Docker - Beginner Biologist 4

Jean-Yves Sgro

2019 - (last updated: 2019-12-19)

1 Learning objectives

- Explore graphical options in specific docker images
- Web-based options
- X11 options

In class these exercises will be run onto the classroom iMacs.

However, as best as I can I'll provide Windows hints and instructions when possible, but a basic understanding of line-command under Windows would be more than useful for that (*e.g.* know

what is **DOS** for example. See APPENDIX C.)

1.1 Requirements

- Be familiar with Docker or follow earlier workshops: “Docker - Beginner Biologist” workshops 1,2,3.
- Docker will be used from a line-command terminal: **`Terminal`** on a Macintosh in the classroom. A rudimentary knowledge of `bash` command-line is necessary.
- If you are a Windows user: `PowerShell` can be used as a Terminal. However, setting Docker to run on Windows is more involved (not covered in class.)
- **Docker username**: downloads will require a (free) username, therefore registration is necessary in order to follow the tutorial. Go to https://hub.docker.com (https://hub.docker.com) and use the button “Sign up for Docker Hub” to register.

2 Set-up

Tutorials will be held in the Biochemistry classroom 201, and Docker has already be installed.

Instruction for installation can be found on the install link[1] of the Docker web site.

> *Note* HTML Version only:

If you are following this document in **HTML format** the code is shown with a colored background:

```
Green background: commands from local computer bash terminal
```

```
Ligh plum background: commands from ALTERNATE local computer bash
  terminal
```

```
White background: standard output of programs.
```

```
Blue background: commands and output when WITHIN a bash container
```

```
Yellow background: commands or output for information. Do not run!
```

2.1 Shared directory

We'll use a shared directory called `dockershare` to share and save files.

Use `mkdir $HOME/dockershare`, see also Appendix A.

2.2 Getting started

To get started we need to open a text terminal as detailed below. In class we'll use a Macintosh.

TASK:

Do one of the following:.

If you are on a Macintosh:

1. Find the `Terminal` icon in the `/Applications/Utilities` directory. Then double-click on the icon and `Terminal` will open.
2. **OR** use the top-right icon that looks like a magnifying glass (*Spotlight Search*,) start typing the word `Terminal` and press return. `Terminal` will open.

If you are on a PC:

1. Find `Power Shell` *e.g.* using Windows search or Cortana. This will open a suitable text-based terminal.

(*Note*: Windows `cmd` does not offer the appropriate commands.)

2.3 Version check

This ensures that Docker is properly installed. The exact running version itself is not very important.

At the `$` or `>` prompt within the window of `Terminal` or `PowerShell` type `docker --version` to check the version currently installed.

```
docker --version
```

```
Docker version 19.03.5, build 633a0ea
```

2.4 Docker login: Required!

Before going further, it is necessary now to login with your Docker Hub ID. You should already have created one before this or the previous workshop. If you need to create an ID now go to https://hub.docker.com (https://hub.docker.com) to register.

TASK:

Docker login:.

```
docker login
```

```
Login with your Docker ID to push and pull images from Docker Hub.
If you don't have a Docker ID, head over to https://hub.docker.com
to create one.
Username: YOUR_DOCKER_ID_HERE
Password:
Login Succeeded
$
```

Note: if you do not login first you will receive an error message when trying to start docker in the next steps.

3 Choosing a docker image

Today's workshop will be an exploration of docker images that provide a graphical interface, either web-based or X11-based. (The latter might not work easily on Windows.)

We'll explore **web-based** and `X11` examples which are the 2 major but very different graphical interfaces:

Web based:

- `NGINX` web server
- `Dovex` Python tool with web interface to explore datasets
- `R/RStudio` web-based server version for the `R` interface

(For a quick overview see section "webapps with docker" (https://docker-curriculum.com/#webapps-with-docker) online[2].)

`X11` -based:

- `X11` software `IGV` genome viewer
- `X11` utilities

3.1 Web-based images

In casual terms a container is running software on top of a host machine but without direct connection to the host files (unless a shared folder is specified) and without connections to the outside. In other words the container is almost like a box without a lid.

In simple terms, communication in and out the computer is done *via* specific “openings” called ***ports***. For example, the port of an `ssh` connection is `port 22`. The default port for a web connection is `port 80`. A port is usually associated with a “protocol” which would be “Hypertext Transfer Protocol” or `http` for the web.

(For more details see Wikipedia links for “Port (computer networking) (https://en.wikipedia.org/wiki/Port_(computer_networking))”[3] and “well-known port numbers (https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers#Well-known_ports)”[4])

> This is important as we’ll have to know or specify (or both) which port are available for our purpose(s).

Note: The local computer is assigned a special web address which can be written in two equivalent ways:

- **localhost**
- **127.0.0.1**

3.2 X11 images

First of all this option is more “tricky” than the web options and might not work in all circumstances, and even less likely on a Windows computer. Hopefully the examples provided will prove useful.

Citing Wikipedia[5]: The **X Window System**(Scheifler and Gettys (1987)) (**X11**, or simply **X**) is a windowing system for bitmap displays, common on Unix-like operating systems.

4 Web server image

As a first exercise we’ll `pull` a docker image for a simple web server running over the linux implementation `alpine` seen in a previous workshop.

For this simple test we’ll use the official *NGINX* docker image. *NGINX is open source software for web serving […].*[6]

The docker hub page for this image is: https://hub.docker.com/_/nginx (https://hub.docker.com/_/nginx)

The purpose of this container is to run a web server, and therefore it is likely that there will be some version of “entry point” (see previous workshop) that will start the web server as soon as the container is activated. This may be suggested from the information provided by the hub page.

The docker file is available online at https://github.com/nginxinc/docker-nginx/blob/master/stable/alpine-perl/Dockerfile (https://github.com/nginxinc/docker-nginx/blob/master/stable/alpine-perl/Dockerfile) and it ends with the following three statements, two of them will be useful later:

```
EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

The last line with `CMD` defines a default behavior when the container is started without specifying a command to be executed. In this case it would start the *nginx* web server, therefore not allowing shell access *by default*.

> ***Note***: `CMD` can be bypassed simply by adding a command at the end of a `docker run` command *e.g.* adding `/bin/bash` to open a shell.
>
> In contrast an `ENTRYPOINT` command is **mandatory** and can only be bypassed by adding *e.g.* `--entrypoint /bin/bash` in the the `docker run` command.
>
> For more on `CMD` and `ENTRYPOINT` see this blog entry[7]: “Docker RUN *vs* CMD *vs* ENTRYPOINT (https://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/)” - (lso archived at bit.ly/2WPDME2 (https://bit.ly/2WPDME2)

TASK:
pull NGIX image.

```
docker pull nginx
```

We can list images with:

```
docker image ls nginx*
```

```
REPOSITORY    TAG         IMAGE ID        CREATED         SIZE
nginx         latest      540a289bab6c    2 weeks ago     126MB
```

4.1 Docker run commands

In this section we'll try a few versions for the `docker run` command to learn about opening and finding *ports* and other useful information.

There are many options that can be listed *via* command: `docker run --help`.

4.1.1 Random port mapping

In this first attempt we'll use the `-P` modifier defined by help as:

- `-P, --publish-all  Publish all exposed ports to random ports`

This `-P` option will allow to map the ports, but as it is said in the help the mapping will be to a *random* number. We'll detail below what that really means.

We'll also introduce `--name` to provide a specific name designation of our choosing rather than a random name given to the container. This will help below to have a "standard" command for *e.g.* addressing the container. In other words, the commands will work for every user. The name chosen below is simply `ng1`.

TASK:
Run command.

