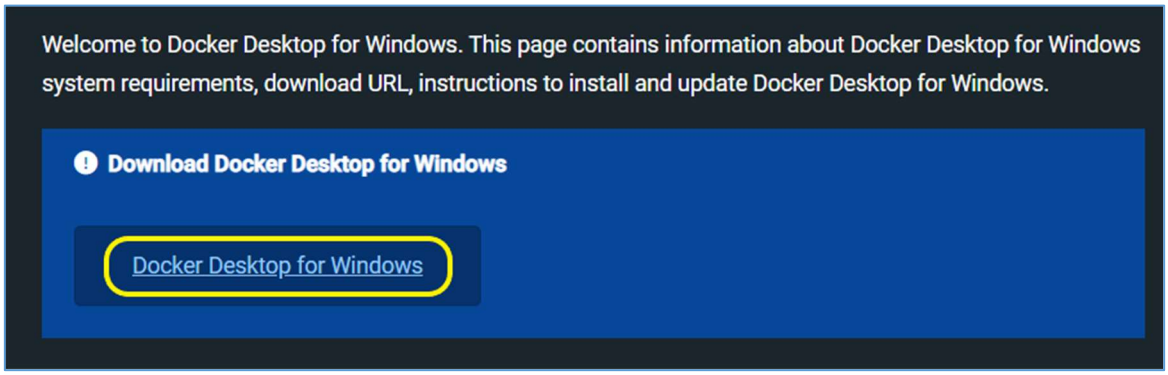
Docker

1. Download and install Docker Desktop on Windows x64 and Docker Engine on Ubuntu and Amazon Linux 2.
2. Run first container “hello-world”.
3. Create first image(based on Ubuntu 20.04).
4. Create first image(based on Centos 7).
5. Create a Docker image which will run a Flask app.
6. Frequently used Docker commands.
7. Docker Compose.
8. Docker Compose project example.
9. Docker Compose. Use volumes.
10. Docker Compose. Use networks.

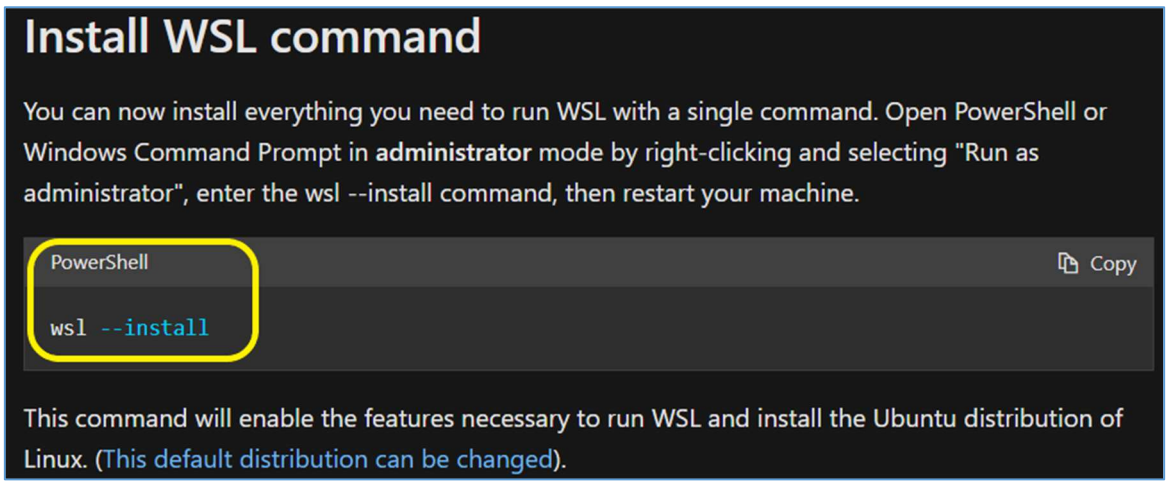
1. Download and install Docker Desktop on Windows x64 and Docker Engine on Ubuntu and Amazon Linux 2.

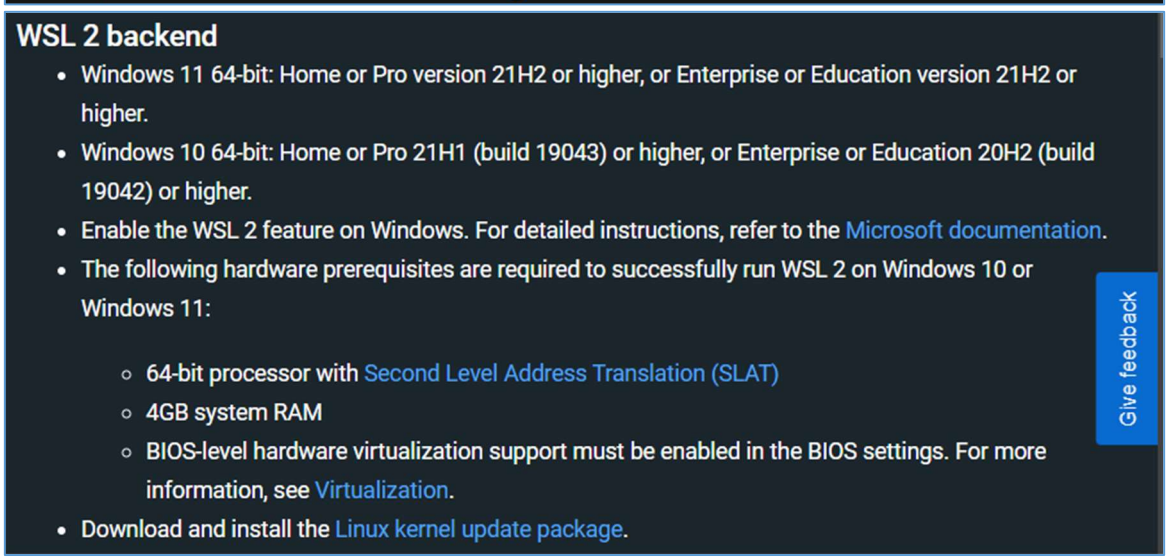
- **Windows**

a. Download newest version Docker Desktop from website



b. Before installing Docker Desktop you have to install WSL 2 backend(Windows Subsystem for Linux version 2)





c. Install Docker Desktop on Windows

Install Docker Desktop on Windows

Install interactively

1. Double-click **Docker Desktop Installer.exe** to run the installer.

   If you haven't already downloaded the installer (`Docker Desktop Installer.exe`), you can get it from **Docker Hub**. It typically downloads to your `Downloads` folder, or you can run it from the recent downloads bar at the bottom of your web browser.

2. When prompted, ensure the **Use WSL 2 instead of Hyper-V** option on the Configuration page is selected or not depending on your choice of backend.

   If your system only supports one of the two options, you will not be able to select which backend to use.

3. Follow the instructions on the installation wizard to authorize the installer and proceed with the install.

4. When the installation is successful, click **Close** to complete the installation process.

5. If your admin account is different to your user account, you must add the user to the **docker-users** group. Run **Computer Management** as an **administrator** and navigate to **Local Users and Groups** > **Groups** > **docker-users**. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

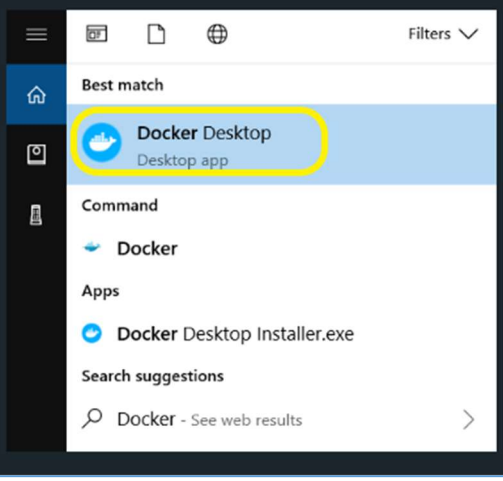
Give feedback

d. Start Docker Desktop

Start Docker Desktop

Docker Desktop does not start automatically after installation. To start Docker Desktop:

1. Search for Docker, and select **Docker Desktop** in the search results.



- **Ubuntu**

  Install using the repository.
  Set up the repository

  a. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

```
$  sudo apt-get update
$  sudo apt-get install \
          ca-certificates \
          curl \
          gnupg \
          lsb-release
```

  b. Add Docker's official GPG key:

```
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

  c. Use the following command to set up the repository:

```
$ echo \
   "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

  Install Docker Engine

  d. Update the apt package index:

```
$ sudo apt-get update
```

  e. Install Docker Engine, containerd, and Docker Compose.

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

  f. Verify that the Docker installation is successful by running the hello-world image

```
$ sudo docker run hello-world
```

- **Amazon Linux**

  Install Docker.

  a. Install Docker with the command's below:

```
$ sudo yum update
$ sudo yum search docker
$ sudo yum install docker
```

  b. Add group membership for the default ec2-user so you can run docker commands.

```
$ sudo usermod -a -G docker ec2-user
$ id ec2-user
```

  c. Enable and start docker service.

```
$ sudo systemctl enable docker.service
$ sudo systemctl start docker.service
```

  d. Check docker service status, run and version:

```
$ sudo systemctl status docker.service
$ docker version
```

  Install docker-compose.

  a. Install using pip (recommended):

```
$ sudo pip3 install docker-compose
```

  b. Manually install

```
$ wget
https://github.com/docker/compose/releases/latest/download/do
cker-compose-$(uname -s)-$(uname -m)
$ sudo mv docker-compose-$(uname -s)-$(uname -m)
/usr/local/bin/docker-compose
$ sudo chmod -v +x /usr/local/bin/docker-compose
```

  c. To verify that the installation was successful, you can use the command

```
$ docker-compose version
```

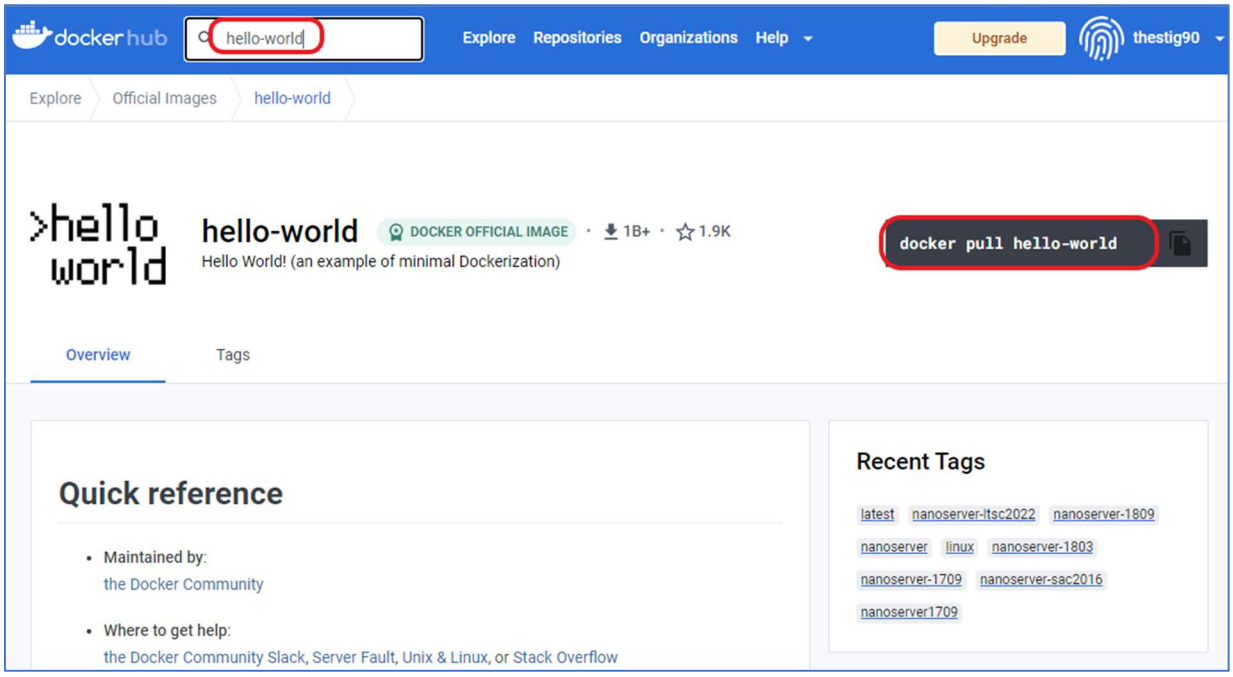
  Bonus: install ctop.

