



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Planificación y Gestión de Infraestructuras TIC

*<Utilización de Vagrant y Ansible para creación,
aprovisionamiento y configuración de máquinas
virtuales en un suministrador de servicios en la nube>*

Autores: Miguel de la Cal Bravo y Félix Ángel Martínez Muela

Titulación: Máster en Ingeniería Informática

Fecha: 18/04/2020



ÍNDICE DE CONTENIDOS

1. Introducción.....	3
2. Puesta en marcha.....	3
3. Tutorial.....	4
3.1. Crear una instancia EC2 con Vagrant.....	4
3.2. Aprovisionar una instancia EC2 con Ansible y Vagrant	7
4. Otras posibilidades.....	11
4.1 Inventarios dinámicos.....	11
5. Ventajas e inconvenientes. Opinión personal	15
6. Referencias	16
ANEXO A. Crear usuario IAM y par de claves EC2.....	17
ANEXO B. Configuración e instalación de paquetes y plugins para usar AWS con Ansible y Vagrant.....	20
Claves de AWS.....	20
Opción 1: Claves dentro del Vagrantfile	20
Opción 2: Claves como variables de entorno.....	20
Opción 3: Claves dentro archivo credenciales.....	21
ANEXO C. Crear y configurar un grupo de seguridad en AWS	22
ANEXO D. Estructura de directorios y ficheros	24



1. INTRODUCCIÓN

En este trabajo utilizaremos las herramientas *Vagrant* y *Ansible*, para crear, aprovisionar y configurar máquinas virtuales en un proveedor de servicios en la nube como, por ejemplo, *Amazon Web Services*, en adelante, *AWS*¹, siendo éste el más utilizado en la actualidad y que utilizaremos a lo largo de este trabajo.

En la actualidad, cada vez es más común crear y desplegar máquinas virtuales en la nube, ya que estos servicios nos ofrecen una gran cantidad de ventajas, por ejemplo:

- Movilidad y disponibilidad
- Disminución de costes
- Servicio ágil
- Escalabilidad
- Descentralización
- Y muchas más...

Ante la necesidad de crear y aprovisionar una gran cantidad de máquinas, nos vemos “obligados” a encontrar una manera de realizar el proceso de forma automatizada, ya que realizar estos trabajos manualmente se convertiría en una tarea muy tediosa y .

Aquí es donde entran *Vagrant* y *Ansible*, las herramientas que emplearemos en este proyecto para facilitar los procesos de creación y aprovisionamiento de máquinas, con el añadido de realizar los despliegues en la nube utilizando los servicios de *AWS*.

Antes de ponernos manos a la obra, asumiremos que el usuario que vaya a realizar los siguientes procedimientos posee cierto manejo y conocimiento de las herramientas *Ansible*, *Vagrant* e instancias *EC2*.

Los archivos del proyecto están en el **repositorio**: https://github.com/miguelcal97/pgitic_aws

2. PUESTA EN MARCHA

Los procedimientos y desarrollos explicados en este documento han sido probados en un sistema operativo *Linux*, en la distribución de *Linux Ubuntu* (basada de *Debian*) en su versión 18.04 LTS. Para ello, ha sido necesaria la instalación del siguiente *software*:

<i>Software</i>	<i>Versión</i>
<i>Ansible</i> ²	2.9.6
<i>Vagrant</i> ³	2.2.7
<i>VirtualBox</i> ⁴	5.2.34

Tabla 1. *Software* utilizado en el proyecto

¹ AWS. <https://aws.amazon.com/es/>

² Ansible. <https://www.ansible.com/>

³ Vagrant. <https://www.vagrantup.com/>

⁴ VirtualBox. <https://www.virtualbox.org/>



Además del *software* indicado en la *Tabla 1*, también necesitaremos abrirnos una cuenta en AWS y registrarnos indicando nuestra **tarjeta de crédito**. En la *capa gratuita*⁵ de este servicio tenemos diferentes recursos que podemos probar sin coste alguno (con mucho cuidado), aunque en este trabajo solo utilizamos los de *Amazon EC2*⁶ para crear máquinas virtuales.

En el *ANEXO A. Crear usuario IAM y par de claves EC2*, se explica detalladamente el proceso de creación de una cuenta en AWS y un usuario IAM y clave para hacer uso de los servicios.

En el *ANEXO B. Configuración e instalación de paquetes y plugins para usar AWS con Ansible y Vagrant*, se muestra cómo configurar nuestro sistema para hacer uso de los recursos de AWS mediante *Vagrant* y *Ansible*.

En el *ANEXO C. Crear y configurar un grupo de seguridad en AWS*, se muestra cómo configurar un grupo de seguridad el cual nos va a permitir habilitar conexiones tanto entrantes como salientes de nuestra máquina *EC2*.

Con nuestro entorno ya configurado, estaremos en disposición de comenzar con diferentes ejemplos de uso, en el siguiente apartado de tutorial.

3. TUTORIAL

Instalados los programas y paquetes necesarios, es momento de ponernos manos a la obra y realizar una serie de tutoriales con ejercicios de ejemplo.

En el *ANEXO D. Estructura de directorios y ficheros*, se explica cómo están estructurados los directorios y ficheros necesarios. Esto es importante de cara a la ejecución de los comandos.

3.1. Crear una instancia EC2 con Vagrant

Empezando con *Vagrant*, nuestro objetivo será el de crear una máquina virtual en AWS (instancia *EC2*) utilizando dicha herramienta. Para ello, nos crearemos un fichero *Vagrantfile* para levantar una instancia *EC2*.

En el fichero del *Listado 1*, definimos las configuraciones de la máquina que deseamos crear:

- De la línea 5 a la 14, se introduce un código para arreglar un *bug* causado por el *plugin* de *vagrant-aws*. De esta manera, podremos levantar la instancia en la nube sin problemas.
- En la línea 17, importamos el *plugin* anteriormente mencionado para hacer uso de los servicios de AWS.
- De la línea 19 a la 46, creamos y configuramos la instancia de AWS. Los parámetros más importantes son:
 - *config.vm.box*: para levantar una instancia en la nube, no haremos uso de un *box* de *Vagrant* como tal, por lo que escogemos el *box* que añadimos “*aws-dummy*”.
 - *aws.keypair_name*: se trata de la clave utilizada para hacer login en las máquinas creadas. Este fichero nos lo descargamos al crearnos el par de claves.

⁵ Capa gratuita AWS. <https://aws.amazon.com/es/free/>

⁶ Amazon EC2. Amazon Elastic Compute Cloud: Se trata de un servicio web que proporciona capacidad informática en la nube segura y de tamaño modificable.