```
docker run --rm --name ng1 -P nginx
```

```

```

The `$` terminal shell prompt is ***not reappearing***, which means that the container is active. However, this terminal is no longer useable for anything else at the moment *i.e.* we cannot type any more commands, even addressed to the local host. Therefore it is necessary to open a new terminal for any command we want to issue on the local host.

- Mac: use menu cascade `Shell` > `New Window` > `Basic` (or another colored option.)
- Windows: open a new `PowerShell`

The `docker run` command above named the container `ng1` and `-P` exposed the container ports to random port(s) on the host. We can find out what the random port is as shown below.

On the *new terminal* window type the following command using the name of the container:

```
docker port ng1
```

```
80/tcp -> 0.0.0.0:32777
```

(*Note*: We could also see this information with `docker ps` under the `PORTS` column as

shown below.)

In this example we see that `port 80` which we saw earlier is the standard default port number for a web server is mapped to port `32777` which means that `port 80` *of the container* would be connected to port `32777` *on the local host* (the computer you are using.)

Therefore you should be able to see the content of the web site in the container with the following web address with a web browser on your computer:

> **`http://localhost:32777`**

Note: ***Adapt the port number*** to what you will see ***on your own terminal***. The random aspect of `-P` means that the number on the local host might not be always the same. So we'll fix this in the next section.

This is what you'll see:

> Summary: we were able to start a container running a web site and accessing the site from a browser running on the local computer.

We'll see shortly that this can be very useful to run specific web-based software.

From this alternative terminal we can also see information about running containers with:

```
docker ps
```

```
CONTAINER ID     IMAGE    COMMAND                  CREATED
STATUS           PORTS                    NAMES
bdadff9f3c9f    nginx     "nginx -g 'daemon of…"   24 minutes ago
Up 24 minutes    0.0.0.0:32777->80/tcp     ng1
```

The output is long an wraps, so it may be clearer to use the followiong command to only see the right-hand part:

```
docker ps | cut -c 105-150
```

```
PORTS                     NAMES
0.0.0.0:32768->80/tcp     ng1
```

TASK:

Turn off container.

There are 2 ways to turn off this container:

1. “kill” the container process on the original terminal (green background in color version of this document) with **`Ctrl`** + **`C`**
2. “stop” the container with `docker stop ng1`. Since we started the container with `--rm` it will be automatically deleted when it is stopped.

4.1.2 Complete command

With this final, complete command we’ll address useful or necessary points.

1. **Detached terminal**: (`-d`) In the previous tests we had to use a secondary terminal because the prompt was not given back after launching the container. This can be changed and the container can be made to run in the background with the “detach” modifier as detailed in help:
   `-d, --detach    Run container in background and print container ID`.
   `-d` will allow the shell prompt back, in effect placing the container processes in the background.
2. **Name**: (`--name`) as above we’ll specify a name of our choosing to designate the container. Here we’ll call it `ng2`.
3. **Shared folder**: (`-v`) this provides a channel of communication for data exchange between the container and the host file system. The folder `/data` will be created within the container and share all files from the dedicated host folder *e.g.* `$HOME/dockershare`. (See Appendix A.)
4. **Port mapping**: (`-p`) We’ll map the relevant port (`port 80`) to a port number of our own choosing with the `-p` (lower case) option.

From help:

`-p, --publish list  Publish a container's port(s) to the host`.

Typical example mapped port names could be: `8080`, `8787`, `8888`.

TASK:

Start new, detached container with shared folder.

```
docker run -d --name ng2 -p 8080:80 --rm -v $HOME/dockershare:/dat
  a  nginx
```

```
aac92902b04e7819a35ab3abb0a2d485689fa980da37842b125c6bf251753f5d
```

Thanks to the `-d` flad we get the prompt back after the long name of the container is echoed on the screen. We can check the state of the container with the command: `docker ps` or `docker container ls` both providing the same output:

```
docker ps
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS          PORTS                   NAMES
aac92902b04e   nginx     "nginx -g 'daemon of…"   3 minutes ago
Up 3 minutes    0.0.0.0:8080->80/tcp   ng2
```

We can note that the `PORTS` column contains `0.0.0.0:8080->80/tcp` reflecting the port mapping.

We can now test if the web site works after opening a web browser to the local host address:

> **`http://localhost:8080`**

Note that since we defined the port ourselves there is no randomness to this `8080` assignment.

EXERCISE:
Time permitting We can create a simple document and view it in the web browser.

- **Step 1**: Create a simple document in the shared folder. (See Appendix A.) For example *Copy/Paste* the following to create a very simple HTML document (or create a file in a way you know how, or even a plain text file.)

```
cd $HOME/dockershare

cat > mytestfile.html <<- EOF
<h1>My Test</h1>
<p>This is a simple HTML file.</p>
EOF
```

- **Step 2**: We'll need access to a shell within the container. We can use the `exec` command that we learned in a previous workshsop. Since the container is detached we can use the

same Terminal session with the command:

```
docker exec -it ng2  /bin/bash
```

```
root@aac92902b04e:/#
```

Therefore the next commands will be issued *within* the container.

- **Step 3**: We need to know where the web site documents are stored. The nginx documentation is not so clear but the answer for this container is: `/usr/share/nginx/html` . Therefore we need to make our file `mytestfile.html` available at this location. One easy way is to simply copy it there, remembering that within the container the shared folder is called `/data` :

```
cd /usr/share/nginx/html
cp /data/mytestfile.html .
ls -l
```

(*Note*: If you are more familiar with `bash` an alternative would be to use a *symbolic link* (with `ln -s`) to the HTML file or a complete folder. This would be most useful for large size files, data files or folders.)

We can now check if this work within the web browser with the local web address:

> **`http://localhost:8080/mytestfile.html`**

If all went according to plan, you should see this in your browser:

We can now exit the container `bash` session. This will return us to the local host `$` prompt.

```
exit
```

```
$
```

Important Note: Since the container was created as *detached* the `exit` command here only

takes us out of the `exec` session of `bash` initiated thereafter. Therefore *the container is still running in the background*, which is expected and wanted behavior. You can verify that this is true in 2 ways:

1. refresh the local host browser page: if the container was no longer running the page would become “not found” or an error such as “can’t connect to the server” may be shown.
2. run the `docker ps` command and see that the container is still active.

4.1.3 Stop container

TASK:

Stop the container.

Now that we are done with this project we need to stop the container and delete it. Since we started the container with `--rm` when it is stopped it will be automatically removed. (Otherwise the `docker rm` command would be needed.)

```
docker stop ng2
```

```
ng2
```

Note: We can verify that this worked by checking with `docker ps` .

5 Dataset explorer: a Python tool

This next brief exercise is based on `Dovex` a *web based tool to quickly provide an interactive overview and enable quick exploration of datasets* from Melbourne Bioinformatics[8] software collection.

The purpose of `Dovex` is to help inspect and explore *tabular datasets* in the form of summaries but also on a large number of optional graphical *interactive plots*.

`Dovex` has been tested on Python 3 and can be installed on the local computer. However, running or installing Python software should not be difficult, but it is often confusing. Running `Dovex` from a docker container will alleviate any necessary installation of Python on the local computer.

5.1 Docker image

TASK:

pull dovex image. `:latest` is assumed.

