Instalacion Debian

fuente: https://docs.docker.com/engine/installation/linux/debian

Cambiar directorio donde se instalan las imagenes:

https://forums.docker.com/t/how-do-i-change-the-docker-image-installation-directory/1169

Configuraciones

La forma recomendada de cambiar configuraciones por defecto es crear el archivo:

```
sudo vim /etc/docker/daemon.json
```

Cambiar rango IP de los contenedores

1. Editar:

```
sudo vim /etc/docker/daemon.json
```

Y añadir:

```
{
  "bip": "10.59.0.1/16"
}
```

2. Reiniciar el servicio:

```
sudo service docker restart
```

Habilitar debug

https://success.docker.com/article/how-do-i-enable-debug-logging-of-the-docker-daemon

1. Editar:

```
sudo vim /etc/docker/daemon.json
```

Y añadir:

```
{
    "debug": true
```

```
}
```

2. Recargar el servicio:

```
sudo kill -SIGHUP $(pidof dockerd)
```

3. Ver logs

```
sudo tail -F /var/log/daemon.log
```

Ejecutar como no root

Para que el usuario “usuario” puede ejecutar docker sin sudo:

1. Añadirle al grupo “docker”

```
sudo usermod -a -G docker usuario
```

2. Cerrar sesión bash. Quizá lo más drástico es cerrar window manager (Gnome, Awesome, etc…)

Crear imagenes

Modificar tamaño de la imagen base

Por defecto la imagen es de 10Gb. Lo podemos ver haciendo:

```
# docker info

Base Device Size: 10 GB
```

Para aumentarlo, paramos docker y lo arrancamos con el siguiente parámetro:

```
# service docker stop
```

Cambiamos /lib/systemd/system/docker.service

```
ExecStart=/usr/bin/docker daemon --storage-opt dm.basesize=20G -H fd://
```

Actualizamos servicio por systemd:

```
systemctl daemon-reload
```

Ahora ya está a 20Gb:

```
# docker info

Base Device Size: 21.47 GB
```

Actualizar imagen previa

1. Arrancar contenedor

```
docker run -t -i training/sinatra /bin/bash
```

2. Hacer algun cambio

```
gem install json
```

3. Hacer commit

```
docker commit -m "Added json gem" -a "Kate Smith" \
0b2616b0e5a8 ouruser/sinatra:v2
```

Desde arhivo de config

1. Crear archivo

```
mkdir sinatra
cd sinatra
vim Dockerfile
```

2. Con el siguiente contenido:

```
# This is a comment
FROM ubuntu:14.04
MAINTAINER Kate Smith <ksmith@example.com>
RUN apt-get update && apt-get install -y ruby ruby-dev
RUN gem install sinatra
```

3. Generar la imagen. El parámetro -t es el nombre y el tag que le ponemos. El . es porque estamos en el directorio, indica la ruta al fichero Dockerfile

```
docker build . -t ouruser/sinatra:v2
```

En un directorio diferente:

```
docker build <path>/sinatra -t ouruser/sinatra:v2
```

Y si tenemos otro nombre de fichero:

```
docker build -f DockerfilePrueba . -t ouruser/sinatra:v2
```

Entrar en un container como root

Si tenemos cambiado el usuario con el que se ejecuta docker podemos entrar como root de esta manera y luego cambiar la contraseña si nos conviene:

```
docker exec -u 0 -it mycontainer bash
```

```
docker exec -u root -it mycontainer bash
```

Cambiar politica de reinicio de contenedor

https://docs.docker.com/config/containers/start-containers-automatically/

Miramos que política tiene:

```
docker inspect container_name| jq -r '.[0].HostConfig.RestartPolicy'
```

```
{
  "Name": "always",
  "MaximumRetryCount": 0
}
```

Flag	Description
no	Do not automatically restart the container. (the default)
on-failure	Restart the container if it exits due to an error, which manifests as a non-zero exit code.
always	Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)
unless-stopped	Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

Para cambiarlo

```
docker update --restart unless-stopped container_name
```

Red

Hay 2 cambios importantes que se estan produciendo ahora mismo (ENE/2016):

- Network. Caracteristica soportada por la version 1.9. Dejara obsoleta la opcion “–link”

- IPs estaticas para contenedores. Esta previsto introducirla en la version 1.10, prevista para febrero de 2016

Hay un excelente artículo que las explica:

https://docs.docker.com/network/network-tutorial-standalone/#use-the-default-bridge-network

Visibilidad contenedor_a <-> contenedor_b

La opción "–link" debe evitarse, está en desuso.

Hay que resolver dos cuestiones:

1. Que haya conectividad a nivel de red entre contenedores 2. Que cada contenedor pueda acceder por nombre al otro contenedor.

