Docker

Corsi GNU/Linux Avanzati 2017

Filippo Cremonese (fcremo)

21 Marzo 2017

POLITECNICO OPEN unix LABS

Come hack with us.

What is docker?

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.

Ok, and what is a containter?

- A docker container is an isolated execution environment
- System resources can be assigned to it (fs, net, etc)
- Resource usage can be restricted (e.g. to a % of CPU time)
- It is not a chroot
- It is not a virtual machine
  - All containers use the same host kernel
  - Lightweight on CPU and RAM

What this talk is not about

- Cloud!
- Devops
- Microservices architecture
- Horizontal scaling
- Orchestration

Note: some of these buzzwords are actually useful and serious things, but are beyond what can be covered in a short talk.

Installing docker

Docker is often available in the repos, so you can (e.g. on Ubuntu)

```
sudo apt-get install docker.io
```

But the version in the repo is usually outdated.

If you want the latest version you can follow instructions at https://docs.docker.com/engine/installation/

Hello world

```
$ docker run alpine /bin/echo 'Hello, world'
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
627beaf3eaaf: Pull complete
Digest: sha256:58e1a1bb75db1b5a...
Status: Downloaded newer image for alpine:latest
```

What just happened?

Docker has:

- Downloaded Alpine Linux image (a light GNU/Linux distro)
- Unpacked it
- Created a new container
- Run /bin/echo 'Hello world' inside it

Interactive hello world

Execute an interactive shell (-i and -t options)

```
$ docker run -i -t alpine /bin/sh
/ # echo Hello World
Hello World
/ # hostname
9422707bd6c6
/ # cat /etc/issue
Welcome to Alpine Linux 3.5
...
/ # exit
$
```

Noninteractive hello world

Execute a container in background (-d option)

```
$ docker run -
d --name hello_world alpine /bin/sh \
  -
c 'while true; do echo Hello World; sleep 1; done;'
c93138893fc1807bc306706598020d68...
```

Every container has a unique name and hash.
To list existing containers run

```
$ docker ps -a
```

The -a option specifies to print all containers, not just running ones.

Starting to get messy

To delete a container use

```
$ docker rm <container_name_or_hash>
```

Running containers can't be deleted, so to delete hello_world we first have to stop it:

```
$ docker stop hello_world
hello_world
$ docker rm hello_world
hello_world
```

Many commands print the name (or hash) of the affected containers.

Enough hello worlds!

How do I run my own applications in docker?

For this demo we'll use Up1, a node.js application.

`https://github.com/Upload/Up1`

It's a client-side encrypted file sharing service.

Creating an image: Dockerfiles

Docker images are usually built from a Dockerfile.
A Dockerfile is a text file containing a list of directives that are executed to create the image.

Example Dockerfile

```
FROM node:latest
MAINTAINER fcremo@users.github.com
EXPOSE 9000:9000
ENV HTTP="true" HTTP_LISTEN="0.0.0.0:9000" [...]
RUN apt-get install -y git && cd /srv && \
    git clone https://github.com/Upload/Up1 && \
    cd Up1/server && npm install && \
    apt-get remove -y git
WORKDIR /srv/Up1/server
COPY server.conf.template server.conf.template
COPY genconfig.sh genconfig.sh
COPY entrypoint.sh entrypoint.sh
COPY config.js.template ../client/config.js.template
RUN chmod +x genconfig.sh entrypoint.sh
ENTRYPOINT /srv/Up1/server/entrypoint.sh
```

Building the example Dockerfile

```
$ git clone https://github.com/fcremo/Up1-
docker.git
$ cd Up1-docker
$ docker build -t fcremo/up1 .
```

It's gonna take a couple of minutes...

Useful commands

Command	Description
docker run	create and run a container
docker [re]start/stop/kill	[re]start/stop/kill a container
docker ps	list containers
docker rm	delete a container
docker rename	rename a container
docker images	list images
docker build	build an image
docker rmi	delete an image
docker network	manage networks
docker help [command]	get documentation

Run the new image

```
$ docker run -e "API_KEY=random1" \
    -e "DELETE_KEY=random2" \
    --name up1 -p 8080:9000 \
    -v /tmp/up1:/srv/Up1/i/ \
    -d fcremo/up1
```

Option	Meaning
-name up1	Gives the container the name up1
-p 8080:9000	Map host:8080 to container:9000
-e KEY=VAL	Set environment variables
-v /host/:/cont/	Mount /host/ to /cont/
-d	Run in background

Configuring containers

When possible use environment variables

- Create configuration files on first run
  - Usually done using templates
    - shell/sed/awk scripts
    - https://github.com/jwilder/dockerize

Otherwise, mount configuration files

- e.g. docker run -v /host/config:/app/config myapp
- Thou shall not rebuild an image to configure an app

Demo: Up1-docker configuration

How Up1-docker creates configuration files using templates.

Isolating and linking containers

By default, new containers are put into the legacy bridge network.
They can communicate to each other but cannot resolve hostnames dynamically.

Docker allows you to create user defined networks.
Containers on the same network can discover (via DNS) and reach each another.

Linking containers

As an example we'll configure a reverse proxy with caddy to serve up1 over tls.
First, create a network and connect up1 container to it:

```
$ docker network create my-network
$ docker network connect my-network up1
```

Then run caddy in a new container attached to that network.

```
$ docker run -d -
v /var/docker/caddy:/root/.caddy \
    -
v /var/docker/caddy/Caddyfile:/etc/Caddyfile \
    -p 80:80 -
p 443:443 --name caddy --restart always
    --network my-network abiosoft/caddy
```

Managing multiple containers

No one wants to manually manage multiple containers.
Docker-compose lets you define services as groups of containers, using a yml file.
We'll use mattermost, a FOSS group chat like slack, as an example.

Getting docker-compose

Follow instructions at

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

docker-compose.yml example

Get docker-compose.yml from

```
https://github.com/jasl8r/docker-mattermost
```

and execute

```
$ docker-compose up
```

Fine

Grazie per l'attenzione!

`http://www.poul.org`