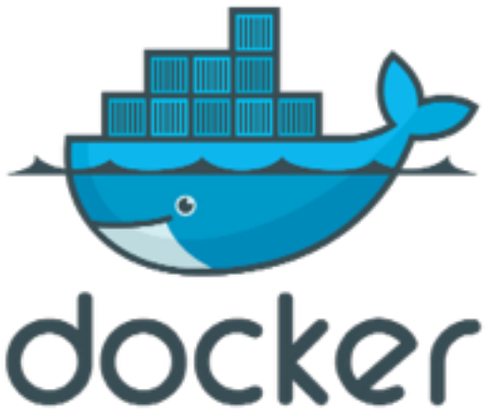
Software Engineering and Architecture
Introduction to Vagrant and Docker

Olivier Liechti
HEIG-VD
olivier.liechti@heig-vd.ch

MSE | MASTER OF SCIENCE IN ENGINEERING

What are Vagrant and Docker?

- Vagrant and Docker are two **complementary** tools.
- Both are related to **virtualization**:
  - Vagrant is used to configure and control "**heavyweight**" VMs
  - Docker is used to create and execute "**lightweight**" Linux containers
- Typically, you use Vagrant to manage a Linux VM, in which you install Docker. Within the VM, you deploy your application services in a collection of Docker containers.



Why Vagrant and Docker in this course?

- Vagrant and Docker are closely related to **continuous delivery**.
- They are useful throughout the **entire software delivery lifecycle**, as they:
  - create **homogenous** environments on developer machines.
  - make it possible to **replicate a complete IT infrastructure** on a developer machine.
  - make it possible to **automate the construction and management** of the IT infrastructure ("infrastructure as code")
- They are associated to a big trend in the industry: **micro-services architectures**.

How to use Vagrant?

- Vagrant was initially created to **manage Virtual Box VMs**. Today, other types of VMs (on local machines and in the cloud) are supported.
- Essentially, the idea is that instead of using the Virtual Box GUI to create, configure, control and use your VMs, you write scripts and use command line tools.
- "**Provisioning**" is the process of installing additional software on top of a "box". There are different ways to do that: shell scripts and DevOps tools such as Puppet, Chef or Ansible.
- The **community is sharing "boxes"**, which you can use as a starting point.

VAGRANT

Discover Vagrant Boxes

https://atlas.hashicorp.com/boxes/search?utf8=✓&sort=&provider=&q=jenkins

ATLAS

Sign up Sign in

Discover Vagrant Boxes

This page lets you discover and use Vagrant Boxes created by the community. You can search by operating system, architecture or provider.

jenkins

Provider filter: virtualbox | vmware_desktop | digitalocean | aws | rackspace | hyperv | parallels

Sort by: Recently Created | Recently Updated | Downloads | Favorites

scottpgallagher/ubuntu-14_04-jenkins Ubuntu 14.04 with Jenkins and Docker — 411 downloads | 0.1.0 | last release 8 months ago

wgarcia/centos65-jenkins Jenkins on CentOS 6.5 — 289 downloads | 0.1.0 | last release 8 months ago

jnj-devops/centos65-jenkins Centos 6.5 and Jenkins 1.576 devbox. — 37 downloads | 0.1.0 | last release 6 months ago

kristallizer/adobe-jenkins — 35 downloads | 0.1.1 | last release 3 months ago

dzuluaga/apigee-ci-jenkins-maven-git-jmeter — 17 downloads | 0.1.0 | last release 11 months ago

