Introduction to Docker and Docker Hub

Andrew J. Stewart

E: drandrewjstewart@gmail.com
T: @ajstewart_lang
G: ajstewartlang

Open and Reproducible Research

Shared Data

We already know this is important for reproducibility.

Shared Code

We already know this is important for reproducibility.

Shared Computational Environment

Why is this important and how do we do it?

Based on https://ropenscilabs.github.io/r-docker-tutorial/

Why do we need to reproduce the computational environment?

Analysis code can ‘break’ - often in one of two ways:

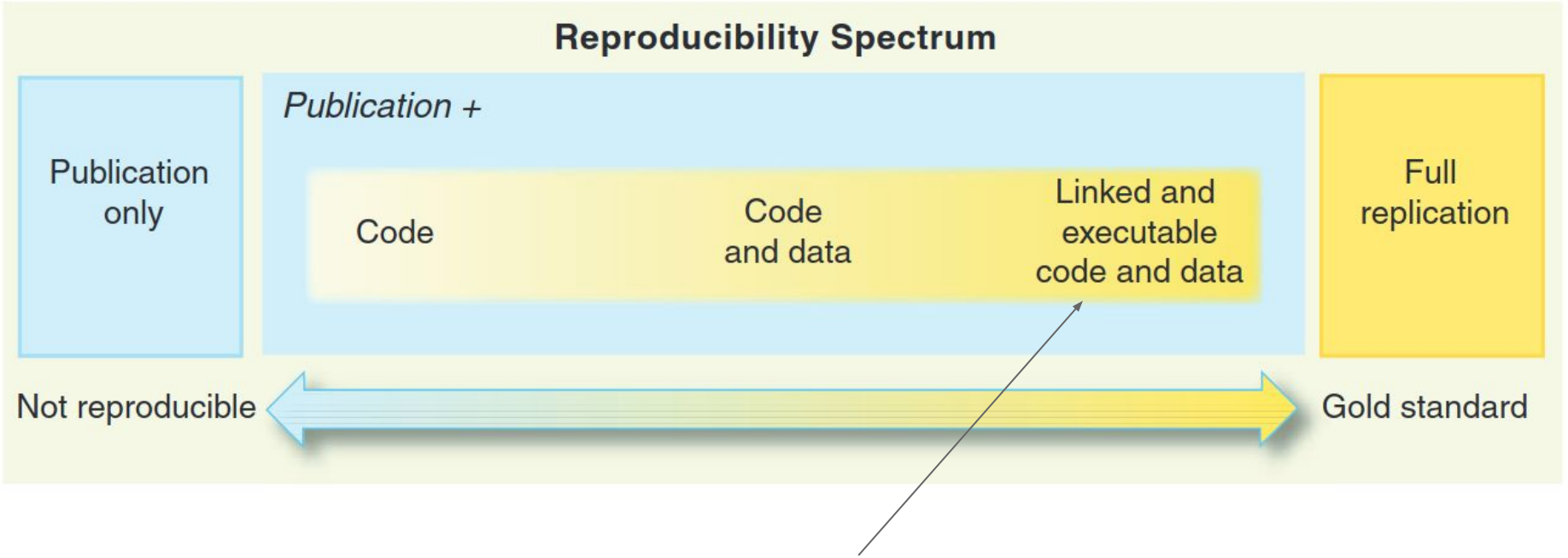
Code that worked previously now doesn’t - maybe a function in an R package was updated (e.g., `lsmeans()` became `emmeans()` so old code using the `lsmeans()` function wouldn’t now run).

Code that worked previously still works - but produces a slightly different result or now throws a warning where it didn’t previously (e.g., convergence/singular fit warnings in `lme4::` version 1.1-19 vs. version 1.1-20).

Why do we need to reproduce the computational environment?

There have even been changes in the way that base R works - in R 3.6 the way in which `sample()` worked differed from how it had worked previously, and in R 4.0, when reading in data `stringsAsFactors = FALSE` (whereas previously `stringsAsFactors = TRUE`).

You need to capture the versions of the different R packages (plus their dependencies) and even the version of R you used in the original analysis.



How do we do this bit? How do we share our code, data, and computational environment with others? Essentially, how do we share a copy of our computer with others?

Introduction to Docker

Docker has been around since 2013 and packages your data, code and all its dependencies in the form called a docker container to ensure that your application works seamlessly in any environment.

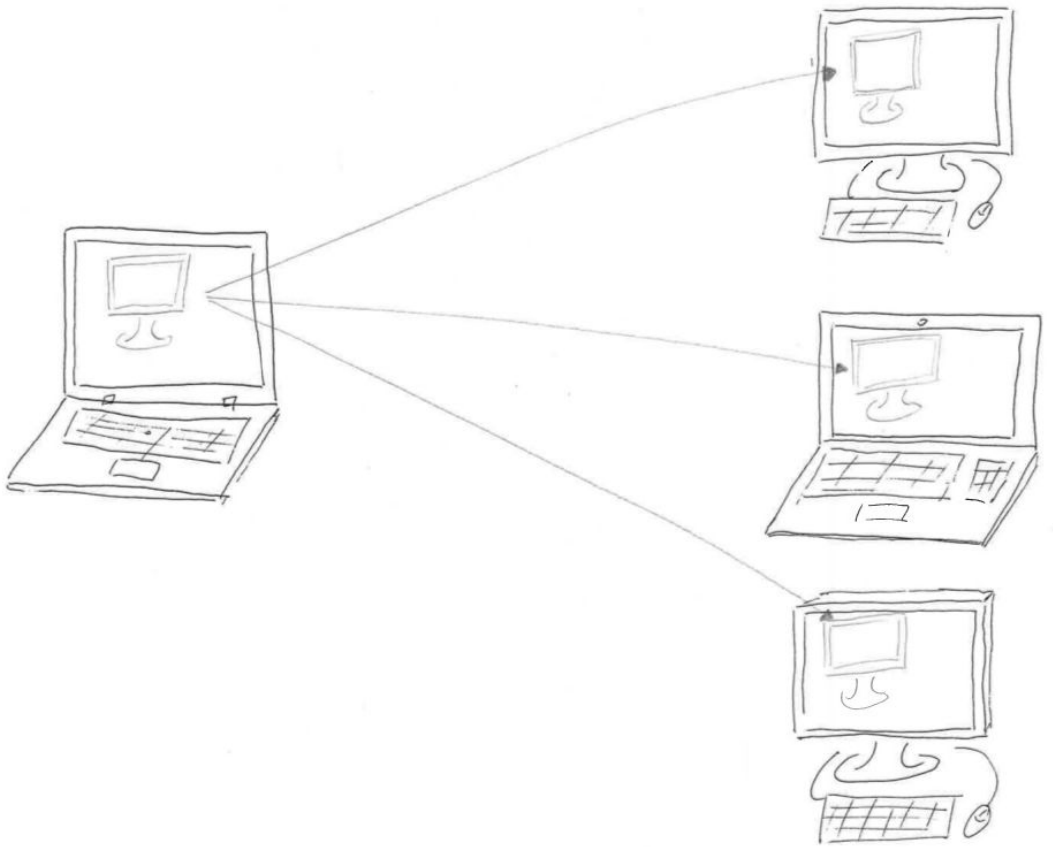
When you run a docker container it's like running your analysis on a computer that has the same configuration as our own one at the point in time when you built the container.

