

TRABAJO FIN DE GRADO

Automatización de arquitectura T.I con Ansible

ASIR

Cosmina Nicolau



Centro de estudios superiores Afuera
Madrid, 2020

INDICE GENERAL

Agradecimientos.....	4
Resumen.....	5
Introducción y objetivos	6
Contexto y comparativas	7
Marco teórico	8
Planificación del proyecto	11
Estimación de recursos del proyecto	11
Tecnologías	12
Herramientas para el desarrollo del software.	13
Marco práctico	16
LAMP	16
Funcionamiento de Ansible	17
Conceptos básicos	17
Implementación.....	20
Configuración del servidor.....	21
Instalación Ansible.....	23
Configuración host remoto	24
Diseño e implementación de código	26
Juego de pruebas.....	45
Análisis de los resultados obtenidos y puntos de mejora	47
Análisis de las competencias adquiridas	47
Conclusión	48
Bibliografía	49
Anexo	50

INDICE DE FIGURAS

Ilustración 1. Ciclo DevOps	7
Ilustración 2. Arquitectura Ansible	9
Ilustración 3. Gráfico popularidad	10
Ilustración 4. Logo VirtualBox.....	12
Ilustración 5. Logo GitHub	13
Ilustración 6. Imagen llaves SSH GitHub	14
Ilustración 7. Link SSH repositorio GitHub	14
Ilustración 8. Logo Visual Studio Code	15
Ilustración 9. Estructura TFG	16

INDICE DE TABLAS

Tabla 1. Diferencias herramientas automatización	10
Tabla 2. Listado de tareas iniciales del proyecto	11
Tabla 3. Especificaciones Raspberry Pi 3	11
Tabla 4. Especificaciones máquina virtual cliente	12

AGRADECIMIENTOS

Quisiera comenzar dando mi agradecimiento a *Óscar Soto Martín*, por acompañarme durante todo el desarrollo de este proyecto desde su principio hasta su fin.

A todos los profesores, por enseñarme todo lo que se y por poder aplicar dichos conocimientos y llevarlos a cabo tanto en este trabajo como en las prácticas.

Gracias también por vuestra disposición y ayuda durante todo el desarrollo y elaboración del TFG, en especial a *Marta Ramírez Trillas* por introducirme en este asombroso mundo de la automatización con Ansible.

Gracias a *Rocío Gutiérrez González* por su gran labor como tutora a lo largo de este curso y por su apoyo durante la realización de este proyecto.

A *Richard Hidalgo Lorite* por su colaboración y ayuda en todo momento con *Ansible*.

A mis compañeros de clase, por todos los buenos y malos ratos que hemos vivido y sobre todo por haber superado juntos esta maravillosa etapa de nuestra vida.

A Rubén Martín Romero por todo su tiempo invertido en aconsejarme y ayudar a rectificar de la mejor forma posible.

Y finalmente, al resto de amigos, familiares y pareja por su apoyo y perseverancia desde siempre.

Gracias.

RESUMEN

La idea principal de este proyecto consiste en obtener un mayor conocimiento y control sobre la automatización de las infraestructuras T.I así como la gestión de configuraciones y despliegue de aplicaciones.

Efectuando una valoración de los objetivos y requisitos, se ha realizado un análisis previo para la implementación de un sistema de automatización, teniendo en cuenta aspectos claves como la funcionalidad y la seguridad.

La automatización se realiza mediante *Ansible*¹, de manera que, desde un servidor, podamos llevar a cabo la gestión de diferentes infraestructuras y equipos.

El primer paso a efectuar es la configuración e implementación del servidor en una *Raspberry pi*², en la que contendrá el software de *Ansible* para administrar los distintos nodos posibles que tendremos a nuestra disposición.

Una vez configurado el servidor, pasaremos a la creación de los clientes o servidores necesarios para el proyecto mediante máquinas virtuales en *Virtual Box*³. En dichas máquinas realizará el despliegue de aplicaciones.

A lo largo de todo el desarrollo de este proyecto se han tenido en cuenta la obtención de datos críticos sobre el funcionamiento de servicios, y tras una sucesión de distintas pruebas realizadas a lo largo de la investigación de este trabajo, se puede ofrecer un resultado satisfactorio que cumple con los objetivos previstos inicialmente.

¹ Plataforma de software libre para configurar y administrar computadoras.

² Ordenador de placa reducida.

³ Software de virtualización de sistemas operativos.

INTRODUCCIÓN Y OBJETIVOS

Actualmente, en el campo de las nuevas tecnologías han ido apareciendo nuevos conceptos como el *Cloud Computing* ⁴ o el *Big Data* ⁵. Estas apariciones incitan a pensar si realmente estamos ante la aparición de nuevas tecnologías innovadoras, o de lo contrario, seguimos ante las mismas tecnologías ya conocidas, pero con distinto nombre.