ctop provides a concise and condensed overview of real-time metrics for multiple containers

```
$ sudo wget
https://github.com/bcicen/ctop/releases/download/v0.7.7/ctop-
0.7.7-linux-amd64 -O /usr/local/bin/ctop
$ sudo chmod +x /usr/local/bin/ctop
```

2. Run first container “hello-world”.

Command `$ sudo docker run hello-world` pulls image “hello-world” from Docker Hub Registry and run container with console output.




```
vagrant@vm1:~$ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:c77be1d3a47d0caf71a82dd893ee61ce01f32fc758031a6ec4cf1389248bb833
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/

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

vagrant@vm1:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
41ae44923430   hello-world   "/hello"   2 minutes ago   Exited (0) 2 minutes ago             dazzling_arding
helli
```

```
vagrant@vm1:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   15 months ago   13.3kB
vagrant@vm1:~$
```

If you would like to use Docker as a **non-root user** you should now consider adding your user to the **“docker” group** with command:

```
$ sudo usermod -aG docker “user_name
```

3. Create first image(based on Ubuntu 20.04).

- **Create image**

```
vagrant@vm1:~$
vagrant@vm1:~$ mkdir dockerfiles
vagrant@vm1:~$ cd dockerfiles/
vagrant@vm1:~/dockerfiles$ touch Dockerfile
vagrant@vm1:~/dockerfiles$ vim Dockerfile
```

```
FROM ubuntu:20.04

ENV TZ=Europe/Kiev
RUN apt-get -y update
RUN DEBIAN_FRONTEND="noninteractive" \
    apt-get -y install apache2
RUN echo "Hi there, what is love?" > /var/www/html/index.html
RUN echo "It is a song..." >> /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-DFOREGROUND"]
~
~
```

```
vagrant@vm1:~/dockerfiles$ ll
total 12
drwxrwxr-x 2 vagrant vagrant 4096 Dec 21 16:50 ./
drwxr-xr-x 6 vagrant vagrant 4096 Dec 21 16:50 ../
-rw-rw-r-- 1 vagrant vagrant  305 Dec 21 16:45 Dockerfile
```

```
vagrant@vm1:~/dockerfiles$ docker image build -t vagrant:v1.0 .
Sending build context to Docker daemon  2.048kB
Step 1/8 : FROM ubuntu:20.04
 ---> d5447fc01ae6
Step 2/8 : ENV TZ=Europe/Kiev
 ---> Using cache
 ---> 17acc0983781
Step 3/8 : RUN apt-get -y update
 ---> Running in 309c35b6f014
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [1779 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [27.7 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [2358 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [972 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
```

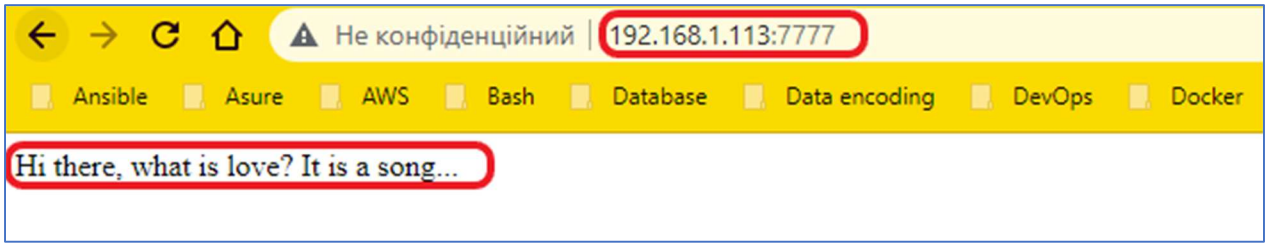
```
Step 7/8 : EXPOSE 80
 ---> Running in 712debe70bb2
Removing intermediate container 712debe70bb2
 ---> 8ed0394232b4
Step 8/8 : CMD ["/usr/sbin/apache2ctl", "-DFOREGROUND"]
 ---> Running in 1d96f36e7ca7
Removing intermediate container 1d96f36e7ca7
 ---> 55f6da8b8329
Successfully built 55f6da8b8329
Successfully tagged vagrant:v1.0
```

```
vagrant@vm1:~/dockerfiles$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
vagrant       v1.0      55f6da8b8329   2 minutes ago   226MB
ubuntu        20.04     d5447fc01ae6   12 days ago     72.8MB
hello-world   latest    feb5d9fea6a5   15 months ago   13.3kB
```

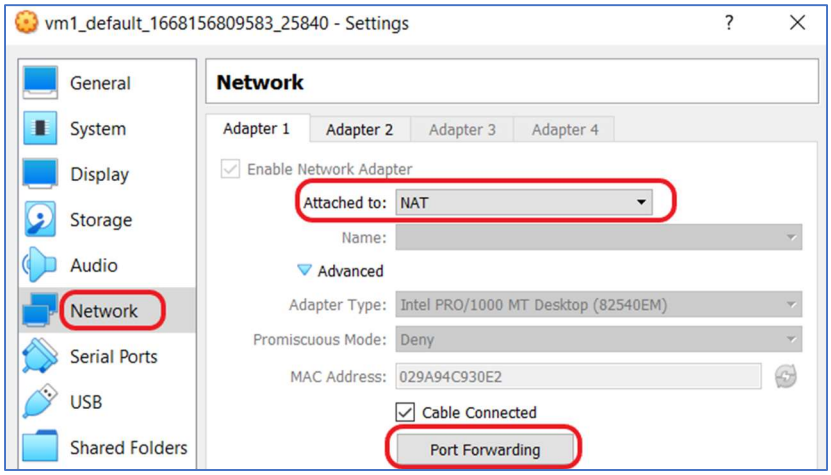
- **Run container**

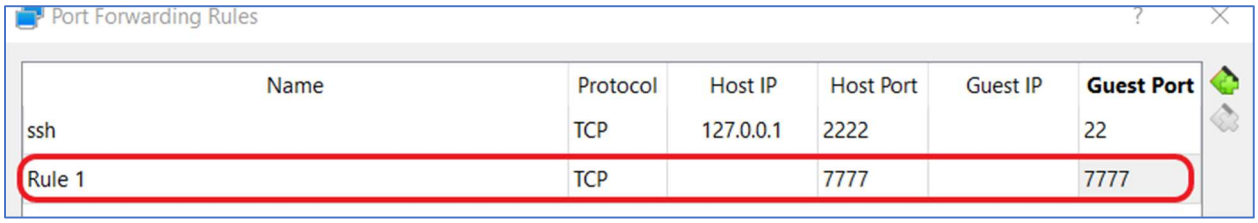
```
vagrant@vm1:~/dockerfiles$ docker container run -d -p 7777:80 vagrant:v1.0
bcbe95467a60fe1a96d3391ace5e8cf7a2468e71f7a7cb6ff2940330a1db8640
vagrant@vm1:~/dockerfiles$ docker container ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS
                  NAMES
bcbe95467a60   vagrant:v1.0   "/usr/sbin/apache2ct…"   8 seconds ago   Up 7 seconds   0.0.0.0:7777->80/tcp,
:::7777->80/tcp   silly_fermi
vagrant@vm1:~/dockerfiles$
```

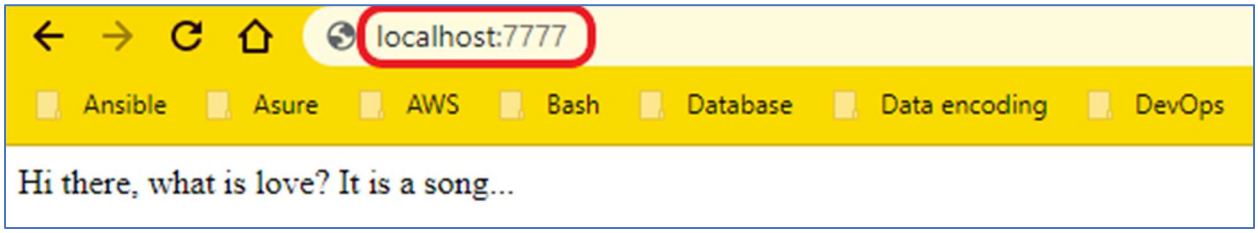
```
vagrant@vm1:~/dockerfiles$ hostname -I
10.0.2.15 192.168.1.113 172.17.0.1
vagrant@vm1:~/dockerfiles$
```




```
vagrant@vm1:~/dockerfiles$ docker container inspect bcbe95467a60
[
    {
    "Networks": {
        "bridge": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
            "NetworkID": "c3843f5fa298874af3cfe02b19666feedcf341ed67264ab3f820c24a465cc11c",
            "EndpointID": "4b0f41a675e8e5fcd9785fd2743cb8c8811093ffc41cba7c7040565150afd9a1",
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
```







- **Stop and delete all stoped container**

```
vagrant@vm1:~/dockerfiles$ docker container stop bcbe95467a60
bcbe95467a60
vagrant@vm1:~/dockerfiles$ docker container ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
vagrant@vm1:~/dockerfiles$ docker container ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
    NAMES
bcbe95467a60   vagrant:v1.0   "/usr/sbin/apache2ct…"   27 minutes ago   Exited (137) 22 seconds ago
    silly_fermi
6907a9bb20c1   17acc0983781   "/bin/sh -c apt-get-…"   36 minutes ago   Exited (127) 36 minutes ago
    quizzical_carson
41ae44923430   hello-world    "/hello"                 3 hours ago      Exited (0) 3 hours ago
    dazzling_ardinghelli
```

```
vagrant@vm1:~/dockerfiles$ docker container

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
vagrant@vm1:~/dockerfiles$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
bcbe95467a60fe1a96d3391ace5e8cf7a2468e71f7a7cb6ff2940330a1db8640
6907a9bb20c1f100fcfead27c0ed75b4b70bf16b6870b42efc1107175216b24c
41ae449234301cd63e6bff07e3b45abcaab8ab40bce7cc23bcd28e0cba3a41ff

Total reclaimed space: 1.2kB
vagrant@vm1:~/dockerfiles$ docker container ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

4. Create first image(based on Centos 7).

- **Create image**

```
[vagrant@vm2 ~]$
[vagrant@vm2 ~]$ mkdir dockerfiles
[vagrant@vm2 ~]$ cd dockerfiles/
[vagrant@vm2 dockerfiles]$ touch Dockerfile
[vagrant@vm2 dockerfiles]$ vim Dockerfile
```

```
FROM centos:7

RUN yum -y update
RUN yum -y install httpd
RUN echo "Hi there, what is love?" > /var/www/html/index.html
RUN echo "It is a song..." >> /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
~
~
```