https://medium.com/the-andela-way/docker-for-beginners-61e8e0ce6a19

With Docker, we can build a container that contains our data, code, and computational environment.

We can share this container with others, and we can also run this container on a computer more powerful than our own.

An instance of a Docker image is called a container. You can have many containers running from the same image.



Computerception

Installing Docker

To install Docker on a Mac, download and install Docker Desktop from here:

https://hub.docker.com/editions/community/docker-ce-desktop-mac/

To install Docker on a Windows PC, download and install Docker Desktop from here:

https://hub.docker.com/editions/community/docker-ce-desktop-windows/

After you have downloaded and installed, restart your machine. Make sure Docker is running before moving on...

Verifying your installation

Once you have downloaded and installed Docker Desktop, open a Terminal window and type the following:

```
$ docker --version
```

On you machine, this is what happens:

```
andrew@andrew-Apollo:~/docker$ docker --version
Docker version 19.03.13, build 4484c46d9d
```

Then type the following (note, on Linux you will need to add sudo to the start):

```
$ docker run hello-world
```

If all is working, you should see:

```
andrew@andrew-Apollo:~/docker$ docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:8c5aeeb6a5f3ba4883347d3747a7249f491766ca1caa47e5da5dfcf6b9b717c0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
      (amd64)
 3. The Docker daemon created a new container from that image which runs the
      executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
      to your terminal.
...
```

Checking which images we have locally

```
andrew@andrew-Apollo:~/docker$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              bf756fb1ae65        10 months ago       13.3kB
```

We see we have one image locally - and it’s maybe not the most exciting in the world.

How about we download an image of R and RStudio and run that as a Docker container? That could be more useful...

Launching RStudio via Docker

Let's run R and RStudio with Docker - we are going to use the *verse* image from Rocker - this is a Docker image that contains R, RStudio and a bunch of Tidyverse packages.

```
$ docker run --rm -p 8787:8787 -e PASSWORD=password rocker/verse
```

The `--rm` flag deletes the Docker container when we quit it - not having this means that our local machine could quickly become full of (unused) Docker containers.

The `-p` flag tells Docker to run the container on a particular locao port, allowing us to access it via our web browser.

The PASSWORD sets the password for logging into RStudio in the container (with the username default set to `rstudio`).

```
andrew@andrew-Apollo:~$ sudo docker run --rm -p 8787:8787 rocker/verse
Unable to find image 'rocker/verse:latest' locally
latest: Pulling from rocker/verse
a4a2a29f9ba4: Pull complete
127c9761dcba: Pull complete
d13bf203e905: Pull complete
4039240d2e0b: Pull complete
94d6b215a63e: Pull complete
c33c7553b8e6: Downloading [==================================================> ]  291.7MB/292MB
5912c8cbac6c: Download complete
feab39048769: Download complete
2db9cf02890b: Download complete
d85ffb72eea2: Downloading [======>                                            ]  72.27MB/548.2MB
```

As Docker can’t find the rocker/verse image locally, it will download it from Docker Hub (a hosting site for Docker images).

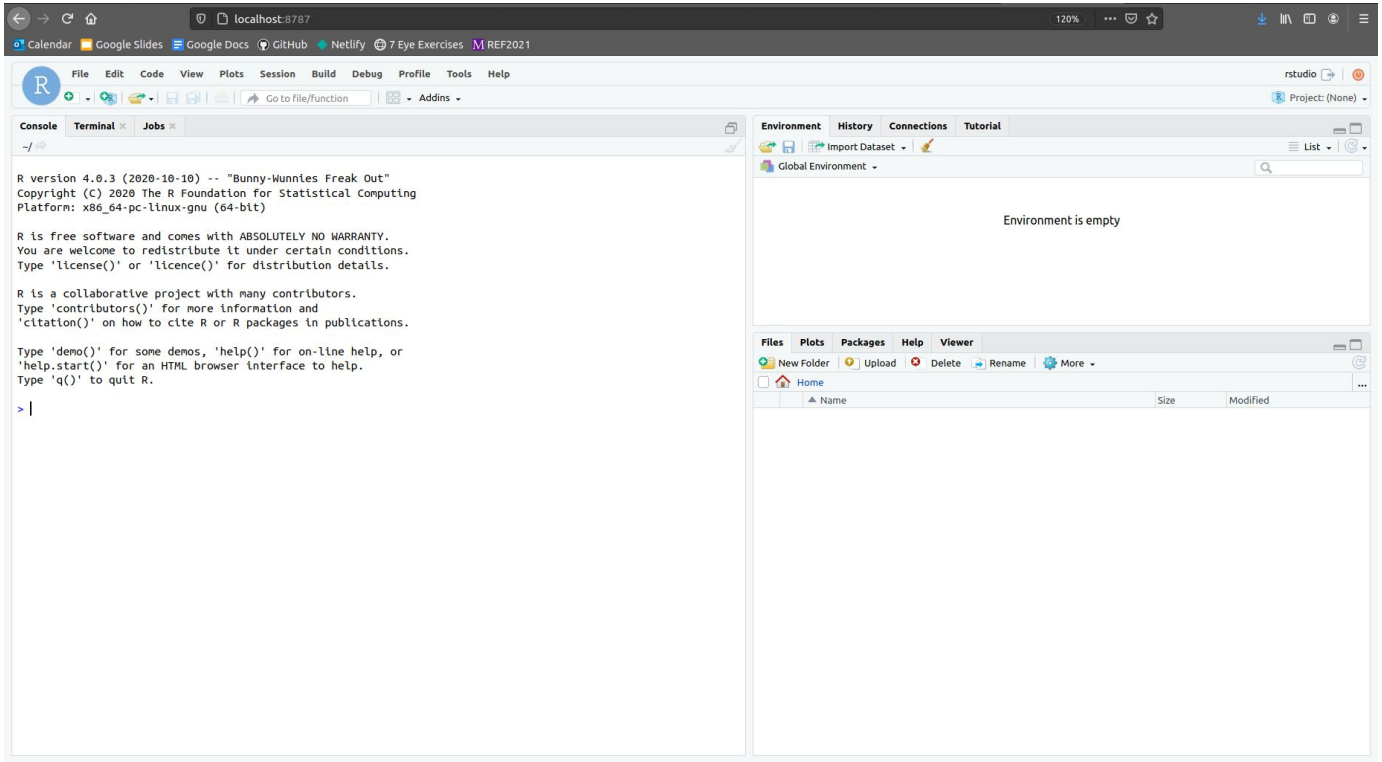
After a little while...

