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

En este trabajo utilizaremos las herramientas *Vagrant* y *Ansible*, para crear, aprovisionar y configurar máquinas virtuales en un suministrador 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.

Aquí es donde entran *Vagrant* y *Ansible*, las herramientas que estudiaremos en este proyecto para facilitar los procesos de creación y aprovisionamiento de máquinas, con el añadido de desplegarlo 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 de las herramientas *Ansible*, *Vagrant* e instancias *EC2*.

2. PUESTA EN MARCHA

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

Software	**Versión**
Ansible[2]	2.9.6
Vagrant[3]	2.2.7
VirtualBox[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 en los recursos), 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* 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*ANEXO B. Configuración e instalación de paquetes y plugins para usar AWS con Ansible y Vagrant, se muestra cómo se configura 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, ya 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.

3.1. Levantar 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 el siguiente 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 40, 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 recurriremos al box *"dummy"*.
    - *aws.keypair_name*: se trata de la clave utilizada para hacer login en las máquinas creadas. Este fichero nos lo descargamos al crearnos nuestra cuenta de *AWS*.

[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 que entra 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 que contiene otros recursos necesarios de la instancia.

```
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 = "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
29          # Cambiar tipo de instancia, por defecto coge una m3.medium
    (no gratuita, depreciada)
30          aws.instance_type = "t2.micro"
31          aws.ami = "ami-006a0174c6c25ac06"
32
33          # Grupo de seguridad que acepte conexiones SSH puerto 22
34          aws.security_groups = ['vagrant']
35
36          # Sobreescribir usuario y ruta de clave privada
37          override.ssh.username = "ubuntu"
38          override.ssh.private_key_path = "~/aws_repo/pgitic-aws.pem"
39      end
40  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*:

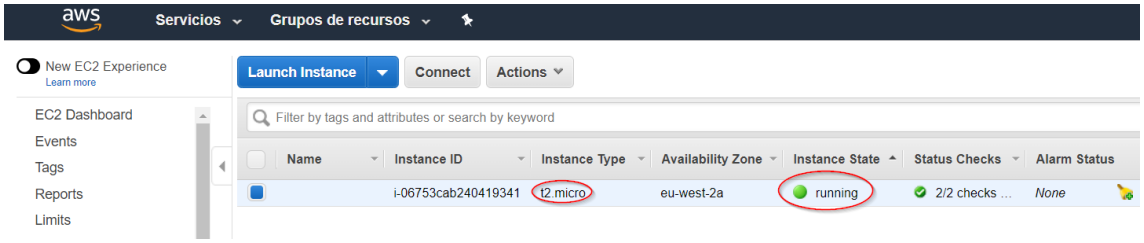
```
$ vagrant up --provider=aws
```

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* y veremos nuestra instancia creada satisfactoriamente. Otra forma de comprobarlo es mediante el comando **vagrant ssh**, en el directorio donde hayamos creado nuestro *Vagrantfile*.



3.2. Aprovisionar una instancia EC2 con Ansible y Vagrant

Tras haber levantado una máquina virtual en la nube con *Vagrant*, vamos a probar a aprovisionarla mediante *Ansible*. Para esto, comenzaremos modificando el contenido del *Vagrantfile* añadiendo las líneas de la 14 hasta la 17 que se muestran en el *Listado 3*:

```
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 = "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 *Vagranfile* actual con respecto al anterior se corresponde con el uso del sistema de aprovisionamiento que trae *Vagrant*, haciendo uso de *Ansible*. 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  ...
```

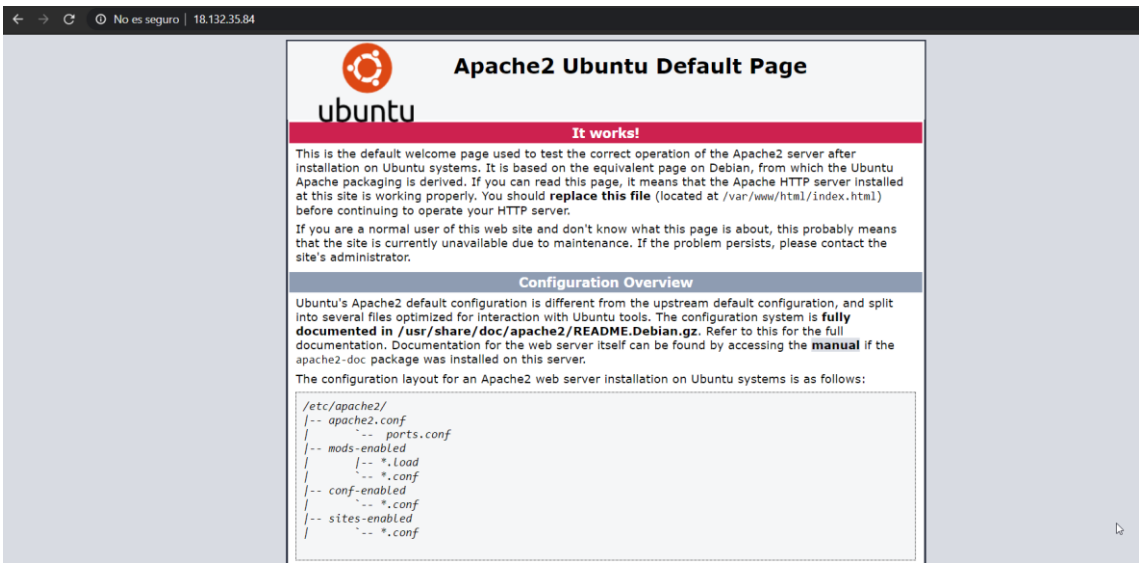
Gracias a este *playbook* de *Ansible*, podremos aprovisionar la máquina en la nube con el siguiente software:

Explicación y resultados

```
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>:80*, podemos comprobar que ha funcionado correctamente:



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*.

4. OTRAS POSIBILIDADES

Explicar otras posibilidades importantes que ofrece la herramienta y que no aparecen en el tutorial.

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

Por ejemplo, podemos utilizar *Ansible* para levantar máquinas en *AWS* sin necesidad de utilizar *Vagrant*. Para ello, podremos hacer uso de los diferentes módulos que posee *Ansible* para la gestión de recursos en *AWS*.

```
18  ---
19  - hosts: localhost
20    gather_facts: no
21    tasks:
22    # Basic provisioning example
23    - name: Crear instancia AWS EC2 t2.micro
24      ec2:
25        key_name: pgitic-aws
26        instance_type: t2.micro
27        image: ami-006a0174c6c25ac06
28        wait: yes
```

```
29          group: vagrant
30          count: 1
31          vpc_subnet_id: subnet-21199e5b
32          assign_public_ip: yes
33          region: eu-west-2
```

Listado 3. Código del playbook para crear una instancia t2.micro en AWS

Para ejecutar este *playbook*, lanzaremos el siguiente comando:

```
$ ansible-playbook
```

Inventarios dinámicos

Otro aspecto muy interesante tiene que ver con los **inventarios dinámicos** de *Ansible*, al hacer uso de máquinas en un suministrador 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** y que se encargará de actualizar directamente nuestro inventario dinámico, teniendo en cuenta que se encuentra instalada la librería de ansible y actualizando el cache 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 comprobar que funciona ejecutamos el siguiente comando:

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

Para que nos lo haga bien, actualizamos la cache.

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

Hecho esto, podemos lanzar un comando Ansible para verificar que se hace el ping correctamente a las máquinas levantadas:

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

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"
}
```

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*[7] 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*[8], 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.

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.

[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).
[8] Terraform. https://www.terraform.io/

6. REFERENCIAS

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

- https://github.com/mitchellh/vagrant-aws
- https://docs.aws.amazon.com/
- https://www.vagrantup.com/docs/
- https://docs.ansible.com/
- https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html
- 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 → *IAM*[9], hacemos click en Usuarios → Añadir usuario(s), tal y como se muestra en la *Figura 1*.

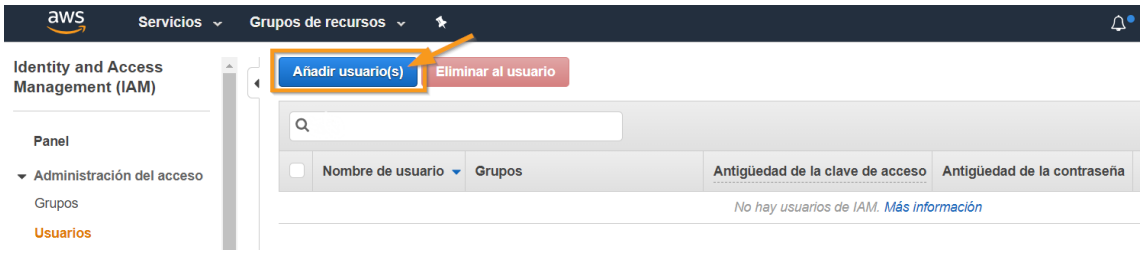


Figura 1. Añadir usuario en IAM de AWS