```
[vagrant@vm2 dockerfiles]$ ll
total 4
-rw-rw-r--. 1 vagrant vagrant 227 Dec 22 20:38 Dockerfile
```

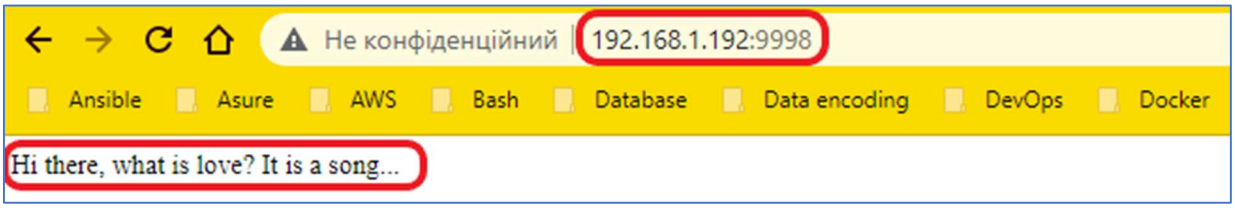
```
[vagrant@vm2 dockerfiles]$ docker image build -t vagrant:v2.0 .
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM centos:7
7: Pulling from library/centos
2d473b07cdd5: Pull complete
Digest: sha256:be65f488b7764ad3638f236b7b515b3678369a5124c47b8d32916d6487418ea4
Status: Downloaded newer image for centos:7
 ---> eeb6ee3f44bd
Step 2/7 : RUN yum -y update
 ---> Running in 0da7bb65b100
Loaded plugins: fastestmirror, ovl
Removing intermediate container 5ff7901df95d
 ---> 37f94dc3de6c
Step 4/7 : RUN echo "Hi there, what is love?" > /var/www/html/index.html
 ---> Running in 5da2c409ba79
Removing intermediate container 5da2c409ba79
 ---> ed05cd2c0b0f
Step 5/7 : RUN echo "It is a song..." >> /var/www/html/index.html
 ---> Running in 30d5caa02edf
Removing intermediate container 30d5caa02edf
 ---> 96f2699603b2
Step 6/7 : EXPOSE 80
 ---> Running in 63bc04e644f0
Removing intermediate container 63bc04e644f0
 ---> 1e499047b307
Step 7/7 : CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
 ---> Running in 3b1b1da78b15
Removing intermediate container 3b1b1da78b15
 ---> f0bba1ad87aa
Successfully built f0bba1ad87aa
Successfully tagged vagrant:v2.0
```

```
[vagrant@vm2 dockerfiles]$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
vagrant      v2.0      f0bba1ad87aa   4 minutes ago    761MB
centos       7         eeb6ee3f44bd   15 months ago    204MB
[vagrant@vm2 dockerfiles]$
```

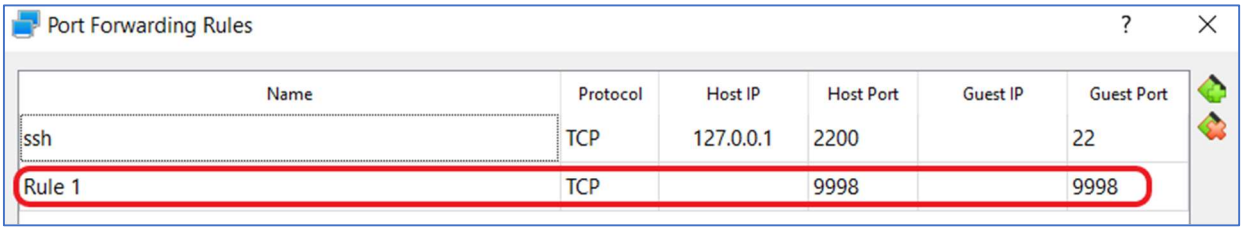
- **Run container**

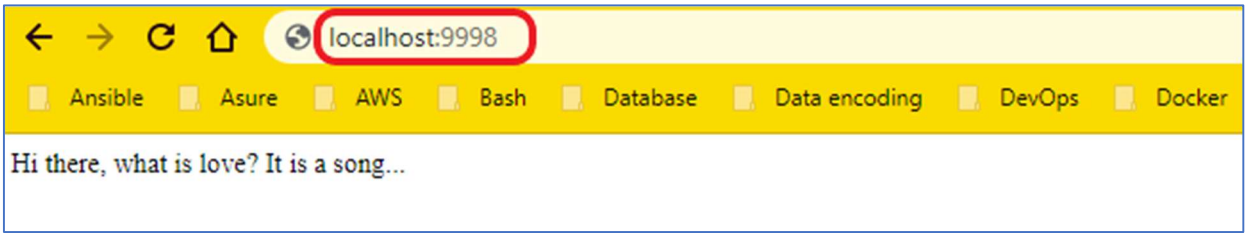
```
[vagrant@vm2 dockerfiles]$ docker container run -d -p 9998:80 vagrant:v2.0
71bb963a3d03cf1db963da00e28bd1302b9f8d277c47652bc51d8ca7ae3bca72
[vagrant@vm2 dockerfiles]$ docker container ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS
                    NAMES
71bb963a3d03   vagrant:v2.0   "/usr/sbin/httpd -DF…"   12 seconds ago   Up 11 seconds   0.0.0.0:9998
->80/tcp, :::9998->80/tcp   reverent_galileo
[vagrant@vm2 dockerfiles]$
```

```
[vagrant@vm2 dockerfiles]$ hostname -I
10.0.2.15 192.168.1.192 172.17.0.1
[vagrant@vm2 dockerfiles]$
```




```
[vagrant@vm2 dockerfiles]$ docker container inspect 71bb963a3d03
[
    {
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "8ab1cb18862cff7139fe254c48d8f16b1b61e307aaf4bc62e6f193dc0e9305cb",
        "EndpointID": "6838c4ff57d4c7e2a9aa54b99eeedfc720eb2cd5f4c56b37baf242b7f3a20d57"

        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
```





- **Stop and delete all stoped container**

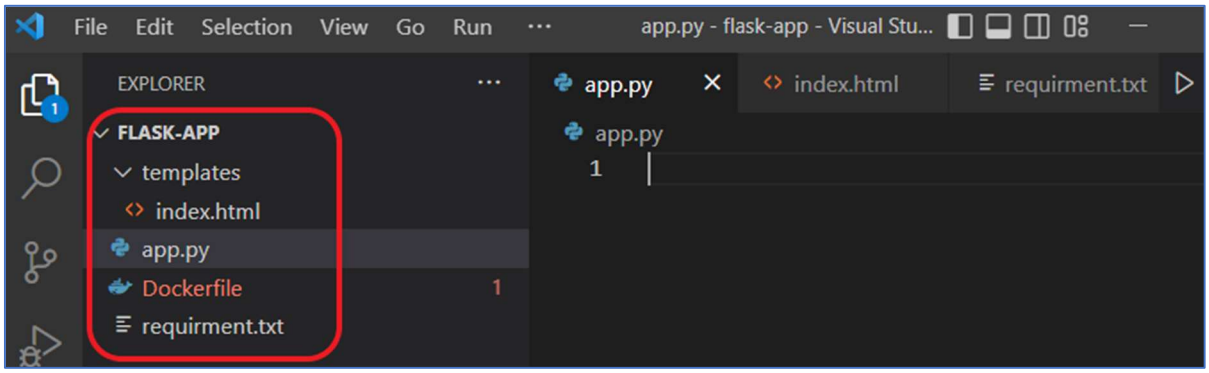
```
[vagrant@vm2 dockerfiles]$ docker container stop 71bb963a3d03
71bb963a3d03
[vagrant@vm2 dockerfiles]$ docker container ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[vagrant@vm2 dockerfiles]$ docker container ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
ORTS     NAMES
71bb963a3d03   vagrant:v2.0   "/usr/sbin/httpd -DF…"   18 minutes ago   Exited (0) 7 seconds ago
         reverent_galileo
[vagrant@vm2 dockerfiles]$
```

```
[vagrant@vm2 dockerfiles]$ docker container

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
[vagrant@vm2 dockerfiles]$
```

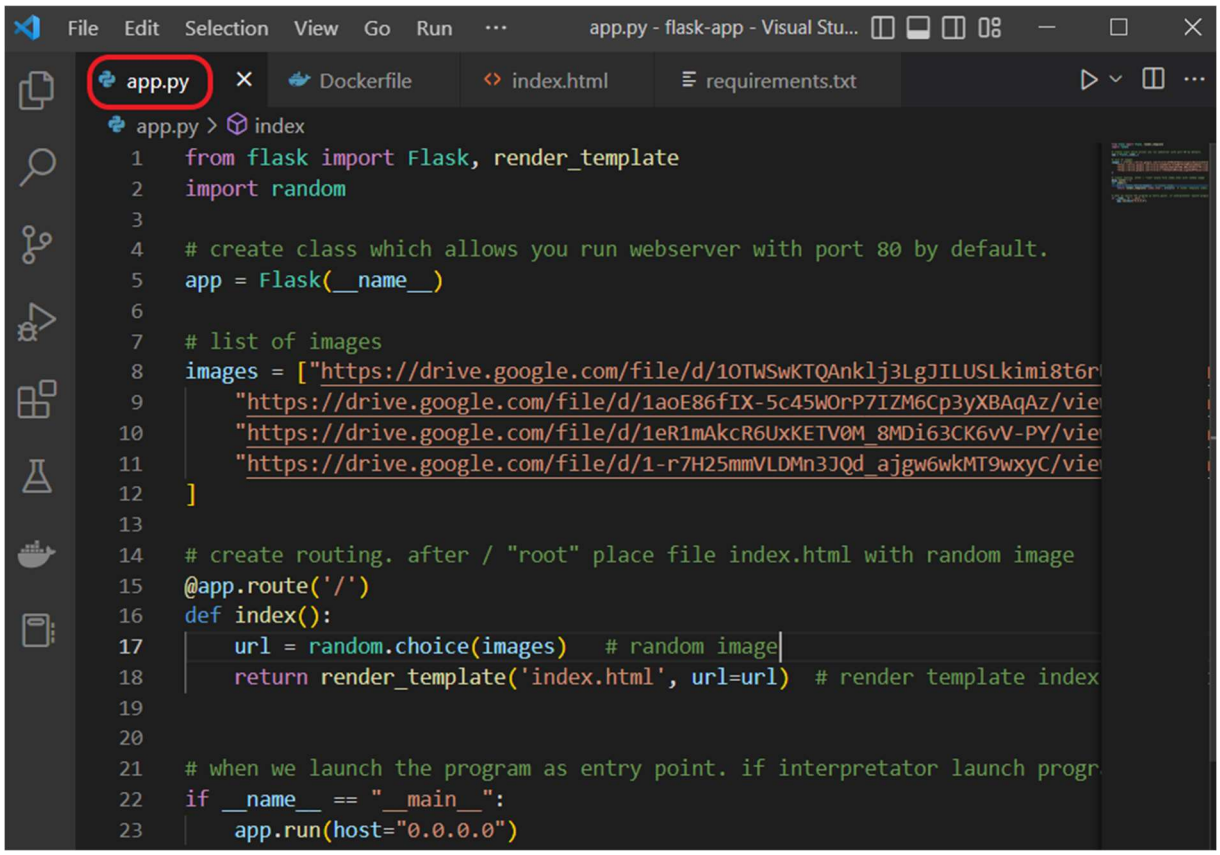
```
[vagrant@vm2 dockerfiles]$
[vagrant@vm2 dockerfiles]$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
71bb963a3d03cf1db963da00e28bd1302b9f8d277c47652bc51d8ca7ae3bca72

