

### Docker Image

<code>docker images</code>	See all available images
<code>docker build -t &lt;name&gt;</code>	Create an image with a pretty name (you must define the <code>Dockerfile</code> in the folder)
<code>docker tag &lt;name&gt; username/repository:tag</code>	This tags an image ready to be sent to a repository
<code>docker push username/repository:tag</code>	Push the image to the remote repository
<code>docker search &lt;keyword&gt;</code>	Search for public repositories

Docker Images are the base for containers and are similar to `.iso` files. They can be for example the image of your app and contain everything needed to run the application.

These images can be local or in repositories (and marked with a tag)

To create images, you must create a `Dockerfile` with some docker commands to specify how that image will be created, for example to setup the environment and a `BaseImage`.

### Services

### Services (cont)

To run it you must first start: `docker swarm init`  
 Then run it giving a name:  
`docker stack deploy -c docker -co mpo se.yml <app_name>`  
 To see the details of containers running in your service, run:  
`docker stack ps <app_name>`  
 Now each time you request your app (via browser, for example), the load-balancer will help you distributing the requests to each replica.  
 To put it down, `docker stack rm <app_name>`  
 This puts down the app, but not the "one-node" swarm we created. Use `docker swarm leave --force`

Docker Swarm is available only for version "3"

### Docker Container

<code>docker run &lt;image&gt;</code>	Run the image, starting a Container
<code>-d</code>	Run in detached mode (in background)
<code>-p 4000:80</code>	Maps the port 80 of the image to the host port 4000
<code>--rm</code>	Removes the container when exited
<code>docker ps</code>	List the running containers (you can check container id)
<code>docker ps -l</code>	List all the containers (you can check container id)
<code>docker stop &lt;container_id&gt;</code>	Stop the container

When you run an image with you are starting a Container, so container is the runtime instance of an image, and consists of the image, an execution environment and a standart set of instructions.

### Swarm

<code>docker swarm init</code>	Initialize a swarm and become swarm manager
<code>docker swarm join</code>	Join a swarm as worker
<code>docker swarm leave --force</code>	Leaves the current swarm

With Docker you can increase resource and capacities by creating a swarm, which are simply several machines (virtual or physical) running a Docker and joined to a cluster.

Swarms have the `swarm manager`, which can issue docker commands normally, and the `workers` which are only there to provide capacity.

Different pieces of the app are called "services" For example, a service for storing application data in a database, a service for the front-end, etc.

Services are just "containers in production." A service only runs one image, but it manages for example what ports it should use and how many replicas of the container should run.

To define a service, you'll need an `docker-compose.yml` file.

For example:

```
version: " 3"
services:
  web:
    image: amicheletti/get-started-part1
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: " 0.1 "
          memory: 50M
        restart_policy:
          condition: on-failure
    ports:
      - " 80: 80"
    networks:
      - webnet
networks:
  webnet:
```

Here you define the image to be loaded, how many replicas, the resource limits and the restart conditions.



By **amicheletti**  
[cheatography.com/amicheletti/](http://cheatography.com/amicheletti/)

Published 17th July, 2017.  
Last updated 14th July, 2017.  
Page 1 of 2.

Sponsored by **CrosswordCheats.com**  
Learn to solve cryptic crosswords!  
<http://crosswordcheats.com>