Esta cuestión es bastante fundamental ya que muchas empresas buscan la forma de innovar con las tecnologías más competitivas del sector, apostando por dichas tendencias mencionadas anteriormente.

Todas las empresas administran grandes cantidades de servidores y de clientes haciendo necesario el uso de una herramienta capaz de automatizar el despliegue y aprovisionamiento de dichas máquinas y la automatización de procesos repetitivos.

Para ello surge la aparición de herramientas de automatización, como pueden ser *Chef* o *Puppet*, pero en este caso nos centraremos en *Ansible*.

Ante la necesidad de solventar dichos problemas surge la idea de este TFG, con el objetivo de utilizar *Ansible* combinándola con otras herramientas que ya conocemos y realizando la implantación de un servidor *LAMP* ⁶.

En conclusión, el objetivo principal de este trabajo consiste en desarrollar un sistema capaz de automatizar la configuración y el despliegue de diferentes tecnologías en infraestructuras T.I.

⁴ Computación en la nube que permite ofrecer servicios a través de la red.

⁵ Conjunto de información con datos de diferentes tipos.

⁶ Sistema de infraestructura de internet basado en Linux, Apache, MariaDB o MySQL y PHP.

CONTEXTO Y COMPARATIVAS

En este apartado se va a analizar la importancia de la automatización y de la seguridad, así como la explicación de la evolución y situación actual de las prácticas de *DevOps*.

DEVOPS

Para comprender mejor el concepto de la automatización y las herramientas que automatizan los procesos nos interesa entender en qué consiste el término *DevOps*.

DevOps es una metodología de trabajo que se centra en la colaboración, comunicación e integración entre desarrolladores de software y el resto de los profesionales TIC. El objetivo principal es ayudar a que el desarrollo del software sea mucho más eficiente, lo que reduce el tiempo de comercialización y aumenta la productividad.

Persigue el objetivo de unificar la integración entre el equipo de desarrollo (DEV) y el equipo de sistemas (OPS). Esta metodología ha sido reconocida por muchos expertos como la mejor manera de promover la cooperación entre equipos de desarrollo y operaciones. Consiste en la práctica de ingenieros de operaciones y desarrollo que trabajan juntos en todo el ciclo de vida de un proyecto, desde el diseño hasta el lanzamiento y soporte.

La parte de desarrollo comienza las tareas desde cero y sigue las siguientes pautas:

- Establecer requisitos.
- Construcción e integración de software.
- Etapa experimental de pruebas o testeo.
- Una vez superadas estas pruebas, el software se pasa al grupo de operaciones.

El equipo de operaciones se encarga de las labores de despliegue y mantenimiento del software, es decir, ofrecen un feedback de forma continua.

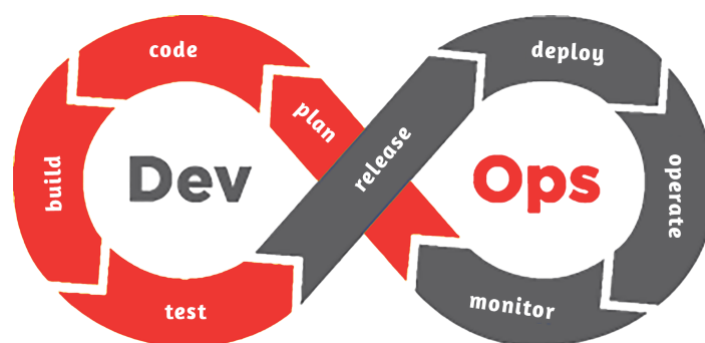


Ilustración 1. Ciclo DevOps

DevOps ofrece una mayor velocidad de innovación que permite servir mejor a los clientes, además de que se adapta a las condiciones cambiantes del mercado para generar mejores resultados comerciales.

AUTOMATIZACIÓN

Los procesos de automatización en *DevOps* ayudan a desarrollar código, testearlo y desplegarlo. Cuanto antes se realicen dichas actividades, antes se obtendrá un feedback del cliente que permitirá mejorar el producto o el servicio.

Las empresas que implementan la automatización reducen los costes operativos e incrementan la velocidad y confiabilidad de las tareas de soporte y desarrollo, además de proporcionar una mejora de análisis.

Para abordar los desafíos que se proponen, se debe adoptar un flujo de trabajo, procesos, tecnologías y protocolos para reducir los riesgos asociados y aumentar el potencial para automatizar los procesos manuales.

Gracias a la automatización los procesos se completan de forma más rápida y eficiente, además de conseguir evitar posibles errores humanos ya que el hecho de realizar a mano diversas tareas puede desembocar en fallos durante el procedimiento.

Existen distintos tipos de automatización de procesos, en nuestro caso, el que vamos a realizar con *Ansible*, se basa en el aprovisionamiento automatizado, que consiste en proporcionar un estado deseado a las máquinas con las que se van a trabajar.

