Auto Hosting

Hosting Automatizado con Ansible y Terraform

ANSIBLE

HashiCorp Terraform

Proyecto de fin de ciclo
CFGS ASIR
Francisco José Romero Morillo
Junio 2019

1. Introducción

Hoy en día cualquier persona o empresa tiene su propia página web, pero para que todo el mundo pueda acceder a esas páginas web se necesita de algún servicio que se encargue de guardar tu página y darle acceso a todo internet para que pueda ser visitada.

De eso se encargan los hostings, servicios que proveen a los usuarios de internet un sistema para almacenar información, imágenes, vídeo, o cualquier contenido accesible vía web.

Habitualmente detrás de los hostings web hay empresas con trabajadores, los cuales se encargan de administrar los servidores que guardan nuestras páginas web.

2. El Proyecto

2.1. Objetivo

El objetivo de mi proyecto es crear un pequeño hosting, con algunas de las funciones que suelen tener los hostings convencionales. En este hosting todo se realizara de manera automática, sin que el usuario ni administradores tengan que realizar las tareas de administración que conlleva un hosting.

2.2. Tecnologías

Para realizar este proyecto voy a utilizar dos tecnologías muy utilizadas en este momento: usare Ansible para automatizar la creación de las páginas web del hosting, y usare Terraform, para la creación de los servidores del hosting.

2.2.1. Ansible

Según la página oficial, Ansible es lo siguiente:

"*Ansible es un lenguaje universal, que desentraña el misterio de cómo se realiza el trabajo. Convierte tareas difíciles en libros de jugadas repetibles. Despliegue protocolos para toda la empresa con solo presionar un botón*."

Yo definiría Ansible como un software que te permite realizar instalaciones y configuraciones, múltiples veces, siempre realizando la misma lista de acciones sin cambios e incluso haciéndolo a múltiples equipos al mismo tiempo.

El funcionamiento de Ansible es simple: creamos unas listas de tareas a las cuales llamamos roles, los cuales ejecutamos sobre los servidores para que se realicen las instalaciones o configuraciones que necesitamos.

Ansible solo necesita dos cosas: conectarse por ssh a los equipos donde va a ejecutar los roles y que en esos equipos este instalado Python.

2.2.2. Terraform

Según la página oficial, Terraform es lo siguiente:

"Terraform es una herramienta que te permite crear, cambiar y mejorar la infraestructura de manera predecible y segura. Es una herramienta de código abierto que codifica las API en archivos de configuración declarativos que pueden compartirse entre los miembros del equipo, tratarse como código, editarse, revisarse y versionarse."

Yo definiría Terraform como un software que te permite administrar la configuración y estructura de tu infraestructura desde una terminal, guardando toda la información y el estado de la infraestructura en ficheros.

Podemos crear y administrar nuestra infraestructura con Terraform mediante providers, un intermediario entre Terraform y las APIs de los principales IaaS (por ejemplo: AWS, GCP, Microsoft Azure y OpenStack), PaaS (por ejemplo Heroku), y SaaS (por ejemplo: Terraform Enterprise, DNSimple y CloudFlare) del mercado.

Terraform tiene providers para cada una de esas plataformas, e incluso existen providers que no son oficiales: por ejemplo, un provider para usar VirtualBox como plataforma.

En Terraform solo tenemos que hacer dos cosas: definir los componentes de nuestra infraestructura (en Terraform se llaman recursos) los cuales pueden ser máquinas virtuales o redes por ejemplo y finalmente ejecutar Terraform para que se creen, modifiquen o eliminen. Así de sencillo.

2.3. Escenario

Mi proyecto es muy sencillo en cuanto al escenario, se compone de lo siguiente:

- Servidor principal: en esta servidor tendremos una aplicación web, desde la cuales podremos crear servidores mediante Terraform. Además de la aplicación web, este servidor tendrá dos componentes más: un servidor de base de datos y un servidor dns.
- Servidores: estos servidores serán los que se crearan mediante la aplicación web del servidor principal usando Terraform, en los cuales también habrá una aplicación web (similar a la del servidor principal), desde la cual podremos instalar las páginas web del servidor. Estas páginas se instalaran mediante Ansible.

