Cheatsheet Docker

Completa - Autor: Sergi García Barea

Docker run

```
docker run -d --name=cont1 ubuntu tail -f /dev/null
```

- Crea un contenedor con la imagen “ubuntu” (al no especificar, toma versión “latest”), le establece un nombre “cont1” y lanza el comando “tail -f /dev/null”, que es un comando cuya ejecución no acaba nunca.

```
docker run -d -p 1200:80 nginx
```

- Crea un contenedor con la versión “latest” de la imagen “nginx” y lo lanza en “background”, exponiendo el puerto 80 del contenedor en el puerto 1200 de la máquina anfitrión.

```
docker run -it -e MENSAJE=HOLA ubuntu:14.04 bash
```

- Crea un contenedor con la imagen “ubuntu”, versión “14.04” y establece la variable de entorno “MENSAJE”.

Docker ps

```
docker ps
```

- Muestra información de los contenedores en ejecución.

```
docker ps -a
```

- Muestra información de todos los contenedores, tanto parados como en ejecución.

Docker start/stop/restart

```
docker start micontenedor
```

- Arranca el contenedor con nombre “mi contenedor”.

```
docker start -ai micontenedor
```

- Arranca el contenedor con nombre “mi contenedor”, enlazando el comando ejecutado al arranque a la entrada y salida estándar de la terminal del anfitrión.

Docker Exec

```
docker exec -it idcont bash
```

- Lanza en el contenedor “idcont” (que debe estar arrancado) el comando “bash”, enlazando la ejecución de forma interactiva a la entrada y salida estándar del anfitrión en la terminal.

```
docker exec -it -e FICHERO=prueba cont bash
```

- Lanza en el contenedor “cont” (que debe estar arrancado) el comando “bash”, estableciendo la variable de entorno “FICHERO” y enlazando la ejecución de forma interactiva a la entrada y salida estándar del anfitrión.

Cheatsheet Docker

Completa - Autor: Sergi García Barea

```
docker exec -d cont touch /tmp/prueba
```

- Lanza en el contenedor “cont” (que debe estar arrancado) el comando “touch /tmp/prueba”. Este comando se ejecuta en segundo plano, generando el fichero “/tmp/prueba”.

Docker attach

```
docker attach idcontainer
```

- Enlaza nuestra terminal la entrada/salida de nuestra al proceso en segundo plano del contenedor “idcontainer”.

Docker logs

```
docker logs -n 10 idcontainer
```

- Muestra las 10 últimas líneas de la salida estandar producida por el proceso en ejecución en el contendor.

Docker cp

```
docker cp idcontainer:/tmp/prueba ./
```

- Copia el fichero “/tmp/prueba” del contenedor “idcontainer” al directorio actual del anfitrión.

```
docker cp ./miFichero idcontainer:/tmp
```

- Copia el fichero “miFichero” del directorio actual del anfitrión a la carpeta “/tmp” del contenedor.

Gestión de imágenes

```
docker images
```

- Información de imágenes locales disponibles.

```
docker search ubuntu
```

- Busca la imagen “ubuntu” en el repositorio remoto (por defecto Docker Hub).

```
docker pull alpine
```

- Descarga localmente imagen “alpine”.

```
docker history alpine
```

- Muestra la historia de creación de la imagen “alpine”.

```
docker rmi ubuntu:14.04
```

- Elimina localmente la imagen “ubuntu” con tag “14.04”.

```
docker rmi $(docker images -q)
```

- Borra toda imagen local que no esté siendo usada por un contenedor.

Cheatsheet Docker

Completa - Autor: Sergi García Barea

```
docker rm IDCONTENEDOR
```

- Borra un contenedor con IDCONTENEDOR.

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

- Para todos los contenedores del sistema.

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

- Borra todos los contenedores parados del sistema.

```
docker system prune -a
```

- Borra todas las imágenes y contenedores parados del sistema.

Creación de imágenes a partir de contenedores

```
docker commit -m “comentario” IDCONTENEDOR usuario/imagen:version
```

- Hace commit de un contenedor existente a una imagen local.

```
docker save -o copiaSeguridad.tar imagenA
```

- Guarda una copia de seguridad de una imagen en fichero “.tar”.

```
docker load -i copiaSeguridad.tar
```

- Restaura una copia de seguridad de una imagen en fichero “.tar”.

