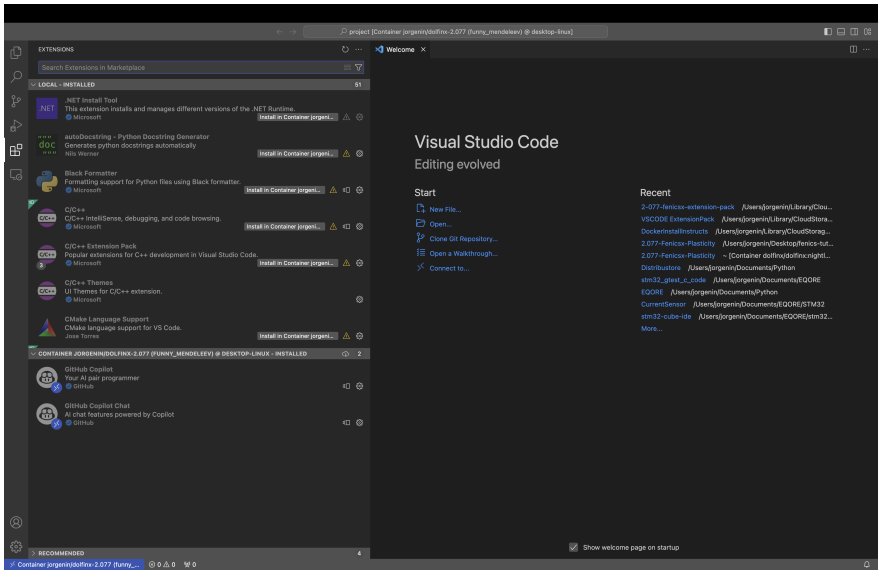


Figure 17: Exensions in the host are not automatically carried over to the conatiner. It is the same process as before to install the extensions into the container.

5.5 Cloning a respository

We can now open up our code with VSCode.

By default VS-Code connects to the root folder of the container. We normally want to avoid adding files into here if possible. What we will do is use the terminal to clone a repository from git into the shared folder we created earlier. We will then open that folder with vscode.

We can open up the terminal by click on the terminal tab on the top of the screen and then clicking on new terminal.

The terminal will open in the bottom of the screen.

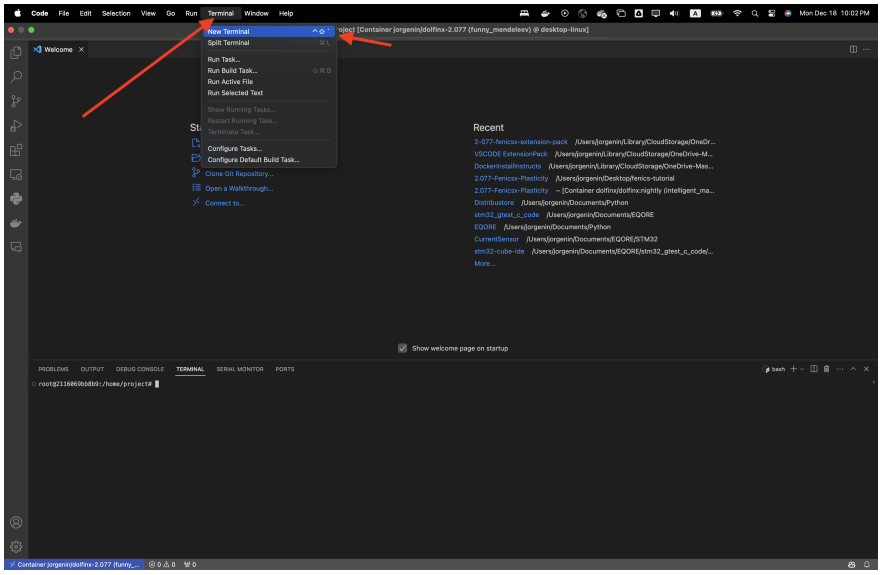


Figure 18: How to open a Terminal in VSCode

Our shared folder is `/home/project` so we will navigate to it by running the following command:

```
> cd /home/project
```

Listing 6: Docker Pull Command

We can then clone the repository by running the following command:

```
> git clone https://github.com/jorgenin/2.077-Fenicsx-Plasticity
```

Listing 7: Docker Pull Command

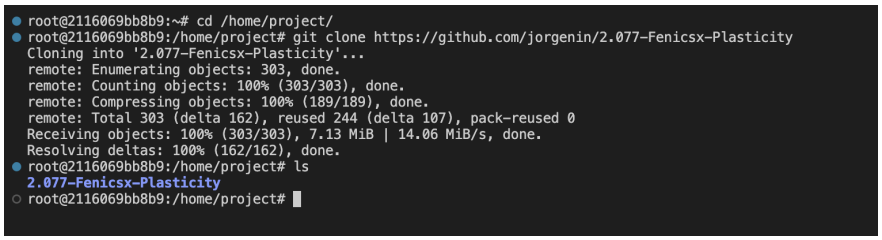


Figure 19: Ouput from the terminal after running commands

After Running the commands we can look in our host computer and see that the files were also in the shared folder. This is because we created the files in our shared folder.