Ambas aplicaciones web están creadas usando Python 3 como lenguaje de programación y bottle como framework.

Tanto el servidor principal del hosting, así como los diferentes servidores que se creen mediante su aplicación web, serán instancias de un IaaS muy usado ahora mismo: Amazon AWS.

Más adelante veremos cómo configuramos Terraform para usar Amazon AWS como provider.

3. Desarrollo

Mi proyecto ha pasado por varias etapas hasta llegar hasta estar terminado y en funcionamiento. En cada una de esas etapas he creado y desarrollado cada uno de los componentes que forman el proyecto.

Durante el desarrollo, mi entorno de pruebas fue una máquina virtual en VirtualBox, cuyo sistema operativo era Debian Stretch. En dicha máquina virtual desarrolle y probé los componentes hasta que lo he montado todo en instancias de Amazon AWS.

Los componentes que forman parte de este desarrollo son los siguientes: roles de Ansible, servicios (base de datos y dns), las aplicaciones web y la configuración de Terraform.

3.1. Roles de Ansible

Para que el proyecto funcione correctamente y realice las funciones que quiero he necesitado crear una serie de roles que ejecutaran las aplicaciones web.

Tenemos roles para dos cosas: para instalar la aplicación web de los servidores y para instalar las páginas web de los servidores.

Dichas páginas web son gestores de contenido (CMS), los cuales son tres para tener un CMS de cada tipo: Drupal (CMS de tipo blog), Prestashop (CMS de tipo shop) y Mediawiki (CMS de tipo wiki).

3.1.1. Roles: Panel de Control

Los roles del panel de control realizan dos funciones fundamentales:

- Primera función: preparar el servidor para la ejecución de Ansible y de la aplicación web del panel.
- Segunda función: instalar y poner en funcionamiento la aplicación web del panel.

Tenemos un rol llamado '*common*' para realizar la primera función y un segundo rol llamado '*install_panel*' para realizar la segunda función.

El rol '*common*' es sencillo: actualizara los paquetes del servidor donde se ejecute e instalara los paquetes necesarios para que la aplicación web y Ansible se ejecuten sin problemas.

El rol '*install_panel*' se encargara de instalar la aplicación web del panel de control en el servidor donde se ejecute. Para instalar la aplicación web seguirá los siguientes pasos:

1. Crear virtualenv para la ejecución de la aplicación.
2. Instalar paquetes necesarios con pip.
3. Copiar todos los ficheros de la aplicación a sus respectivos directorios.
4. Activar la aplicación web.

3.1.2. Roles: CMS

Tenemos diferentes roles para los tres CMS que he elegido, en total serán 6 roles, los cuales son de dos tipos:

- Primer tipo: este tipo de playbook se encargara de crear el directorio principal para la nueva página web y de crear el fichero de configuración (el site.conf en el directorio sites-availables) para el servidor web.
- Segundo tipo: este tipo de playbook se encargara de descargar e instalar el cms que hayamos elegido para crear la nueva página web.

El primer tipo de rol tiene como nombre '*hosting_NOMBRE*', donde '*NOMBRE*' es el nombre del CMS que se instalara. Por lo tanto, tenemos los siguientes roles: '*hosting_drupal*', '*hosting_prestashop*' y '*hosting_mediawiki*'.

El segundo tipo de rol tiene como nombre '*install_NOMBRE*', donde '*NOMBRE*' es el nombre del CMS que se instalara. Por lo tanto, tenemos los siguientes roles: '*install_drupal*', '*install_prestashop*' y '*install_mediawiki*'.

Durante los roles de instalación se realizaran las siguientes tareas:

1. Crear de la base de datos para el cms.
2. Crear del usuario para la base de datos.
3. Descargar del cms.
4. Copiar todos los ficheros del cms a sus respectivos directorios.
5. Instalar el cms usando una herramienta:
    a. Drupal: para instalar drupal usaremos drush, una herramienta que te permite instalar drupal desde una terminal.
    b. Prestashop: para instalar prestashop usaremos un script que viene en los ficheros de prestashop, el cual te permite instalar prestashop desde una terminal.
    c. Mediawiki: al igual que prestashop, instalaremos mediawiki usando un script que viene en los ficheros de mediawiki.
6. Poner en funcionamiento el cms.