```
docker pull supernifty/dovex
```

We can list images with:

```
docker image ls */dovex
```

```
REPOSITORY        TAG      IMAGE ID      CREATED         SIZE
supernifty/dovex  latest   8b7d9debceef  6 months ago    1.24GB
```

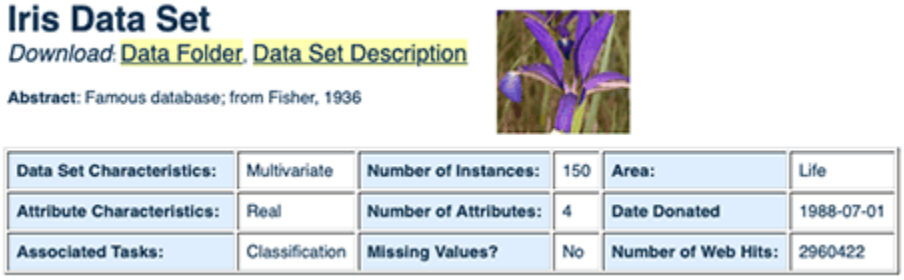
5.2 Datasets

The documentation says that 2 datasets are provided. This is true for the online test version of `Dovex` [9].

The documentation (see below) also states that the 2 datasets are located within `/app/uploads` on the container, and that assumes that the web interface has access to this directory.

5.3 Local Iris dataset

We'll use one of them (*Iris dataset*) that we can download from the Machine Learning archive web page archive.ics.uci.edu/ml/datasets/Iris (http://archive.ics.uci.edu/ml/datasets/Iris) which provides detailed information.



Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	2960422

Typically, the datasets in the archive *separate* "data" and "information," including possible *column headers* for the data file. This is why on this page there are 2 links as shown on the image above. You can read in APPENDIX B how the column headers were added to the data that you can download directly:

```
cd $HOME/dockershare

curl -o iris-names.csv  https://static-bcrf.biochem.wisc.edu/tutor
  ials/docker/iris-names.csv
```

Note: `curl` is similar to `wget` that is not installed by default on Macs.

This is the file that we'll open in the next session.

5.4 Docker run: start container

Installation and running instructions are available on both:

- GitHub: https://github.com/supernifty/dovex (https://github.com/supernifty/dovex)
- Docker Hub: https://hub.docker.com/r/supernifty/dovex (https://hub.docker.com/r/supernifty/dovex)

The information offered in these pages is useful to understand how to launch the `Dovex` program but does not provide a *user manual*, perhaps because the graphical interface for `Dovex` is web-based and rather intuitive.

We already know enough to understand and launch this tool with docker from the information provided. The critical information is in plain text: *By default, the app stores uploaded datasets in the uploads directory*, but also within the suggested `docker run` command:

- port mapping: `5000`
- container shared directory location: `/app/uploads`

We'll add more to the suggested command:

- `--rm` to automatically delete the container when we are done
- `--name` to give a name to our container to easily address it *e.g.* `dovex1`

Mapping shared directories

We can verify the contents of the `/app/uploads` directory with an `entrypoint` command as we learned on previous workshops:

```
docker run  --rm --entrypoint /bin/bash supernifty/dovex -c "ls
  uploads"
```

```
forestfires.csv
iris.data
```

IF we wanted to keep access to this directory for test while at the same time providing access to our own data we should map the `dockershare` directory to a *different directory* within the container. Below we'll use `/app/explore` which will create that directory within the container to contain the shared files. The `uploads` directory will therefore remain available for the built-in test buttons on the web page.

We could also add a detach `-d` command but for now it is useful to see the screen output provided by the container. This means that we will not get the prompt back and we may need to open an alternate terminal later.

TASK:

Start dovex container. `:latest` is assumed.

Based on the details above the following `docker run` command will start the container as we expect. (If you need to create the shared directory `dockershare` see APPENDIX A.)

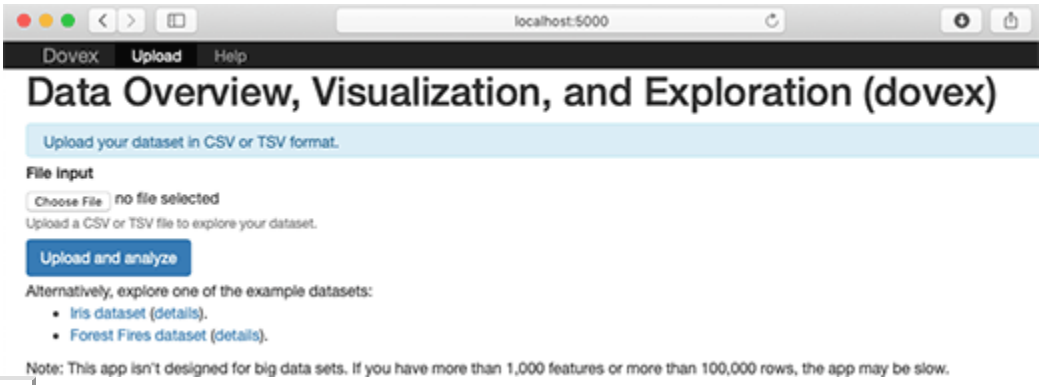
```
docker run --name dovex1 --rm -p 5000:5000 \
-v $HOME/dockershare:/app/explore supernifty/dovex
```

```
 * Serving Flask app "main" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production envi
ronment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 886-406-121
```

To use the application point your browser to one of these equivalent addresses:

> **`http://127.0.0.1:5000/`** or **`http://localhost:5000`**

You should see a web page like this:



TASK:

Click “Iris dataset” link.

As a first approach, below the phrase “Alternatively, explore one of the example datasets:” click on the link **Iris dataset**. This dataset is the one that was included with the docker image within the `uploads` directory.

Note: This will immediately bring a table in the page, while at the same time you may notice that on the Terminal some information of what page(s) have been loaded appear. This is typical *log* data that could be recorded on an actual web site to monitor which pages are most accessed.

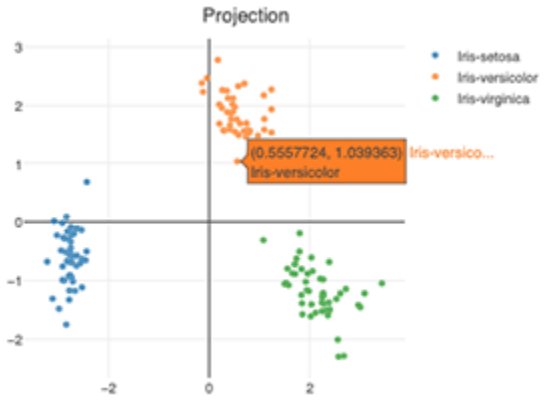
```
172.17.0.1 - - [14/Nov/2019 22:57:57] "GET /explore/iris.data H
TTP/1.1" 200 -
172.17.0.1 - - [14/Nov/2019 22:57:57] "GET /data/iris.data HTTP
/1.1" 200 -
```

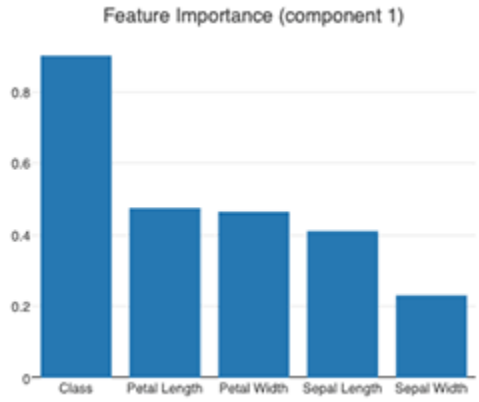
EXERCISE:
Time permitting

There are many useful modes of explorations, we can just look at one of them called **PCA** that helps distinguish “groups” of data. For this, follow these steps:

- Click on button link **Dimensionality Reduction**
- Under *Select a projection algorithm:* select *PCA*
- Under *Label data point with:* select *Class*

- Click on button **Start Analysis**





EXERCISE:
Time permitting another dataset, for example the “other” Iris dataset that we downloaded earlier.

Hint: This dataset column headers are lower case while the dataset provided in the container has an uppercase first letter e.g. “class” *vs* “Class” as a way to distinguish them.

5.5 Exit

Since we did not detach the container process on the terminal it is necessary to follow the instructions to `Press CTRL+C to quit`.

---

6 RStudio server

The docker image used for this example may provide you with a more practical application.

`R` and `RStudio` are heavily used in data analysis. `RStudio` is a useful graphical interface to the software `R`. This exercise demonstrates using `R` wihtin `RStudio` *via* a web interface running in a docker container.

While it is possible to install `R` and `RStudio` natively on your own comptuter it may be useful to run an older version, or, as we’ll see in a future tutorial, create your own docker image with specific `R` options.

With all the experience of the previous section we’ll now follow the process to:

1. `pull` the relevant docker image
2. activate the image with detached mode, shared folder and port mapping.
3. run a small exercise to see that it all works

6.1 Docker image

After this workshop you may want to explore other possible images - as of this writing there are 1,029 entries if searching for “rstudio” on the docker hub. For now we’ll use the image `rocker/rstudio` [10].

TASK:
Pull image.

