

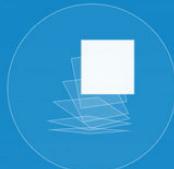
docker

Que es?



BUILD DEVELOPER WORKFLOWS

Docker allows you to compose your application from microservices, without worrying about inconsistencies between development and production environments, and without locking into any platform or language.



SHIP REGISTRY SERVICES

Docker lets you design the entire cycle of application development, testing and distribution, and manage it with a consistent user interface.



RUN MANAGEMENT

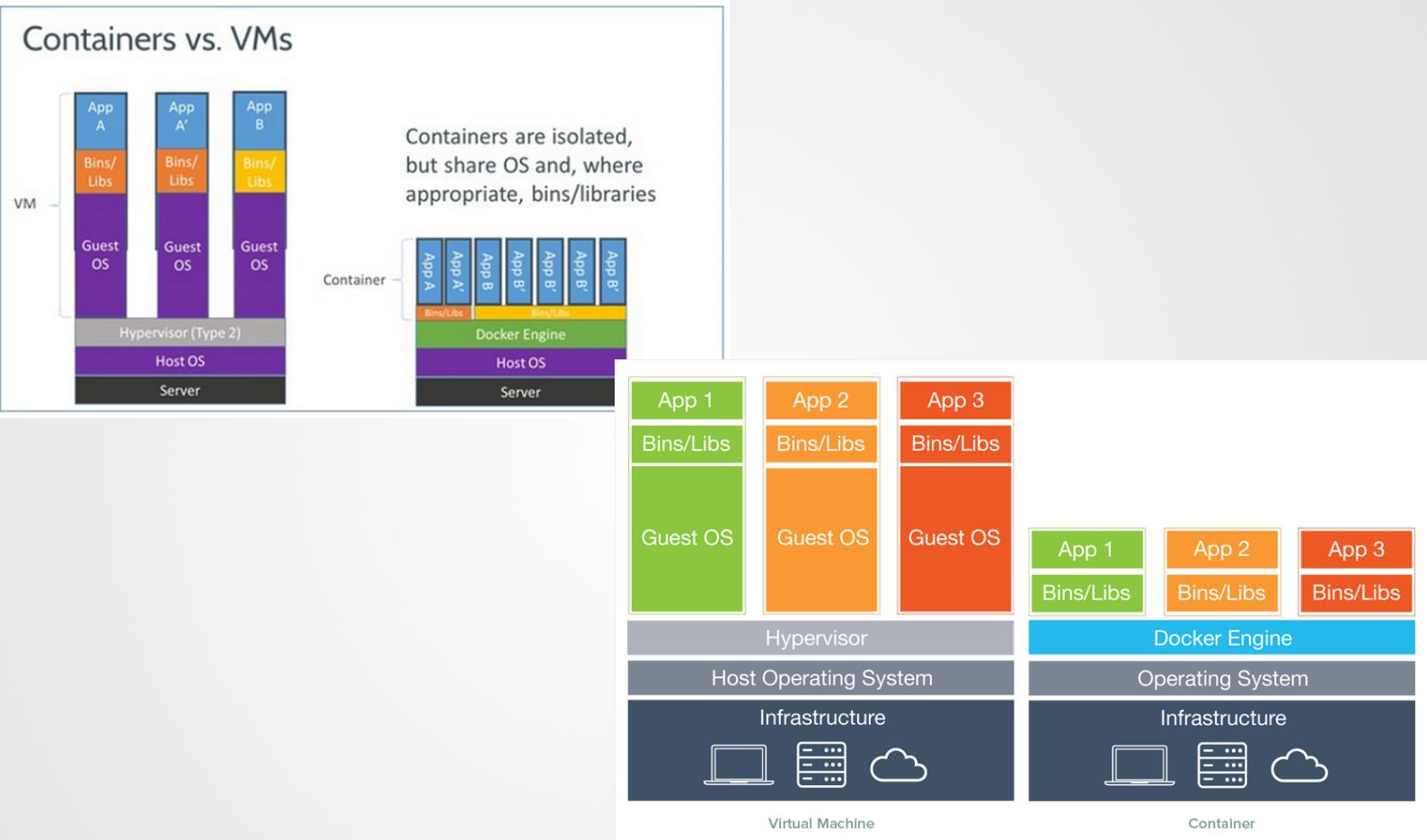
Docker offers you the ability to deploy scalable services, securely and reliably, on a wide variety of platforms.