Uno de los primeros pasos para establecer un entorno *DevOps* consiste en escoger la herramienta que utilizaran en ambas partes, tanto los de desarrollo como los de sistemas, que son los que se van a detallar en el siguiente apartado.

MARCO TEÓRICO

HERRAMIENTAS DE AUTOMATIZACIÓN

A la hora de facilitar la realización de las tareas repetitivas a las que nos podemos enfrentar existen diversas herramientas que ayudan a gestionar estos procesos a que se automaticen y, por lo tanto, trabajar con mayor eficacia.

PUPPET

Puppet consiste en una herramienta *OpenSource*⁷ (escrita con *Ruby*⁸) que se basa en un lenguaje de dominio propio que permite generar estados finales en los servidores por medio de “manifests”⁹, además da soporte de monitorización de cambios, lo que quiere decir que permite configurar como queremos que este el servidor en un momento determinado. Tiene soporte en Windows y Linux.

En resumen, se utiliza para configurar, administrar, implementar, orquestar diversas aplicaciones y servicios en toda la infraestructura de una organización.

⁷ Hace referencia al código abierto (modelo de desarrollo de software basado en colaboración abierta).

⁸ Lenguaje de programación interpretado, reflexivo y orientado a objetos.

⁹ Es un archivo con los metadatos de un grupo de archivos adjuntos que forman parte de un conjunto o unidad coherente.

CHEF

Se trata de una herramienta de automatización que proporciona una manera de definir la infraestructura como código, esto último significa que administra la infraestructura escribiendo código en lugar de usar procesos manuales.

Este software utiliza un lenguaje de dominio específico denominado DSL para escribir configuraciones de sistema.

ANSIBLE

Ansible es una plataforma de automatización de código abierto que automatiza el aprovisionamiento de software, la gestión de configuraciones y el despliegue de aplicaciones, lo que viene a ser una herramienta de orquestación.

Ansible permite gestionar los distintos nodos a través de SSH, para su uso solo es necesario Python en el servidor remoto en el que se va a ejecutar para poder utilizarlo. Trabaja con ficheros YAML para realizar las configuraciones que se propagan por los distintos nodos.

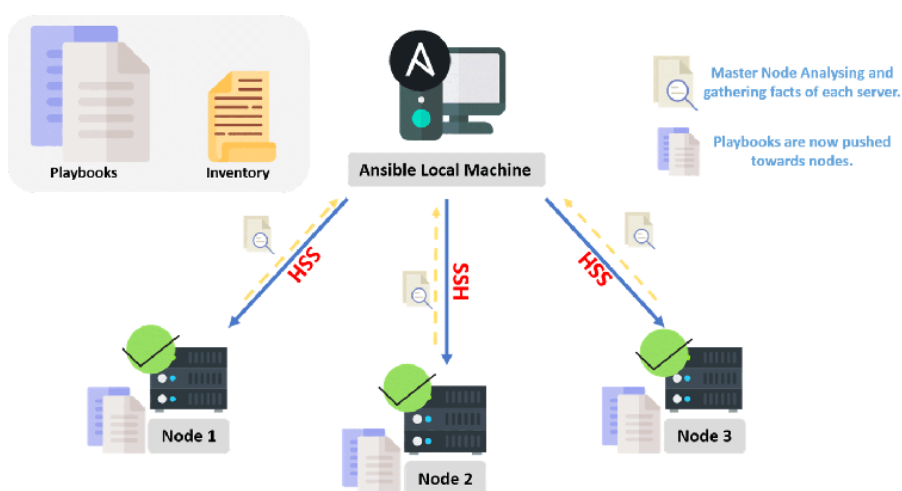


Ilustración 2. Arquitectura Ansible

A diferencia de las anteriores herramientas que se han ido detallando anteriormente, Ansible no utiliza un agente en el *host* remoto, usa SSH (necesario en todos los nodos que hay que administrar).

Ansible puede automatizar distintos tipos de tareas:

- Tareas de aprovisionamiento: que consisten en configurar los distintos servidores que necesita una infraestructura.
- Gestión de la configuración: modifica la configuración de una aplicación, sistema operativo o dispositivo. También es capaz de iniciar y detener servicios, instalar o actualizar aplicaciones e implementar políticas de seguridad.
- Implementación de aplicaciones: hace más fácil automatizar la implementación de aplicaciones desarrolladas internamente en sistemas de producción.

¿POR QUÉ ANSIBLE?

La principal razón de escoger esta herramienta es la simplicidad. Puede considerarse una herramienta “todo en uno”, es decir, todos los pasos de automatización se pueden realizar con *Ansible*, esto se debe a que está desarrollado en Python y además utiliza miles de paquetes existentes de la comunidad de Python para crear sus propios módulos.

