

ANSIBLE PARA DEV+OPS

DÍA 3 – PARTE 2

QUE VEREMOS HOY

- Aprovisionamiento con Vagrant
- Interacción con servicios cloud
- Interacción con docker
- Labs:
 - Interacción con AWS
 - Generación de contenedores mediante ansible

VAGRANT

- Herramienta para el despliegue rápido de máquinas virtuales basado en plantillas
- Plantillas basadas en VM precocinadas para
 - VirtualBox
 - Vmware
 - Parallels
- <http://vagrant.up>

VAGRANT – 1 SOLA VM

- vagrant init centos/7
- vagrant up

VAGRANTFILE

- Permite personalizar una imagen de vagrant
 - Provisioner para shell scripts, ansible, chef...
 - Carácterísticas de RAM y CPU
- Permite hacer despliegues de más de una VM
- Es un fichero ruby, que usa una DSL específica
- <https://www.vagrantup.com/docs/vagrantfile/>

VAGRANT - 3 VMS

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # Use the same key for each machine
  config.ssh.insert_key = false

  config.vm.define "vagrant1" do |vagrant1|
    vagrant1.vm.box = "ubuntu/trusty64"
    vagrant1.vm.network "forwarded_port", guest: 80, host: 8080
    vagrant1.vm.network "forwarded_port", guest: 443, host: 8443
  end
  config.vm.define "vagrant2" do |vagrant2|
    vagrant2.vm.box = "ubuntu/trusty64"
    vagrant2.vm.network "forwarded_port", guest: 80, host: 8081
    vagrant2.vm.network "forwarded_port", guest: 443, host: 8444
  end
  config.vm.define "vagrant3" do |vagrant3|
    vagrant3.vm.box = "ubuntu/trusty64"
    vagrant3.vm.network "forwarded_port", guest: 80, host: 8082
    vagrant3.vm.network "forwarded_port", guest: 443, host: 8445
  end
end
```

DEMO DE VAGRANT

- Probadlo en casa!

ANSIBLE + CLOUD SERVICES

- Cosas que veremos
 - Ansible y AWS
 - Ansible y vSphere
- Conceptos a tener en cuenta
 - Ejecución en el control host (host: localhost)
 - Delegación de tareas al control host

ANSIBLE + AWS

- Ansible tiene una serie de módulos que permiten integrar AWS con Ansible para aprovisionamiento y configuración
- http://docs.ansible.com/ansible/latest/guide_aws.html
- Depende de boto (librería de python)
- Requieren clave de API
- No guardan idempotencia en un fichero, si no que comprueban a cada ciclo
 - Tiempo de ejecución alto
 - Algunos componentes (como los security groups) no tienen bien resuelta la idempotencia

ANSIBLE + AWS: CREAR INSTANCIAS EC2

```
- name: Provision a set of instances
  ec2:
    key_name: my_key
    group: test
    instance_type: t2.micro
    image: "{{ ami_id }}"
    wait: true
    exact_count: 5
    count_tag:
      Name: Demo
    instance_tags:
      Name: Demo
  register: ec2
```

ANSIBLE + AWS: CREAR INSTANCIA EC2