Docker Hub

```
docker login
```

- Permite introducir credenciales del registro (por defecto “Docker Hub”).

```
docker push usuario/imagen:version
```

- Permite subir al repositorio una imagen mediante “push”.

Ejemplo de Dockerfile

```
FROM alpine
LABEL maintainer="email@gmail.com"
#Actualizamos e instalamos paquetes con APK para Alpine
RUN apk update && apk add apache2 php php-apache2 openrc tar
#Copiamos script para lanzar Apache 2
ADD ./start.sh /start.sh
#Descargamos un ejemplo de <?php phpinfo(); ?> por enseñar como bajar algo de Internet
#Podría haber sido simplemente
#RUN echo "<?php phpinfo(); ?>" >  /var/www/localhost/htdocs/index.php
ADD https://gist.githubusercontent.com/SyntaxC4/5648247/raw/94277156638f9c309f2e36e19bff378ba7364907/info.php
/var/www/localhost/htdocs/index.php
# Si quisiéramos algo como Wordpress haríamos
```

Cheatsheet Docker

Completa - Autor: Sergi García Barea

```
#ADD http://wordpress.org/latest.tar.gz /var/www/localhost/htdocs/wordpress.tar.gz
#RUN tar xvzf /var/www/localhost/htdocs/wordpress.tar.gz && rm -rf /var/www/localhost/htdocs/wordpress.tar.gz
# Usamos usuario y grupo www-data. El grupo lo crea Apache, pero si quisiéramos crear grupo
# Grupo www-data RUN set -x && addgroup -g 82 -S www-data
RUN adduser -u 82 -D -S -G www-data www-data
# Hacemos todos los ficheros de /var/www propiedad de www-data
# Y damos permisos a esos ficheros y a start.sh
RUN chown -R www-data:www-data /var/www/ && chmod -R 775 /var/www/ && chmod 755 /start.sh
#Indicamos puerto a exponer (para otros contenedores) 80
EXPOSE 80
#Comando lanzado por defecto al instalar el contendor
CMD /start.sh
```

- Ejemplo de fichero “Dockerfile”.

Gestión de redes

```
docker network create redtest
```

- Creamos la red “redtest”

```
docker network ls
```

- Nos permite ver el listado de redes existentes.

```
docker network rm redtest
```

- Borramos la red “redtest”.

```
docker run -it --network redtest ubuntu /bin/bash
```

- Conectamos el contenedor que creamos a la red “redtest”.

```
docker network connect IDRED IDCONTENEDOR
```

- Conectamos un contenedor a una red.

```
docker network disconnect IDRED IDCONTENEDOR
```

- Desconectamos un contenedor de una red

Volúmenes

```
docker run -d -it --name appcontainer   -v /home/sergi/target:/app nginx:latest
```

- Creamos un contenedor y asignamos un volumen con “binding mount”.

```
docker run -d -it --name appcontainer   -v micontenedor:/app nginx:latest
```

- Creamos un contenedor y asignamos un volumen Docker llamado “micontenedor”.

```
docker volume create/ls/rm mivolumen
```

- Permite crear, listar o eliminar volúmenes Docker.

```
docker run -d -it --tmpfs /app nginx
```

Completa - Autor: Sergi García Barea

- Permite crear un contenedor y asociar un volumen “tmpfs”.

```
docker volume rm $(docker volume ls -qf dangling=true)
```

- Borra todos los volúmenes huérfanos (sin contenedor asociado).

```
docker run --rm --volumes-from contenedor1 -v /home/sergi/backup:/backup ubuntu bash -c "cd /datos && tar cvf /backup/copiaseguridad.tar ."
```

- Permite realizar una copia de seguridad de un volumen asociado a “contenedor1” y que se monta en “/datos”. Dicha copia finalmente acabará en “/home/sergi/backup” de la máquina anfitrión.

```
docker volume rm $(docker volume ls -q)
```

- Permite eliminar todos los lúmenes de tu máquina.

