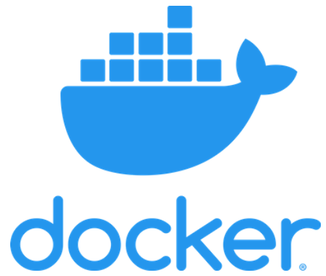
Docker Deep Dive

Chapter #7



Docker Compose

Docker

COMPOSE

www.networknuts.net

DOCKER COMPOSE

Introduction to Docker Compose

MULTI CONTAINER APP

Running a multi container application

DOCKER COMPOSE

Deploy & manage multi container applications.

MODERN APPLICATIONS

Modern apps are made of small services that interact to form a useful application - microservices

DOCKER COMPOSE

Compose let us describe a complete application with single file.

Docker Compose

Docker compose: deep dive

Most modern apps are made of multiple smaller services that interact to form a useful app. We call this microservices. A simple example might be an app with the following four services:

- web front-end
- ordering
- catalog
- back-end database

Instead of gluing everything together with scripts and long docker commands, Docker Compose lets you describe an entire app in a single declarative configuration file. You then deploy it with a single command.

Once the app is deployed, you can manage its entire lifecycle with a simple set of commands. You can even store and manage the configuration file in a version control system!

Compose background

Initially it was called Fig. Fig was a powerful tool, created by a company called Orchard, and it was the best way to manage multi-container Docker apps. It was a Python tool that sat on top of Docker, and allowed you to define entire multi-container apps in a single YAML file. You could then deploy the app with the fig command-line tool. Fig could even manage the entire life-cycle of the app.

In 2014, Docker, Inc. acquired Orchard and re-branded Fig as Docker Compose. The command-line tool was renamed from fig to docker-compose , and ever since the acquisition, it’s been an external tool that gets bolted on top of the Docker Engine. Even though it’s never been fully integrated into the Docker Engine, it’s always been immensely popular and very widely used.

