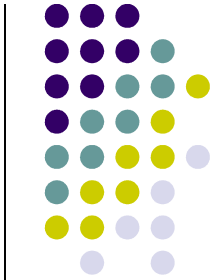
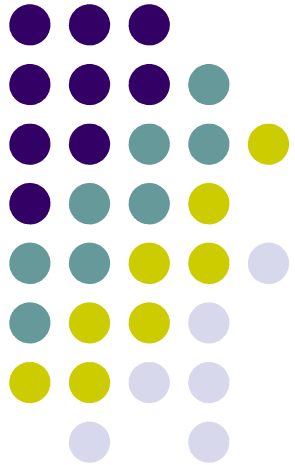


Get out of my cloud: interacción con OpenStack y uso de contenedores con Docker

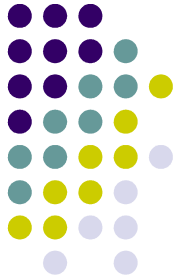


Seminario de Informática Mirian Andrés
26/05/2016

Jesús María Aransay Azofra
Universidad de La Rioja
Departamento de Matemáticas y Computación

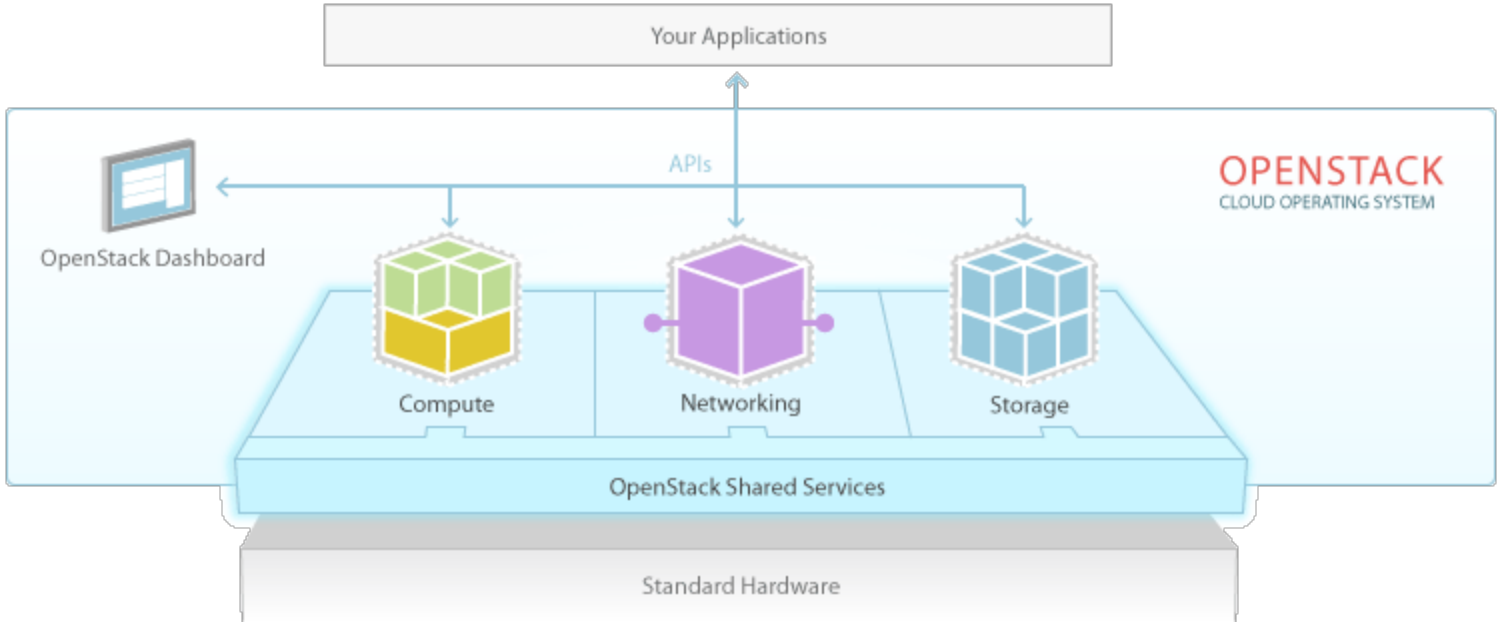
This work is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Index



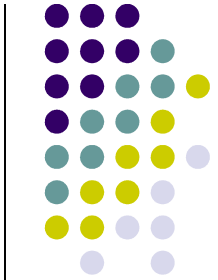
1.1. OpenStack: introduction

OpenStack: IaaS (Infrastructure as a service)



OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter

1.1. OpenStack: introduction



OpenStack

Getting Started (Nebula = NASA + RackSpace):
http://docs.openstack.org/icehouse/training-guides/content/operator-getting-started.html

Open Source: http://www.openstack.org/community/ (API REST internally implemented in Python)

Supported by several enterprises (IBM, HP...):
http://www.openstack.org/foundation/companies/

1.2. OpenStack: ways of interaction

OpenStack: Interacting with OpenStack

At least, four different ways of interaction with OpenStack are available:

1. Dashboard; https://iaas.ceta-ciemat.es/dashboard/

1.2. OpenStack: ways of interaction

OpenStack: Interacting with OpenStack

At least, four different ways of interaction with OpenStack are available:

2. CLI; shell, bash...; http://docs.openstack.org/cli-reference/

1.2. OpenStack: ways of interaction

OpenStack: Interacting with OpenStack

At least, four different ways of interaction with OpenStack are available:

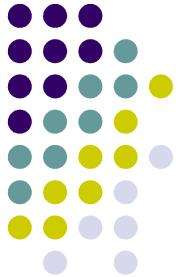
3. SDKs for Python, Ruby, Java, Node.js...; http://developer.openstack.org/

1.2. OpenStack: ways of interaction

OpenStack: Interacting with OpenStack

At least, four different ways of interaction with OpenStack are available:

4. libcloud Apache: https://libcloud.apache.org/



1.2. OpenStack: ways of interaction

OpenStack: Interacting with OpenStack

At least, four different ways of interaction with OpenStack are available:

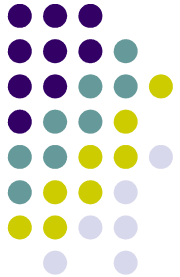
1. Dashboard; https://iaas.ceta-ciemat.es/dashboard/

2. CLI; shell, bash...; http://docs.openstack.org/cli-reference/

3. SDKs for Python, Ruby, Java, Node.js...; http://developer.openstack.org/

4. libcloud Apache: https://libcloud.apache.org/

1.2. OpenStack: ways of interaction



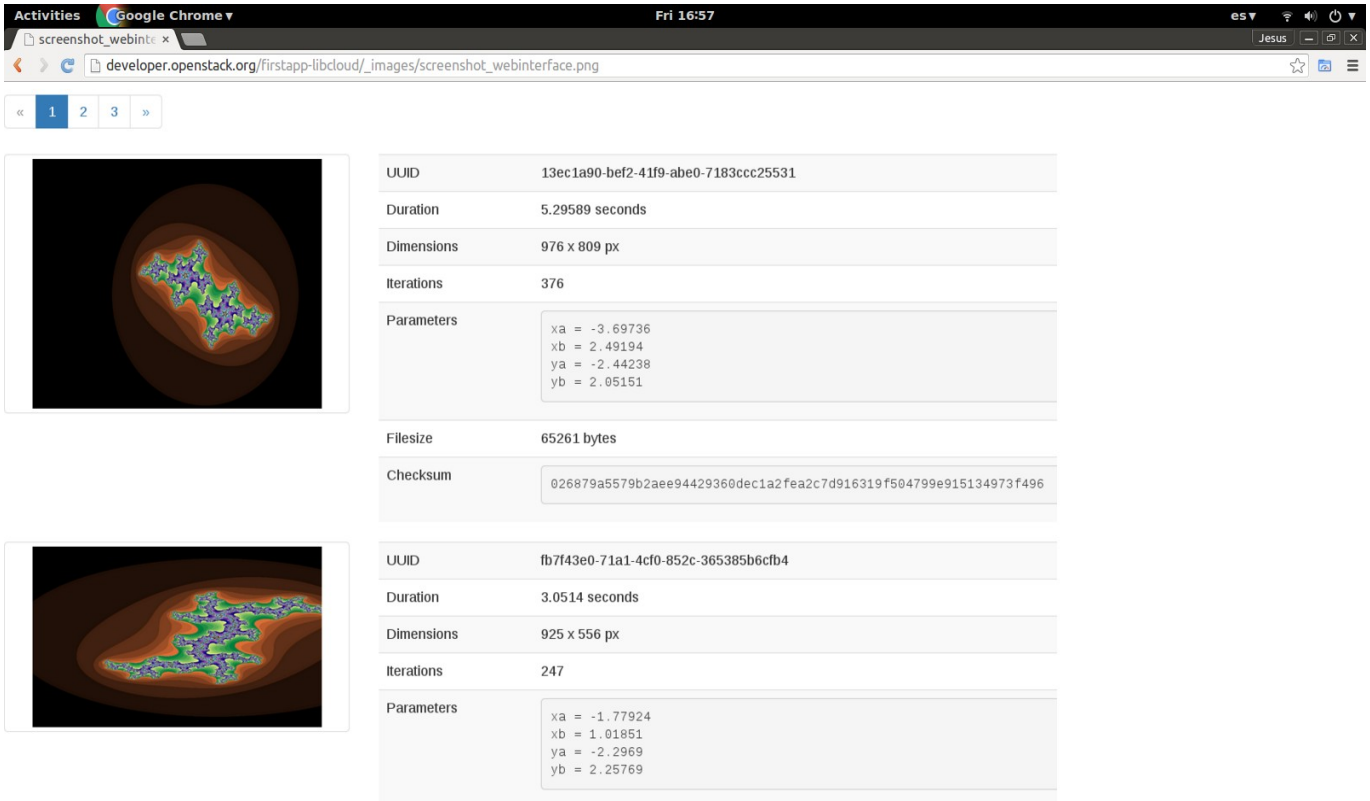
Basically, all these forms of interaction are based on a REST API