```
docker pull rocker/rstudio
```

This image may take a bit longer as it is of larger size than what we have tried before.

```
docker image ls  rocker/*
```

```
REPOSITORY          TAG       IMAGE ID        CREATED          SIZE
rocker/rstudio      latest    0dfdbece112b    16 hours ago     1.36
GB
```

The hub page provides very important and critical information:

Quickstart:

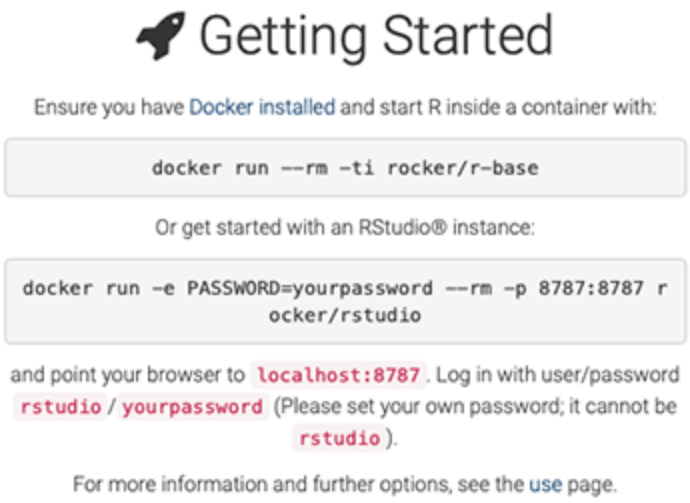
```
docker run --rm -p 8787:8787 -e PASSWORD=yourpasswordhere rocker/rstudio
```

Visit `localhost:8787` in your browser and log in with username rstudio and the password you set. **NB: Setting a password is now REQUIRED**. Container will error otherwise.

Note that all commands documented here work in just the same way with any container derived from `rocker/rstudio`, such as `rocker/tidyverse`.

This is valuable information, that, however, is not always present so clearly on all hub pages. For example, the *tidyverse* version mentioned is on hub page hub.docker.com/r/rocker/tidyverse (https://hub.docker.com/r/rocker/tidyverse) but does not provide the critical `docker run` example, and one would not understand that the image fails because a password is necessary and has to be provided when the container is launched.

One could argue that this version is based on the rstudio version but having information on the page itself is most useful. Some more search on the hub or on the web would lead one to the rocker project web site www.rocker-project.org (https://www.rocker-project.org) where the relevant information is revealed:



In the same way, unfortunately, a large number of entries on the docker hub do not provide all the necessary information for a "casual" user.

Some more information can be gleaned from the *dockerfile*[11] on the hub page.

The last few lines read:

```
EXPOSE 8787

## automatically link a shared volume for kitematic users
VOLUME /home/rstudio/kitematic

CMD ["/init"]
```

The code `EXPOSE 8787` means that *internally* the container will not use the default `port 80` but `port 8787`. This in turn is reflected on the port mapping command.

The `CMD` command suggests that the software within the container will be initiated when the container starts as we experienced in the previous section. Therefore we may want to detach the running container from the terminal with `-d` as we did before.

The information about *kinematic* users provides useful information to be able to locate the data we would want to share from the local host. In the command below we'll map the shared container to `/home/rstudio/data`

6.2 Docker run: start container

The suggested command above will work, but adding the elements we tested in the previous section will make it even easier:

- `-v` for a shared directory, always extremely useful to be able to access your own data. (See Appendix A.)

- `-d` to detach the container.
- `--name` to provide a name of our choosing, *e.g.* `rs1`

The final command is shown below.

TASK:
Start container.

The command is shown below with the *line continuation* symbol `\` to allow writing the command on multiple lines, which is useful for clarity:

```
docker run --rm -d --name rs1 \
-v $HOME/dockershare:/home/rstudio/data \
-p 8787:8787 -e PASSWORD=yourpasswordhere rocker/rstudio
```

Note that `yourpasswordhere` is a valid password but in real life situation a better password would be advised for security.

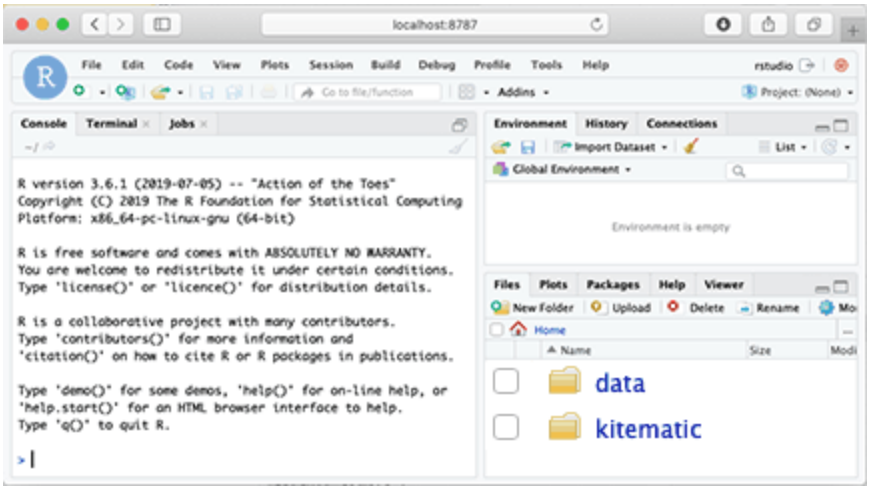
Open a web browser to the local host address with the port number provided:

> **`http://localhost:8787`**

Use `rstudio` as the username and the password used on the line command. Optionally click the "stay signed in" button. Then press return or click on the Sign in button. (See image.)

This will bring a new "R session" within `RStudio`. Note the presence of the `data` directory shown bottom right on the figure below. This is the result of sharing a directory, providing access to file to the session, as well as allowing saving data files from the session.

Note that we could also choose to simply use the `kinematic` directory as the holder of the data we share by changing the `-v` portion of the command to:
`-v $HOME/dockershare:/home/rstudio/kinematic`.



From this point forward using the software would work in the same way as on a native installation.

The data to be shared with the session can be placed within the shared directory. To obtain access to a shell looking within the docker container we could, as before, use the `exec` command and the `exit` command when done.

