Docker Basics Cheat Sheet

by amicheletti via cheatography.com/39488/cs/12295/

Docker Image

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

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:
```

Services (cont)

```
- "80:80"
networks:
- webnet
networks:
webnet:
```

Here you define the image to be loaded, how many replicas, the resource limits and the restart conditions.
To run it you must first start: `docker swarm init`
Then run it giving a name:
`docker stack deploy -c docker-compose.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

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

Cheatography

Docker Basics Cheat Sheet

by amicheletti via cheatography.com/39488/cs/12295/

Swarm	
`docker swarm init`	Initialize a swarm and become `swarm manager`
`docker swarm join`	Join a swarm as `worker`
`docker swarm leave --force`	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.

By **amicheletti**
cheatography.com/amicheletti/

Published 17th July, 2017.
Last updated 14th July, 2017.