DOCKER ENGINE

INFRASTRUCTURE

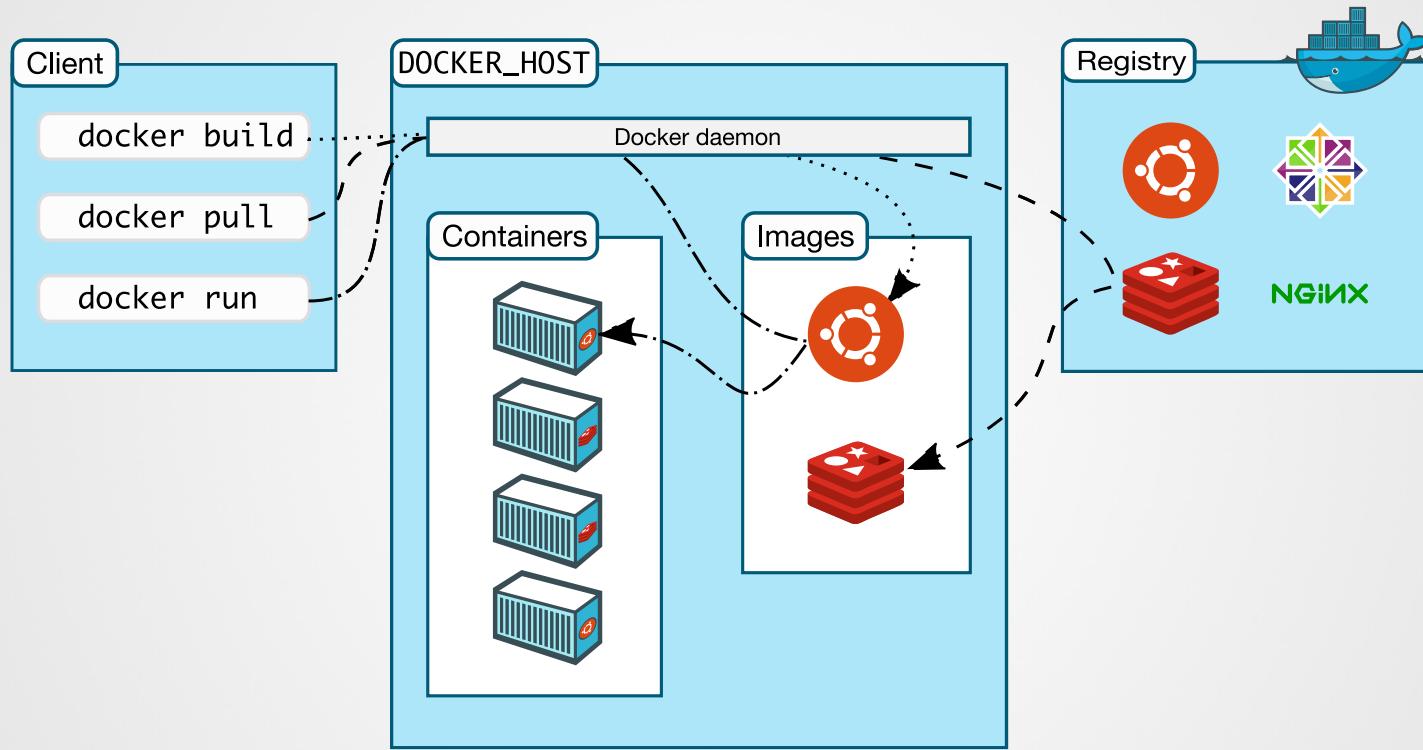
*Docker containers **wrap** up a piece of software in a complete filesystem that contains **everything it needs to run**: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always **run the same**, regardless of the environment it is running in.*

Containers vs. VMs



**Para que sirve?
Casos de Uso?**

Terminologia



Image

- Una imagen es un **template de solo lectura**, a partir del cual se **instancian los contenedores**
- Compuesta por una serie de Layers
- Analogo a una Clase en OOP
- Toda imagen tiene una imagen base, que es e la cual hereda

