Software Platforms

LM in Computer Engineering

Massimo Maresca

From Linux namespaces to Docker

- 2000-2010: namespaces/cgroups technologies in Linux introduced to virtualize processors, memory, performance, networking, mass memory etc.
- 2014: Docker packages the namespaces/cgroup technologies in a platform and introduces the concept of Container
- Main elements:
  - Image
  - Container
  - Network
  - Volume

Docker

- Do what indicated in
- https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository
- sudo groupadd docker
- sudo usermod -aG docker $USER

Docker command - Images

- docker image pull httpd
- docker image ls
- docker image rm [-f] httpd
- docker image help

un immagine docker è in sostanza un programma,

lista le varie immagini docker

eliminare un immagine

c'è un repository personale in docker (sul sito) in cui puoi mettere le tue immagini

un container è un programma in esecuzione. è un immagine in esecuzione

Docker command - Containers

- docker container run --publish 80:80 --name webhost -d httpd
- docker container ls
- docker container ls -a
- docker container rm [-f] httpd
- docker container prune
- docker container help

porta che espone

porta da cui invia

puo sia usare immagini in locale sia usare immagini caricate nel repository

- Add forwarding port P to guest port 80
- Access from Browser: localhost:P

Docker container commands

- docker container rm -f <container-id>
- docker container run -p host-port:container-port --name <name> -d <image id>
- docker exec -it <Container ID> /bin/bash
  - -d : detach
  - -it: interactive terminal
- docker container logs <container-id>
- docker container stop <container-id>
- docker container inspect <container-id>
- docker container stats webhost (example: cpu)

Running Tomcat in a Container

- docker image pull tomcat:9.0 tomcat che costruito sopra java sopra linux per cui installa tutto quando esegui questo comando.
- docker container run --publish 9090:8080 --name mytomcat -d tomcat
- docker container ls
- Configure VM Port Forwarding such that:

  Host Port 9095 <-> Guest Port 9090

In the end we have: costruisci un collegamento

  Host Port 9095 <-> Guest Port 9090 <-> mytomcat port 8080

- docker exec -it <Container ID> /bin/bash
- Copy content of webapps.dist to webapps (cp –r command)

Running Ubuntu/Alpine in a Container

- docker image pull ubuntu
- docker container run -it --publish 9090:8080 ubuntu bash
  - apt-get update
  - apt-get install openjdk-8-jre
  - apt-get install wget
  - wget <tomcat download URL>
- docker container ls
- docker container stop <id>
- docker container ls -a
- docker container start -ai <id>
- docker image pull alpine
- docker run -it alpine sh
  - apk add bash
  - bash

scarichiamo ubuntu da dentro ubuntu

adesso che lo abbiamo runnato abbiamo un ubuntu dentro ad un altro ubuntu, è come se avessimo un vm ubuntu dentro un ubuntu.

installa varie cose dentro questo nuovo ubuntu, java toamcat ecc

Networking

- Containers belong to an internal network: bridge

  docker network ls

  docker network inspect bridge

posso creare un network

- Example:
  - Activate Web Server container
  - Inspect Web Server container for IPAddress
  - Activate Alpine container shell
  - apk add curl
  - From Alpine curl Web Server

Networking

- Install tools
  - apt-get install net-tools
  - apt-get install tcpdump
  - apt-get install iputils-ping
- New network creation
  - docker network create <name>
  - docker container run -d --network <network name> --name <Container Name> nginx
  - docker network connect <network id> <container id>
  - docker container inspect <container id>

Docker Names and IP Address

- On all networks but bridge hosts can be referenced by name
- Experiment:
  - create ubuntu -it --network mynet --name c1
    install net-tools and iputils-ping
  - create ubuntu -it --network mynet --name c2
    install net-tools and iputils-ping
  - inspect mynet
  - Inspect c1: get IP Address
  - Inspect c2: : get IP Address
  - ping c1 from c2 by IP Address
  - ping c1 from c2 by Container name

Volumes

docker volume create <name>
docker volume ls
docker volume inspect <id>
docker volume rm <name>
docker container run --mount source=<mount_point>,target=<local_dir>

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.

Experiment:
Start two containers based on ubuntu:latest and share a directory
In a VM shell:
% docker volume create mmvol
% docker container run --name u1 --mount source=mmvol,target=/v1 -it ubuntu bash

In another VM shell:
% docker container run --name u2 --mount source=mmvol,target=/v1 -it ubuntu bash
Check directory sharing

Container Creation/ Image Update through commit

Image creation: docker commit [Container Id] [ImageId]

Example: start from ubuntu

```
% docker container run -it ubuntu bash
```

Add Packages: ping, vi, java 8, wget, tomcat

```
% apt update
% apt install net-tools
% apt install iputils-ping
% apt install vim
% apt install openjdk-8-jdk
% apt install wget
% apt install curl
% wget <tomcat download url>+
<install tomcat>
```

Commit (create a new image)

```
% docker commit <Container Id> [Image Id]
```

Test

Open new container and see whether packages are there and work.