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 2*), 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*.

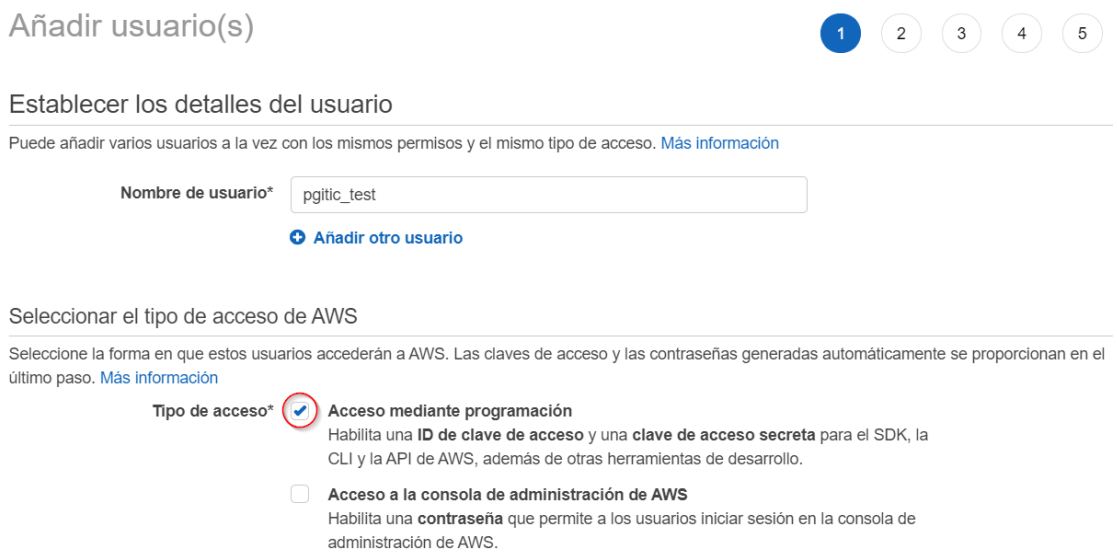


Figura 2. Crear usuario con tipo de acceso mediante programación

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 3*. Estas claves deberemos guardarlas en un lugar seguro y no perderlas, ya que si no nos quedaremos sin poder hacer uso de estas.

[9] IAM. Identity and Access Management.

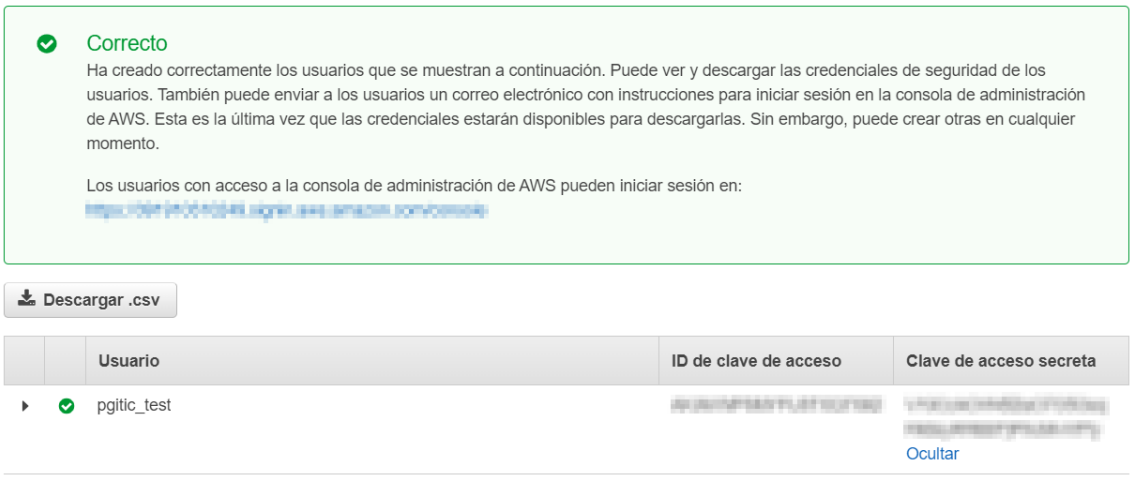


Figura 3. Claves de acceso para el usuario creado

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 4*, nuestro usuario *pgitic_test* ya se ha creado correctamente.

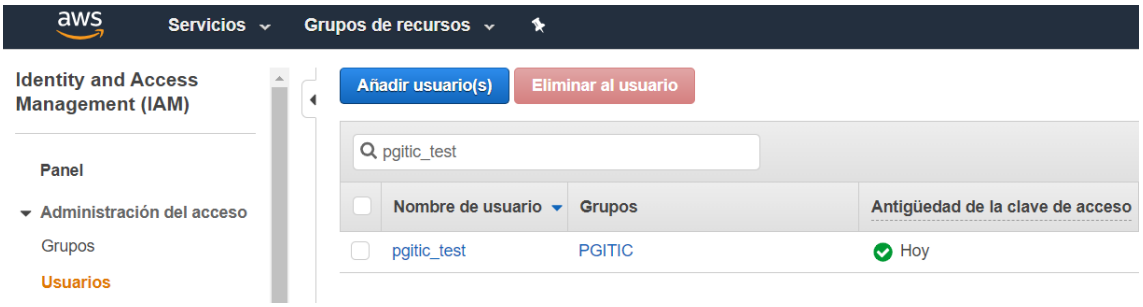


Figura 4. Usuario pgitic_test creado en IAM

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”.

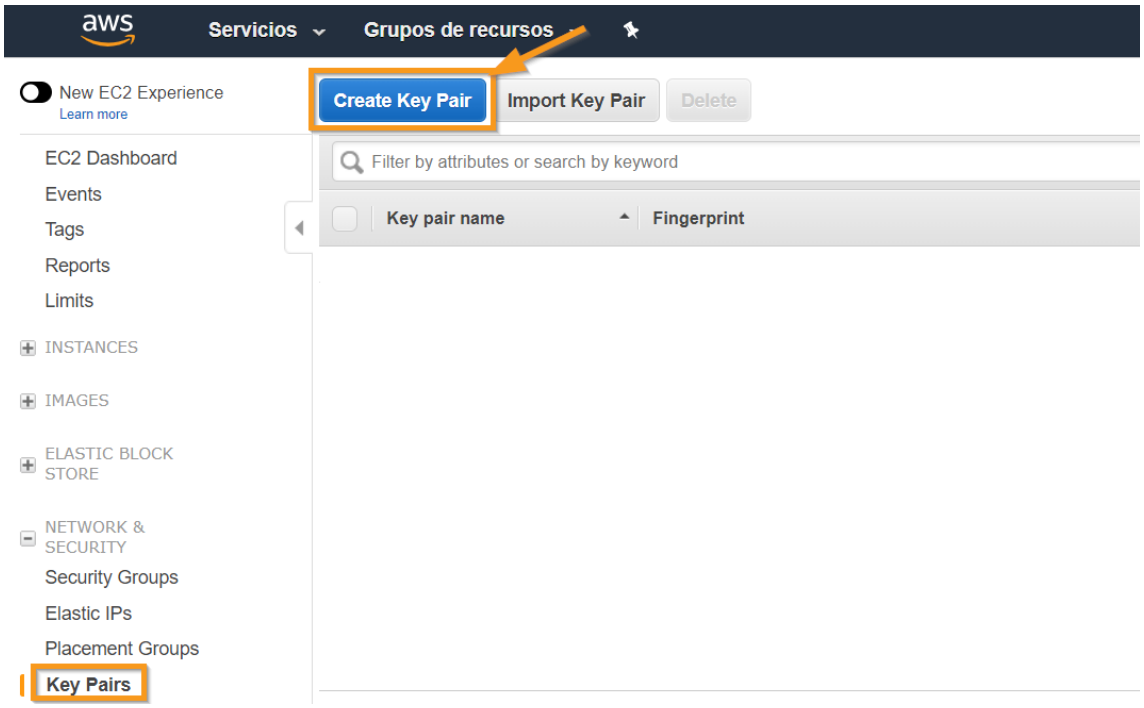


Figura 5. Crear par de claves para EC2

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

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
```