Container

- Un contenedor es a una maquina virtual lo que un hilo a un proceso
- Gracias a que comparten el kernel del sistema operativo subyacente, son muy rapidos
- En OOP seria un Objeto
- Son **transientes y descartables**
- En ciertos casos pueden convertirse en imagenes
(Not the docker way tho)

Tag

- Una etiqueta permite que existan distintas versiones de la misma imagen en un repositorio
- Por ejemplo:
 - python:3.5, python:2.7
 - ubuntu:trusty, ubuntu:xenial
 - alpine:3.4
- Nos brindan mayor **flexibilidad y organizacion**

Repository

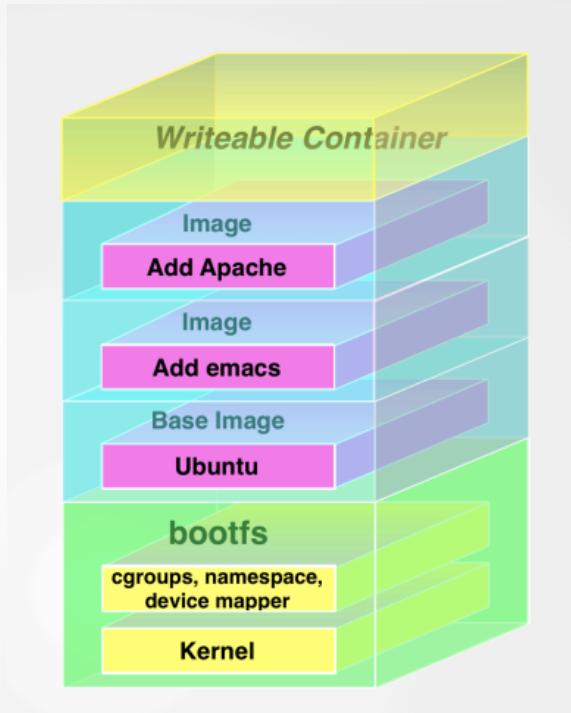
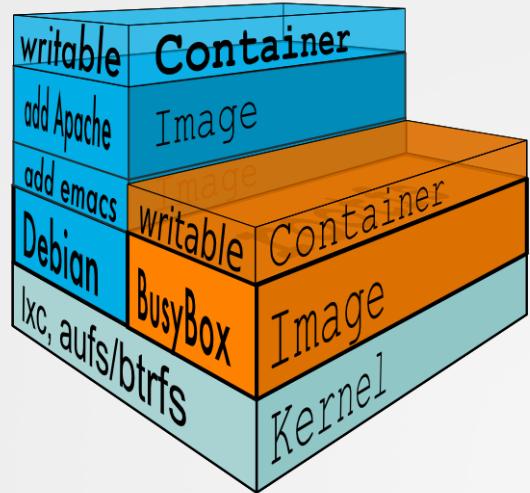
- Un Repositorio (generalmente DockerHub) es un lugar de donde podemos **subir y descargar IMAGES**
- Existen repos oficiales:
 - ubuntu, centos, python, redis, mongodb, mysql, mariadb, jarvis
- Generalmente los repos oficiales sirven de base images

