Docker

I. Before we get started

```
# docker --version
Docker version 18.09.1, build 4c52b90
```

Docker Engine

Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process `dockerd`.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (CLI) client `docker`.

The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manage Docker objects, such as images, containers, networks, and volumes.

Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose

How to Use Docker Run Command with Examples

Docker Exercises

For each exercise, we will ask you to give the shell command(s) to:

1. Create a virtual machine with docker-machine using the virtualbox driver, and named Char.

Get started with Docker Machine and a local VM

Use `docker-machine ls` to list available machines.

In this example, no machines have been created yet.

```
$ docker-machine ls
```

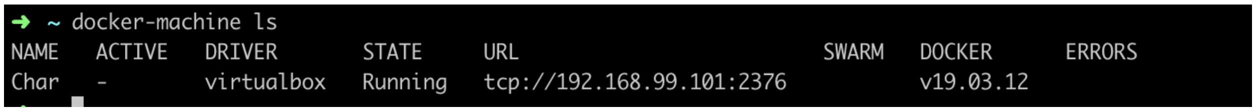
```
 NAME    ACTIVE    DRIVER    STATE    URL    SWARM    DOCKER    ERRORS
```

Create a machine.

```
docker-machine create //create a VM

--driver //the driver specified

Char //name of your VM
```

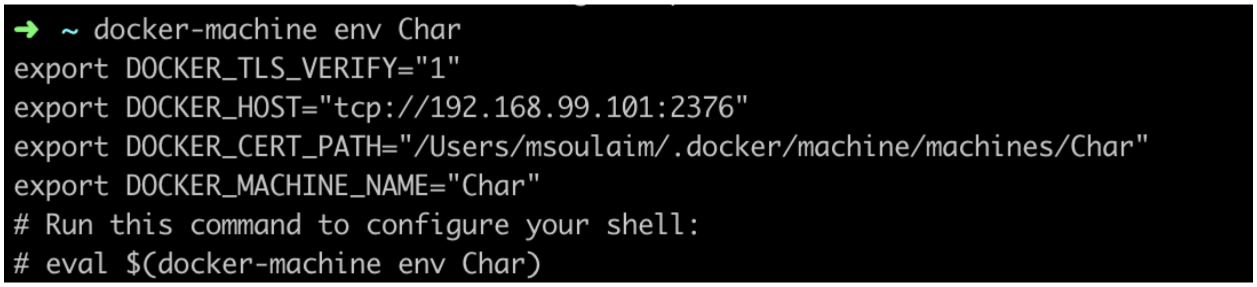
```
docker-machine create --driver virtualbox Char
```

This command downloads a lightweight Linux distribution (boot2docker) with the Docker daemon installed, and creates and starts a VirtualBox VM with Docker running.



```
➜ ~ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER       ERRORS
Char   -        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.12
```

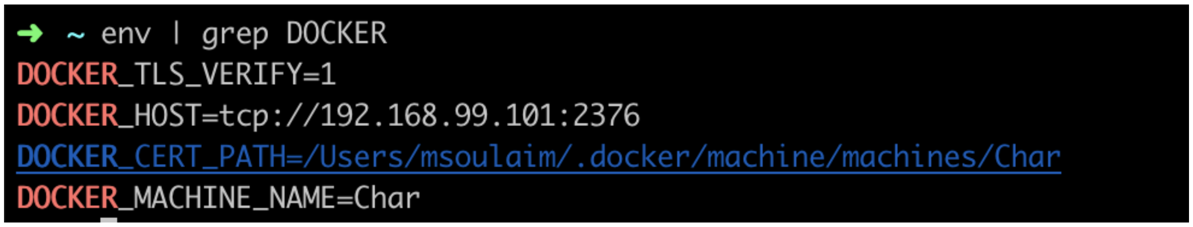
-----display the env var of your new vm



```
➜ ~ docker-machine env Char
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:2376"
export DOCKER_CERT_PATH="/Users/msoulaim/.docker/machine/machines/Char"
export DOCKER_MACHINE_NAME="Char"
# Run this command to configure your shell:
# eval $(docker-machine env Char)
```

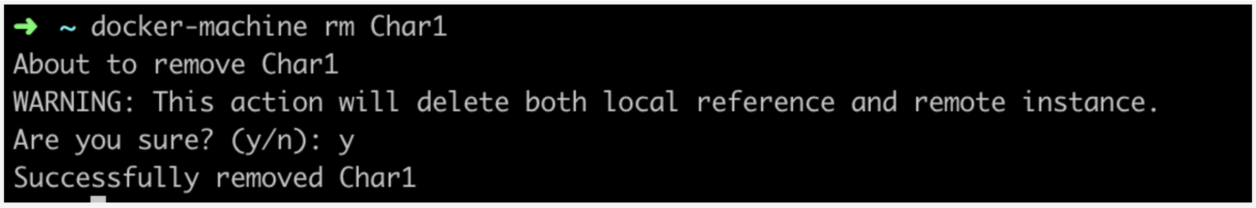
----add the VM env to your shell

```
$ eval "$(docker-machine env Char)"
```



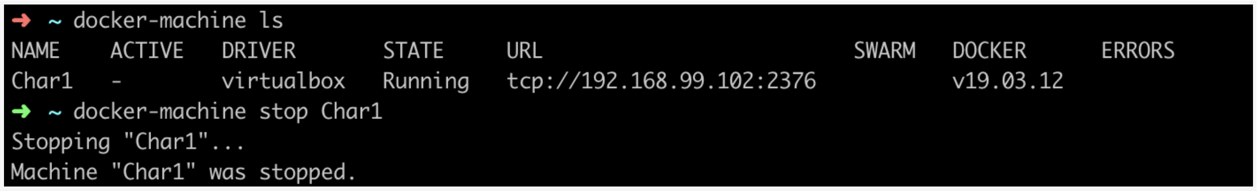
```
➜ ~ env | grep DOCKER
DOCKER_TLS_VERIFY=1
DOCKER_HOST=tcp://192.168.99.101:2376
DOCKER_CERT_PATH=/Users/msoulaim/.docker/machine/machines/Char
DOCKER_MACHINE_NAME=Char
```

delete a machine.



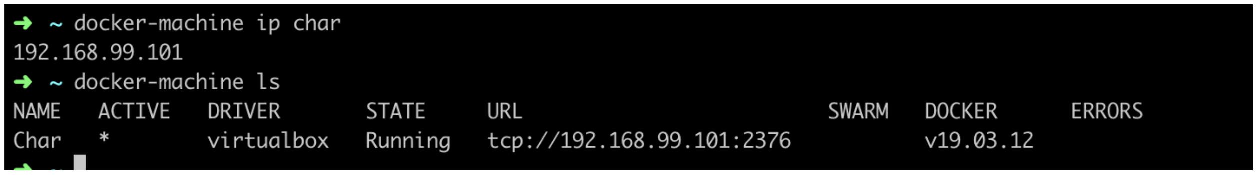
```
➜ ~ docker-machine rm Char1
About to remove Char1
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed Char1
```

stop a machine.



```
➜ ~ docker-machine ls
NAME     ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
Char1    -        virtualbox   Running   tcp://192.168.99.102:2376           v19.03.12
➜ ~ docker-machine stop Char1
Stopping "Char1"...
Machine "Char1" was stopped.
```

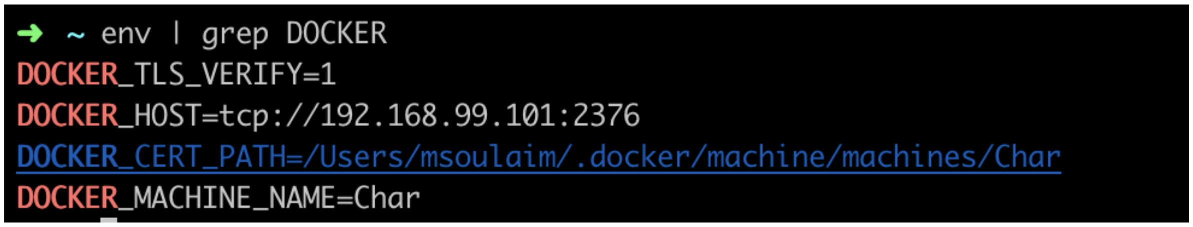
2. Get the IP address of the Char virtual machine.



```
➜ ~ docker-machine ip char
192.168.99.101
➜ ~ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
Char   *        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.12
```

3. Define the variables needed by your virtual machine Char in the general env of your terminal, so that you can run the docker ps command without errors. You have to fix all four environment variables with one command, and you are not allowed to use your shell's builtin to set these variables by hand.