IMPORTANTE: la gracia es que si se destruye un contenedor y se vuelve a crear, se pueda detener y arrancar el otro contenedor. Con "–link" si se destruye el contenedor "enlazado", al detener el contenedor que lo referenciaba e intentar arrancarlo de nuevo NO podremos.

Sin docker compose

En este ejemplo:

- Tenemos 3 contenedores 3, "sever_a", "server_b" y "server_c"
- server_a y server_b tienen conectividad
- Desde server_a se puede llegar a server_b por nombre
- Desde server_b se puede llegar a server_a por nombre
- Si se destruye server_b, se vuelve a crear y se detiene server_a, se podrá volver a arrancar server_a
- server_c no llega a nivel de red (ni resuelve el nombre) de server_a ni de server_b

1. Crear la red

```
docker network create network-private
```

2. Creamos los 3 contenedores 3

```
docker run --name server_a \
 -ti \
 --network network-private \
 --net-alias server_a \
 -d debian
```

```
docker run --name server_b \
 -ti \
 --network network-private \
 --net-alias server_b \
```

```
 -d debian
```

```
docker run --name server_c \
 -ti \
 -d debian
```

3. Desde “server_a” llegamos a “server_b” por nombre:

3.1. Nos conectamos

```
docker exec -ti server_a bash
```

3.2. Ejecutamos:

```
ping server_b
```

Resultado esperado similar a:

```
PING server_b (192.168.224.3) 56(84) bytes of data.
64 bytes from server_b.network-private (192.168.224.3): icmp_seq=1 ttl=64
time=0.222 ms
64 bytes from server_b.network-private (192.168.224.3): icmp_seq=2 ttl=64
time=0.134 ms
^C
--- server_b ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 0.134/0.178/0.222/0.044 ms
root@34e909656151:/#
```

3.3. Salimos

```
exit
```

4. Re-creamos “server_b”

```
docker stop server_b && docker rm server_b
```

```
docker run --name server_b \
 -ti \
 --network network-private \
 --net-alias server_b \
 -d debian
```

5. Detenemos y arrancamos “server_a”

```
docker stop server_a
```

Al arrancarlo:

```
docker stop server_a
```

NO nos da ningún error. Si hubiésemos usado la opción “–link” no hubiéramos podido arrancarlo

6. Repetir el paso 3. Debemos obtener el mismo resultado

7. Limpiamos

```
docker stop server_a && docker rm server_a
docker stop server_b && docker rm server_b
docker stop server_c && docker rm server_c
docker network rm network-private
```

Con docker compose

1. Crear el siguiente archivo:

```
version: '3.7'
services:

 server_a:
  container_name: server_a
  image: debian
  stdin_open: true
  tty: true
  networks:
   network-private:
    aliases:
    - server_a

 server_b:
  container_name: server_b
  image: debian
  stdin_open: true
  tty: true
  networks:
   network-private:
    aliases:
    - server_b

 server_c:
  container_name: server_c
  image: debian
  stdin_open: true
  tty: true
networks:
  network-private:
    name: network-private
```

2. Levantar el entorno:

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

3. Realizar las mismas pruebas que en el apartado anterior (re-crear el contenedor se tendrá que hacer a mano, sin docker-compose)

4. Para limpiar:

4.1. Eliminar a mano “server_b”

```
docker stop server_b && docker rm server_b
```

4.2. Pararlo todo

```
docker-compose down
```

Puertos

Para especificar, por ejemplo, que el puerto 8000 del host se conecta al puerto 80 del contenedor:

```
docker run -d -p 8000:80 training/webapp python app.py
```

Sistema de archivos

- Montar un volumen del anfitrion como volumen de datos en el contenedor:

```
docker run -d -P --name web -v /host/dir:/container/dir training/webapp
python app.py
```

- Crear y montar un contenedor de volumen de datos

```
docker create -v /dbdata --name dbdata training/postgres /bin/true
docker run -d --volumes-from dbdata --name db1 training/postgres
docker run -d --volumes-from dbdata --name db2 training/postgres
etc...
```

Comandos varios

https://coderwall.com/p/ewk0mq/stop-remove-all-docker-containers

- One liner to stop / remove all of Docker containers:

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

- Obtener todos los contenedores:

```
docker ps -a -q
```

Comandos varios, luego se iran explicando:

```
docker run -i -t ubuntu /bin/bash
```

```
docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world;
sleep 1; done"
```

- Pre-descargar una imagen

```
docker pull centos
```

- Conectarse a un contenedor

```
docker exec -u root -it container_name bash
```

