Despliegue de Aplicaciones: Taller de Docker y Vagrant

Evolución y Gestión de la Configuración

DOCKERIZANDO APLICACIONES

¿Qué queremos conseguir?

- Tener empaquetada nuestra aplicación y sus dependencias en una imagen para poder desplegarla donde queramos simplemente con

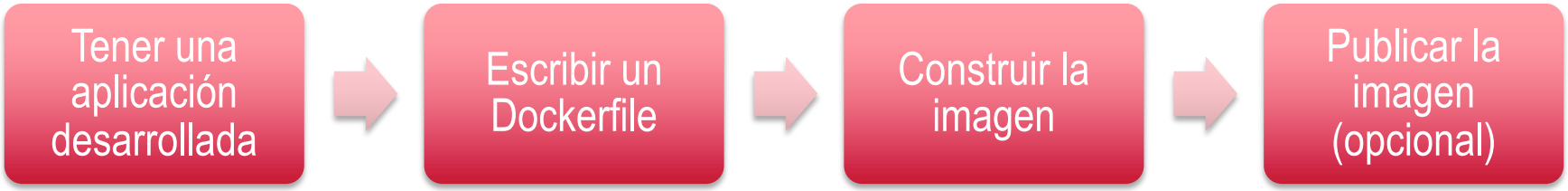
```
> docker run miAplicacion
```

- https://github.com/EGCETSII/1920-Practica-1

Imágenes de docker

- Una imagen es una colección de archivos
- Se parte de una imagen base y luego se construyen imágenes personalizadas encima
- Un Dockerfile o un Containerfile es un fichero que describe las instrucciones para construir una nueva imagen
- Las imágenes están en capas y cada capa representa un diff de la capa anterior

Pasos para Dockerizar una aplicación



Nuestra aplicación: Un “Hello world” hecho en python con el framework Flask

```
# Importamos el modulo de flask para poder usar ese framkework
from flask import Flask

# Constructor de Flask
app = Flask(__name__)

# En flask tenemos distintas rutas para distintas funciones
@app.route('/')

# ‘/’ está asociada a la función hello_world().
def hello_world():
        return 'Hello World’

# ‘/’hello/name está asociada a la función hello_name().
@app.route('/hello/<name>’)
def hello_name(name):
        return 'Hello %s!' % name

# Función principal
if __name__ == '__main__':
        app.run(host=“0.0.0.0”)
```

El Dockerfile

```
# Base image
FROM python:3

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY holamundo.py ./

CMD [ "python", "./holamundo.py" ]
```

Consejos para escribir Dockerfiles: https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

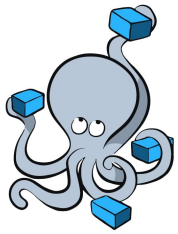
Construimos la imagen y la comprobamos

```
> docker build -t flaskhello -f
Containerfile .

> docker images


> docker run -it --rm -p 8020:80 flaskhello
```

EJECUTANDO DECIDE EN CONTENEDORES CON DOCKER COMPOSE

En Docker se recomienda seguir el principio de responsabilidad única:

1 Contenedor = 1 Servicio



docker Compose

Actualizar para la nueva versión de Decide

```
services:
  db:
    restart: always
    container_name: decide_db
    image: postgres:14.9-alpine
    volumes:
      - db:/var/lib/postgresql/data
    networks:
      - decide
    environment:
      - POSTGRES_PASSWORD=postgres
```

```
Django==4.1
pycryptodome==3.15.0
djangorestframework==3.14.0
django-cors-headers==3.13.0
requests==2.28.1
django-filter==22.1
psycopg2==2.9.4
coverage==6.5.0
jsonnet==0.18.0
django-nose==1.4.6
django-rest-swagger==2.2.0
selenium==4.7.2
pynose == 1.4.8
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD':'postgres',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

