Linux Containers and Dockers

Quando, vantaggi e svantaggi

Dr. Fabio Fumarola
Contents

• The Evolution of IT
• The Solutions: Virtual Machines vs Vagrant vs Docker
• Differences
• Examples
  – Vagrant
  – Docker
• P.S. CoreOS
From 1995 to 2015

Client-Server App

Well-defined stack:
- O/S
- Runtime
- Middleware

Monolithic Physical Infrastructure

Thin app on mobile, tablet

Assembled by developers using best available services

Running on any available set of physical resources (public/private/virtualized)
2015 in Detail

**Static website**

- nginx 1.5 + modsecurity + openssl + bootstrap

**Background workers**

- Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

**User DB**

- postgresql + pgv8 + v8

**Queue**

- Redis + redis-sentinel

**Analytics DB**

- hadoop + hive + thrift + OpenJDK

**Web frontend**

- Ruby + Rails + sass + Unicorn

**API endpoint**

- Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

**Development VM**

**QA server**

**Public Cloud**

**Production Cluster**

**Customer Data Center**

**Disaster recovery**

**Contributor’s laptop**

**Production Servers**
Challenges

• How to ensure that services interact consistently?
• How to avoid to setup N different configurations and dependencies for each service?
• How to migrate and scale quickly ensuring compatibility?
• How to replicate my VM and services quickly?
### How to deal with different confs?

|---------------------|----------------|-----------|--------------------|----------------|--------------|----------------------|------------------|
1. VIRTUAL MACHINES
Virtual Machines

- Run on top of an Hypervisor

**Pros**
- fully virtualized OS
- Totally isolated

**Cons**
- Needs to take a snapshot of the entire VM to replicate
- Uses a lot of space
- Slow to move around
Hypervisors Trend

2011
  – XEN: Default choice given Rackspace and Amazon use
  – KVM: Bleeding edge users

2012
  – KVM: Emerges as the lead
  – XEN: Loses momentum
Hipervisors Trend

2013
- KVM: Maintains lead (around 90%+ for Mirantis)
- VMware: Emerges as a surprising second choice
- Containers (LXC, Parallels, Docker): Web Hosting and SAS focused
- Xen and HyperV: Infrequent requests (XenServer.org)

2014 – 2015
- ???
2. VAGRANT
Vagrant

• Open source VM manager released in 2010
• It allows you to script and package VMs config and the provisioning setup via a VagrantFile
• It is designed to run on top of almost any VM tool: VirtualBox, VMVare, AWS, OpenStack
• It can be used together with provisioning tools such as shell scripts, Chef and Puppet.

1. https://github.com/cloudbau/vagrant-openstack-plugin
Vagrant: idea

Use a VagrantFile to install
1. an operating system
2. Required libraries and software

and finally run programs and processes of your final application
Vagrant: Feature

- Command-Line Interface
- Vagrant Share
- VagrantFile
- Boxes
- Provisioning
- Networking
- Synced Folders
- Multi-Machine
- Providers
- Plugins

https://www.vagrantup.com/downloads
Vagrant: Demo

- It allows us to interact with Vagrant
- It offers the following commands: box, connect, destroy, halt, init, login, package a vm, rdp, ...

https://docs.vagrantup.com/v2/cli/index.html
Vagrant Example

1. Download and install VirtualBox and Vagrant

$ mkdir vagrant_first_vm && cd vagrant_first_vm
$ vagrant init

2. This will place a VagrantFile in the directory

3. Install a Box

$ vagrant box add ubuntu/trusty64

4. Using a Box -> https://vagrantcloud.com/

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end
Vagran: Start

1. Start the box
   
   ```bash
   $ vagrant up
   ```

2. Login into the vm
   
   ```bash
   $ vagrant ssh
   ```

3. You can destroy the vm by
   
   ```bash
   $ vagrant destroy
   ```
Vagrant: Synced Folders

- By default, it shares your project directory to the /vagrant directory on the guest machine.

  ```
  $ vagrant up
  $ vagrant ssh
  $ ls /vagrant
  --Vagrantfile
  ```

- If you create a file on your guest OS the file will be on the vagrant vm.

  ```
  $ touch pippo.txt
  $vagrant ssh
  $ls /vagrant/
  ```
Vagrant: Provisioning

