# AUTONOMOUS DRONES LAB: WORKING WITH DOCKER AND JUPYTER

PATRICK MARCIANO

Last update: 27/02/2020

## CONTENTS

## 1 GENERAL

This tutorial has been put together in order to set you up easily on the lab server. We do not go over all the possibilities of each bash commands, but with more research, one can fully use the powerful functionalities of Docker and Jupyter. For specific assistance and/or improvement of this tutorial, please address your requests to patrickm@mail.tau.ac.il

## 2 DOCKER

### 2.1 Why and what is Docker?

Docker is a software enabling virtualization of environments. As several people are working on the same server, there is a need of isolating each and everyone's libraries, versions and dependencies of software that may differ from one person to another. The common work environment is called a "container" which is based on an "image", those terms will be explained further in this tutorial.

### 2.2 A choice of image

The image is the term given to a source file that can be executed as a container. This image can be manually built from all your requirements on distribution, software and version. An image can also be shared in a public or private repository for others to benefit from it or for deploying your work on another server without the troubles of software version dependencies. A good repository to start with is Docker hub, which hosts many different images and can be of help if one need with certitude certain packages. Link to Docker hub.

### 2.3 Your container

Once an image is pulled into the server, one can start running a container. This container will contain every software and packages that are present in the image. We can regard a container as a "git branch" where the image is the "master branch", i.e, if you are making changes to a running container and want to keep those changes in your image (to run other containers or run your working environment in another server) you will need to commit your container to that image. We can regard a container as an environment where your changes will not affect anyone working on an other container (from the same or different image), the only shared aspect of Docker is the server resources.

### 2.4 More information

If you are interested to learn more in details what is Docker, an image or a container, the following links will be of great help:

- Official Docker Documentation
- Hitesh Choudhary's introduction to Docker

# 3 BASIC WORKFLOW

In this section, we are presenting a step by step guide that can be followed in order to set up your own working environment with docker.

## 3.1 Connecting to the lab server

### 3.1.1 *Establishing the university VPN connection*

In order to access the server, you will need first to establish a secured connection through the university VPN. For that matter, Tel-Aviv University has published this guide.

### 3.1.2 *Login to the lab server*

Once you are securely connected to the VPN, you can login to the lab server through ssh. Open your terminal enter the following command in order to establish ssh connection to the lab server.

```
ssh <user_name>@asl.tau.ac.il
```

If you were not given a personal username and password, use the following:

- Username: yehon2
- Password: yehon123

## 3.2 Using Docker

In this section we will use the official Python image from Docker hub on the lab server running on Linux Ubuntu 18.04.1 LTS.

### 3.2.1 *Image installation*

We are visiting Docker hub website and looking for the official Python image.
From there we can find all the versions available.

```
docker pull python:<version>
```

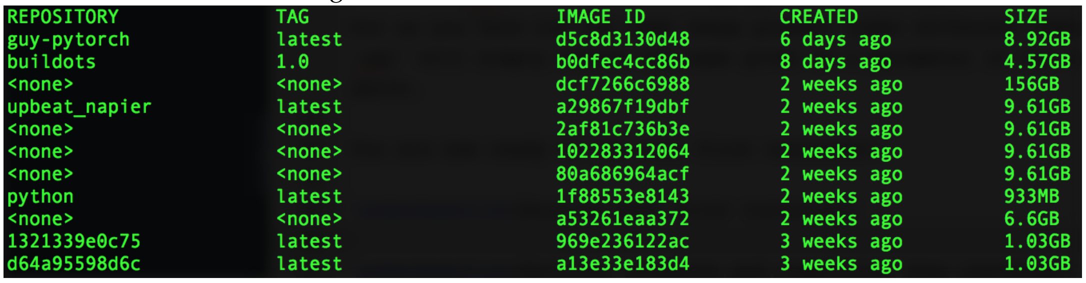
Simply replace <version> with one of the available version found. If you want the latest available version, simply remove the :<version> as follow:

```
docker pull python
```

Once done you should be able to find your image with the following command:

```
docker images
```

You should see a list of images available as follow:



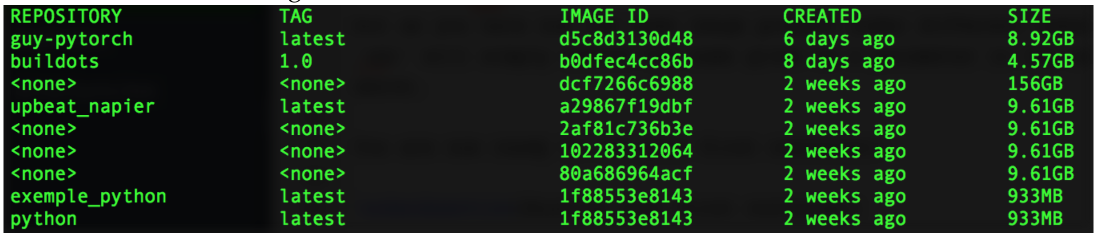
```
REPOSITORY          TAG         IMAGE ID        CREATED         SIZE
guy-pytorch         latest      d5c8d3130d48    6 days ago      8.92GB
buildots            1.0         b0dfec4cc86b    8 days ago      4.57GB
<none>              <none>      dcf7266c6988    2 weeks ago     156GB
upbeat_napier       latest      a29867f19dbf    2 weeks ago     9.61GB
<none>              <none>      2af81c736b3e    2 weeks ago     9.61GB
<none>              <none>      102283312064    2 weeks ago     9.61GB
<none>              <none>      80a686964acf    2 weeks ago     9.61GB
python              latest      1f88553e8143    2 weeks ago     933MB
<none>              <none>      a53261eaa372    2 weeks ago     6.6GB
1321339e0c75        latest      969e236122ac    3 weeks ago     1.03GB
d64a95598d6c        latest      a13e33e183d4    3 weeks ago     1.03GB
```

Notice that the created field only represents when this image was created, in this case, the Python image was updated on Docker hub 2 weeks ago.

Now it is time to give your image a name. In order to recognize the container ownership, it is important to name it after your name, this will avoid confusion and help us assisting anyone who encounters issues. For the sake of this tutorial, let us call it 'example python'.

```
docker tag python exemple_python
docker images
```

We can now see the image with our new name:

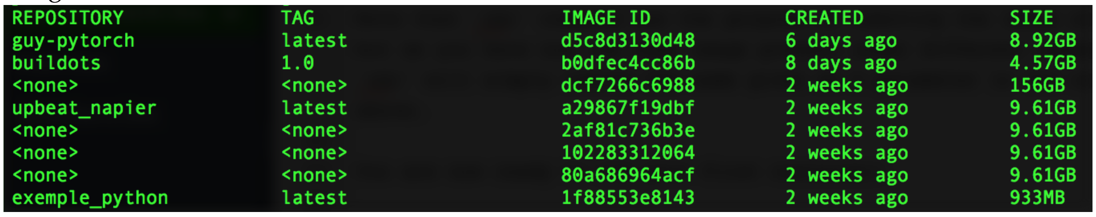


```
REPOSITORY          TAG                  IMAGE ID        CREATED          SIZE
guy-pytorch         latest               d5c8d3130d48    6 days ago       8.92GB
buildots            1.0                  b0dfec4cc86b    8 days ago       4.57GB
<none>              <none>               dcf7266c6988    2 weeks ago      156GB
upbeat_napier       latest               a29867f19dbf    2 weeks ago      9.61GB
<none>              <none>               2af81c736b3e    2 weeks ago      9.61GB
<none>              <none>               102283312064    2 weeks ago      9.61GB
<none>              <none>               80a686964acf    2 weeks ago      9.61GB
exemple_python      latest               1f88553e8143    2 weeks ago      933MB
python              latest               1f88553e8143    2 weeks ago      933MB
```

Now remove the old name:

```
docker rmi <old_name>
docker images
```

We can now see that the image called 'python' was removed and we are left with our newly named image:



```
REPOSITORY          TAG                  IMAGE ID        CREATED          SIZE
guy-pytorch         latest               d5c8d3130d48    6 days ago       8.92GB
buildots            1.0                  b0dfec4cc86b    8 days ago       4.57GB
<none>              <none>               dcf7266c6988    2 weeks ago      156GB
upbeat_napier       latest               a29867f19dbf    2 weeks ago      9.61GB
<none>              <none>               2af81c736b3e    2 weeks ago      9.61GB
<none>              <none>               102283312064    2 weeks ago      9.61GB
<none>              <none>               80a686964acf    2 weeks ago      9.61GB
exemple_python      latest               1f88553e8143    2 weeks ago      933MB
```

Note that 'rmi' command has the purpose of removing the image all together, but as you have now the same image present under different names (tag), 'rmi' will simply remove the name given as a parameter in the command above. You are now ready to run your first container!