```
docker exec -it rs1  /bin/bash
```

```
root@aac92902b04e:/#
```

6.3 Verify that it works

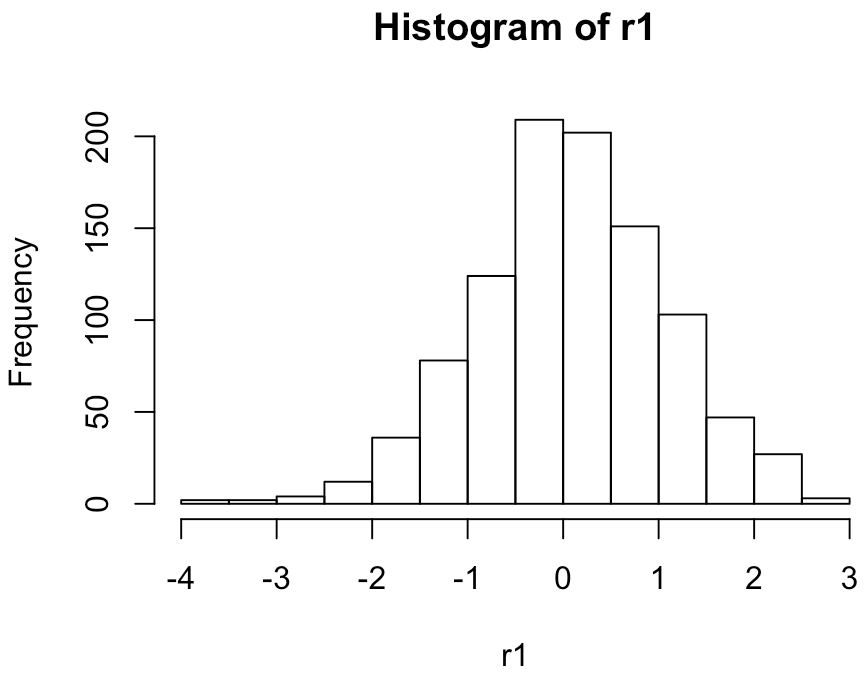
EXERCISE:

Time permitting We can quickly verify that this works. The commands would be `R` commands to be given on the left side console. For example by entering commands such as:

```
# random numbers object:
r1 <- rnorm (1000)
# data spread
summary(r1)
# distribution plot
hist(r1)
```

Results:

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.57977 -0.52039  0.08894  0.08786  0.76438  2.86953
```



The histogram would appear within the “Plots” tab within the `RStudio` bottom right panel.

7 Web interface summary

We have explored useful options for using web-based graphical user interface (GUI) docker images. This is perhaps the easiest GUI.

There are many docker images that take advantage of this GUI. Another example not studied in this workshop could be accessing “Python Jupyter Notebooks” for example the `jupyter/datascience-notebook` [12] which comes with a heavy documentation[13].

In this case there is one more layer of security added, based on “tokens” so that the web page to open would look like:

```
http://127.0.0.1:8888/?token=ea2712974027eb22c78c1ab5dc84c9cf7aa4af4d34e8417d
```

This is beyond the scope of this workshop.

---

8 X11 software

X11 is the graphical interface of the Unix/Linux world. Readily accessible in MacOS as well with the addition of `XQuartz` [14].

In Windows access might be possible *via* additional software such as `xming` [15] or `VcXsrv` [16].

See also https://dev.to/darksmile92/run-gui-app-in-linux-docker-container-on-windows-host-4kde (https://dev.to/darksmile92/run-gui-app-in-linux-docker-container-on-windows-host-4kde)

What is X11 exactly?[17] "X11" is, strictly speaking, a communication protocol. The full name of this software distribution is "the X Window System". Historically, this software distribution was made by MIT; today it is maintained by the X.Org Foundation.[18] The X11 protocol allows applications to create objects such as windows and use basic drawing primitives.

(MIT: Massachusetts Institute of Technology[19])

8.1 A real life example

In one of the classes that I teach I was once confronted with a strange situation involving a genome browser called `IGV` [20] that runs on the `Java` platform. The current `IGV` program was running on `Java` version 8, but for some strange reason he had `Java` version 10 on his Mac, a version that was, strangely, not yet available.

Even more strange he was not able to (easily) remove this "future" version from his Mac.

The solution I found was to create a docker image to solve this problem! (We'll learn in a later workshop how to create your own docker images.) This image is still available on the docker hub on this page: hub.docker.com/r/jysgro/igv (https://hub.docker.com/r/jysgro/igv)

8.2 Pull container

TASK:

Pull container.

This container was uploaded onto the hub with the a `tag` specifying the `IGV` version that it contains, in this case `2.4.11` which was the latest version at the time. Therefore the default

inferred tag `latest` *will not work* and the tag needs to be specified. (*Note*: this tag can be found on the `Tags` tab of the hub page.)

```
docker pull jysgro/igv:2.4.11
```

We can check the list:

```
docker image ls jysgro/*
```

```
REPOSITORY  TAG      IMAGE ID       CREATED         SIZE
jysgro/igv  2.4.11   dd832a2178d1   16 months ago   937MB
```

The large majority of X11 software are graphical in nature and therefore need to be ***displayed somewhere*** - usually the screen of the current, local computer.

However, X11 can also display graphical windows on another computer, and in our case we are interested in displaying the graphical interface onto the local computer, even if the graphical window, in fact, originates *from within the docker* container.

X11 relies on an "environment variable" named `DISPLAY` to know where to send the graphics. Therefore, we'll have to add information about `DISPLAY` on the `docker run` command but also inform the local computer that it should allow this process to go through. (See the details below.)

There are **multiple steps necessary** to the success of running this type of X11-based software within a docker container.

1. X11 software must be able to accept network connections
2. The local computer must be instructed that X11 information should go through. This is done with program `xhost`
3. Add instruction for the docker container when launched with `docker run` .

TASK:

Set-up.

Let's accomplish these steps one by one:

- **Step 1**: For Macintosh `XQuartz` activate the option "*Allow connections from network clients*" in settings (menu: `Xquartz > Preferences...` and click on `Security tab` .)

- **Step 2**: type `xhost + 127.0.0.1` on a local terminal to add "local host" in the authorized list.

```
xhost + 127.0.0.1
```

Note: `xhost + localhost` is an alternative command.

- **Step 3**: we'll use `-e` to specify the `DISPLAY` environment variable (see `docker run --help`)

See also the hub page for more specific details.