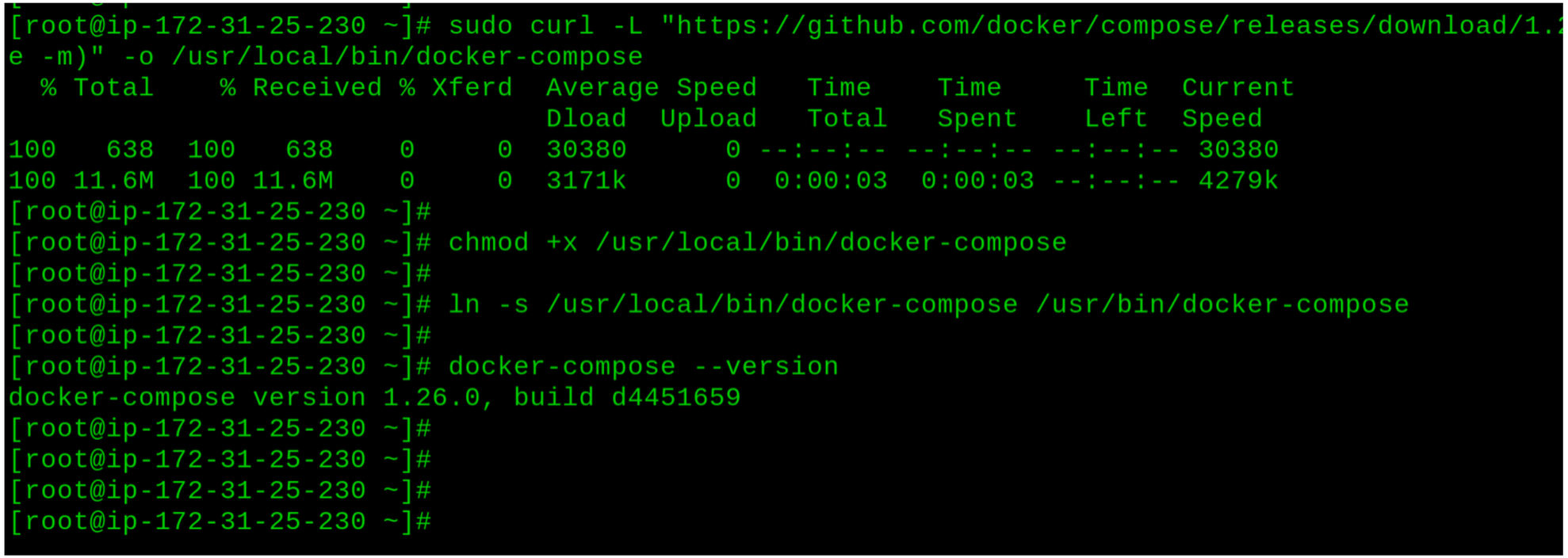
Compose installation

#1 - First download the current stable version of docker-compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

#2 - Make the binary executable
sudo chmod +x /usr/local/bin/docker-compose

#3 - Also create a symbolic link to /usr/bin or any other directory in your path
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

#4 - Check the version of docker-compose
docker-compose --version


[root@ip-172-31-25-230 ~]# sudo curl -L "https://github.com/docker/compose/releases/download/1.2
e -m)" -o /usr/local/bin/docker-compose
  % Total   % Received % Xferd Average Speed   Time   Time   Time Current
                                 Dload Upload   Total   Spent   Left Speed
100   638 100   638   0   0 30380   0 --:--:-- --:--:-- --:--:-- 30380
100 11.6M 100 11.6M   0   0 3171k   0 0:00:03 0:00:03 --:--:-- 4279k
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# chmod +x /usr/local/bin/docker-compose
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker-compose --version
docker-compose version 1.26.0, build d4451659
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#

Deploying app with compose

We will deploy a very simple app using docker-compose. You'll need the following 4 files from https://github.com/networknuts/counter-app.git:

Dockerfile
app.py
requirements.txt
docker-compose.yml

Clone the Git repo locally.

```
[root@ip-172-31-25-230 docker-compose]#
[root@ip-172-31-25-230 docker-compose]# git clone https://github.com/networknuts/counter-app.git
Cloning into 'counter-app'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 15 (delta 3), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (15/15), done.
[root@ip-172-31-25-230 docker-compose]#
[root@ip-172-31-25-230 docker-compose]# ls
counter-app  docker-deep-dive-counter
[root@ip-172-31-25-230 docker-compose]#
[root@ip-172-31-25-230 docker-compose]# cd counter-app/
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# ls
app.py  docker-compose.yml  Dockerfile  README.md  requirements.txt
[root@ip-172-31-25-230 counter-app]#
```

You will find these files:

- app.py is the application code (a Python Flask app)
- docker-compose.yml is the Docker Compose file that describes how Docker should deploy the app
- Dockerfile describes how to build the image for the web-fe service
- requirements.txt lists the Python packages required for the app

The app.py file is obviously the core of the application. But docker-compose.yml is the glue that sticks all the app components together. Let’s use Compose to bring the app up, using

docker-compose up -d OR docker-compose up &

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker-compose up &
[1] 3291
[root@ip-172-31-25-230 counter-app]# Creating network "counter-app_counter-net" with the default driver
Creating volume "counter-app_counter-vol" with default driver
Building web-fe
Step 1/5 : FROM python:3.4-alpine
3.4-alpine: Pulling from library/python
8e402f1a9c57: Pull complete
cda9ba2397ef: Pull complete
aafecf9bbbfd: Pull complete
bc2e7e266629: Pull complete
e1977129b756: Pull complete
Digest: sha256:c210b660e2ea553a7afa23b41a6ed112f85dbce25cbcb567c75dfe05342a4c4b
Status: Downloaded newer image for python:3.4-alpine
 ---> c06adcf62f6e
Step 2/5 : ADD . /code
 ---> 2abb8defce43
Step 3/5 : WORKDIR /code
 ---> Running in 93009d689b8b
Removing intermediate container 93009d689b8b
 ---> 9e8387895155
```

docker-compose up is the most common way to bring up a Compose app (we’re calling a multi-container app defined in a Compose file a Compose app). It builds all required images, creates all required networks and volumes, and starts all required containers.

By default, docker-compose up expects name of the Compose file to docker-compose.yml or docker-compose.yaml .

If your Compose file has a different name, you need to specify it with the -f flag. The following example will deploy an application from a compose file called prod-my-app.yml

 # docker-compose -f prod-my-app.yml up

Now that the app is built and running, we can use normal docker commands to view the images, containers, networks, and volumes that Compose created.

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
docker-deep-dive-counter_web-fe     latest              94022cacbb0a        52 minutes ago      84MB
counter-app_web-fe                  latest              a1748a1199b3        About an hour ago   84.5MB
redis                               alpine              e008f2ff99d0        4 days ago          31.5MB
centos                              latest              470671670cac        4 months ago        237MB
python                              3.4-alpine          c06adcf62f6e        14 months ago       72.9MB
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
```