```
- name: Provision a set of instances
  ec2:
    key_name: my_key
    group: test
    instance_type: t2.micro
    image: "{{ ami_id }}"
    wait: true
    exact_count: 5
    count_tag:
      Name: Demo
    instance_tags:
      Name: Demo
  register: ec2
```

ANSIBLE + AWS: ANSIBLE VS CLOUDFORMATION

- CF mantiene idempotencia de forma más controlable que Ansible
- CF tiene una sintaxis más confusa que Ansible
- Si ya se tiene desarrollado la capa de infraestructura con CF, Ansible tiene un módulo para ejecutar scripts en CF

ANSIBLE + VSPHERE

- Módulo vsphere_guest
 - http://docs.ansible.com/ansible/latest/vsphere_guest_module.html
- Muy complejo
- Debe ejecutarse en un nodo con visibilidad directa al cluster de vsphere
- Depende de pvmomi(módulo de python)

DONDE EJECUTAR AWS/VSPHERE

- Generalmente se ejecutarán en el nodo de control
- Los Playbooks entonces deberán apuntar a host: localhost para todas las tareas
- También existe la posibilidad de que las tareas individuales relacionadas con infraestructura estén delegadas mediante la opción “delegate_to:”

ANSIBLE + DOCKER

- Entendiendo la arquitectura de Docker
- Administrar contenedores de Docker con Ansible
- Usar Ansible para crear imágenes de Docker
- Administrar imágenes de Docker con Ansible
- Recoger facts de contenedores Docker con Ansible

ENTENDIENDO LA ARQUITECTURA DE DOCKER

Development

Developer machine

IDE & Tools

Scripts



CI



Production/Test

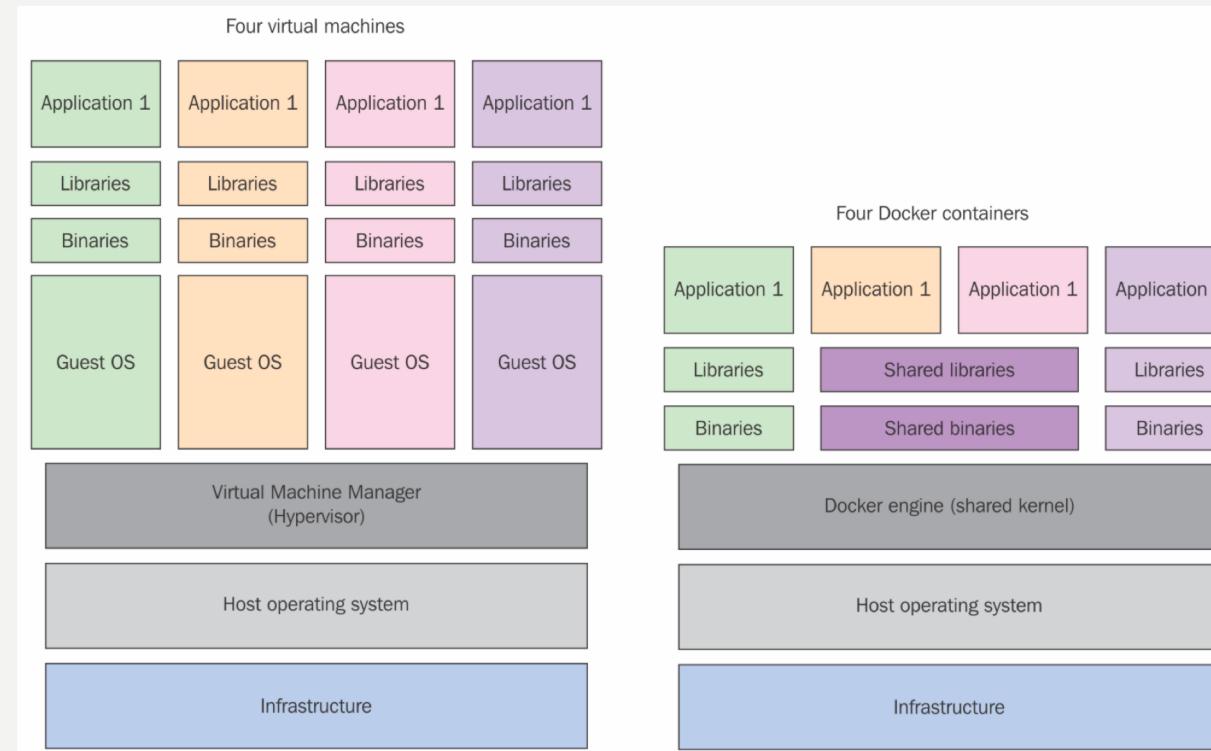


Developers pull and push source, rebuilding their containers to test changes locally, updating container definitions