With additional folder sharing the command will be, using `\` as the line-continuation:

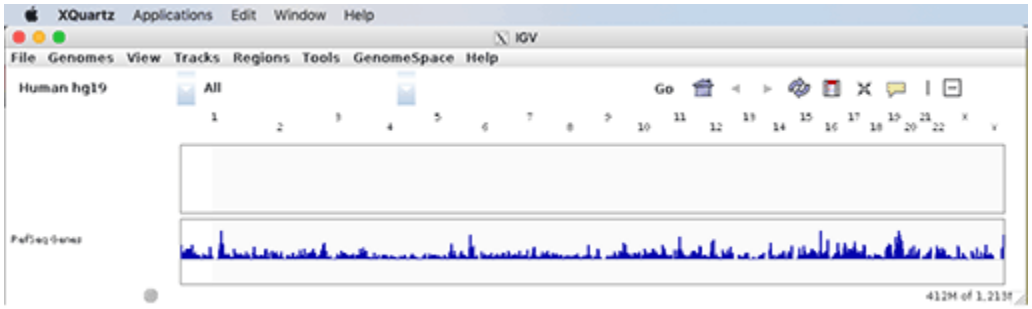
```
docker run -it --rm \
-v /Users/$USER/dockershare:/data \
-e DISPLAY=docker.for.mac.localhost:0 \
jysgro/igv:2.4.11 /bin/sh
```

```
#
```

To start the `IGV` genome viewer issue the following command. (*Note*: the `&` symbol places the software to run in the background therefore releasing the prompt for further commands if desired.)

```
igv &
```

If everything was set well, in spite of expected warnings, a graphcial window based on Human Genome version *hg19* will be shown by default, similar to the image below.



The `IGV` software could then be used to load and inspect genomic files such as `.sam` or `.bam` files derived from Next Gen sequencing mapping of reads onto a genome, or `.vcf` files describing sequence variants.

This is not the purpose of this workshop so we can now exit the program with the **File** > **Exit** menu cascade or simply clicking the red circle of the window (Macintosh.)

Then we need to exit the container to stop it and return to the local host prompt.

```
exit
```

EXERCISE:
Time permitting we can have a little fun with a very small, well know X11 program named `eyes` that draws eyes on the screen that follow the mouse arrow as it moves.

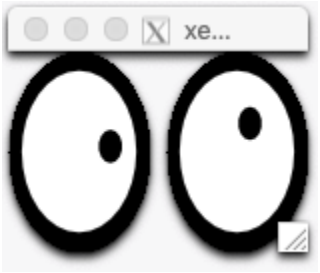
- Pull image (there are multiple options, this one is the most popular)

```
docker pull gns3/xeyes
```

- Start the container, using `-d` to detach it. There is no need to shared a directory. The `xhost` command from above should still be active. If not issue `xhost + 127.0.0.1` again.

```
docker run -it --rm -d  -e DISPLAY=docker.for.mac.localhost:0 gns3
 /xeyes
```

This will launch a window with eyes. The pupil of the eyes will follow the mouse movements:



When closing the window, the docker container will be deleted since we used `--rm` in the command.

- **Restart** the container with an `entrypoint` option:

```
docker run -it --rm  --entrypoint "/bin/bash"  \
-e DISPLAY=docker.for.mac.localhost:0 gns3/xeyes
```

- Then start the `xeyes` program with `&` to place the process in the background.

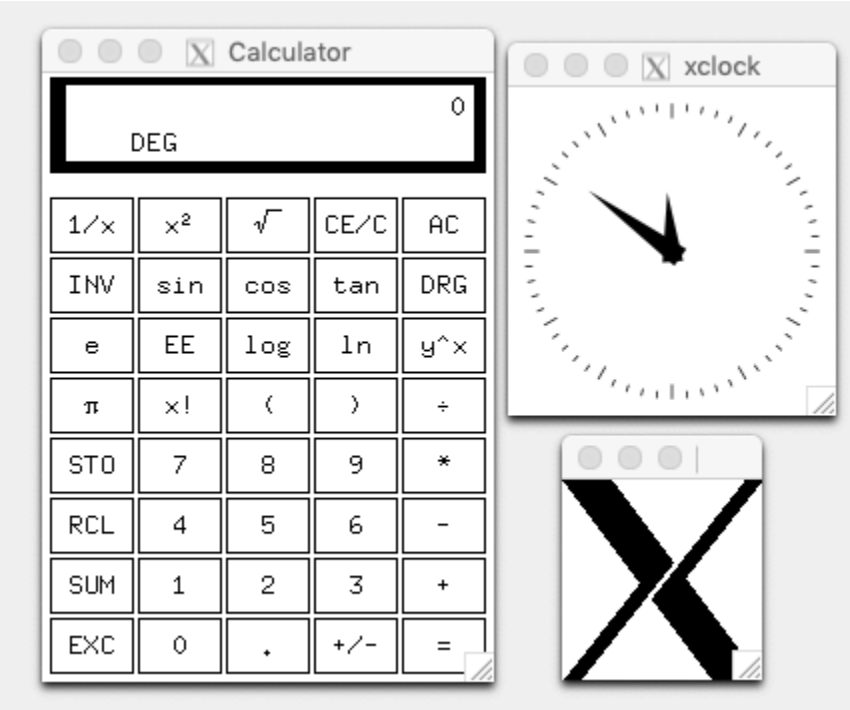
```
xeyes &
```

We can then also launch another useful program called `xclock` that shows time:

```
xclock &
```

The following command will show the X11 logo

```
xlogo &
```



There are more useful *X* programs:

- calculator: `xcalc`
- word processors: `xditview`, `xedit` (best with shared folder)

Finally, exit the container:

```
exit
```

```
$
```

9 EMBOSS interactive

EXERCISE:
Time permitting

We have worked with the EMBOSS programs in a previous workshop. Now that we know how to use `X11` it is possible to use some of the EMBOSS programs that have a graphical output.

that we can display as an `X11` image without the need to save the graphic images into a file as we did before. This allows for faster intereacion.

Below is an exercise inspired by an EMBOSS tutorial[21] (archived 15APR2018[22]) based on a verion of rhodopsin.

The EMBOSS container does not contain the databases, so instead of calling a sequence from a database as in the tutorial (*e.g.* `embl:xlrhodop`) you can download a human version of rhodhopsin in FastA format called `rho_homo_sapiens.fa` either from a link within the tutorial web site or directly as shown below after starting the docker container.

TASK:

Download `rho.fasta` .

The container does not offer the `wget` command so we'd better download the file before starting the container. You could manually download the sequence file ahead of time and place the file within the `dockershare` directory.

On a Mac the command `curl` (Copy URL) does a similar job as `wget` so we can use the command line to do the download as shown, specifying the name of the downladed file with `-o` :

```
cd $HOME/dockershare
curl -o rho_homo_sapiens.fa  https://static-bcrf.biochem.wisc.edu/
  tutorials/docker/rho_homo_sapiens.fa
```

TASK:

Xhost.

It may be necessary to re-run the `xhost` command:

```
xhost + localhost
```

Then verify that `localhost` is now in the list

```
xhost
```

```
access control enabled, only authorized clients can connect
INET6:localhost
INET:localhost
```

TASK:

Start EMBOSS container.

The docker image might still be within the computer account that you are using since we used it in a previous workshop. If it is not the case the image will automatically be pulled from docker hub.

```
docker run -it --rm -v $HOME/dockershare:/data -e DISPLAY=docker.f
  or.mac.localhost:0  pegi3s/emboss
```

We are now ready to use EMBOSS, including with interactive graphics. Some of the commands below are informational, others prepare files for further commands, therefore proceeding in order is recommended.

Exercise 1: check the file content with *e.g.*:

- Show file content on screen with `more` , or `head`
- Use EMBOSS to know sequence length:

```
infoseq -only  -length rho_homo_sapiens.fa
```

Exercise 2: Identify Open Reading Frames (ORF) - graphics output

```
plotorf rho_homo_sapiens.fa
```

Then press return after `[x11]` and a window will open.

```
Plot potential open reading frames in a nucleotide sequence
Graph type [x11]:
```

The longest ORF is on frame 3 (circled in figure above) and is the most likely candidate for a protein translation. It begins at about 100 and ends at about 1200. We'll now use `getorf` to

identify the exact start and end points for our translation.

