Secrets

Primer ejemplo: postgres

En este caso levantaremos un servido de bases de datos con postgres y otro servicio, adminer (herramienta de gestión de bases de datos de código abierto, gratuita y basada en PHP). Necesitamos pasarle el usuario, clave y base de datos a postgres. La primera solución, no recomendable en un entorno de producción, es introducir esos valores en el fichero compose

```
version: '3.1'

services:
  db:
    image: postgres
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mysupersecretpassword
      POSTGRES_DB: mydatabase

  adminer:
    image: adminer
    ports:
     - 8080:8080
```

Para gestionar este tipo de información quenecesita el contenedor en el despliegue se puede usar `docker secret`.

> a secret is a blob of data, such as a password, SSH private key, SSL certificate, or another piece of data that should not be transmitted over a network or stored unencrypted in a Dockerfile or in your application's source code. You can use Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it. Secrets are encrypted during transit and at rest in a Docker swarm. A given secret is only accessible to those services which have been granted explicit access to it, and only while those service tasks are running.

Se puede crear un secreto redireccionando la salida de un comando (o a partir de un fichero). Observe como creamos tres secretos, y a continuación los listamos

```
ubuntu@manager:~/src/secrets$ echo "myuser" | sudo docker secret create pg_user -
nqfkihla5j5n4z1d72via08rq
ubuntu@manager:~/src/secrets$ echo "mysupersecretpassword" | sudo docker secret
create pg_password -
p02zpc0uqjm01b1lyfb76gvdv
ubuntu@manager:~/src/secrets$ echo "mydatabase" | sudo docker secret create
pg_database -
o6o15f0e6vo4da0er7wq7j9j1
ubuntu@manager:~/src/secrets$ sudo docker secret ls
ID                          NAME          DRIVER    CREATED              UPDATED
o6o15f0e6vo4da0er7wq7j9j1   pg_database             28 seconds ago       28
seconds ago
p02zpc0uqjm01b1lyfb76gvdv   pg_password             39 seconds ago       39
seconds ago
nqfkihla5j5n4z1d72via08rq   pg_user                 About a minute ago   About a
minute ago
ubuntu@manager:~/src/secrets$
```

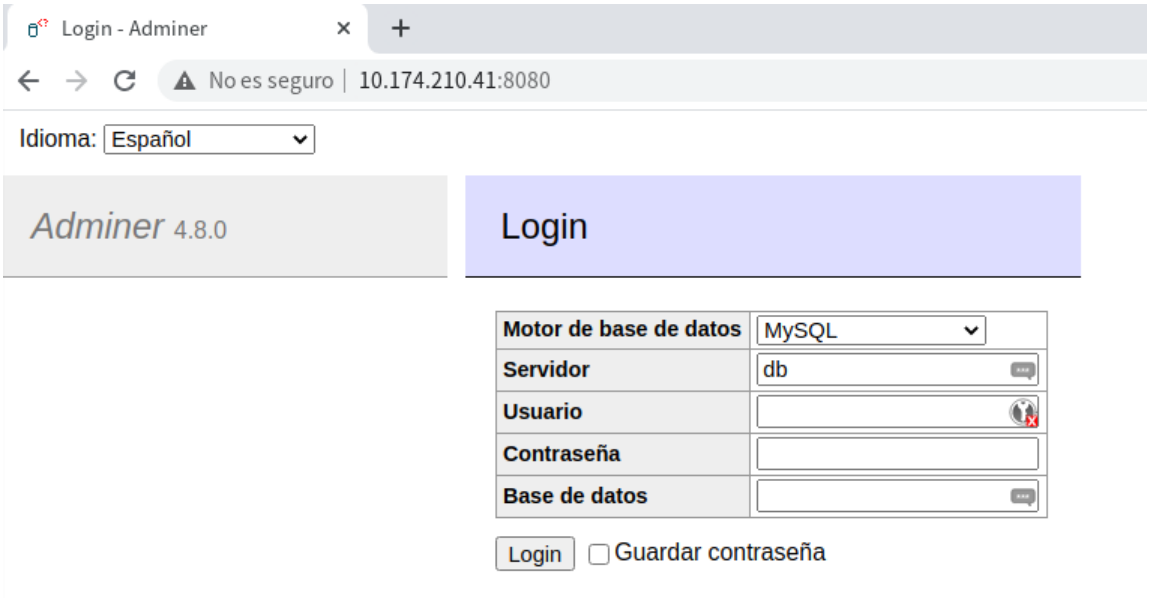
Una vez creado los secretos creamos un fichero docker-compose que los utilize. Los secretos se montan (por defecto) en el directorio `/run/secrets` del contenedor. Por eso las variables de entorno en el fichero compose se asocian a los ficheros `/run/secrets/pg_user`, etc. En el bloque de secrets se indica su procedencia, externa (veremos en otro ejemplo como usar un fichero)

```
~/src/secrets$ cat docker-compose-postgres.yml
version: '3.1'

services:
    db:
        image: postgres
        restart: always
        environment:
            POSTGRES_USER_FILE: /run/secrets/pg_user
            POSTGRES_PASSWORD_FILE: /run/secrets/pg_password
            POSTGRES_DB_FILE: /run/secrets/pg_database
        secrets:
           - pg_password
           - pg_user
           - pg_database

    adminer:
        image: adminer
        ports:
         - 8080:8080

secrets:
  pg_user:
    external: true
  pg_password:
    external: true
  pg_database:
    external: true
```