Total reclaimed space: 1.483kB
[vagrant@vm2 dockerfiles]$
[vagrant@vm2 dockerfiles]$ docker container ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[vagrant@vm2 dockerfiles]$
```

5. Create a Docker image which will run a Flask app.

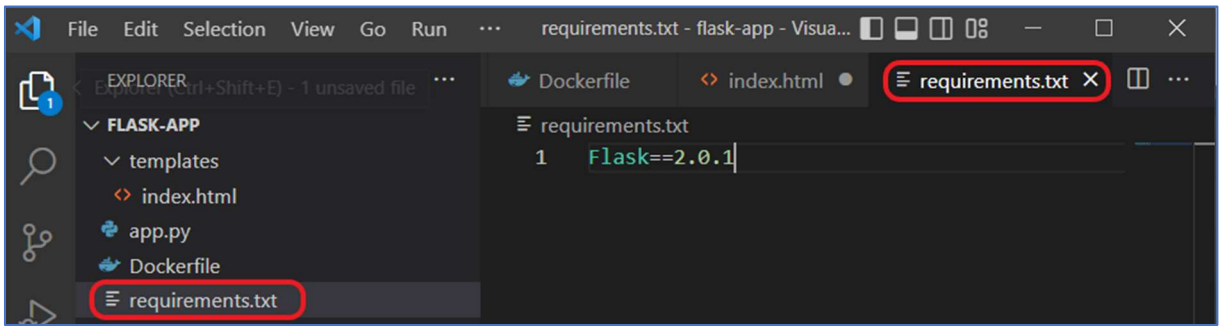
- Creating a directory called **flask-app** with following files:



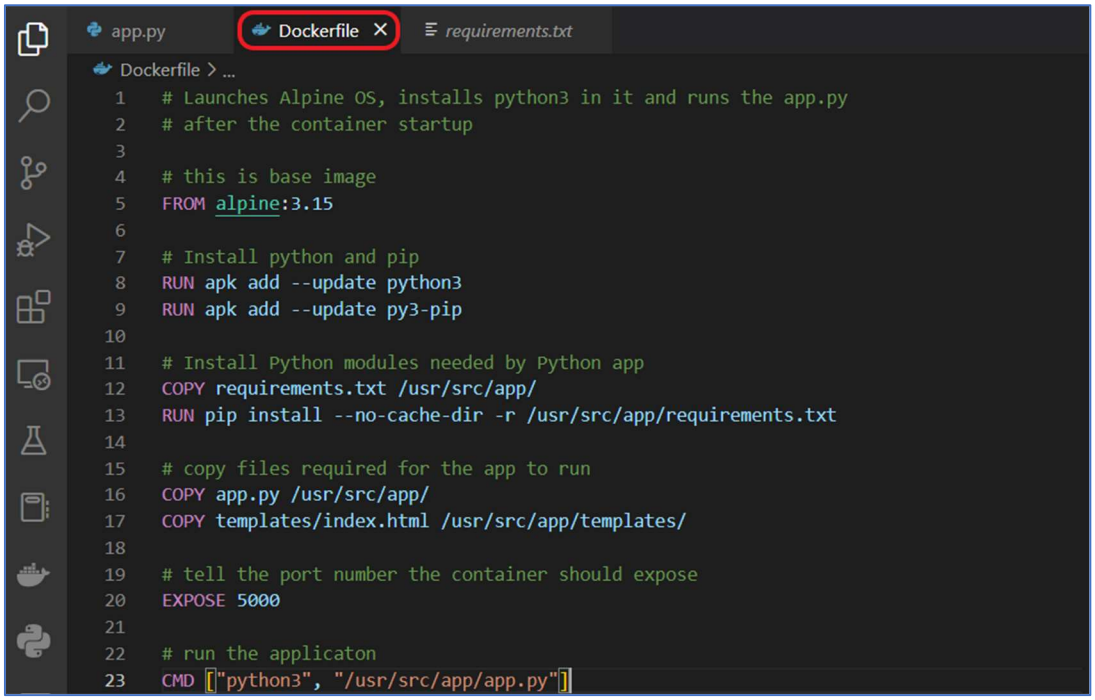
- Content of **app.py**:



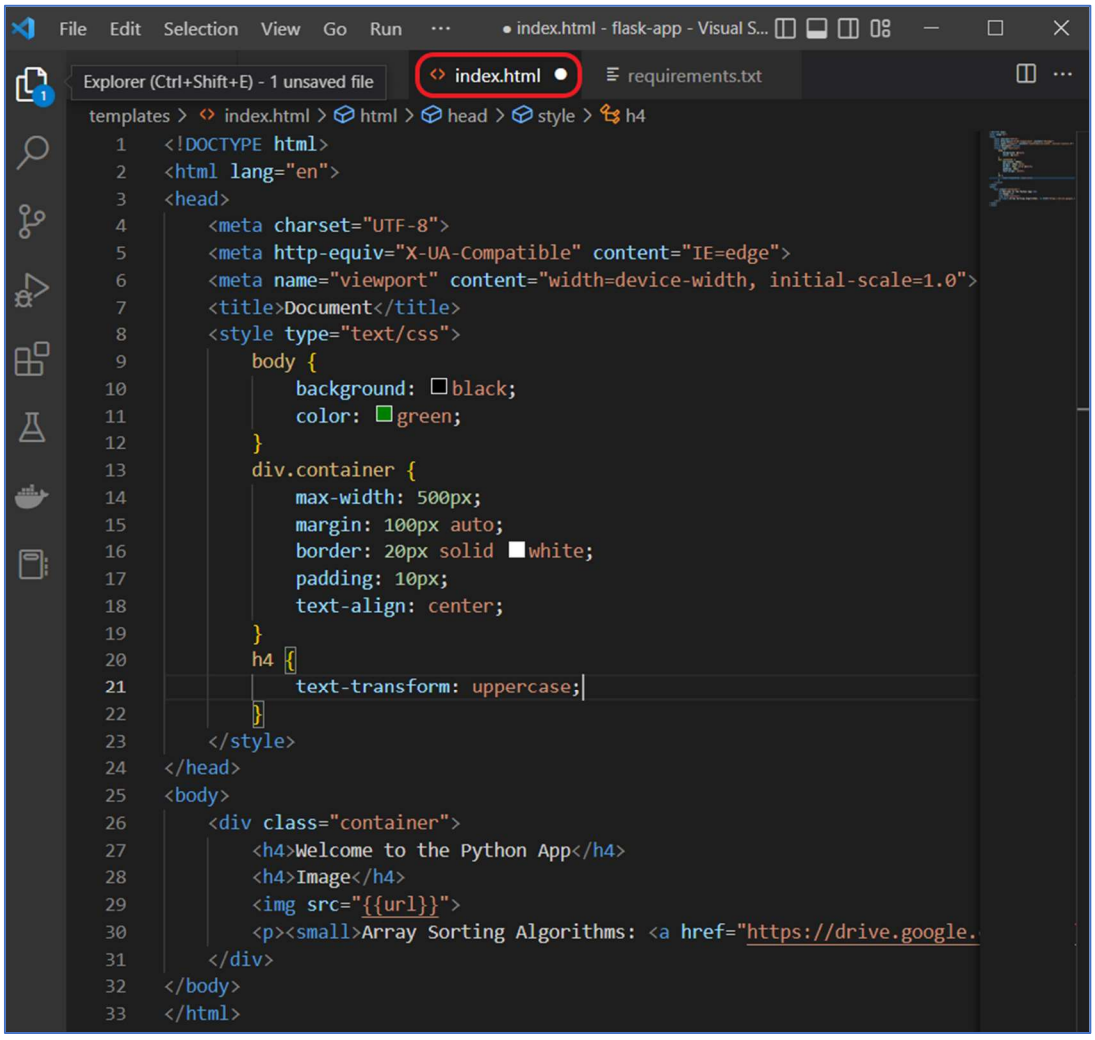
- Content of **requirements.txt**;



- Content of **Dockerfile**:



- Content of **index.html**:



- Docker image building:

```
(venv) vagrant@vm1:~/programming/docker/flask-app$ docker image build -t vagrant/myapp .
Sending build context to Docker daemon  11.16MB
Step 1/8 : FROM alpine:3.15
3.15: Pulling from library/alpine
9621f1afde84: Pull complete
Digest: sha256:cf34c62ee8eb3fe8aa24c1fab45d7e9d12768d945c3f5a6fd6a63d901e898479
Status: Downloaded newer image for alpine:3.15
 ---> c4fc93816858
Step 2/8 : RUN apk add --update py3-pip
 ---> Running in 039e4c85067d
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/community/x86_64/APKINDEX.tar.gz
(1/39) Installing libbz2 (1.0.8-r1)
```

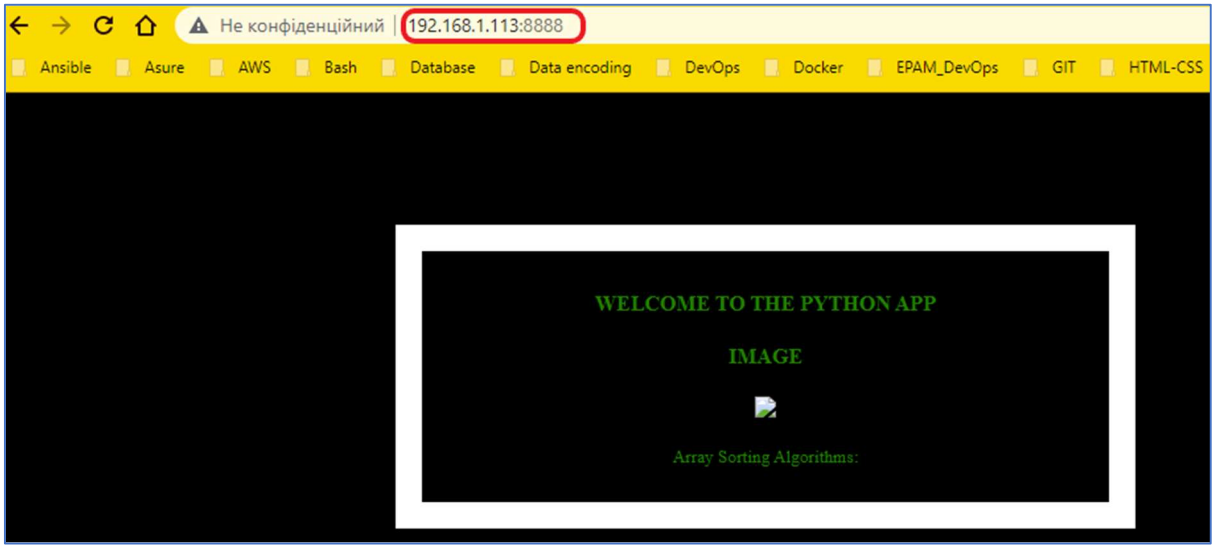
```
Step 3/8 : COPY requirements.txt /usr/src/app/
 ---> 3f6787ab6b65
Step 4/8 : RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
 ---> Running in 4184fb71225c
Collecting Flask==2.0.2
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
Collecting click>=7.1.2
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.0
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting Werkzeug>=2.0
  Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.1.tar.gz (18 kB)
Using legacy 'setup.py install' for MarkupSafe, since package 'wheel' is not installed.
Installing collected packages: MarkupSafe, Werkzeug, Jinja2, itsdangerous, click, Flask
    Running setup.py install for MarkupSafe: started
    Running setup.py install for MarkupSafe: finished with status 'done'
Successfully installed Flask-2.0.2 Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2 click-8.1.3 itsd
angerous-2.1.2
Removing intermediate container 4184fb71225c
 ---> e06232f87c63
Step 5/8 : COPY app.py /usr/src/app/
 ---> 8ff30aae61ed
Step 6/8 : COPY templates/index.html /usr/src/app/templates/
 ---> 3424fd993a54
Step 7/8 : EXPOSE 5000
 ---> Running in 78a01041bcec
Removing intermediate container 78a01041bcec
 ---> 9f2d016cfe51
Step 8/8 : CMD ["python", "/usr/src/app/app.py"]
 ---> Running in e15f7d683ca4
Removing intermediate container e15f7d683ca4
 ---> fef2eb9234c4
Successfully built fef2eb9234c4
Successfully tagged vagrant/myapp:latest
```