3.1.3. Roles: Variables

Debido a que la instalación de las páginas web debe ser siempre diferente, habrá información que cambiara de una instalación a otra. Esa información, las preferencias del cliente, se guardaran en forma de variables que Ansible leerá al momento de ejecutar los roles.

Esas variables se rellenaran usando la aplicación web del panel de control y son las siguientes:

- *domain*: en esta variable guardaremos tanto el nombre del dominio web como el nombre del directorio principal de la página web.

- *mysql_db*: en esta variable guardaremos el nombre de la base de datos que usara el cms.
- *mysql_user*: en esta variable guardaremos el usuario que tendrá permisos sobre la base de datos del cms.
- *mysql_password*: en esta variable guardaremos la contraseña de acceso a la cuenta del usuario del cms.
- *site_name*: en esta variable guardaremos el nombre que veremos al entrar en la página web.
- *admin_user*: en esta variable guardaremos el nombre completo del administrador del cms.
- *admin_password*: en esta variable guardaremos la contraseña del administrador del cms.
- *name_user*: en esta variable guardaremos el nombre del usuario con el que accederemos al cms.
- *surname_user*: en esta variable guardaremos el apellido del usuario con el que accederemos al cms.
- *server*: en esta variable guardaremos el nombre que tendrá el servidor que creemos.

Estas variables se usaran durante la ejecución de los roles, pero eso no quiere decir que todas se vayan a usar en todos los roles. Hay variables que son para unos roles y otras variables para el resto, pero he decidido juntarlas todas en un solo fichero de variables.

3.2. Servicios necesarios

Para montar el hosting y ponerlo en funcionamiento he necesitado montar un par de servicios. He necesitado una base de datos para guardar toda la información y un servidor dns para facilitar el acceso a todo.

3.2.1. Servidor de base de datos

En este pequeño hosting será necesario guardar datos necesarios para su funcionamiento, datos de usuarios y de servidores que guardaremos en una base de datos. La base de datos está montada en el servidor principal.

Entre las opciones que tenemos para montar una base de datos he elegido mysql y serán dos tablas: una para usuarios y otra para los servidores. El contenido de dichas tablas es la siguiente:

- Tabla ‘Usuarios’: en esta tabla guardaremos todos los datos de los usuarios, siendo estos los datos que contiene la tabla: nombre, apellidos, dni, email, nombre de usuario y contraseña.
- Tabla ‘Servidores’: en esta tabla guardaremos todos los datos importantes de los servidores, esos datos son los siguientes: nombre, tipo, usuario (es decir, el dueño), dirección ip pública y contraseña del panel (Contraseña para acceder al panel de control).

El nombre de la base de datos, el usuario y la contraseña que usare para acceder a esta base de datos es 'hosting' en los tres casos. El script para crear la base de datos, el usuario y las tablas lo podemos encontrar en el repositorio de github, en un fichero llamado '*database_hosting*' dentro del directorio de la aplicación web del hosting.

3.2.2. Servidor de nombres (DNS)

Además de un servidor de base de datos ha sido necesario montar un servidor de nombres o DNS, pues aunque los servidores tendrán su propia dirección ip publica con la cual accederemos, se les asignara un nombre mediante la aplicación web del hosting.

El servidor dns está montado también en el servidor principal y para montarlo he decidido usar un servidor *dnsmasq*. Podría haber montado un servidor *Bind9*, pero no he sido capaz, ya que amazon da unas direcciones ip aleatorias, de rangos y redes diferentes cada vez.

Dnsmasq funciona de una manera muy sencilla: guardamos en el fichero */etc/hosts* de la maquina principal los servidores del hosting y cuando hagamos una consulta al servidor dns, este nos responderá leyendo los datos desde su */etc/hosts*.

3.3. Aplicaciones web

El funcionamiento del hosting se basa en la ejecución de dos aplicaciones web, las cuales he desarrollado en Python 3.