- *aws.instance_type*: en este parámetro especificamos el tipo de instancia *EC2* que deseamos levantar. En nuestro caso elegimos una instancia *t2.micro*, ya que esta es la única que se encuentra dentro de la capa gratuita de AWS de instancias *EC2*.
- *aws.ami*: además del tipo de instancia *EC2*, es necesario especificar un identificador de imagen *AMI*⁷ que contiene los recursos necesarios de la instancia. Sería nuestro box en AWS, por así decirlo.
- *aws.security_groups*: creamos la instancia con la política de seguridad *vagrant*, creado en el ANEXO C. *Crear y configurar un grupo de seguridad en AWS*.
- *override.ssh.private_key_path*: indica la clave *pem* que generamos para *EC2*.

```
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  # Solución al bug con el plugin de vagrant vagrant-aws
5  class Hash
6    def slice(*keep_keys)
7      h = {}
8      keep_keys.each { |key| h[key] = fetch(key) if has_key?(key) }
9      h
10   end unless Hash.method_defined?(:slice)
11   def except(*less_keys)
12     slice(*keys - less_keys)
13   end unless Hash.method_defined?(:except)
14 end
15
16 # Importamos el plugin del proveedor AWS
17 require 'vagrant-aws'
18
19 # Crear y configurar una instancia de AWS
20 Vagrant.configure("2") do |config|
21   config.vm.box = "aws-dummy"
22   config.ssh.keys_only = false
23
24   config.vm.provider :aws do |aws, override|
25
26     # Especificar el Keypair a utilizar
27     aws.keypair_name = "pgitic-key"
28     # Insertar claves directamente en Vagrant
29     #aws.access_key_id = "XXXXXXXXXXXXXXXXXXXX"
30     #aws.secret_access_key = "XXXXXXXXXXXXXXXXXXXX"
31     # Cambiar tipo de instancia, por defecto coge una m3.medium
32     # (no gratuita, obsoleta)
33     aws.instance_type = "t2.micro"
34     aws.ami = "ami-006a0174c6c25ac06"
35     # Cambiar región, por defecto coge us-east-1
36     aws.region = "eu-west-2"
37     # Grupo de seguridad que acepte conexiones SSH puerto 22
38     aws.security_groups = ['vagrant']
39     # Etiquetas
40     aws.tags = {
41       'Provider' => 'Vagrant'
42     }
43     # Sobreescribir usuario y ruta de clave privada
44     override.ssh.username = "ubuntu"
45     override.ssh.private_key_path = "../config/pgitic-aws.pem"
46   end
47 end
```

⁷ AMI. Amazon Machine Image: se trata de una imagen de máquina de Amazon, utilizado para crear máquinas dentro de Amazon Elastic Compute Cloud (EC2).



```
46 end
```

Listado 1. Código del fichero Vagrantfile

Con nuestro *Vagrantfile* configurado, en el directorio donde éste se encuentra podemos lanzar el siguiente comando para comenzar a levantar nuestra primera instancia *EC2* en *AWS* con *Vagrant*:

```
$ vagrant up --provider=aws
```

Como podemos apreciar, necesitamos añadir el parámetro del proveedor con el valor *AWS* (*--provider=aws*). Si todo va bien, deberíamos obtener un resultado similar al siguiente:

```
Bringing machine 'default' up with 'aws' provider...
==> default: Warning! The AWS provider doesn't support any of the Vagrant
==> default: high-level network configurations (`config.vm.network`). They
==> default: will be silently ignored.
==> default: Launching an instance with the following settings...
==> default: -- Type: t2.micro
==> default: -- AMI: ami-006a0174c6c25ac06
==> default: -- Region: eu-west-2
==> default: -- Keypair: pgitic-key
==> default: -- Security Groups: ["vagrant"]
==> default: -- Block Device Mapping: []
==> default: -- Terminate On Shutdown: false
==> default: -- Monitoring: false
==> default: -- EBS optimized: false
==> default: -- Source Destination check:
==> default: -- Assigning a public IP address in a VPC: false
==> default: -- VPC tenancy specification: default
==> default: Waiting for instance to become "ready"...
==> default: Waiting for SSH to become available...
==> default: Machine is booted and ready for use!
```

De esta manera, habremos conseguido levantar nuestra primera máquina virtual en *AWS* con *Vagrant*. Para comprobarlo, podemos dirigirnos a la interfaz de *AWS* (Servicios → *EC2* → *Instances*) y veremos nuestra instancia creada satisfactoriamente. Otra forma de comprobarlo es mediante el comando **vagrant ssh**, en el directorio donde tenemos nuestro *Vagrantfile*, y de esta manera nos conectaremos a la máquina.

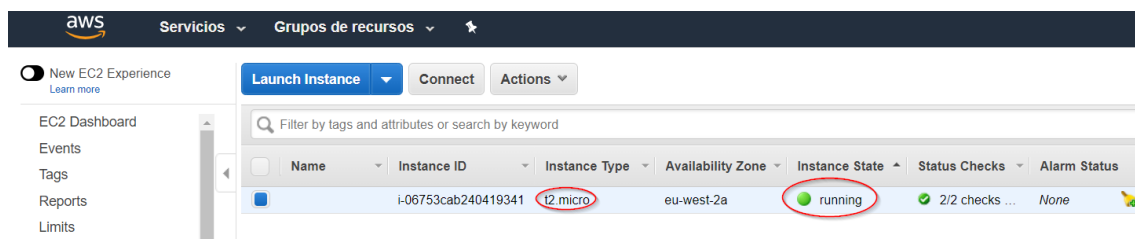


Figura 1. Instancia *t2.micro* creada satisfactoriamente con *Vagrant*

Si deseamos parar la máquina podremos hacerlo también con *Vagrant*:

```
$ vagrant halt
```

Por otra parte, si queremos borrarla, lo podemos hacer igualmente con el comando de *Vagrant*:

```
$ vagrant destroy
```

Así, limpiaremos nuestro entorno de trabajo y terminamos el primer ejemplo del tutorial.

3.2. Crear y aprovisionar una instancia EC2 con Ansible y Vagrant

Tras haber completado nuestro primer ejemplo, vamos a probar a crear una nueva instancia *EC2* con *Vagrant* y aprovisionarla con *Ansible*. Para esto, comenzaremos modificando el contenido del *Vagrantfile* añadiendo las líneas de la 14 hasta la 17 que se muestran en el *Listado 4*:

```
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3  ...
4  # Crear y configurar una instancia de AWS
5  Vagrant.configure("2") do |config|
6      config.vm.box = "aws-dummy"
7      config.ssh.keys_only = false
8
9      config.vm.provider :aws do |aws, override|
10         ...
11     end
12
13     # Aprovisionar instancia AWS
14     config.vm.provision "ansible" do |ansible|
15         ansible.playbook = "provision.yml"
16         ansible.host_key_checking = "false"
17     end
18 end
```

Listado 2. Modificación del fichero Vagrantfile para aprovisionar la máquina con Ansible

La novedad del *Vagrantfile* actual con respecto al anterior se corresponde con el uso del sistema de aprovisionamiento que trae *Vagrant*, para poder hacer uso de *Ansible* tras crear la máquina. A continuación, se explican las líneas añadidas:

- *ansible.playbook*: indica el *playbook* de *Ansible* que lanzaremos para aprovisionar la máquina, una vez esta haya sido creada, en este caso, se llamará *provision.yml*.
- *ansible.host_key_checking*: ponemos este parámetro a *false*, para que podamos ejecutar nuestro *playbook* sin necesidad de conectarnos previamente por *ssh* a la máquina.