Actualmente cuenta con 1300 módulos en varias áreas de la infraestructura T.I como por ejemplo módulos web, de base de datos, de red, nube y Windows entre otros.

Tiene una tasa de aprendizaje muy rápida, con fácil instalación y configuración inicial. Está basado en lenguaje YAML, se trata de un lenguaje de configuración simple, en cambio *Puppet* y *Chef* usan Ruby, que es bastante más complicado de aprender.

	Lenguaje	Aprendizaje	Agentes	Configuración
Ansible	Python	Normal/Sencillo	No	Playbooks ¹⁰ (YAML)
Puppet	Ruby	Difícil	Sí	Manifest (Ruby)
Chef	Ruby	Muy difícil	Sí	Cookbooks (Ruby)

Tabla 1. Diferencias herramientas automatización

Ansible está repleto de características que proporcionan muchas eficiencias de tiempo y recursos para una infraestructura T.I. Además, no requiere agentes en servidores remoto, esto implica que lo convierte en una herramienta segura y elimina los posibles problemas de compatibilidad entre las versiones de servidor y agente.

Asimismo, no necesita mantener abiertas las conexiones entre la máquina que se encarga de la administración y el nodo central.

Como podemos apreciar en el siguiente gráfico, *Ansible* muestra una tendencia creciente.

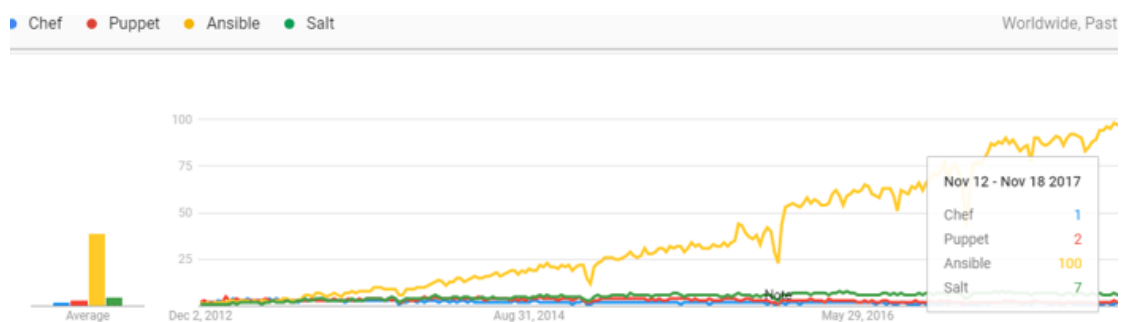


Ilustración 3. Gráfico popularidad

¹⁰ Libro de jugadas donde se definen las tareas en Ansible en formato YAML

PLANIFICACIÓN DEL PROYECTO

DEFINICIÓN DE TAREAS

A continuación, se muestra un desglose de las tareas propuestas inicialmente para alcanzar los objetivos propuestos del proyecto.

Tarea	Descripción
1	Configuración Raspberry Pi e instalación S.O como servidor.
2	Instalación y configuración de Ansible en el servidor.
3	Configuración máquina virtual cliente.
4	Creación de los ficheros para la implantación del LAMP
5	Establecer conexión entre <i>host</i> remoto y servidor por SSH intercambiando claves.
6	Realizar implantación del LAMP en el servidor.

Tabla 2. Listado de tareas iniciales del proyecto

ESTIMACIÓN DE RECURSOS DEL PROYECTO

Como acabamos de definir en el apartado anterior, tenemos un servidor implantado en una Raspberry Pi con Ansible interactuando con una máquina virtual.

A continuación, se van a detallar los recursos necesarios para poner en marcha dicha implementación:

CÁLCULOS DE MEMORIA, DISCO Y ELECCIÓN S.O

RASPBERRY PI 3

S.O	Ubuntu Server 20.04 LTS
RAM	1 GB
SD	32 GB

Tabla 3. Especificaciones Raspberry Pi 3

Se utiliza la versión sin entorno gráfico, esto implica que toda la administración se realiza bajo líneas de comando, lo que permite que los recursos hardware se utilicen de forma más óptima y la gestión de las tareas automatizadas sean más precisas y rápidas.

MÁQUINA VIRTUAL CLIENTE

S.O	Debian 9.12
RAM	2 GB
Disco	10 GB

Tabla 4. Especificaciones máquina virtual cliente

Se implanta un S.O Debian por diversas razones, la principal es porque permite disponer de paquetes estables y en cuanto a las instalaciones, ofrece más opciones de personalización.

Está disponible para más arquitecturas, por lo cual puede ser ejecutada en más equipos que Ubuntu y además de que es más seguro y posee un mayor rendimiento.

TECNOLOGÍAS

TECNOLOGÍAS DE VIRTUALIZACIÓN

De igual manera que hemos visto las herramientas de automatización, para poder trabajar con dichas herramientas necesitamos virtualizar el entorno donde se van a lanzar las automatizaciones.