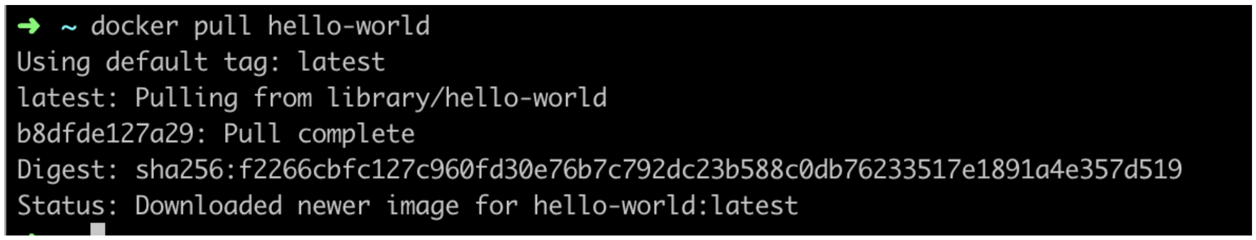
docker service ps



```
➜ ~ env | grep DOCKER
DOCKER_TLS_VERIFY=1
DOCKER_HOST=tcp://192.168.99.101:2376
DOCKER_CERT_PATH=/Users/msoulaim/.docker/machine/machines/Char
DOCKER_MACHINE_NAME=Char
```

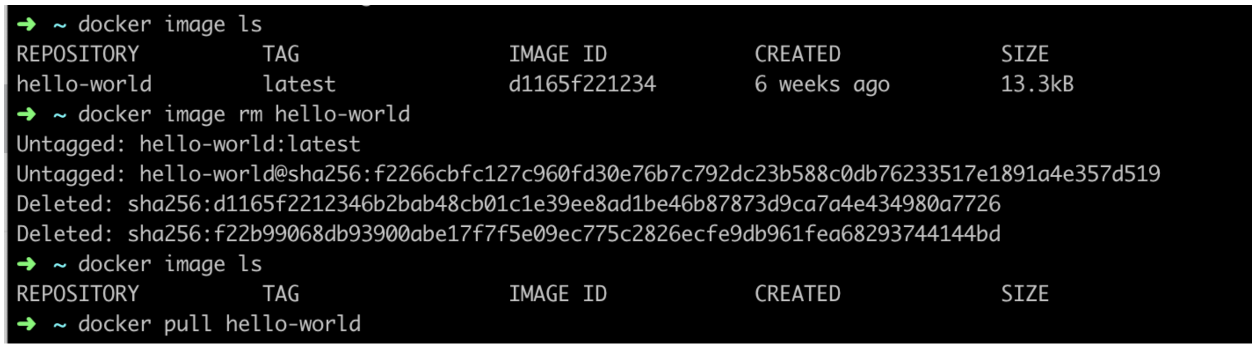
4. Get the hello-world container from the Docker Hub, where it's available.

hello-world



```
➜ ~ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:f2266cbfc127c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519
Status: Downloaded newer image for hello-world:latest
```

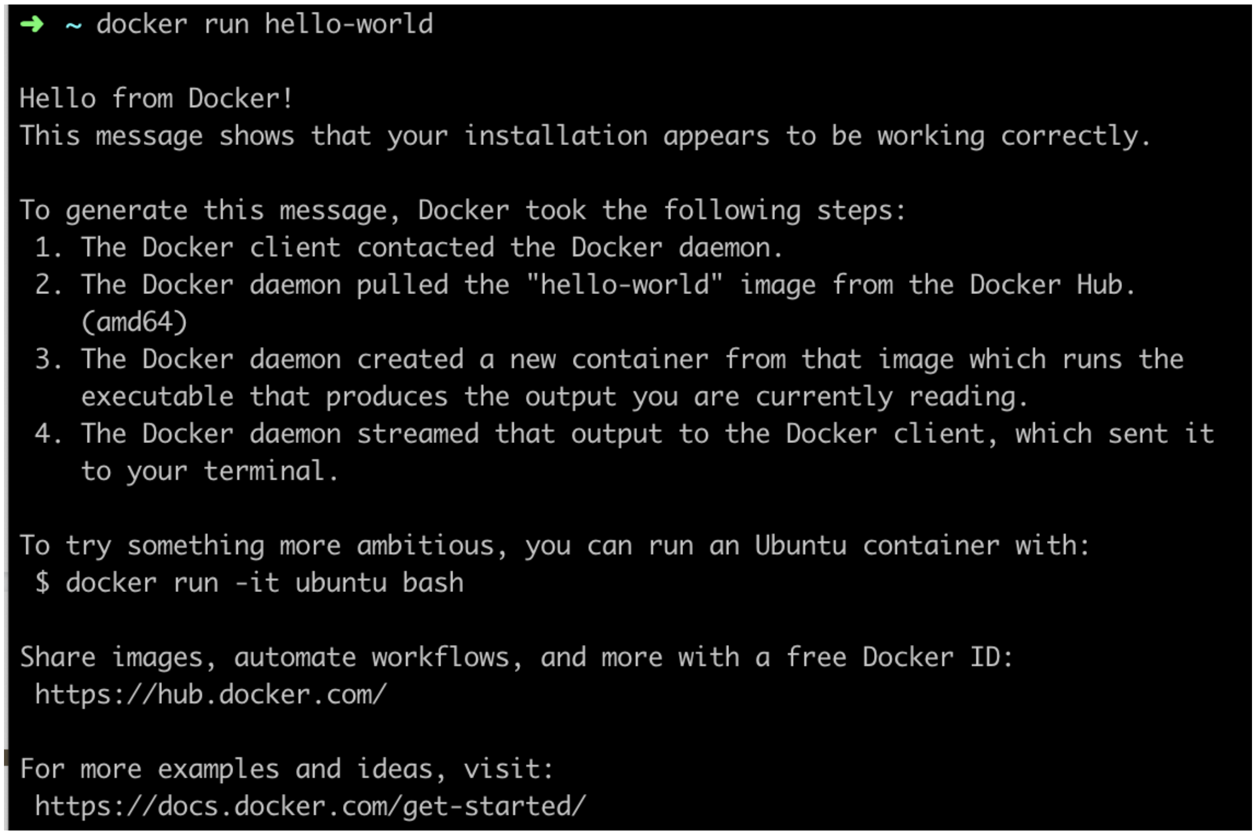
cli reference – docker image rm // docker image ls



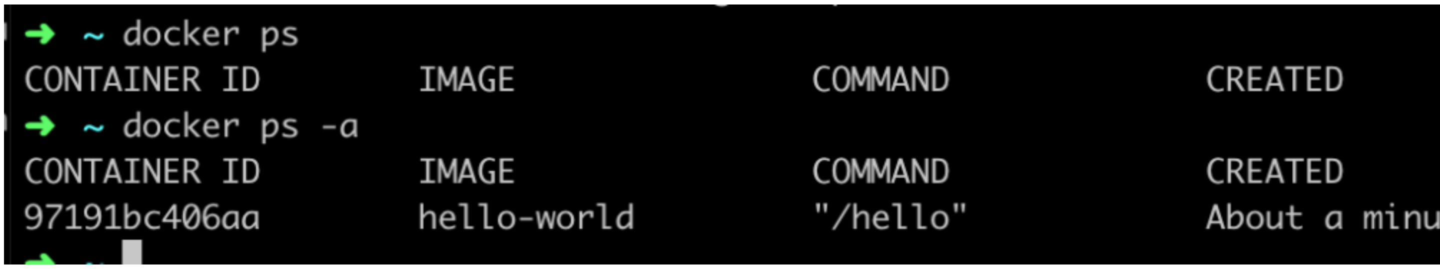
```
➜ ~ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              d1165f221234        6 weeks ago         13.3kB
➜ ~ docker image rm hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:f2266cbfc127c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519
Deleted: sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be46b87873d9ca7a4e434980a7726
Deleted: sha256:f22b99068db93900abe17f7f5e09ec775c2826ecfe9db961fea68293744144bd
➜ ~ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
➜ ~ docker pull hello-world
```

5. Launch the hello-world container, and make sure that it prints its welcome message, then leaves it.

hello-world container will start, past this shit and stop hisself



```
➜ ~ docker run hello-world

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/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```



```
➜ ~ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
➜ ~ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
97191bc406aa        hello-world         "/hello"            About a minu
```

[How to List / Start / Stop / Docker Containers {Easy Way}]

```
docker ps // running containers
```

```
docker ps -a // running and stopped containers
```

6. Launch an nginx container, available on Docker Hub, as a background task. It should be named overlord, be able to restart on its own, and have its 80 port attached to the 5000 port of Char. You can check that your container functions properly by visiting http://<ip-de-char>:5000 on your web browser.