- Liberar espacio eliminando imagenes:

```
docker images --no-trunc| grep none | awk '{print $3}' | xargs -r docker rmi
```

- Eliminar imagenes:

```
docker rmi mi_imagen
```

- Obtener IP local del contenedor, una vez conocido el container ID

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' container_id
```

- Obtener puertos que tiene abiertos internamente (con jq):

```
docker inspect container_id |jq .[].NetworkSettings.Ports
```

- Renombrar container:

```
docker rename CONTAINER NEW_NAME
```

- Conectarse al terminal de un contenedor (debe haberse arrancado con -ti):

```
docker attach my_container
```

- Desconectarse de la consola sin detener el contenedor:

```
CTRL + p + q
```

Imagenes

A continuacion una lista de imagenes y servicios y sus particularidades

Odoo

https://hub.docker.com/_/odoo/

8.x

Punto de partida:

```
/srv/docker/data
.
+-- odoo-8
|   +-- conf
|       +-- openerp-server.conf
+-- odoo-9
|   +-- conf
|       +-- openerp-server.conf
+-- postgres-odoo-8
|   +-- data
+-- postgres-odoo-9
    +-- data
```

Archivo de configuracion de Odoo:

```
/srv/docker/data/odoo-9/conf/openerp-server.conf
```

```
[options]
addons_path = /usr/lib/python2.7/dist-packages/openerp/addons,/mnt/extra-
addons
data_dir = /var/lib/odoo
auto_reload = True
db_user = user
db_password = secret
; admin_passwd = admin
; csv_internal_sep = ,
; db_maxconn = 64
; db_name = False
; db_template = template1
; dbfilter = .*
; debug_mode = False
; email_from = False
; limit_memory_hard = 2684354560
; limit_memory_soft = 2147483648
; limit_request = 8192
; limit_time_cpu = 60
; limit_time_real = 120
; list_db = True
; log_db = False
; log_handler = [':INFO']
```

```
; log_level = info
; logfile = None
; longpolling_port = 8072
; max_cron_threads = 2
; osv_memory_age_limit = 1.0
; osv_memory_count_limit = False
; smtp_password = False
; smtp_port = 25
; smtp_server = localhost
; smtp_ssl = False
; smtp_user = False
; workers = 0
; xmlrpc = True
; xmlrpc_interface =
; xmlrpc_port = 8069
; xmlrpcs = True
; xmlrpcs_interface =
; xmlrpcs_port = 8071
```

Arrancar (por soleares) un contenedor con Postgres. En este ejemplo uso un directorio del host para almacenar la base de datos y el mismo usuario que hemos definido antes:

```
docker run -d -e POSTGRES_USER=user -e POSTGRES_PASSWORD=secret -v
/srv/docker/data/postgres-odoo-9/data:/var/lib/postgresql/data --name
postgres-odoo-9 postgres
```

Arrancar contenedor con odoo apuntando a ese contenedor postgresql y al archivo de config:

```
docker run -p 127.0.0.1:8069:8069 --link postgres-odoo-9:db -t -v
/srv/docker/data/odoo-9/conf:/etc/odoo --name odoo-9 odoo
```

Probar:

```
http://localhost:8069
```

10.x

Creo que no hay cambio respecto a la 8.0:

1. Lanzar un contender con la base de datos Postgre:

```
docker run --name odoo10-db \
 -e POSTGRES_USER=odoo \
 -e POSTGRES_PASSWORD=odoo \
 -v /home/usuario/data/docker/data/odoo10-db:/var/lib/postgresql/data \
 -d postgres:9.4
```

2. Lanzar el contenedor con Odoo:

```
docker run --name odoo10 \
 -p 8069:8069 \
 --link odoo10-db:db \
 -v /home/usuario/data/docker/data/odoo10:/etc/odoo \
 -d odoo
```

3. Acceder:

http://localhost:8069

PHP

- Escucha en el puerto 8000
- Dejar en el directorio del host "/srv/docker/5-apache" las webs a servir
- Enlazo con un docker "mariadb" para poder usarlo como base de datos, le llamaré "mysql"

```
docker run --name 7-apache \
 -p 8000:80 \
 --link mariadb:mysql \
 -v /home/usuario/data/docker/data/7-apache:/var/www/html \
 -d php:7-apache
```

Instalar módulos PHP

AVISO: si se usa un php.ini propio cascará la instalación, habrá que referenciar en ese archivo las rutas a los .po que toque

Instalar bibliotecas necesarias para trabajar con MySQL/MariaDB:

1. Nos conectamos al contenedor

```
docker exec -u root -it 5-apache bash
```

2. Instalamos (ver documentación de esa imagen de docker)

- Para imagen de PHP 5