Once the Docker image has been downloaded, open a web browser:

If you are on a PC, type the following into your web browser:
http://192.168.99.100:8787

If you are on a Mac or Linux machine, type the following into your web browser:
http://localhost:8787

And you should see RStudio running in your browser...



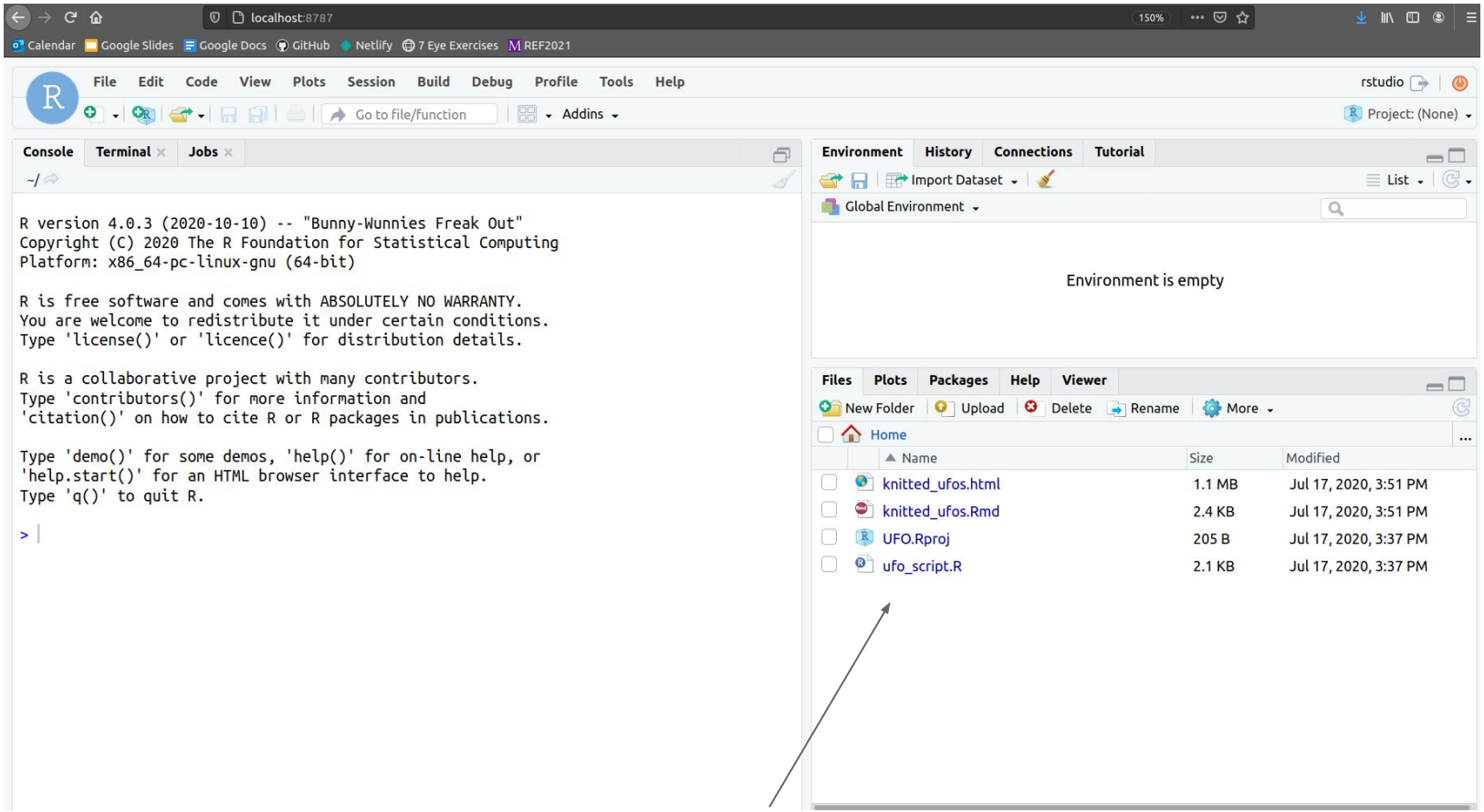
Everything in our container disappears when we shut it down.

If you quit the browser, and shutdown the container using CTRL-C in the Terminal then everything you were doing in the container disappears.

So, what we want to do is link the container with a folder location on our local machine that might have data in it, or might be where we want to save scripts etc. that we create in our Docker container. To do this we need to navigate to the folder we want to be able to access/save to locally.

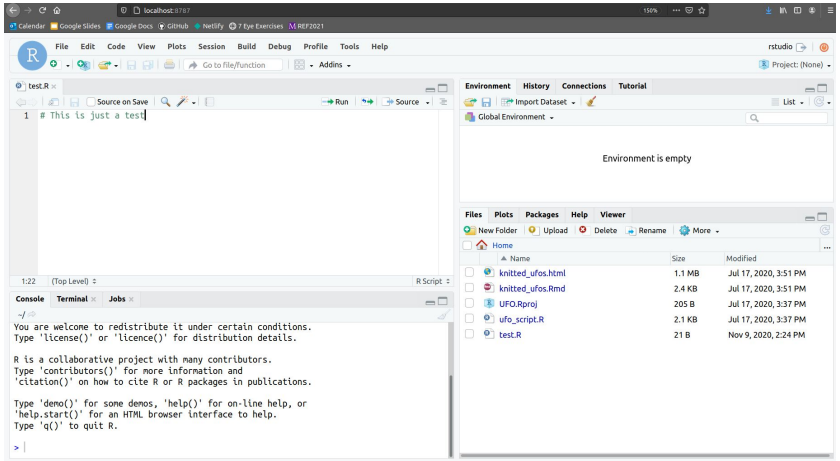
```
$ docker run --rm -p 8787:8787 -v $(pwd):/home/rstudio/ -e PASSWORD=password rocker/verse
```

Then open your web browser as before...