Existen distintas tecnologías que permiten la creación de máquinas virtuales, algunas de las más conocidas son *Vagrant*¹¹, *VmWare*¹², *Microsoft Azure*¹³ o *VirtualBox*.

VIRTUALBOX

Es un software de virtualización de sistemas operativos. Es posible instalar sistemas operativos adicionales, también conocidos como sistemas invitados, dentro de otro sistema operativo anfitrión.

En este proyecto se usará *VirtualBox* para la configuración de la máquina cliente del servidor por diversas razones:

- Es un software gratuito.
- Es multiplataforma y multiejecución.
- Es compatible con máquinas virtuales de *VMware*.
- Capacidad para personalizar hardware.
- Permite la clonación de máquinas virtuales.



Ilustración 4. Logo VirtualBox

En conclusión, son suficientes las razones por las que utilizar esta aplicación de virtualización en nuestro equipo ya que nos brinda una diversidad de opciones con las que podemos realizar todo tipo de pruebas en nuestros equipos virtuales de forma gratuita.

¹¹ Herramienta de código abierto para la creación de ambientes virtuales de desarrollo.

¹² Es un programa que permite ejecutar una computadora virtual dentro de una computadora física.

¹³ Plataforma de computación en la nube, uno de sus principales usos es la implementación de máquinas virtuales

HERRAMIENTAS PARA EL DESARROLLO DEL SOFTWARE:

GITHUB

Se trata de un sistema de gestión de proyectos y control de versiones de código en la nube, lo que quiere decir que permite a desarrolladores almacenar y administrar código, además de llevar un registro y control de cualquier cambio que se ejecute sobre el código.



Ilustración 5. Logo GitHub

También se le conoce como uno de los repositorios online más grades de trabajo colaborativo en todo el mundo.

GIT

Es un sistema de control de versiones, lo que significa que ayuda a registrar los cambios realizados al código y poder restaurar el código borrado o modificado.

REPOSITORY

O también llamado repositorio, es un directorio donde se almacenan los archivos del proyecto. Puede estar ubicado en el almacenamiento de *GitHub* o en un repositorio local en nuestro propio ordenador.

PUSH

Lo que realiza un *push* es publicar lo que se encuentra en nuestro servidor local y llevarlo al servidor remoto de GitHub.

PULL REQUEST

Un *pull request* significa que se informa a los demás usuarios si se ha enviado al repositorio principal un cambio del código o de la estructura.

PASOS PARA INSTALACIÓN DE GIT Y PUBLICACIÓN DEL PROYECTO DESDE CARPETA LOCAL

Nos situamos en el terminal y lanzamos el siguiente comando:

```
root@server:/home/server/# apt-get install git
```

Y una vez instalado realizamos la configuración inicial de *GitHub*:

```
root@server:/home/server/# git config --global user.email "cosmi310599@gmail.com"
```

```
root@server:/home/server/# git config --global user.name "cosmi310599"
```

Después de especificar los datos de la cuenta de *GitHub* creamos un repositorio local.

```
root@server:/home/server/# git init tfg-lamp-ansible
```

Lo que conseguimos con este comando es, además de crear la carpeta TFG-ANSIBLE-LAMP, crear la subcarpeta .INIT que hace que TFG-ANSIBLE-LAMP sea reconocido como un repositorio local en Git.

Nos ubicamos en la carpeta creada recientemente y creamos el archivo README para escribir en el repositorio.

Este archivo se utiliza para describir lo que contiene el repositorio o de lo que va a tratar el proyecto.

```
root@server:/home/server/tfg-lamp-ansible# gedit README
```

Y guardamos los cambios realizando un commit:


```
root@server:/home/server/tfg-lamp-ansible# git commit -m "Creación repositorio"
```

A continuación, desde nuestro servidor, copiamos la llave pública¹⁴ (id_rsa.pub) y la introducimos en el siguiente apartado en GitHub en SETTINGS → SSH AND GPG KEYS →

SSH keys

[New SSH key](#)

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

**root@server**
b0:5c:76:97:a1:aa:26:d1:fa:46:0b:cb:91:1f:63:d9
Added on 10 May 2020
Last used within the last week — Read/write

Delete

Ilustración 6. Imagen llaves SSH GitHub

A continuación, nos vamos a nuestro repositorio y en el siguiente apartado observamos que *GitHub* nos proporciona dos opciones a la hora de clonar o descargar el repositorio, mediante HTTPS o SSH, escogemos la última opción y copiamos el siguiente link:

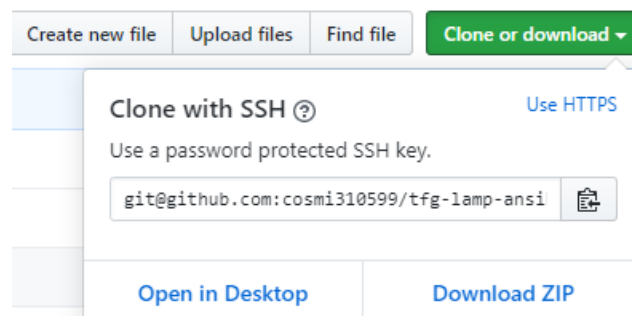


Ilustración 7. Link SSH repositorio GitHub

```
root@server:/home/server/tfg-lamp-ansible# git@github.com:cosmi310599/tfg-lamp-ansible.git
```

¹⁴ Las dos claves pertenecen a la misma persona que recibirá el mensaje. Una clave es pública y se puede entregar a cualquier persona, la otra clave es privada y el propietario debe guardarla de modo que nadie tenga acceso a ella.

VISUAL STUDIO CODE

Es un editor de programación multiplataforma desarrollado por Microsoft. Está diseñado para que funcione tanto en Windows como en Linux e iOS ¹⁵.



Ilustración 8. Logo Visual Studio Code

Este editor permite la edición de código en multitud de lenguajes de programación como Java ¹⁶, C++ ¹⁷, Python ¹⁸, Javascript ¹⁹, HTML²⁰ y CSS ²¹ entre otros.

Además de que se actualiza constantemente, esta herramienta cuenta con multitud de *plugins*²² para trabajar en la nube y descargar proyectos directamente.

Cuenta con *Intellisense*, este término hace referencia a la capacidad que tiene un editor de texto para predecir la instrucción que vamos a escribir. Gracias a esto el editor permite autocompletar el código que se va escribiendo proporcionando un mayor nivel de productividad y acortando la posibilidad de producir errores de sintaxis.

Esta herramienta permite trabajar con código, pero está separado del compilador: permite editar o crear nuevo código.

He escogido *Visual Studio Code* para el desarrollo del código del proyecto porque es un editor multiplataforma ligero y rápido que agiliza el tiempo de trabajo.

¹⁵ Es el sistema operativo de Apple.

¹⁶ Lenguaje de programación de propósito general con sintaxis derivada de C y C++.

¹⁷ Lenguaje de programación diseñado para extender el lenguaje de programación C.

¹⁸ Lenguaje de scripting independiente de plataforma y orientado a objetos.

¹⁹ Lenguaje de programación, que se utiliza principalmente para la creación de páginas web dinámicas.

²⁰ Lenguaje de marcado que se utiliza para el desarrollo de páginas en internet.

²¹ Lenguaje utilizado para presentación de documentos HTML.

²² Es un fragmento o componente de código hecho para ampliar las funciones de un programa o de una herramienta.

MARCO PRÁCTICO

ESTRUCTURA

El entorno con el que vamos a trabajar se encuentra descrito en la siguiente ilustración.

Tendremos un servidor *Ubuntu* donde se encontrará alojado el software de *Ansible* con el código necesario para realizar las actividades propuestas. Este servidor se conectará mediante *SSH* a un nodo remoto con S.O *Debian* implantado en una máquina virtual en *VirtualBox* donde una vez se ejecute correctamente el *playbook* de *Ansible*, se convertirá en un servidor *LAMP*.

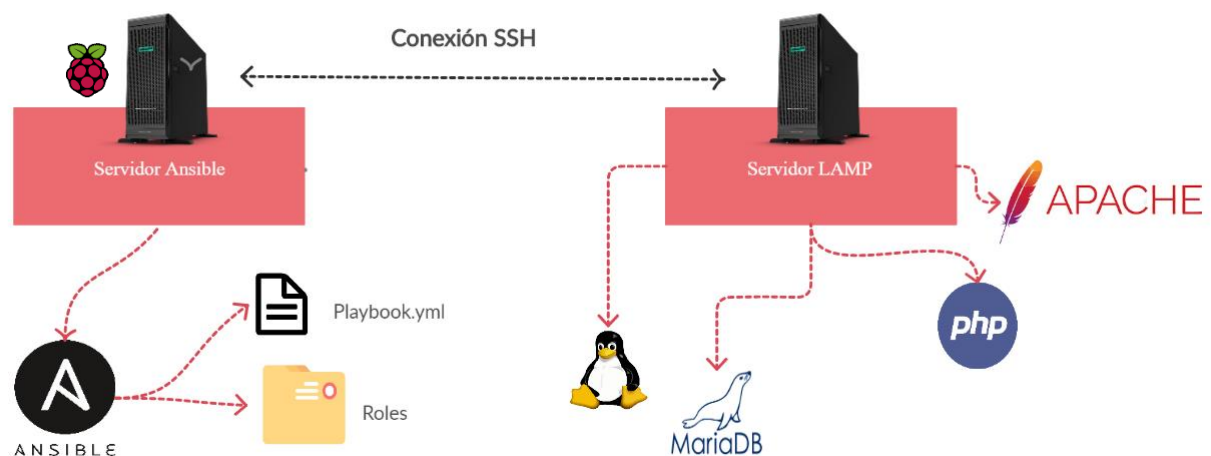


Ilustración 9. Estructura TFG

LAMP

Como podemos observar en el esquema provisto anteriormente, mediante *Ansible* se procederá a la implantación de un servidor **LAMP** en el *host* remoto.