```
CSRF_TRUSTED_ORIGINS = ['http://10.5.0.1:8000', 'http://localhost:8000']
```

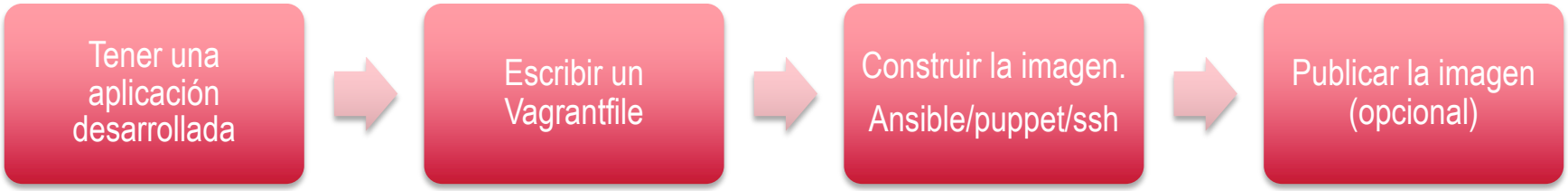
```
from python:3.10-alpine

RUN apk add --no-cache git postgresql-dev gcc libc-dev
RUN apk add --no-cache gcc g++ make libffi-dev python3-dev build-base

RUN pip install gunicorn
RUN pip install psycopg2
RUN pip install ipdb
RUN pip install ipython

WORKDIR /app

RUN git  NUESTRO_REPO .
RUN pip install -r requirements.txt

WORKDIR /app/decide

# local settings.py
ADD docker-settings.py /app/decide/local_settings.py

RUN ./manage.py collectstatic
```

VIRTUALIZANDO APLICACIONES

Imágenes en vagrant

- Una imagen un fichero de disco más un fichero de configuración
- Se parte de una imagen base y luego se construyen imágenes personalizadas encima
- Un Vagrant file define las opciones de arranque de la máquina
- Vagrant no se encarga del aprovisionamiento (instalación de apps y dependencias)

Pasos para VMizar una aplicación



Nuestra aplicación: Un “Hello world” hecho en python con el framework Flask

```
# Importamos el modulo de flask para poder usar ese framkework
from flask import Flask

# Constructor de Flask
app = Flask(__name__)

# En flask tenemos distintas rutas para distintas funciones
@app.route('/')

# ‘/’ está asociada a la función hello_world().
def hello_world():
        return 'Hello World’

# ‘/’hello/name está asociada a la función hello_name().
@app.route('/hello/<name>’)
def hello_name(name):
        return 'Hello %s!' % name