We can now access all the data, code etc. in our local folder.

Writing locally…



Check out the local folder and you'll see the new `test.R` file. This file remains even after you shut down the container…

```
andrew@andrew-Apollo:~/R_Work/UFO
$ ls
knitted_ufos.html  knitted_ufos.Rmd
test.R  UFO.Rproj  ufo_script.R
```

Adding packages to our Docker image

Imagine we want to install the Gapminder package. We'd install it in RStudio as we normally would by typing `install.packages("gapminder")` in the Console window. But if were then to close down the Docker image, we'd lose this package. What we want to do is update our Docker image so it now also includes the gapminder package install.

In another Terminal window (with the container in which you've already installed the Gapminder packagealready running in our browser), type `docker ps:`

```
andrew@andrew-Apollo:~/R_Work/UFO$ docker ps
[sudo] password for andrew:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
28ea8bd297fd        rocker/verse        "/init"             2 minutes ago       Up About a minute   0.0.0.0:8787->8787/tcp   thirsty_cohen
```

You'll see the running container (within which we have installed the gapminder package). Let's use the docker commit command to create new docker image.

Use the container ID associated with what is on your machine - mine is 28ea8bd297fd for the container that's running.

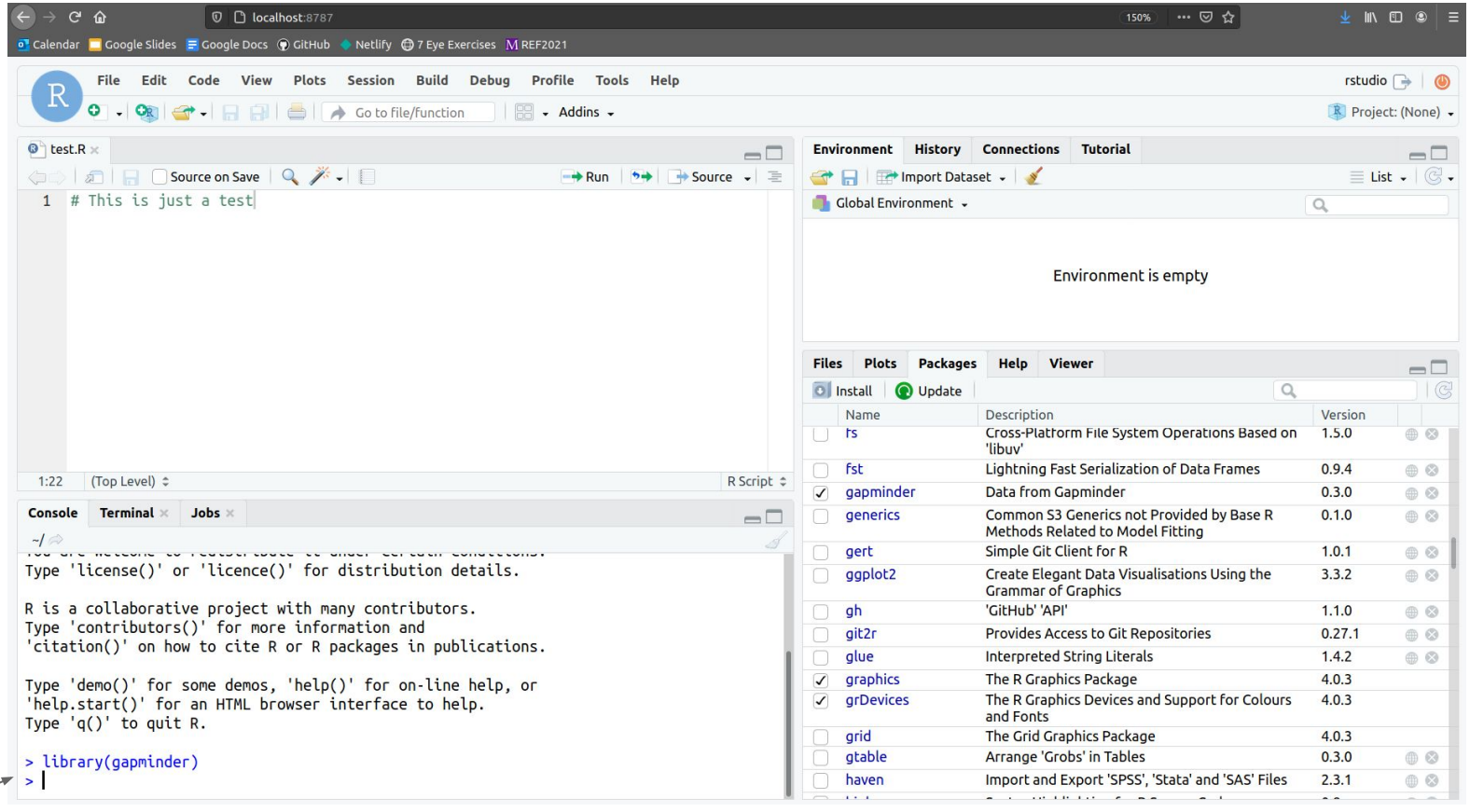
```
$ docker commit -m "verse + gapminder" 28ea8bd297fd verse_gapminder
$ docker images

REPOSITORY        TAG                 IMAGE ID         CREATED           SIZE
verse_gapminder   latest              d005249ae1d3     17 seconds ago    3.64GB
rocker/verse      latest              891a38d55b3e     3 days ago        3.64GB
rocker/rstudio    latest              031fb686eef9     3 days ago        1.91GB
hello-world       latest              bf756fb1ae65     10 months ago     13.3kB
```

By using `docker commit`, we've created a new image from the running container that I've called `verse_gapminder`. The `-m` flag followed by what's in quote is the commit message - a little like with git commit messages.

Now, if we shut down our current image, and launch the verse_gapminder one, we should have an instance that already has the gapminder package installed.

```
$ docker run --rm -p 8787:8787 -v  $(pwd):/home/rstudio/ -e PASSWORD=password verse_gapminder
```



Yes! The gapminder package is already installed so we can just load it using `library()` ...

Dockerfiles for creating our own Docker images

We can create our own Docker image - which might be a combination of a pre-built Docker image (e.g., R 4.0.3) plus an R script we have written and our data files that perhaps we want to share with someone. If they then run this image, they'll be running a Docker container *exactly* the same as the one on our own machine.