The counterapp_web-fe:latest image was created by the build: . instruction in the docker-compose.yml file. This instruction caused Docker to build a new image using the Dockerfile in the same directory. It contains the application code for the Python Flask web app, and was built from the python:3.4-alpine image. Check contents of the Dockerfile.

```
FROM python:3.4-alpine            << Base image
ADD . /code                       << Copy app into image
WORKDIR /code                     << Set working directory
RUN pip install -r requirements.txt  << install requirements
CMD ["python", "app.py"]          << Set the default app
```

Compose has named the newly built image as a combination of the project name (counter-app), and the resource name as specified in the Compose file (web-fe). Compose has removed the dash (-) from the project name. All resources deployed by Compose will follow this naming convention. The redis:alpine image was pulled from Docker Hub by the image: "redis:alpine" instruction in the .Services.redis section of the Compose file.

The counterapp_web-fe container is running the application’s web front end. This is running the app.py code and is mapped to port 5000 on all interfaces on the Docker host.

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker container ls
CONTAINER ID        IMAGE                COMMAND                  CREATED             STATUS              PORTS
MES
289fa8330070        redis:alpine         "docker-entrypoint.s…"   13 minutes ago      Up 13 minutes       6379/tcp
unter-app_redis_1
932bd5803a3f        counter-app_web-fe   "python app.py"          13 minutes ago      Up 13 minutes       0.0.0.0:5000->5000/tcp
unter-app_web-fe_1
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
```

The following network and volume listings show the counterapp_counter-net and counterapp_counter-vol networks and volumes.

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker network ls
NETWORK ID          NAME                      DRIVER              SCOPE
89092db16721        bridge                    bridge              local
a1e616a2c6bc        counter-app_counter-net   bridge              local
fcffef57df95        host                      host                local
6daae933de21        none                      null                local
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker volume ls
DRIVER              VOLUME NAME
local               2c19793a7a6fb440eb57d9fffe299309226d63de34cea15b8ea0ed7cf9ab5feb
local               13c57e96de8e495e98d8bd325a5bfac45603e56d6addb497f81aa5e809c7161d
local               963e98c038e1e1eff76db66eb88b2a76f56e68913d4dce0ae903fc98df6dba97
local               26939b904d5ae609a0198bdf6be59402d766d7365b9dcbc66e0d7c5f676c6cbb
local               counter-app_counter-vol
local               docker-deep-dive-counter_counter-vol
```