# Función principal
if __name__ == '__main__':
        app.run()
```

El Vagrantfile

```
Vagrant.configure("2") do |config|
        config.vm.box = "ubuntu/bionic64“
        config.vm.network "forwarded_port", guest: 80, host: 8080
        config.vm.provision "shell", path: 'provision.sh’
end
```

El aprovisionamiento

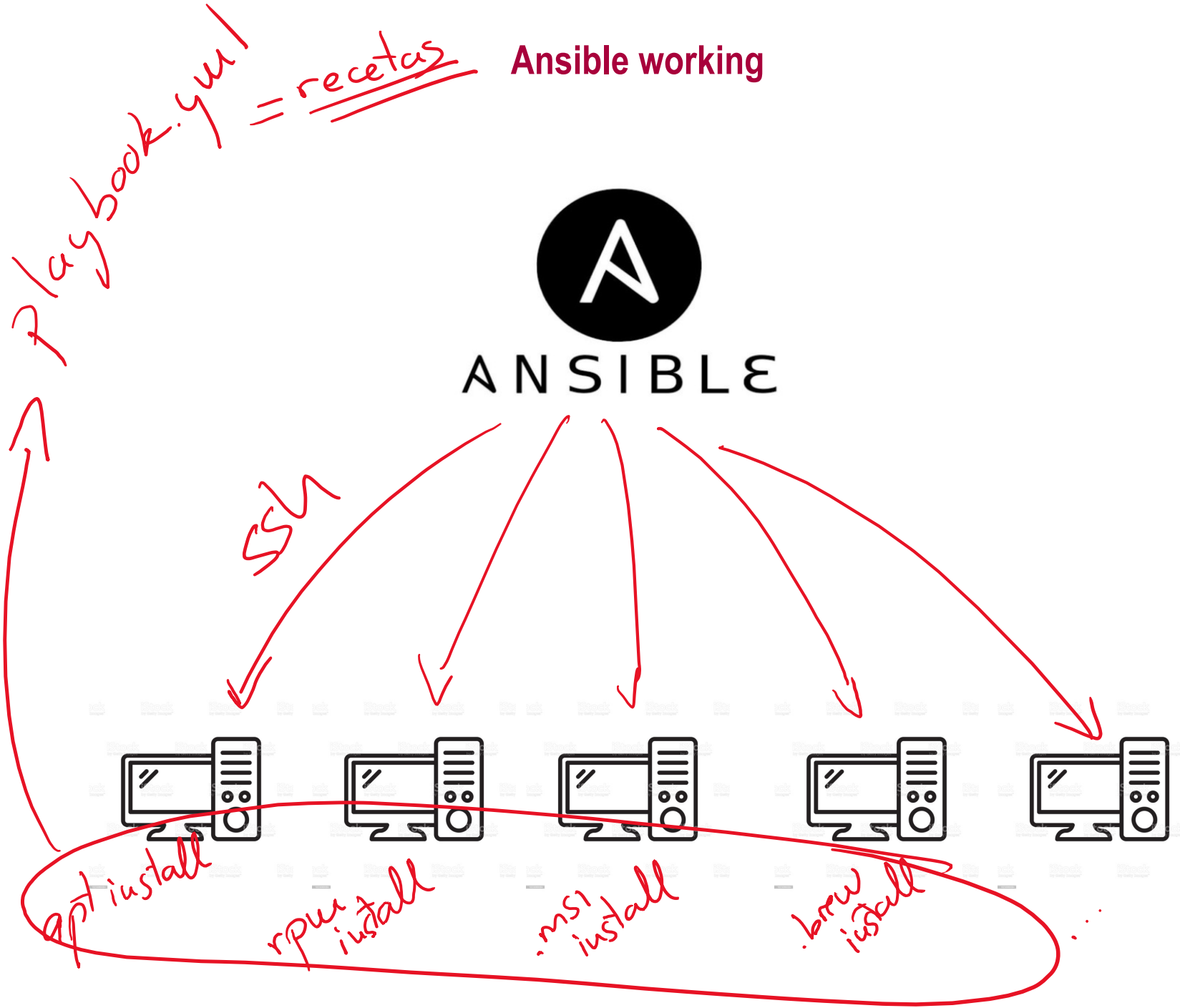
```
sudo apt update
sudo apt upgrade –y
sudo apt install -y  git python3 python3-pip screen
git clone https://github.com/EGCETSII/1920-Practica-1.git
cd 1920-Practica-1
pip3 install -r requirements.txt
screen -m -d python3 holamundo.py
```

EJECUTANDO DECIDE EN VAGRANT CON ANSIBLE

Ansible

Ansible is quite often called “a loop for ssh”. It is a bit an oversimplification, however – yes it allows you to loop over your multiple hosts (physical or virtual) and apply changes.

Ansible working

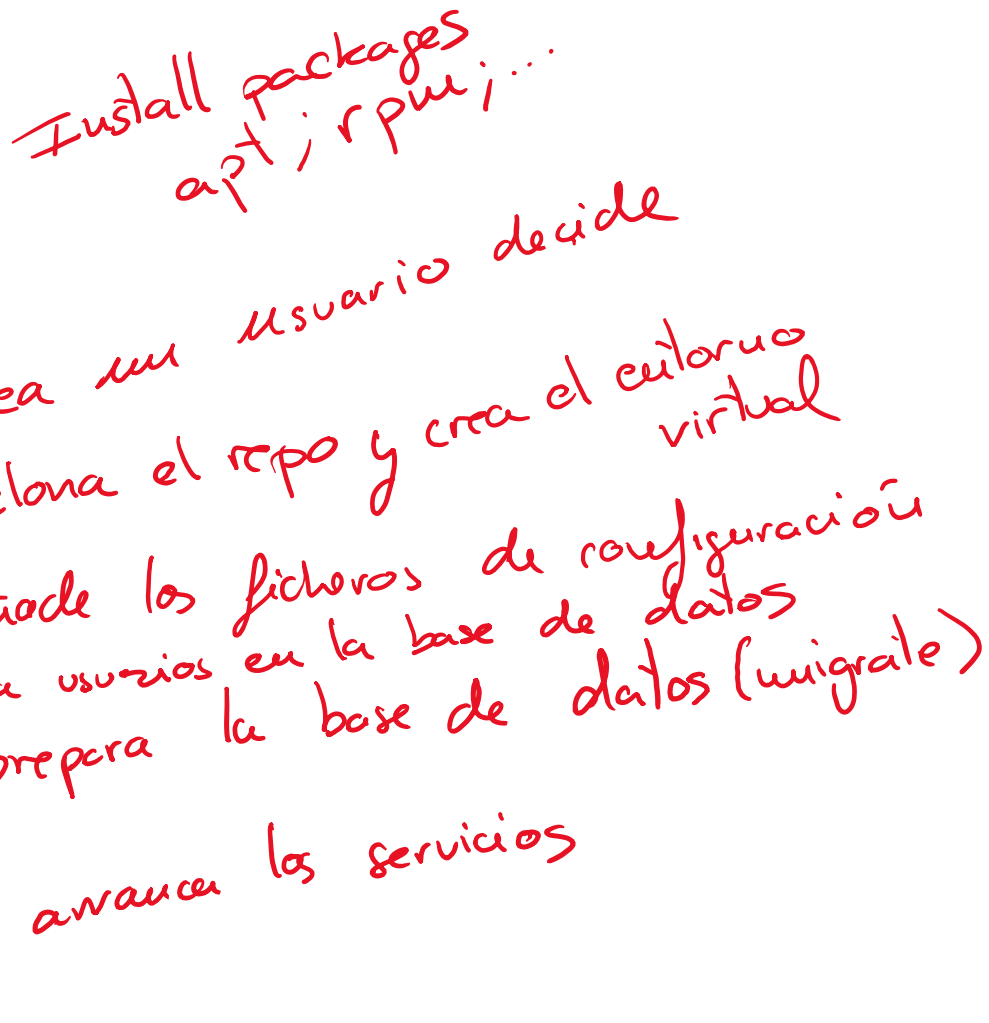


DECIDE ON VAGRANT

- What we do need to run decide?
  - Python
  - Webserver
  - Postgres
- How to provision this?
  - Ssh?
    - Sudo apt install python3-pip postgressql …
    - Pip install …
    - Etc etc
    - But wat about if we do run this on alpine instead of Ubuntu? And if we are not on a Debian system?
  - Ansible to the rescue

The playbook