The structure of my R Project folder looks like this:

```
|--data
|    |--my_data.csv
|--script
|    |--plot_my_data.R
|--Dockerfile
|--script_data_docker_demo.Rproj
```

And some hidden files like `.Rproj.user`

We can create a file called `.dockerignore` in which we list the files we don't want in our Docker image.

Dockerfiles for creating our own Docker images

We build a new Docker image by creating a Dockerfile. Here's an example Dockerfile created using a simple text editor - note to preserve the folder structure (e.g., for the my_data.csv file) you need to add the folder name on the right of the ADD command:

```
FROM rocker/rstudio:4.0.3
FROM rocker/verse:4.0.3

ADD data/my_data.csv /home/rstudio/data/
ADD script/plot_my_data.R /home/rstudio/script/
ADD script_data_docker_demo.Rproj /home/rstudio/
```

I save the Dockerfile in my R Project folder that contains the folders `data/`, `script/` and the file `script_data_docker_demo.Rproj`

Using a text editor I create a `.dockerignore` text file containing the file names I want Docker to ignore during the build - let's exclude the Dockerfile itself:

```
Dockerfile
```

Building our new Docker image

Let’s build our new Docker image by running the docker build command:

```
$ docker build -t my-r-demo-image .
```

Call the image whatever you like - the dot at the end tells Docker that all the resources to build the image (including the Dockerfile) are in the current directory.

```
andrew@andrew-Apollo:~/R_Work/script_data_docker_demo$ docker build -t my-r-demo-image .
Sending build context to Docker daemon  29.18kB
Step 1/5 : FROM rocker/rstudio:4.0.3
 ---> 031fb686eef9
Step 2/5 : FROM rocker/verse:4.0.3
 ---> 891a38d55b3e
Step 3/5 : ADD data/my_data.csv /home/rstudio/
 ---> Using cache
 ---> 56b7e4742e97
Step 4/5 : ADD script/plot_my_data.R /home/rstudio/
 ---> 450fc910efe8
Step 5/5 : ADD script_data_docker_demo.Rproj /home/rstudio/
 ---> 1a14cb5bfdd4
Successfully built 1a14cb5bfdd4
Successfully tagged my-r-demo-image:latest
```

Running our new Docker image

We can check our new image is available locally by using the command `$ docker images`:

```
andrew@andrew-Apollo:~/R_Work/script_data_docker_demo$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED                  SIZE
my-r-demo-image     latest              1a14cb5bfdd4        About a minute ago   3.64GB
verse_gapminder     latest              d005249ae1d3        23 hours ago
3.64GB
rocker/verse        4.0.3               891a38d55b3e        4 days ago
3.64GB
rocker/verse        latest              891a38d55b3e        4 days ago           3.64GB
rocker/rstudio      4.0.3               031fb686eef9        4 days ago
1.91GB
rocker/rstudio      latest              031fb686eef9        4 days ago           1.91GB
hello-world         latest              bf756fb1ae65        10 months ago
13.3kB
```

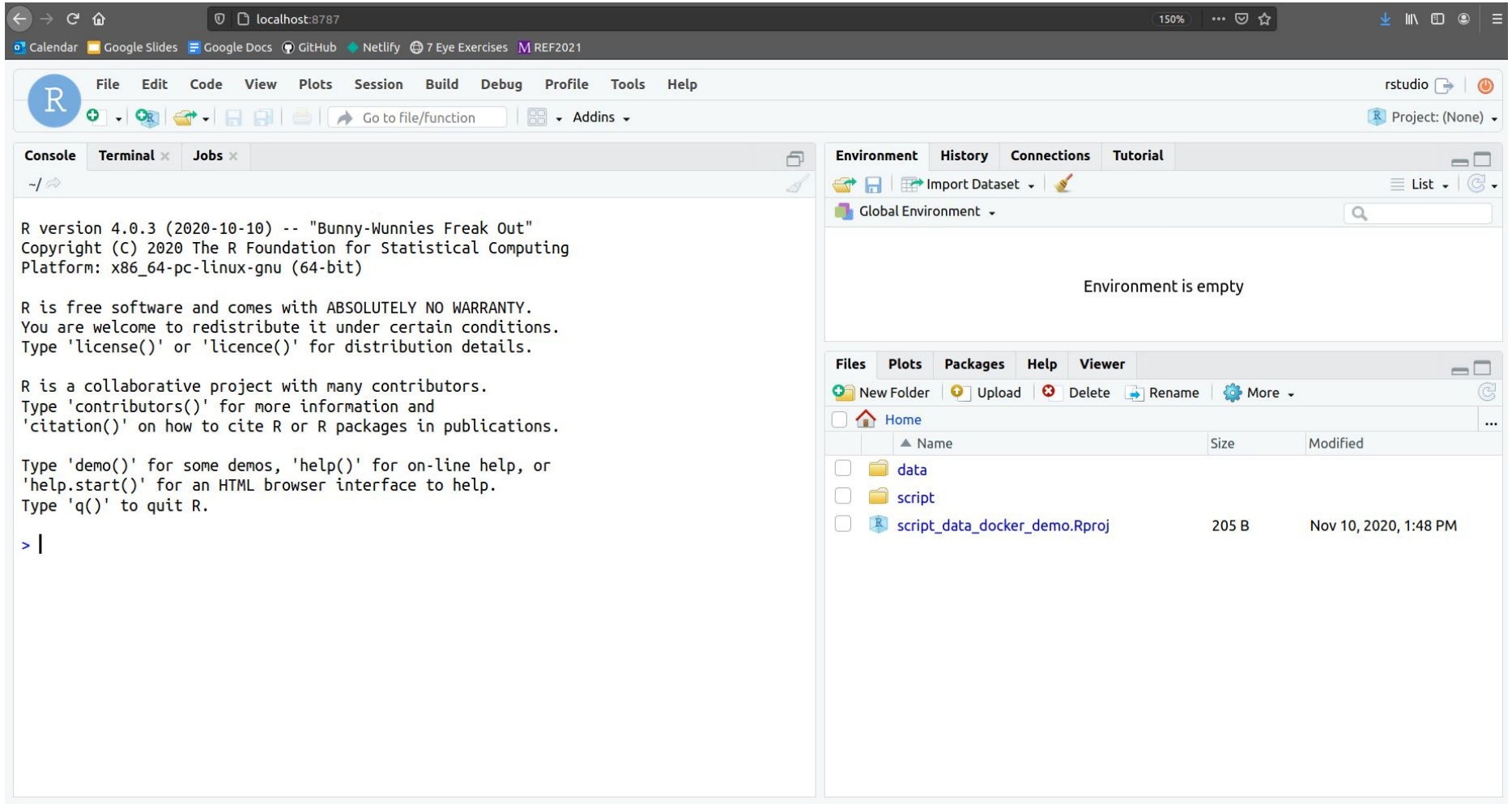
And we can see it’s there...

