Getting your hands dirty with Docker!

Intro

this tutorial is mainly intended to give the reader a hand-on experience on Docker Commands. to achieve maximum understanding of Docker technology, we have created a simple tutorial that aims at introducing the reader to the main concepts of Docker technology. So, it is highly recommended to read the tutorial before proceeding with docker commands. You can reach the tutorial from the following link.

To start with docker, make sure that docker is already running on your machine after installation. If it is not running , just run it according to your operating system and wait until it prompt you that Docker is running. following that, open `Windows PowerShell` in case you are using `Microsoft Windows` by searching it in `Start Menu`. Alternatively, if you are using `Linux-based OS`, just Start a new `Terminal window` and let us start playing with docker:

My First Docker Image (hello-world)

Docker hello-world image is an introductory image that introduces the user to the performed processes by docker to run a container from a certain image, to run `hello-world` image type this command in your PowerShell or terminal window :

```
docker run hello-world
```

docker run command start a new container from a certain image. If the image is not available locally on your machine, docker will automatically download it from docker HUB and start a new container from the image once it has been downloaded. Don't worry about that, we will delve deeper into the details later on. For Now, it is expected to see the following output on your terminal:

```
PS C:\Users\DELL\Desktop> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
                         Digest:
sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86
d7e4e
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working
correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the
Docker Hub.(amd64)
 3. The Docker daemon created a new container from that image
which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client,
which sent it
    to your terminal.
To try something more ambitious, you can run an Ubuntu container
with:
 $ docker run -it ubuntu bash
Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/
```

Congratulations!! you have just downloaded your first image and created your first container. Good Job.

Docker Commands

Basically, there are two types of Docker commands including `Command-Line Interface (CLI)` & Docker `Instructions Commands`. Docker CLI are used mainly for managing docker components such as images and containers, while Docker instruction files are mainly used for creating Dockerfiles. Docker CLI can be divided into six categories including :

1. Important flags
2. General commands
3. Container commands
4. Image commands
5. Volume commands.
6. Network commands

We will pass in a detailed manner through each category and provide a specific example to the most important commands in each category. it is worth noting that docker has a wide range commands that can not be covered in one tutorial so bear in that how much you practice, how much you gain experience and knowledge. One more thing to to mention before starting our journey is that docker provides a detailed description of its commands. So, in case you are stuck with a certain commands do not hesitate and feel free to ask Docker using the help command as follow:

```
docker [command name] --help
eg: docker version --help
```

Docker CLI

Important flags

Flags are not docker commands, but they play an important role in controlling the results of the commands you give for docker to perform, so, you can think of the flags to docker commands as the salt to the food. And thus, you have to gain a good knowledge in dealing with them. In fact, the same flag has different meanings in different contexts of the command, but we list here some of the most important and common flags in docker commands along with their discerption in the following table:

Flag	Discerption
-a	all: list all items for a certain docker component
-d	detach: run a command in the background and make the terminal available for other commands
-f	force : force command to be performed.
-p	publish: explicitly express a port in a network
-q	
-t	tag
-v	volume: mount volume to a container upon creation
-it	interactive: perform command in the interactive mode
--name	name: set a user-defined name to a certain docker component
--help	help: gives a detailed description of a specific command
--net	network: assign certain network to a container

General commands

In the following section, we are going to describe a set of

`docker version` commands: gives a detailed description about the docker version you are using.

```
docker version
```

```
docker -v
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker version
```

```
Client: Docker Engine - Community
 Version:           19.03.8
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        afacb8b
 Built:             Wed Mar 11 01:23:10 2020
 OS/Arch:           windows/amd64
 Experimental:      false
Server: Docker Engine - Community
 Engine:
  Version:          19.03.8
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
  Git commit:       afacb8b
  Built:            Wed Mar 11 01:29:16 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

`docker ps` command: gives a list of either active containers or all containers (active and switched off).

```
docker ps
```

```
docker ps-a
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES


PS C:\Users\DELL\Desktop\docker-hadoop-master> docker ps -a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                     PORTS
NAMES
668b8cf2bc59        hello-world         "/hello"            5
minutes ago       Exited (0) 5 minutes ago
focused_yonath
```

`docker info` command: gives a detailed description about docker docker components including but limited to: the number of running, stopped paused containers, number of images, docker version, volumes, product license, CPU and memory usage.