Note: You need to **close** the image display to get the `#` prompt back on the container.

Exercise 3: Identify exact start and end points for translation

We add `-opt` to see useful optional parameters:

- We'll chose the `standard` genetic code by pressing return or typing `0`
- Since the ORF is largest, we can eliminate small ones by provinding a large expected length, *e.g.* `500`
- For the output option we'll choose `3` and accept the default sequence name ending with `.orf`.

```
getorf -opt rho_homo_sapiens.fa
```

```
Find and extract open reading frames (ORFs)
Genetic codes
         0 : Standard
         1 : Standard (with alternative initiation codons)
         2 : Vertebrate Mitochondrial
// truncated output //
Code to use [0]:
Minimum nucleotide size of ORF to report [30]: 500
Maximum nucleotide size of ORF to report [1000000]:
Type of sequence to output
         0 : Translation of regions between STOP codons
         1 : Translation of regions between START and STOP codons
         2 : Nucleic sequences between STOP codons
         3 : Nucleic sequences between START and STOP codons
         4 : Nucleotides flanking START codons
         5 : Nucleotides flanking initial STOP codons
         6 : Nucleotides flanking ending STOP codons
Type of output [0]: 3
protein output sequence(s) [rho_homo_sapiens.orf]:
```

You can type the content of the file on the screen with `cat rho_homo_sapiens.orf`

Question: Can you find the *begin* and *end* nucleotide numbers of this ORF on the origial `.fa` file?
Hint: `head`

Answers: begin __ __

Answers: end __ __ __ __

Question: Can you identify the START and STOP codon?

Answers: Yes / No: START SEQUENCE __ __ __

Answers:Yes / No: STOP SEQUENCE __ __ __

Optional exercise: use the EMBOSS program `needle` to align the `.orf` sequence to the `.fa` complete sequence. *e.g.* `needle rho_homo_sapiens.fa rho_homo_sapiens.orf` and accepting defaults. This would help to locate the exact location and sequence of the STOP codon. Use `more rho_homo_sapiens.needle` to inspect the file, and `q` to quit viewing.

Answer: STOP SEQUENCE __ __ __

Exercise 4: Translation

The sequence `rho_homo_sapiens.orf` can be translated with the EMBOSS program `transeq`.

```
transeq rho_homo_sapiens.orf
```

```
Translate nucleic acid sequences
protein output sequence(s) [rho_homo_sapiens_1.pep]:
```

Note that `_1` is automatically added to the default output file.

Question: Can you guess where the `_1` comes from?

Answer: _____

Exercise 5: Secondary structure prediction - graphics output

In a previous workshop we used the EMBOSS program `pepinfo`. The difference here is that we'll see the result as a series of **two** an `X11` live-displayed images.

```
pepinfo rho_homo_sapiens_1.pep
```

```
Plot amino acid properties of a protein sequence in parallel.
Graph type [x11]:
Output file [rho_homo_sapiens_1_1.pepinfo]:
```

Note: it is necessary to type `return` or `Enter` to see **two** graphic images.

Note that it is necessary to close the image to regain access to the `#` prompt.

Exercise 6: Predicting transmembrane regions - graphics output

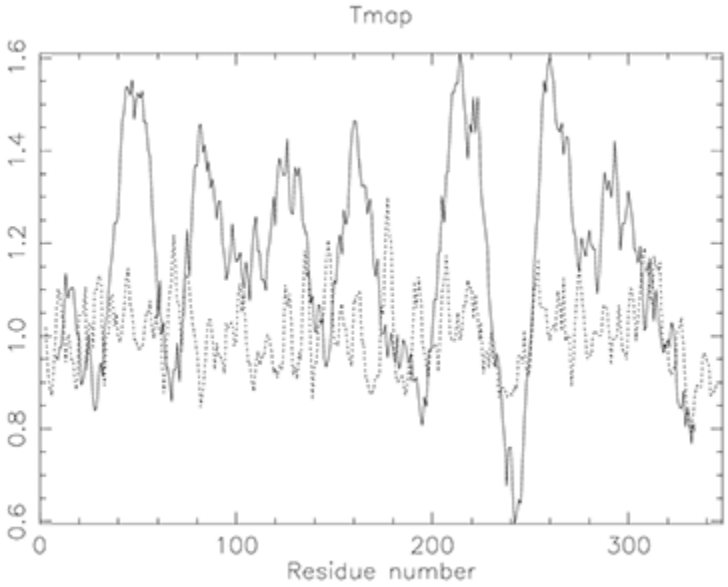
The results from the `pepinfo` hydropathy plot showed seven highly hydrophobic regions

within `rho_homo_sapiens_1.pep` *. Could these be transmembrane domains? We can use the EMBOSS program* `tmap` *to investigate this possibility*:

```
tmap rho_homo_sapiens_1.pep
```

```
Plot amino acid properties of a protein sequence in parallel.
Graph type [x11]:
Output file [rho_homo_sapiens_1_1.pepinfo]:
```

Note that it is necessary to close the image to regain access to the `#` prompt.



The original tutorial image shows thick black bars above the predicted transmembrane regions. This may be an output from an older version of `tmap` as no options were found to add these automatically.

They further explain that: *Taken in combination with the results from* `pepinfo` *, we can see that there may be seven transmembrane helices in this protein*.

Note: You can see the peptide sequence of the transmembrane regions from the secondary output file: with `more rho_homo_sapiens_1.tmap` .

TASK:

Stop container.

Since we started the container with `--rm` once we exit the container will be deleted automatically.

```
exit
```

For more details on all these exercises you can refer to the original tutorial: http://emboss.sourceforge.net/docs/emboss_tutorial/node4.html (http://emboss.sourceforge.net/docs/emboss_tutorial/node4.html)

10 Clean-up

If you have started any container without the `--rm` option it will be in a stopped state.

To list stopped container:

```
docker container ls -a
```

To remove a container use `docker rm` and add the `DOCKER ID` provided in the list.

Note that some containers might need to be stopped before it is allowed to delete them. For this use `docker stop` and add the `DOCKER ID` provided in the list.

11 Summary of commands learned or reviewed

Docker Commands	Comment
`docker --version`	Short output of version
`docker login`	Required. Register at docker.com
`docker pull`	download a docker image from hub.docker.com
`tag`	some docker images require a specific tag
`docker image ls`	list docker image. Equiv command: `docker images`
`docker container ls`	list active containers
`docker ps`	list active containers
`docker port`	print mapped ports
`-P`	`docker run` option to map to random port number
`-p`	`docker run` option to map to specified port number
`-d`	`docker run` option to detach container: background running
`-e`	`docker run` option to specify an environment variable

Docker Commands	Comment
`docker exec`	executes a command on running container
`docker stop`	stop a running container
`--entrypoint`	`docker run` option to bypass default command

Shell Commands / Variables	Comment
`$HOME`	shell variable designated the default home folder
`cd $HOME/dockershare`	change directory to `dockershare` located in `$HOME`
`cat > mytestfile.html <<- EOF`	create a file from *stdin* until EOF
`exit`	terminate a shell session
`DISPLAY`	Environment variable to display graphical interface under `X11`