```
docker run -d -p 5000:80 --name overlord --restart always nginx
```

get an nginx container //docker pull nginx // nginx

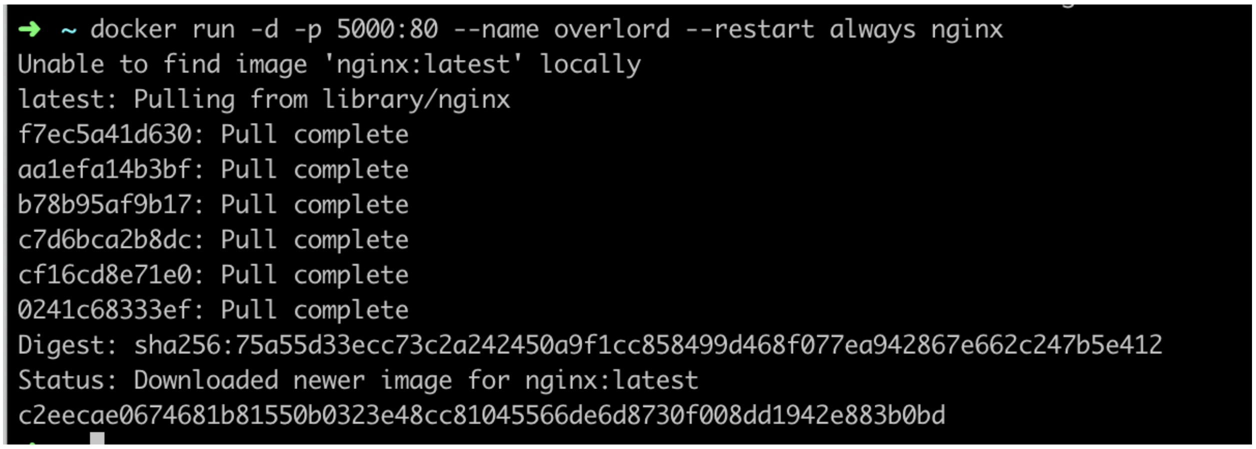
```
-d //Run container in background            //docker run
```

--name //Assign a name to the container

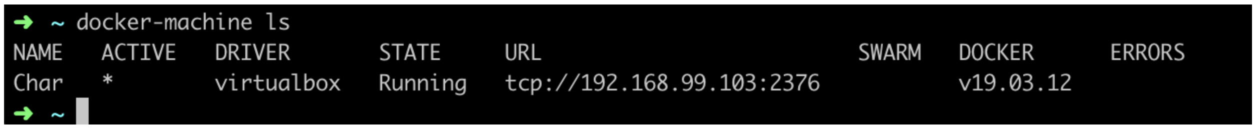
--restart //to specify a container's restart policy

Always // restart the container regardless of the exit status

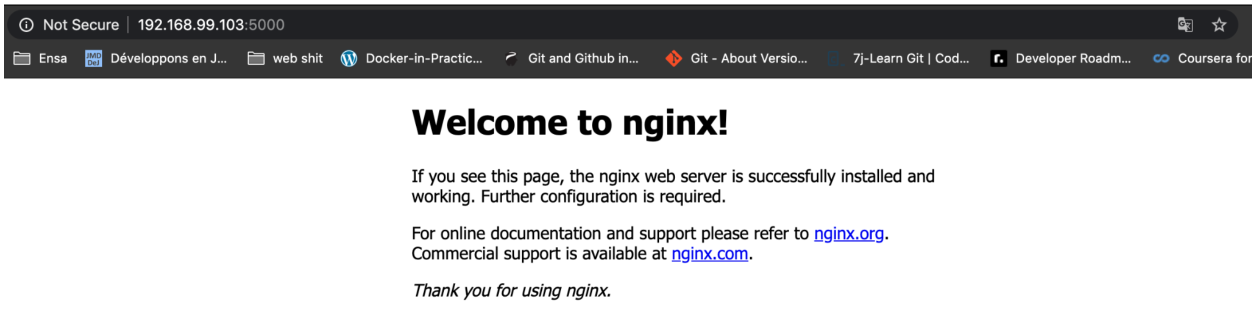
-p 5000:80 //Map TCP port 80 in the container to port 5000 on the Docker host. //Container networking



```
➜ ~ docker run -d -p 5000:80 --name overlord --restart always nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f7ec5a41d630: Pull complete
aa1efa14b3bf: Pull complete
b78b95af9b17: Pull complete
c7d6bca2b8dc: Pull complete
cf16cd8e71e0: Pull complete
0241c68333ef: Pull complete
Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
Status: Downloaded newer image for nginx:latest
c2eecae0674681b81550b0323e48cc81045566de6d8730f008dd1942e883b0bd
```



```
➜ ~ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
Char   *        virtualbox   Running   tcp://192.168.99.103:2376           v19.03.12
➜ ~
```



Not Secure | 192.168.99.103:5000

Ensa · Développons en J... · web shit · Docker-in-Practic... · Git and Github in... · Git - About Versio... · 7j-Learn Git | Cod... · Developer Roadm... · Coursera for

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

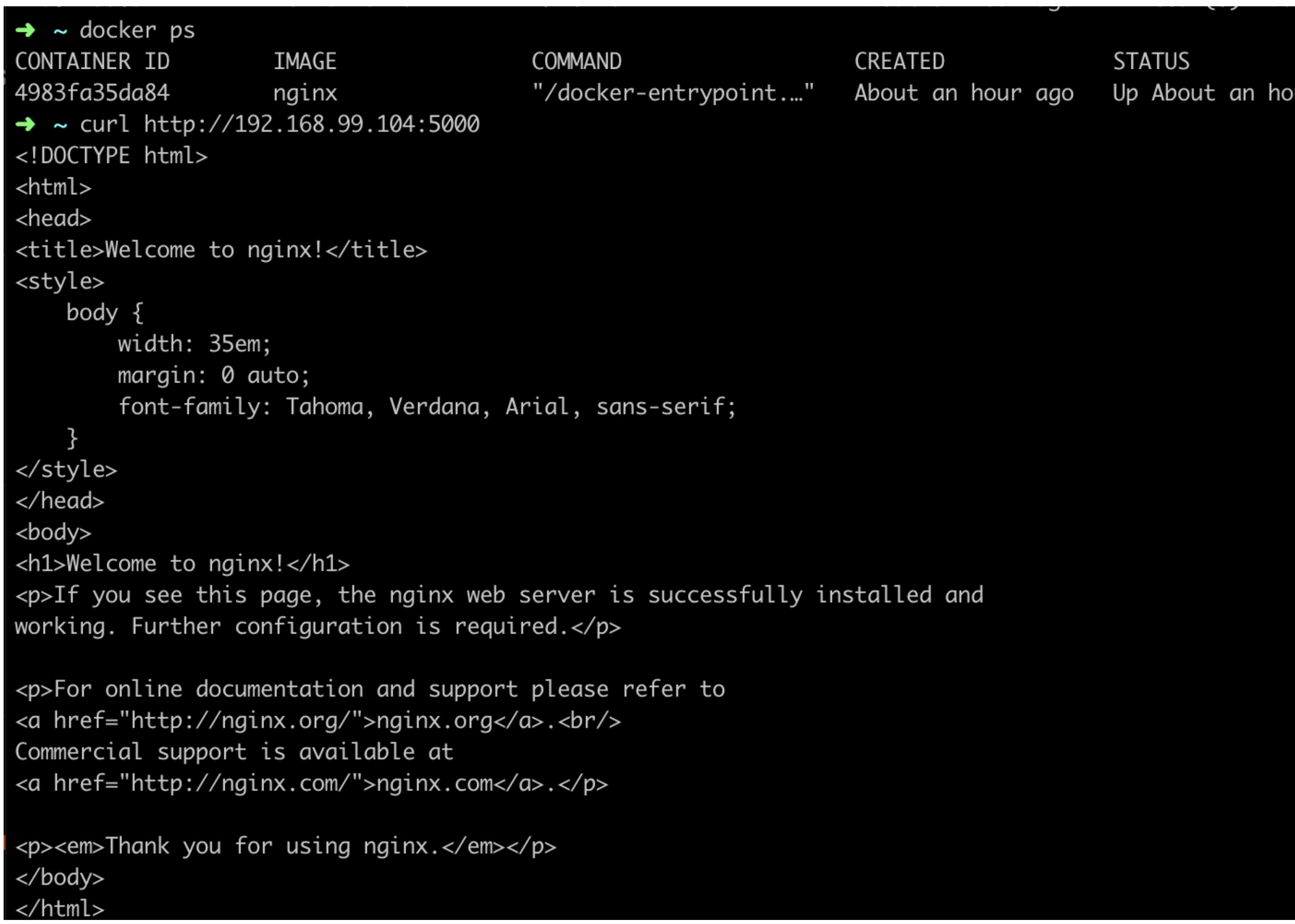
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

check the name is (overlord)

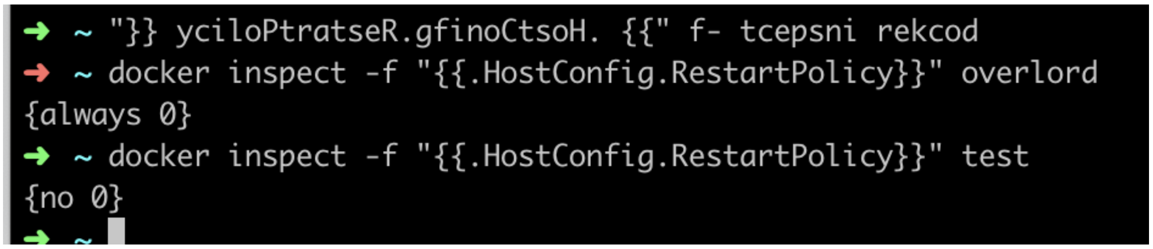
check the port bind to it (0.0.0.0:5000->80/tcp)

check if nginx is working (curl)