```yaml
---
- hosts: all

  tasks:
    - include: packages.yml
      tags: ["packages"]
    - include: user.yml
    - include: python.yml
      tags: ["app"]
    - include: files.yml
      tags: ["files"]
    - include: database.yml
      tags: ["database"]
    - include: django.yml
      tags: ["django"]
    - include: services.yml
      tags: ["services"]
```



The vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/bionic64“
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.provider "virtualbox" do |v|
      v.memory = 512
      v.cpus = 1
  end

config.vm.provision "ansible" do |ansible|
    ansible.compatibility_mode = '2.0'
    ansible.playbook = "playbook.yml"
    ansible.extra_vars = { ansible_python_interpreter:"/usr/bin/python3" }
  end

end
```

Actualizar para la nueva versión de Decide

```
config.vm.box = "ubuntu/jammy64"

config.vm.network "forwarded_port", guest: 80, host: 8080

config.vm.provider "virtualbox" do |v|
    v.memory = 2048
    v.cpus = 3
end
```

```
- name: Git clone
  become: yes
  become_user: decide
  git:
    repo: 'NUESTRA_URL'
    dest: /home/decide/decide
    version: master
```

```
- name: Install packages
  become: true
  apt:
    name: "{{ packages }}"
    update_cache: yes
  vars:
    packages:
    - acl
    - git
    - postgresql
    - python3
    - python3-pip
    - python3-psycopg2
    - python3-virtualenv
    - virtualenv
    - nginx
    - libpq-dev
    - python-setuptools
    - build-essential
    - python3-dev
    - make
    - m4
```

```
ALLOWED_HOSTS = ['*']
ALLOWED_ORIGINS = ['http://*', 'https://*', 'https://localhost:8080', 'http://localhost:8080']
CSRF_COOKIE_SECURE = True
CSRF_COOKIE_SAMESITE = 'Strict'
CSRF_TRUSTED_ORIGINS = ALLOWED_ORIGINS.copy()
```

```
- name: Change permissions on decide (Make static directory tree readable)
  become: yes
  file:
    path: /home/decide
    owner: "decide"
    recurse: yes
    group: decide
    mode: "555"
```