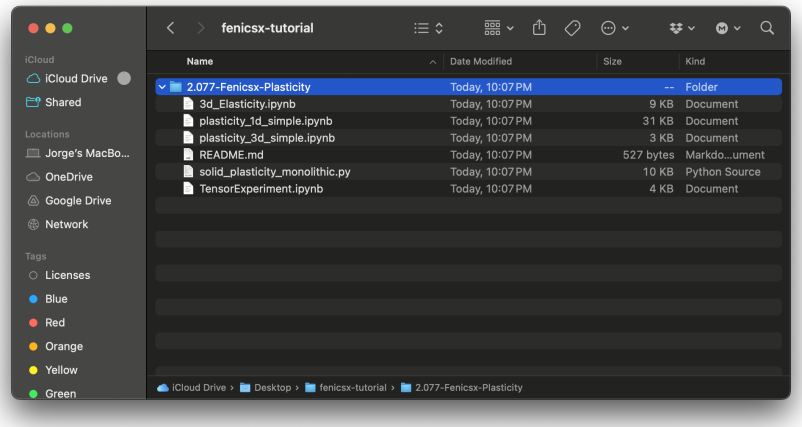


Figure 20: The git clone from the container also created the files in our shared folder. This is because we shared the folder initially when initializing the container.

5.6 Opening Folder and Running Example

We can now open up the folder in vscode and run the example.

We can do this in two ways, either in the welcome screen we can click open, or we can click on file and then on open. You can see this in figure 21.

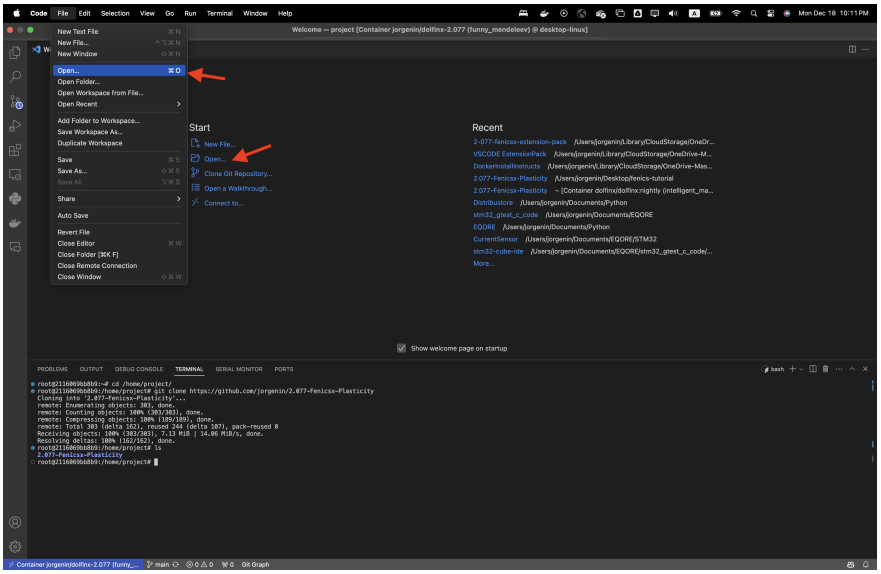


Figure 21: How to open a folder in VSCode

Either method opens the same window where we can select the folder we want to open. in this case we want to open the `2.077-Fenicsx-Plasticity` folder.