```
➜ ~ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
4983fa35da84        nginx               "/docker-entrypoint.…"   About an hour ago   Up About an ho
➜ ~ curl http://192.168.99.104:5000
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

check if the restart policy is {always} (overlord) // no (test/debian)



```
➜ ~ "}} yciloPtratseR.gfinoCtsoH. {{" f- tcepsni rekcod
➜ ~ docker inspect -f "{{.HostConfig.RestartPolicy}}" overlord
{always 0}
➜ ~ docker inspect -f "{{.HostConfig.RestartPolicy}}" test
{no 0}
➜ ~
```

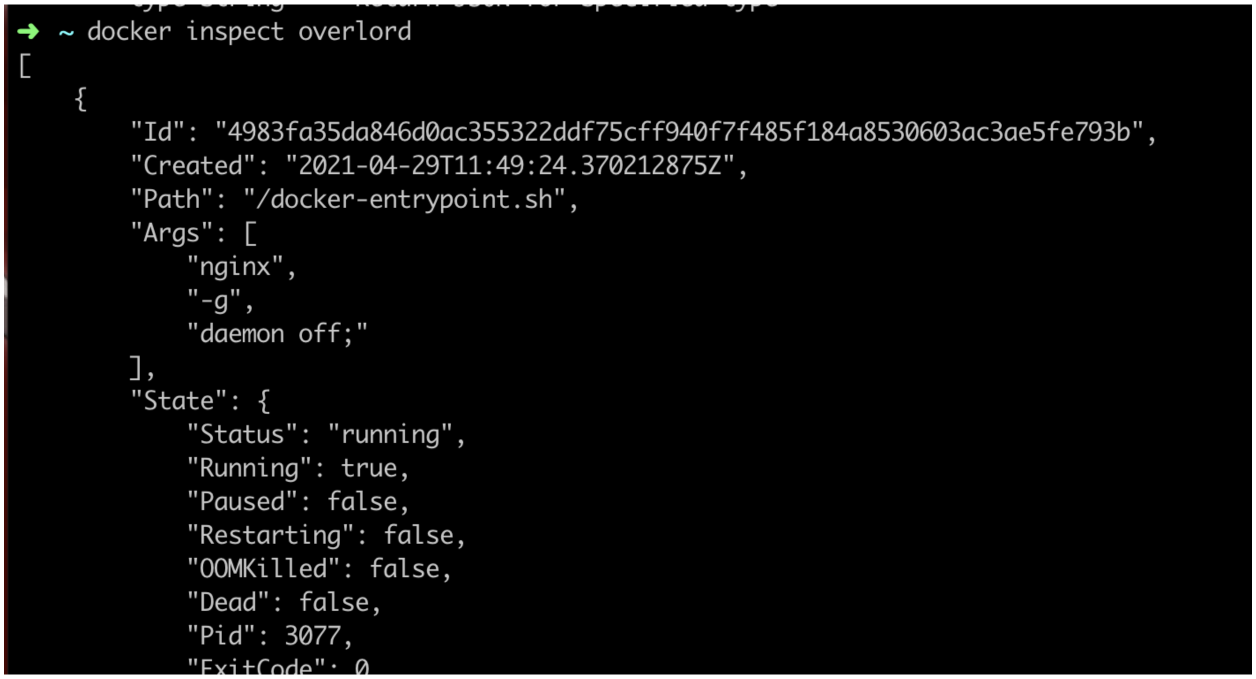
7. Get the internal IP address of the overlord container without starting its shell and in one command.

[How to Get A Docker Container IP Address - Explained with Examples]

Usually Docker uses the default 172.17. 0.0/16 subnet for container networking.

Docker inspect //is a great way to retrieve low-level information on Docker objects

$ docker inspect overlord

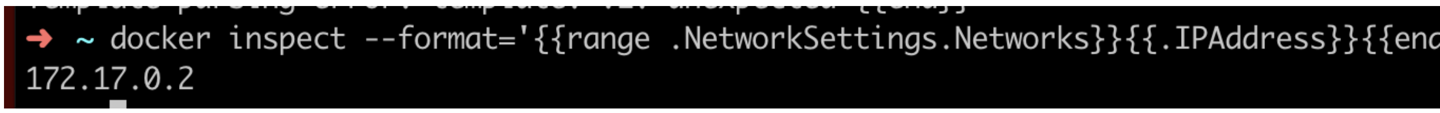


```
➜ ~ docker inspect overlord
[
    {
        "Id": "4983fa35da846d0ac355322ddf75cff940f7f485f184a8530603ac3ae5fe793b",
        "Created": "2021-04-29T11:49:24.370212875Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3077,
            "ExitCode": 0
```

you can pick out any field from the returned JSON

[docker inspect] //here to how to get it

$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' overlord



```
➜ ~ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{enc
172.17.0.2
```

8. Launch a shell from an alpine container, and make sure that you can interact directly with the container via your terminal, and that the container deletes itself once the shell's execution is done.

Alpine Linux is a security-oriented, lightweight Linux distribution based on musl libc and busybox.

Docker run reference

```
-t              : Allocate a pseudo-tty
-i              : Keep STDIN open even if not attached
```

For interactive processes (like a shell), you must use `-i -t` together in order to allocate a tty for the container process. `-i -t` is often written `-it` as you'll see in later examples.

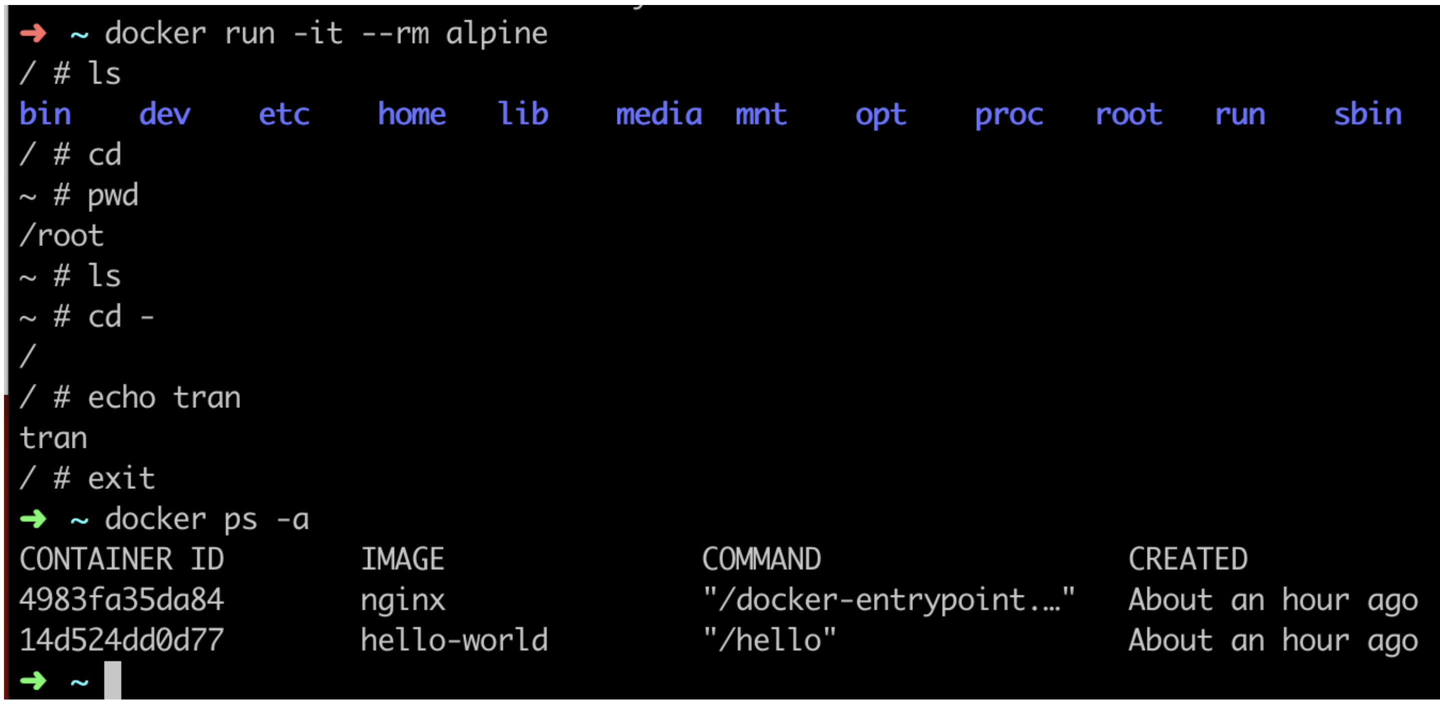
```
docker run --name test -it debian
```

This example runs a container named `test` using the `debian:latest` image. The `-it` instructs Docker to allocate a pseudo-TTY connected to the container's stdin; creating an interactive `bash` shell in the container. In the example, the `bash` shell is quit by entering `exit 13`. This exit code is passed on to the caller of `docker run`, and is recorded in the `test` container's metadata.