Las iniciales **LAMP** hacen referencia a los siguientes cuatro componentes: *Linux*, *Apache*, *MySQL* y *PHP*. Todos ellos forman la infraestructura en el servidor, haciendo posible la creación y el alojamiento de páginas web dinámicas.

- **LINUX**: como sistema operativo.
- **APACHE**: como servidor web.
- **MYSQL (O MARIADB)**: como gestor de bases de datos relacional.
- **PHP**: como lenguaje de scripts orientado a objetos.

FUNCIONAMIENTO ANSIBLE

Como hemos visto anteriormente, al inicio de este proyecto, se hace un pequeño inciso de lo que es *Ansible*. A continuación, se procederá a explicar los conceptos básicos, su estructura y como funciona de una manera más detallada.

Ansible es una plataforma de automatización de código abierto que se utiliza para la gestión de la configuración, la implementación de aplicaciones y la automatización de tareas. Además, puede organizar la T.I para ejecutar tareas en secuencia y crear una cadena de eventos que deben de ocurrir en varios servidores o dispositivos diferentes.

Ansible permite definir su infraestructura como código de una manera simple. La infraestructura como código se basa en el proceso de gestión y aprovisionamiento de infraestructura informática y de redes y su configuración a través de archivos de definición procesables por la máquina (código), en lugar de la configuración física del hardware o el uso de herramientas de configuración interactivas.

Hace uso del módulo *ad-hoc* ²³ para ejecutar comandos de Shell en múltiples máquinas. Esto llega a ser útil si se está aprovisionando muchos servidores ya que se reduce el tiempo de dicho aprovisionamiento para que sea más fácil y rápido replicar su infraestructura.

CONCEPTOS BÁSICOS

Es importante comprender estos conceptos para poder entender el funcionamiento del código que se detallará posteriormente.

INVENTARIO

Ansible funciona contra múltiples sistemas en su infraestructura al mismo tiempo. Lo hace seleccionando partes de los sistemas enumerados en el archivo de inventario de Ansible que por defecto lo encontramos en la siguiente ruta: */etc/Ansible/hosts*.

A través de los archivos de inventario, se pueden especificar grupos significativos de *hosts*. También se pueden especificar variables de grupo o variables de *host* que ayudarán a controlar como interactuará *Ansible* con los *hosts* remotos.

Ejemplo:

```
[servidores-debian]
máquina1.debian ansible_host=192.168.0.5
máquina2.debian ansible_host=192.168.0.6

[servidores-ubuntu]
máquina1.ubuntu ansible_host=192.168.0.7

[servidores-centos]
máquina1.ubuntu ansible_host=192.168.0.8
```

** Más adelante se profundizará más sobre los parámetros descritos en el ejemplo.

²³ Este módulo permite realizar acciones de forma simple y comprobar la conexión de los elementos del inventario.

Los encabezados entre corchetes son nombres de grupo que se utilizan para clasificar sistemas, es decir, que sistemas se están controlando y con qué propósito.

MÓDULOS

Ansible funciona con varios módulos que se pueden ejecutar directamente en *hosts* remotos o mediante *playbooks*. Estos módulos son capaces de controlar los recursos del sistema como servicios, paquetes o archivos.

Los módulos usan “*facts*” (hechos) para determinar qué acciones ejecutar en función del estado de la máquina *host*. *Ansible* es **idempotente**, es decir, los módulos de *Ansible* pueden determinar si una tarea se ejecutará o no, y con esto garantizar que, no importa cuántas veces se ejecuten los scripts, que siempre permanecerán en el mismo estado.

Dicho de otro modo, gracias a la **idempotencia** conseguimos prevenir efectos secundarios en la ejecución repetitiva de scripts y aseguramos que ninguna operación se realizará una vez que el sistema ha alcanzado el estado deseado.

TAREAS

Son acciones o pasos que definen el estado esperado de la máquina cliente en un momento determinado. *Ansible* utiliza tareas como una forma de ordenar acciones para que se ejecuten cuando se ejecute un *playbook*.

Se ejecutan de forma secuencial, lo que implica que una tarea no se ejecutará si la anterior no se ha completado correctamente.

Las tareas se utilizan junto con los módulos para lograr un resultado particular. Es importante tener en cuenta que una tarea solo puede realizar una operación a la vez, es decir, una tarea para una operación. A veces hacemos uso de bucles para que una tarea pueda realizar más de una operación.

