# What is a Docker container

Directly from docker web site: “A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.” This portability with no dependencies makes deployments via Docker be very quick and “simply” scalable.

# What is YOLO V4

YOLO is an NN architecture that performs object detection in images or videos.

[This github page](https://github.com/AlexeyAB/darknet) hosts one implementation of YOLO V4 and documentation about its features and dependencies. This implementation supports CUDA GPU.

The installation of YOLO V4 requires many resources (I tried on my laptop and just to install one of the dependencies of YOLO V4 (opencv) took almost 10 GB of space) and in general have several dependencies to be managed… you may not want to install it on your laptop, so guess what? Yes, AWS.

# YOLO V4 Container

To create a YOLO docker container you have to write in a file named Dockerfile:

* The OS that will run within the container
* The commands to install in the OS any package/library that might be needed (similar to a User Data script for an EC2)

Attached to this repository you can find the Dockerfile used to create the container that will host YOLO V4 (# is used to comment).

When docker creates the docker container, it takes all commands from this file.

Note: this Dockerfile points also to a “keyboard” file which is attached as well to this repository.

```

###################### Dockerfile ######################

# From the NVIDIA repository - Download an UBUNTU docker image that has already installed the CUDA and cuDNN library versions needed to run YOLO

FROM nvidia/cuda:10.2-cudnn7-devel-ubuntu18.04

#Copy the file keyboard (from the current directory in the local machine) to the folder /etc/default/keyboard that will be created in the Docker container

COPY ./keyboard /etc/default/keyboard

# Create a folder named code in the root directory and enter it (like mkdir /code + cd /code)

WORKDIR /code

#This flag ensures that no prompts are shown when the installation is performed

ENV DEBIAN\_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y \

sudo\

--reinstall keyboard-configuration\

wget\

python3-pip\

python3-opencv\

libopencv-dev\

libomp-dev\

python-qt4 \

python-pyside \

python3-pip \

python3-pyqt5\

nano\

git\

cmake

RUN pip3 install\

scikit-learn\

tqdm\

numpy\

jupyter\

pandas

# Clone the darknet Github in the current folder

RUN sudo git clone https://github.com/AlexeyAB/darknet.git

#####################################################

```

The keyboard file referenced by the Dockerfile is

```

###################### Keyboard ######################

# Check /usr/share/doc/keyboard-configuration/README.Debian for

# documentation on what to do after having modified this file.

# The following variables describe your keyboard and can have the same

# values as the XkbModel, XkbLayout, XkbVariant and XkbOptions options

# in /etc/X11/xorg.conf.

XKBMODEL="pc105"

XKBLAYOUT="us"

XKBVARIANT="intl"

XKBOPTIONS=""

# If you don't want to use the XKB layout on the console, you can

# specify an alternative keymap. Make sure it will be accessible

# before /usr is mounted.

# KMAP=/etc/console-setup/defkeymap.kmap.gz

BACKSPACE="guess"

#####################################################

```

Once these files are created, you are ready to create the Docker image.

Note: in case you are not too interested in creating your own Docker image for YOLO, you can jump to section “How to run the YOLO Docker image on a P2 EC2” – indeed you can use the image that I pushed to my repository.

# How to install Docker

Please visit <https://docs.docker.com/engine/install/ubuntu/>

# How to create a Docker image

In what follows I assume that Docker is already installed.

To create the image (note you may need to start your commands with “sudo”):

1. Open a terminal
2. Go into the folder that hosts the Dockerfile and the keyboard file and run
   1. >> docker build -t <container-name> .
   2. For example
   3. >> docker build -t yolo-container .
3. To list the Docker images currently available:
   1. >> docker images
4. Optionally you can tag your container and push it to your Docker repository
   1. >> docker tag yolo-container <docker-account-name>/<custom-image-name>
   2. >> docker login
   3. Type your user and then the pwd
   4. >> docker push <docker-account-name>/<custom-image-name>

# How to run the YOLO Docker image on a P2 EC2

We are targeting to run the Docker YOLO image on a P2 EC2 because they are equipped with GPUs.

1. Start a P2 (or higher) EC2 instance and make sure to:
   1. select a storage with 80 GB
   2. Select an AMI UBUNTU with the Deep Learning tools already installed; in my case I chose and tested “Deep Learning Base AMI (Ubuntu 18.04) Version 30.0”
   3. **Be careful, P2 instances are expensive, you may want to try with a Spot Instance**
2. Once you are logged in:
   1. >> sudo apt-get update
3. Install docker (below I show the “dirty” way **deprecated for Production environments**):
   1. >> curl -fsSL https://get.docker.com -o get-docker.sh
   2. >> sudo sh get-docker.sh
4. Pull the YOLO docker image available at asonnellini repository
   1. >> docker pull asonnellini/yolo-container-nvidia10.2-cudnn7
5. Check the image has been downloaded
   1. >> docker images
6. Run the container:
   1. >> sudo docker run -it --gpus all asonnellini/yolo-container
7. You will be prompted “inside the container” inside the folder /code (think why?)
8. There you should find the “darknet” folder and enter it:
   1. >> cd darknet
9. Now you have to compile YOLO ensuring that GPUs are enabled
   1. >> nano Makefile
10. Set the following flags to 1:
    1. GPU
    2. CUDNN
    3. CUDNN\_HALF
    4. OPENCV
    5. AVX
    6. OPENMP
11. Save the changes and exit the file
12. Still from within the folder darknet run the “make” command:
    1. >> make
13. Check whether the installation ends with no errors
14. To test whether the installation was successful, try to run YOLO using its already trained weights:
    1. Still from inside WGET the weights:
    2. >> wget <https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights>
15. Run the prediction on a test image
    1. >> ./darknet detector test ./cfg/coco.data ./cfg/yolov4.cfg ./yolov4.weights data/dog.jpg -thresh 0.25
16. The terminal should show an output similar to this:

data/dog.jpg: Predicted in 169.375000 milli-seconds.

bicycle: 17%

bicycle: 92%

dog: 98%

truck: 92%

car: 24%

pottedplant: 33%