Previous Next

Help Terms Privacy Security

How to use Vagrant?

```
Usage: vagrant [options] <command> [<args>]

    -v, --version                    Print the version and exit.
    -h, --help                       Print this help.

Common commands:
     box             manages boxes: installation, removal, etc.
     connect         connect to a remotely shared Vagrant environment
     destroy         stops and deletes all traces of the vagrant machine
     global-status   outputs status Vagrant environments for this user
     halt            stops the vagrant machine
     help            shows the help for a subcommand
     init            initializes a new Vagrant environment by creating a Vagrantfile
     login           log in to Vagrant Cloud
     package         packages a running vagrant environment into a box
     plugin          manages plugins: install, uninstall, update, etc.
     provision       provisions the vagrant machine
     rdp             connects to machine via RDP
     reload          restarts vagrant machine, loads new Vagrantfile configuration
     resume          resume a suspended vagrant machine
     share           share your Vagrant environment with anyone in the world
     ssh             connects to machine via SSH
     ssh-config      outputs OpenSSH valid configuration to connect to the machine
     status          outputs status of the vagrant machine
     suspend         suspends the machine
     up              starts and provisions the vagrant environment
     version         prints current and latest Vagrant version
```

The Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "phusion/ubuntu-14.04-amd64"
  config.vm.network "private_network", ip: "192.168.42.42"

  config.vm.provision "shell", path: "provision.sh", privileged: false

  # config.vm.network "forwarded_port", guest: 80, host: 8080
  # config.ssh.forward_agent = true
  # config.vm.synced_folder "../data", "/vagrant_data"

  config.ssh.forward_x11 = true

  # config.vm.provider "virtualbox" do |vb|
  #   # Don't boot with headless mode
  #   vb.gui = true
  #
  #   # Use VBoxManage to customize the VM. For example to change memory:
  #   vb.customize ["modifyvm", :id, "--memory", "1024"]
  # end

end
```

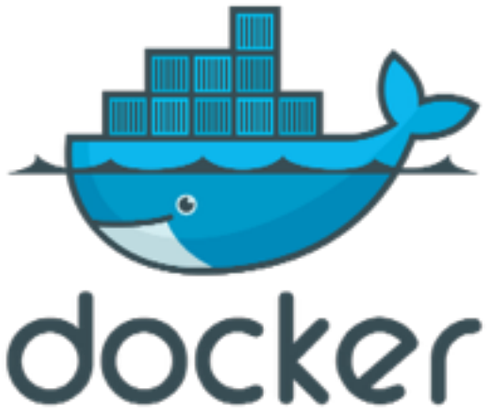
Where to install the software?

- Vagrant was used before Docker became really popular. At that point, most people were installing a lot of software directly on top of the “box”.
- For instance, a web development company would create a “box” with its standard tools (web framework, database, build tools, etc.).
- **When Vagrant is used with Docker, the tendency is not to install too much software on top of the box**. Instead, the software is installed within the Docker containers.
- There are several Linux distributions that have been created to provide a streamlined and optimized environment for running containers: CoreOS, Project Atomic and Snappy Ubuntu.

VAGRANT

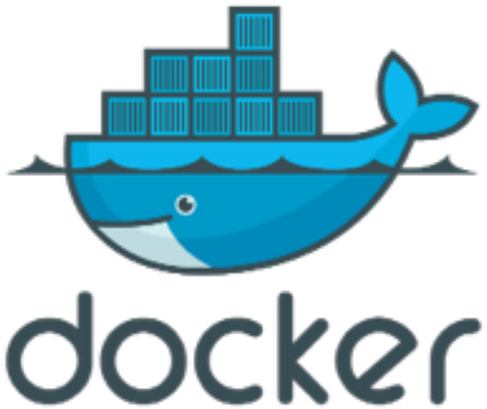
What is Docker and how to use it?

- Docker is built on top of **Linux Containers** (LXC), which was developed to run multiple isolated environments on top a single host.
- Docker makes it possible to create and control linux container in a standard way, with **command line tools and APIs**.
- Docker is closely related to the **micro-services architectural style**.
  - The application is decomposed into independent services.
  - Every service can be developed with the most appropriate technology (some services may be developed with Node.JS & MongoDB, others with Java & MySQL, etc.).
  - Services are deployed and managed independently.
  - Services communicate with standard interfaces.



Key concepts

- You typically execute **one service** in every Docker container:
  - one container for the Node.JS application service
  - one container for the MongoDB persistence service
  - one container for the nginx reverse proxy service
- Every container has its own IP address.
- If you need to scale, you can **run several containers of the same type** (and balance the load between them).
- Docker containers that run services are **not persistent**. It is not a big deal if they die, as they can be quickly restarted (in a known initial state).



How to use Docker?

- Before running **Docker containers**, you need **Docker image(s)**.
- The **community** provides a lot of Docker images that you can use as a starting point. You create your own images by extending existing ones.
- Every Docker image is defined in a **Dockerfile**, which indicates things like:
  - The base image (i.e. the “ancestor” that we extend)
  - The commands to run at image creation time
  - The network ports to expose
  - The command to run at container startup time

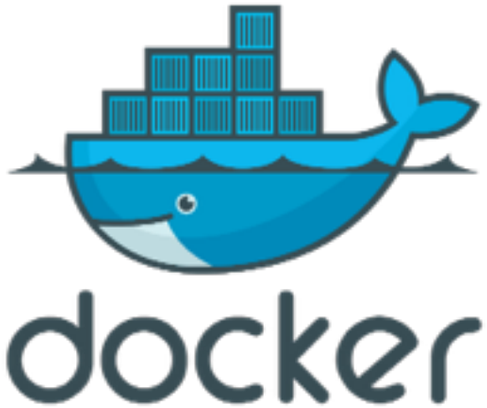
docker

Create an image from a Dockerfile