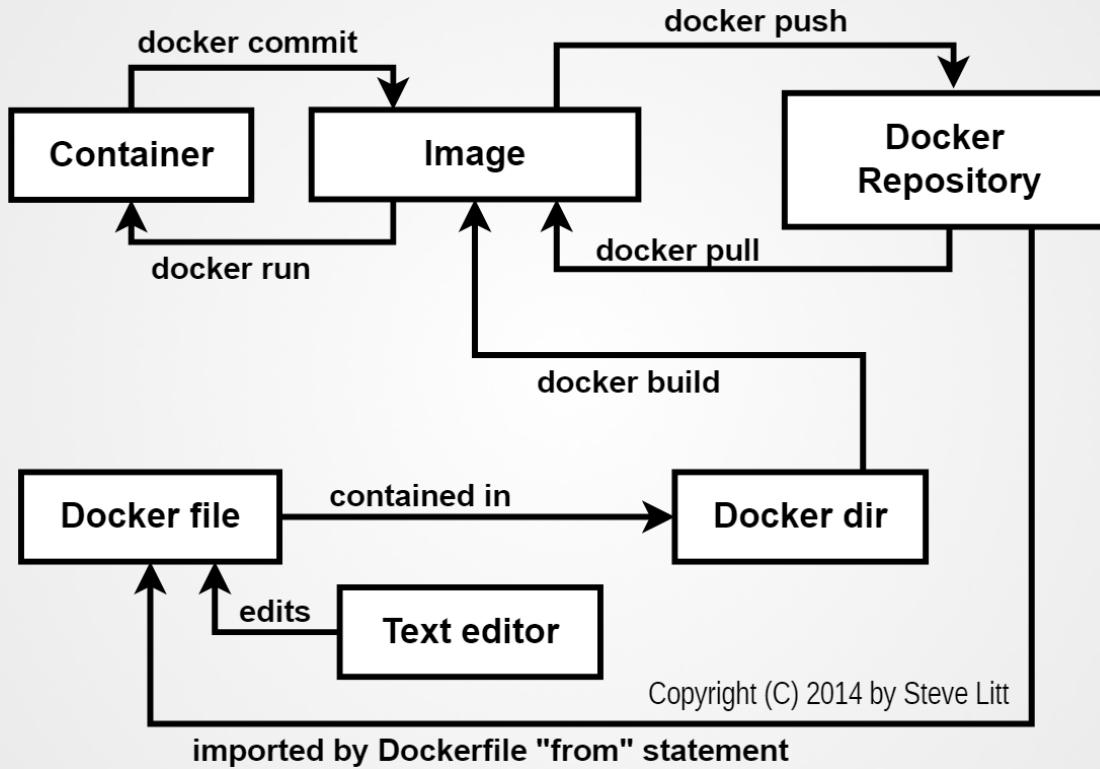
Volume

- Un volumen es un directorio especial dentro de un contenedor (o designado mediante un Dockerfile dentro de una imagen) que ignora las limitaciones del Union File System.
- Los **datos escritos son persistentes**, incluso si el contenedor se elimina.
- Pueden ser reutilizados entre contenedores.
- Se suele usar un contenedor con varios volúmenes como base para el resto de los containers

Image Layer

- Una imagen esta compuesta de muchas capas
- Cada capa contiene los cambios entre su padre y ella
- Esto permite mejorar el **cacheado y distribucion** de imagenes mediante el sistema de base images
- Es bueno saber que comandos generan una nueva imagen y agruparlos de la mejor manera posible





docker

MINGW64:/c/Users/ramar



docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at <https://docs.docker.com>

Start interactive shell

ramar@PLATINUMICE MINGW64 ~
\$

run

Crea un container a partir de una image y lo inicia.

Forma: `docker run [opciones] imagen [comando]`

- **-i:** sesion interactiva
- **-t:** alocacion de una pseudo tty
 - en general yo uso **-it** siempre, de esta manera me "meto" dentro de la terminal del contenedor
- **--rm:** una vez que el container se detenga sera eliminado. util para hacer pruebas y no perder control del HDD

run

- **-v:** Define un volumen
 - **-v \$DIR:\$GUESTDIR** esto nos permite compartir archivos
 - Si \$DIR es un nombre, defino un named volume
 - Si no, defino una particion de archivos
- **-e:** define una variable de entorno
 - **-e VAR=valor**
- **-p:** expone o mapea puertos
 - **-p 22:** abre el puerto 22 (sshd)
 - **-p 10080:80** Esto hace el puerto 80 del container accesible a traves del 10080

run

Ejemplo:

```
docker run -it \
--rm \
-e NAME="ramiro" \
-v ./code \
-p 10022:22 \
gibdfrcu/base:alpine \
/bin/bash echo \
hello world
```

build

Crea una Imagen a partir de un archivo Dockerfile

FORMA: `docker build [opciones] ruta`

- Generalmente se usa "." como ruta, todo lo que esta en esta carpeta pasa a formar parte del **contexto**
- **--file CADENA:** nombre del archivo para construir. por defecto es Dockerfile
- **--tag VALOR:** permite etiquetar una imagen para luego subirla
- **--pull:** intenta siempre usar la version mas actualizada
- Ejemplo
 - `docker build --file archivoDocker.txt .`

pull

Descarga una imagen del dockerhub

FORMA: docker pull [opciones] nombre[:tag]