https://wiki.openstack.org/wiki/OpenStackRESTAPI
(compatible with Rackspace API: http://api.rackspace.com/)

http://developer.openstack.org/api-ref.html (see some url examples in http://developer.openstack.org/api-ref-compute-v2.1.html)

1.3. OpenStack: deploying a microservices infrastructure

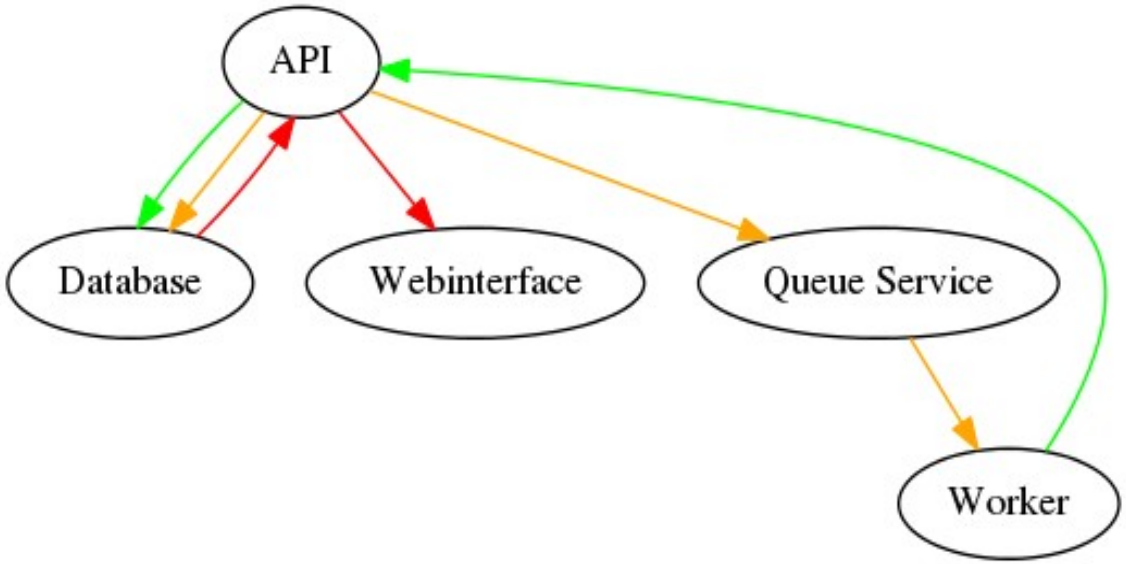
faafo: First App Application for OpenStack



Julia Sets: https://github.com/openstack/faafo/blob/master/faafo/worker/service.py
https://github.com/openstack/faafo/blob/master/doc/source/references.rst