Para facilitar su desarrollo he utilizado un framework para Python: Bottle. Podría haber desarrollado las aplicaciones usando otro framework más avanzado como Flask o Django, pero he preferido utilizar Bottle, pues es un framework que se utilizar.

3.3.1. Aplicación: hosting

Esta aplicación web está montada en el servidor principal y será la aplicación con la cual crearemos los servidores del hosting.

Una vez accedamos a la aplicación, entraremos en la página principal (*Inicio*), donde nos dará la bienvenida y nos dirá que podemos hacer en la aplicación. Podremos ver los tipos de servidores disponibles para crear el nuestro en la pestaña '*Servidores*'.

Para crear nuestros servidores, la aplicación tiene varias características:

- ***Registro de usuarios***

  Para poder crear servidores desde la aplicación web necesitaremos una cuenta de usuario, la cual podremos crear desde el registro de usuarios que tiene la aplicación.

  Cuando accedemos a la zona de creación de usuarios (en la aplicación se entra en la pestaña '*Registro'*), lo primero que veremos será un formulario html con los siguientes campos: nombre, apellidos, dni, email, nombre de usuario y contraseña. Introducimos los datos que nos pide el formulario, le damos a '*Registrarse*' y nuestra cuenta de usuario estará creada.

Este registro funciona de la siguiente forma: la aplicación web recoge los datos del formulario html y los introduce en la base de datos mysql. Antes de registrar tus datos en la base de datos, codifica tu contraseña de acceso para evitar guardar la contraseña en claro.

Para cifrar la contraseña he utilizado el módulo de Python '*hashlib*', el cual te permite cifrar texto en varios algoritmos diferentes, a elegir entre algoritmos como sha1, sha224, sha256, sha384, sha512, md5 u otros algoritmos.

Entre los algoritmos disponibles para usar con hashlib he usado md5 para la aplicación web por ser un algoritmo cuyas cadenas cifradas son algo más cortas que las del resto de algoritmos.

Hashlib tiene muchas funciones, pero yo he utilizado tres de ellas:

- *new()*: esta función cifra una cadena de texto en un algoritmo que indiques. Se utiliza de la siguiente forma:

```
h = hashlib.new("md5",b"cadena")
```

- *md5()*: esta función cifra una cadena de texto en algoritmo md5. Hace lo mismo que la función *new*, pero es más rápida. Se utiliza de la siguiente forma:

```
h = hashlib.md5(b"cadena")
```

- *hexdigest()*: esta función devuelve la cadena cifrada en hexadecimal. Se suele utilizar para imprimir la cadena cifrada. Se utiliza de la siguiente forma:

```
h.hexdigest()
```

La forma en la que he usado hachlib en la aplicación web es la siguiente:

```
h = hashlib.md5(password.encode('utf-8')).hexdigest()
```

Una vez termine la aplicación y nuestra cuenta este creada, nos volverá a mandar a la pestaña de registro.

- ***Inicio de sesión***

Una vez tenemos creada nuestra cuenta de usuario es momento de iniciar sesión. En la aplicación web nos vamos a la pestaña '*Login*', introducimos los datos que nos pide (usuario y contraseña) un formulario html y al darle a '*Login*' iniciaremos sesión y la aplicación nos mandara a la pestaña del perfil de usuario.

Para mantener la sesión iniciada he utilizado un módulo de Python llamado '*bottle_session*', un administrador de sesiones que mantiene la sesión utilizando un cookie y almacenando un hash asociado con dicho cookie en una base de datos clave-valor redis.

Para utilizar Bottle_session tenemos que importar el módulo SessionMiddleware de beaker.middleware e instalar en el equipo el servidor de redis. Luego, añadir las siguientes líneas a la aplicación web:

```
session_opts = {
    'session.type': 'memory',
    'session.cookie_expires': 1200,
    'session.auto': True
}
app = bottle.app()
plugin = bottle_session.SessionPlugin(cookie_lifetime=600)
app.install(plugin)
```

Una vez hemos añadido dichas líneas al código de la aplicación web, debemos hacer lo siguiente:

1. En cada función que definamos en los *route* de la aplicación debemos añadir la variable '*session*' para indicar que vamos a usar *botte_session* en dicha función. Por ejemplo:

```
@route('/')
def inicio(session):
```

2. Debemos crear una nueva variable que es donde guardaremos el usuario que mantiene la sesión iniciada. En mi aplicación he utilizado la variable '*user*'. El valor por defecto de esta variable va a ser '*None*', para indicar que no hay una sesión iniciada.
3. Siempre que necesitemos usar la variable '*user*' en alguna plantilla html, debemos añadirla al *return template*.

