AN INTRODUCTION TO OS VIRTUALIZATION (AND DOCKER)

Davide Neri, PhD Student, University of Pisa
davide.neri@di.unipi.it

Mauriana Pesaresi Seminar
20/04/2017
Summary

1. Virtualization
   1. Hardware (HW) virtualization
   2. Operating system (OS) virtualization
   3. Performance comparisons (HW vs OS virtualization)

2. Docker platform
   a. Docker image
   b. Docker container
   c. Docker registry

3. Try Docker
Virtualization and Cloud

- Maximize resources
- Shared system
- Cost reduction
- Decouple applications from the hardware
Hardware virtualization

Hypervisor (VMM)* manages multiple Virtual machines (VMs) on a single host.

*type1 loaded directly on the Hw (data center), type 2 loaded in an OS running on the HW (desktop)
Operating system (OS) virtualization

The **kernel** of an operating system allows the existence of multiple isolated user-space instance, instead of just one. Such instances are called **containers**.

(*) A *kernel* is the part of the operating system that mediates access to system resources. It's responsible for enabling multiple applications to effectively share the hardware by controlling access to CPU, memory, disk I/O, and networking.
Hardware VS OS virtualization

- Lightweigth virtualization
- Less resource consumptions (CPU, Memory)
- Faster start-up
Performance comparisons

Time to scale:
- 1 container: 8 secs
- 1 VM: 3 mins

A. M. Joy, "Performance comparison between Linux containers and virtual machines," 2015 International Conference on Advances in Computer Engineering and Applications,
A CONTAINER ENGINE
Docker

Docker is a platform for developing, shipping, and running applications using **container-based** technology.
Docker engine

The Docker engine (aka. Docker daemon) is the program that enables containers to be built, shipped and run.

Uses the linux kernel features (cgroups, namespaces,...) to run isolated containers.
Linux kernel features (some...)

cgroups

Control Groups provides resource limiting and metering (e.g. memory, CPU, network, I/O).

“Limit How much you can use”

Namespaces

Provides processes with their own view of the system (namespaces example: pid, net, user…)

“Limits what you can see (and therefore use)”
Docker architecture

Client-server architecture:

- **Server:** Docker Engine (daemon)
- **Client:** `docker` command line interface.
Docker platform

Key components of the Docker platform:

- **Docker image**: e.g. ubuntu:16.04
- **Docker container**
- **Docker registry**
Docker: **build** an image

An *Image* is a read-only template composed by a filesystem and parameters to use at runtime.
Docker image

References a list of read-only layers that represent filesystem differences. Layers are stacked on top of each other to form a base for a container’s root filesystem.

Add {your code}

Add Python

Add pip
Docker: **run** a container

A *container* is a running instance of an image. From a single image multiple containers can be run.

```bash
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]`
```
Hello-world with Docker

Other examples:

```bash
docker run alpine ping www.di.unipi.it
docker run -it ubuntu bash
```
Docker: **pull/push** to a registry

The registry is where we store our images. It can be a local registry or the public registry (Docker Hub: https://hub.docker.com/)

```bash
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
docker push [OPTIONS] NAME[:TAG]
```
Docker registry

https://hub.docker.com/
TRY DOCKER

http://labs.play-with-docker.com/
THE END

Q&A

Davide Neri
davide.neri@di.unipi.it