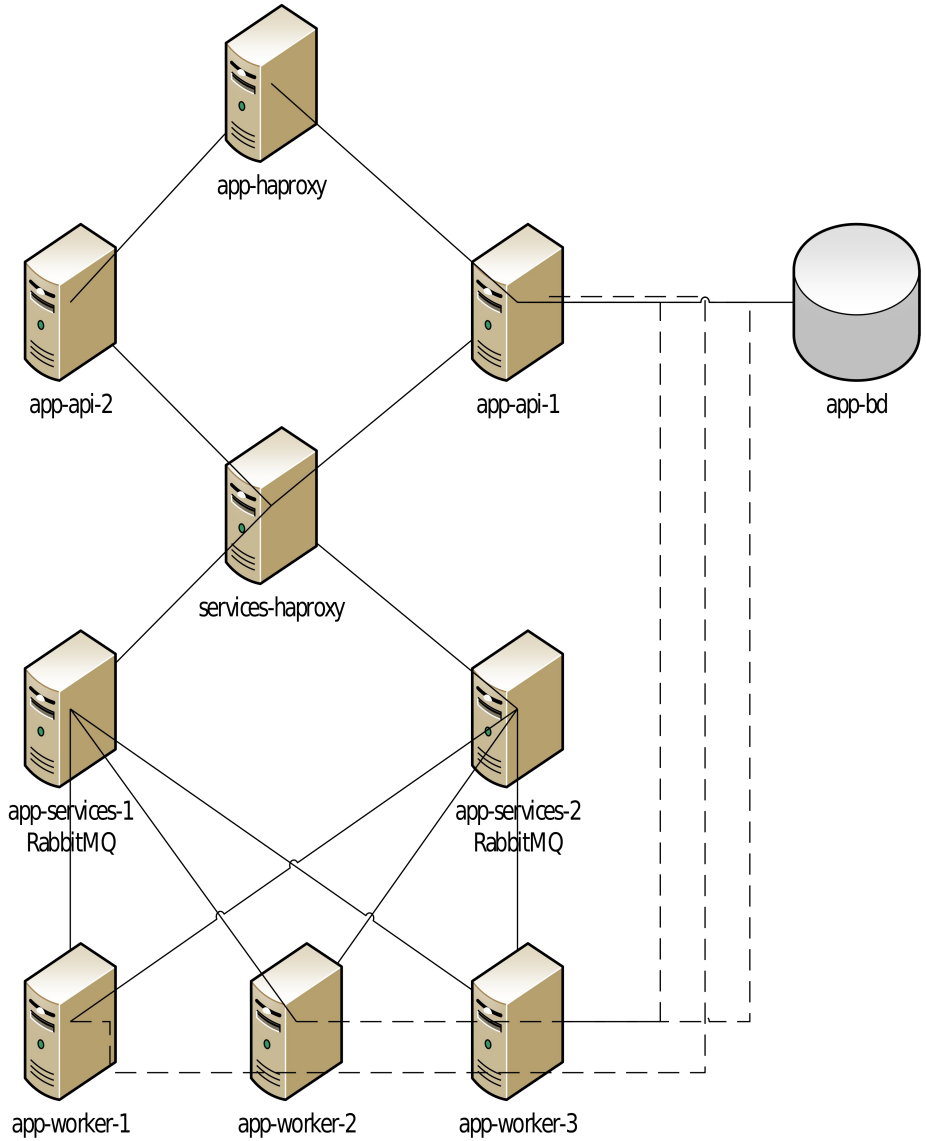
1.3. OpenStack: deploying a microservices infrastructure

OpenStack: Application infrastructure



http://developer.openstack.org/firstapp-libcloud/

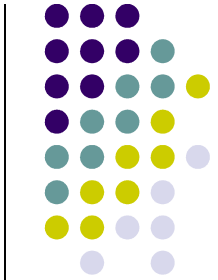
1.3. OpenStack: deploying a microservices infrastructure