```
docker-php-ext-install mysql mysqli pdo pdo_mysql
```

- Para imagen de PHP 7

```
docker-php-ext-install mysqli pdo pdo_mysql
```

2.1. **Caso especial: GD**

```
apt-get update
apt-get install libpng-dev
docker-php-ext-install gd
```

2.2. **Caso especial: mcrypt**

* Para imagen PHP <7.2

```
apt-get update
apt-get install libmcrypt-dev
docker-php-ext-install mcrypt
```

* Para imagen PHP >=7.2

```
pecl install mcrypt-1.0.1
docker-php-ext-install mcrypt
```

TODO: (PHP >=7.2) ver si se ha instalado realmente o no:

```
install ok: channel://pecl.php.net/mcrypt-1.0.1
configuration option "php_ini" is not set to php.ini location
You should add "extension=mcrypt.so" to php.ini
```

3. Salimos

```
exit
```

4. (Desde el host) Reiniciamos el contenedor para que los cambios tomen efecto:

```
docker restart 5-apache
```

Habilitar mod_rewrite

```
docker exec 5-apache a2enmod rewrite
docker exec 5-apache service apache2 restart
```

A mi se me paró el contenedor, arrancarlo pues:

```
docker start 5-apache
```

MariaDB

- Almacena las bases de datos en el directorio del host “/srv/docker/data/mariadb/databases”
- Toma config adicional del directorio “/srv/docker/data/mariadb/config”:

```
+-- mariadb
    +-- config
    |   +-- custom.cnf
    +-- databases
```

custom.cnf

```
[mysqld]
bind-address=0.0.0.0
```

```
docker run --name mariadb \
 -v /home/usuario/data/docker/data/mariadb/databases:/var/lib/mysql \
 -v /home/usuario/data/docker/data/mariadb/config:/etc/mysql/conf.d \
 -e MYSQL_ROOT_PASSWORD=mysecret \
 -d mariadb:latest
```

- Para hacer una ingesta de archivo .sql:

```
docker exec -i mariadb mysql -u root -psecret < /path/to/file.sql
```

- Para interactuar vía cliente mysql (ya NO es necesario el “export TERM=dumb”):

```
docker exec -ti mariadb bash
mysql -u root -p
grant all on db.* to 'db'@'%' identified by 'db';
flush privileges;
exit;
```

Configuración que viene por defecto:

```
# MariaDB database server configuration file.
#
# You can copy this file to one of:
# - "/etc/mysql/my.cnf" to set global options,
# - "~/.my.cnf" to set user-specific options.
#
# One can use all long options that the program supports.
# Run program with --help to get a list of available options and with
# --print-defaults to see which it would actually understand and use.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html

# This will be passed to all mysql clients
# It has been reported that passwords should be enclosed with ticks/quotes
# escpecially if they contain "#" chars...
# Remember to edit /etc/mysql/debian.cnf when changing the socket location.
[client]
port        = 3306
socket      = /var/run/mysqld/mysqld.sock

# Here is entries for some specific programs
# The following values assume you have at least 32M ram

# This was formally known as [safe_mysqld]. Both versions are currently
parsed.
[mysqld_safe]
socket      = /var/run/mysqld/mysqld.sock
```

```
nice        = 0

[mysqld]
skip-host-cache
skip-name-resolve
#
# * Basic Settings
#
#user       = mysql
pid-file    = /var/run/mysqld/mysqld.pid
socket      = /var/run/mysqld/mysqld.sock
port        = 3306
basedir     = /usr
datadir     = /var/lib/mysql
tmpdir      = /tmp
lc_messages_dir = /usr/share/mysql
lc_messages = en_US
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address       = 127.0.0.1
#
# * Fine Tuning
#
max_connections     = 100
connect_timeout     = 5
wait_timeout        = 600
max_allowed_packet  = 16M
thread_cache_size       = 128
sort_buffer_size    = 4M
bulk_insert_buffer_size = 16M
tmp_table_size      = 32M
max_heap_table_size = 32M
#
# * MyISAM
#
# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched. On error, make copy and try a repair.
myisam_recover_options = BACKUP
key_buffer_size     = 128M
#open-files-limit   = 2000
table_open_cache    = 400
myisam_sort_buffer_size = 512M
concurrent_insert   = 2
read_buffer_size    = 2M
read_rnd_buffer_size    = 1M
#
# * Query Cache Configuration
#
# Cache only tiny result sets, so we can fit more in the query cache.
```

