Prácticas

1. Crear un contenedor

Objetivo: crear una imagen Docker que permita la ejecución del código Python, y después instanciarla. Las manipulaciones se hacen en una Ubuntu 19.10.

1. Primero instale Docker:

```
# apt-get install docker.io
```

2. Cree un directorio hello donde pondrá las diferentes fuentes:

```
# mkdir hello ; cd hello
```

3. Cree un script Python simple llamado hello.py que muestre "Hello World !", y compruebe su funcionamiento en local:

```
# cat hello.py
print("Hello World !")
# python ./hello.py
Hello World !
```

4. Visite el Docker Hub, https://hub.docker.com/, y busque "python". El primer resultado debería ser la imagen oficial. Observe su nombre "python", servirá para la instrucción FROM del archivo Dockerfile.

5. Cree un archivo Dockerfile usando la imagen encontrada, copie el archivo fuente y lance el comando Python asociado:

```
# cat Dockerfile
FROM python
COPY hello.py /src/
CMD ["python", "/src/hello.py"]
```

6. Cree una imagen y llámela python-hello. Docker descargará la imagen python desde el Hub y ejecutará las diferentes etapas del Dockerfile, para generar la imagen final:

```
# docker build -t python-hello .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM python
latest: Pulling from library/python
dc65f448a2e2: Pull complete
346ffb2b67d7: Pull complete
...
1fa0065c6287: Pull complete
Digest: sha256:2b8823de22d5434cd9b7aff963e9299ef52605402bd0d4ef23e45f55333a5d4c
Status: Downloaded newer image for python:latest
---> efdecc2e377a
Step 2/3 : COPY hello.py /src
---> ab3245d18db7
Step 3/3 : CMD ["python", "/src/hello.py"]
---> Running in 1c78455f3eef
Removing intermediate container 1c78455f3eef
---> 6c70101ea242
Successfully built 6c70101ea242
Successfully tagged python-hello:latest
```

7. El comando **docker image** debe devolver (al menos) dos imágenes: la imagen python de base, y la suya. Observe las dos imágenes, python y python-hello, y note la diferencia en los layers (capas). Podrá observar que su imagen contiene todas las capas de la imagen python, más otras dos. Estas dos capas están vinculadas a las dos instrucciones del Dockerfile:

```
# docker inspect python
...
        "Layers": [
          "sha256:ce8168f123378f7e04b085c96727170l3d1d28b2aa726361bb132c1c64fe76ac",
...
           "sha256:b6d8f557ceb3cc3b98c9325e031fb4a6dec784264374fbe8c373273216bf5073"
        ]
# docker inspect python-hello
                    "Layers": [
        "sha256:ce8168f123378f7e04b085c96727170l3d1d28b2aa726361bb132c1c64fe76ac",
...
       "sha256:b6d8f557ceb3cc3b98c9325e031fb4a6dec784264374fbe8c373273216bf5073"
"sha256:b6d8f557ceb3cc3b98c9325e031fb4a6dec784264374fbe8c373273216bf5073",
        "sha256:e5bb80204d72a41e766cc5296eed0491335f02049eb874371e8aed29e091215c"
        ]
```

8. Instancie su imagen, deberá tener la imagen siguiente:

```
# docker run python-hello
Hello World !
```

9. ¿Qué pasaría si, en lugar de ese pequeño script, usted hubiera usado un producto como, por ejemplo, django, descargable en la dirección https://www.djangoproject.com/? Con algunas adaptaciones en el Dockerfile (instalación de dependencias, o más bien django, con pip usando un comando **RUN**), habría obtenido un framework web completo.