```
docker info
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker info
Client:
 Debug Mode: false

Server:
 Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
 Images: 38
 Server Version: 19.03.8
 Storage Driver: overlay2
  Backing Filesystem: <unknown>
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local
logentries splunk syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 4.19.76-linuxkit
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 1.943GiB
 Name: docker-desktop
 ID: QOOD:VQ3W:HZ2T:BPD6:BYCO:Q7OX:MRAD:OINU:BRID:6A3L:IV4F:MCY2
 Docker Root Dir: /var/lib/docker
 Debug Mode: true
  File Descriptors: 36
  Goroutines: 53
  System Time: 2020-03-14T10:34:31.539918336Z
  EventsListeners: 3
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
 Product License: Community Engine
```

`docker stats` command: gives a detailed information about the resources consumption such as CPU, RAM and network of your local machine by the running containers.

```
docker Stats
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker stats
CONTAINER ID        NAME                CPU %               MEM
USAGE / LIMIT     MEM %               NET I/O             BLOCK
I/O           PIDS
b621c883dea9        resourcemanager     0.07%
315.9MiB / 1.943GiB   15.88%              34kB / 23.5kB       0B /
0B              230
58efd121c839        nodemanager1        0.05%
209.1MiB / 1.943GiB   10.51%              8.5kB / 20.9kB      0B /
0B              79
13d873dd13b4        historyserver       0.12%
181.2MiB / 1.943GiB   9.11%               1.84kB / 164B       0B /
0B              46
8b8e7fcd64f0        datanode1           0.11%
187.2MiB / 1.943GiB   9.41%               12.6kB / 22.5kB     0B /
0B              52
975aec01391f        datanode3           0.04%
188.2MiB / 1.943GiB   9.46%               12.7kB / 22.5kB     0B /
0B              52
3032f548cfe5        datanode2           0.04%
211.3MiB / 1.943GiB   10.62%              13.7kB / 23.3kB     0B /
0B              52
4f6eddd41e8a        namenode            0.04%
196.2MiB / 1.943GiB   9.86%               62kB / 20.9kB       0B /
0B              56
```

To obtain the stats of a specific container:

```
docker stats <container-name>
```

or

```
docker stats <container-ID>
```

Image commands

`docker image pull` command: allow the user to download image from either docker hub or docker registry into your local machine using the following structure:

```
docker image pull <image-name>[:tag]
```

For instance, pulling the the latest version of `hello-world` can be achieved through the following command:

```
docker pull image hello-world:latest
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker image pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
Digest:
sha256:f9dfddf63636d84ef479d645ab5885156ae030f611a56f3a7ac7f2fdd86
d7e4e
Status: Image is up to date for hello-world:latest
docker.io/library/hello-world:latest
```

note that docker will look for the image on your local machine if it is not available, docker will download it directly from `docker HUB`. Further, the `tag` keyword represents the version of the image you are going to download. Image name and tag can be directly obtained from docker HUB.

`docker image push` command: allow the user to transfer to transfer his/her own created into either private registry or public registry (docker HUB) using the following structure:

```
docker image push [registry-name/][username/]<image-name>[:tag]
```

For instance, I will push the previously downloaded hello-world image to my public registry on docker HUB as follow:

```
```

Container commands

`docker container run` commands: run command enables the user to create new container from a certain image. there are multiple options that can be specified when creating new container, however, the main structure of the command is as follow:

```
docker container run <image-name>
```

applying the previous command (as is) will enforce docker to create and start new container with auto-generated name as shown in the following example with `nginx` image:

```
docker container run nginx
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker container run nginx

PS C:\Users\DELL\Desktop> docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
21fd4052f73f        nginx               "nginx -g 'daemon of…"
28 seconds ago      Up 25 seconds       80/tcp
hungry_galileo
```

you can observe that docker randomly assigned `hungry_galileo` . And thus, to create a new container with specific name or to create multiple containers from the same image with different names :

```
docker container run --name <container-name> <image-name>
```

As an example, I will create 2 container from `nginx` image with different names as follow

```
docker container run --name foo nginx
```

```
docker container run --name bar nginx
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker container run --name foo nginx
PS C:\Users\DELL\Desktop> docker container run --name bar nginx
PS C:\Users\DELL\Desktop> docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
be940a8e8cd7        nginx               "nginx -g 'daemon of…"
16 seconds ago      Up 14 seconds       80/tcp              bar
cf054af93659        nginx               "nginx -g 'daemon of…"
36 seconds ago      Up 34 seconds       80/tcp              foo
21fd4052f73f        nginx               "nginx -g 'daemon of…"   3
minutes ago       Up 3 minutes        80/tcp
hungry_galileo
```

Another option is to run a new container and remove it directly after it stopped is achieved by applying the following command:

```
docker container run --rm <image-name>
```

As an example, I will apply the previous command with `nginx` image as follow:

```
docker container run --rm nginx
```

```
PS C:\Users\DELL\Desktop> docker container run --rm nginx
PS C:\Users\DELL\Desktop> docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
05b3801ee3f7        nginx               "nginx -g 'daemon of…"
14 seconds ago      Up 12 seconds       80/tcp
admiring_agnesi
be940a8e8cd7        nginx               "nginx -g 'daemon of…"   2
minutes ago       Up 2 minutes        80/tcp              bar
cf054af93659        nginx               "nginx -g 'daemon of…"   2
minutes ago       Up 2 minutes        80/tcp              foo
21fd4052f73f        nginx               "nginx -g 'daemon of…"   5
minutes ago       Up 5 minutes        80/tcp
hungry_galileo
PS C:\Users\DELL\Desktop\docker-hadoop-master> docker container
stop admiring_agnesi
admiring_agnesi
PS C:\Users\DELL\Desktop\docker-hadoop-master> docker container ls
-a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                      PORTS
  NAMES
be940a8e8cd7        nginx               "nginx -g 'daemon of…"   3
minutes ago       Up 3 minutes                80/tcp
bar
cf054af93659        nginx               "nginx -g 'daemon of…"   3
minutes ago       Up 3 minutes                80/tcp
foo
21fd4052f73f        nginx               "nginx -g 'daemon of…"   6
minutes ago       Up 6 minutes                80/tcp
hungry_galileo
668b8cf2bc59        hello-world         "/hello"
31 minutes ago      Exited (0) 31 minutes ago
  focused_yonath
```

As you can observe, once the container stopped, it has been directly removed .

Some containers are designed to be created, perform specific task and then stop. A clear example of such containers is `python` container. to keep it running after creation, this command can help:

```
docker container run -td <image-name>
```

As an example, assuming that python image is already pulled to your local machine which is satisfied in my case:

```
docker container run -td python
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker run python
PS C:\Users\DELL\Desktop> docker container ls
```

```
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
be940a8e8cd7        nginx               "nginx -g 'daemon of…"   7
minutes ago       Up 7 minutes        80/tcp              bar
cf054af93659        nginx               "nginx -g 'daemon of…"   7
minutes ago       Up 7 minutes        80/tcp              foo
21fd4052f73f        nginx               "nginx -g 'daemon of…"
10 minutes ago      Up 10 minutes       80/tcp
hungry_galileo
PS C:\Users\DELL\Desktop> docker container run -td python
99e70b02d833c10c9ae5a5e28e6fb47a910bc4861c5320b017b9654c3c1d392e
PS C:\Users\DELL\Desktop> docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
99e70b02d833        python              "python3"                7
seconds ago       Up 5 seconds
gifted_perlman
be940a8e8cd7        nginx               "nginx -g 'daemon of…"   8
minutes ago       Up 8 minutes        80/tcp              bar
cf054af93659        nginx               "nginx -g 'daemon of…"   8
minutes ago       Up 8 minutes        80/tcp              foo
21fd4052f73f        nginx               "nginx -g 'daemon of…"
11 minutes ago      Up 11 minutes       80/tcp
hungry_galileo
```

As you can observe that in the first run the container created and exited directly, while in the second run it is created and kept running.

An additional useful run command, is to start a new container and apply a specific command inside it, in case it allows for this manner, this can be achieved by running a container in the interactive mode according to the following structure:

```
docker container run -it <image-name>
```

A good example of this is running a container interactively from python image which allow you to run python commands directly after creation as follow:

```
docker container run -it python
```

Expected output:

```
PS C:\Users\DELL\Desktop> docker container run -it python
Python 3.8.2 (default, Feb 26 2020, 14:58:38)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> 5+6
11
>>> print('the great docker')
the great docker
```

Ultimately, sometimes we need to define a port in the container to be known by our local machine to allow us reach it using the web browser, this can be achieved by the following structure:

```
docker container run -d -p port:port <image-name>
```

As an example, applying such run command on `nginx` image will publish the specified port 80 to be known by our local machine as a way to reach it through the web browser as follow:

```
docker container run -d -p 80:80 nginx
```

Thus the container can be reached by typing [localhost:80]() in the browser.

---

up to this point, we assume you gained a good knowledge with docker, so that, the upcoming sections will be described without examples. For now, your job is to start applying each of the upcoming commands by yourself. bear in mind that "Practice makes perfect"

`docker container ls` commands: `ls` command is identical to `ps` and have the same format .

for active container :

```
docker container ls
```

for all containers:

```
docker container ls -a
```

`docker container rm` commands: `rm` command enables the user to delete created container. there are multiple options that can be specified when deleting container as follow:

removing one or more containers:

```
docker container rm <container-name_1> <container-name_2> ...
<container-name_n>
```

Force stop a container in case it is running and remove it:

```
docker rm -f <container-name>
```

Force stop all containers in case they are running and remove them:

```
docker rm -f $(docker ps -a -q)
```

Remove all stopped containers

```
docker rm $(docker ps -q -f “status=exited”)
```

`docker container prune` command: prune command allows docker to search for unused container and remove them according to the following structure:

```
docker container prune <container-name>
```

`container lifecycle` commands: life cycle commands enable the user to manage the operation status of Docker containers and include:

create but do not start:

```
docker container create <image-name>
```

Stop a running container

```
docker container stop <container-name>
```

Start a stopped container

```
docker container start <container-name>
```

Restart a running container :

```
docker container restart <container-name>
```

Pause a running container:

```
docker container pause <container-name>
```

Resume a paused container :

```
docker container unpause <container-name>
```

Force stop of a certain container:

```
docker container kill <container-name>
```

`performing commands inside container`: there are two command that enables you to connect into a certain container and and perform some operations inside it. these commands are `attach` and `exec` .

`docker attach` command: This command attaches your terminal to a running container using the container name as follow:

```
docker container attach <container-name>
```

you have to ensure that the attached container is running in the interactive mode before performing this command. once you are attached to the container, you can act as you are in the terminal of the container. But the problem with this commands is once you exit the container terminal, the container will in turn stop running. So, this command is useful when you are planning to run a certain command and stop the container. But what is the solution if we want to keep our container up and running after performing our commands inside it?

`docker exec` command: this command overcome the previously mentioned problems by applying it as follow:

```
docker container exec -it <container-name> bash
```

Consequently, a `born again shell` (bash) terminal will be opened to allow you apply your commands inside the container while keeping it running once you exit the bash.

Volumes Commands

`docker volume create` command: create command allows user to establish new container according to the following structure:

```
docker volume create <volume-name>
```

to create volume with specific name:

```
docker volume create --name <volume-name>
```

`docker volume inspect` command: return a detailed information about certain volume such as creation date, driver directory, name according to the following structure:

```
docker volume inspect <volume-name>
```

To directly return the volume directory :

```
docker volume inspect -f "{{json .Mountpoint}}" <volume-name>
```

`docker volume ls` command: list all created volumes according to the following structure:

```
docker volume ls
```

`docker volume prune` command: remove all unused volumes according to the following structure:

```
`docker volume prune
```

`docker volume rm` command: remove one or more volumes according to the following structure:

```
docker volume rm <volume-name>
```

to assign previously created volume to a container on starting:

```
docker run -it --name <container-name> -v <volume-name> <image-
name>
```

Network Commands

`docker network connect` command: connect a container to a specific network according to the following structure:

```
docker network connect <network-name> <container-name>
```

`docker network disconnect` command: disconnect a container from a specific network according to the following structure:

```
docker network disconnect <network-name> <container-name>
```

`docker network create` command: create a new network according to the following structure:

```
docker network create <network-name>
```

to specify the type of the created network {bridge,host,null, overlay,macvlan}:

```
docker network create -d <network-type> <network-name>
```

To start up a container and add it to a specific network:

```
docker run -it --net=<network-name> --name <container-name>
<image-name>
```

`docker network inspect` command: return a detailed information about certain network such as creation date, driver directory, name according to the following structure:

```
docker network inspect <network-name>
```

`docker network ls` command: list all created networks according to the following structure:

```
docker network ls
```

`docker network prune` command: remove all unused networks according to the following structure:

```
`docker network prune
```

`docker network rm` command: remove one or more networks according to the following structure:

```
docker network rm <network-name>
```

Docker Instruction Commands

Docker instruction commands are the building blocks of dockerfiles, these instructions specify the roadmap for docker to build an image out of dockerfile