Running our new Docker image

We can now run the image as we have run images previously:

```
$ docker run --rm -p 8787:8787 -v /home/rstudio/ -e PASSWORD=password my-r-demo-image
```

Then open the browser window as you've done previously...

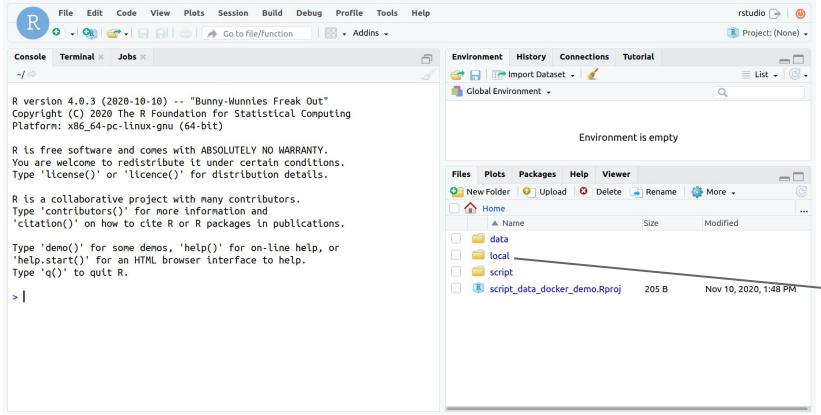

localhost:8787
150%
Calendar
Google Slides
Google Docs
GitHub
Netlify
7 Eye Exercises
REF2021
File
Edit
Code
View
Plots
Session
Build
Debug
Profile
Tools
Help
rstudio
Go to file/function
Addins
Project: (None)
Console
Terminal
Jobs
~/
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment
History
Connections
Tutorial
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
New Folder
Upload
Delete
Rename
More
Home
Name
Size
Modified
data
script
script_data_docker_demo.Rproj
205 B
Nov 10, 2020, 1:48 PM

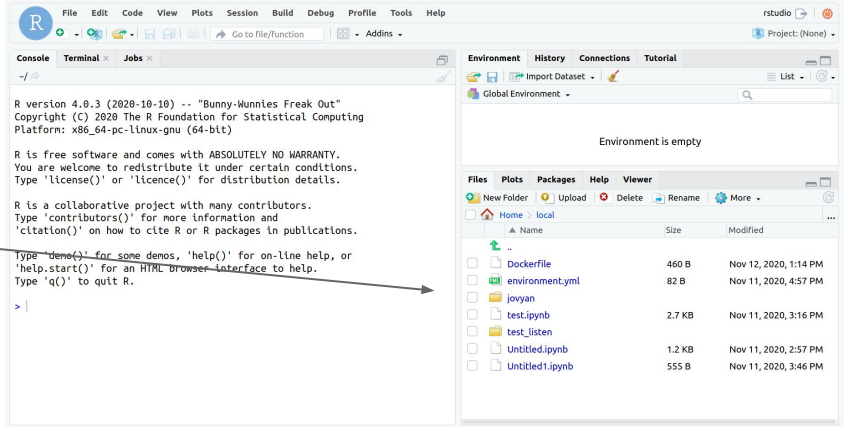
Accessing local files plus the ones in the container...

If you use docker run as follows:

```
$  docker run --rm -p 8787:8787 -v $(pwd):/home/rstudio/local/ -e PASSWORD=password my-r-demo-image
```

You can then access your local working directory as well as any files that are in the Docker container.





Sharing our image with others

We need to put our image on Docker Hub so others can download it.

Go to Docker Hub:

https://hub.docker.com/

And create a free account.

Now, in the Terminal log in to Docker Hub using:

```
$ docker login --username=yourhubusername
```

Enter your Docker Hub password where prompted.

Use the `docker images c` ommand to see what images you have locally:

```
andrew@andrew-Apollo:~/R_Work/script_data_docker_demo$ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
my-r-demo-image       latest              6996ff92a2c3        18 minutes ago      3.64GB
verse_gapminder       latest              d005249ae1d3        24 hours ago        3.64GB
rocker/verse          4.0.3               891a38d55b3e        4 days ago          3.64GB
rocker/verse          latest              891a38d55b3e        4 days ago          3.64GB
rocker/rstudio        4.0.3               031fb686eef9        4 days ago          1.91GB
rocker/rstudio        latest              031fb686eef9        4 days ago          1.91GB
hello-world           latest              bf756fb1ae65        10 months ago       13.3kB
```

We can see that the IMAGE ID of the image we just built (`my-r-demo-image`) is `6996ff92a2c3`

Let's tag our image ready for pushing to Docker Hub:

```
$ docker tag 6996ff92a2c3 ajstewartdata/my-image-to-share:firsttry
```

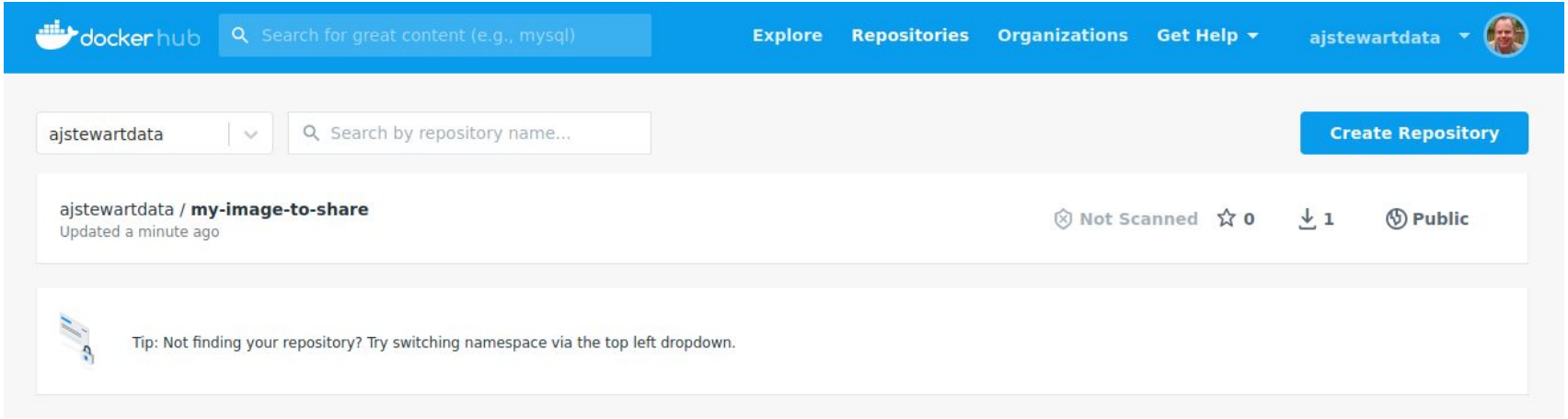
Where `ajstewartdata` is my Docker Hub username, `my-image-to-share` is the name I want the image to have locally and on Docker Hub, and `firsttry` is the tag I want on it (I may end up having more than one version of the same image name and the tag should indicate which version is which). Then use docker push:

```
$ docker push ajstewartdata/my-image-to-share
```

