11_gestion_de_imagenes

October 15, 2020

1 Gestión de imágenes.

El concepto de "contenedor" implica que éstos pueden ser creados con el menor esfuerzo a partir de imágenes disponibles en repositorios locales y en línea.

ADVERTENCIA:

Se da por sentado de que las celdas de los capítulos previos fueron ejecutadas y que existen imágenes cargadas en el sistema.

1.1 Los *registry* y *Docker Hub*.

Una de las funcionalidades más relevantes de Docker es la de poder acceder a determinados repositorios en línea y traer imágenes o *Dockerfiles* para crear contenedores de forma local. A estos concentradores de repositorios se les conoce como *registry*.

1.1.1 *Docker Hub*.

El *registry* al que Docker accede de forma predeterminada es *Docker Hub*.

Docker Hub cuenta con un amplio catálogo de repositorios públicos y permite a cualquier persona darse de alta y crear repositorios propios tanto públicos como privados dependiendo del plan de suscripción. El plan gratuito sólo permite crear repositorios públicos.

Docker Hub se encuentra en https://hub.docker.com/

Cada vez que se utiliza `docker run` el daemon busca la imagen a la que se hace referencia en el sistema local y en caso de no encontrarla, la busca en Docker Hub.

1.1.2 Identificación de una imagen o un *Dockerfile* en un *registry* mediante etiquetas.

Cada repositorio de un *registry* tiene un nombre único y a su vez cada imagen dentro del repositorio tienen un nombre y una etiqueta o *tag*.

El *tag* que se busca y asigna por defecto es *latest*.

Las imágenes de docker pueden ser identificadas en un *registry* usando la siguiente estructura:

```
<nombre del repositorio>/<nombre de la imagen>:<tag>
```

En el caso de acceder a *Docker Hub* sólo es necesario indicar el tag. En caso contrario es necesario indicar la URL del servidor del *registry*.

```
<URL>/<nombre del repositorio>/<nombre de la imagen>:<tag>
```

Ejemplo:

- El repositorio oficial de CentOS es https://hub.docker.com/_/centos y en este repositorio se encuentran varios *Dockerfiles* de diversas versiones de CentOS.
- La estructura que identifica al *Dockerfile* para construir una imagen de CentOS 6.10 es:

`centos:6.10`

1.2 El comando `docker search`.

El comando `docker search` permite realizar una búsqueda de imágenes en los repositorios de *Docker Hub* a partir de un término.

`docker search <argumentos> <termino>`

Referencia: https://docs.docker.com/engine/reference/commandline/search/

Ejemplo:

Se realizará una búsqueda de todas las imágines y Dockerfiles que coincidan con el término *Jupyter*.

[ ]:
```
sudo docker search jupyter
```

ADVERTENCIA: Existe una enorme cantidad de recursos disponibles en *Docker Hub*, pero no todos estos recursos son seguros y sólo una pequeña parte de estos ha sido validado por Docker o por los fabricantes. Se recomienda utilizar únicamente recursos oficiales o de plena confianza.

1.3 El comando `docker image`.

Este comando cuenta a su vez con subcomandos especializados para la gestión de imágenes.

1.3.1 El comando `docker image ls`.

Este comando es idéntico a `docker images` y despliega un listado de las imágenes existentes en un entorno local.

`docker image ls`

Referencia: https://docs.docker.com/engine/reference/commandline/image_ls/

Ejemplo:

[ ]:
```
sudo docker image ls
```

[ ]:
```
sudo docker images
```

1.3.2 El comando `docker image build`.

Este comando es idéntico a `docker build` y permite crear una imagen a partir de un *Dockerfile*.

`docker image build <dockerfile>`

Referencia: https://docs.docker.com/engine/reference/commandline/image_build/

En el capítulo `10_dockerfile.ipynb` se exploró la forma de crear imágenes a partir de un *Dockerfile*.

1.3.3 El comando `docker image pull`.

Este comando es idéntico a `docker pull` y permite acceder a un *registry* y descargar o construir una imagen a partir de un repositorio.

`docker image pull <fuente de la imagen>`

Referencia: https://docs.docker.com/engine/reference/commandline/image_pull/

Ejemplos:

- Se construirá la imagen oficial de la versión 6.10 de CentOS desde *Docker Hub*.

[ ]:
```
sudo docker image pull centos:6.10
```

[ ]:
```
sudo docker images
```

- Se descargará la imagen cloudevel/cd331:latest* localizada en https://hub.docker.com/r/cloudevel/cd331/tags

[ ]:
```
sudo docker image pull cloudevel/cd331:latest
```

[ ]:
```
sudo docker images
```

1.3.4 El comando `docker image inspect`.

Este comando regresa un documento JSON con los detalles de una imagen.

`docker image inspect <imagen>`

Referencia: https://docs.docker.com/engine/reference/commandline/image_inspect/

Ejemplo:

- Se desplegarán los detalles de la imagen `centos:6.10`.

[ ]:
```
sudo docker image inspect centos:6.10
```

1.3.5 El comando `docker image history`.

Este comando despliega la historia de modificaciones de una imagen.

`docker image history <imagen>`

Referencia: https://docs.docker.com/engine/reference/commandline/image_history/

Ejemplo

- Se desplegarán la historia de la imagen `cloudevel/cd331:latest`.