Container upload to a Registry

Possibility to create your own repository and upload/distribute images:

- Go to docker homepage
- Create account (example: massimomaresca)
- Create repository (example: sw-platforms (massimomaresca/sw-platforms))
- Manage repository through the web
- Login through the CLI (docker login)
- Up/Download images through the CLI: commit, push, pull

Example:

- First save running container in local image repository with user/repository:<Image Id>
  - docker commit <container Id> <user_id>/<Repository Name>
  - For example: docker commit abcd massimomaresca/xyz

- Then upload image to docker repository
  - docker push massimomaresca/xyz

Automatic Image Building

Dockerfile: instructions to build an image

The goals are:
- to document
- to edit
- to be able to repeat

the process that leads to the creation of an image.

Docker command: docker build –f <Dockerfile> [default Dockerfile]
–t <Image Name>
<Dockerfile path>

Image Building Scripts

Dockerfile: instructions to build an image

General Syntax: # Comment
INSTRUCTION arguments

Instructions:

FROM	From what
ENV	Environment
WORKDIR	Working Directory
ADD	Add Files
COPY	Copy files
RUN	Run a program
EXPOSE	Expose a port

Summary of items

- Pull Image from Docker Repository
- Run Container (-it, -d) or Exec Container
- Update Image by running Container and installing additional Packages and Application Programs
- Commit Container to Local Image Repository
- Push Image to Docker Web Repository

- Create and Inspect Network
- Run Container with Network attachment
- Inspect Container

- Create and Inspect Volume
- Run Container Mounting Shared Volume

- Docker Build to make Image Building Automatic

Summary of items through an example

- pull from docker repository
- docker image pull ubuntu_latest
- docker container run ubuntu
- adduser <new_user>
- apt-get update
- apt-get install net-tools
- apt-get update tcpdump
- apt-get update iputils-ping
- apt-get install openjdk-8-jre
- apt-get install wget
- apt-get install curl
- apt-get install vim
- wget tomcat 9 from appropriate mirror

  e.g., https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.56/bin/apache-tomcat-9.0.56.tar.gz

- place tomcat in /usr/local/share/ (gunzip + tar xf)
- configure +rwx tomcat
- configure <new_user> environment variable

  export CATALINA_HOME=< Tomcat 9 folder>
- configure tomcat (users, remote administration, etc.)
- on another terminal: commit <container> <new-image-name>

Image Building Example #1 – From Scratch

See folder: from-scratch
The image is just an executable file.

hello.c

```
#include <stdio.h>
const char message[] ="Here I am";
void main() {
        printf("%s\n", message);
}
```

Makefile

```
hello: hello.c
gcc -static hello.c -o hello
```

Dockerfile

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

Image Building Example #2 – From busybox

See folder: from-busybox-hello
The image starts from busybox, a small Linux-like system, includes a file called hello_msg, copies it to another file called new_hello_msg, and prints such a file on the stdout.

Dockerfile

```
# Starting from small size Linux –like system
FROM busybox
# Copying a text from local folder to image
COPY hello_msg  /
# Copying the message from a file to another file (to show COPY command)
RUN cp /hello_msg /new_hello_msg
# Executing a command on container
CMD ["cat", "/new_hello_msg”]
```

Image Building Example #3 – From openjdk

See folder: from-openjdk-to-tomcat
The image starts from an image available in the Docker repository, adds a pre-loaded Tomcat servlet container, takes care of it deployment and logs the process, exposes the 8080 port and runs it.

Dockerfile

```
FROM openjdk:8-jre
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir /usr/local/tomcat
COPY /apache-tomcat-9.0.30.tar.gz /usr/local
RUN ls -l /usr/local
RUN tar xf /usr/local/apache-tomcat-9.0.30.tar.gz  -C /usr/local
RUN mv -T /usr/local/apache-tomcat-9.0.30 /usr/local/tomcat
RUN rm /usr/local/apache-tomcat-9.0.30.tar.gz*;
RUN ls -l /usr/local# RUN
RUN ls -l /usr/local/tomcat/
#
EXPOSE 8080
WORKDIR /usr/local/tomcat/bin
#
CMD ["catalina.sh", "run"]
```

Then run container as follows: docker container run --publish 9090:8080 -d <Image_Id>
And try: curl localhost:9090 on host