Además de incluir estas novedades el *Vagrantfile*, también es necesario crearnos nuestro *playbook* de *Ansible*, el cual llamaremos *provision.yml*. El contenido de este *playbook* quedaría de la siguiente manera:

```
1  ---
2  - name: Aprovisionar instancia EC2 con Apache
3    hosts: all
4    remote_user: ubuntu
5    become: yes
6    tasks:
7      - name: Actualizar cache apt e instalar apache2
8        apt:
9          update_cache: yes
10         name: apache2
11
12     - name: Verificar localhost
13       uri:
14         url: http://localhost:80
15         method: GET
16         status_code: 200
17     ...
```

Listado 3. Código del playbook de Ansible "provision.yml"



Gracias a este *playbook* de *Ansible*, podremos aprovisionar la máquina en la nube con el *software* de *Apache* (previa actualización de caché *apt*), y verificar la conexión con el puerto 80:

```
$ vagrant up --provider=aws

Bringing machine 'default' up with 'aws' provider...
...
PLAY [Aprovisionar instancia EC2] *****

TASK [Gathering Facts] *****
ok: [default]

TASK [Actualizar cache apt e instalar apache2] *****
changed: [default]

TASK [Verificar localhost] *****
ok: [default]

PLAY RECAP *****
default                : ok=2    changed=1    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

Si nos dirigimos al navegador, en la dirección <IP pública instanciaEC2>:80, podemos comprobar que el aprovisionamiento se ha realizado correctamente:

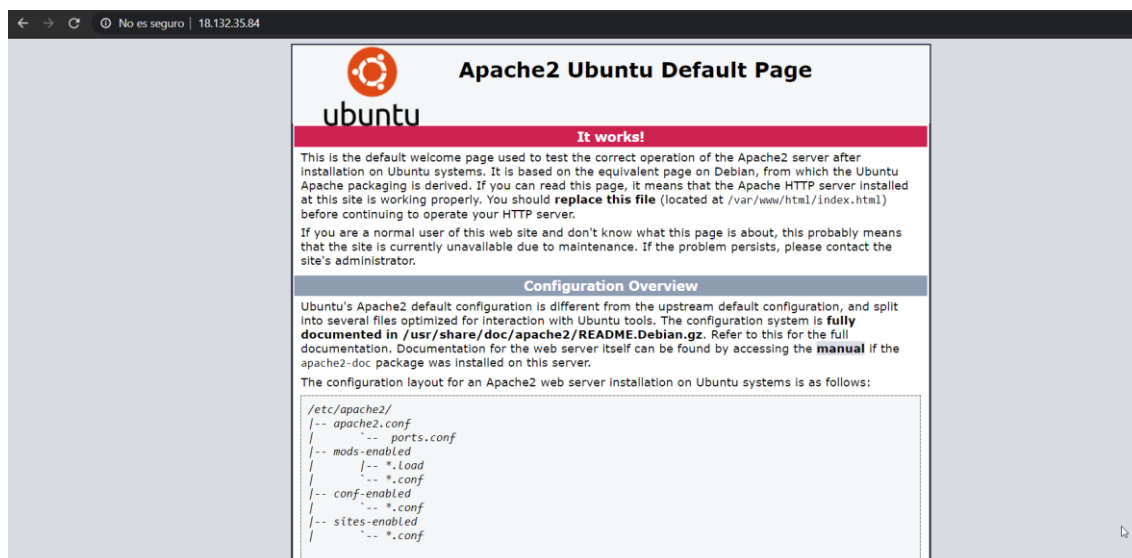


Figura 2. Instancia EC2 aprovisionada con el software de Apache

De esta manera, damos por terminado el segundo tutorial de *Ansible* y *Vagrant* en el que hemos conseguido crear una máquina y aprovisionarla con el *software* de *Apache*, cumpliendo así con el objetivo principal de esta memoria.

De nuevo, para limpiar nuestro entorno de trabajo de AWS eliminaremos la máquina mediante el siguiente comando de *Vagrant*:

```
$ vagrant destroy
```




3.3 Playbooks de Ansible para crear y aprovisionar máquinas en AWS

En el último ejemplo del tutorial, veremos alguno de los módulos de *Ansible* para AWS.

Por ejemplo, podemos utilizar *Ansible* para levantar máquinas en AWS sin necesidad de utilizar *Vagrant*. Para ello, haremos uso de los diferentes módulos que posee *Ansible* para la gestión de recursos en AWS. En el *playbook* *aws_deploy.yml* que se muestra en el *Listado 4*, comenzaremos creando una instancia EC2 *t2.micro*, mediante el módulo *ec2* de *Ansible* (líneas 5 a la 16) con la misma configuración que cuando la creamos con *Vagrant* en el apartado 3.1. *Crear una instancia EC2 con Vagrant*.

Una vez se haya creado la instancia, la añadiremos al grupo de *hosts* “*ec2_pgitic*” en tiempo de ejecución con el módulo *add_host* (ver líneas 18 hasta 22), esperaremos a que esta máquina esté disponible para conectarnos por *SSH* mediante el módulo *wait_for_connection* (líneas 24 a la 30).

Hecho esto, el *playbook* concluirá con un nuevo *play* que realizará lo mismo que en el *provision.yml* (revisar líneas 32 hasta la 45).

```
1  ---
2  - name: Crear instancia EC2 y añadirla a un grupo de hosts
3    hosts: localhost
4    tasks:
5      - name: Crear instancia AWS EC2 t2.micro
6        ec2:
7          key_name: pgitic-key
8          instance_type: t2.micro
9          image: ami-006a0174c6c25ac06
10         wait: yes
11         group: vagrant
12         count: 1
13         vpc_subnet_id: subnet-21199e5b
14         assign_public_ip: yes
15         region: eu-west-2
16         register: ec2
17
18      - name: Añadir nueva instancia al grupo de hosts
19        add_host:
20          hostname: "{{ item.public_ip }}"
21          groupname: ec2_pgitic
22          loop: "{{ ec2.instances }}"
23
24      - name: Esperar a que la conexión SSH esté levantada
25        delegate_to: "{{ item.public_dns_name }}"
26        wait_for_connection:
27          delay: 60
28          timeout: 300
29          loop: "{{ ec2.instances }}"
30        remote_user: ubuntu
31
32      - name: Aprovisionar instancia EC2 creada con Ansible
33        hosts: ec2_pgitic
34        become: yes
35        remote_user: ubuntu
36        tasks:
37          - name: Actualizar cache apt e instalar apache2
38            apt:
39              update_cache: yes
40              name: apache2
```



```
41 - name: Verificar localhost
42   uri:
43     url: http://localhost:80
44     method: GET
45     status_code: 200
46   ...
```

Listado 4. Código del *playbook* *aws_deploy.yml* para crear una instancia *t2.micro* en AWS

Para ejecutar este *playbook*, lanzaremos el siguiente comando, haciendo uso del parámetro de clave privada para poder entrar a la máquina creada y aprovisionarla con *Ansible*:

```
$ ansible-playbook ./examples/aws_deploy.yml
  --private-key=./config/pgitic-key.pem
  --ssh-common-args '-o StrictHostKeyChecking=no'
```

Y obtendremos como resultado algo similar a lo siguiente, cambiando la dirección IP:

```
PLAY [Crear instancia EC2 y añadirla a un grupo de hosts]
*****
TASK [Gathering Facts] *****
ok: [localhost]

TASK [Crear instancia AWS EC2 t2.micro] *****
changed: [localhost]

TASK [Añadir nueva instancia al grupo de hosts] *****
changed: [localhost] => (item={u'ramdisk': None, u'kernel': None,
u'root_device_type': u'ebs', u'private_dns_name': u'ip-172-31-22-134.eu-
west-2.compute.internal'...