[ ]:
```
sudo docker image history cloudevel/cd331:latest
```

1.3.6 El comando `docker image rm`.

Este comando es idéntico a `docker rmi` y elimina la imagen de un entorno local.

`docker image rm <imagenes>`

Es importante hacer notar que no es posible eliminar una imagen si existen contenedores creados a partir de éstas.

Referencia: https://docs.docker.com/engine/reference/commandline/image_rm/

Ejemplo:

- Se creará y ejecutará un contenedor a partir de la imagen más reciente de la versión 16.04 del repositorio oficial de Ubuntu.

```
[ ]: sudo docker run -dit --name ubuntu-ejemplo ubuntu:16.04
```

```
[ ]: sudo docker images
```

- Se intentará eliminar la imagen `ubuntu:16.04`, pero debido a que existe un contenedor construido a partir de dicha imagen, no será posible.

```
[ ]: sudo docker image rm ubuntu:16.04
```

- Se detendrá y eliminará al contenedor `ubuntu-ejemplo`.
- Se eliminará la imagen `ubuntu:16.04`.

```
[ ]: sudo docker stop ubuntu-ejemplo
```

```
[ ]: sudo docker rm ubuntu-ejemplo
```

```
[ ]: sudo docker image rm ubuntu:16.04
```

```
[ ]: sudo docker images
```

1.3.7 El comando `docker image prune`.

Este comando eliminará todas aquellas imágenes que no estén siendo utilizadas.

`docker image prune`

Referencia: https://docs.docker.com/engine/reference/commandline/image_prune/

Ejemplo:

- Se eliminarán todas las imágenes que no estén siendo utilzadas.

```
[ ]: sudo docker run -dit --name ubuntu-exp ubuntu
```

```
[ ]: sudo docker images
```

```
[ ]: sudo docker image prune -af
```

[ ]:
```
sudo docker images
```

1.3.8 El comando `docker image import`.

Este comando es idéntico a `docker import` y permite importar el contenido de un archivo con extensión `.tar`, desde el cual se puede construir una imagen.

`docker image import <archivo> <nombre>:<tag>`

Referencia: https://docs.docker.com/engine/reference/commandline/image_import/

[ ]:
```
sudo docker images
```

[ ]:
```
sudo docker ps -a
```

[ ]:
```
sudo docker image import nginx.tar servidor:arch1
```

[ ]:
```
sudo docker images
```

1.3.9 El comando `docker image load`.

Este comando es idéntico a `docker load` y es similar a `docker image import`, pero se puede escoger entre un archivo tar o STDIN.

Referencia: https://docs.docker.com/engine/reference/commandline/image_load/

1.3.10 El comando `docker image save`.

Este comando es idéntico a `docker save` y puede guardar una o varias imágenes en un archivo tar o STDOUT.

Referencia: https://docs.docker.com/engine/reference/commandline/image_save/

[ ]:
```
sudo docker image save servidor:arch1 -o servidor.tar
```

1.3.11 El comando `docker image tag`.

Este comando es idéntico a `docker tag` y se utiliza para asignarle nombre y tag a una imagen.

Referencia: https://docs.docker.com/engine/reference/commandline/image_tag/

[ ]:
```
sudo docker images
```

[ ]:
```
sudo docker tag nginx_image cloudevel/nginx:latest
```

[ ]:
```
sudo docker image list
```

1.4 El comando `docker login`.

Es muy importante que los *registry* cuenten con métodos de autenticación que garanticen la seguridad de los usuarios, por lo que para aceder a *Docker Hub* y a *registries* que hayan implementado dichos métodos es necesario presentar las credenciales adecuadas.

El comando `docker login` permite la autenticación de un usuario a un *registry*.

`docker login <argumentos>`

Hay muchas formas de autenticarse mediante `docker login` y se recomienda consultar la siguiente referncia:

https://docs.docker.com/engine/reference/commandline/login/

En caso de que el registro sea exitoso, se creará una sesión en el equipo local que permitiría interacrtuar con el *registry*.

Ejemplo:

- La siguiente celda permitirá acceder a un usuario que ya ha sido registrado en *Docker Hub* utilizando un nombre y una constraeña válida, la cual debe de ser escrita dentro del archivo `contrasena.txt`.
  - El argumento `cloudevel` del parámetro `-u` debe de ser sustituido por un nombre de usuario válido.
  - El parámetro `--password-stdin` permite capturar la contraseña desde la entrada estándar de una terminal. En este caso, este parámetro usará como contraseña el contenido que el comando `cat` extarerá del archivo `contrasena.txt`.

ADVERTENCIA:

Para poder ejecutar varios ejemplos de este capítulo es necesario que el registro de esta celda haya sido exitoso.

[ ]:
```
cat contrasena.txt | sudo docker login -u pythonistaio --password-stdin
```

1.4.1 El comando `docker image push`.

Este comando se utiliza para subir una imagen a un registry.

Referencia: https://docs.docker.com/engine/reference/commandline/image_push/

[ ]:
```
sudo docker pull cloudevel/cd331:latest
```

[ ]:
```
sudo docker images
```

[ ]:
```
sudo docker tag cloudevel/cd331:latest pythonistaio/cd331:2006
```

[ ]:
```
sudo docker images
```

[ ]:
```
sudo docker push pythonistaio/cd331:2006
```

[ ]:
```
sudo docker save pythonistaio/cd331:2006 -o docker.tar
```