

Procedimiento

Índice

1. [Instalación](#)
2. [AWS ami](#)
3. [Vagrant Box](#)
4. [Conclusiones](#)
5. [Bibliografía](#)

1. Instalación

La instalación es muy sencilla, solo hace falta añadir el repositorio y ya estaría listo para usar.

```
[vagrant@fedora32 ~]$ sudo dnf install -y dnf-plugins-core
Fedora Modular 32 - x86_64                59 kB/s | 25 kB      00:00
Fedora Modular 32 - x86_64 - Updates      115 kB/s | 23 kB      00:00
Fedora Modular 32 - x86_64 - Updates      211 kB/s | 331 kB     00:01
Fedora 32 - x86_64 - Updates              85 kB/s | 15 kB      00:00
Fedora 32 - x86_64 - Updates              1.2 MB/s | 5.1 MB     00:04
Fedora 32 - x86_64                        87 kB/s | 25 kB      00:00
Package dnf-plugins-core-4.0.18-1.fc32.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[vagrant@fedora32 ~]$ sudo dnf config-manager --add-repo https://rpm.releases.hashicorp.com/fedora/hashicorp.repo
Adding repo from: https://rpm.releases.hashicorp.com/fedora/hashicorp.repo
```

Una vez añadido el repositorio solo quedaría hacer un `dnf -y install packer`.

También podemos hacerlo descargando el binario:

```
[vagrant@fedora32 ~]$ wget https://releases.hashicorp.com/packer/1.7.2/packer_1.7.2_linux_amd64.zip
--2021-05-10 12:57:49-- https://releases.hashicorp.com/packer/1.7.2/packer_1.7.2_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.133.183, 2a04:4e42:1f::439
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.133.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28738303 (27M) [application/zip]
Saving to: 'packer_1.7.2_linux_amd64.zip'

packer_1.7.2_linux_amd64.zip  100%[=====>] 27.41M  32.6MB/s  in 0.8s

2021-05-10 12:57:50 (32.6 MB/s) - 'packer_1.7.2_linux_amd64.zip' saved [28738303/28738303]
[vagrant@fedora32 opt]$ sudo unzip packer_1.7.2_linux_amd64.zip
Archive:  packer_1.7.2_linux_amd64.zip
  inflating: packer
[vagrant@fedora32 opt]$ sudo ln -s /opt/packer /usr/bin/packer
[vagrant@fedora32 opt]$ packer -v
1.7.2
```

2. AWS ami

Esta ami customizada partirá de una ami base de *Ubuntu Server 20.04 LTS*, la cual tendrá apache2 y php instalado por medio del provisioner **"shell"** y se la pasará el fichero *index.html* por medio del provisioner **"file"**.

install.sh:

```
#!/bin/bash
sleep 30
sudo apt update
sudo apt -y install apache2
sudo apt -y install software-properties-common
sudo add-apt-repository ppa:ondrej/php
sudo apt -y install php7.4

sudo rm -f /var/www/html/index.html

sudo systemctl start apache2.service
```

index.php:

```
<?php
phpinfo();
?>
```

image.json:

```

{
  "builders": [
    {
      "type": "amazon-ecs",
      "profile": "{{ user `aws_profile` }}",
      "region": "eu-west-3",
      "ami_name": "diego_ami",
      "source_ami": "ami-0f7cd40eac2214b37",
      "instance_type": "t2.micro",
      "ssh_username": "ubuntu"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "script": "install.sh"
    },
    {
      "type": "file",
      "source": "index.php",
      "destination": "/tmp/"
    },
    {
      "type": "shell",
      "inline": ["sudo cp /tmp/index.php /var/www/html/"]
    }
  ]
}

```

- **Builders:**

- Primero le indicamos que **Builder** queremos usar, en este caso el de **AWS** concretamente para crear amis para instancias EC2.
- El profile se lo pasaremos como argumento por tal de aprovechar el CLI de AWS donde tenemos almacenados nuestras **access key** y **secret key**.
- Le indicamos también la región donde usaremos **AWS**.
- El nombre que tendrá la ami que crearemos.
- Es necesario que le indiquemos una **source_ami** ya que lo que hará **Packer** será copiar una ami base de referencia y en esta copia aplicar los cambios que hayamos especificado.
- También es necesario indicar el **instance_type** ya que **Packer** para que pueda crear la imagen customizada primero tiene que lanzar una instancia y en esa instancia aplicar todas las configuraciones que hayamos especificado, luego destruirá esta instancia y generará la imagen.
- El **ssh_username** es para que **Packer** al momento de crear la instancia se pueda conectar a esta, en este caso es *"ubuntu"* ya que estamos usando la ami de un S.O. ubuntu y estos en **AWS** cuentan con el usuario ubuntu para poder conectarse.

- **provisioners:**

- Usaremos dos tipos de provisioners:
 - **shell:** Este provisioner nos permite ejecutar comandos de shell, en el cual le pasaremos un fichero `.sh` el cual contiene la instalación de apache2 y php además de una pequeña configuración.
 - **file:** Es el cual nos permite tranferir ficheros de la máquina host al host remoto donde se lanzará la instancia, en este caso en concreto le estamos pasando un fichero de nombre `"index.php"` que contiene una función que muestra información sobre el módulo php `phpinfo()` .

Cabe resaltar que primero tranferimos el fichero `index.php` a `/tmp/` ya que el provisioner `file` no cuenta con permisos de root ni está en el grupo de sudoers y por lo tanto no tiene persmiso de tranferir directamente el fichero a `/var/www/html/`. Entonces lo que hemos hecho es primero pasarlo a `/tmp/` dónde todo el mundo tiene permisos tanto de escritura como de lectura y luego con comandos de shell lo copiamos a `/var/www/html`

3. Vagrant Box

Este box customizado partirá de un box con una imagen de [ubuntu/xenial64](#) que tendrá instalados y configurados apache, php y ldap.