Image Building Example #4: Tomcat shared dir

See folder: from-openjdk-to-tomcat-shared-dir

The image:

- starts from an image available in the Docker repository,
- adds a pre-loaded Tomcat servlet container,
- takes care of its deployment and logs the process,
- exposes the 8080 port
- activates Tomcat

- Use script “create” to build image “u0”

- Use scripts “activate1” and “activate2” to run two containers (“cu1” and “cu2”) based on image “u0”.
- Enter the two containers through: docker exec –it <cu1 or cu2> bash
- The two containers share directory /ext-dir where the Web apps must be located (see ext-dir in file “server.xml”)

Image Building Example #4: Tomcat shared dir

Dockerfile

```
FROM openjdk:8-jre
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir /usr/local/tomcat

COPY /apache-tomcat-9.0.30.tar.gz /usr/local
RUN tar xf /usr/local/apache-tomcat-9.0.30.tar.gz  -C /usr/local
RUN rm /usr/local/apache-tomcat-9.0.30.tar.gz*;
RUN mv -T /usr/local/apache-tomcat-9.0.30 /usr/local/tomcat
#
RUN rm /usr/local/tomcat/webapps/manager/META-INF/context.xml
ADD context.xml /usr/local/tomcat/webapps/manager/META-INF/context.xml
RUN rm /usr/local/tomcat/conf/tomcat-users.xml
ADD tomcat-users.xml /usr/local/tomcat/conf/tomcat-users.xml
ADD manager.xml /usr/local/tomcat/conf/Catalina/localhost/manager.xml
RUN rm /usr/local/tomcat/conf/server.xml
ADD server.xml /usr/local/tomcat/conf/server.xml
#
EXPOSE 8080
WORKDIR /usr/local/tomcat/bin
CMD ["catalina.sh", "run"]
```

Image Building Example #4: Tomcat shared dir

In server.xml

…

Host name="localhost" appBase="/ext-dir"

…

Activation

Replace /home/max/ *with your path*

```
docker run -p 8081:8080 -v /home/max/shared-webapps:/ext-dir -d --name cu1 u0
docker run -p 8082:8080 -v /home/max/shared-webapps:/ext-dir -d --name cu2 u0
```

Docker Homework and Test

Test #1

- Activate two containers running Tomcat each of which sharing its webapp directory with a directory on the host VM, so as to allow deploying/undeploying Web Applications on the host VM.
- Connect the two containers to a “docker defined network” called mynet
- Name the two containers respectively “tomcat1” and “tomcat2”.
- Configure tomcat1 so as to establish a correspondence between port 8080 and port 8081 port in the host VM.
- Configure tomcat2 so as to establish a correspondence between port 8080 and port 8082 port in the host VM.
- Configure Vbox so as to allow a browser running in the Guest system (Window or Linux) to access the two servlet containers though the host VM using static port forwarding. è host in realtà
- Access each of the two containers from the Guest browser. Deploy/Undeploy Web Applications in the two Tomcat servlet containers.

Test #2

- Create a third container to be used as a Load Balancer (based on Nginx) to redirect external access to the two tomcat servlet containers alternatively.
- Connect the Load Balancer to the same network as the two servlet containers.
- Create the configuration file using both the IP addresses (in a first version) and the names assigned to the containers.
- Access the Load Balancer from the host VM through curl, and then access to the load Balancer through a browser running in the Windows host.

Example #5: Load Balancing through Nginx

Create the Tomcat Image as in the previous experiment.
Create a Load Balancer Image from the Nginx image in which you edit the configuration file located in /etc/nginx/nginx.conf as follows:

```
http {
        .....
   server {
        listen 8080;
        location / {
            proxy_pass http://backend;
        }
    }
    upstream backend {
        server <Ip Address/Name>:8080;
        server <Ip Address/Name>:8080;
    }
}
```

Example #5: Load Balancing through Nginx

Create the two Tomcat Docker Containers and the Load Balancer Docker Container making sure that they access the same network (e.g., maxnet) and that they expose the appropriate ports (the two Tomcat containers expose port 8080) the Load Balancer exposes port 8080 and, for example translates Container port 8080 to VM host port 8085. Finally, the VM platform (e.g., Oracle Vbox) creates a correspondence between host port 9095 and guest port 8085, where host denote the hosting machine (typically MS Windows), whereas guest refers to the Linux Ubuntu VM.

The alternance of Load Balancing can be accessed on a Browser, provided that you disable caching, by repeatedly access localhost:9095, which forwards to the VM:8085, where the Load Balancer receives the requests and forwards them to the two Tomcat containers using a round robin scheduling.

The two Tomcat Docker Containers share the same webapps folder.
A modified ROOT servlet presents a home page which displays the address of the server.