El inicio de sesión funciona de la siguiente manera: la aplicación web recoge los datos introducidos en el formulario html, cifra la contraseña introducida en el formulario con hashlib, busca en la base de datos la contraseña para el usuario introducido en el formulario, compara ambas cadenas (la contraseña cifrada y la contraseña obtenida en mysql) y si ambas cadenas coinciden, inicia sesión.

Si las cadenas coinciden, la variable '*user*' pasara a ser el nombre del usuario que ha iniciado sesión:

```
session['name'] = username
```

Si las cadenas no coinciden, pasara a ser '*None*':

```
session['name'] = "None"
```

- ***Perfil de usuario***

  El perfil de usuario en la aplicación solo es visible si hemos iniciado sesión. Accederemos a ella en la pestaña '*Perfil*' y en esta podemos ver y administrar los servidores que tenemos creados en el hosting, si no tenemos ninguno nos mostrara un mensaje diciendo que podemos contratar uno nuevo en la pestaña '*Contratar*'.

  El perfil funciona de la siguiente manera: con la sesión iniciada, comprueba qué valor tiene la variable '*user*'. Si el valor es '*None*', nos mostrara el mensaje de contratar un nuevo servidor, sin embargo, si el valor es el nombre de usuario, nos mostrara una tabla con los servidores que tenemos contratados.

  Para mostrarnos los servidores, la aplicación hace una búsqueda de los servidores del usuario en la base de datos mysql y nos devuelve los datos en forma de tabla, dándonos dos opciones:

  - Administrar: consiste en un enlace que nos llevara al panel de control del hosting.
  - Dar de baja: consiste en unos comandos con Terraform que eliminaran el servidor.

- ***Crear servidores***

  Esta pestaña, al igual que la pestaña '*Perfil*', es una pestaña que solo es accesible una vez hemos iniciado sesión. Una vez accedamos a ella, nos encontraremos con un pequeño formulario html que nos pedirá dos datos: nombre del servidor, tipo de servidor y contraseña para el panel (esta contraseña es la que usaremos para acceder al panel de control del servidor).
  Una vez introduzcamos los datos que nos pide el formulario, le daremos a 'Contratar' y entonces se creara el servidor.

  Esta característica de la aplicación del hosting funciona de la siguiente manera: la aplicación web recoge los datos del formulario html, crea el servidor con Terraform (mas adelante veremos cómo se hace) y añade el servidor a dos sitios: a la base de datos mysql y al servidor dns.

  Una vez el servidor esta creado, la aplicación web nos mandara a nuestro perfil de usuario para que podamos administrar nuestros servidores.

- ***Desconexión***

  Esta característica también es solo visible cuando hemos iniciado sesión y tiene poco que explicar: sirve para cerrar nuestra sesión de usuario.

  La aplicación web hace lo siguiente para cerrar nuestra sesión: le da a la variable '*user*' el valor '*None*' y nos manda a la página principal.

- ***Páginas de error***

  Durante la creación de usuarios o de servidores puede darse una situación: que queramos crear un usuario o servidor que ya "existe": que el dni que hemos elegido para el nuevo usuario este en uso o que el nombre del servidor ya lo tenga otro servidor.

  Para evitar problemas, la aplicación web comprueba si existe el dni del usuario o el nombre del servidor (haciendo un select en la base de datos) y si ya existen, nos mandara a una página de error, notificándonos que los datos ya existen y que por favor los cambiemos.

3.3.2. Aplicación: panel

Esta aplicación web estará montada en cada servidor mediante los roles "*common*" y "*install_panel*", los cuales se ejecutaran al crear un servidor desde la aplicación del hosting. Con esta aplicación instalaremos los cms que serán las páginas de dicho servidor.