- **--all-tags:** descarga todas las etiquetas disponibles para la imagen. **CUIDADO!!! NI SUEÑES EN USARLO EN UTN!!!!!!**
- ejemplo:
 - docker pull gibdfrcu/base:ubuntu

push

Contracara de pull, este nos permite subir una de nuestras imagenes a docker hub (previo uso de docker login)

FORMA: `docker push nombre[:tag]`

- En general, nombre siempre sera de la forma *nuestro_user/nombre_imagen*. (puedo equivocarme, no juegue mucho con push)
- Tag nos permite diferenciar versiones de soft, por ejemplo: ubuntu:latest, ubuntu:xenial, ubuntu:trusty
- Distintas tag pueden hacer referencia a la misma imagen

exec

Nos permite ejecutar un comando determinado en un contenedor instanciado. Muy util en conjuncion con docker-compose

FORMA: `docker exec [opciones] container comando`

- **-i:** Analogo al -i de run
- **-t:** Analogo al -t de run
- container puede ser un id de contenedor o el nombre que se le dio al instanciar el mismo

ps

Lista los contenedores.

FORMA: `docker ps [opciones]`

- **-a:** muestra todos los contenedores (no solo los que estan corriendo, util para ver el uso y estados)
- **-q:** quiet mode, devuelve solo los ids de los contenedores. SUPER UTIL para scripts de administracion

ps

Ejemplo:

docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4c28d610e7df	d13ce3e20c68	"/bin/sh -c 'apk add "	About an hour ago	Exited (2) About an hour ago		loving_liskov
6c9267c01c30	783bebb7bc93	"/bin/sh -c 'pip inst"	2 hours ago	Exited (1) 2 hours ago		ecstatic_montalcini
6e0b10fb391c	1c5c21ae3b88	"/bin/sh -c 'echo [su]"	3 hours ago	Exited (1) 3 hours ago		high_poincare
30eaaf33bbec	a718dc5aead4	"/bin/sh -c 'addgroup"	3 hours ago	Exited (1) 3 hours ago		jovial_banach
7f3345a08d86	a718dc5aead4	"/bin/sh -c 'groupadd"	3 hours ago	Exited (1) 3 hours ago		modest_wing
6a3c44403908	gibdfrcu/base	"/usr/sbin/sshd -D"	21 hours ago	Exited (0) 3 hours ago		gibddata
3027b3260a48	gibdfrcu/kafka:latest	"start-kafka.sh"	21 hours ago	Exited (0) 3 hours ago		gibdkafk
aae19c88fb2c	docker_gibd_zook	"/bin/sh -c '/usr/bin/"	21 hours ago	Exited (137) 3 hours ago		gibdzook

images

Análogo a ps, pero en vez de para contenedores para imágenes.

FORMA: docker images [opciones]

Ejemplo:

docker images -a

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	36672c611cac	About an hour ago	339.4 MB
<none>	<none>	6109c5fc9a65	About an hour ago	339.4 MB
<none>	<none>	102fa1c64b25	About an hour ago	294.6 MB
<none>	<none>	e1bf6d3a2081	About an hour ago	4.804 MB
<none>	<none>	d13ce3e20c68	About an hour ago	294.6 MB
<none>	<none>	d6689aa1a565	About an hour ago	4.803 MB
<none>	<none>	38bb4c92f8d8	About an hour ago	4.803 MB
<none>	<none>	ba34fb1aacc6	About an hour ago	4.804 MB
<none>	<none>	e737cfb198e3	About an hour ago	4.803 MB
<none>	<none>	783bebb7bc93	2 hours ago	400.1 MB
<none>	<none>	7fecfc4b32f3	2 hours ago	376.9 MB
<none>	<none>	19977687e9e9	2 hours ago	376.9 MB
<none>	<none>	784fce48e68f	2 hours ago	376.9 MB
gibdfrcu/base	alpine-python3	f50a365f42af	2 hours ago	376.9 MB
<none>	<none>	1c5c21ae3b88	3 hours ago	341 MB
<none>	<none>	d2b646c1aafc	3 hours ago	341 MB
<none>	<none>	5bd152bfd9b5	3 hours ago	341 MB
<none>	<none>	8016a3ad3564	3 hours ago	341 MB
<none>	<none>	486f3a44e458	3 hours ago	341 MB

rm

Elimina un contenedor DETENIDO (si no es asi, hay que usar docker stop).