```
query_cache_limit       = 128K
query_cache_size        = 64M
# for more write intensive setups, set to DEMAND or OFF
#query_cache_type       = DEMAND
#
# * Logging and Replication
#
# Both location gets rotated by the cronjob.
# Be aware that this log type is a performance killer.
# As of 5.1 you can enable the log at runtime!
#general_log_file        = /var/log/mysql/mysql.log
#general_log             = 1
#
# Error logging goes to syslog due to
/etc/mysql/conf.d/mysqld_safe_syslog.cnf.
#
# we do want to know about network errors and such
#log_warnings       = 2
#
# Enable the slow query log to see queries with especially long duration
#slow_query_log[={0|1}]
slow_query_log_file = /var/log/mysql/mariadb-slow.log
long_query_time = 10
#log_slow_rate_limit    = 1000
#log_slow_verbosity = query_plan

#log-queries-not-using-indexes
#log_slow_admin_statements
#
# The following can be used as easy to replay backup logs or for
replication.
# note: if you are setting up a replication slave, see README.Debian about
#       other settings you may need to change.
#server-id      = 1
#report_host        = master1
#auto_increment_increment = 2
#auto_increment_offset  = 1
#log_bin            = /var/log/mysql/mariadb-bin
#log_bin_index      = /var/log/mysql/mariadb-bin.index
# not fab for performance, but safer
#sync_binlog        = 1
expire_logs_days    = 10
max_binlog_size         = 100M
# slaves
#relay_log      = /var/log/mysql/relay-bin
#relay_log_index    = /var/log/mysql/relay-bin.index
#relay_log_info_file    = /var/log/mysql/relay-bin.info
#log_slave_updates
#read_only
#
# If applications support it, this stricter sql_mode prevents some
```

```
# mistakes like inserting invalid dates etc.
#sql_mode       = NO_ENGINE_SUBSTITUTION,TRADITIONAL
#
# * InnoDB
#
# InnoDB is enabled by default with a 10MB datafile in /var/lib/mysql/.
# Read the manual for more InnoDB related options. There are many!
default_storage_engine  = InnoDB
# you can't just change log file size, requires special procedure
#innodb_log_file_size   = 50M
innodb_buffer_pool_size = 256M
innodb_log_buffer_size  = 8M
innodb_file_per_table   = 1
innodb_open_files   = 400
innodb_io_capacity  = 400
innodb_flush_method = O_DIRECT
#
# * Security Features
#
# Read the manual, too, if you want chroot!
# chroot = /var/lib/mysql/
#
# For generating SSL certificates I recommend the OpenSSL GUI "tinyca".
#
# ssl-ca=/etc/mysql/cacert.pem
# ssl-cert=/etc/mysql/server-cert.pem
# ssl-key=/etc/mysql/server-key.pem

#
# * Galera-related settings
#
[galera]
# Mandatory settings
#wsrep_on=ON
#wsrep_provider=
#wsrep_cluster_address=
#binlog_format=row
#default_storage_engine=InnoDB
#innodb_autoinc_lock_mode=2
#
# Allow server to accept connections on all interfaces.
#
#bind-address=0.0.0.0
#
# Optional setting
#wsrep_slave_threads=1
#innodb_flush_log_at_trx_commit=0

[mysqldump]
quick
quote-names
```

```
max_allowed_packet  = 16M

[mysql]
#no-auto-rehash # faster start of mysql but no tab completion

[isamchk]
key_buffer      = 16M

#
# * IMPORTANT: Additional settings that can override those from this file!
#   The files must end with '.cnf', otherwise they'll be ignored.
#
!includedir /etc/mysql/conf.d/
```

Zabbix

Zabbix Server

https://hub.docker.com/r/zabbix/zabbix-server-mysql/

```
docker run --name zabbix-server \
 --link mariadb:mysql  \
 -e DB_SERVER_HOST="mariadb" \
 --link mariadb:mysql \
 -e MYSQL_DATABASE="zabbix_server" \
 -e MYSQL_USER="zabbix_server" \
 -e MYSQL_PASSWORD="zabbix_server" \
 -v /home/usuario/data/docker/data/zabbix-
server/externalscripts:/usr/lib/zabbix/externalscripts \
 -d zabbix/zabbix-server-mysql
```

Zabbix Web

```
docker run --name zabbix-web \
 --link zabbix-server:zabbix-server \
 --link mariadb:mysql  \
 -e DB_SERVER_HOST="mariadb" \
 -e MYSQL_DATABASE="zabbix_server" \
 -e MYSQL_USER="zabbix_server" \
 -e MYSQL_PASSWORD="zabbix_server" \
 -e ZBX_SERVER_HOST="zabbix-server" \
 -e TZ="Europe/Madrid" \
 -p 8000:80 \
 -d zabbix/zabbix-web-apache-mysql
```

La base de datos que usa es la que provee el Zabbix Server al que se enlace.

Se puede acceder a través de:

http://localhost:8000

Usuario: Admin Contraseña: zabbix

Repositorios

hub.docker.com

Tenemos una imagen y la queremos subir a nuestro repositorio hub de docker:

```
# docker images
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
oraclelinux                         6.6                 9636e42b38e2
5 months ago        157.7 MB
```

Le cambiamos el nombre al de nuestro repositorio:

```
# docker tag 9636e42b38e2 iwanttobefreak/weblogic121
```

Nos logamos en nuestro repositorio:

```
# docker login --username=<usuario> --email=<correo>
```

Subimos la imagen:

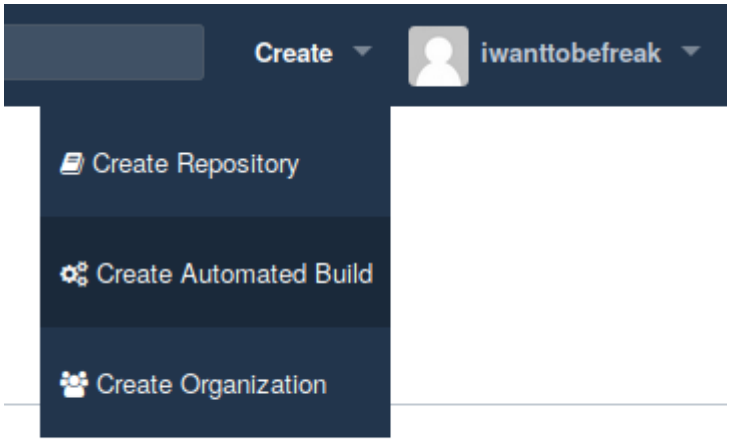
```
# docker push iwanttobefreak/weblogic121
The push refers to a repository [docker.io/iwanttobefreak/weblogic121]
5f70bf18a086: Pushed
3b101413fff5: Pushed
latest: digest:
sha256:9acc7e03325faa04fe466d9b2c22c5241fb8c0c276e3bbcfeaa66714c894f98e
size: 2
```

Nos crea un repositorio en nuestro hub de docker

Asociar hub de docker con github

Si queremos mantener nuestro docker desde github y que se generen las imagenes automáticamente: **Automated Build**

Primero creamos nuestro repositorio en GitHub. Luego lo creamos en docker.Vamos a Create/Create Automated Build



Seleccionamos GitHub y nuestro repositorio de github. Ponemos descripición y pulsamos create

Repositorio en local y GITHUB

Para descargar un repositorio de GITHUB:

```
git clone https://github.com/iwanttobefreak/docker-weblogic1036.git
```

Para subir el repositorio:

```
git add <fichero>
git commit -m "cambio realizado"
git push
```

Borrar containers e imagenes que no se usan

Borra los containers que no tengan guión “-” en el nombre y las imágenes sin repositorio:

```
#!/bin/bash
docker ps -a | awk {'print $NF'}| grep -v -E '\-|NAMES' | xargs -i docker rm
{}
docker images | awk '{if ( $1 == "<none>" ) printf $3"\n"}' | xargs -i
docker rmi {}
```

Script de borrado de imagenes y containers huérfanos

```
#!/bin/bash

while read linea
do
  docker rmi -f $linea
done < <(docker images | awk '{if ( $2 == "<none>" ) printf $3"\n"}')
```

```
while read linea
do
  imageid=`echo $linea|awk {'print $2'}`
  docker images| grep $imageid
  if [ `echo $?` -eq 1 ]
  then
    containerid=`echo $linea|awk {'print $1'}`
    echo "Borrando container: "$containerid" "$imageid
    docker rm $containerid
  fi
done < <(docker ps -a )
```

Docker en Glusterfs

Para poder correr docker en glusterfs cambiamos el sistema de archivos de aufs a devicemapper

Si hacemos

```
# docker info
```

```
Storage Driver: aufs
```

Lo cambiamos a devicemapper. Si usamos systemd es en el fichero:

```
/etc/systemd/system/docker.service
```

Cambiamos –storage-driver a devicemapper

```
[Service]
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2376 -H
unix:///var/run/docker.sock --storage-driver devicemapper --tlsverify --
tlscacert /etc/docker/ca.pem --tlscert /etc/docker/server.pem --tlskey
/etc/docker/server-key.pem --label provider=generic
```

Recargamos systemctl y reiniciamos docker

```
systemctl daemon-reload
/etc/init.d/docker restart
```

Ya podemos escribir en glusterfs:

```
Storage Driver: devicemapper
```

Arrancar contenedores automáticamente

https://docs.docker.com/engine/admin/start-containers-automatically/

```
docker update --restart=unless-stopped mycontainer
```

Docker compose

https://docs.docker.com/compose/

Apache Php y Mysql

Levanta un frontend con apache y php que conecta con otro contendedor con mysql:

```
docker-compose.yml
```

```
version: '2'
services:
  bbdd:
    image: mysql:5
    environment:
     - MYSQL_ROOT_PASSWORD=dadada
     - MYSQL_DATABASE=prova
     - MYSQL_USER=armando
     - MYSQL_PASSWORD=bronca
    restart: always
  web:
    image: eboraas/apache-php
    links:
      - bbdd:bbdd
    ports:
      - 8081:80
    volumes:
     - ./html/:/var/www/html/
    restart: always
```

Creamos el fichero dentro de la carpeta html/mysql.php

```
<?php
// Conectando, seleccionando la base de datos
$link = mysql_connect('bbdd', 'armando', 'bronca')
    or die('No se pudo conectar: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('prova') or die('No se pudo seleccionar la base de datos');

// Realizar una consulta MySQL
$query = 'SELECT * FROM pet';
$result = mysql_query($query) or die('Consulta fallida: ' . mysql_error());

// Imprimir los resultados en HTML
```

```
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";

// Liberar resultados
mysql_free_result($result);

// Cerrar la conexión
mysql_close($link);
?>
```

Estadísticas

Para mostrar estadísticas de consumo de recursos por contenedor:

```
watch -n 5 'docker stats --no-stream --format "table
{{.Name}}\t{{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}" | sort -r -k 3 -h'
```

Usar variables de entorno en tiempo de arranque

1. Crear script de arranque

```
mkdir /tmp/aux
cd /tmp/aux
vim entrypoint.sh
```

Con el siguiente contenido:

```
#!/bin/bash
echo "Content of VAR1 is: _"$VAR1"_"
```

2. Crear Dockerfile:

```
cd /tmp/aux
vim Dockerfile
```

Con el siguiente contenido:

```
FROM debian:testing

COPY entrypoint.sh /

RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

3. Probar

```
docker build . -t localhost/test && docker stop test && docker rm test &&
docker run --name test -ti -e VAR1=patata localhost/test
```

La última linea de la salida debería ser:

```
Content of VAR1 is: _patata_
```

4. Limpiar

```
docker stop test && docker rm test && docker rmi localhost/test
```

Timezone

Para ganar tiempo dejo algunos ejemplos de cómo configurar el timezone dependiendo de la imagen

Alpine

Hay que hacer 2 cosas:

1. Instalar el paquete “tzdata”

```
apk add tzdata
```

2. Configurar la variable de entorno “TZ”

Debian (p.ej.python:3-stretch)

Hay que hacer 1 cosa:

1. Crear un enlace simbólico. En este ejemplo usamos la variable “$TZ”, pero se puede “hardcodear” a “Europe/Madrid” por ejemplo

```
ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

Errores

docker: Error response from daemon: rpc error: code = 2 desc = "oci runtime error: could not synchronise with container process: no subsystem for mount".

Parece que es debido a la instalación de docker, seguí las instrucciones de instalación para debian y todo ok:

https://docs.docker.com/engine/installation/linux/debian/

1. Uninstall old versions

```
sudo apt-get remove docker docker-engine
```

2. Set up the repository

2.1. Install packages to allow apt to use a repository over HTTPS:

```
sudo apt-get install \
     apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common
```

2.2. Add Docker’s official GPG key:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
```

2.3. Use the following command to set up the stable repository. You always need the stable repository, even if you want to install edge builds as well.

```
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/debian \
   $(lsb_release -cs) \
   stable"
```

3. Install Docker CE

```
sudo apt-get update; sudo apt-get install docker-ce
```

Forzar detención contenedor estado "restart"

Escenario:

1. Hemos detenido un contenedor cuya política de reinicio era “unless-stopped”:

```
docker stop dns
dns
```

2. El contenedor está en estado “restart”:

```
docker ps | grep dns
5cb2e9ca4508        keducoop/dns:v2                          "./dockerdns --
domai…"   3 months ago        Restarting (1) About an hour ago
0.0.0.0:53->53/udp                          dns
```

Solución

1. Cambiar la política de reinicio del contenedor a “no”:

```
docker update --restart=no dns
dns
```

2. Ahora **SI** se podrá detener el contenedor sin que vuelva a intentar iniciarse:

```
docker stop dns
dns
```

Comprobar:

```
docker ps | grep dns
```

ERROR: Service 'logrotate' failed to build: invalid reference format

Escenario:

* Uso docker compose * Uso build dentro del docker compose * La imagen y la versión se pasan por parámetro

docker-compose.yml

```
version: '3.7'
services:

 logrotate:
  build:
   context: ./path/to/dir
   args:
    - IMAGE=${IMAGE}
    - VERSION=${VERSION}
...
```

Dockerfile (dentro de './path/to/dir')

```
ARG IMAGE

ARG VERSION

FROM $IMAGE:$VERSION
...
```

.env

```
IMAGE=
VERSION=latest
```

Solución:

Revisar que las variables “IMAGE” y “VERSION” sean coherentes. En este caso por error IMAGE estaba vacía

ERROR: yaml.parser.ParserError: while parsing a block mapping

Error completo:

```
ERROR: yaml.parser.ParserError: while parsing a block mapping
  in "./docker-compose.analytics-celery-v3.yml", line 1, column 1
expected <block end>, but found '<block mapping start>'
  in "./docker-compose.analytics-celery-v3.yml", line 14, column 2
```

Asegurarse que todos los servicios están alineados a la misma altura.

KO

```
version: '3.7'
services:

  logrotate-legacy:
    build:
      context: ./services/logrotate/
      dockerfile: Dockerfile
    container_name: logrotate-legacy
    environment:
      - LOGROTATE_LOGFILES=/var/log/plc/*.log
    volumes:
      - /var/log/plc/:/var/log/plc/:rw

 logrotate:
  container_name: ${LOGROTATE_NAME}
```

```
  image: ${LOGROTATE_IMAGE_LOCAL}/${LOGROTATE_NAME}:${LOGROTATE_VERSION}
  restart: ${LOGROTATE_RESTART}
  volumes:
   - ${LOGROTATE_VOLUME_PLC_HOST}:${LOGROTATE_VOLUME_PLC_CONTAINER}
  build:
   #context: ./services/logrotate
   context: ./services/logrotate-v2/
   args:
    - IMAGE=${LOGROTATE_IMAGE}
    - VERSION=${LOGROTATE_VERSION}
    - LOGROTATE_LOGFILES=${LOGROTATE_LOGROTATE_LOGFILES}

networks:
 network-logrotate:
  name: ${NETWORK_LOGROTATE}
```

OK

```
version: '3.7'
services:

 logrotate-legacy:
    build:
      context: ./services/logrotate/
      dockerfile: Dockerfile
    container_name: logrotate-legacy
    environment:
      - LOGROTATE_LOGFILES=/var/log/plc/*.log
    volumes:
      - /var/log/plc/:/var/log/plc/:rw

 logrotate:
  container_name: ${LOGROTATE_NAME}
  image: ${LOGROTATE_IMAGE_LOCAL}/${LOGROTATE_NAME}:${LOGROTATE_VERSION}
  restart: ${LOGROTATE_RESTART}
  volumes:
   - ${LOGROTATE_VOLUME_PLC_HOST}:${LOGROTATE_VOLUME_PLC_CONTAINER}
  build:
   #context: ./services/logrotate
   context: ./services/logrotate-v2/
   args:
    - IMAGE=${LOGROTATE_IMAGE}
    - VERSION=${LOGROTATE_VERSION}
    - LOGROTATE_LOGFILES=${LOGROTATE_LOGROTATE_LOGFILES}

networks:
 network-logrotate:
  name: ${NETWORK_LOGROTATE}
```

does not match any of the regexes: '^x-'

Error completo:

```
ERROR: The Compose file './docker-compose.rafa.yml' is invalid because:
networks.network-rafa-db value 'network-proxy' does not match any of the
regexes: '^x-'
```

Causa

Un bloque, en este caso dentro de networks, no está alineado correctamente

Solución

1. Identificar el bloque, en este caso “networks”

2. Revisar las entradas. Una de ellas tiene un espacio de más o de menos

Servidor tarda en apagarse esperando por contenedores docker

https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=989490#14

Docker build con variables

Si queremos tener versionado un docker, por ejemplo con versiones diferentes de node.js desde el build.
Instala por defecto la version 16.13.0

```
FROM debian

ARG NODE_VERSION=16.13.0

RUN wget
https://nodejs.org/download/release/v$NODE_VERSION/node-v$NODE_VERSION-linux
-x64.tar.xz
RUN mkdir -p /usr/local/lib/nodejs

RUN tar -xJvf node-v$NODE_VERSION-linux-x64.tar.xz -C /usr/local/lib/nodejs

CMD ["bash", "-l"]
```

Si quisiera crear una imagen con otra versión:

```
docker build --build-arg NODE_VERSION=18.15.0 -f Dockerfile -t node:18.15 .
```