--rm   Automatically remove the container when it exits

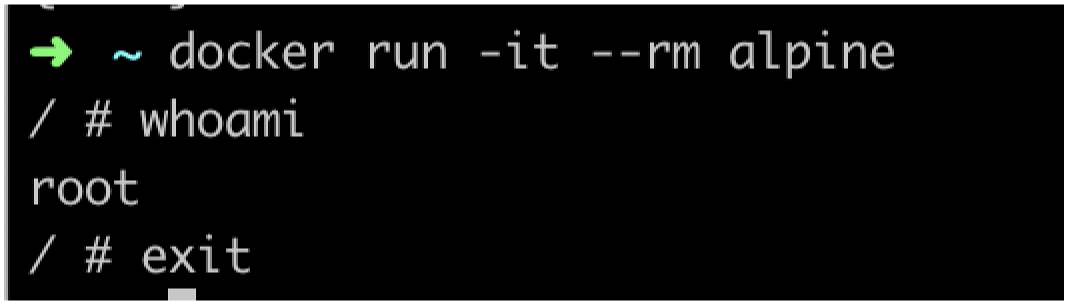
docker run -it --rm alpine

docker ps -a will list all available containers (open/stopped), there is no alpine container, alpine removed after #exit



```
➜ ~ docker run -it --rm alpine
/ # ls
bin     dev     etc     home    lib     media   mnt     opt     proc    root    run     sbin
/ # cd
~ # pwd
/root
~ # ls
~ # cd -
/
/ # echo tran
tran
/ # exit
➜ ~ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
4983fa35da84        nginx               "/docker-entrypoint.…"   About an hour ago
14d524dd0d77        hello-world         "/hello"                 About an hour ago
➜ ~
```

check if you are root on alpine



```
➜ ~ docker run -it --rm alpine
/ # whoami
root
/ # exit
```

9. From the shell of a debian container, install via the container's package manager everything you need to compile C source code and push it onto a git repo (of course, make sure before that the package manager and the packages already in the container are updated). For this exercise, you should only specify the commands to be run directly in the container.

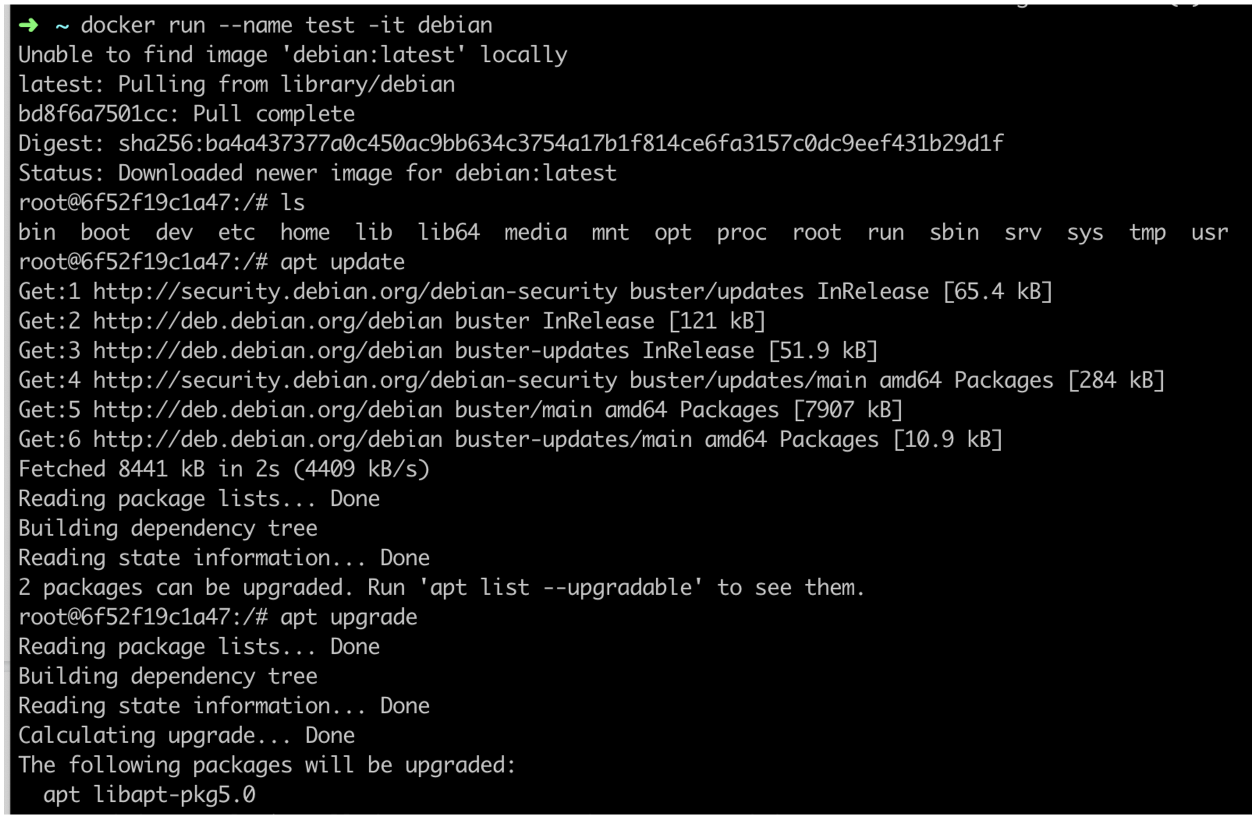
```
docker run --name test -it --rm debian //run the container debian named test
shell
```

apt update && apt upgrade

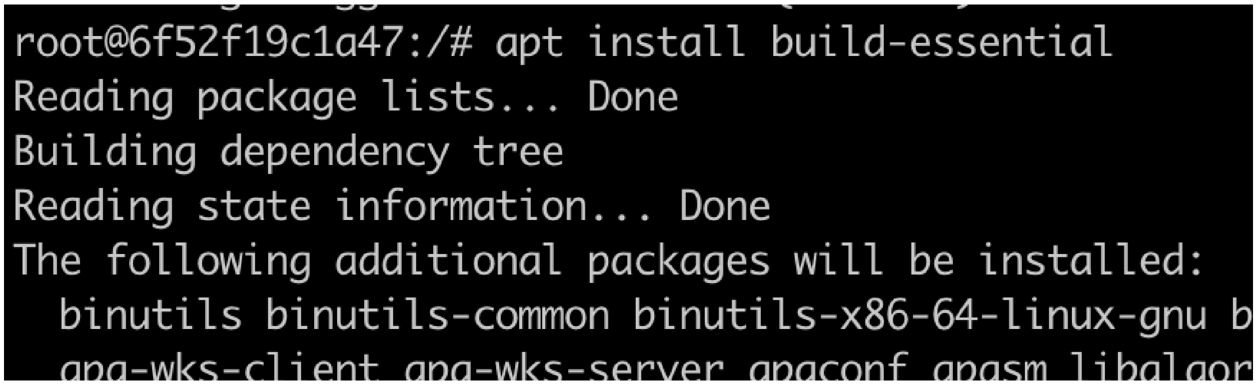
apt install build-essential

apt install gcc

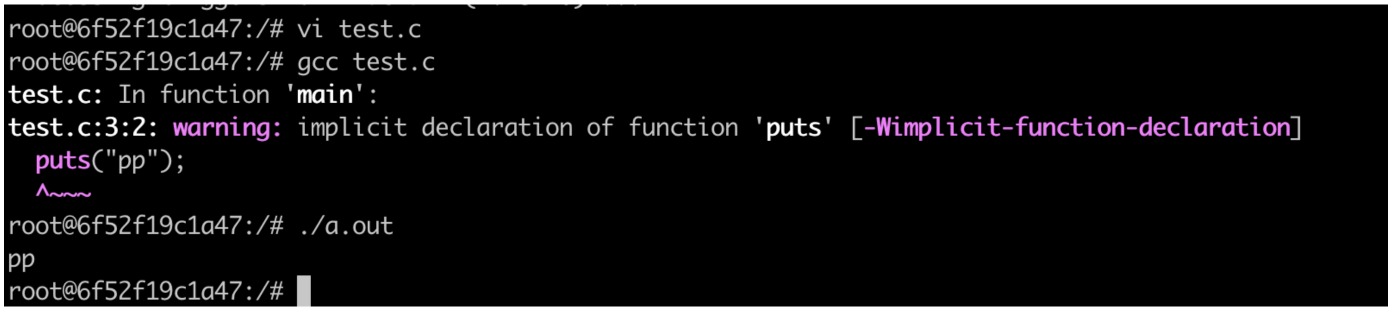
apt   install git



```
➜  ~ docker run --name test -it debian
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
bd8f6a7501cc: Pull complete
Digest: sha256:ba4a437377a0c450ac9bb634c3754a17b1f814ce6fa3157c0dc9eef431b29d1f
Status: Downloaded newer image for debian:latest
root@6f52f19c1a47:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr
root@6f52f19c1a47:/# apt update
Get:1 http://security.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:2 http://deb.debian.org/debian buster InRelease [121 kB]
Get:3 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:4 http://security.debian.org/debian-security buster/updates/main amd64 Packages [284 kB]
Get:5 http://deb.debian.org/debian buster/main amd64 Packages [7907 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [10.9 kB]
Fetched 8441 kB in 2s (4409 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@6f52f19c1a47:/# apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apt libapt-pkg5.0
```