FORMA: `docker rm [opciones] contenedor [contenedores]`

- **-f:** remueve de manera forzada (usando SIGKILL, no recomendado)
- **-v:** remueve tambien los volumenes asociados al contenedor
- contenedor/es puede ser el nombre o el id, y si son varios, separados por un espacio

rmi

Utilizado para eliminar Imagenes.

FORMA: `docker rmi [opciones] imagen [imagenes]`

- **-f:** elimina de manera forzada la imagen. Hay veces que por dependencia entre imagenes o imagenes y contenedores ni siquiera con -f podremos eliminar
- Imagen/es puede ser el nombre o el id, y si son varias separadas por un espacio

stop

Utilizado para detener contenedores en ejecucion.

FORMA: `docker stop [opciones] contenedor [contenedores]`

- **-t entero:** Espera *entero* segundos antes de detener el contenedor
- contenedor/es puede ser el nombre o el id, y si son varios, separados por un espacio

Dockerfile

- Definicion de una imagen
- Cada instruccion en un dockerfile crea una nueva capa
- Si se modifican instrucciones anteriores, todas las siguientes se ejecutan de nuevo (cache invalidation)
- Se busca minimizar el tamaño de la imagen resultante
- Se busca generalizar mediante variables de entorno y tags

```
1 FROM gibdfrcu/base:alpine
2
3 MAINTAINER Ramiro Rivera <ramarivera@gmail.com>
4
5 ENV STORM_VERSION="0.9.4" \
6     SUPERVISOR_VERSION="3.3.0" \
7     SUPERVISORD_CONF="/etc/supervisord.conf" \
8     SUPERVISORD_D_CONF="/etc/supervisord.d"
9
10 ENV STORM_HOME="/opt/apache-storm-${STORM_VERSION}"
11
12
13 RUN wget -q -O - http://mirrors.sonic.net/apache/storm/apache-storm-${STORM_VERSION}/apache-storm-${STORM_VERSION}.tar.gz | tar -xzf - -C /opt
14 RUN pip install supervisor==$SUPERVISOR_VERSION
15
16
17 RUN mkdir $SUPERVISORD_D_CONF && \
18     \
19     addgroup storm && \
20     adduser -D -h /home/storm -G storm -s /bin/bash  storm && \
21     echo 'storm:storm' | chpasswd && \
22     chown -R storm:storm $STORM_HOME && \
23     mkdir /var/log/storm && \
24     chown -R storm:storm /var/log/storm && \
25     \
26     ln -s $STORM_HOME/bin/storm /usr/bin/storm
27
28
29 COPY storm.yaml $STORM_HOME/conf/
30 COPY cluster.xml $STORM_HOME/logback/
31 COPY start-supervisor.sh /usr/bin/
32 COPY config-supervisord.sh /usr/bin/
33
34 RUN cat $STORM_HOME/conf/storm.yaml
35
36 RUN chmod +x /usr/bin/start-supervisor.sh /usr/bin/config-supervisord.sh  && \
37     echo [supervisord] | tee -a $SUPERVISORD_CONF && \
38     echo nodaemon=true | tee -a $SUPERVISORD_CONF && \
39     echo [include] | tee -a $SUPERVISORD_CONF && \
40     echo files=/etc/supervisord.d/*.conf | tee -a $SUPERVISORD_CONF
```

FROM

- Define que imagen se toma como base para construir la imagen actual.
- Puede especificarse con o sin tag (mejor con, ya que nos aseguramos de siempre tener la misma base)

Ejemplo

FROM gibdrcu/base:alpine

MANTAINER

- Hasta donde se, es solo una metadata como para poder contactar o acreditar el uso de una imagen.
- Generalmente es de la forma *nombre <mail>*