Ya sólo nos queda lanzar el stack. docker secret es un servicio que se debe ejecutar en un cluster, leugo debe iniciar primero swarm

```
ubuntu@manager:~/src/secrets$  sudo docker swarm init
....
ubuntu@manager:~/src/secrets$ sudo docker stack deploy -c docker-compose-
postgres.yml postgres
Ignoring unsupported options: restart

Creating network postgres_default
Creating service postgres_adminer
Creating service postgres_db
ubuntu@manager:~/src/secrets$ sudo docker stack  ls
NAME       SERVICES   ORCHESTRATOR
postgres   2          Swarm
ubuntu@manager:~/src/secrets$  sudo docker stack services postgres
ID             NAME               MODE         REPLICAS   IMAGE             PORTS
xmek7dchrth1   postgres_adminer   replicated   1/1        adminer:latest
*:8080->8080/tcp
b2c4jkyfqq9u   postgres_db        replicated   1/1        postgres:latest
ubuntu@manager:~/src/secrets$
```

Y ahora puede acceder a adminer usando la IP del manager



Segundo ejemplo: wordpress

https://docs.docker.com/engine/swarm/secrets

En este caso los secreto se comparten entre los dos servicios, y vamos a almacenar las claves en un fichero externo al contenedor que se añade a través de un secreto.

Creamos los ficheros con las claves:

```
ubuntu@manager:~/src/secrets$ openssl rand -base64 20 > db_password.txt
ubuntu@manager:~/src/secrets$ openssl rand -base64 20 > db_root_password.txt
ubuntu@manager:~/src/secrets$ cat db_password.txt db_root_password.txt
g51FhZPdz8Z+VbZ3noRXB64csOM=
oahfswhEygpzGMkKYQQ/OZAKBKE=
ubuntu@manager:~/src/secrets$
```

El fichero compose sería:

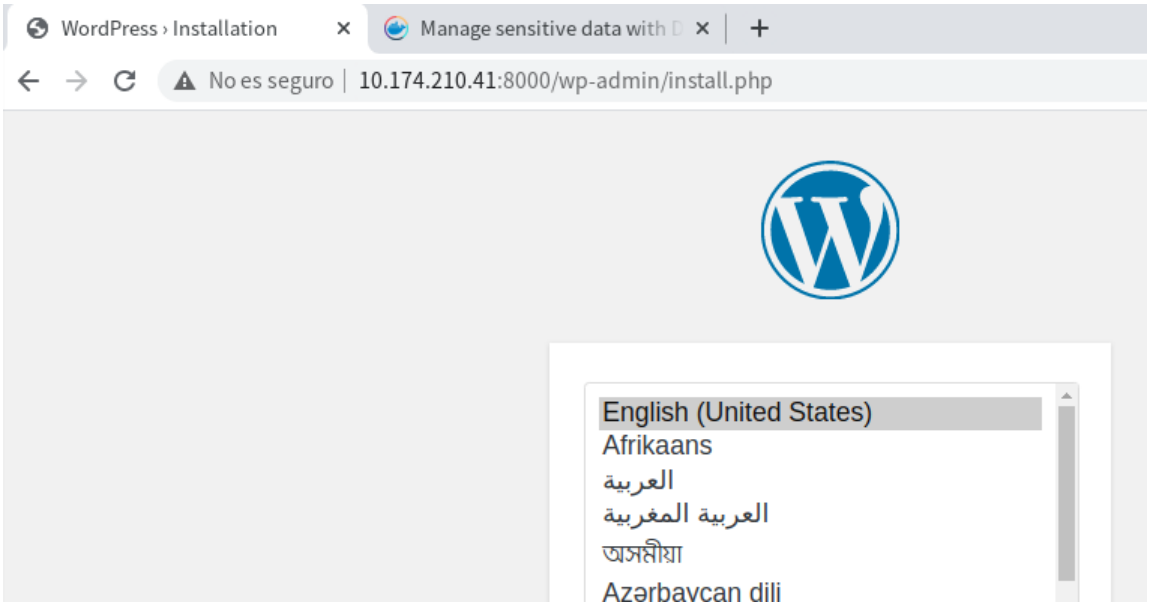
```
version: "3.3"

services:
   db:
     image: mysql:latest
     volumes:
       - db_data:/var/lib/mysql
     environment:
       MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD_FILE: /run/secrets/db_password
     secrets:
       - db_root_password
       - db_password

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD_FILE: /run/secrets/db_password
     secrets:
       - db_password


secrets:
   db_password:
     file: db_password.txt
   db_root_password:
     file: db_root_password.txt

volumes:
    db_data:
```

Y para desplegarlo

```
ubuntu@manager:~/src/secrets$ sudo docker stack deploy -c docker-compose.yml wp
Creating network wp_default
Creating secret wp_db_root_password
Creating secret wp_db_password
Creating service wp_wordpress
Creating service wp_db
ubuntu@manager:~/src/secrets$ sudo docker stack ls
NAME       SERVICES   ORCHESTRATOR
postgres   2          Swarm
wp         2          Swarm
```

```
ubuntu@manager:~/src/secrets$ sudo docker stack ps wp
ID             NAME             IMAGE              NODE      DESIRED STATE
CURRENT STATE           ERROR     PORTS
lm0sapddofwz   wp_db.1          mysql:latest       manager   Running
Running 3 seconds ago
vh98qd27rvfa   wp_wordpress.1   wordpress:latest   manager   Running
Running 3 seconds ago
ubuntu@manager:~/src/secrets$
```

Si accede a la IP del manager pueto 8000 accederá a la página inicial de configuración de wordpress:



No tan secreto: config

Si la información nos es "sensible" es posible usar config. En este enlace tiene un ejemplo de uso de config para almacenar el fichero de inicio del servidor

https://blog.ruanbekker.com/blog/2019/02/28/use-swarm-managed-configs-in-docker-swarm-to-store-your-application-configs/

Publicar (compartir) imagen en dockerhub

Es necesario tener un cuenta en docker. Tiene un ejemplo en el tutorial de inicio, apartado 4.

https://docs.docker.com/get-started/04_sharing_app/

Si ya tiene creada la imagen

```
~/src/app$ docker images
REPOSITORY          TAG      IMAGE ID    CREATED      SIZE
getting-started       latest    810dee6c806c  5 seconds ago   223MB
...
```