CI responds to source changes, running builds and tests. Successful builds trigger pushing new versions of container images into container repository and push new version further into CI pipeline for deployment/verification

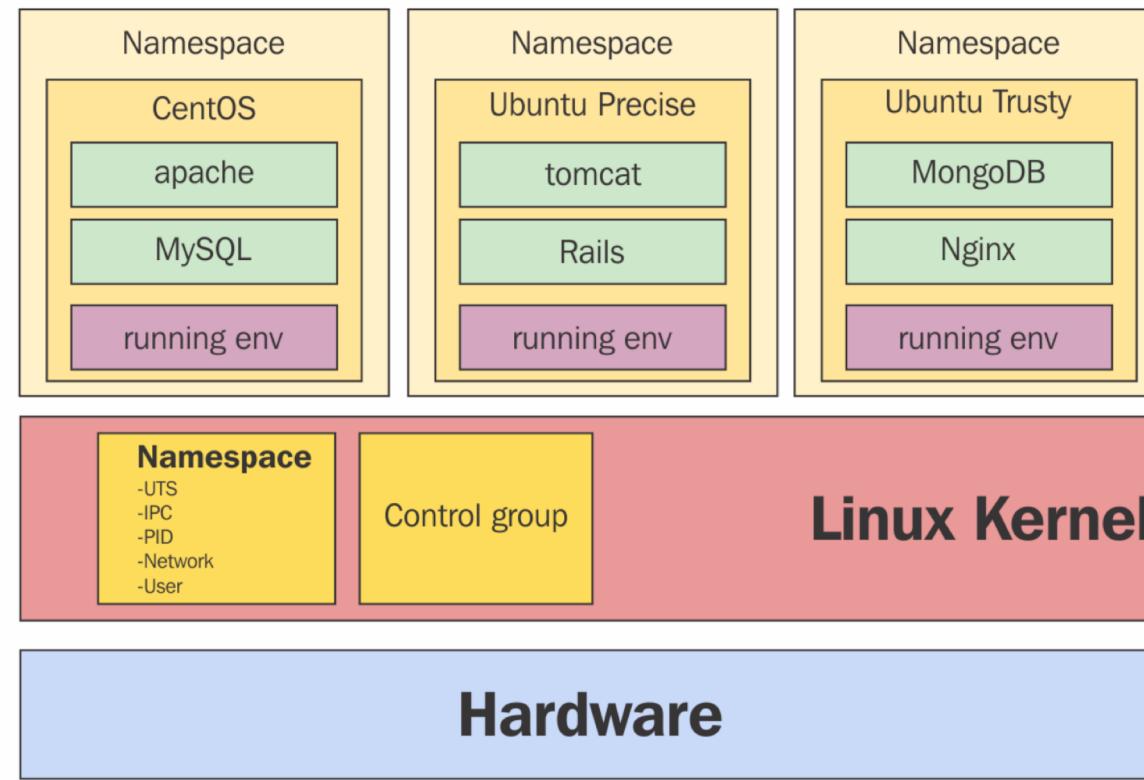
Production/Test environments destroy/recreate containers with the latest appropriate version pulled from container repository.

ENTENDIENDO LA ARQUITECTURA DE DOCKER



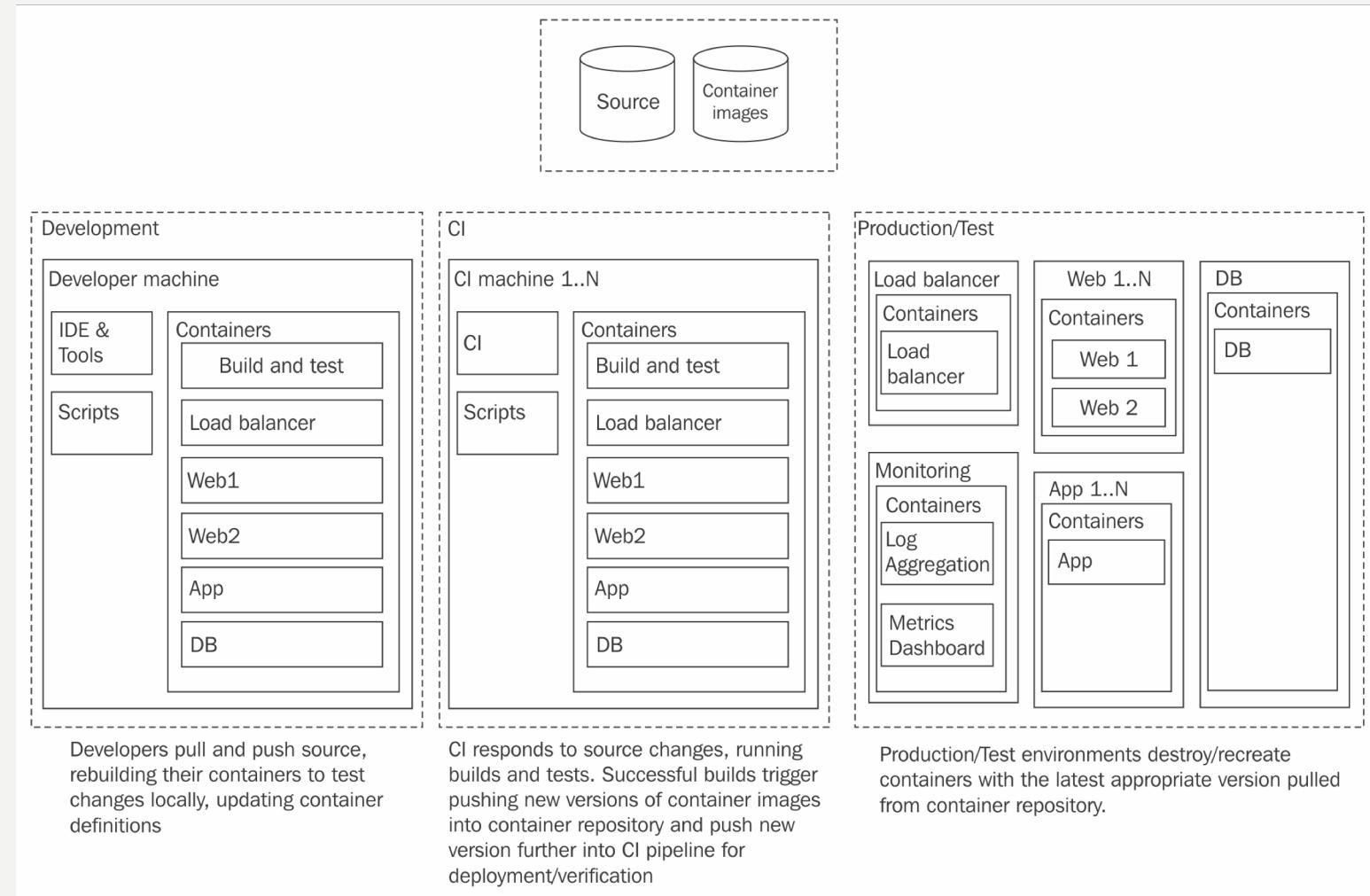
ENTENDIENDO LA ARQUITECTURA DE DOCKER

Linux Container - aka LXC



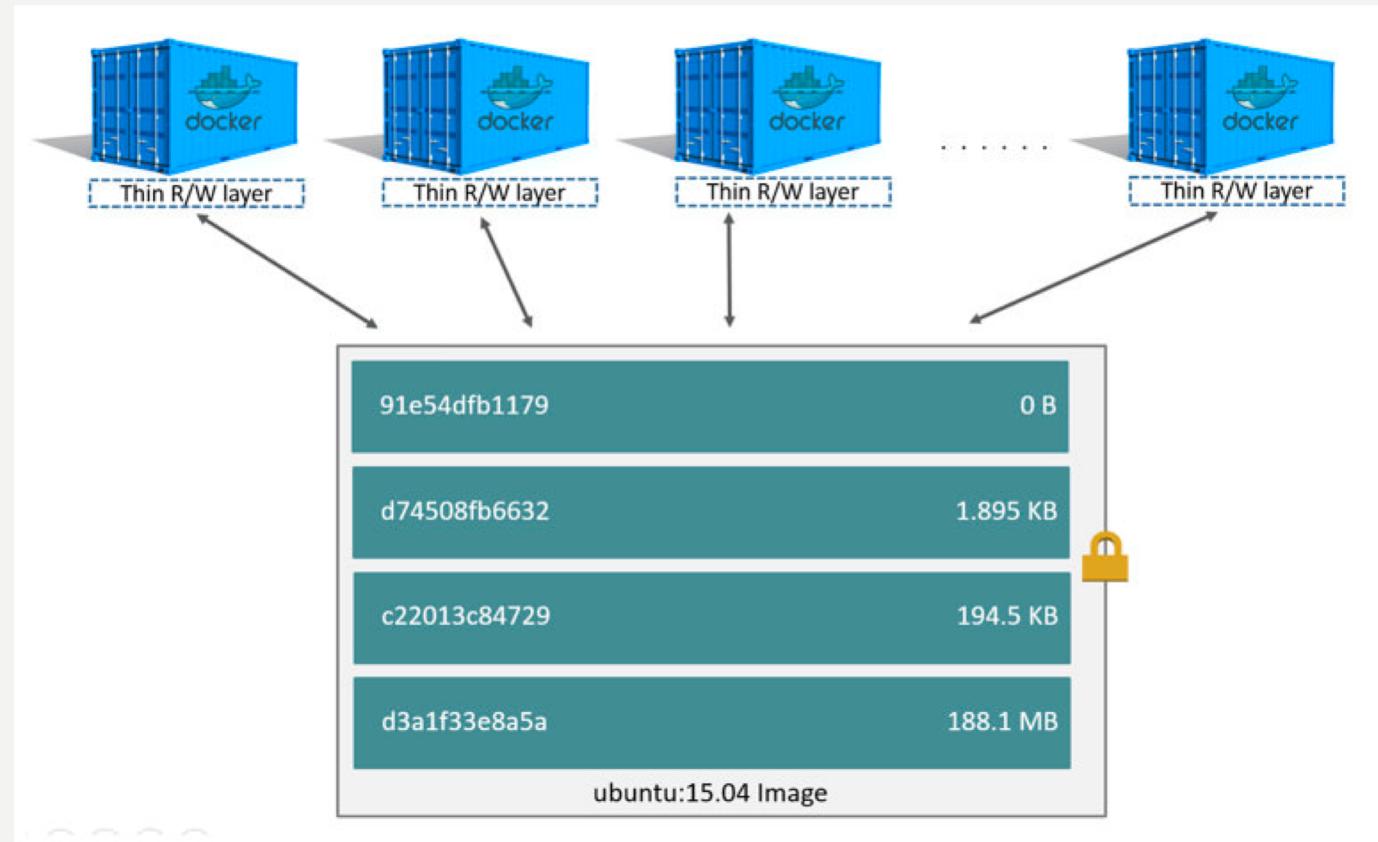
ENTENDIENDO LA ARQUITECTURA DE DOCKER

Múltiples contenedores para múltiples entornos



LAYERED FS

- Docker utiliza una serie de capas para crear una imagen
- Cada capa representa un paso en la creación de la imagen
- La RW de un container en ejecución se escribe en una nueva capa



LAYERED FS

- La compatibilidad depende de la versión de Linux y su fabricante

Linux distribution	Recommended storage drivers
Docker CE on Ubuntu	aufs , devicemapper , overlay2 (Ubuntu 14.04.4 or later, 16.04 or later), overlay , zfs , vfs
Docker CE on Debian	aufs , devicemapper , overlay2 (Debian Stretch), overlay , vfs
Docker CE on CentOS	devicemapper , vfs
Docker CE on Fedora	devicemapper , overlay2 (Fedora 26 or later, experimental), overlay (experimental), vfs

DOCKERFILE

- Define como crear una imagen de contenedor
- FROM -> IMAGEN
- RUN -> Comandos para preparar
- ADD -> Copiar algo dentro
- VOLUME -> Montar volumen del host
- CMD -> Comando por defecto