```
vagrant@vm1:~$ docker image ls
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
vagrant/myapp   latest    fef2eb9234c4   5 hours ago      74.2MB
vagrant         v1.0      55f6da8b8329   3 days ago       226MB
ubuntu          20.04     d5447fc01ae6   2 weeks ago      72.8MB
alpine          3.15      c4fc93816858   4 months ago     5.59MB
hello-world     latest    feb5d9fea6a5   15 months ago    13.3kB
vagrant@vm1:~$
```

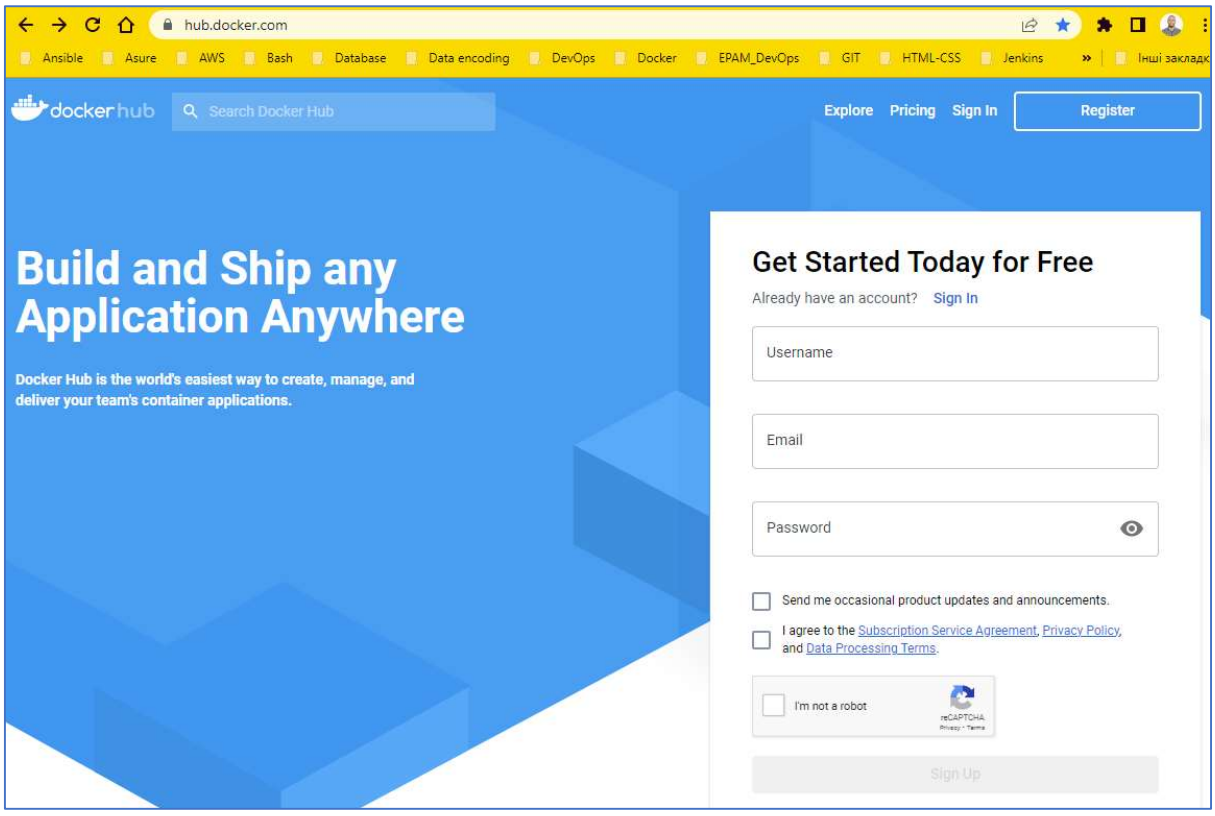
- Running container with our web application:

```
vagrant@vm1:~/programming/docker/flask-app$ docker container run -p 8888:5000 --name myapp vagrant/myapp
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI s
erver instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

```
vagrant@vm1:~$ hostname -I
10.0.2.15 192.168.1.113 172.17.0.1
vagrant@vm1:~$
```



- Creating an account in DockerHub :

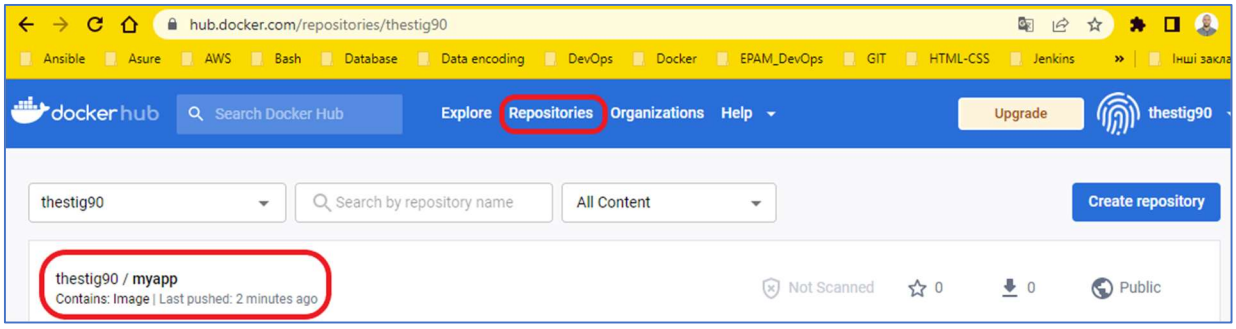


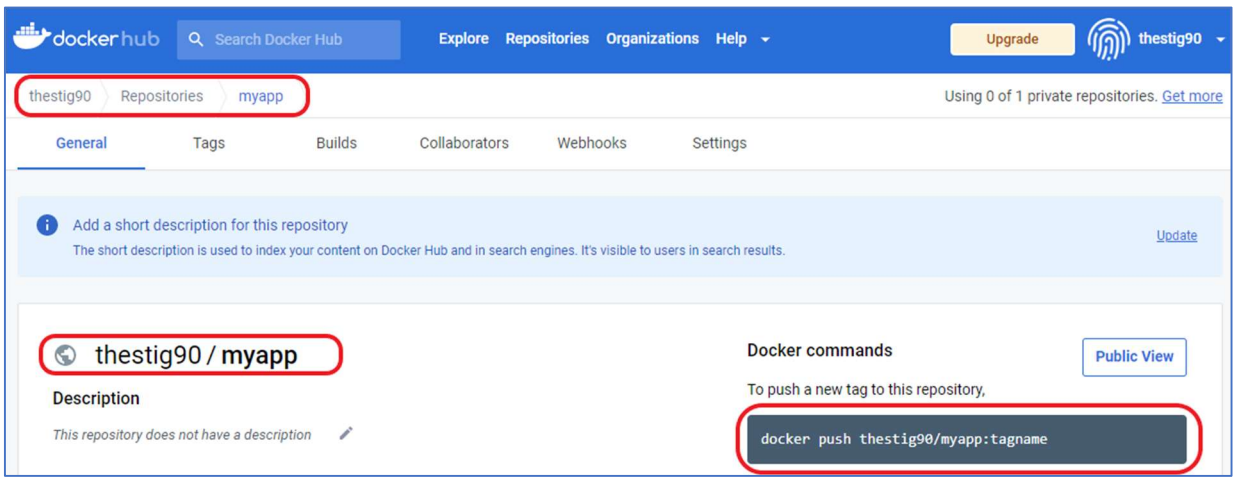
- Login to Docker registry:

```
vagrant@vm1:~/programming/docker/flask-app$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker
ver to https://hub.docker.com to create one.
Username: thestig90
Password:
WARNING! Your password will be stored unencrypted in /home/vagrant/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- Pushing to Docker registry:

```
vagrant@vm1:~/programming/docker/flask-app$ docker tag vagrant/myapp thestig90/myapp
vagrant@vm1:~/programming/docker/flask-app$
vagrant@vm1:~/programming/docker/flask-app$ docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
thestig90/myapp     latest    ec1c0de1a355   41 minutes ago   74.4MB
vagrant/myapp       latest    ec1c0de1a355   41 minutes ago   74.4MB
vagrant             v1.0      55f6da8b8329   3 days ago       226MB
ubuntu              20.04     d5447fc01ae6   2 weeks ago      72.8MB
alpine              3.15      c4fc93816858   4 months ago     5.59MB
hello-world         latest    feb5d9fea6a5   15 months ago    13.3kB
vagrant@vm1:~/programming/docker/flask-app$
vagrant@vm1:~/programming/docker/flask-app$ docker push thestig90/myapp
Using default tag: latest
The push refers to repository [docker.io/thestig90/myapp]
a6c636627706: Pushed
657ed308b191: Pushed
74efca48eb3a: Pushed
04643e6be1ad: Pushed
b78e760dfafd: Pushed
b5085eebe67d: Pushed
34d5ebaa5410: Mounted from library/alpine
latest: digest: sha256:166bcdb29dbfe2d33d2f13cb917533ace619ed713a62c5a9ec903d4b2da5f05f
```





6. Frequently used Docker commands.

```
$ docker stop $(docker ps -a -q)
```
#stop all containers [you need stop before delete]

```
$ docker rmi $(docker images-a -q)
```
remove all images

```
vagrant@vm1:~/programming/docker/flask-app$ docker
```

```
Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.9.1-docker)
  compose*    Docker Compose (Docker Inc., v2.14.1)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  scan*       Docker Scan (Docker Inc., v0.23.0)
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
```

7. Docker Compose.

- Install **Docker Compose**

You can run Compose on macOS, Windows, and 64-bit Linux.

- Prerequisites

Docker Compose relies on **Docker Engine** for any meaningful work, so make sure you have **Docker Engine** installed either locally or remote, depending on your setup.

On desktop systems like **Docker Desktop** for Mac and Windows, **Docker Compose** is included as part of those desktop installs.

On Linux systems, first install the **Docker Engine** for your OS as described earlier.

To run Compose as a non-root user:

```
$ sudo usermod -aG docker “user_name”
```

- Installation

In the latest version **Docker Compose** has been included into a Docker Engine, so you don’t need to install them separately.

To download and install Compose standalone, run:

```
$ curl -SL
https://github.com/docker/compose/releases/download/v2.14.2/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose
```

Apply executable permissions to the standalone binary in the target path for the installation:

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Test and execute compose commands using:

```
$ docker compose
```

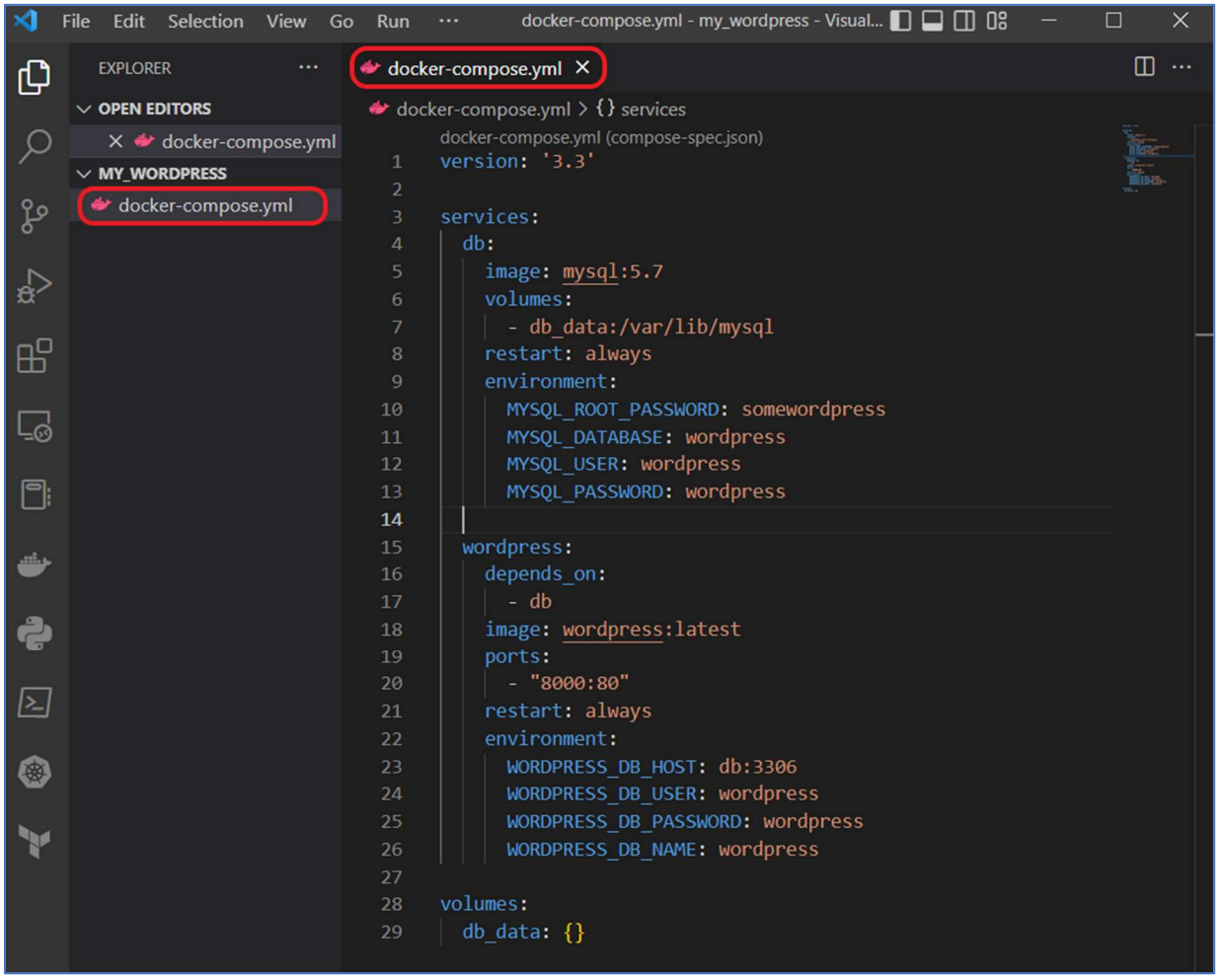
8. Docker Compose project examle.

- Create an empty project directory with **docker-compose.yml** and change into.