### 3.2.2 *Running your first container*

First, get the list of available image installed on the server:

```
docker images
```

In order to run a container based one of the image available:

```
docker run -it <image_name>:<version> bash
```

Or else from the image ID directly:

```
docker run -it <image_ID> bash
```

You could replace bash with another script or program living inside your image, but this is outside the scope of this tutorial.

You are now inside your container, you can consider yourself in a separated and isolated working environment. You can install other packages and software within it as you would do on a regular linux terminal. Note that you are logged in this container as a root user. You can create other users as well (from within your container) and open your container as a specific user with the following command:

```
docker run -u <username> -it <image_name>:<version> bash
```

If you have exited your container or closed the terminal session, you can always enter a running container by first listing the running containers:

```
docker container ls
```

And then:

```
docker exec -it <container_name> bash
```

### 3.2.3 *Exiting and committing your container*

Now that you have done some work, and necessarily changed and/or install additional packages, you may want to commit it in order to bring the image up to date with your container. If you are still inside the container, exit it:

```
exit
```

Display the containers:

```
docker container ls -a
```

Finally, commit your container using the corresponding container ID, image name and version.

```
docker commit <container_ID> <image_name>:<version>
```

Note that if you enter the same <version> tag as your image, this will update the image. If you enter a different <version> tag, a new image will be added to the image list, do not increment the version all the time unless necessary for your project, as it will take up space the server.

## 3.3 An important note on memory

Make sure to remove unnecessary data from your container before committing as the image will become too bulky over time. A good rule of thumb would be to keep only functional files (scripts, software,...) and delete (after downloading to your local machine of course) bulky data sets and results.

Note that in the case of using a Jupyter server (will be explained in the fourth section of this tutorial), deleting a file from the graphical interface does not mean the file was deleted. The file is actually moved to the following folder : /.local/share/Trash/files Before committing your container, make sure this folder is emptied first.

## 3.4 Important guidelines to follow

In order to share the lab server resources efficiently, it is crucial that each and everyone of us follow the following guidelines:

- Your image must be named after your name, or at least contain clear ownership indications, for example "python-image-Tomer" or "Tomer" are acceptable image names, "python-image" is not.
- You must commit your work only after having deleted files that are not necessary for your environment to function (data sets, trash, ...). Or else this will be permanently written in your image history and make your image too bulky over time
- Do not keep images and/or containers that are unused
- If you are unsure whether an action may affects someone else work, always ask first
- Always keep in mind that you are sharing server resource with other people project and that your action may affect hours of work time

Failing to comply to those guidelines will affect other people at some point, the admin will then have to solve this issue. If your images is creating the problem, you will be requested to delete it and create a new one following the guidelines mentioned above.

# 4 WORKING WITH JUPYTER SERVER

Many of you will find it useful to have Jupyter installed as part as their image. Jupyter offers a graphic interface for managing your file and enables you to create notebook in order to present and reproduce your experiment flows. If you have already used Jupyter in the past, you will find that the only difference in using it inside a container resides in the way of connecting and accessing it.

## 4.1 Installing Jupyter

In order to install Jupyter, you will need the following:

- Python 2.7 or Python3 (>3.3)
- Pip installed

Simply run the following command:

```
pip install jupyterlab
```

If you have several Python versions and want to target only one specifically:

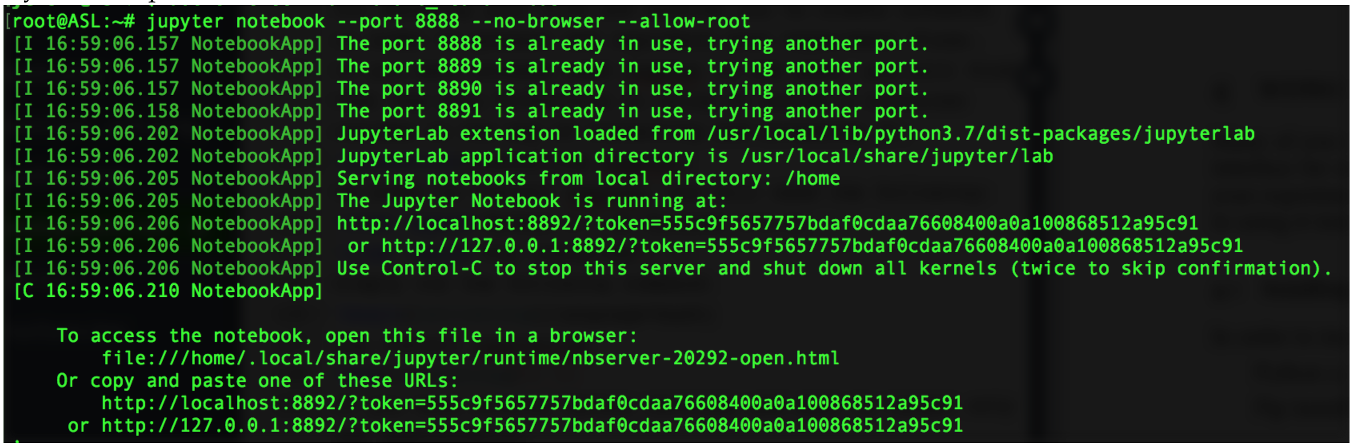
```
python3.7 -m pip install jupyterlab   #example for Python 3.7
```

## 4.2 Launching a Jupyter Server

In order to launch a Jupyter Server, you will need the following command to be executed:

```
jupyter notebook --port 8888 --no-browser --allow-root
```

The last flag is necessary as you are probably, at this point, logged in as root. If you logged in as a user, there is no need for that flag to be put. The port number can be explicitly written like in our example or generated automatically by Jupyter. In any case, if the port specified is not available, Jupyter will try the next port one after the other until it finds an available one.



```
root@ASL:~# jupyter notebook --port 8888 --no-browser --allow-root
[I 16:59:06.157 NotebookApp] The port 8888 is already in use, trying another port.
[I 16:59:06.157 NotebookApp] The port 8889 is already in use, trying another port.
[I 16:59:06.157 NotebookApp] The port 8890 is already in use, trying another port.
[I 16:59:06.158 NotebookApp] The port 8891 is already in use, trying another port.
[I 16:59:06.202 NotebookApp] JupyterLab extension loaded from /usr/local/lib/python3.7/dist-packages/jupyterlab
[I 16:59:06.202 NotebookApp] JupyterLab application directory is /usr/local/share/jupyter/lab
[I 16:59:06.205 NotebookApp] Serving notebooks from local directory: /home
[I 16:59:06.205 NotebookApp] The Jupyter Notebook is running at:
[I 16:59:06.206 NotebookApp] http://localhost:8892/?token=555c9f5657757bdaf0cdaa76608400a0a100868512a95c91
[I 16:59:06.206 NotebookApp]  or http://127.0.0.1:8892/?token=555c9f5657757bdaf0cdaa76608400a0a100868512a95c91
[I 16:59:06.206 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:59:06.210 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/.local/share/jupyter/runtime/nbserver-20292-open.html
    Or copy and paste one of these URLs:
        http://localhost:8892/?token=555c9f5657757bdaf0cdaa76608400a0a100868512a95c91
     or http://127.0.0.1:8892/?token=555c9f5657757bdaf0cdaa76608400a0a100868512a95c91
```

In this example, the port 8888 was not available and we were assigned to port 8891. Note the port number and the token you are being given, it will be required for the next step.