Para poder acceder a la aplicación, debemos acceder a la siguiente dirección: "*servidor.autohosting.com/panel*" y al entrar nos encontraremos con un formulario de inicio de sesión.

Tendremos que introducir los datos de acceso (el usuario es "*webmaster*" y la contraseña es la que introducimos en la creación del servidor), y si son correctos, nos mandara entonces a la página principal del panel de control. En la página principal nos dirán lo que podemos hacer en la aplicación: administrar el servidor, haciendo diferentes tareas como instalar los cms o ver los logs.

En esta aplicación no tendremos tantas características como la del hosting, pero tendremos lo siguiente:

- ***Instalación de CMS***

  La principal característica de cada servidor será alojar las páginas web del cliente, las cuales serán gestores de contenido, en el caso de este hosting, podrán ser un drupal, un prestashop o un mediawiki.

  Para instalar estos cms accederemos a la pestaña 'Software', donde veremos botones que nos llevaran al menú de instalación de cada uno de estos gestores de contenido.

  Si accedemos al menú de cualquiera de los tres gestores, veremos un formulario html que nos pedirá los datos necesarios para la instalación de dicho gestor. En los tres gestores los datos serán los mismos, serán los datos que utilizaremos en los roles, que son los siguientes: base de datos, usuario de la base de datos, clave de acceso del usuario, dominio, nombre del sitio web, nombre del administrador, clave del administrador, nombre y apellido del usuario.

La instalación del gestor de contenidos es sencilla: la aplicación web recogerá los datos introducidos en el formulario html, y los guardara en el fichero de variables de Ansible. Luego ejecutara el rol de Ansible correspondiente con el comando '*ansible-playbook*'.

- ***Visualización de logs***

  Una característica muy cómoda de un panel de control es la posibilidad de ver los logs de los diversos servicios instalados en el servidor.

  Podremos acceder a esta característica en la pestaña 'Logs', donde nos mostrara tres botones: un botón para ver los logs de acceso de apache, otro botón para ver los logs de errores de apache y otro botón para ver los logs de errores de mysql.

  Esta característica es sencilla: lee los ficheros de log de cada servicio y los imprime por pantalla.

- ***Desconexión***

  Esta característica es exactamente como la de la aplicación web del hosting, cierra nuestra sesión para que no sea posible acceder a las funciones del panel de control.

  Como en la aplicación web del hosting, hace lo siguiente para cerrar nuestra sesión: le da a la variable 'user' el valor 'None' y nos manda a la página principal.

Ambas aplicaciones web están adaptadas para ejecutarse con apache2 y wsgi, para ponerlas en funcionamiento hay que hacer lo siguiente:

1. Crear un fichero cuyo nombre será el nombre de la aplicación, con extensión wsgi, cuyo contenido es el siguiente:

```
sys.path = ['/var/www/html/aplicacion/'] + sys.path

os.chdir(os.path.dirname(__file__))

# Inicialice app with SessionMiddleware environ

application =
beaker.middleware.SessionMiddleware(bottle.default_app(),
aplicacion.session_opts)
```

   Donde pone "aplicación" pondremos el nombre de la aplicación web, además deberemos importar los siguientes módulos de Python: sys, os, Bottle, beacker.middleware y "aplicación" (la aplicación la importaremos como un módulo).

2. Añadiremos las siguientes líneas al virtual host de la aplicación web:

```
WSGIDaemonProcess bottle user=www-data group=www-data
python-home=/home/admin/bottle python-
path=/var/www/html/aplicacion
```

```
WSGIProcessGroup bottle
WSGIScriptAlias / /var/www/html/aplicacion/aplicacion.wsgi
<Directory /var/www/html/apliacion/>
	Require all granted
	<Files aplicacion.wsgi>
		Require all granted
	</Files>
</Directory>
```

3. Crearemos el virtualenv que usaran las aplicaciones web, el cual se llamara "Bottle" y estará en el directorio home del usuario. Luego debemos importar los paquetes necesarios desde el fichero requeriments.txt de la aplicación web:

```
virtualenv -p /usr/bin/python3 bottle
source bottle/bin/activate
pip3 install -r requeriments.txt
```

3.4. Funcionamiento de Terraform

Ya explicamos antes que es Terraform y para que lo vamos a usar en el proyecto: para crear los servidores del hosting. Ahora vamos a ver cómo funciona Terraform y como lo usaremos en el proyecto.

