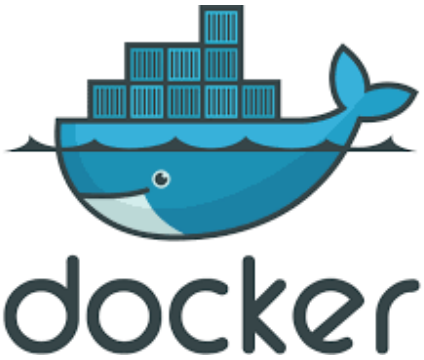

METRICS 2nd RAMI CASCADE CAMPAIGN

Aerial Domain

Docker basic guidelines

FADA-CATEC
Pablo León Barriga
Francisco J. Pérez Grau

January 20, 2023

Contents

Abstract

A basic tutorial explaining very basics of Docker as well as basic usage to start using it practically, oriented to METRICS RAMI 2nd Cascade Campaign for Aerial Robots as part of ICUAS'23 UAV Competition.

1 Work environment and requirements

In this section, the "Virtual Machine" (Docker container) provided to participants will be described along with the software and hardware requirements for getting started.

1.1 Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow you to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, competitors and organizers can rest assured that the application will run on any other Linux machine regardless of any customized settings that the machine might have that could differ from the machine used for writing and testing the code.

Thus, a Docker container was chosen as a starting point for the teams participating in this competition. To ease the start-up, a complete guide for installation and running the container is provided below.

For a better understanding of what Docker is and how to use it, some basic concepts will be described hereafter.

- **Docker Images**: the blueprints of our application which form the basis of containers. This image won't be modified by any of the containers, unless they save (commit) their changes into a new version of the image.
- **Docker Container**: is created from Docker images and runs the actual application. It contains everything you need to run it – code, runtime, system tools, system libraries, and settings. Once you run an image and create a container, you should start and attach to the same container in order to access the modifications you make. Containers are not erased unless you consciously remove them. **Caution** with commands as *docker container prune* or *docker container rm <container-id-or-name>*, you can lose all your changes!!
- **Host Machine**: is the computer where the guest operating system, in this case a Docker container, runs on. The resources of the host machine are shared with the guest machine, so no hardware needs to be virtualized.

1.2 Host Requirements

Docker host can be run on multiple hosts OS (Windows, macOS, Linux), but Ubuntu is recommended in order to connect easily your **GPU** (if you want to

use Machine Learning solutions) or your **X server** (if you want to directly run **GUIs** from the Docker container).

For those teams which won't use any of the highlighted features, feel free to choose your favourite OS. For those who plan to use the Docker container as a regular Ubuntu (despite of the desktop environment), please refer to the following recommendations.

If you are going to use Machine Learning algorithms, you must have at least a GeForce 400 Series Nvidia GPU in order to install nvidia-drivers >=525.60.11.

The fully tested operating system until now is Ubuntu 20.04 with nvidia-drivers>=525.60.11.

A lower version of Ubuntu may work, but lower nvidia-drivers as well as CUDA≠11.8 leads to CUDA mismatched version runtime errors. It is okay if CUDA is not installed on the host, but having different versions of CUDA on host and guest leads to runtime errors as well.

If you have already installed CUDA and different projects depending on it, and you don't want to break them, it is **strongly recommended** to get a **fresh Ubuntu** installation on a disk partition or removable device (SD or persistent pendrive).

The Docker image offered is an **Ubuntu 20.04** with **ROS**, **Darknet**, **nvidia-drivers-525**, **CUDA 11.8** and **CUDNN8.7**.

2 Getting Started

In this section, the dependencies installation, the initial execution of the Docker image, as well as the useful container's commands to reopen and save changes will be discussed.

2.1 Dependencies

The main dependencies for running the Docker image are basically two: **Nvidia drivers>= 525** and **Docker**.

2.1.1 Nvidia drivers

For those who have not installed nvidia-drivers yet, the easiest way to do it is through the Ubuntu's *Software & Updates* manager as described below or in Figure 1.