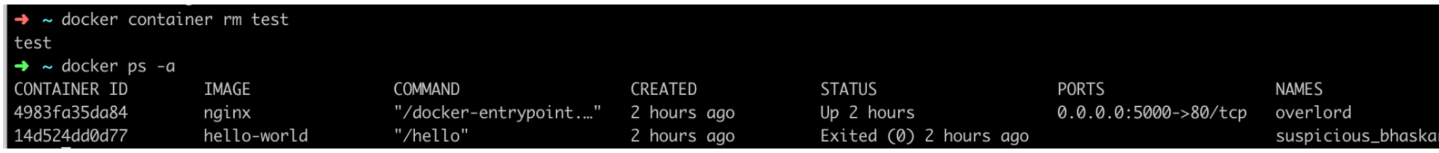


```
root@6f52f19c1a47:/# apt install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu b
  gpg-wks-client gpg-wks-server gpgconf gpgsm libalgor
```



```
root@6f52f19c1a47:/# vi test.c
root@6f52f19c1a47:/# gcc test.c
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'puts' [-Wimplicit-function-declaration]
  puts("pp");
  ^~~~
root@6f52f19c1a47:/# ./a.out
pp
root@6f52f19c1a47:/#
```

1. Use **docker** ps to get the name of the **existing container**.
2. Use the command **docker** exec -it <**container** name> /bin/bash to get a bash shell in the **container**.
3. Or directly use **docker** exec -it <**container** name> <command> to **execute** whatever command you specify in the **container**.
4. remove a container



```
➜ ~ docker container rm test
test
➜ ~ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                   PORTS                  NAMES
4983fa35da84        nginx               "/docker-entrypoint.…"   2 hours ago         Up 2 hours               0.0.0.0:5000->80/tcp   overlord
14d524dd0d77        hello-world         "/hello"                 2 hours ago         Exited (0) 2 hours ago                          suspicious_bhaska
```

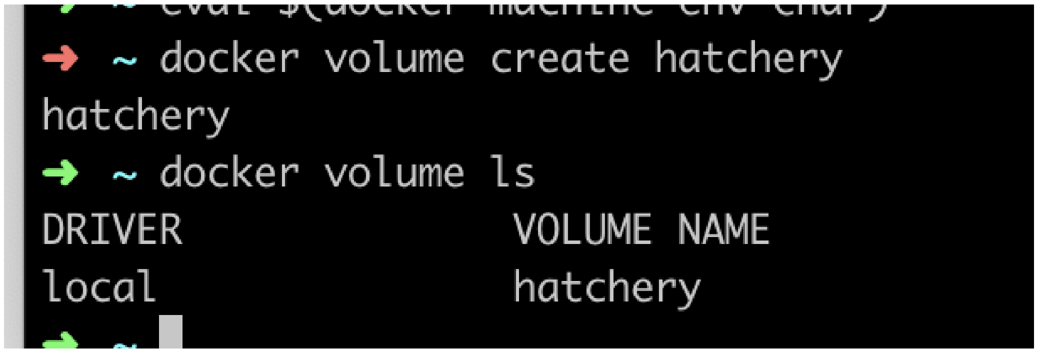
10. Create a volume named hatchery.

11. List all the Docker volumes created on the machine. Remember. VOLUMES.

Use volumes

a volume is a docker memory (outside the containers but accessible by them)

Volumes can be more safely shared among multiple containers.



```
➜  ~ docker volume create hatchery
hatchery
➜  ~ docker volume ls
DRIVER              VOLUME NAME
local               hatchery
➜  ~
```

12. Launch a mysql container as a background task. It should be able to restart on its own in case of error, and the root password of the database should be Kerrigan. You will also make sure that the database is stored in the hatchery volume, that the container directly creates a database named zerglings, and that the container itself is named spawning-pool.

docker run -d --name spawning-pool --restart=on-failure:10//

`on-failure[:max-retries]`	Restart only if the container exits with a non-zero exit status. Optionally, limit the number of restart retries the Docker daemon attempts.

-e //// the operator can set any environment variable in the container

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup.

`--volume , -v`	Bind mount a volume

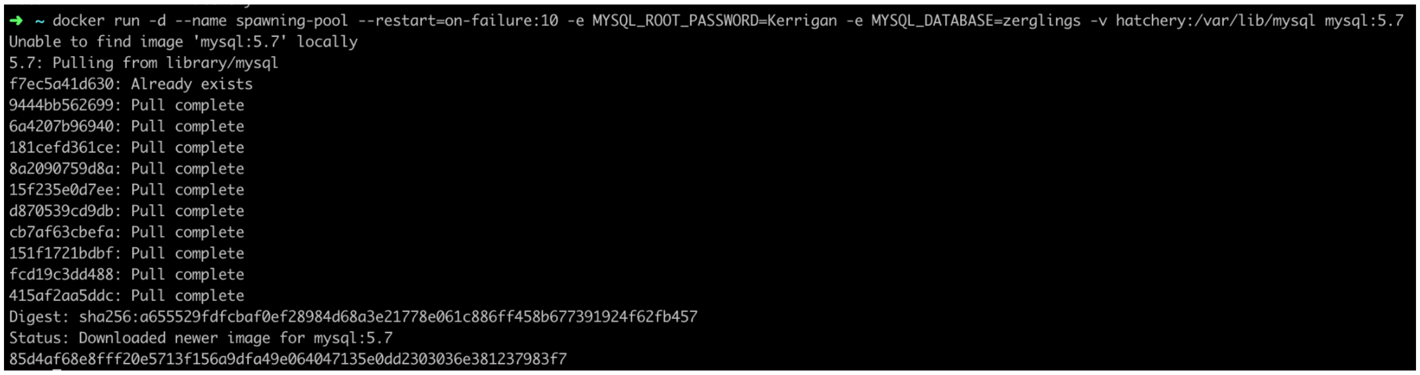
The `-v hatchery:/var/lib/mysql` part of the command mounts the `hatchery` directory from the underlying host system as `/var/lib/mysql` inside the container, where MySQL by default will write its data files.

`5.7` is the tag specifying the MySQL version you want

docker run -d --name spawning-pool --restart=on-failure:10

-e MYSQL_ROOT_PASSWORD=Kerrigan -e MYSQL_DATABASE=zerglings

-v hatchery:/var/lib/mysql mysql:5.7



```
➜ ~ docker run -d --name spawning-pool --restart=on-failure:10 -e MYSQL_ROOT_PASSWORD=Kerrigan -e MYSQL_DATABASE=zerglings -v hatchery:/var/lib/mysql mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
f7ec5a41d630: Already exists
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
cb7af63cbefa: Pull complete
151f1721bdbf: Pull complete
fcd19c3dd488: Pull complete
415af2aa5ddc: Pull complete
Digest: sha256:a655529fdfcbaf0ef28984d68a3e21778e061c886ff458b677391924f62fb457
Status: Downloaded newer image for mysql:5.7
85d4af68e8fff20e5713f156a9dfa49e064047135e0dd2303036e381237983f7
```

13. Print the environment variables of the spawning-pool container in one command, to be sure that you have configured your container properly.

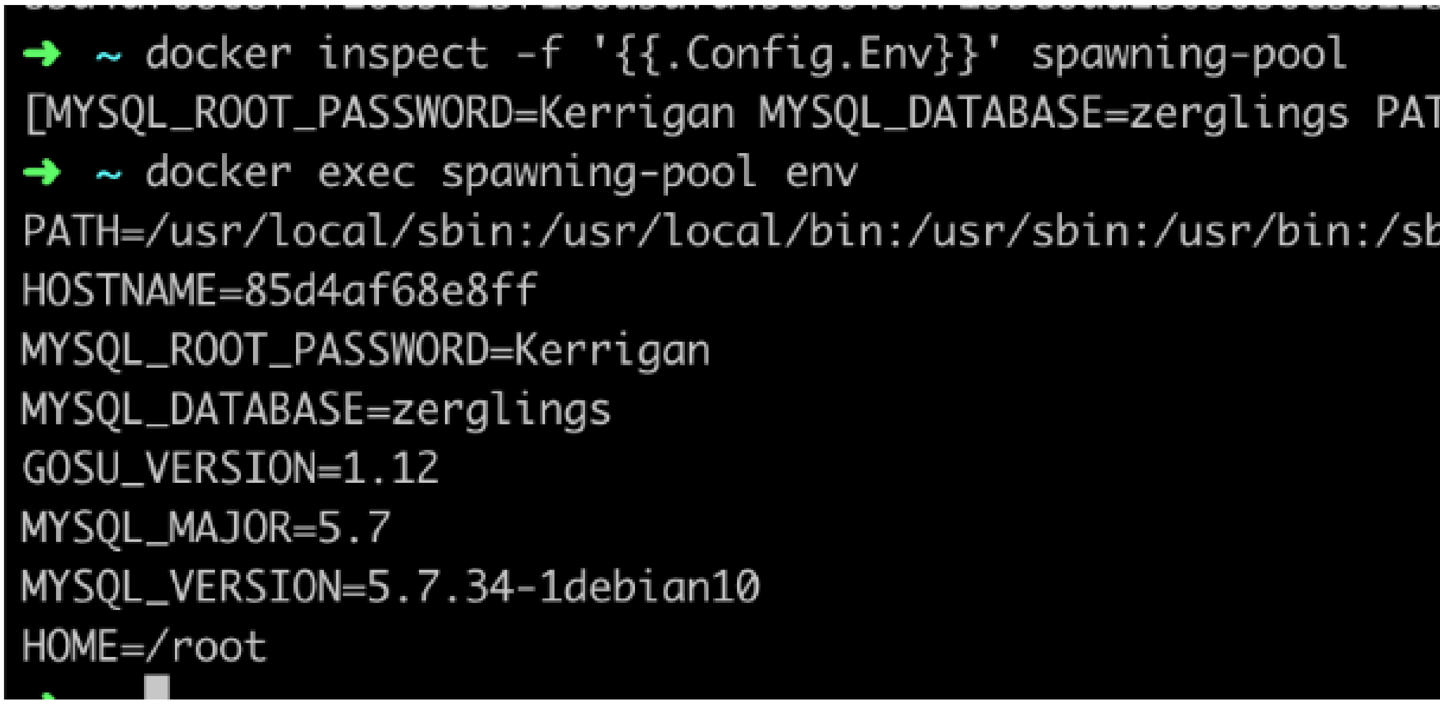
docker inspect -f '{{.Config.Env}}' spawning-pool