```
docker build [OPTIONS] PATH | URL | -

Build a new image from the source code at PATH

  --force-rm=false         Always remove intermediate containers, even after unsuccessful builds
  --no-cache=false         Do not use cache when building the image
  --pull=false             Always attempt to pull a newer version of the image
  -q, --quiet=false        Suppress the verbose output generated by the containers
  --rm=true                Remove intermediate containers after a successful build
  -t, --tag=""             Repository name (and optionally a tag) to be applied to the resulting image in case of success
```



Basic commands

docker run myimage

docker run -it myimage /bin/bash

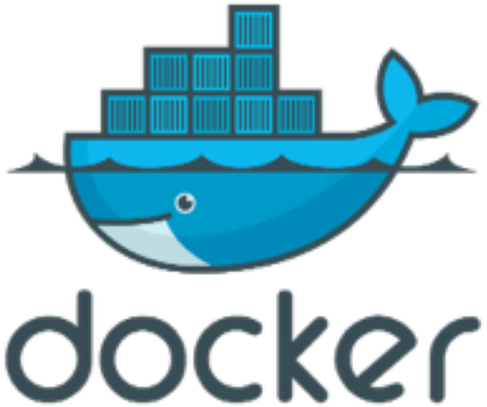
Use this command if you want to explore the file system of the containers run from the “myimage” image.

docker ps

Use this command to see the list of running containers

docker ps -a