Win → Type "Software & Updates"→ go to *Additional Drivers* tab → Select nvidia-drivers-525 or higher → "Apply Changes"

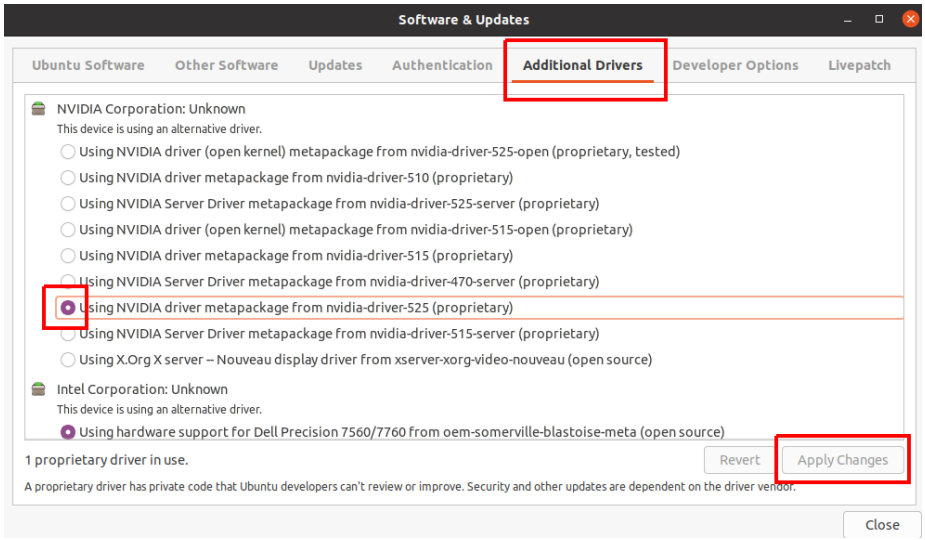

Figure 1: Ubuntu's Software & Updates manager nvidia-drivers installation.

2.1.2 Docker

Installing Docker is also really straightforward. You can go to the official webpage[1] and download it, or open a new terminal and type the following commands

[1] `https://www.docker.com/get-started`

if you are on an Ubuntu distribution:

Install Docker[2]:

```
$ sudo apt-get update
$ sudo apt-get -y install apt-transport-https ca-certificates curl gnupg
lsb-release

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu (lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Fix permission issues[3]:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ newgrp docker
```

Install nvidia-docker[4]: It is recommended to follow the instructions provided in the following link, as the instructions are too long to fit clearly on this document.
https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#install-guide/

At this point, you may want to log out and log in again, instead of typing **newgrp** command in order to apply permission changes performed with **usermod** to each new terminal you open.

Pull Docker image

Once Docker is installed and working, the next step is to download the competition base Docker image. In order to get it, you only have to type the next command:

```
$ docker pull metricscascade/ubuntu20-rami-cascade-campaign
```

[2] https://docs.docker.com/engine/install/ubuntu/
[3] https://docs.docker.com/engine/install/linux-postinstall/
[4] https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#install-guide/

3 Basic Usage

Once the steps of the previous sections are done, we are ready to start using the METRICS RAMI Docker image. In this section, the basic Docker usage needed to run and work with the created container will be covered.

3.1 Running the Docker Image

To create a container from the *ubuntu20_rami_cascade_campaign* image provided, the **docker run** command will be used. This only needs to be run the first time, though you may want to create different containers to try different things.

User: *metrics* **Password:** *metrics*

Note that each time you run the docker image, a new container completely independent from the rest will be created, and changes performed to one of them won't be visible to other containers, unless you copy those changes via **docker cp** command.

The first step will be figuring out which is the image identifier, or image id. Run on a terminal the following command and you will see something like this:

```
$ docker images

REPOSITORY                TAG                 IMAGE ID       CREATED
metricscascade/ubunt...   latest              13dceeb6c45a   4 days ago
```

Now copy your image id and you can directly run this image with the next command, and see a new bash session as below:

```
$ docker run -it <image-id>
metrics@1c45e4bfc539:~$
```

As far as you want to use different features such as running GUIs, using the host GPU on the Docker container, or connect host and guest to the same network in order to share roscore, different arguments must be passed to **docker run** cmd.

3.1.1 Connect guest to host's network

If you just want to be able to communicate both host and guest, you must add the –**net=host** flag.

```
$ docker run -it --net=host <image-id>
metrics@hostname:~$
```

If you have ROS installed on your host machine, you can now check that running a roscore in your host will allow you to make a **rostopic list** in the container.