```
## Pull base image.
FROM dockerfile/ubuntu

# Install Nginx.
RUN \
    add-apt-repository -y ppa:nginx/stable && \
    apt-get update && \
    apt-get install -y nginx && \
    rm -rf /var/lib/apt/lists/* && \
    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
    chown -R www-data:www-data /var/lib/nginx

# Define mountable directories.
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/log/nginx", "/var/www/html"]

# Define working directory.
WORKDIR /etc/nginx

# Define default command.
CMD ["nginx"]

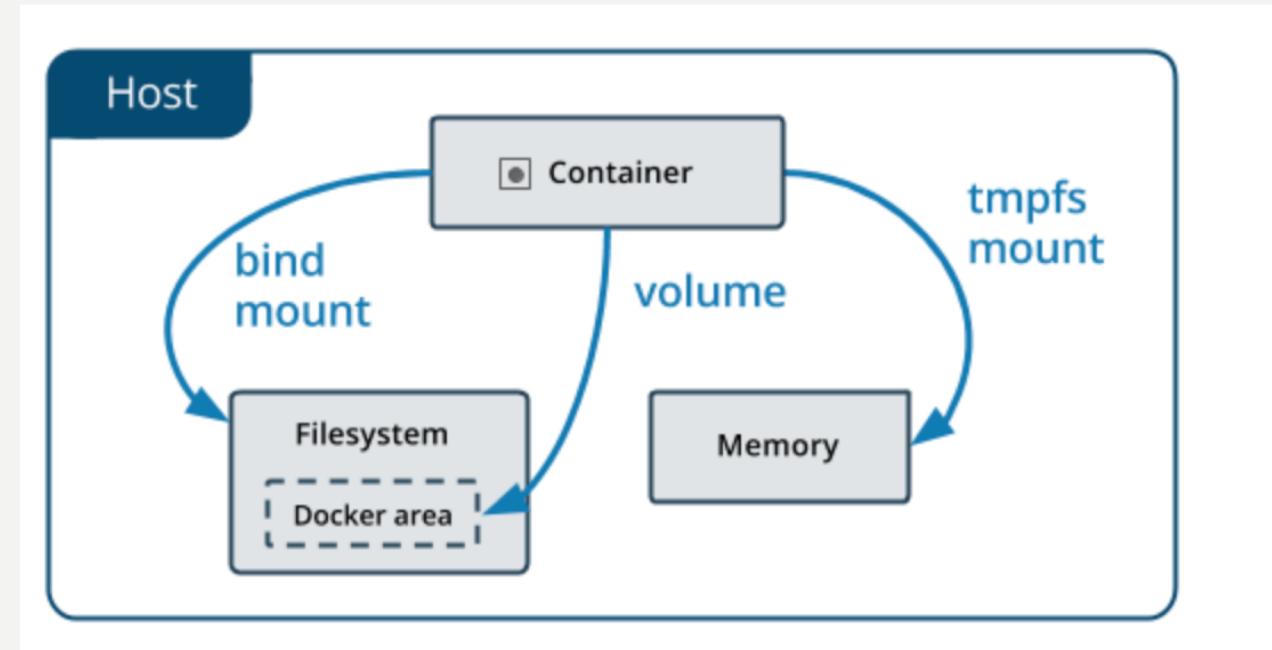
# Expose ports.
EXPOSE 80
EXPOSE 443
```

DOCKER REPOSITORY

- Permite almacenar imágenes de docker ya compiladas
- Vía red
- Existe el Docker Hub -> Repositorio Público
- La imagen de repository permite montar repositorios privados

DOCKER VOLUMES

- Contenedores de almacenamiento
- Para actividades intensivas de escritura / persistencia
- Se puede montar en varios contenedores a la vez



DOCKER COMPOSE

- Permite generar despliegues complejos con elementos relacionados
- Los nombres de cada contenedor son resolvibles desde todos los elementos
- Permite crear entornos enteros de forma descriptiva

ORQUESTRADORES DE CONTAINERS

- Mesos/Marathon/DC OS -> el más veterano , utiliza varias tecnologías opensource para orquestar procesos, entre ellos contenedores
- Kubernetes -> de google, descendiente de Borg el sistema de administración de LXC interno de google
- Docker Swarm -> el producto de Docker, muy bueno para desarrolladores pero bastante problemas de rendimiento como para usarlo en producción

ADMINISTRAR CONTENEDORES DE DOCKER CON ANSIBLE

Development

Developer machine

IDE & Tools

Containers

Build and test

Scripts

Load balancer

Web1

Web2

CI

CI machine 1..N

Scripts

Load balancer

Web1

Web2

Production/Test

Load balancer

Web 1..N

Web 1

Web 2

DB

Containers

DB

Developers pull and push source, rebuilding their containers to test changes locally, updating container definitions

CI responds to source changes, running builds and tests. Successful builds trigger pushing new versions of container images into container repository and push new version further into CI pipeline for deployment/verification

Production/Test environments destroy/recreate containers with the latest appropriate version pulled from container repository.

CREANDO CONTENEDORES