TASK [Esperar a que la conexión SSH esté levantada]
*****
ok: [localhost -> ec2-3-8-120-33.eu-west-2.compute.amazonaws.com] =>
(item={u'ramdisk': None, u'kernel': None, u'root_device_type': u'ebs',
u'private_dns_name': u'ip-172-31-22-134.eu-west-2.compute.internal'...

PLAY [Aprovisionar instancia EC2 creada con Ansible] *****

TASK [Gathering Facts] *****
ok: [3.8.120.33]

TASK [Actualizar cache apt e instalar apache2] *****
changed: [3.8.120.33]

TASK [Verificar localhost] *****
ok: [3.8.120.33]

PLAY RECAP *****
3.8.120.33          : ok=3    changed=1    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
localhost        : ok=4    changed=2    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```



En esta ocasión, habremos levantado nuevamente el servicio de *Apache* en la instancia *EC2* de *AWS*, pero lo habremos hecho únicamente con la herramienta *Ansible*, de una manera mucho más cómoda que si hubiéramos utilizado *Vagrant* sin instalar el *plugin* de *vagrant-aws*.

En este caso, si deseamos eliminar la máquina nos dirigimos a la consola de *AWS* y seleccionamos la instancia para posteriormente cambiarle el estado a “*Terminated*”.

4. OTRAS POSIBILIDADES

Más allá de haber utilizado *Vagrant* y *Ansible* para crear y aprovisionar máquinas en la nube, en muchas ocasiones podremos utilizar estas herramientas para otras posibilidades que nos ofrecen.

4.1 Inventarios dinámicos y comandos ad-hoc

Un aspecto muy interesante tiene que ver con los inventarios dinámicos de *Ansible*, al hacer uso de máquinas en un proveedor de servicios en la nube.

IMPORTANTE: Para evitar tener que ejecutar estos comandos, hemos desarrollado directamente un *script* el cual se encuentra dentro de */inventory*, llamado *inventory.sh*. Dicho *script* se encargará de actualizar directamente nuestro inventario dinámico, teniendo en cuenta que se encuentra instalada la librería de *ansible* y actualizando la caché, previo paso a ejecutar el comando que nos actualiza el inventario.

Para empezar, es necesario descargarse mediante *pip* la librería de *ansible*:

```
$ pip install ansible
```

Para obtener el inventario de hosts en la nube de forma dinámica, haremos uso de un script en lenguaje *Python* llamado *ec2.py*⁸, que hemos extraído de la web oficial de *Ansible*. Para comprobar que funciona ejecutamos el siguiente comando:

```
$ ./inventory/ec2.py --list
```

En caso de obtener algún fallo, éste puede deberse a que la caché no está actualizada. Para que nos lo haga bien, actualizamos la caché mediante el siguiente comando:

```
$ ./inventory/ec2.py --refresh-cache
```

Hecho esto, podemos lanzar un comando *Ansible* para verificar que se hace el *ping* correctamente a las máquinas levantadas. Utilizaremos como inventario el *script* *ec2.py*, que nos devolverá las máquinas levantadas en *AWS*, usuario *ubuntu* y la clave privada que nos generamos.

```
$ ansible all -i ./inventory/ec2.py -u ubuntu -m ping --private-key  
./config/pgitic-key.pem
```

⁸ https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#inventory-script-example-aws-ec2



El resultado será similar al siguiente, cambiando la dirección IP de la máquina:

```
18.130.16.60 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Siguiendo en la línea del comando anterior, podríamos hacer uso de los diferentes módulos de *Ansible* contra las máquinas levantadas en la nube, haciendo uso de comandos *ad-hoc* para ejecutar tareas automáticamente de forma rápida y sencilla.

```
$ ansible all -i ./inventory/ec2.py -u ubuntu -m command -a "id"
--private-key ./config/pgitic-key.pem
18.130.16.60 | CHANGED | rc=0 >>
uid=1000 (ubuntu) gid=1000 (ubuntu)
groups=1000 (ubuntu) , 4 (adm) , 20 (dialout) , 24 (cdrom) , 25 (floppy) , 27 (sudo) , 29 (audio) , 30 (dip) , 44 (video) , 46 (plugdev) , 108 (lxd) , 114 (netdev)
```

De la misma manera, podríamos aprovechar los inventarios dinámicos para lanzar *playbooks* sobre los nodos levantados en *AWS*, pasando los argumentos necesarios de inventario y clave.

4.2 AWS Systems Manager para realizar automatizaciones con Ansible

Por último, hemos descubierto que *AWS* ofrece una sección llamada *Systems Manager*, desde la cual podemos crear y lanzar *playbooks* de *Ansible* directamente sobre nuestras instancias de *EC2* creadas en *AWS*⁹. Para hacer esto, nos dirigimos a Servicios → Administración y Gobierno → *Systems Manager*, hacemos click en *Instances & Nodes* → *State Manager*.

Dentro de la página, pincharemos en el botón “*Create association*”:

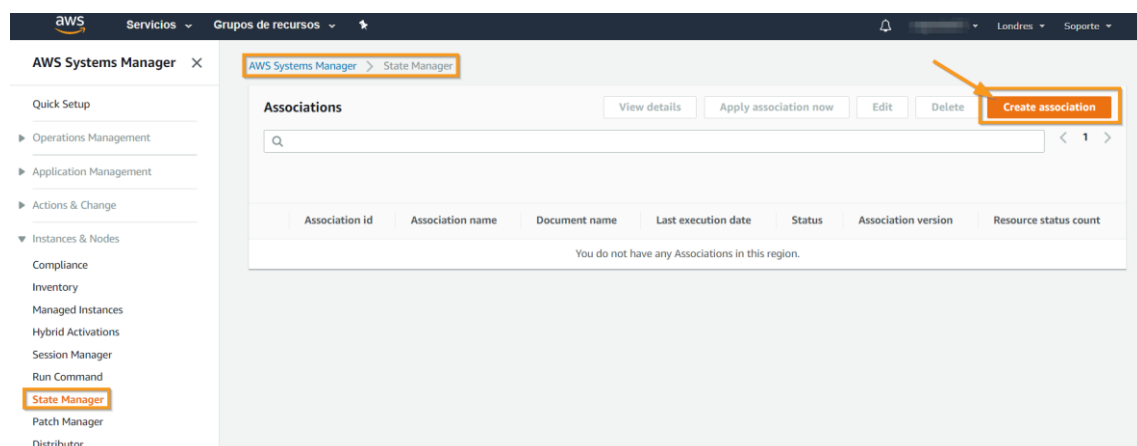


Figura 3. Creación de una asociación en AWS Systems Manager

⁹ NOTA. Es necesario que la instancia tenga instalado *Ansible* y el agente *amazon-ssm-agent*. En el *playbook* *aws_deploy_ssm.yml* del repositorio se realizan las configuraciones necesarias para Ubuntu.



Una vez estemos en el nuevo menú, le daremos el nombre de la asociación que prefiramos, en nuestro caso la hemos llamado “*Ansible_Install_Apache*”. Además, es importante establecer el tipo de documento a *AWS-RunAnsiblePlaybook*, tal y como se muestra en la *Figura 4*.

Name - optional
Provide a name for your Association.

Ansible_Install_Apache

Document

Document
AWS-RunAnsiblePlaybook

Document description
Use this document to run Ansible playbooks on Amazon EC2 managed instances. Specify either YAML text or URL. If you specify both, the URL parameter will be used. Use the extravar parameter to send runtime variables to the Ansible execution. Use the check parameter to perform a dry run of the Ansible execution. The output of the dry run shows the changes that will be made when the playbook is executed.

Use the service-linked role [AWSServiceRoleForAmazonSSM](#) to allow State Manager to manage AWS resources on your behalf.

× < 1 >

Document name prefix: Equals: AWS-RunAnsiblePlaybook × Clear filters

	Name	Owner	Platform types	Document type
<input checked="" type="radio"/>	AWS-RunAnsiblePlaybook	Amazon	Linux	Command

Figura 4. Nombre de la asociación y tipo de documento AWS-RunAnsiblePlaybook

Dentro del apartado de “*Parameters*”, en el campo de *Playbook* añadimos el contenido del *playbook* de *Ansible* que deseamos ejecutar, en nuestro caso, copiaremos el contenido del fichero *provision.yml* visto anteriormente. Otra opción podría ser la de añadir la *url* de un *playbook* de nuestro repositorio (campo *Playbookurl*), lo cual puede ser lo más cómodo en muchas ocasiones.

Parameters

Playbook
(Optional) If you don't specify a URL, then you must specify playbook YAML in this field.