Ejemplo

MANTAINER Ramiro Rivera <riverar@frcu.utn.edu.ar>

ENV

- Define variables de entorno que persistiran no solo durante el proceso de construccion de la imagen/contenedor.
- Lo recomendable es usar NOMBRE="VALOR" para prevenir problemas con valores, espacios, etc

Ejemplo

```
ENV JAVA_HOME="/opt/java" ALGO="/root"
```

COPY/ADD

- Permite hacer accesibles a la imagen/container archivos del equipo.
- La diferencia es que ADD es capaz de descargar archivos / extraer comprimidos (se recomienda en general usar COPY)

Ejemplo

COPY ["./arch1", "./arch2", "/home/destino/"]

COPY ./archivo1 /home/destino/destinoarchivo1

RUN

- El comando principal del Dockerfile, con el cual ejecutamos instrucciones en el momento de construccion de la imagen

Ejemplo

RUN apt-get update && apt-get install nano

RUN apk add --update wget

USER/WORKDIR

- USER cambia el usuario logueado dentro de la imagen (similar a su) y WORKDIR modifica el cwd (similar a cd).
- En general no los uso mucho, pero la diferencia entre WORKDIR xx y RUN cd xx es que el "cd xx" solo dura en el contexto de ese RUN

CMD

- CMD define el comando que ejecutara el contenedor al instanciarse.
- Podriamos por ej definir que el comando sea iniciar el httpd en modo no daemon.
- Se recomienda usar la "exec form" (Ver ejemplo). Caso contrario, el comando se ejecuta como parametro de "/bin/sh -c"

Ejemplo

CMD ["ejecutable", "param1", "param2"]

CMD ["ping", "-c", "google.com"]

ENTRYPOINT

- Nos permite definir un contenedor como ejecutable por asi decirlo.
- Al usarlo en exec form, los parametros pasados con docker run se convierten en parametros del ejecutable en entrypoint. Si no se usa exec form, los parametros **SE IGNORAN**

Ejemplo

```
ENTRYPOINT ["ping"]  
ENTRYPOINT ping google.com
```

EXPOSE

- Nos permite definir que puertos seran alcanzados desde otros contenedores o desde el equipo host. Es como que fuera abrir un puerto en un router o antivirus.
- Expose **NO** mapea puertos HOST a puertos CONTAINER

Ejemplo

EXPOSE 22 8080 95

docker-compose

build

Construye las imagenes para poder iniciar luego una composicion

FORMA: docker-compose build [opciones]

- **--pull:** Siempre descarga la imagen mas nueva disponible
- **--no-cache:** No utiliza ningun cache

up

Inicia una composicion. Debe existir un archivo de composicion en el directorio actual.

FORMA: `docker-compose up [opciones]`

- **-d:** modo background
- **--force-recreate:** obliga a reconstruir imagenes por mas que no se detecten cambios de configuracion

Para finalizar una composicion se utiliza CTRL+C

docker-compose.yml

- YAML ain't markup language
- Define interacciones y configuraciones de los contenedores

```
1 version: "2"
2 volumes:
3   ui_home:
4     driver: local
5 services:
6   gibd_zook:
7     container_name: zookeeper
8     image: gibdfrcu/zookeeper:latest
9     ports:
10    - "2181:2181"
11   gibd_kafk:
12     container_name: kafka
13     image: gibdfrcu/kafka:alpine
14     ports:
15    - "9092:9092"
16   environment:
17     KAFKA_ADVERTISED_HOST_NAME: 192.168.99.100
18     KAFKA_CREATE_TOPICS: "testo:1:1"
19     KAFKA_ZOOKEEPER_CONNECT: gibd_zook:2181
20   volumes:
21    - /var/run/docker.sock:/var/run/docker.sock
22   links:
23    - gibd_zook:zk
```

version

- Existen actualmente dos versiones. Lo recomendable es usar la mas nueva, declarandola con `version: "2"`

services

- Seccion que define para cada contenedor la configuracion particular.
- Cada elemento dentro de services debe tener una identificacion unica y valida.