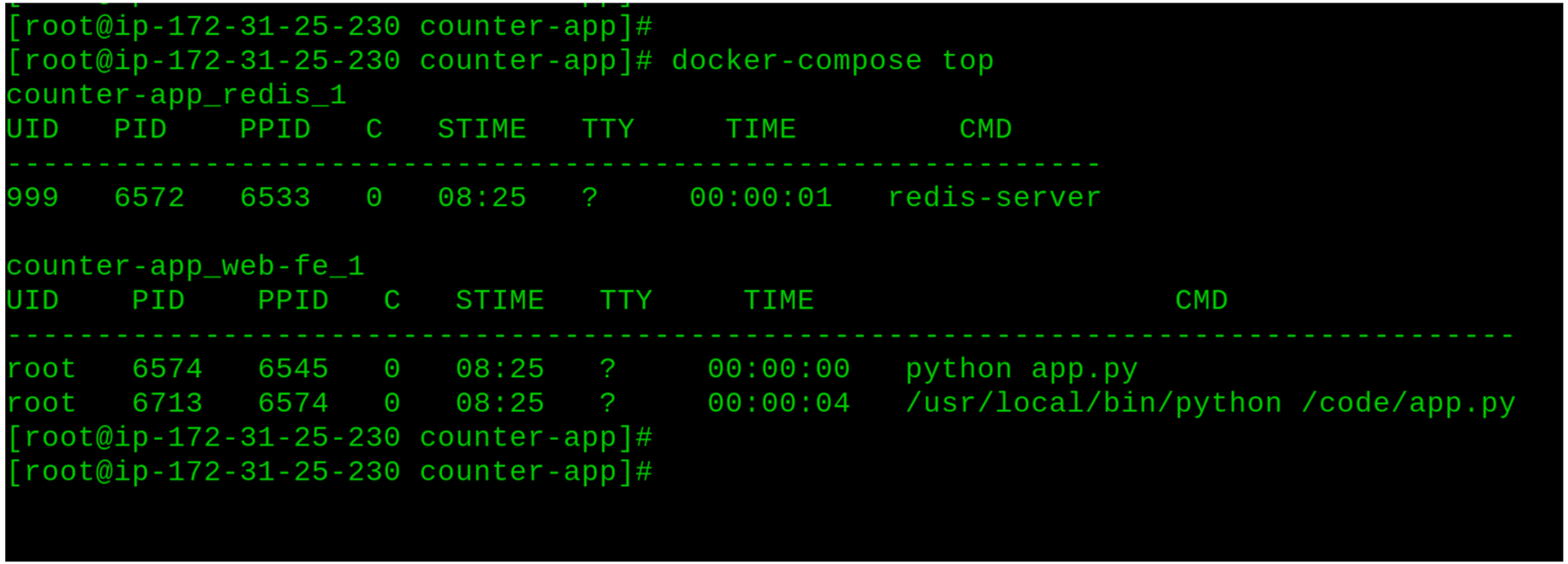
With the application successfully deployed, you can point a web browser at your Docker host on port 5000

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# curl http://0.0.0.0:5000
Network Nuts welcome Docker Deep Divers! You've visited me 1 times.
[root@ip-172-31-25-230 counter-app]# curl http://0.0.0.0:5000
Network Nuts welcome Docker Deep Divers! You've visited me 2 times.
[root@ip-172-31-25-230 counter-app]# curl http://0.0.0.0:5000
Network Nuts welcome Docker Deep Divers! You've visited me 3 times.
[root@ip-172-31-25-230 counter-app]#
```

You can check the current state of the application using: **docker-compose ps**