Software within containers	Comment
`NGINX`	A web server
`Dovex`	web-based quick exploration of datasets
`RStudio`	Server web-based version of RStudio interface to `R`
`IGV`	Java-based “Integrative Genome Viewer”
`X11`	communication protocol for graphics displays in Linux/Unix
`xeyes`, `xclock`, `xlogo`, `xcalc`	`X11` utilities
`infoseq`, `plotorf`, `getorf`, `transeq`, `pepinfo`, `tmap`	`EMBOSS` programs

12 APPENDIX A

12.1 Create shared directory

```
cd $HOME
mkdir dockershare
```

13 APPENDIX B

IRIS DATASET

Typically, the datasets in the archive separate *data* and information, including possible *column headers* for the data file. this is why on this page there are 2 links as shown on the image above:

- *Data Folder* is a link to archive.ics.uci.edu/ml/machine-learning-databases/iris/ (http://archive.ics.uci.edu/ml/machine-learning-databases/iris/) that provides a list of files to download.
- *Data Set Description* is a link to file `iris.names` (see below)

If you click on *Data Folder* a list appears:

- Parent Directory
- Index
- bezdekIris.data
- iris.data
- iris.names

The 2 items of interest for today are: * iris.data and iris.names`.

You can manually download the files, but it is easier to use the *web get* command `wget` to do so as shown below. We'll save the files within the `dockershare` directory (see APPENDIX A.)

```
cd $HOME/dockershare

wget http://archive.ics.uci.edu/ml/machine-learning-databases/iris
  /iris.data

wget http://archive.ics.uci.edu/ml/machine-learning-databases/iris
  /iris.names
```

We can briefly inspect the data files

```
# show first 3 lines:
head -3 iris.data
```

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
```

This indicates that this is a comma-separated format (`csv`)with 5 columns and that indeed there are no column headers. The relevant information is found within the `iris.names` plain text file. You can use a word processor to open it, or simply use `cat iris.names` to print its content onto the screen. The relevant information, also found on the web page is:

```
7. Attribute Information:
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
      -- Iris Setosa
      -- Iris Versicolour
      -- Iris Virginica
```

One way to add the headers could be to open the file in a word processor (or a spreadsheet software) and add the column names (lines marked 1., 2., 3., 4., and 5.)

Alternatively, we can use simple `bash` script commands to extract the relevant information and add it to the data.

The *relevant information* would vary from dataset to dataset and therefore the following commands are specific to *this* version of the Iris dataset.

We can note that the word `class:` followed by colon only appears once in the `iris.name` document. We can use that to “grab” this line as well as the 4 preceeding lines with `fgrep` :

```
fgrep -B4 "class:" iris.names
```

```
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
```

The next step is to transform this input to remove the numbers on the left side and some blank space at the beginning of the line (`cut`), and place the content of each lines onto a single line separated by a comma (`paste`):

```
fgrep -B4 "class:" iris.names | cut -c 7-30 | paste -s -d ',' -
```

```
sepal length in cm,sepal width in cm,petal length in cm,petal widt
h in cm,class:
```

For better clarity we can remove the redundant **in cm** and remove the colon after **class**:

```
fgrep -B4 "class:" iris.names | cut -c 7-30 | paste -s -d ',' - |
  sed -e 's/in cm//g' -e 's/://g'
```

```
sepal length ,sepal width ,petal length ,petal width ,class
```

Finally we can save this output into a new file, `iris-names.csv` and then append the data below the column headers. Rewriting it all with line-continuation `\`:

```
# extract column names into a file
fgrep -B4 "class:" iris.names \
| cut -c 7-30 \
| paste -s -d ',' - \
| sed -e 's/in cm//g' -e 's/://g' > iris-names.csv

# append data:
cat iris.data >> iris-names.csv

#check results with first 3 lines:
head -3 iris-names.csv
```

```
sepal length ,sepal width ,petal length ,petal width ,class
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

To download a copy the final file:

```
wget https://static-bcrf.biochem.wisc.edu/tutorials/docker/iris-na
  mes.csv
```

Note: “sepal length” with a trailing blank space is also unique and we could also use this feature to find the next 4 lines instead: `fgrep -A4 "sepal length " iris.names`.

14 APPENDIX C

Windows users may run into more difficulties depending on set-up and admin privileges. Docker with fancy variable commands will only run in PowerShell.

Here are useful links:

- PowerShell: Environment Variables: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables)
- Get started with Docker for Windows: https://docs.docker.com/docker-for-windows/ (https://docs.docker.com/docker-for-windows/)
- Unable to share drives:
    - https://github.com/docker/for-win/issues/2946 (https://github.com/docker/for-win/issues/2946)
    - https://github.com/docker/for-win/issues/1352 (https://github.com/docker/for-win/issues/1352)

REFERENCES

Scheifler, R. W., and J. Gettys. 1987. “The X Window System.” *ACM Transactions on Graphics* 5 (2): 79–109. https://apps.hci.rwth-aachen.de/borchers-old/cs377a/materials/p79-scheifler.pdf (https://apps.hci.rwth-aachen.de/borchers-old/cs377a/materials/p79-scheifler.pdf).

---

1. https://docs.docker.com/install/ (https://docs.docker.com/install/)↩
2. https://docker-curriculum.com/#webapps-with-docker (https://docker-curriculum.com/#webapps-with-docker)↩
3. https://en.wikipedia.org/wiki/Port_(computer_networking) (https://en.wikipedia.org/wiki/Port_(computer_networking))↩
4. https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers#Well-known_ports (https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers#Well-known_ports)↩
5. https://en.wikipedia.org/wiki/X_Window_System (https://en.wikipedia.org/wiki/X_Window_System)↩
6. https://www.nginx.com/resources/glossary/nginx/ (https://www.nginx.com/resources/glossary/nginx/)↩
7. https://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/ (https://goinbigdata.com

/docker-run-vs-cmd-vs-entrypoint/)↩

8. https://www.melbournebioinformatics.org.au/project/human-genomics/ (https://www.melbournebioinformatics.org.au/project/human-genomics/)↩
9. https://dovex.org (https://dovex.org)↩
10. https://hub.docker.com/r/rocker/rstudio (https://hub.docker.com/r/rocker/rstudio)↩
11. https://hub.docker.com/r/rocker/rstudio/dockerfile (https://hub.docker.com/r/rocker/rstudio/dockerfile)↩
12. https://hub.docker.com/r/jupyter/datascience-notebook/tags (https://hub.docker.com/r/jupyter/datascience-notebook/tags)↩
13. https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html (https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html)↩
14. http://xquartz.macosforge.org/landing/ (http://xquartz.macosforge.org/landing/)↩
15. http://www.straightrunning.com/XmingNotes/ (http://www.straightrunning.com/XmingNotes/)↩
16. https://sourceforge.net/projects/vcxsrv/ (https://sourceforge.net/projects/vcxsrv/)↩
17. https://unix.stackexchange.com/questions/276168/what-is-x11-exactly (https://unix.stackexchange.com/questions/276168/what-is-x11-exactly)↩
18. http://www.x.org/ (http://www.x.org/)↩
19. http://www.mit.edu (http://www.mit.edu)↩
20. http://software.broadinstitute.org/software/igv/ (http://software.broadinstitute.org/software/igv/)↩
21. http://emboss.sourceforge.net/docs/emboss_tutorial/node4.html (http://emboss.sourceforge.net/docs/emboss_tutorial/node4.html)↩
22. https://bit.ly/2KFxUZe (https://bit.ly/2KFxUZe)↩