```
vagrant@vm1:~/programming/docker$
vagrant@vm1:~/programming/docker$ mkdir my_wordpress
vagrant@vm1:~/programming/docker$
vagrant@vm1:~/programming/docker$ cd my_wordpress/
vagrant@vm1:~/programming/docker/my_wordpress$
vagrant@vm1:~/programming/docker/my_wordpress$ touch docker-compose.yml
vagrant@vm1:~/programming/docker/my_wordpress$
vagrant@vm1:~/programming/docker/my_wordpress$ ll
total 8
drwxrwxr-x 2 vagrant vagrant 4096 Dec 25 19:17 ./
drwxrwxr-x 4 vagrant vagrant 4096 Dec 25 19:16 ../
-rw-rw-r-- 1 vagrant vagrant    0 Dec 25 19:17 docker-compose.yml
vagrant@vm1:~/programming/docker/my_wordpress$
```

- Create **docker-compose.yml** file.

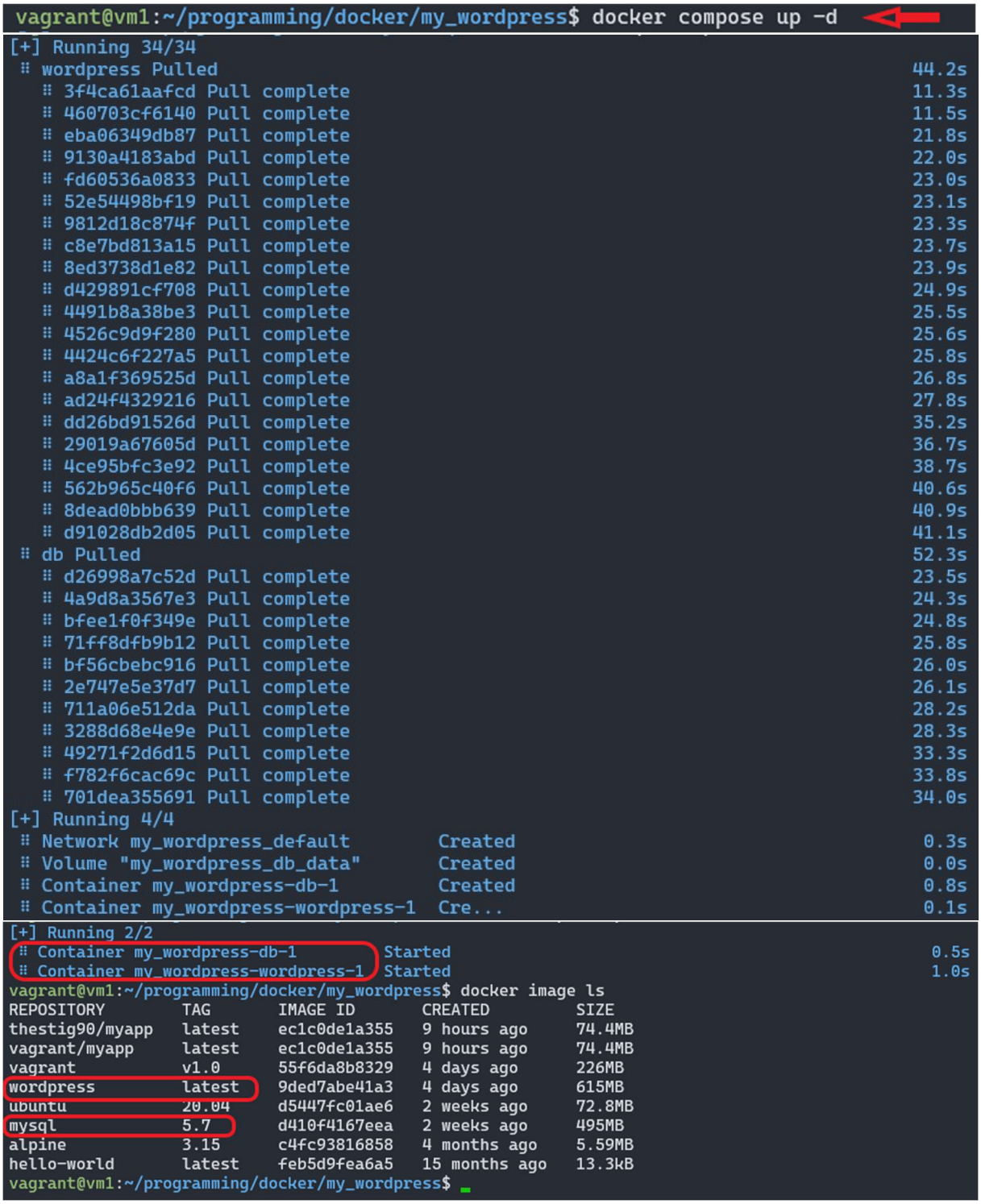
This **docker-compose.ym** file starts your WordPress blog and a separate MySQL instance with a volume mount for data persistence:



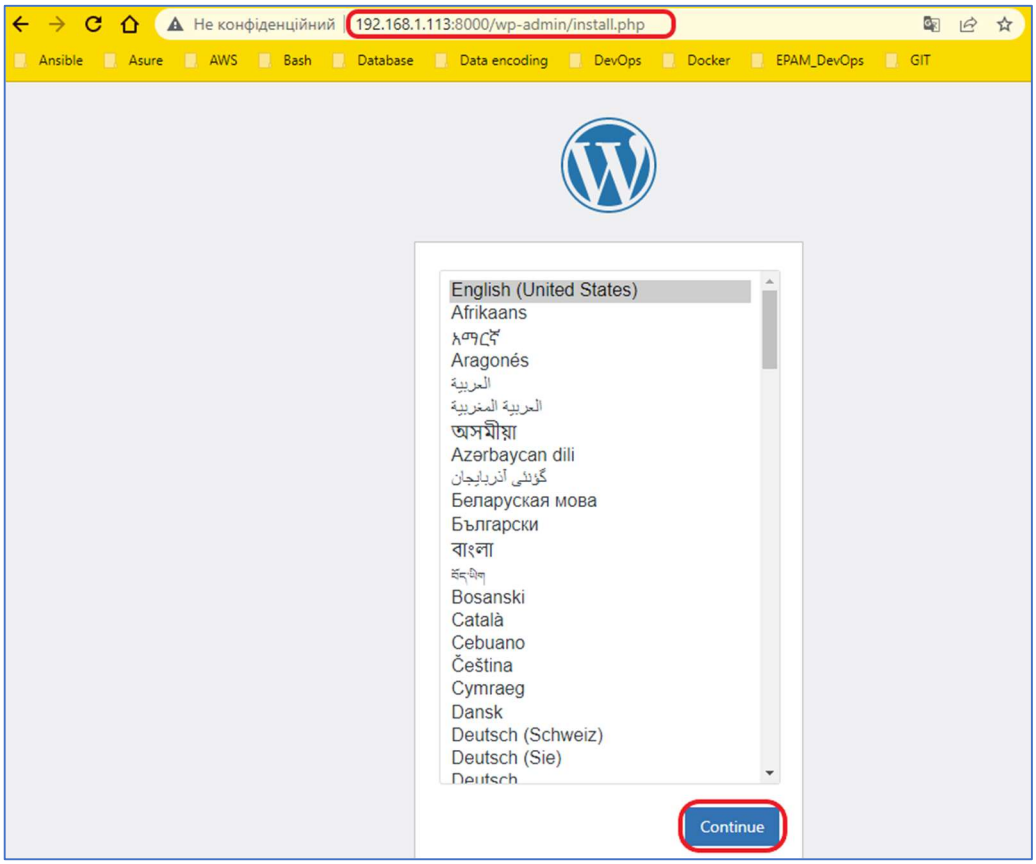
- Build the project.

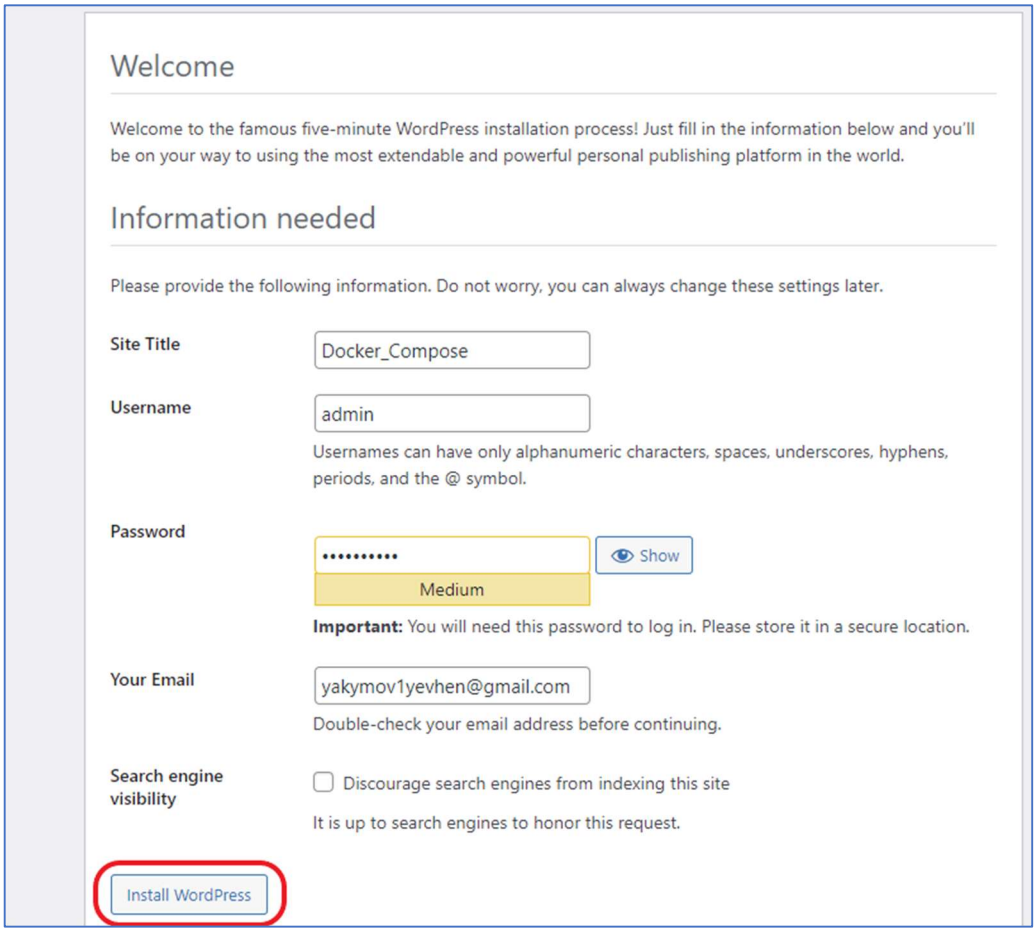
```
$ docker compose up -d
```

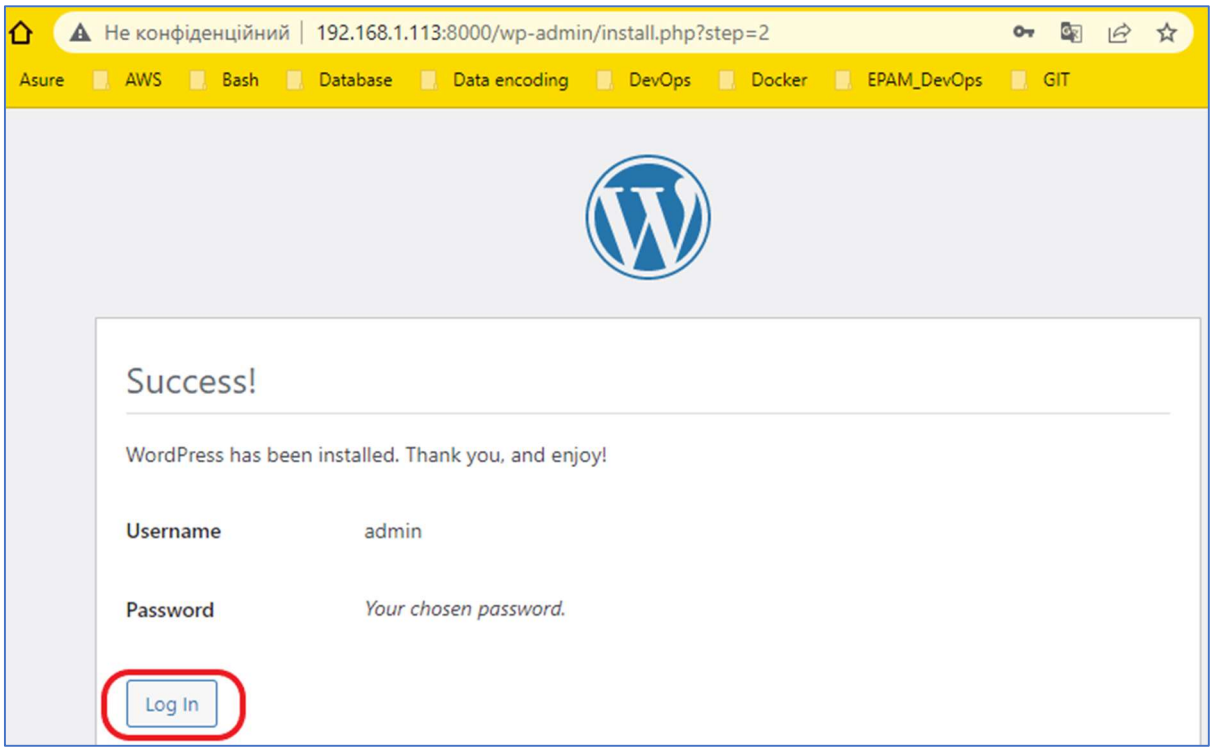
This runs **docker-compose up** in detached mode, pulls the needed **Docker images**, and starts the **wordpress** and **database containers**, as shown in the example below:



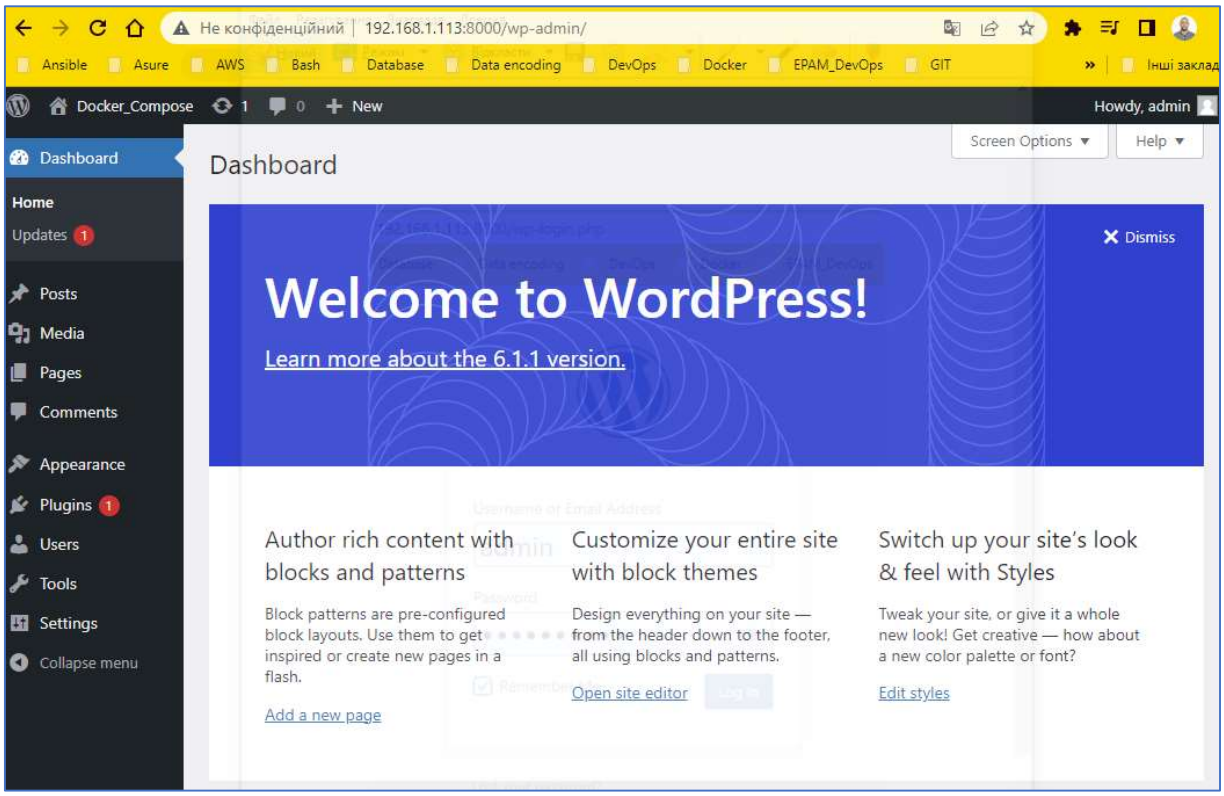
- Wordpress realization.

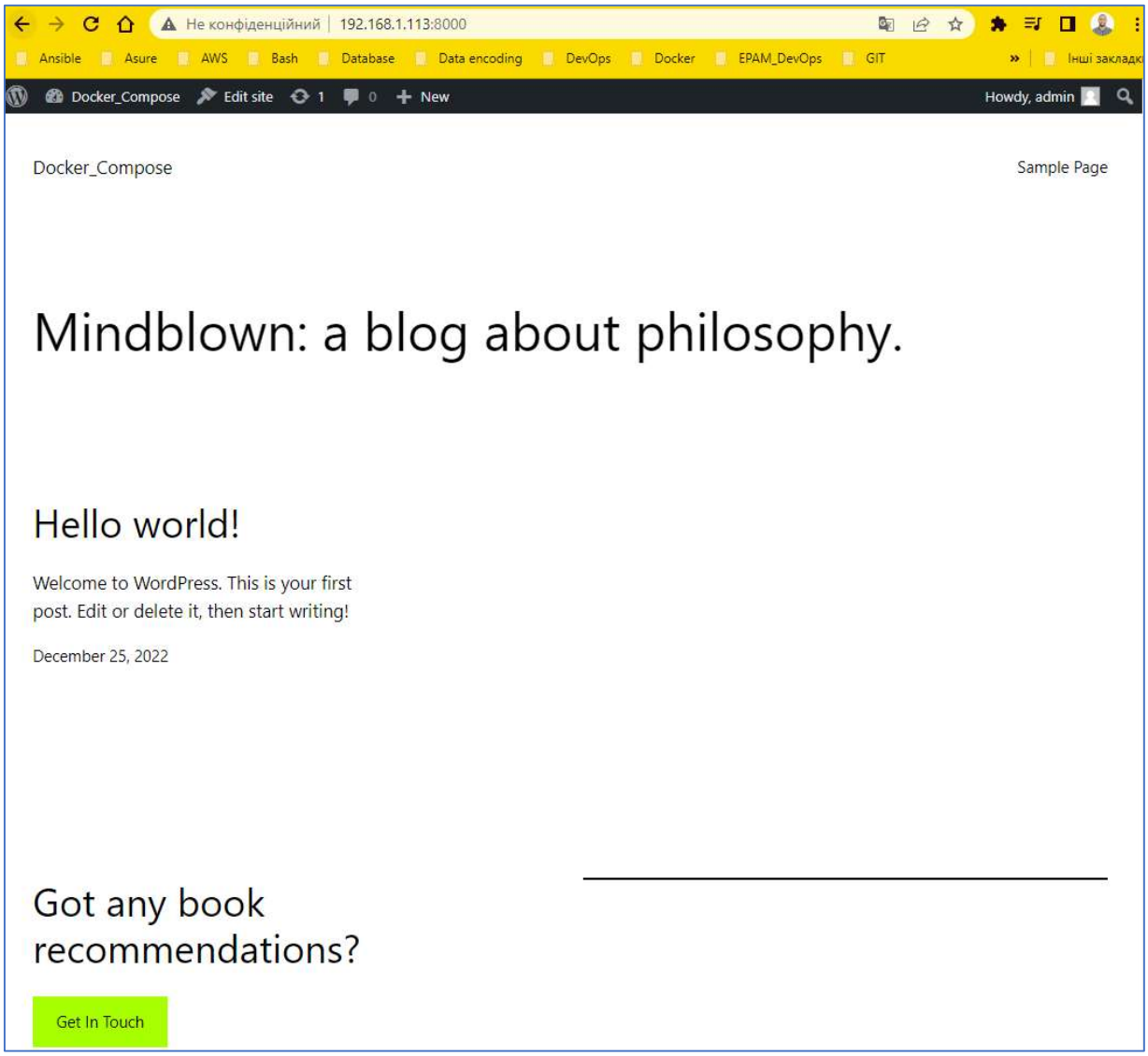





Не конфіденційний | 192.168.1.113:8000/wp-admin/install.php?step=2
Asure
AWS
Bash
Database
Data encoding
DevOps
Docker
EPAM_DevOps
GIT
Success!
WordPress has been installed. Thank you, and enjoy!
Username
admin
Password
Your chosen password.
Log In


192.168.1.113:8000/wp-login.php
Database
Data encoding
DevOps
Docker
EPAM_DevOps
Username or Email Address
admin
Password
Remember Me
Log In
Lost your password?
← Go to Docker_Compose


Не конфіденційний | 192.168.1.113:8000/wp-admin/
Ansible
Asure
AWS
Bash
Database
Data encoding
DevOps
Docker
EPAM_DevOps
GIT
Docker_Compose
1
0
New
Howdy, admin
Dashboard
Home
Updates 1
Posts
Media
Pages
Comments
Appearance
Plugins 1
Users
Tools
Settings
Collapse menu
Screen Options
Help
Dismiss
Welcome to WordPress!
Learn more about the 6.1.1 version.
Author rich content with blocks and patterns
Block patterns are pre-configured block layouts. Use them to get inspired or create new pages in a flash.
Add a new page
Customize your entire site with block themes
Design everything on your site — from the header down to the footer, all using blocks and patterns.
Open site editor
Switch up your site's look & feel with Styles
Tweak your site, or give it a whole new look! Get creative — how about a new color palette or font?
Edit styles


Не конфіденційний | 192.168.1.113:8000
Ansible
Asure
AWS
Bash
Database
Data encoding
DevOps
Docker
EPAM_DevOps
GIT
Docker_Compose
Edit site
1
0
New
Howdy, admin
Docker_Compose
Sample Page
Mindblown: a blog about philosophy.
Hello world!
Welcome to WordPress. This is your first post. Edit or delete it, then start writing!
December 25, 2022
Got any book recommendations?
Get In Touch

9. Docker Compose. Use volumes.

- Use volumes.

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

-Volumes are easier to back up or migrate than bind mounts.

- You can manage volumes using Docker CLI commands or the Docker API.

- Volumes work on both Linux and Windows containers.

- Volumes can be more safely shared among multiple containers.

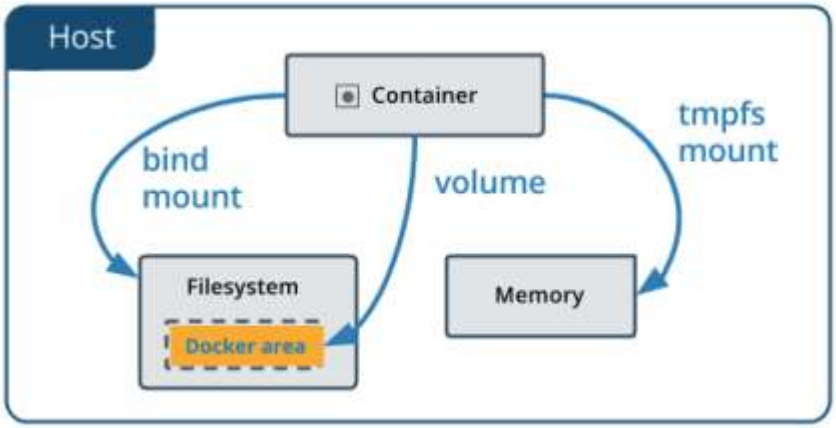
-Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.

- New volumes can have their content pre-populated by a container.

- Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

-Volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.

If your container generates non-persistent state data, consider using a **tmpfs mount** to avoid storing the data anywhere permanently, and to increase the container's performance by avoiding writing into the container's writable layer.



- Create and manage volumes.

You can create and manage volumes outside the scope of any container.

Create a volume:

```
$ docker volume create my_vol
```

List volumes:

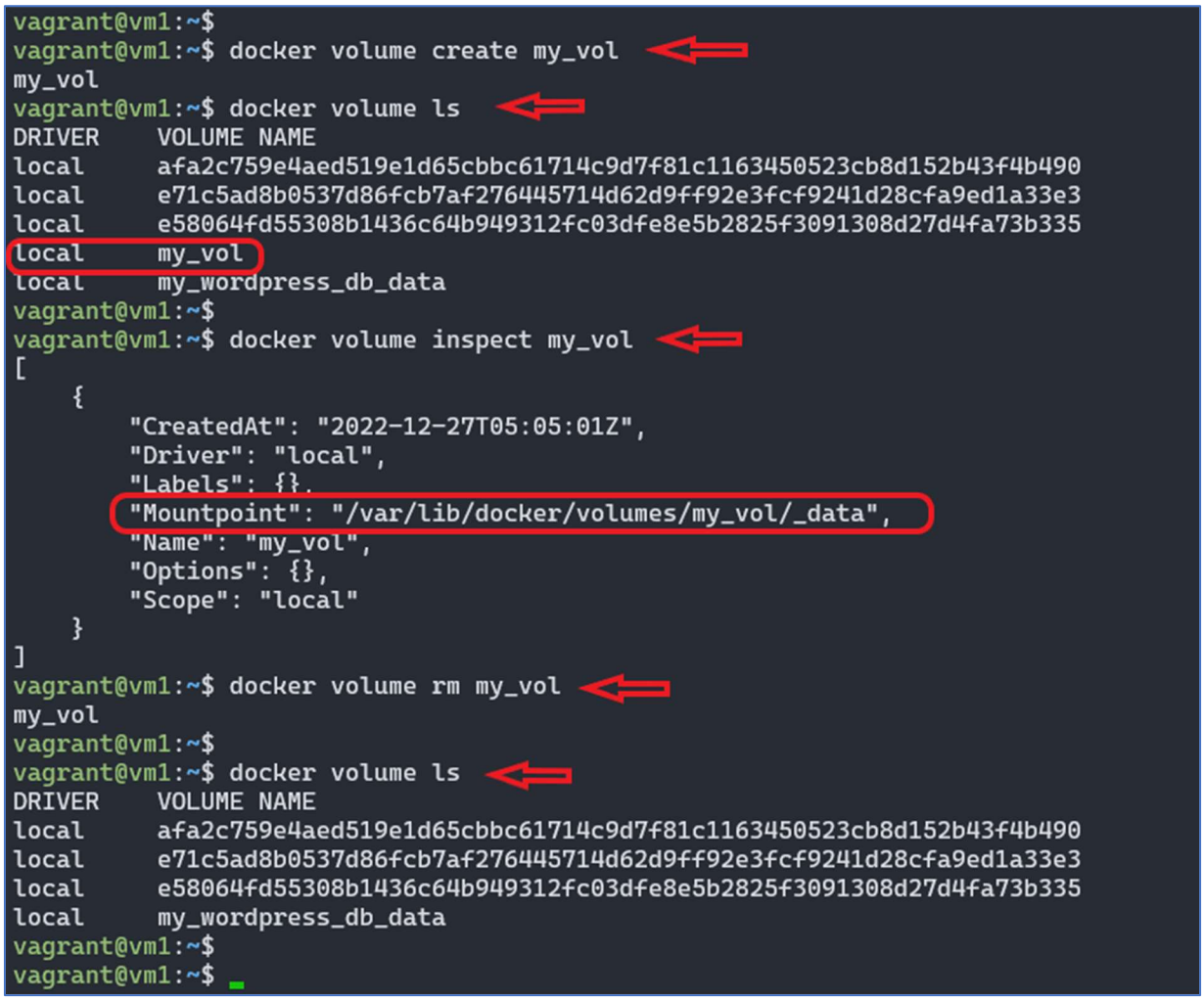
```
$ docker volume ls
```

Inspect a volume:

```
$ docker volume inspect my_vol
```

Remove a volume:

```
$ docker volume rm my_vol
```



10. Docker Compose. Use networks.

One of the reasons **Docker containers** and services are so powerful is that you can connect them together, or connect them to non-Docker workloads.

- Network drivers.

bridge: The default network driver. If you don't specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate. See bridge networks.

host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. See use the host network.

overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See overlay networks.

ipvlan: IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration. See IPvlan networks.

macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack. See Macvlan networks.

none: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services. See disable container networking.

Network plugins: You can install and use third-party network plugins with Docker. These plugins are available from Docker Hub or from third-party vendors. See the vendor's documentation for installing and using a given network plugin.

- Create and manage bridge networks.

In terms of networking, a **bridge network** is a Link Layer device which forwards traffic between network segments. A bridge can be a hardware device or a software device running within a host machine's kernel.

Bridge networks apply to containers running on the same Docker daemon host. For communication among containers running on different Docker daemon hosts, you can either manage routing at the OS level, or you can use an **overlay network**.

User-defined bridges provide automatic **DNS resolution** between containers:

Containers on the default **bridge network** can only access each other by **IP addresses**, unless you use the --link option, which is considered legacy. On a **user-defined bridge** network, containers can **resolve each other by name or alias**. Imagine an application with a web front-end and a database back-end. If you call your containers web and db, the web container can connect to the db container at db, no matter which Docker host the application stack is running on.

Use the **docker network create** command to create a user-defined bridge network.:

```
$ docker network create my_net
```

List networks:

```
$ docker network ls
```

Inspect a network:

```
$ docker network inspect my_net
```

Remove a network:

```
$ docker network rm my_net
```

```
vagrant@vm1:~$ docker network create my_net
28146931dc8344d6a9f4a3c9ad136c47dd7fb2262d37b25b9e0355ce6ac63e61
vagrant@vm1:~$
vagrant@vm1:~$ docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
b422ebe974a1   bridge                 bridge    local
17580c3eabbb   host                   host      local
28146931dc83   my_net                 bridge    local
21c069d61709   my_wordpress_default   bridge    local
cd01fc8417ac   none                   null      local
vagrant@vm1:~$
vagrant@vm1:~$ docker network inspect my_net
[
    {
        "Name": "my_net",
        "Id": "28146931dc8344d6a9f4a3c9ad136c47dd7fb2262d37b25b9e0355ce6ac63e61",
        "Created": "2022-12-27T05:37:06.3708228682",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
vagrant@vm1:~$ docker network rm my_net
my_net
vagrant@vm1:~$ docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
b422ebe974a1   bridge                 bridge    local
17580c3eabbb   host                   host      local
21c069d61709   my_wordpress_default   bridge    local
cd01fc8417ac   none                   null      local
vagrant@vm1:~$
```