```
- name: Create a data container
  docker_container:
    name: mydata
    image: busybox
    volumes:
      - /data
```

DESTRUYENDO CONTENEDORES

```
- name: Remove MYSQL container
  docker_container:
    name: mysql
    state: absent
```

CREANDO Y PARANDO CONTENEDORES

- El parámetro state permite crear, eliminar, parar y arrancar contenedores

```
# The following task launches a mysql docker container
- name: MySQL Database Container Launch
  docker:
    name: database
    image: mysql:1.0
    state: started
```

Development

Developer machine

IDE & Tools

Scripts

Container
Builder

Build/test

Load balancer

Web1

Web2

A

D

CI

CI machine

Container
Builder

Build/test

Scripts

Load balancer

Web1

Web2

A

D

Production/Test

Load balancer

Container

Balance

Container

Web 1

Web 2

App 1..N

Containers

Log

Aggregation

App

App

DB

Containers

DB

USANDO ANSIBLE PARA CREAR IMÁGENES DOCKER

Developers pull and push source, rebuilding their containers to test changes locally, updating container definitions

CI responds to source changes, running builds and tests. Successful builds trigger pushing new versions of container images into container repository and push new version further into CI pipeline for deployment/verification

Production/Test environments destroy/recreate containers with the latest appropriate version pulled from container repository.

IMÁGENES DE DOCKER

- Se crean nativamente mediante Dockerfiles y el comando “docker build”
- Se puede aprovechar la capacidad del Dockerfile para lanzar una playbook de ansible dentro del contenedor:

```
# This DOCKERFILE creates a docker image with Ubuntu 14.04 and Ansible installed
# It also executes a playbook upon startup
FROM ansible/ubuntu14.04-ansible:stable

# This Defines the location for Ansible playbooks as /srv/example
ADD ansible /srv/example
WORKDIR /srv/example

# Execute Ansible with the playbook's primary entry point as myplaybook.yml
RUN ansible-playbook myplaybook.yml -c local
CMD [ "--help" ]
```

RECOGER FACTS DE DOCKER IMAGES

```
- name: Inspect a single Docker image
  docker_image_facts:
    name: foo/centos-7
```

LABS

- Pull de una imagen centos 7
- Crear nuestro primer Dockerfile
- Crear un contenedor que lance redis de forma “docker compliant”
- Subir la imagen al Docker Registry
- Que un compañero la baje
- Crear una imagen aprovechando el rol que tenemos de apache-webpage
- Exponer el puerto 80 como 8080 y acceder
- Instalar docker compose
- Lanzar la voting app (<https://github.com/dockersamples/example-voting-app>)