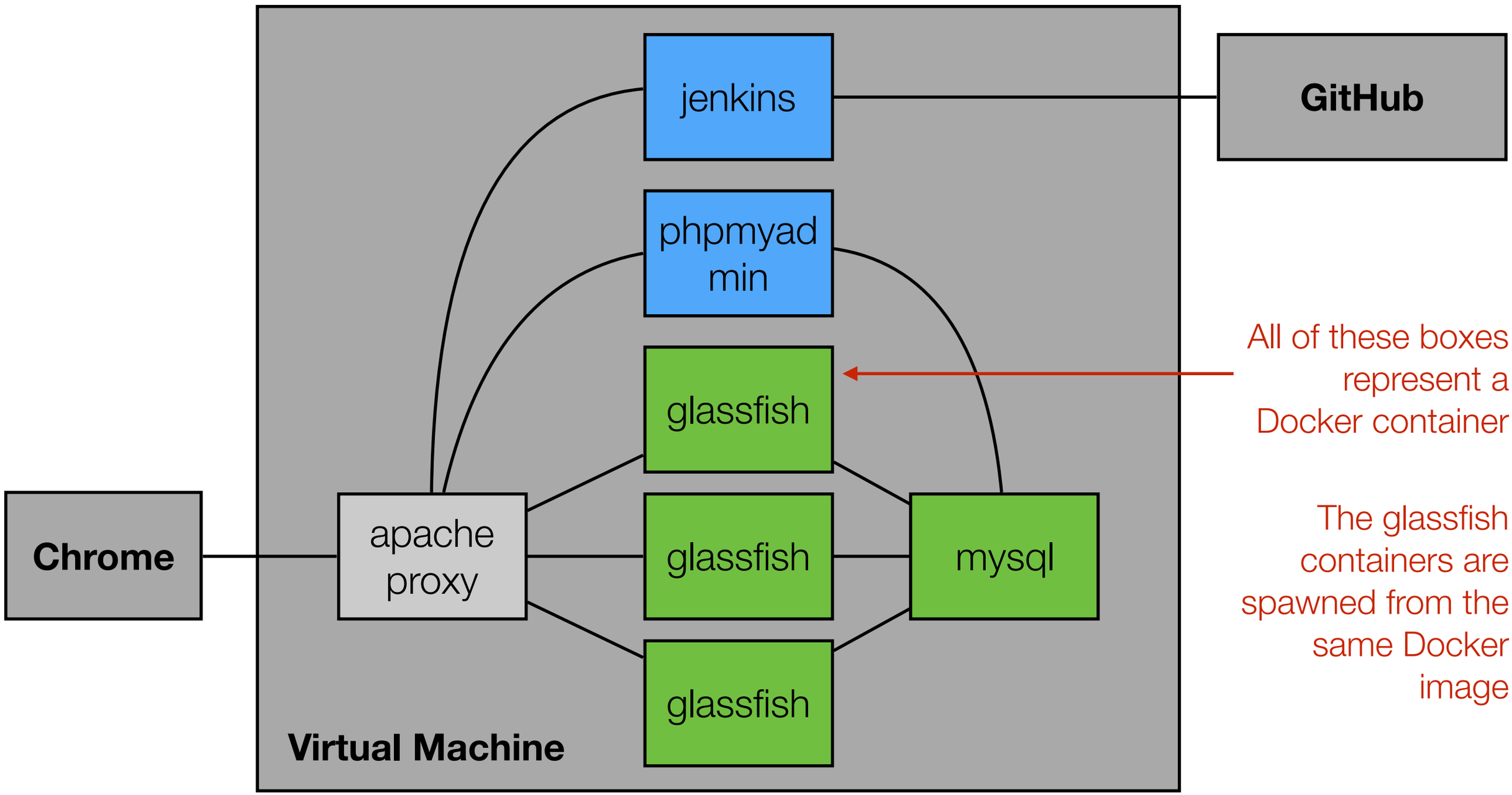
Use this command to see the list of all containers (also “old” ones)



Concepts

- **Networking & links**
  - Docker containers have IP addresses; cross-container communication can be blocked (**firewall**) by default and is recommended.
  - When starting a container, it is possible to "**link**" it to others (this expresses a dependency between services).
- **Data volume containers**
  - While containers that host application services are not persistent, Docker also makes it possible to use **data volume containers**.
  - If you use Docker to provide a persistence service (Postgres, MongoDB, etc.), you typically have (at least) one container for the DB service and one container for the data.

Building a CI pipeline with Vagrant & Docker



Building a CI pipeline with Vagrant & Docker

https://github.com/lotaris/docker-demo
https://github.com/wasadigi/Teaching-HEIGVD-AMT/blob/master/lectures/lecture-08/Docker.pdf