https://github.com/openstack/faafo/blob/master/contrib/install.sh

https://git.openstack.org/cgit/openstack/faafo/commit/

1.3. OpenStack: deploying a microservices infrastructure



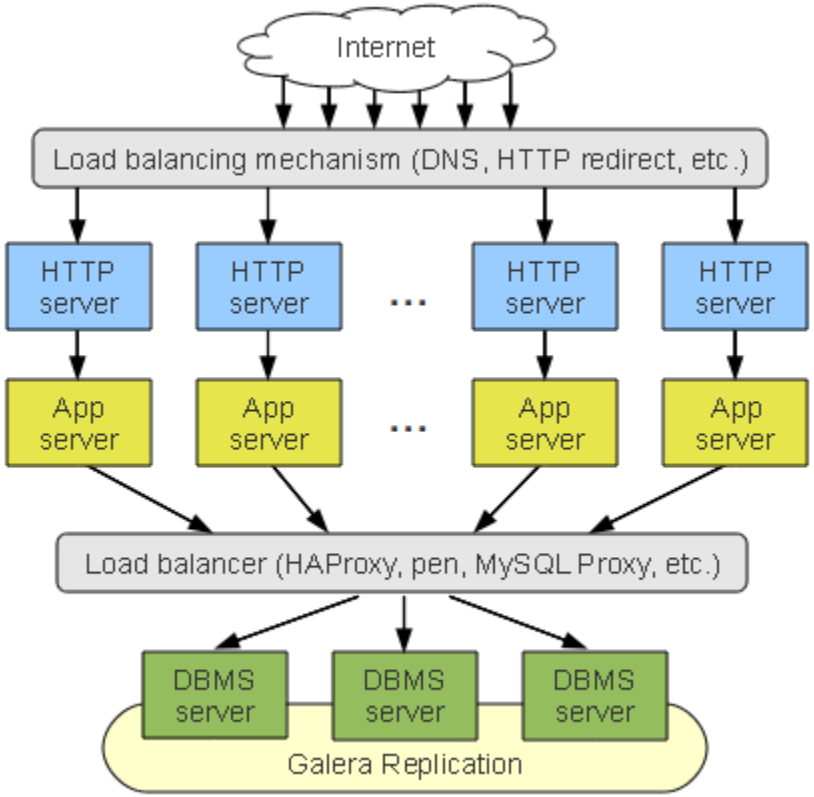
- Instalation script for every microservice
- https://github.com/openstack/faafo/blob/master/contrib/install.sh

- Python source code of the example
- https://git.openstack.org/cgit/openstack/faafo/commit/

- Python script of the whole infrastructure deployment (by @CarlosTiradoG)
- https://gist.github.com/catirado/ecad1c28275fb87033a7

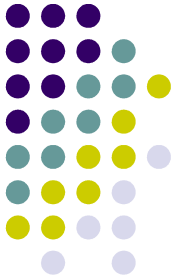
1.3. OpenStack: deploying a microservices infrastructure

A more familiar application architecture, including also load balancing



1.4. Deploying containers in the cloud: Docker

1.4. Deploying containers in the cloud: Docker

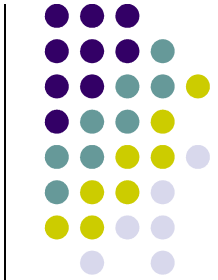


- in the previous example, every microservice was using a single **Virtual Machine**

- virtualisation environments impose an overhead in the host system, in terms of resources and costs

- **containers** are **operating-system-level virtualisation** environments

1.4. Deploying containers in the cloud: Docker



- **Differences** among containers and virtual machines

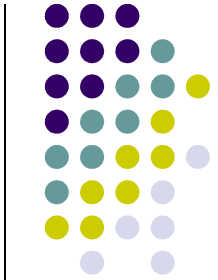
  - **Containers** are executed in **virtual partitions** using the **OS** calls

  - **Containers** use the **same OS** (at least the same kernel) as the **host machine**

  - Containers are more **lightweight** and **easier to distribute**, and therefore also to package applications

  - They are specially suited to run **multiple isolated applications** on a **single** (virtual) **machine**

1.4. Deploying containers in the cloud: Docker



- A **container** is a **group of processes** running on an operating system that are **isolated** from other such groups of processes