[install.sh](#):

```
#!/bin/bash
sudo sleep 30
# Update Packages
sudo apt-get update
# Upgrade Packages
sudo apt-get upgrade

# Basic Linux Stuff
sudo apt-get install -y git

# Apache
sudo apt-get install -y apache2

# Enable Apache Mods
sudo a2enmod rewrite

#Add Ondrej PPA Repo
sudo apt-add-repository ppa:ondrej/php -y
sudo apt-get update

# Install PHP
sudo apt-get install -y php7.2

# PHP Apache Mod
sudo apt-get install -y libapache2-mod-php7.2

# Restart Apache
sudo service apache2 restart

# PHP Mods
sudo apt-get install -y php7.2-common
sudo apt-get install -y php7.2-mcrypt
sudo apt-get install -y php7.2-zip

# set slapd pass
sudo debconf-set-selections <<< 'slapd slapd/root_password password jupiter'
sudo debconf-set-selections <<< 'slapd slapd/root_password_again password jupiter'

# install ldap moduls
sudo DEBIAN_FRONTEND=noninteractive apt-get -y install slapd
sudo apt-get install -y ldap-utils

# configure ldap
sudo rm -rf /etc/ldap/slapd.d/*
sudo rm -rf /var/lib/ldap/*
sudo cp /home/vagrant/ldap/DB_CONFIG /var/lib/ldap/.
sudo slaptest -Q -f /home/vagrant/ldap/slapd.conf -F /etc/ldap/slapd.d &> /dev/null
sudo slapadd -F /etc/ldap/slapd.d -l /home/vagrant/ldap/edt.org.ldif
sudo chown -R openldap.openldap /etc/ldap/slapd.d
sudo chown -R openldap.openldap /var/lib/ldap
sudo cp /home/vagrant/ldap/ldap.conf /etc/ldap/ldap.conf
```

```
sudo service slapd start
sudo service slapd restart
```

index.php:

```
<?php
phpinfo();
?>
```

image.json:

```
{
  "builders": [
    {
      "communicator": "ssh",
      "source_path": "ubuntu/xenial64",
      "provider": "virtualbox",
      "type": "vagrant"
    }
  ],
  "provisioners": [
    {
      "type": "file",
      "source": "ldap",
      "destination": "~/",
    },
    {
      "type": "file",
      "source": "index.php",
      "destination": "~/",
    },
    {
      "type": "shell",
      "script": "install.sh"
    }
  ]
}
```

Lo que hará **Packer** será lanzar un box base, en este caso ubuntu/xenial64 que, concretamente, contiene la imagen ubuntu 16.04 LTS, y conectarse por ssh.

Una vez lanzada empieza a hacer las configuraciones de los diferentes **provisioners** que hayamos especificado, luego que se haya configurado e/o instalado todo correctamente crea la imagen y la empaqueta en un **box**.

Cuando haya acabado todo este proceso hace un `vagrant destroy` del box base.

- **builders:**

- communicator: Como se comunicará packer con el box de vagrant.
- source_path: La ruta en la cual obtiene el box (puede ser de la Vagrant Cloud, de una URL, o de un box local).
- provider: Que provider en concreto usará el box para poder ser lanzado, en este caso virtualbox.
- type: El tipo de imagen, en este caso box por ser de **Vagrant**.

- **provisioners:**

- file: Nos permite tranferir elementos desde la máquina real al box creado temporalmente.
- shell: Nos permite ejecutar comando de shell *"inline"* o como un fichero de *"script"* en el box lanzado temporalmente.

Una vez tenemos todo configurado, hacemos `packer build -force -debug image.json` y nos empezará a crear el box.

Una vez creado el box customizado tenemos que añadirlo a **Vagrant** con el comando `vagrant box add output-vagrant/package.box --name diego/ubuntu` siendo **output-vagrant/** el directorio donde se encuentra el box customizado.

Creamos un fichero **Vagrantfile** diciendole las configuraciones que necesitemos:

Ejemplo Vagrantfile:

```
Vagrant.configure("2") do |config|
  config.vm.box = "diego/ubuntu"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 386, host: 3386
  config.vm.network "private_network", ip: "192.168.50.4"
  config.vm.provider "virtualbox" do |vb|
    end
  end
end
```

Hacemos un `vagrant up` y comprobamos que todo funciona correctamente:

PHP Version 7.2.34-21+ubuntu16.04.1+deb.sury.org+1



System	Linux ubuntu-xenial 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86_64
Build Date	May 1 2021 11:52:36
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-mcrypt.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini, /etc/php/7.2/apache2/conf.d/20-zip.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no

```
[isx2031424@i01 diego_ubuntu]$ ldapsearch -x -LLL -H ldap://192.168.50.4 -b "dc=edt,dc=org" -s base
dn: dc=edt,dc=org
dc: edt
description: Escola del treball de Barcelona
objectClass: dcObject
objectClass: organization
o: edt.org
```

4. Conclusiones

Con packer he descubierto los dos tipos de infraestructura que son la mutable y la inmutable, de las cuales me he centrado en la infraestructura inmutable ya que es la solución que da **Packer** ante el problema explicado en la introducción.

Y podemos usar esta ami customizada ,que ya cuenta con la instalación y configuración de apache y php, en AWS para desplegar la infraestructura y nos ahorramos tener que acudir a un proveedor para realizar esta instalación y configuración. También pasaría con **Vagrant** y la box customizada.

5. Bibliografia

<https://www.packer.io/docs>

<https://www.packer.io/>

<https://github.com/hashicorp/packer>

<https://openwebinars.net/>