Si no instalamos dicho paquete, obtendremos el siguiente error:

```
$ ansible-playbook <nombre_playbook.yml>

PLAY [localhost]
*****************************************************************************
TASK [Crear instancia AWS EC2 t2.micro]
*****************************************************************************
fatal: [localhost]: FAILED! => {"changed": false, "msg": "boto required
for this module"}

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

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
```

Puede ser que nos salte un error de dependencias, para lo cual tendríamos que instalar también otro plugin con el siguiente comando:

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

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.

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='XXXXXXXXXXXXX'
$ export AWS_SECRET_ACCESS_KEY='XXXXXXXXXXXXXXXXXXXXXXXXX'
```

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 = XXXXXXXXXXXXX
aws_secret_access_key = XXXXXXXXXXXXXXXXXXXXXXXX
```

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:

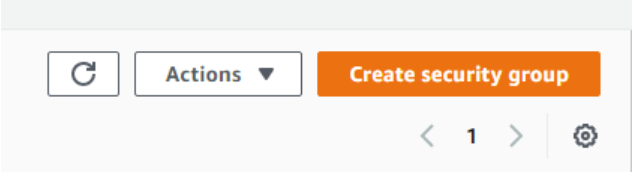
1. Configuramos la zona en la cual se desplegará nuestra instancia en el archivo **config**.
2. Rellenamos nuestras credenciales dentro del archivo **credentials**.
3. Ejecutamos el script ./configuration.sh

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

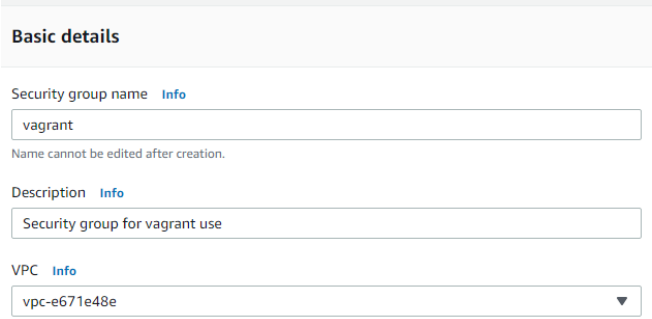
```
$ chmod +x configuration.sh
```

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

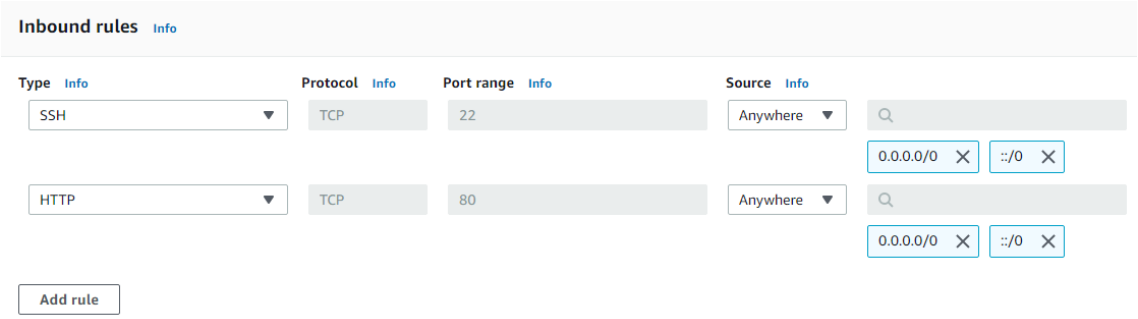
Para que nuestras instancias funcionen es necesario que se configuren en función a un grupo de seguridad, el cual determinará que puertos quedan abiertos de entrada, y desde que direcciones se pueden acceder a dichos puertos, también aplicable a las conexiones de salida.
Procedemos a entrar en la consola de administración de AWS. Una vez dentro nos dirigimos a Servicios → EC2 → Network and Security → Security Groups.
Pinchamos en "Create security group"



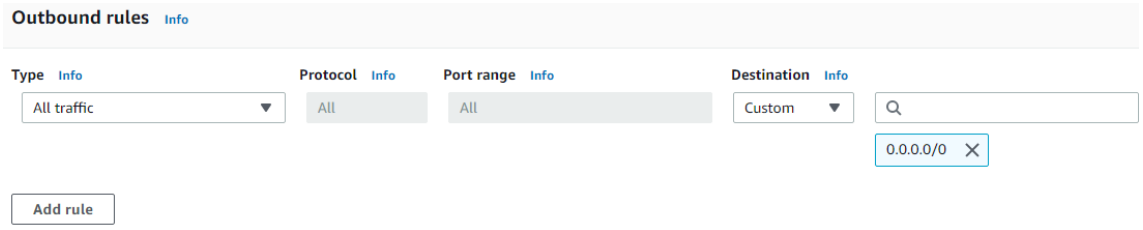
Rellenamos los datos, asignándole el nombre de **vagrant**, importante mantener dicho grupo para no tener que cambiarlo en el Vagrantfile. Seleccionamos el VPC que viene por defecto u otro si tenemos configurado uno aparte.



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.



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 la misma mediante Ansible.


Outbound rules Info
Type Info
Protocol Info
Port range Info
Destination Info
All traffic
All
All
Custom
0.0.0.0/0
Add rule