3.4.1. Funcionamiento

Para poder usar Terraform, debemos descargarnos el programa desde la página oficial, entrando en: https://www.terraform.io/downloads.html y bajarnos la versión necesaria para nuestro sistema operativo.

En mi caso me descargue la versión para Linux de 64 bits, y la puse dentro de un directorio llamado "*terraform*", dentro del directorio "*home*".

Una vez hemos descargado el programa, debemos crear un directorio donde estarán los ficheros de configuración de Terraform: el *main.tf* y los ficheros de estado. Estos ficheros, en la aplicación web del hosting, están en el directorio root de la aplicación web, dentro de un directorio llamado "*terraform*".

Dentro de dicho directorio, creamos el fichero main.tf (los ficheros de estado se crean solos al ejecutar Terraform), donde deberemos añadir dos cosas: las credenciales de acceso al provider que vayamos a usar (en mi caso, Amazon AWS) y los recursos que vayamos a crear (redes, instancias, etc.).

Una vez tengamos creado el fichero main.tf, es hora de ejecutar los comandos de Terraform, que son los siguientes:

- Terraform init: se usa para inicializar un directorio de trabajo que contiene los archivos de configuración de Terraform. Este es el primer comando que debe ejecutarse después de escribir una nueva configuración de Terraform o clonar una existente desde el control de versiones.
  Es decir, este comando prepara el directorio para la ejecución de Terraform.
- Terraform plan: se utiliza para crear un plan de ejecución. Terraform realiza una actualización, a menos que esté explícitamente desactivada, y luego determina

qué acciones son necesarias para lograr el estado deseado especificado en los archivos de configuración.
Es decir, nos muestra que cambios se realizaran, si se crearan o borraran recursos.

- Terraform apply: se utiliza para aplicar los cambios necesarios para alcanzar el estado deseado de la configuración.
  Es decir, hace los cambios planeados en el plan de ejecución: crea o borra los recursos que están planeados.

Con estos tres comandos podemos controlar Terraform perfectamente, hay más comandos útiles para manejar Terraform, como los que te permiten ver los datos de un recurso:

- Terraform state list: se utiliza para enumerar los recursos que hay listados en los ficheros de estado de Terraform.
  Este comando nos mostrara los recursos que hemos creado con el comando apply.
- Terraform state show: se utiliza para mostrar los atributos de un recurso de Terraform.

El comando Terraform state show se utiliza en la aplicación web del hosting para obtener la dirección ip pública del servidor.

La configuración de las credenciales del provider y los datos de los recursos dependen del provider que estemos usando, por eso voy a hablaros de cómo funciona en el caso de Amazon AWS.

3.4.2. Amazon AWS

Amazon Web Services es una plataforma de Amazon en la que podemos acceder a una colección de servicios de computación en la nube pública, entre ellos la posibilidad de crear instancias en las que poder montar nuestros servidores.

AWS te permite crear una cuenta gratuita con la cual probar sus servicios, y dicha cuenta es la que estoy usando para el proyecto.

Para poder utilizar AWS con Terraform, debemos añadir dos claves de acceso: el ID de clave de acceso y la clave de acceso secreta, dos claves que podemos crear en el menú de usuario de AWS, dentro de "*Mis credenciales de seguridad*".

Dichas claves se añaden al fichero main.tf de Terraform de la siguiente manera:

```
provider "aws" {
  access_key = AWS_ID_KEY
  secret_key = AWS_SECRET_KEY
  region     = AWS_REGION
}
```

Una vez hemos creado dichas credenciales, debemos añadir los recursos que queremos crear.