or

The docker exec command runs a new command in a running container. (env is the command)

using docker exec

docker exec spawning-pool env



```
➜  ~ docker inspect -f '{{.Config.Env}}' spawning-pool
[MYSQL_ROOT_PASSWORD=Kerrigan MYSQL_DATABASE=zerglings PAT
➜  ~ docker exec spawning-pool env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sb
HOSTNAME=85d4af68e8ff
MYSQL_ROOT_PASSWORD=Kerrigan
MYSQL_DATABASE=zerglings
GOSU_VERSION=1.12
MYSQL_MAJOR=5.7
MYSQL_VERSION=5.7.34-1debian10
HOME=/root
```

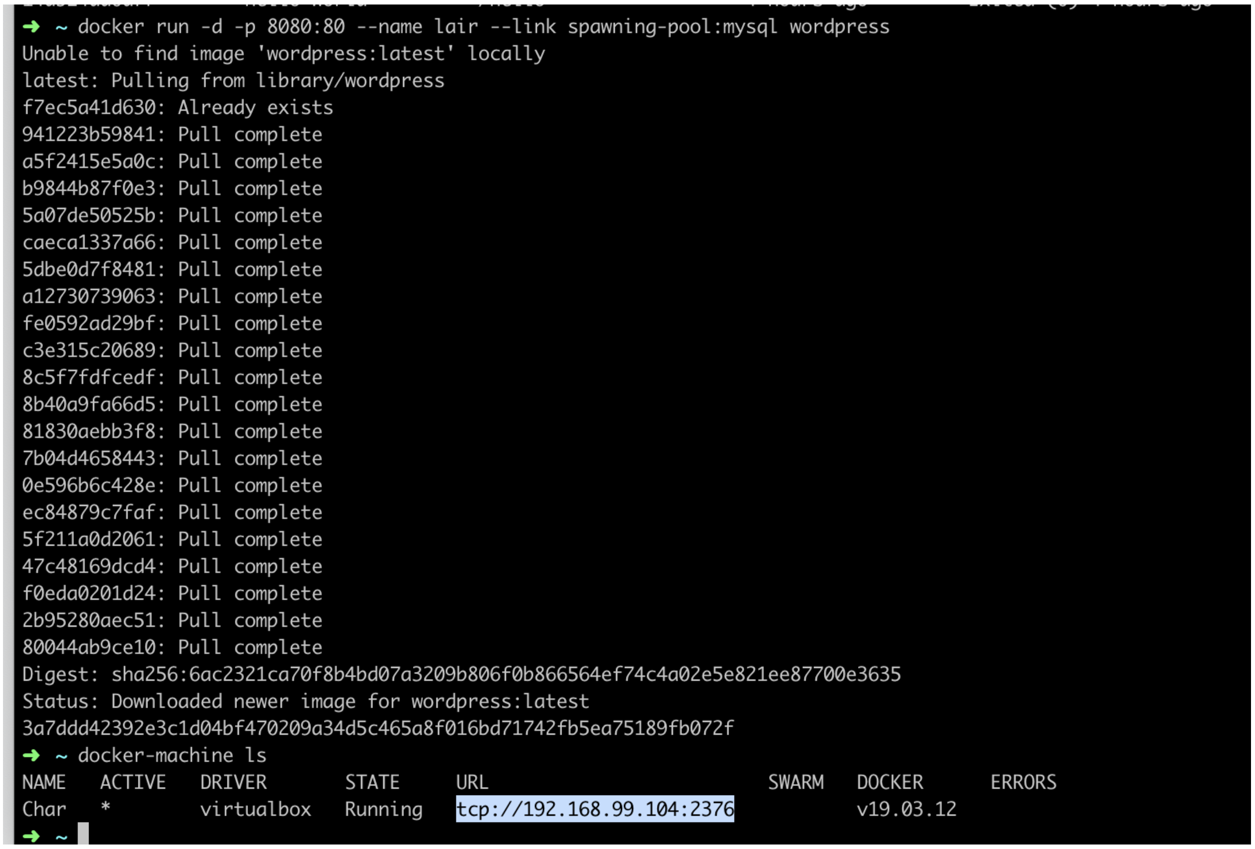
14. Launch a wordpress container as a background task, just for fun. The container should be named lair, its 80 port should be bound to the 8080 port of the virtual machine, and it should be able to use the spawning-pool container as a database service. You can try to access lair on your machine via a web browser, with the IP address of the virtual machine as a URL. Congratulations, you just deployed a functional Wordpress website in two commands!

```
--link=""  : Add link to another container (<name or id>:alias or <name or id>)
```

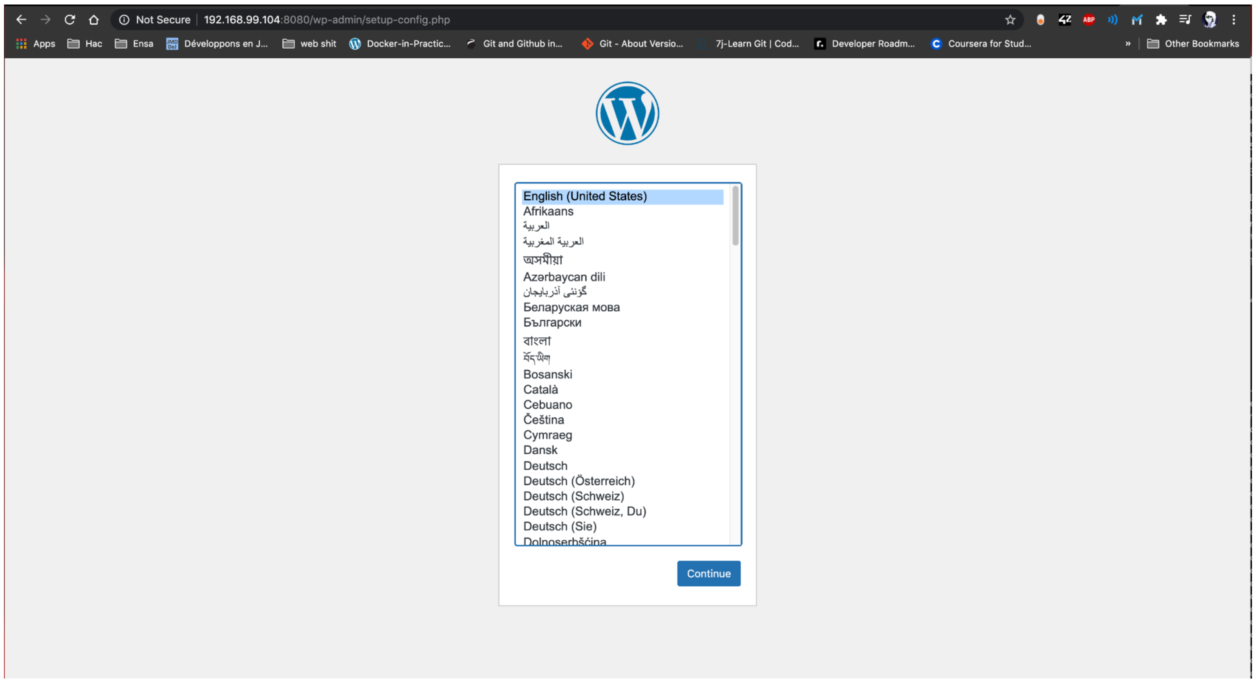
Links allow containers to discover each other and securely transfer information about one container to another container. When you set up a link, you create a conduit between a source container and a recipient container. The recipient can then access select data about the source. To create a link, you use the `--link` flag.

```
docker run -d --name lair -p 8080:80 --link spawning-pool:mysql wordpress
```

wordpress



```
➜ ~ docker run -d -p 8080:80 --name lair --link spawning-pool:mysql wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
f7ec5a41d630: Already exists
941223b59841: Pull complete
a5f2415e5a0c: Pull complete
b9844b87f0e3: Pull complete
5a07de50525b: Pull complete
caeca1337a66: Pull complete
5dbe0d7f8481: Pull complete
a12730739063: Pull complete
fe0592ad29bf: Pull complete
c3e315c20689: Pull complete
8c5f7fdfcedf: Pull complete
8b40a9fa66d5: Pull complete
81830aebb3f8: Pull complete
7b04d4658443: Pull complete
0e596b6c428e: Pull complete
ec84879c7faf: Pull complete
5f211a0d2061: Pull complete
47c48169dcd4: Pull complete
f0eda0201d24: Pull complete
2b95280aec51: Pull complete
80044ab9ce10: Pull complete
Digest: sha256:6ac2321ca70f8b4bd07a3209b806f0b866564ef74c4a02e5e821ee87700e3635
Status: Downloaded newer image for wordpress:latest
3a7ddd42392e3c1d04bf470209a34d5c465a8f016bd71742fb5ea75189fb072f
➜ ~ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
Char   *        virtualbox   Running   tcp://192.168.99.104:2376           v19.03.12
➜ ~
```