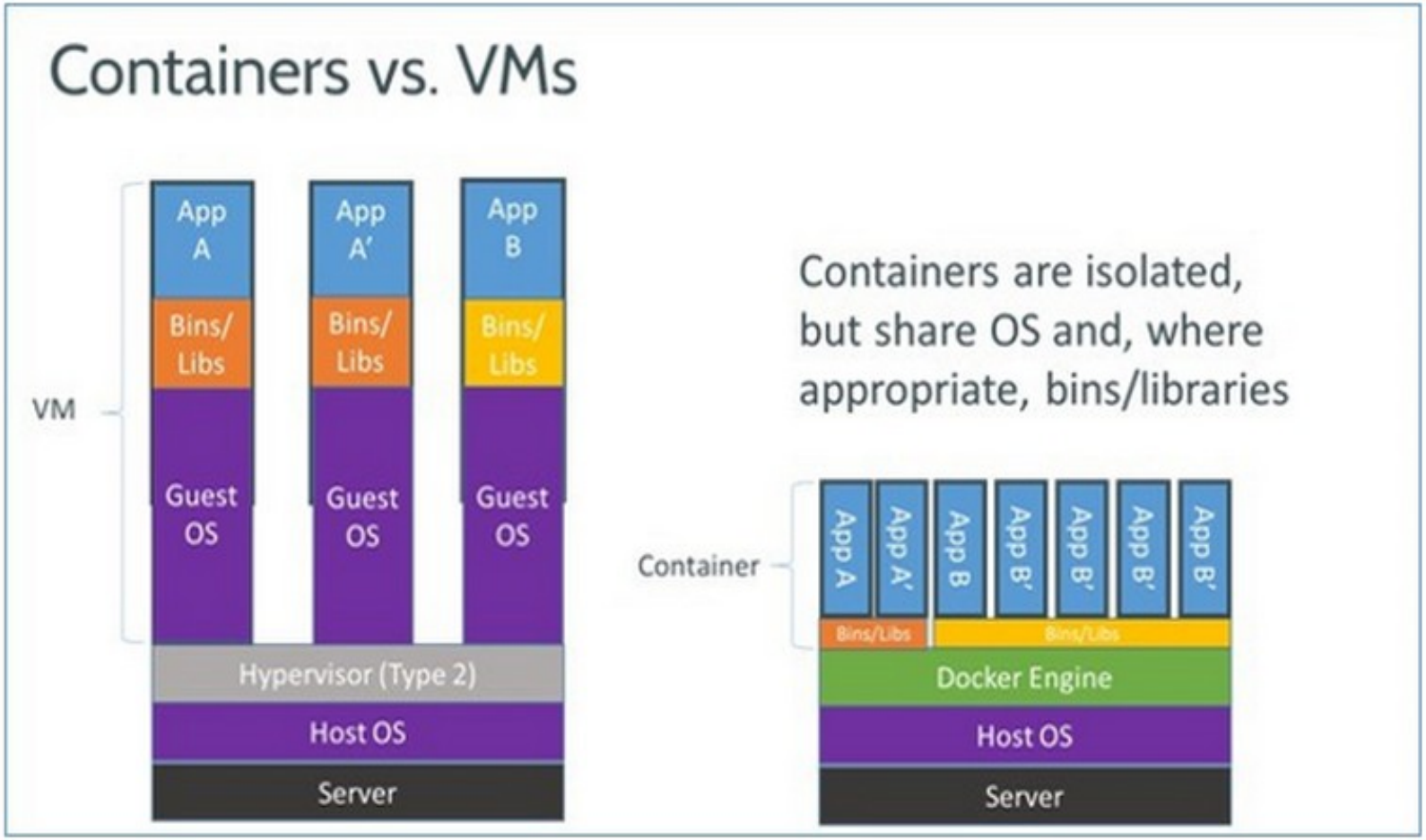
- There are several levels of **isolation** involved in containers

  - Solaris containers, called Zones, can be **allocated network interfaces** and network bandwidth regulated

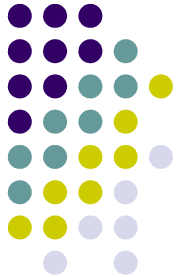
  - A container can **interact** (or kill) exclusively **processes in its container**

  - On the other hand, the **host machine** can see and manage every **process in every container**

1.4. Deploying containers in the cloud: Docker



1.4. Deploying containers in the cloud: Docker



Description of Docker. At the software level:

- `docker` is a single program

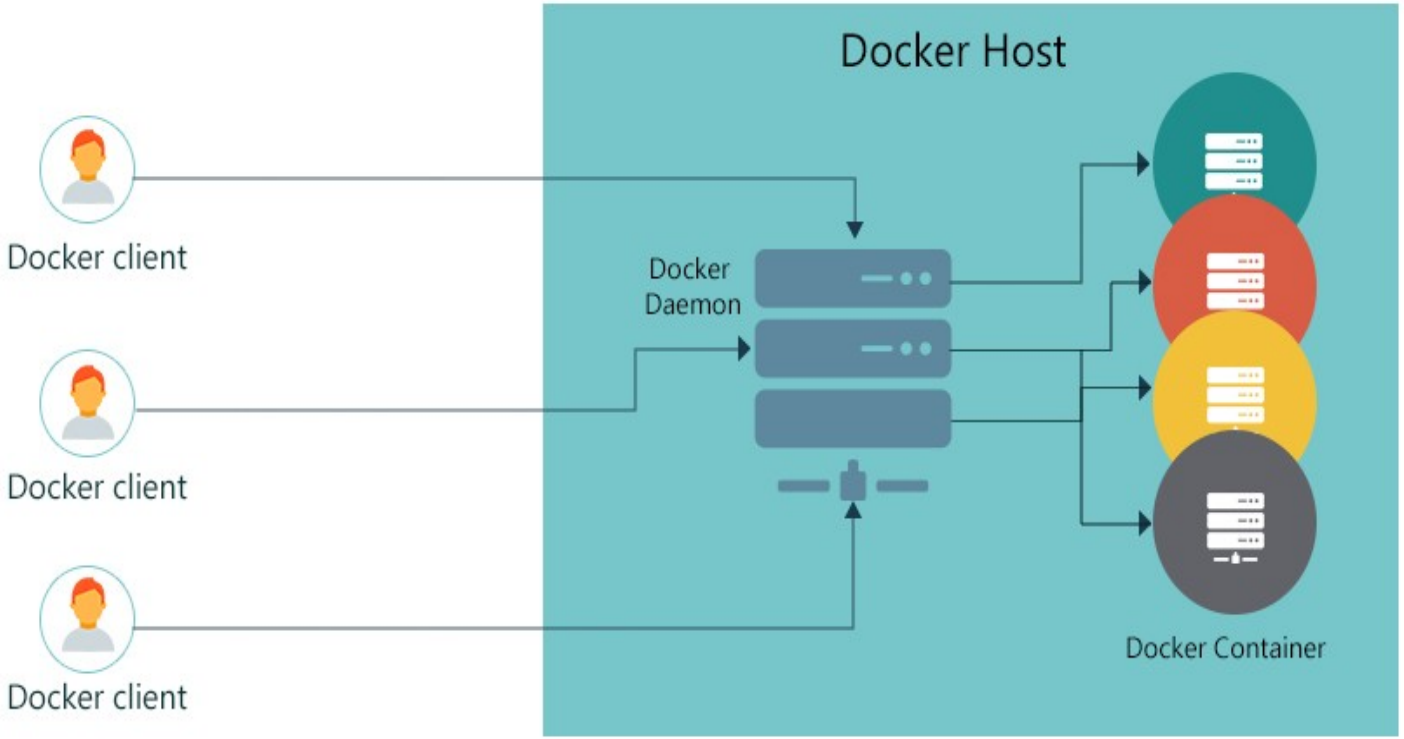
- Docker is a client/server architecture (Unix sockets or TCP ports, or both)

- the Docker daemon (`docker -d`) can run on any number of servers

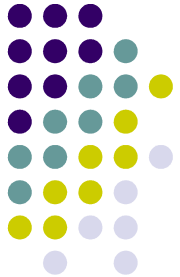
- a single client (`docker run`) can address any number of servers

- an additional piece of software, the registry, stores Docker images and metadata about those images

1.4. Deploying containers in the cloud: Docker



1.4. Deploying containers in the cloud: Docker



Description of Docker. At the network level:

- each container behaves as a **host** on a private network

- a Docker server behaves as a **virtual bridge**

- **containers are clients** behind the virtual bridge

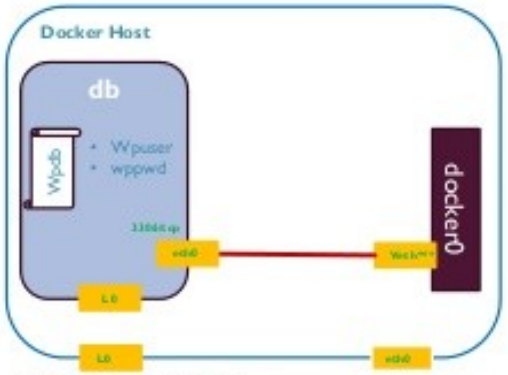
- each container has an own IP address, allocated to the virtual interface

- ports of the host can be bind to containers ports'

1.4. Deploying containers in the cloud: Docker

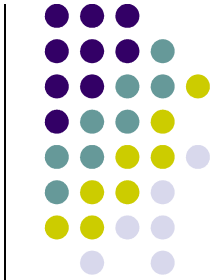
HOW DOCKER NETWORKS A CONTAINER ?