```
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker-compose ps
        Name                      Command               State           Ports
-------------------------------------------------------------------------------------
counter-app_redis_1    docker-entrypoint.sh redis ...   Up      6379/tcp
counter-app_web-fe_1   python app.py                    Up      0.0.0.0:5000->5000/tcp
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
```

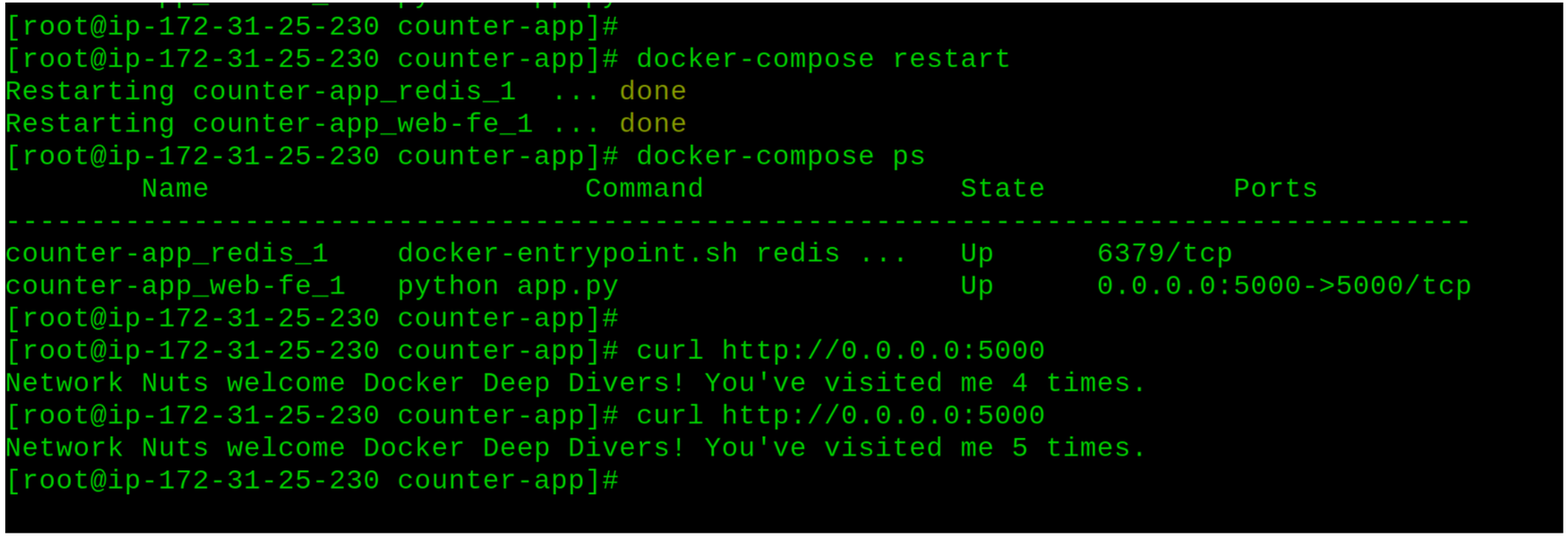
Use the **docker-compose top** command to list the processes running inside of each service (container).



Use the docker-compose stop command to stop the app without deleting its resources. Then show the status of the app with docker-compose ps .

```
[root@ip-172-31-25-230 counter-app]# docker-compose ps
        Name                      Command               State           Ports
-------------------------------------------------------------------------------------
counter-app_redis_1    docker-entrypoint.sh redis ...   Up      6379/tcp
counter-app_web-fe_1   python app.py                    Up      0.0.0.0:5000->5000/tcp
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker-compose stop
Stopping counter-app_redis_1  ... done
Stopping counter-app_web-fe_1 ... done
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]# docker-compose ps
        Name                      Command               State    Ports
-------------------------------------------------------------------------
counter-app_redis_1    docker-entrypoint.sh redis ...   Exit 0
counter-app_web-fe_1   python app.py                    Exit 0
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
```

You can delete a stopped Compose app with the **docker-compose rm** command. This will delete the containers and networks the app is using, but it will not delete volumes or images. Nor will it delete the application source code (app.py , Dockerfile , requirements.txt , and docker-compose.yml) in your project directory. Restart app with the **docker-compose restart** command.



Use the **docker-compose down** command to stop & delete the app with a single command.

```
[root@ip-172-31-25-230 counter-app]# docker-compose down
Stopping counter-app_redis_1  ... done
Stopping counter-app_web-fe_1 ... done
Removing counter-app_redis_1  ... done
Removing counter-app_web-fe_1 ... done
Removing network counter-app_counter-net
[root@ip-172-31-25-230 counter-app]#
[root@ip-172-31-25-230 counter-app]#
```

The app is now deleted. Only its images, volumes and source code remain.

WHAT WE LEARNED

- Docker Compose is a Python application that we install on top of the Docker Engine.
- It lets us define multi-container apps in a single declarative configuration file & deploy it with a single command.
- Compose files can be YAML or JSON, and they define all of the containers, networks, volumes, and secrets that an application requires.
- We then feed the file to the docker-compose command line tool, and Compose instructs Docker to deploy it.
- Once the app is deployed, we can manage its entire lifecycle using the many docker-compose sub-commands.
- Compose file is an excellent source of application documentation — it defies all the services that make up the app, the images they use, ports they expose, networks and volumes they use, and much more.