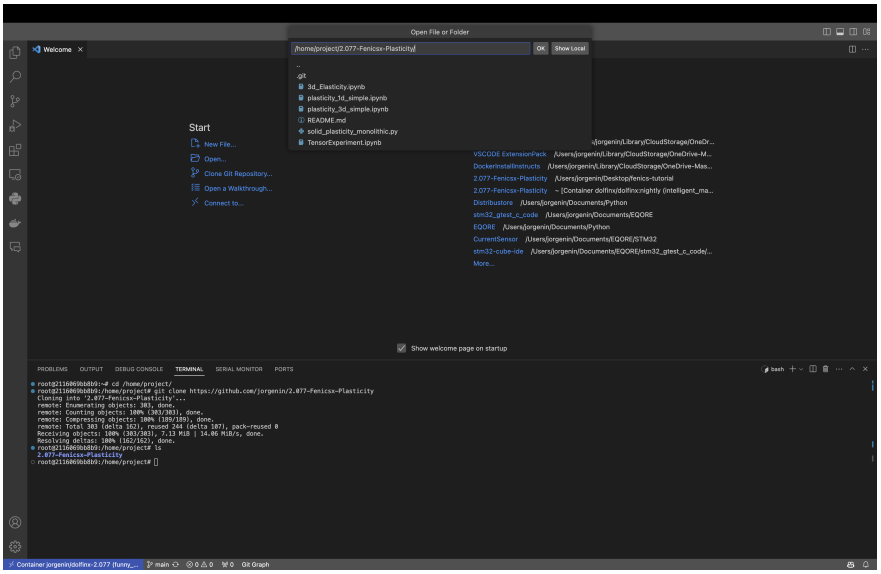


Figure 22: Selecting the FeniCSx Plasticity folder

We can now see the folder in the file explorer tab in VSCode. We can open up any file in the project and edit / or run it as necessary.

Jupyter notebooks will open inline. We can make sure we are using the right python verion (3.10 at the time of writing) and then clicking run all to run the entire example as seen in figure 23.

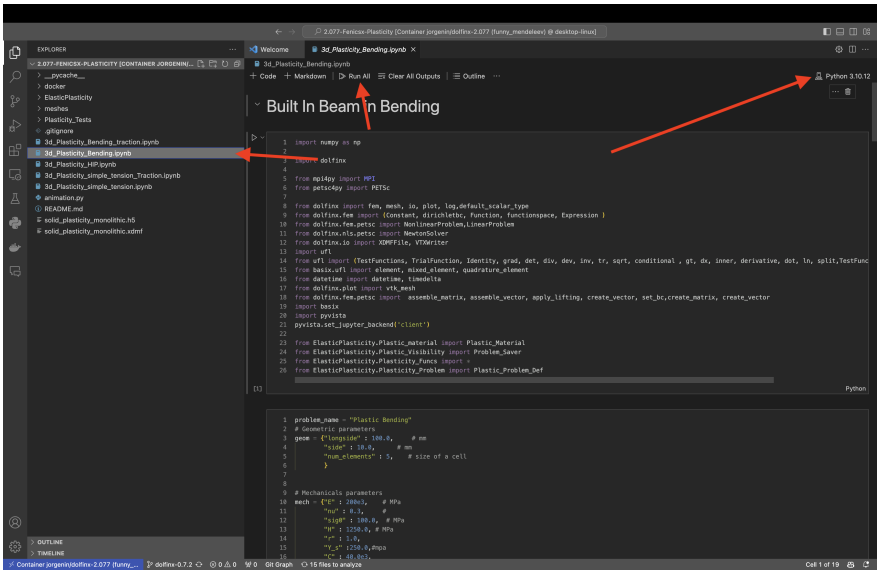


Figure 23: Running the Jupyter Notebook

6 Appendix

Here is a link to my github respository: GitHub FEniCSX Finite Plasticity

7 Creating a new Docker Image

From time to time you may want to create you're own custom docker image. For example if there is a new update to FEniCSx or if you want specific dependencies automatically intstalled in an image. This section will be covering how to do this, and then uploading that image to DockerHUB, to share the image with others.

7.1 Docker Files

Images are normally defined by docker files. These are essentially a list of commands that docker will run to create the image. You can find the docker file that was used in this tutorial here.

```
FROM dolfinx/dolfinx:nightly
ARG TARGETARCH
SHELL ["/bin/bash", "-c"]
LABEL maintainer="Jorge Nin <jorgenin@Mit.edu>"

# Install dependencies
RUN apt-get update && apt-get clean && apt-get install -y \
    xvfb vim ffmpeg


# Install python dependencies
RUN pip3 install --upgrade pip

RUN if [[ ${TARGETARCH} =~ "arm64" ]]; then \
    pip3 install "https://github.com/finsberg/vtk-aarch64/releases/download/vtk-9.2.6-cp310/vtk-9.2.6.dev0-cp310-cp310-linux_aarch64.whl";\
    fi


RUN pip3 install "pyvista[all,trame]" jupyterlab ipython ipywidgets

WORKDIR /home/project
```

`FROM` declares what the base image we will be using is. `FROM dolfinx/dolfinx:nightly` will create the image from the nightly version of FEniCSx. We could also use `FROM dolfinx/dolfinx:latest` to get the latest stable version of FEniCSx. `ARG TARGETARCH` is a command that stores the target architecture (eithr amd64 or arm64) into a variable we can access.