3.4.3. Terraform en la aplicación

Una vez tenemos todo listo para ejecutar Terraform, debemos ver como ejecuta Terraform la aplicación web del hosting. La aplicación sigue los siguientes pasos:

1. Añade el recurso que queremos crear al main.tf. El formato de los recursos cambia dependiendo del tipo de recurso, las instancias se crean siguiendo un formato, el cual se puede ampliar añadiendo más campos. Un ejemplo (el que usa la aplicación web) sería el siguiente:

```
resource "aws_instance" "nombre_servidor" {
instance_type = "t2.micro"
ami = "ami-023143c216b0108ea"
key_name = "amazon"
vpc_security_group_ids = ["sg-45cbb829"]
tags = { Name = "nombre_servidor" }
}
```

   Explicare que es cada cosa en la declaración del recurso:

   - Aws_instance: es el tipo de recurso que vamos a crear, en este caso, una instancia de amazon aws.
   - Nombre_servidor: es el nombre que tendrá el recurso en Terraform.
   - Instance_type: son las características de la instancia (ram, disco duro… etc). Si el recurso se creara en openstack seria parecido a lo que llamamos un "sabor".
   - Ami: es la imagen del sistema operativo que vamos a usar. Hay ami's gratuitas, de amazon y de la comunidad. También podemos crear las nuestras propias.
   - Key_name: es la clave ssh que se usara para acceder a la instancia.
   - Vpc_segurity_group_ids: es el grupo de seguridad que tendrá la instancia, es decir los puertos que tendrá abiertos.
   - Tags (Name): es el nombre que recibirá la instancia en amazon.
2. Una vez se ha añadido el recurso al fichero main.tf, se ejecuta un *Terraform plan* y seguidamente un *Terraform apply*.

4. Conclusiones

Tras realizar este proyecto he visto que con unas pocas herramientas y un poco de trabajo se puede montar un pequeño hosting que funcione adecuadamente.

Trabajar con Terraform es muy interesante, tiene grandes capacidades y puede usarse en muchas plataformas diferentes, lo cual facilita mucho su uso. He hablado poco del funcionamiento de Terraform, pero hay mucho más allá de lo que he usado en este proyecto.

Podemos reutilizar nuestro código de Terraform creando módulos e importándolos en nuestros nuevos proyectos. Podemos usar Packer, otra herramienta de los desarrolladores de Terraform, una herramienta con la cual podremos crear nuestras propias ami's.

Trabajar con Ansible también es muy cómodo, una vez tienes tus roles creados solo tienes que ejecutarlos para que se realicen las instalaciones y configuraciones que has definido en dichos roles. Te facilita mucho el trabajo que puedas ejecutar el mismo playbook en diferentes equipos al mismo tiempo sin límites, lo cual te permite hacer las cosas una sola vez, no teniendo que hacer la misma tarea en cada equipo.

Con este proyecto he podido aprender una nueva herramienta que antes no había usado nunca (Terraform) y profundizar en el uso de Ansible, que aunque ya sabía usarlo he aprendido mucho durante el transcurso del proyecto.

5. Después del proyecto

Tres meses para desarrollar un proyecto puede ser poco tiempo, menos tiempo quizás si el objetivo es desarrollar aplicaciones web, las cuales necesitan tiempo y dedicación para su creación. Durante el transcurso del proyecto me he encontrado con muchos problemas y dudas que he ido resolviendo, y muchas ideas que he ido añadiendo al proyecto. También hubo ideas que se quedaron en eso, ideas que podría haber añadido.

Hay muchas cosas que pueden cambiarse en este proyecto: cosas que pueden mejorarse, cosas que pueden cambiarse por otras y cosas que pueden añadirse. Un par de ejemplos de cambios que podrían hacerse son:

1. La aplicación web del hosting te permite crear servidores, eligiendo nombre y tipo, hubiera estado muy bien la opción de que también nos permitiera añadir una clave ssh con la cual pudiéramos acceder mediante ssh y administrar el servidor nosotros mismos.
2. La aplicación web del panel no te permite acceder a los ficheros, pues no tiene un servidor ftp, hubiera estado muy bien que los servidores llevaran un servidor ftp para acceder a ellos, e incluso un phpmyadmin para administrar las bases de datos.

En definitiva, hay muchas cosas que se pueden mejorar en el proyecto, mejorar las aplicaciones, mejorar los roles, añadir funcionalidad… pero eso es algo que ya se quedara fuera de este proyecto.