Los pasos son

1. Acceder a https://hub.docker.com/

2. Crear un repositorio de tipo público, por ejemplo
    1. nombre: getting-started
    2. Público
    3. Create
3. Acceder desde CLI a dockerhub

```
~/src/app$ docker login -u <user>
```

4. Modificar la etiqueta (tag) de la imagen local (es necesario que la etiqueta de la imagen a subir tenga el formato user/etiqueta)

```
~/src/app$ docker tag getting-started <user>/getting-started
```

5. Subir la imagen

```
~/src/app$ docker push <user>/getting-started-test`
```

6. Salir de la sesión con docker logout

```
~/src/app$ docker logout
```

Otra opción para no tener que modificar la etiqueta de la imagen es haberla creado con la etiqueta correcta desde el principio si pensaba compartirla:

```
~/src/app$ docker build -t <user>/test .
```

Y en este enlace tiene otro ejemplo similar: https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

> The goal of this exercise is to create a Docker image which will run a Flask app.
>
> 1. Create a Python Flask app
> 2. Write a Dockerfile
> 3. Build the image
> 4. test the image
> 5. Push your image to hub.docker.com

Crear un registro docker local

Se puede mantener un registro propio para distribur sus imágenes en local

```
$ docker service create --name registry --publish published=5000,target=5000
registry:2
$ docker service ls
wbgfqi6e93vgatef1pbklmggm
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
getting-started           latest    810dee6c806c   45 minutes ago   223MB
registry                  <none>    678dfa38fcfa   2 months ago     26.2MB
$
```

Usaremos una de las imágenes ya descargadas para añadir al registro local. Primero debe etiquetarla con el nombre (o IP) y puerto del servicio (*when the first part of the tag is a hostname and port, Docker interprets this as the location of a registry, when pushing)*.

```
$ docker tag getting-started:latest localhost:5000/getting-started
$ docker images
REPOSITORY                        TAG       IMAGE ID       CREATED          SIZE
getting-started                   latest    810dee6c806c   51 minutes ago   223MB
localhost:5000/getting-started    latest    810dee6c806c   51 minutes ago   223MB
registry                          <none>    678dfa38fcfa   2 months ago     26.2MB
$
```

```
$ docker push localhost:5000/getting-started
Using default tag: latest
The push refers to repository [localhost:5000/getting-started]
ed3bd20764dc: Pushed
16fa28787de9: Pushed
edae3f83f629: Pushed
4992425f60b2: Pushed
729a2badad91: Pushed
ec7c69516687: Pushed
0fcbbeeeb0d7: Pushed
latest: digest:
sha256:9d0cd8fbbe871a90cd896a9c7ce90f5bee40ba5545a16924aae70b59dee30da4 size:
1789
$
```

Borramos la imagen local:

```
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
registry     <none>    678dfa38fcfa   2 months ago     26.2MB
$
```

Y hacemos una prueba de descarga local:

```
$ docker pull localhost:5000/getting-started
Using default tag: latest
latest: Pulling from getting-started
...
Digest: sha256:9d0cd8fbbe871a90cd896a9c7ce90f5bee40ba5545a16924aae70b59dee30da4
Status: Downloaded newer image for localhost:5000/getting-started:latest
localhost:5000/getting-started:latest
$
```

Para eliminar el servicio de registro

```
$ docker service ls
ID             NAME       MODE         REPLICAS   IMAGE        PORTS
wbgfqi6e93vg   registry   replicated   1/1        registry:2   *:5000->5000/tcp
$ docker service rm registry
registry
$ docker service ls
ID        NAME      MODE      REPLICAS   IMAGE     PORTS
$ docker pull localhost:5000/getting-started
Using default tag: latest
Error response from daemon: Get http://localhost:5000/v2/: dial tcp
127.0.0.1:5000: connect: connection refused
$
```

Más información:

- https://docs.docker.com/registry/deploying/