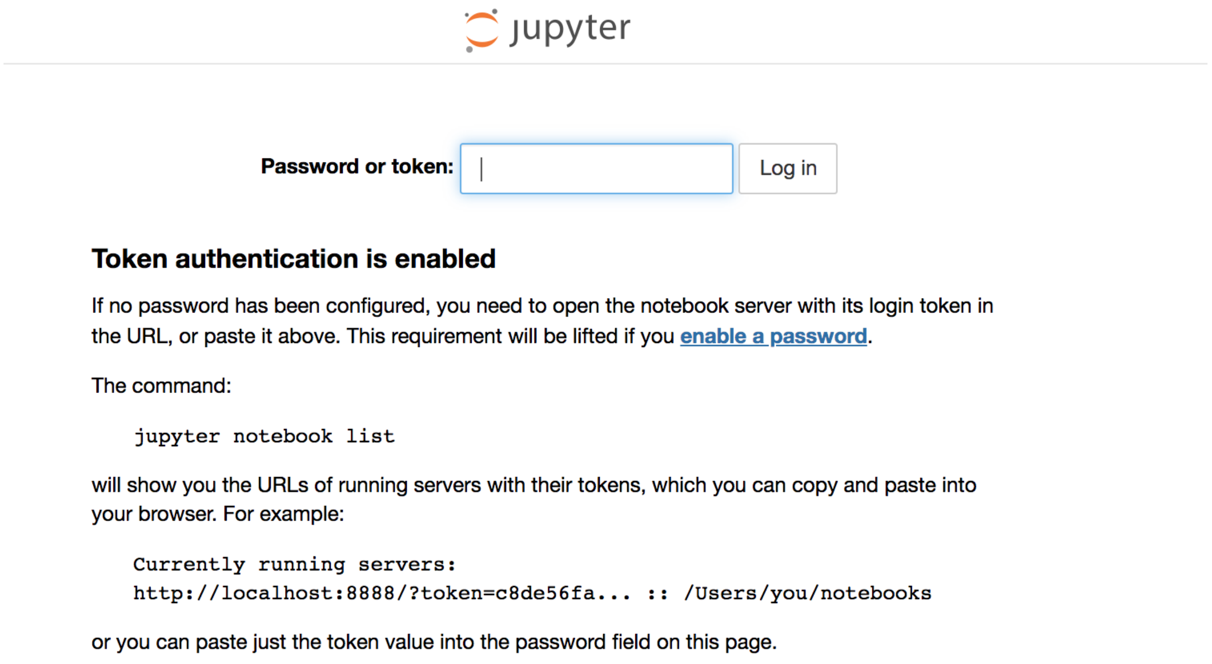
## 4.3 SSH tunnel

On a local machine, accessing the Jupyter server is as simple as browsing the address http://localhost:<port number>.
In order to do so whith a Jupyter server launched from a remote machine, we need to create a SSL tunnel.

```
ssh -N -L localhost:<machine port>:localhost:<Jupyter port> <username>@ASL.tau.ac.il
```

- <machine port> can be replaced with any port number available on your local machine.
- <Jupyter port> must be replaced with the port number you have been given in the previous step.
- <username> is replaced with your server username

You will be requested to enter your password. Once your password is entered, no response will be shown and that is perfectly normal. Keep this terminal tab open as long as you are using the Jupyter server.
You can now browse https://localhost:<machine port>. You will see the following page:



**Token authentication is enabled**

If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.

The command:

```
jupyter notebook list
```

will show you the URLs of running servers with their tokens, which you can copy and paste into your browser. For example:

```
Currently running servers:
http://localhost:8888/?token=c8de56fa... :: /Users/you/notebooks
```

or you can paste just the token value into the password field on this page.

Enter the token you have been given on the previous step and log in. You will now see the file tree, with the root folder being the folder from where you launched the Jupyter server. If you launched the Jupyter server from ~/, you will be able to navigate through all your files bellow ~/, but not above.

## 4.4 Your container IP address

Note that on certain images such as tensorflow, your container will have a different IP address than the server. As a result, you can not simply use 'localhost' but will have to replace it with your container's IP address. In order to find your container IP address:

```
docker inspect <container_id>
```

# 5 ADDITIONAL USEFUL COMMANDS

rerun an exited container:

```
docker start <containerID>
```

enter a running container:

```
docker exec -it <containerID> bash
```

pause a container:

```
docker pause <containerID>
```

stop (exit) a container:

```
docker stop <containerID>
```

filtering commands:

list of all containers, only IDs and image:

```
docker ps -a | awk ' {print $1,$2 } '
```

list of all containers, only IDs and image filtered by image name:

```
docker ps -a | awk ' {print $1,$2 } ' | grep <imagename>:<version>
```

remove all containers based on certain image:

```
docker ps -a | awk ' {print $1,$2 } ' | grep <imagename>:<version> | awk ' {print $1 }'
    | xargs -I {} docker rm {}
```