When you push to Docker Hub, things will look like this:

```
andrew@andrew-Apollo:~/R_Work/script_data_docker_demo$ docker push
ajstewartdata/my-image-to-share
The push refers to repository [docker.io/ajstewartdata/my-image-to-share]
5939e7708621: Pushed
341ce97a9d76: Pushed
9ab07b54a958: Pushed
53d09736d4fc: Mounted from rocker/verse
c4e589c9fa3b: Mounted from rocker/verse
7709ccc33c1e: Mounted from rocker/rstudio
c999d4926966: Mounted from rocker/rstudio
5dda9a3ad0c3: Mounted from rocker/rstudio
87dde4dc1f78: Mounted from rocker/rstudio
05f3b67ed530: Mounted from rocker/rstudio
ec1817c93e7c: Mounted from rocker/rstudio
9e97312b63ff: Mounted from rocker/rstudio
e1c75a5e0bfa: Mounted from rocker/rstudio
```

And if I go to my account on Docker Hub I see this…



Your turn to download and run my image `ajstewartdata/my-image-to-share`using:

```
$ docker pull ajstewartdata/my-image-to-share:firsttry
$ docker run --rm -p 8787:8787 -v /home/rstudio/ -e PASSWORD=password ajstewartdata/my-image-to-share:firsttry
```

Docker and Python

If you're using Python via conda, you need to effectively create the conda environment that you're using on your local machine inside your Docker container. In addition to writing a Dockerfile, you need to create an environment.yml file.

The environment.yml file contains information about what Python libraries etc you need to have in you container. Here is an example YAML file where we're asking for Python 3.8 plus the flask and numpy libraries.

```
name: myenv
channels:
  - conda-forge
dependencies:
  - python=3.8
  - numpy
```

Dockerfile for conda environments

```
FROM continuumio/miniconda3
FROM jupyter/datascience-notebook

WORKDIR /app

# Create the environment:
COPY environment.yml .
RUN conda env create -f environment.yml

# Make RUN commands use the new environment:
SHELL ["jupyter", "notebook", "--port=8888", "--no-browser", "--ip=0.0.0.0",
"--allow-root"]

# The code to run when container is started:
COPY test.ipynb .
CMD ["jupyter", "notebook", "--port=8888", "--no-browser", "--ip=0.0.0.0",
"--allow-root"]
```

Building our image

Once we have our Dockerfile and environment.yml file written, we can then use `docker build`:

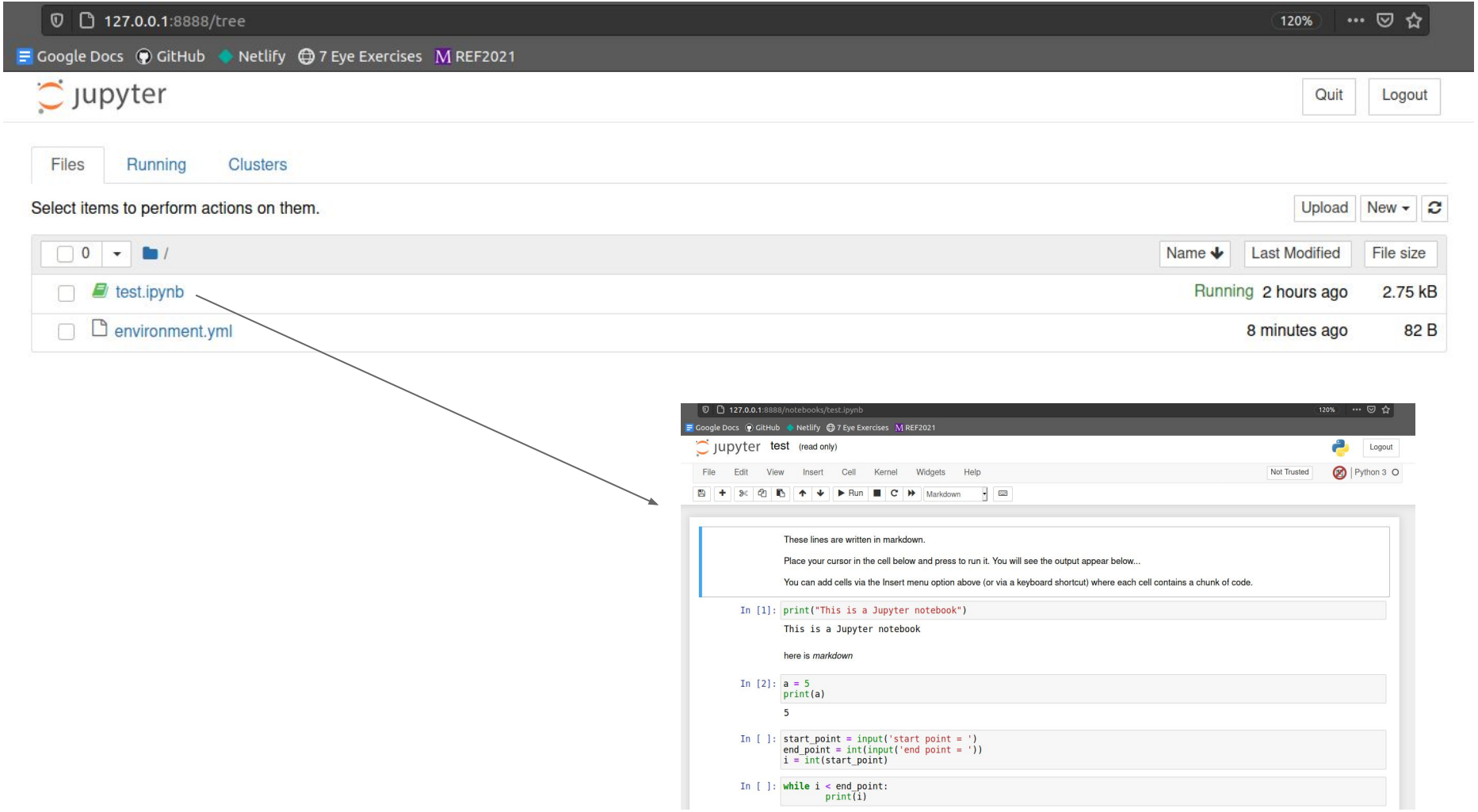
```
$ docker build -t my_python_image .
```