Ejemplo básico de fichero “docker-compose.yml”

```
version: "3.9"
services:
  db:
        image: mariadb:10.11.2
        volumes:
        - db_data:/var/lib/mysql
        environment:
            MARIADB_ROOT_PASSWORD: somewordpress
            MARIADB_DATABASE: wordpress
            MARIADB_USER: wordpress
            MARIADB_PASSWORD: wordpress
  wordpress:
        image: wordpress:latest
        ports:
        - "8000:80"
        environment:
            WORDPRESS_DB_HOST: db:3306
            WORDPRESS_DB_USER: wordpress
            WORDPRESS_DB_PASSWORD: wordpress
            WORDPRESS_DB_NAME: wordpress
volumes:
  db_data:
```

Principales comandos de “Docker Compose”

```
docker compose up -d
```

- Inicia el sistema definido en “***docker-compose.yml***” en segundo plano. Genera y descarga imágenes requeridas.

```
docker compose down
```

- Detiene y elimina los contenedores según la configuración de “***docker-compose.yml***”.

```
docker compose build/pull
```

- Construye/descarga las imágenes de contenedores según la configuración de “***docker-compose.yml***”.

Cheatsheet Docker

Completa - Autor: Sergi García Barea

```
docker compose ps
```

- Muestra información de los contenedores según la configuración de "***docker-compose.yml***".

```
docker compose up -d --scale web=3
```

- Similar a "***docker compose up -d***" solo que además, el servicio definido como "web" en el fichero "***docker-compose.yml***" lo escala creando 3 copias y realizando balanceo automático si se realiza una petición al host llamado como el servicio "***web***".

Principales comandos de "Kubernetes"

```
kubectl apply -f "fichero.yaml"
```

- Aplica en Kubernetes la configuración especificada en "fichero.yaml".

```
kubectl create deployment midespliegue --image=sergarb1/flaskparakubernetes --port=5000
```

- Crea un despliegue basado en una imagen dada y en el puerto 5000.

```
kubectl expose deployment midespliegue --type=LoadBalancer --name=midespliegue-http
```

- Crea un servicio de tipo "LoadBalancer" exponeniendo "midespliegue".

```
kubectl get pods; kubectl get services; kubectl get deployments
```

- Muestra información de pods, servicios o despliegues.

```
kubectl scale deployment midespliegue --replicas=3
```

- Escala horizontalmente un despliegue a 3 réplicas.

```
kubectl autoscale deployment midespliegue --min=5 --max=10
```

- Configura autoescalado horizontal, aceptando entre 5 y 10 réplicas.

```
kubectl delete pod/deployment/service/autoscale nombre
```

- Permite eliminar un pod, despliegue, servicio o autoescalado.

Principales comandos de "MniKube"

```
minikube start
```

- Inicia la máquina virtual que contiene MiniKube y pone el cluster Kubernetes en marcha

```
minikube service miservicio
```

- Nos permite acceder a un servicio dentro de MiniKube desde la máquina local.

```
minikube tunnel
```

- Mientras esté en ejecución, expone un servicio dentro de MiniKube a la máquina local

Cheatsheet Docker

Completa - Autor: Sergi García Barea

Ejemplo de fichero YAML despliegue/servicio/persistencia con Kubernetes

```
#Definimos la información del servicio
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
        app: wordpress
spec:
  ports:
        #El servicio se expone en el puerto 80
        - port: 80
  selector:
        app: wordpress
        tier: frontend
  #Aplicamos balanceo de carga para facilitar su escalado horizontal
  type: LoadBalancer
---
#Definimos un volumen persistente
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
        app: wordpress
spec:
  #Indica que solo puede ser montado para lectura/escritura por un nodo. Para el resto lectura.
  #En este caso, se usa para modificar un fichero de configuración.
  accessModes:
        - ReadWriteOnce
  resources:
        requests:
        storage: 20Gi
---
#definimos el despliegue
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
        app: wordpress
spec:
  selector:
        matchLabels:
        app: wordpress
        tier: frontend
  strategy:
        type: Recreate
  template:
        metadata:
        labels:
        app: wordpress
        tier: frontend
        spec:
```

Cheatsheet Docker

Completa - Autor: Sergi García Barea

```
        #Imagen
        containers:
        - image: wordpress:4.8-apache
        name: wordpress
        #Indicamos variables de entorno
        env:
        - name: WORDPRESS_DB_HOST
        value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
        value: CEFIREdocker
        ports:
        - containerPort: 80
        name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
        mountPath: /var/www/html
        volumes:
        - name: wordpress-persistent-storage
        persistentVolumeClaim:
        claimName: wp-pv-claim
```