3.1.2 Sharing host's X server (runnig GUIs)

First of all, before running the container with the Xserver options, you will need to type in your host:

```
$ xhost +
access control disabled, clients can connect from any host
```

You may need to reuse this command each time you restart your host machine. **Be careful if you are on a public network, because this may lead to security problems. If you want to give access specifically to the docker container you would have to google a bit. If you want to turn on access control again, just type **xhost -** and you will return to normal state.

Next, you should run your Docker image with the following command:

```
$ docker run -it --env=DISPLAY --volume=$HOME/.Xauthority:/home/metrics/.Xauthority \
--privileged <image-id>
```

Now you can try to open gedit on your container, or run rqt_image_view for example.

3.1.3 Connecting host's GPU

If you plan to use the GPU, you should take a close look to requirements (subsection 1.2). Once you have fulfilled them, type the following on a terminal:

```
$ ls -la /dev | grep nvidia
crw-rw-rw-  1 root root    195,   0 Oct 25 19:37 nvidia0
crw-rw-rw-  1 root root    195, 255 Oct 25 19:37 nvidiactl
crw-rw-rw-  1 root root    251,   0 Oct 25 19:37 nvidia-uvm
```

These are the device's mountpoints you would have to indicate to your Docker image running step as follows:

```
$ docker run -it --gpus all <image-id>
```

Now if you run the previous step command, you should see nvidia devices again, now on your container.

As the main ML dependencies are installed with a specific hardware, which difficultly will be the same as your computer, one of the first steps before using your GPU will be to reinstall nvidia-drivers, CUDA and CUDNN on the container.

```
metrics@hostname:~$ sudo apt install -y --reinstall nvidia-driver-525
nvidia-cuda-toolkit libcudnn8=8.7.0.84-1+cuda11.8 libcudnn8-dev=8.7.0.84-1+cuda11.8
```

This will take a few minutes to finish, and will ask you to enter your input keyboard language before it ends.

3.1.4 All together

Read carefully subsections 3.1.2 and 3.1.3 for pre- and post- running actions, but you can run the docker image with this command:

```
$ docker run -it --privileged --net=host --env=DISPLAY --gpus all <image-id>
```

3.2 Connect to your Docker container

As mentioned before, you will normally run the Docker image only once, you should select which features you would like to use and run accordingly the given Docker image. If you have tested every subsection's commands, you will have many containers at this point. You can check how many containers you have initialized by typing **docker ps -a**. If you want to remove any of those containers, you can run **docker container rm <container-id-or-name>**, where the container id can be also the name or also a list of multiple space separated ids/names.

If otherwise you only have your desired container created and you want to reopen it, then you can just type:

```
$ docker start -a -i <container-id-or-name>
```

or

```
$ docker start <container-id-or-name>
$ docker attach <container-id-or-name>
```

If you have already started the container, but you want to open a new terminal on the same container, you can do:

```
$ docker exec -ti  <container-id-or-name> /bin/bash
```

Note that if you attach to the container, and you exit this session, either via closing the terminal, or via typing exit, or pressing ctrl + D , every bash launched via **docker exec** will also terminate immediately. So you may prefer to do:

```
$ docker start <container-id-or-name>
$ docker exec -ti  <container-id-or-name> /bin/bash
```

3.3 Saving containers/images

Once you have finished working, or as a backup, you may want to save your container as a Docker image and save this image as a file.

To save your container as a new Docker image, you would have to type:

```
$ docker container commit -m 'Optional msg' <container-id-or-name>
repo_name:optional tag
```

For example:

```
$ docker container commit -m 'Added final solution' <container-id-or-name>
galactic_team
```

or

```
$ docker container commit <container-id-or-name> galactic_team:example_tag
```

Now you can run **docker images** and check that the new image is created with your repository name and your tag if specified. You should go to Docker hub[5] webpage, create your account and create a repository, so you can later upload images by pushing them. In this case, the repository name must be of the type *username/repo_name[:optional_tag]*.

```
$ docker login -u <username> -p <password> #Only once
$ docker push username/repo\_name[:optional\_tag]
```

If you would like to export a Docker image as a file instead of uploading it, you can simply run:

```
$ docker save -o filename.tar repo_name[:optional_tag]
```

For loading the image saved onto a file:

```
$ docker load -i filename.tar
```

3.4 Other useful commands

For tons of useful commands and options for shown commands, please refer to Docker reference[6].

[5] https://hub.docker.com/
[6] https://docs.docker.com/reference/