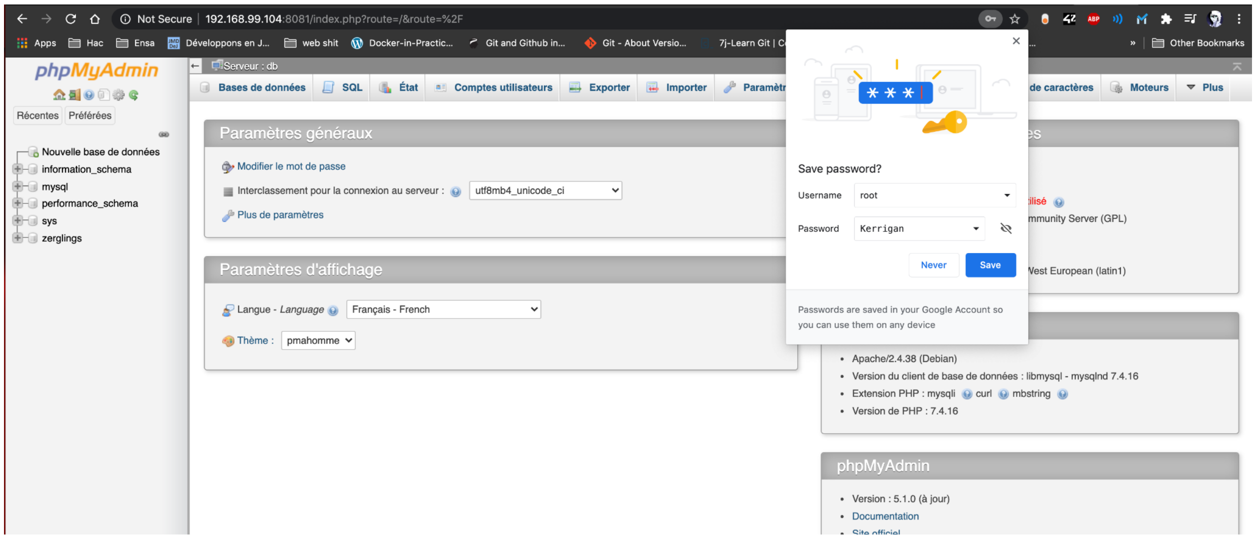
Not Secure | 192.168.99.104:8080/wp-admin/setup-config.php

```
http://192.168.99.104:8080
```

15. Launch a phpmyadmin container as a background task. It should be named roach-warden, its 80 port should be bound to the 8081 port of the virtual machine and it should be able to explore the database stored in the spawning-pool container.

```
docker run --name myadmin -d --link mysql_db_server:db -p 8080:80 phpmyadmin
```

docker run -d -p 8081:80 --name roach-warden --link spawning-pool:db phpmyadmin/phpmyadmin



Not Secure | 192.168.99.104:8081/index.php?route=/&route=%2F

Serveur : db

Paramètres généraux

Modifier le mot de passe

Interclassement pour la connexion au serveur : utf8mb4_unicode_ci

Plus de paramètres

Paramètres d'affichage

Langue - *Language*: Français - French

Thème : pmahomme

Save password?

Username: root

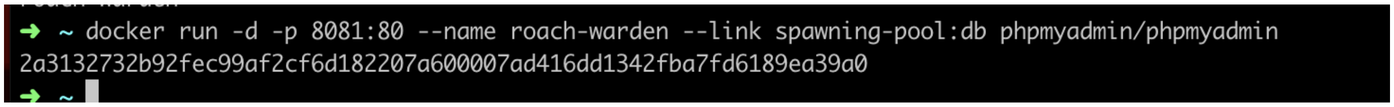
Password: Kerrigan

Passwords are saved in your Google Account so you can use them on any device

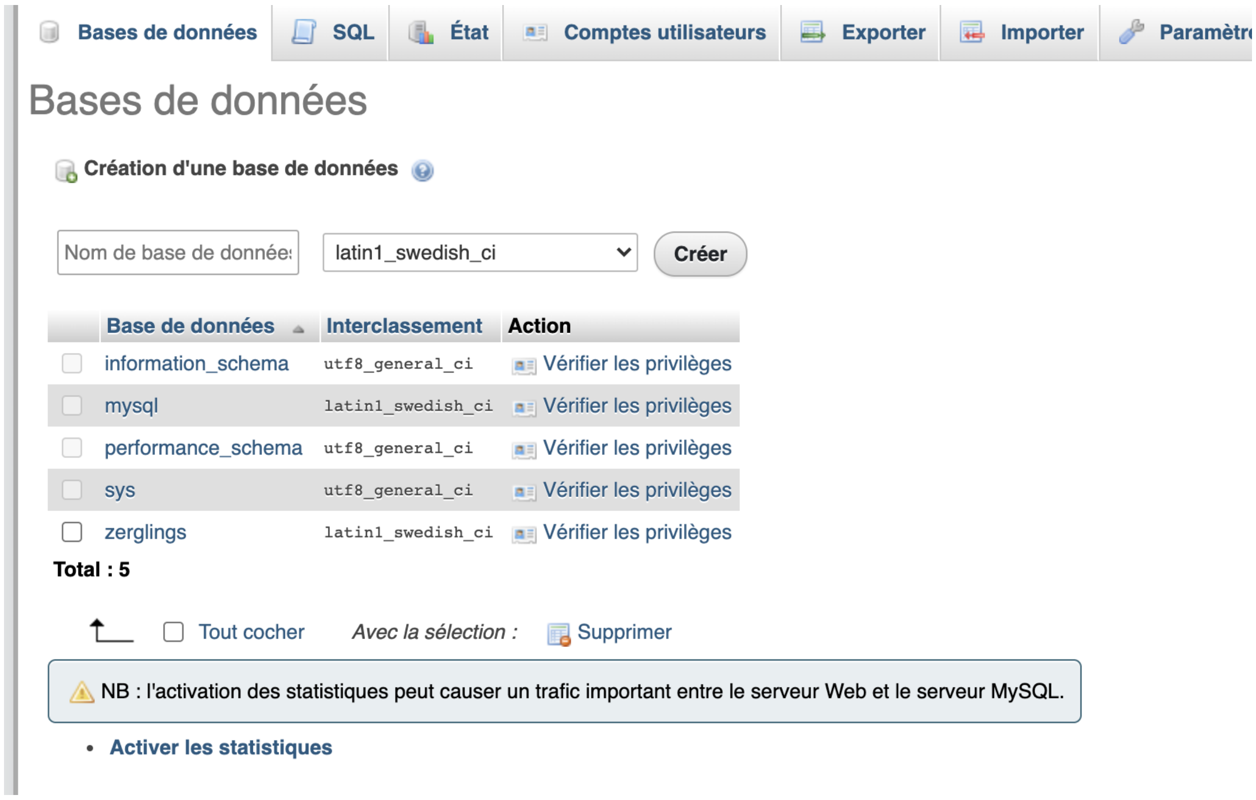
- Apache/2.4.38 (Debian)
- Version du client de base de données : libmysql - mysqlnd 7.4.16
- Extension PHP : mysqli curl mbstring
- Version de PHP : 7.4.16

phpMyAdmin

- Version : 5.1.0 (à jour)
- Documentation



```
➜ ~ docker run -d -p 8081:80 --name roach-warden --link spawning-pool:db phpmyadmin/phpmyadmin
2a3132732b92fec99af2cf6d182207a600007ad416dd1342fba7fd6189ea39a0
➜ ~
```



Bases de données

Création d'une base de données

Nom de base de donnée: | latin1_swedish_ci | Créer

	Base de données	Interclassement	Action
☐	information_schema	utf8_general_ci	Vérifier les privilèges
☐	mysql	latin1_swedish_ci	Vérifier les privilèges
☐	performance_schema	utf8_general_ci	Vérifier les privilèges
☐	sys	utf8_general_ci	Vérifier les privilèges
☐	zerglings	latin1_swedish_ci	Vérifier les privilèges

Total : 5

☐ Tout cocher *Avec la sélection :* Supprimer

NB : l'activation des statistiques peut causer un trafic important entre le serveur Web et le serveur MySQL.

- Activer les statistiques

[phpmyadmin/phpmyadmin]()

II. Insérez votre texte ici

Insére

III. Insérez votre texte ici

Insére

IV. Insérez votre texte ici

Insére

V. Insérez votre texte ici

Insére