services - container-name

- Nombre que se le asignara al contenedor y por el cual podra ser referenciado dentro y fuera del compose

services - image

- Imagen en la cual se basara para instanciar el contenedor en cuestion

services - build

- Define el directorio relativo o absoluto donde se encuentra un Dockerfile con el cual construir el contenedor

services - ports

- Permite definir tanto el mapeo HOST:CONTAINER como tambien en que puertos escucha el container, usando rangos/interfaces

```
gibd_nimb:  
  container_name: storm-nimbus  
  image: gibdfrcu/storm-nimbus  
  ports:  
    - "3773:3773"  
    - "3772:3772"  
    - "6627:6627"
```

```
gibd_nimb:  
  container_name: storm-nimbus  
  image: gibdfrcu/storm-nimbus  
  ports:  
    - "3772-3773:3772-3773"  
    - "6627:6627"
```

services - environment

- Analogo a ENV del Dockerfile, define variables de entorno

```
environment:  
    KAFKA_ADVERTISED_HOST_NAME: 192.168.99.100  
    KAFKA_CREATE_TOPICS: "testo:1:1"  
    KAFKA_ZOOKEEPER_CONNECT: gibd_zook:2181
```

services - links

- Si bien no es algo especialmente util en la version 2, lo practico es que deja explicita la relacion de dependencia entre contenedores y nos permite la asignacion de alias para comunicacion local

```
gibd_supv:  
  container_name: storm-supervisor  
  image: gibdfrcu/storm-supervisor:alpine  
  ports:  
    - "8000:8000"  
  links:  
    - gibd_nimb:nimbus  
    - gibd_zook:zk
```

services - volumes

- Define los volumenes que utiliza el contenedor.
Pueden ser named volumes o volumenes para compartir archivos entre el host y container

```
gibd_stui:  
  container_name: storm-ui  
  image: gibdfrcu/storm-ui:alpine  
  volumes:  
    - ui_home:/opt/apache-storm-0.9.4  
    - ./code:/code
```

services - volumes_from

- Permite montar volúmenes de otros contenedores en **EL MISMO PUNTO DE MONTAJE**. Util para compartir info y hacer backups, pero debe ser usado con cuidado

```
gibd_data:  
    container_name: data  
    command: y >> /dev/null  
    image: gibdfrcu/base:alpine  
    volumes_from:  
        - gibd_stui  
        - gibd_zook  
        - gibd_nimb  
        - gibd_supv  
        - gibd_kafk
```

volumes

- Permite la definicion de named volumes.

```
volumes:  
  ui_home:  
    driver: local
```

docker-machine

ip

- \$(docker-machine ip): al ejecutar en un shell nos devuelve la ip local del docker host
 - Al decir ip local, me refiero a la ip por la que podemos acceder a los contenedores por ej a traves de interfaces web o ssh.
 - Es la ip de la maquina virtual linux en la Host Only Network

Advanced

Automated-Build

- Un build automatizado nos permite definir un repositorio de github/bitbucket, en el cual se encuentra un Dockerfile junto con su contexto.
- A partir de ese dockerfile, podemos indicar que cada vez que se realize un push se reconstruya la imagen en dockerhub, con que etiqueta y que rama del repositorio.
- Tambien podemos definir triggers (ej, si se actualiza la imagen base, actualiza todas las que dependan de ella)

Bash variables

- Definicion:
 - VAR = expr
 - VAR = \$(command)
 - generalmente definidas con mayusculas
- Uso:
 - \$VAR
- Interpolacion:
 - Reemplaza a la concatenacion de strings en la mayoria de los casos
 - "una cadena con una \${VAR} interpolada"

Bash Basics

- **;** : permite usar varios comandos en una instruccion, siempre ejecuta el segundo comando
- **&&** : Ejecuta el segundo comando solo si el primero finalizo correctamente (short circuit)
- **** : Continuacion de linea, util para la organizacion visual en un Dockerfile
- **grep**: filtrado de streams, util para realizar busquedas
- **sed**: reemplazo/busqueda en streams
- **awk**: Herramienta para reportes y manipulacion de texto
- **echo**: analogo al writeline o print de otros lenguajes
- **|** : operador pipe
- **if [] fi** : bloque if