- Let’s install Apache via a bootstrap.sh file

```
#!/usr/bin/env bash

apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /vagrant /var/www
```

- If you create a file on your guest os the file will be on the vagrant vm. (vagrant reload --provision)

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
end
```
Vagrant: Networking

- **Port Forwarding**: Allows you to specify ports on the guest machine to share via a port on the host machine

  ```ruby
  Vagrant.configure("2") do |config|
    config.vm.box = "hashicorp/precise32"
    config.vm.provision :shell, path: "bootstrap.sh"
    config.vm.network :forwarded_port, host: 4567, guest: 80
  end
  ```

- By running `vagrant reload` or `vagrant up` we can see on [http://127.0.0.1:4567](http://127.0.0.1:4567) our apache

- It supports also bridge configurations and other configurations ([https://docs.vagrantup.com/v2/networking/](https://docs.vagrantup.com/v2/networking/))
Vagrant: Share and Provider

- It is possible to share Vagrant box via vagrant cloud (but?)

Providers
- By default Vagrant is configured with VirtualBox but you can change the provider

```
$ vagrant up --provider=vmware_fusion
$ vagrant up --provider=aws
```

- How?

```
$ vagrant plugin install vagrant-aws
```
Vagrant: AWS Vagrantfile

Vagrant.configure("2") do |config|
  # config.vm.box = "sean"

  config.vm.provider :aws do |aws, override|
    aws.access_key_id = "AAAAIIIIYYYY4444AAAA"
    aws.secret_access_key = "c344441LoollU322223526LabcdeQL12E34At3mm"
    aws.keypair_name = "iheavy"

    aws.ami = "ami-7747d01e"

    override.ssh.username = "ubuntu"
    override.ssh.private_key_path = "/var/root/iheavy_aws/pk-XHHHHHMMMAABPEDEFGHOAOJH1QBH5324.pem"
  end
end
3. DOCKER
Quick Survey

• How many people have heard of Docker before this Seminar?
• How many people have tried Docker?
• How many people are using Docker in production?
What is Docker?

“Docker is an open-source engine to easily create lightweight, portable, self-sufficient containers from any application. The same container that a developer builds and test on a laptop can run at scale, in production, on VMs, OpenStack cluster, public clouds and more.”

Docker.io
Docker in simple words

• It is a technology that allow you running applications inside containers (not VM)
• This assures that libraries and package needed by the application you run are always the same.
• This means you can make a container for Memcache and another for Redis and they will work the same in any OS (also in Vagrant).
How does docker work?

• LinuX Containers (LXC)
• Control Groups & Namespaces (CGroups)
• AUFS
• Client – Server with an HTTP API
LXC- Linux Containers

• It is a user-space interface for the Linux kernel containment features
• Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.
• Currently LXC can apply the following kernel features to contain processes:
  – Kernel namespaces (ipc, uts, mount, pid, network and user)
  – Apparmor and SELinux profiles
  – Seccomp policies
  – Chroots (using pivot_root)
  – Kernel capabilities & Control groups (cgroups)
cgroups

• Control groups is a Linux kernel feature to limit, account and isolate resource usage (CPU, memory, disk I/O, etc) of process groups.

• Features:
  – Resource limitation: limit CPU, memory...
  – Prioritization: assign more CPU etc to some groups.
  – Accounting: to measure the resource usage.
  – Control: freezing groups or check-pointing and restarting.
LCX based Containers

- It allows us to run a Linux system within another Linux system.
- A container is a group of processes on a Linux box, put together is an isolated environment.

- From the inside it looks like a VM
- From the outside, it looks like normal processes
Docker Features

• VE (Virtual Environments) based on LXC
• Portable deployment across machines
• Versioning: docker include git-like capabilities for tracking versions of a container
• Component reuse: it allows building or stacking already created packages. You can create ‘base images’ and then running more machine based on the image.
• Shared libraries: there is a public repository with several images (https://registry.hub.docker.com/)
Why are Docker Containers lightweight?

Original App
(No OS to take up space, resources, or require restart)

Copy of App
No OS. Can Share bins/libs

Modified App
Union file system allows us to only save the diffs Between container A and container A'

VMs

Containers
Docker Installation Ubuntu

• AUFS support
  
  ```bash
  $ sudo apt-get update
  $ sudo apt-get install linux-image-extra-`uname –r`
  ```

• Add docker repo
  
  ```bash
  $ sudo sh –c “curl https://get.docker.io/gpg | apt-key add -”
  $ sudo sh –c “echo deb http://get.docker.io/ubuntu docker
  \n  main > /etc/apt/sources.list.d/docker.list”
  ```

• Install
  
  ```bash
  $> sudo apt-get update
  $> sudo apt-get install lxc-docker
  ```
Docker Installation Vagrant

- Clone the docker repository
  
  $ git clone https://github.com/dotcloud/docker.git

- Startup the vagrant image
  
  $ vagrant up

- SSH into the image
  
  $ vagrant ssh

- Docker client works normally
BASE COMMANDS
Docker: hello world

- Get one base image

  $ docker pull ubuntu

- List images on your system

- Print hello world

  $ docker run ubuntu:12.10 echo “hello world”
Detached mode

- Run in Docker using the detached flag (-d)
  
  ```
  $ docker run -d ubuntu sh -c "while true; do echo hello world; sleep 1; done"
  $ docker ps
  ```

- Get the container’s id
  
  ```
  $ docker ps
  ```

- Attach to the container
  
  ```
  $ docker attach <container id>
  ```

- Stop/Start/Restart the container
  
  ```
  $ docker stop <container id>
  ```
Public Index & Network

• Pull an apache image from the public repo

$ docker search apache
$ docker pull creack/apache2

• Run the image and check the ports

$ docker run –d creack/apache2
$ docker ps

• Expose public ports

$ docker ps
Using Docker: the interactive way

$ docker run -i -t ubuntu bash
root@82fdsfs4885:/#
root@82fdsfs4885:/# apt-get update
root@82fdsfs4885:/# apt-get install memcached
root@82fdsfs4885:/# exit

• **Commit the Image**

$ docker commit `docker ps -q -l` user/memcached

• **Start the image**

$ docker crun -d -p 11211 -u daemon user/memcached memcached
Docker: app using scripts

• Write a Dockerfile

