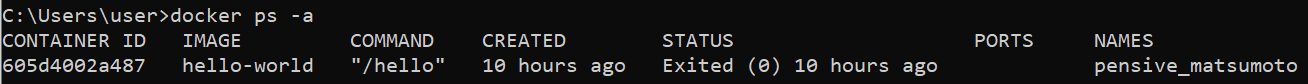
**docker images** -> shows the images stored in a local repo.

**docker ps -a** -> shows all the containers that we have.



we can change/assign name to the container if we do not do it the Docker will assign the name by himself.

In a column STATUS we can see the current status of all containers, and it is informative if for example we want to stop a specific container, or to see how many containers are running.

**docker ps** ->shows the running container.

**docker pull nazvanie\_image** -> если у нас есть image но мы хотим узнать на всякий случай не было ли обновлений. Скачается последняя версия этого image.

**DELETING**

**docker system prune -a** -> will remove all containers that currently are not in use. and images (((.

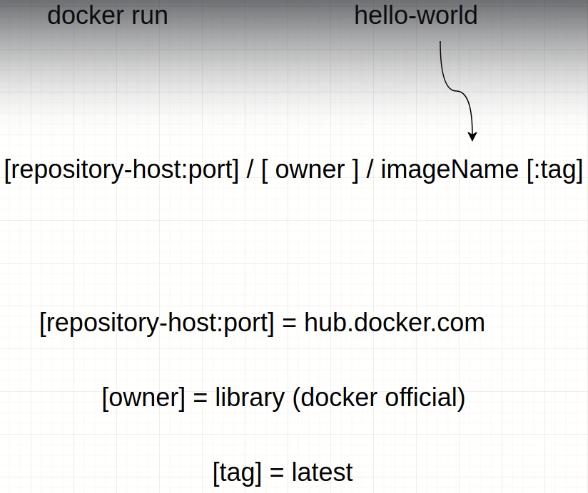
**docker container prune** -> will remove all stopped containers.

**docker rmi image\_id** -> will remove specified image

**docker rm -f name\_of\_container** -> removes/delete specified container

**docker system prune -f** ->will delete <none> images

**docker rmi image\_id\_1 image\_id\_2** ← удаляет имеджы с указанными ID

**The way the commands compose:**

всё что в квадратных скобках это указывается опционно.

1) docker run hello-world

2) docker run hub.docker.com/library/hello-world:latest

Both commands should have the same output.

Entry point

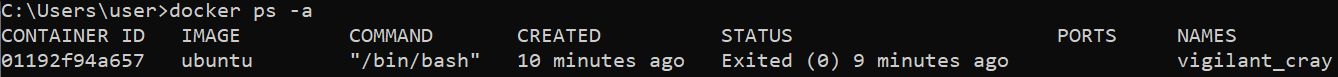
it is the command that is starts the downloaded image, it is used with run comand

**Run commands** (works with images)

**docker run hello-world** ->runs the image as by default.

**docker run -d hello-world** ->-d means to Detach (отсоединить) the Container. This flag (-d)causes Docker to start the container in "detached" mode. A simple way to think of this is to think of -d as running the container in "the background, Runs the image without output, might be helpful if you run big images with long output. The only output would be the id of the container. Runs in the background.

If for example we type docker run ubuntu -> in that case we do not provide any information about input/output and Docker doesn't know what to do with it, and the Entry point of ubuntu is to give the shell to the user. You can always check it out using docker ps -a, and under the column command see the entry point.

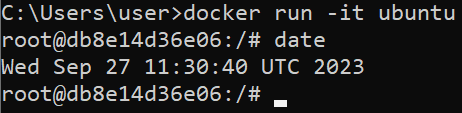


**docker run -t ubuntu** ->now we attached the standard output by providing ‘-t’ option



and we see the standard output, but if we try to enter Ubuntu instructions it will show as nothing, because Docker is running this VM/container with only standard output options. So whatever you type it is not going to that container and we are not able to interact with it.

**docker run -it ubuntu** ->now we are providing standard input/output and in the dockers running container we are able to write/send commands to vm ubuntu and see the output . So we have an ubuntu vm that we can interact with.



The docker run command creates a new brand container every time when you enter run, so it creates a new VM every time we use the run command. Whatever you have done in the specific container it remains in that specific container, you can download Java in it and play with some other applications. but when you close the container everything that you created is left in that container. You cannot expect it to be in a different container. It's like 2 different computers.

**docker run –name my\_name\_for\_that\_container ubuntu** -> creates and gives the name to the container/vm. In order to do it you need to use option –name and provide the name, then choose the image of which the container would be created.

**docker run –entrypoint=date ubuntu** ->in that case we tell the docker to run ubuntu starting with specific instruction.

You override the default entrypoint.

**docker run -p Host\_port:container\_port image** ->This is the format for port mapping.

For example:

docker run -p 4444:80 mongo

**docker run -v /path/to/host/directory:/path/to/container/directory image** ->This is the format for volume mapping (see file docker terminology)

**docker network create *name*** -> to give the name to our network!

**docker run --network=*name* nginx** -> we attaching nginx to the network (adding vm to the network)

**docker run --network=*name* ubuntu** -> we attaching ubuntu to the network

If you know when you’re creating a container that you won’t want to keep it around once you’re done, you can run **docker run --rm imageName** to automatically delete it when it exits:

**docker logs containerID/containerName**-> shows the logs of the specified container. When you run a container for example and stop it you are still able to see the output of run time of that container in spite of that it was stopped.

**stop comands**

**docker stop container\_name**-> stops container, after container is stoped it is possible to restart it, but if you plan stop and reuse container again it might be useful to start the original image with option ‘-d’ in that case it is possible to restart the container in interactive mode

docker exec [OPTIONS] CONTAINER COMMAND [ARG...] -> The docker exec command runs a new command in a running container. works only if the container is running.

For example:

docker run -it ubuntu

docker stop ubuntu

docker restart ubuntu->it will restart the container but not the terminal in ubuntu

docker exec -it ubuntu bash -> will send the command to the running container and it will open the terminal

docker stop $(docker ps -a -q) ← will stop all running containers

The ADD command requires a source and a destination.

ADD source destination

* If source is a file, it is simply copied to the destination directory.
* If source is a directory, its *contents* are copied to the destination, but the directory itself is not copied.
* source can be either a tarball or a URL (as well).
* **source needs to be within the directory where the docker build command was run**

**build**

docker build -t=delete\_me\_experement .

build -t=delete\_me\_experement <- - инструкция на создание image

. <- - точкай обозначается директория где лежит Dockerfile, здесь это текущая директория поэтому точка

**tag**

Дать новый лэйбл для имедж

docker tag dimbas/image\_name:old\_tag dimbas/image\_name:${env.BUILD\_NUMBER}

${env.BUILD\_NUMBER} <- -здесь пример из дженкинс, когда в качестве тега имеджу присваивается номер (BUILD\_NUMBER) который берётся из обьекта env