And once the image is built, we can run it as we did with our RStudio images:

```
$ docker run -p 8888:8888 my_python_image
```

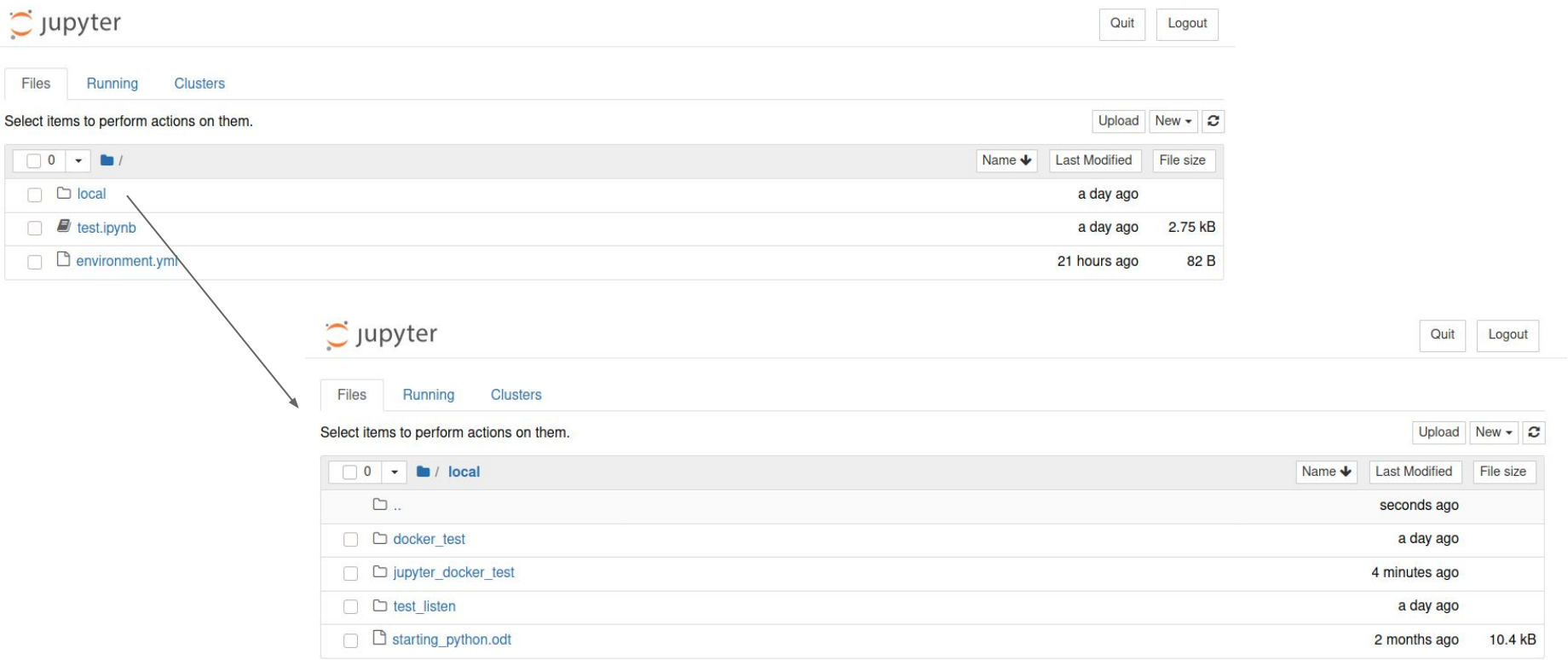
You'll then see in your console something like this - on Mac and Linux machines, copy the last http address and paste it into your browser:

```
To access the notebook, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/nbserver-6-open.html
Or copy and paste one of these URLs:
http://3f0a4fe66ed0:8888/?token=8dfd5c50b99613b2f41b0532040f01bcfd9765d5a766ad7e
or http://127.0.0.1:8888/?token=8dfd5c50b99613b2f41b0532040f01bcfd9765d5a766ad7e
```


127.0.0.1:8888/tree
120%
Google Docs
GitHub
Netlify
7 Eye Exercises
REF2021
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
File size
test.ipynb
Running 2 hours ago
2.75 kB
environment.yml
8 minutes ago
82 B
127.0.0.1:8888/notebooks/test.ipynb
jupyter test (read only)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
These lines are written in markdown.
Place your cursor in the cell below and press to run it. You will see the output appear below...
You can add cells via the Insert menu option above (or via a keyboard shortcut) where each cell contains a chunk of code.
In [1]:
print("This is a Jupyter notebook")
This is a Jupyter notebook
here is markdown
In [2]:
a = 5
print(a)
5
In [ ]:
start_point = input('start point = ')
end_point = int(input('end point = '))
i = int(start_point)
In [ ]:
while i < end_point:
print(i)

As with our RStudio container, if you want to mount your local working directory so you can access it from within the container, you can use:

```
$ docker run -p 8888:8888 -v $(pwd):/app/local test
```



Deleting/Pruning Docker Images

If you are building Docker images, you can sometimes end up with quite a lot of old images that are no longer needed, and images known as ‘dangling images’ that are Docker layers that aren’t connected to any of your tagged images. All these take up disk space. You can remove dangling images with:

```
$ docker system prune
```

and delete old images with:

```
$ docker rmi <IMAGE ID>
```

To get the list of local images plus their IMAGE ID numbers you can type:

```
$ docker images
```

Summary

- Containerisation is a key component of being able to create and share a fully reproducible analysis.
- Docker has become the “go to” way of building images and running containers.
- In an ideal world, all research reports should be using containerisation as a way of “capturing” the full analytical workflow.
- In this workshop we’ve covered building images and running R and Python containers for analysis, linking those with a local directory, and updating pre-existing images to incorporate additions you might have made using the `docker commit` command.