`RUN` just runs a command in the "terminal" of the image. `WORKDIR` sets the default working directory of the image.

7.2 Building the Image

Once you've modified the docker file to your liking you can build the image by running the following command:

```
> docker build -t "image name" .
```

Here `"image name"` will be the name of the image you create.

You can than use this image as normal to create a container and use it for development.

7.2.1 Building an image for different architectures

By default the image you build is only for the architecture you are currently running on.

However we may want to build one image for arm64 and another for amd64 for exmaple so that both intel and M1 macs and use the image. Doing this is a bit more invovled but not too difficult.

First we must create a custom builder that can cross compile the images. We can do this by running the following command:

```
> docker buildx create --name mybuilder --bootstrap --use
```

We can then run the `docker build ls` command to see the builders. It should output something like this:

```
docker buildx ls
NAME/NODE     DRIVER/ENDPOINT               STATUS   BUILDKIT PLATFORMS
mybuilder *   docker-container
  mybuilder0  unix:///var/run/docker.sock   running  v0.12.1  linux/amd64, linux/amd64/v2,
    linux/amd64/v3, linux/arm64, linux/riscv64, linux/ppc64le, linux/s390x, linux/386, linux
    /mips64le, linux/mips64, linux/arm/v7, linux/arm/v6
default       docker
  default     default                       running  v0.12.3  linux/amd64, linux/arm64, linux
    /arm/v7, linux/arm/v6
```

We can then build the multiple images like so:

```
docker buildx build --platform linux/amd64,linux/arm64 -t <username>/<image>:latest --push .
```

This will build the image for both amd64 and arm64 and then push it to docker hub. We will talk more about pushing to docker hub in the next section.

7.3 Pushing to Docker Hub

Once we've created the image, and tested we may want to share it with others.

We can do this by pushing it to docker hub. Here is the docker hub for the image we've used in this example. Opening the page you can see the different tags that we have availble.

As a part of pushing to dockerhub you must create an account.

After you've created an account and logged in through the docker dashboard, you may use the docker push command. To do so first we must tag the images. Normally images are named according to the following format:

`<username>/<imagename>:<tag>` The tag can be the verison of the image, or any other tag that describes the difference.

We can tag our image in two ways, during building or aftewards.

7.3.1 Tagging during building

During building we can tag our image with the `-t` flag. We can actually tag the same image with multiple flags.

For example we can run the following command:

```
docker build  -t jorgenin/dolfinx-2.077:v0.8 -t jorgenin/dolfinx-2.077:latest -t jorgenin/
  dolfinx-2.077:v0.8.1 -t jorgenin/dolfinx-2.077:nightly .
```

And we will actually tag the image with `v0.8`, `latest`, `v0.8.1`, and `nightly`.

7.3.2 Tagging after building

We can tag the image after building by running:

```
docker tag <image id> <username>/<imagename>:<tag>
```

We can do this multiple times to add multiple tags for example:

```
docker tag dolfinx jorgenin/dolfinx-2.077:0.8
docker tag dolfinx jorgenin/dolfinx-2.077:latest
docker tag dolfinx jorgenin/dolfinx-2.077:nightly
docker tag dolfinx jorgenin/dolfinx-2.077:0.8.1
```

7.3.3 Pushing to Docker Hub

Once we've tagged the image we can push the images to docker hub. We can push as single image by running

```
docker push <username>/<imagename>:<tag>
```

Or we can push all of the tags by running:

```
docker push -a <username>/<imagename>
```

We could have also pushed during building by running:

```
docker build -t <username>/<imagename>:<tag> --push .
```

7.4 Example of full command

Here is an example of the full command that was used for building the `jorgenin/dolfinx-2.077` images.

```
docker buildx build --platform linux/amd64,linux/arm64 -t jorgenin/dolfinx-2.077:v0.8 -t
  jorgenin/dolfinx-2.077:latest -t jorgenin/dolfinx-2.077:v0.8.1 -t jorgenin/dolfinx
  -2.077:nightly . --push
```

This command builds multiple tags of the image for both amd64 and arm64 and then pushes them to dockerhub.