```dockerfile
# Memcached
FROM ubuntu
MAINTAINER Fabio Fumarola

RUN apt-get update
RUN apt-get install –y memcached

ENTRYPOINT ["memcached"]
USER daemon
EXPOSE 11211
```

• Build and Start the image

```
$ docker build –t=fabio/memcached
$ docker run –d fabio/memcached memcached
```
Other Commands

- Docker cp: copy a file from container to host
- Docker diff: print container changes
- Docker top: display running processes in a container
- Docker rm /rmi: delete container/image
- Docker wait: wait until container stop and print exit code

Docker vs Vagrant?

- Less memory for Dockers w.r.t VMs
- With a VM you get more isolation, but is much heavier. Indeed you can run 1000 of Dockers in a machine but not thousand of VMs with Xen.
- A VM requires minutes to start a Docker seconds

There are pros and cons for each type.
- If you want full isolation with guaranteed resources a full VM is the way to go.
- If you want hundred of isolate processes into a reasonably sized host then Docker might be the best solution
CORE OS
CoreOS

- A minimal operating system
- Painless updating: utilizes active/passive scheme to update the OS as single unit instead of package by package.
- Docker container
- Clustered by default
- Distributed System tools: etcd key-value store
- Service discovery: easily locate where service are running in the cluster
- High availability and automatic fail-over
CoreOS

Clustered by default

High availability and automatic fail-over
Docker with CoreOS

Features

• Automatically runs on each CoreOS machine
• Updated with regular automatic OS updates
• Integrates with etcd
• Networking automatically configured

Example Akka cluster + Docker + CoreOS
https://github.com/dennybritz/akka-cluster-deploy
References

- [https://docs.vagrantup.com/v2/](https://docs.vagrantup.com/v2/)
- [https://github.com/patrickdlee/vagrant-examples](https://github.com/patrickdlee/vagrant-examples)
- [https://linuxcontainers.org/](https://linuxcontainers.org/) LXC
- [https://medium.com/@_marcos_otero/docker-vs-vagrant-582135beb623](https://medium.com/@_marcos_otero/docker-vs-vagrant-582135beb623)
- [https://coreos.com/using-coreos/docker/](https://coreos.com/using-coreos/docker/)