- docker run --name db -d -e MYSQL_ROOT_PASSWORD=Memzoh78 -e MYSQL_DATABASE=wpdb -e MYSQL_USER=wpuser -e MYSQL_PASSWORD=wppwd mysql



- option to ***docker run*** :
  - --net=bridge (default)
  - --net=host
  - --net=container:NAME_or_ID
  - --net=none

1.4. Deploying containers in the cloud: Docker

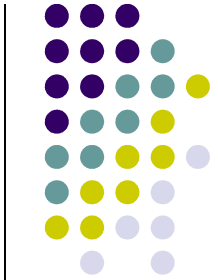


- Some **case uses** (or workflows...) where Containers are worth a try (1 / 3)

  - **PaaS** (Platform as a Service); because of their **ease of configuration** and **maintenance**, and their **low resource consumption**, they shape an ideal solution for Platform as a Service providers

https://www.quora.com/What-is-the-relationship-between-PaaS-and-containers-like-Docker

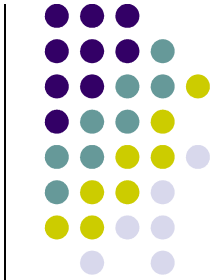
1.4. Deploying containers in the cloud: Docker



- Some **case uses** (or workflows...) where Containers are worth a try (2 / 3)

  - **Testing and developing environments**. Similar to **Virtual Machines**, but with lower resource demand and consumption, containers are well suited for testing and developing environments

1.4. Deploying containers in the cloud: Docker



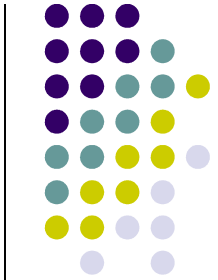
- Some **case uses** (or workflows...) where Containers are worth a try (3 / 3)

  - **Creating ephemeral machines**; for instance, it is trivial to create and provision a container for a "**one-use purpose**", such as **building an application for Jenkins** in an "ad-hoc" machine, and destroy it afterwards;

http://www.stackengine.com/implications-of-docker-ephemeral-compute/

Instead of "**daemons**", several tasks could be performed through **containers**

1.4. Deploying containers in the cloud: Docker



- **LXC** (2008, GNU GPL): https://linuxcontainers.org/

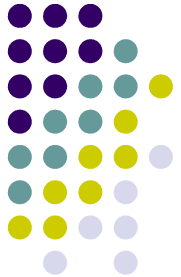
- **Solaris containers** (2004, Proprietary): http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html

- **Virtuozzo** (2000, Proprietary): http://www.virtuozzo.com/

- **Docker** (2013, Apache License 2.0): https://www.docker.com/

Comparison: https://www.flockport.com/lxc-vs-docker/

1.4. Deploying containers in the cloud: Docker



- **Docker** was first introduced in the Python Developers Conference (**March, 2013**); Solomon Hykes

- The project was **open-sourced** and made **available on GitHub**: https://github.com/docker/docker

- Docker promises:

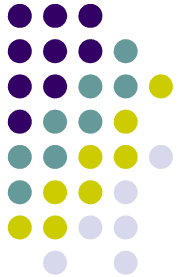
  - **Encapsulate** the process of creating a **distributable artifact** for any **application**

  - **Deploying** applications **at scale** into **any environment**

  - **Streamlining** the **workflow** of **agile** software organizations

  - **Easing** the DevOps **communication** and **transference** processes

1.4. Deploying containers in the cloud: Docker



Benefits of the **Docker workflow**

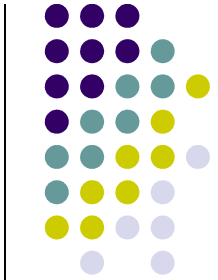
- **Packaging software** in a way that leverages the skills developers already have (simplifying or avoiding the need of **build engineers**)

- **Bundling application software** and required **OS filesystems** together in a single **standardised image** format

- Using **packaged artifacts** to **test and deliver** the exact **same artifact** to all systems in all environments

- **Abstracting software applications** from the **hardware** without sacrificing resources

1.4. Deploying containers in the cloud: Docker



Conclusions

- Systems like Docker define a **standardised container** for software

- A **container** can be **distributed** containing the software and everything needed for it to run, **instead of distributing software as a package**

- Being self-contained, **containers eliminate dependencies and conflicts**

- Containers are an **efficient way** to **provide** shared services, with the exact amount of resources (instead of virtual machines)