```
---  
- name: Aprovisionar instancia EC2 con Apache  
  hosts: all
```

Playbookurl
(Optional) If you don't specify playbook YAML, then you must specify a URL where the playbook is stored. You can specify the URL in the following formats: http://example.com/playbook.yml or s3://examplebucket/plabook.url. For security reasons, you can't specify a URL with quotes.

Figura 5. Parámetros “*Playbook*” y “*Playbookurl*” del tipo de documento seleccionado

Por último, únicamente nos quedará seleccionar las instancias *EC2* sobre las que queramos ejecutar dicho *playbook*. En nuestro caso seleccionaremos una máquina que no hayamos aprovisionado previamente, para así automatizar la instalación de *Apache* sobre la misma.

En la *Figura 6*, podemos ver cómo elegir las máquinas objetivo del *playbook* de *Ansible*.



Targets

Targets are the instances you would like to associate with this document. You can choose to target by both managed instance and tag.

Select targets by

- ☐ Selecting all managed instances in this region under this account
- ☐ Specifying tags
- ☒ Manually Selecting Instance

i-03edae07d34ed07c4 X

<input checked="" type="checkbox"/>	Name	Instance id	Instance state	Availability zone	Ping st
<input checked="" type="checkbox"/>	-	i-03edae07d34ed07c4	running	eu-west-2a	Online

Figura 6. Selección de máquinas sobre las que ejecutar el playbook

Para terminar, podemos elegir otras opciones como programar la ejecución o lanzarla en el momento, escribir la salida en un contenedor S3 de *Amazon* (no lo hemos probado por miedo a que no estuviera incluido en la capa gratuita), etc.

Una vez finalizada la configuración de la asociación, haremos click en el botón “*Create Association*” y comenzará a ejecutarse, ya que configuramos que se lanzase en el momento.

El resultado de la ejecución no lo podremos ver con la misma normalidad que al lanzar un playbook directamente, a no ser que guardemos la salida de la ejecución en un contenedor S3.

De todas maneras, en la *Figura 7* se muestra cómo la ejecución finalmente se ha completado con éxito:

Associations

View details

Apply association now

Edit

Delete

Create association

Q

Association Name: Equal: Ansible_Install_Apache

Clear filters

	Association id	Association name	Document name	Last execution date	Status	Association version	Resource status count
<div></div>	ce9a9aa2-06e2-4bb7-ba4d-e0464ea5c1a1	Ansible_Install_Apache	AWS-RunAnsiblePlaybook	Mon, 13 Apr 2020 18:12:24 GMT	<div>Success</div>	1	Success:1

Figura 7. Ejecución de un playbook desde AWS Systems Manager

Nuevamente, si nos dirigimos al navegador podremos comprobar que se ha aprovisionado correctamente la máquina.

Aunque de esta manera no es la mejor opción de lanzar un *playbook* sobre las instancias (puesto que además estas necesitan también tener instalado *Ansible*), es otra alternativa posible de AWS para poder lanzar automatizaciones que considerábamos interesante incluir en este proyecto.

Esta asociación permanecerá dentro de la sección “*State Manager*”, y podremos utilizarla en futuras ocasiones sobre diferentes instancias cambiando el parámetro “*Targets*” seleccionando aquellas que queramos configurar.



5. VENTAJAS E INCONVENIENTES. OPINIÓN PERSONAL

Las *ventajas* de utilizar herramientas como *Vagrant* y *Ansible* para la creación y aprovisionamiento de máquinas son muy grandes, pues nos facilitan la labor de tratar con procesos muy repetitivos que pueden ser realizados de forma automática, y de una manera muy sencilla.

Con esto no sólo ahorraremos tiempo, sino que además evitaremos posibles errores humanos al realizar tareas tan repetitivas en diferentes máquinas como, por ejemplo, saltarnos algún paso del aprovisionamiento de estas.

En cuanto a los *inconvenientes*, en nuestras prácticas hemos visto que *Vagrant* no es la mejor opción para crear máquinas en la nube, ya que esta tecnología no hace uso de sus *boxes*, sino de las imágenes del proveedor, en nuestro caso las *AMI* de *AWS*. Con *Ansible* podemos hacer uso de algunos de sus módulos para levantar máquinas de *AWS*, y aprovisionarlas posteriormente mediante *playbooks* gracias a sus módulos.

En cuanto a nuestra experiencia, conocemos otra herramienta alternativa para desplegar máquinas virtuales la cual nos ha funcionado muy bien. Dicha herramienta es *Terraform*¹⁰, la cual hace uso de recursos de un proveedor en la nube, que podemos configurar sus parámetros mediante ficheros con extensión *.tf*, haciendo uso del lenguaje de *Terraform*.

De forma declarativa, con esta herramienta podemos indicar los recursos que deseamos tener creados en el proveedor de servicios en la nube. En primer lugar, inicializamos el proyecto de *Terraform* y creamos dos archivos *main.tf* y *provider.tf* para desarrollar así nuestro “plan de recursos”. Hecho esto, podemos comprobar si se ha añadido, modificado o eliminado algún recurso, y realizar el aprovisionamiento de forma automática de estos recursos.

En esta ocasión, cogeremos del repositorio los archivos de la carpeta */terraform* dentro de */src*, y nos cambiaremos a dicho directorio para lanzar los siguientes comandos de *Terraform*:

```
$ terraform init
$ terraform plan
$ terraform apply
```

De esta manera, podríamos reemplazar *Vagrant* por *Terraform* para la creación de máquinas en la nube, y luego aprovisionarlas a nuestro gusto con *Ansible*.

Para terminar, nuestra conclusión es que *Vagrant* funciona muy bien junto con *VirtualBox* para crear máquinas virtuales en nuestra máquina anfitrión, haciendo uso de los *boxes*. Sin embargo, consideramos que *Vagrant* no es la mejor alternativa para crear máquinas virtuales en la nube por las razones explicadas anteriormente. Por ello, optaríamos por utilizar *Terraform* para la creación de máquinas virtuales y *Ansible* para el aprovisionamiento y gestión de configuraciones de forma automatizada.

¹⁰ Terraform. <https://www.terraform.io/>



6. REFERENCIAS

A continuación, se adjuntan las referencias utilizadas para la elaboración de esta memoria:

- Repositorio plugin Vagrant AWS: <https://github.com/mitchellh/vagrant-aws>
- Tutorial Vagrant levantar instancias EC2 en AWS:
<https://openwebinars.net/blog/vagrant-sobre-aws-amazon-videotutorial/>
- Inventarios dinámicos en Ansible para AWS:
https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#inventory-script-example-aws-ec2
- Playbooks de Ansible desde AWS Systems Manager:
<https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state-manager-ansible.html>
- Terraform y Ansible para aprovisionar máquinas EC2 en AWS:
<https://alex.dzyoba.com/blog/terraform-ansible/>
- Documentación de AWS: <https://docs.aws.amazon.com/>
- Documentación de Vagrant: <https://www.vagrantup.com/docs/>
- Documentación de Ansible: <https://docs.ansible.com/>
- Documentación de Terraform: <https://www.terraform.io/>
- Bash scripting cheatsheet: <https://devhints.io/bash>

ANEXO A. CREAR USUARIO IAM Y PAR DE CLAVES EC2

Una vez validada nuestra cuenta en AWS con nuestra tarjeta de crédito, tendremos que crearnos un usuario en la sección de Servicios → Seguridad, Identidad y Conformidad → IAM¹¹, hacemos click en Usuarios → Añadir usuario(s), tal y como se muestra en la *Figura 8*.

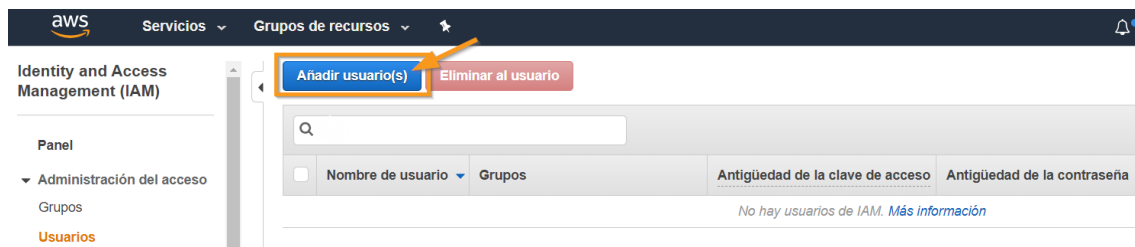


Figura 8. Añadir usuario en IAM de AWS

En el siguiente paso, elegiremos nuestro nombre de usuario, y nos aseguraremos de marcar la opción de tipo de acceso “**Acceso mediante programación**” (ver *Figura 9*), de tal forma que nos habilitará una ID de clave de acceso y una clave de acceso secreta (`aws_access_key_id` y `aws_secret_access_key`) para acceder más adelante a los servicios de AWS.

Añadir usuario(s) 1 2 3 4 5

Establecer los detalles del usuario

Puede añadir varios usuarios a la vez con los mismos permisos y el mismo tipo de acceso. [Más información](#)

Nombre de usuario*

[+ Añadir otro usuario](#)

Seleccionar el tipo de acceso de AWS

Seleccione la forma en que estos usuarios accederán a AWS. Las claves de acceso y las contraseñas generadas automáticamente se proporcionan en el último paso. [Más información](#)

Tipo de acceso* ☒ **Acceso mediante programación**
Habilita una **ID de clave de acceso** y una **clave de acceso secreta** para el SDK, la CLI y la API de AWS, además de otras herramientas de desarrollo.

☐ **Acceso a la consola de administración de AWS**
Habilita una **contraseña** que permite a los usuarios iniciar sesión en la consola de administración de AWS.

Figura 9. Crear usuario con tipo de acceso mediante programación

Tras este paso, podremos personalizar dicho usuario con diferentes permisos, crearlo dentro de un grupo, con etiquetas, etc. Una vez hayamos llegado al final de la creación del usuario, tras revisar las opciones de creación, se nos generará la ID de clave de acceso y la clave de acceso secreta anteriormente mencionadas, tal y como se ve en la *Figura 10*. Estas claves deberemos guardarlas en un lugar seguro y no perderlas, ya que si no nos quedaremos sin poder hacer uso de estas.

¹¹ IAM. Identity and Access Management.

Correcto

Ha creado correctamente los usuarios que se muestran a continuación. Puede ver y descargar las credenciales de seguridad de los usuarios. También puede enviar a los usuarios un correo electrónico con instrucciones para iniciar sesión en la consola de administración de AWS. Esta es la última vez que las credenciales estarán disponibles para descargarlas. Sin embargo, puede crear otras en cualquier momento.

Los usuarios con acceso a la consola de administración de AWS pueden iniciar sesión en:

[https://console.aws.amazon.com/console/home](#)

Descargar .csv

	Usuario	ID de clave de acceso	Clave de acceso secreta
▶	pgitic_test		 Ocultar

Figura 10. Claves de acceso para el usuario creado

Llegados a este punto, ya habremos creado nuestra cuenta *IAM* satisfactoriamente, y podremos hacer uso de los servicios de *AWS* a través de las claves generadas. Como se puede ver en la *Figura 11*, nuestro usuario *pgitic_test* ya se ha creado correctamente.

Servicios ▾ Grupos de recursos ▾ ⭐

Identity and Access Management (IAM)

Panel

▼ Administración del acceso

Grupos

Usuarios

Añadir usuario(s) Eliminar al usuario

<input type="checkbox"/>	Nombre de usuario ▾	Grupos	Antigüedad de la clave de acceso
<input type="checkbox"/>	pgitic_test	PGITIC	Hoy

Figura 11. Usuario *pgitic_test* creado en IAM

Además de crear un usuario en *IAM*, también necesitaremos generar un par de claves o *keypair* en *EC2*, para posteriormente levantar nuestras instancias.

Para ello, nos dirigiremos a la sección de Servicios → *EC2* → *Network & Security* → *Key Pairs*, y haremos click en “*Create Key Pair*”.

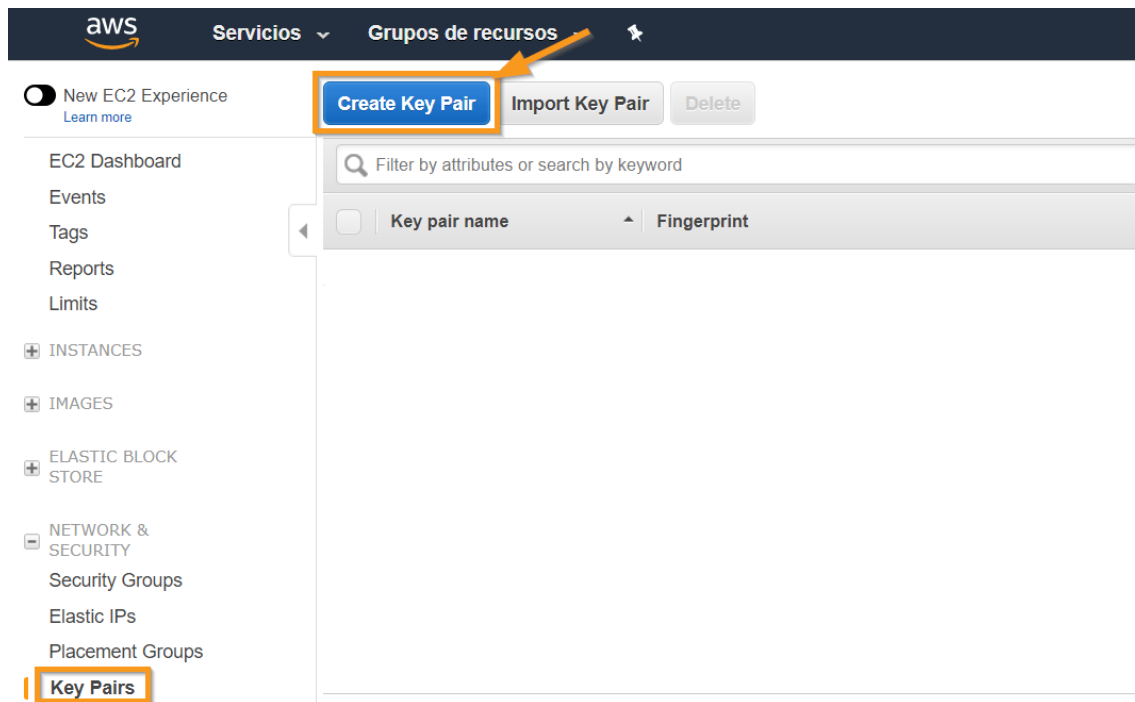


Figura 12. Crear par de claves para EC2

Nos pedirá un nombre para el par de claves, que llamaremos “*pgitic-key*”, y se nos descargará un fichero *.pem* que deberemos guardar de forma segura dentro de la carpeta */config*, ya que lo necesitaremos en los ejemplos del tutorial.



ANEXO B. CONFIGURACIÓN E INSTALACIÓN DE PAQUETES Y PLUGINS PARA USAR AWS CON ANSIBLE Y VAGRANT

Paquetes y plugins

Para ejecutar los módulos de *Ansible* para instancias *Amazon EC2* de AWS, será necesario instalar el paquete “*boto*” de *pip* (si no tenemos *pip* instalado, necesitaremos instalar el paquete *python-pip* con *apt*). Para ello, ejecutaremos el siguiente comando:

```
$ pip install boto
```

En cuanto a la parte de *Vagrant*, será necesario instalar un plugin llamado *vagrant-aws* para esta herramienta. Para ello lanzamos el siguiente comando:

```
$ vagrant plugin install vagrant-aws
```

En caso de que nos salte un error de dependencias, tendríamos que instalar también otro *plugin* de *vagrant* con el siguiente comando:

```
$ vagrant plugin install --plugin-version 1.0.1 fog-ovirt
```

Hecho esto, únicamente nos quedará añadir el box *dummy*, ejecutando en la línea de comandos:

```
$ vagrant box add aws-dummy https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box
```

Claves de AWS

Existen diferentes maneras de almacenar las claves de nuestro usuario AWS que será necesario para la creación de instancias. Nosotros recomendamos la opción 3.

Opción 1: Claves dentro del Vagrantfile

Dentro del *Vagrantfile* existen dos campos comentados:

```
aws.access_key_id = "XXXXXX"  
aws.secret_access_key = "XXXXXX"
```

Donde podemos rellenarlo con nuestros datos y funcionará sin problemas.

Opción 2: Claves como variables de entorno

Para trabajar cómodamente con AWS, es recomendable establecer las variables de entorno, por lo que estableceremos los valores de nuestras claves *aws_access_key_id* y *aws_secret_access_key*:

```
$ export AWS_ACCESS_KEY_ID='XXXXXXXXXXXXXXXXXXXX'  
$ export AWS_SECRET_ACCESS_KEY='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```



Adicionalmente, podemos exportar la región en la siguiente variable de entorno:

```
$ export AWS_DEFAULT_REGION='eu-west-2'
```

Opción 3: Claves dentro archivo credenciales

IMPORTANTE: Para facilitar los pasos posteriores se ha desarrollado un script el cual ejecuta las instrucciones posteriores. El script se encuentra en la ruta */config* el cual se llama *configuration.sh* y debemos ejecutar una vez hemos rellenado *config* y *credentials*.

Otra manera de establecer estos ajustes es creando el directorio *~/.aws*, creando dentro los siguientes ficheros:

Contenido del fichero *config*:

```
[default]
region = eu-west-2
```

Contenido del fichero *credentials*:

```
[default]
aws_access_key_id = XXXXXXXXXXXXXXXX
aws_secret_access_key = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

En el fichero *config* definimos la región en la que crearemos nuestras máquinas en AWS.

En el fichero *credentials* configuramos las claves de ID y acceso para poder hacer uso de nuestra cuenta en AWS.

Para automatizar tal tarea se ha creado un script sh el cual accediendo a la carpeta */config* encontraremos. Pasos:

1. Configuramos la zona en la cual se desplegará nuestra instancia en el archivo *config*.
2. Rellenamos nuestras credenciales dentro del archivo *credentials*.
3. Ejecutamos el *script* *./configuration.sh*.

Nota: en caso de que el *script* nos dé fallo debemos ejecutar el siguiente comando:

```
$ chmod +x configuration.sh
```

ANEXO C. CREAR Y CONFIGURAR UN GRUPO DE SEGURIDAD EN AWS

Para que nuestras instancias funcionen es necesario que se configuren en función a un grupo de seguridad, el cual determinará que puertos quedan abiertos de entrada, y desde que direcciones se pueden acceder a dichos puertos, también aplicable a las conexiones de salida.

Procedemos a entrar en la consola de administración de AWS. Una vez dentro nos dirigimos a Servicios → EC2 → Network and Security → Security Groups.

Pinchamos en “Create security group”:

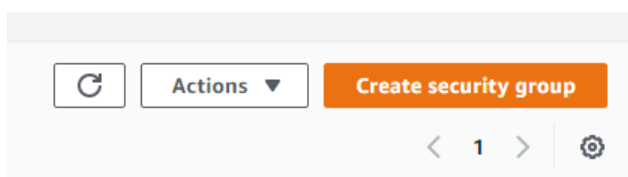


Figura 13. Crear grupo de seguridad en EC2

Rellenamos los datos, asignándole el nombre de *vagrant*, importante mantener dicho grupo para no tener que cambiarlo en el *Vagrantfile*. Seleccionamos el VPC que viene por defecto u otro si tenemos configurado uno aparte.

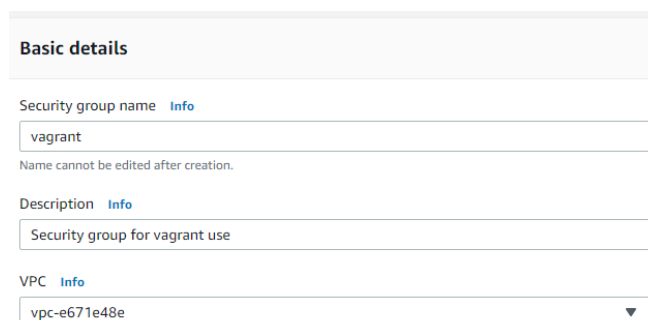
A screenshot of the 'Basic details' section of the AWS Security Groups 'Create' wizard. It contains three input fields: 'Security group name' with the value 'vagrant', 'Description' with the value 'Security group for vagrant use', and a 'VPC' dropdown menu currently showing 'vpc-e671e48e'. Each field has an 'Info' link next to it. A note below the name field states 'Name cannot be edited after creation.'

Figura 14. Nombre y descripción del grupo de seguridad

Creamos las reglas de entrada (*Inbound rules*). En nuestro caso tenemos que habilitar si o si las conexiones por *ssh*, las provenientes del puerto 22. También habilitamos las del puerto 80 debido a que para comprobar que funciona vamos a lanzar un servidor apache en dicho puerto.

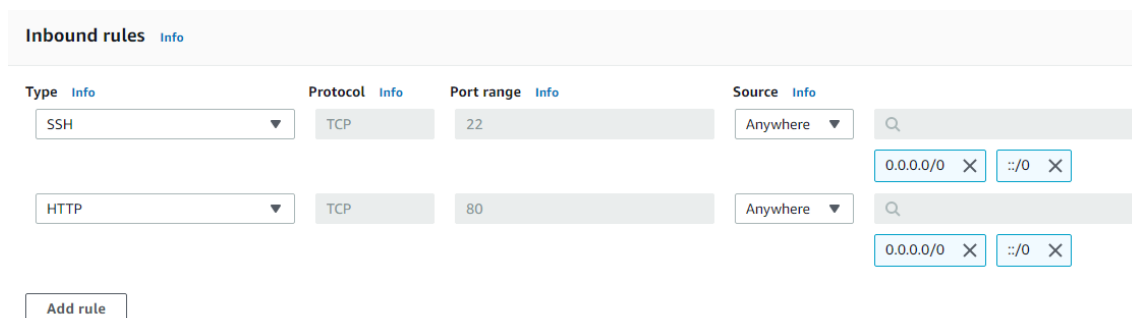
A screenshot of the 'Inbound rules' configuration section in the AWS console. It features a table with columns: Type, Protocol, Port range, and Source. Two rules are listed: one for SSH (Type: SSH, Protocol: TCP, Port range: 22, Source: Anywhere) and one for HTTP (Type: HTTP, Protocol: TCP, Port range: 80, Source: Anywhere). To the right of each rule, there are IP address input fields showing '0.0.0.0/0' and '::/0'. At the bottom left, there is an 'Add rule' button.

Figura 15. Configuración de reglas entrantes del grupo de seguridad



Las reglas externas las dejamos configuradas por defecto, permitiendo todo tipo de conexiones hacia fuera de nuestra instancia *EC2*.

Esto es todo lo necesario para que nuestra instancia *EC2* se pueda comunicar con el exterior, en nuestro caso para que *Vagrant* pueda darle las órdenes necesarias a la hora de creación de la máquina y aprovisionamiento de esta mediante *Ansible*.

Outbound rules [Info](#)

Type Info	Protocol Info	Port range Info	Destination Info
All traffic ▼	All	All	Custom ▼
			<input type="text" value="0.0.0.0/0"/>

[Add rule](#)

Figura 16. Configuración de reglas de salida del grupo de seguridad

ANEXO D. ESTRUCTURA DE DIRECTORIOS Y FICHEROS

Con el fin de facilitar la estructura del espacio de trabajo, en este anexo se muestra cómo debe tener el usuario ubicados sus ficheros y directorios.

```
$ tree
.
├── src
│   ├── config
│   │   ├── config
│   │   ├── configuration.sh
│   │   ├── credentials
│   │   └── pgitic-key.pem
│   ├── examples
│   │   ├── aws_deploy.yml
│   │   └── aws_deploy_ssm.yml
│   ├── inventory
│   │   ├── ec2.ini
│   │   ├── ec2.py
│   │   └── inventory.sh
│   ├── terraform
│   │   ├── main.tf
│   │   └── provider.tf
│   ├── provision.yml
│   └── Vagrantfile
```

Todos los comandos, salvo que se indique lo contrario, se ejecutarán dentro del directorio `/src` de nuestro proyecto.

En el directorio raíz `/src` encontramos los siguientes archivos:

- *Vagrantfile*: fichero de configuración *Vagrant*, utilizando el proveedor AWS.
- *provision.yml*: *playbook* de *Ansible* para el aprovisionamiento de la máquina AWS creada con *Vagrant*.

En el directorio `/config` encontramos los archivos:

- *config*: configuración de parámetros de AWS.
- *configuration.sh*: script que automatiza la creación y copiado de directorios y ficheros de configuración.
- *credentials*: credenciales de AWS para crear instancias *EC2*.
- *pgitic-key.pem*: clave utilizada para acceder a las instancias de *EC2* creadas.

En el directorio `/examples` encontramos los archivos:

- *aws_deploy.yml*: *playbook* que automatiza la creación de una instancia *EC2* y la aprovisiona con *Apache*, sin necesidad de utilizar *Vagrant*.
- *aws_deploy_ssm.yml*: *playbook* que automatiza la creación de una instancia *EC2* y la aprovisiona con el *software* necesario para gestionarla desde *AWS Systems Manager*.



En el directorio */inventory* encontramos los archivos:

- *ec2.ini*: inventario que genera el *script* de *Python* para inventarios dinámicos de *Ansible*.
- *ec2.py*: *script* que genera un inventario dinámico con las instancias *EC2* levantadas en *AWS*.
- *inventory.sh*: *script* que comprueba todo lo necesario para que se pueda hacer uso de los inventarios dinámicos de *Ansible* con *AWS*.

En el directorio */terraform* encontramos los archivos:

- *main.tf*: fichero de *Terraform* donde definimos los recursos que queremos crear, a través de *AWS* y también aprovisionarlo con un *playbook* de *Ansible*.
- *provider.tf*: fichero de *Terraform* donde se indican las configuraciones del proveedor *AWS* para crear los recursos del *main.tf*.