PLAYBOOK

Los *playbooks* son el lenguaje de configuración, implementación y orquestación de *Ansible*. Describen un conjunto de pasos en un proceso de TI general.

Básicamente son archivos *YAML* que describen el estado deseado del *host* o un grupo de *hosts* declarados en el archivo de inventario.

De forma básica, los *playbook* se pueden usar para administrar configuraciones e implementaciones en máquinas remotas, sin embargo, pueden secuenciar implementaciones de varios niveles que impliquen actualizaciones continuas y delegar acciones a otros *hosts*.

ROLES

Con los roles podemos crear una estructura de ficheros y directorios para separar los elementos y así poder reutilizarlos fácilmente.

Los roles se basan en la idea de incluir archivos y combinarlos. Cada *playbook* se asocia a un conjunto de roles y cada rol está representado por llamadas a lo que *Ansible* define como tareas.

A la hora de definir los roles debemos tener un directorio con la siguiente estructura de subdirectorios:

```
└─ roles
  └─ Nombre-rol
    ├── defaults
    ├── files
    │   └─ ficheros_configuración.conf.j2
    ├── handlers
    │   └─ main.yml
    ├── tasks
    │   └─ main.yml
    ├── templates
    └─ vars
```

- **DEFAULTS:** en este directorio están definidos los valores predeterminados para las variables de prioridad más baja, dicho de otra manera, si una variable no se define en ningún otro lugar, se utilizará la definición dada en este directorio en el archivo *main.yml*.
- **FILES:** en este directorio se almacenarán los ficheros que se deben copiar en los *hosts* que pertenecen a ese rol.
- **TEMPLATES:** incluye las plantillas de archivos que se desplegarán en las máquinas remotas.

Los directorios de *files* y *templates* tienen un propósito similar, ambos contienen archivos y plantillas que se utilizan dentro del rol. Lo bueno de la existencia de estos directorios es que *Ansible* no necesita una ruta para los recursos almacenados en ellos cuando trabaja en el rol.

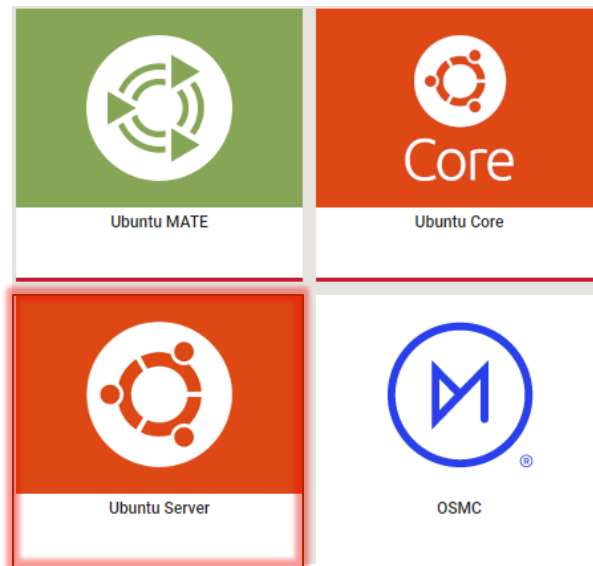
- **TASKS:** incluye el archivo *main.yml* con la lista de tareas a ejecutar en los *hosts* que pertenecen a este rol. La ejecución de una tarea puede desencadenar la ejecución de acciones.
- **HANDLERS:** este directorio se utiliza para almacenar los “handlers”, que son igual que las tareas, pero solo se ejecutan cuando otra tarea lo llame. Los *handlers* se ejecutan siempre al final del playbook.
- **VARS:** indicamos las variables a utilizar para este rol.

IMPLEMENTACIÓN

CONFIGURACIÓN RASPBERRY PI

Vamos a proceder a detallar los pasos necesarios para la configuración de la *Raspberry Pi* como servidor con sistema operativo *Ubuntu*.

Accedemos a la página oficial de [Raspberry](#) y seleccionamos [Downloads](#):



Como podemos observar, tenemos a nuestra disposición multitud de imágenes de sistemas operativos para instalar en la *Raspberry pi*.

En este caso se implementará un S.O *Ubuntu Server* porque dispone de una instalación flexible, potente y sin configuración.

Descargamos la imagen en función de la arquitectura que tenga la *Raspberry pi* y una vez tengamos la imagen necesitamos un *software* específico que nos permita grabar la imagen en la tarjeta SD que en este caso será la memoria principal de nuestra *Raspberry pi*.

Tenemos a nuestra disposición multitud de programas que cumplen con la función requerida, en este caso vamos a hacer uso del programa *Win32 Disk Imager*, se trata de una aplicación que graba imágenes CD o DVD en memorias USB o tarjetas SD, creando un lector de discos virtual.

Antes de realizar estos pasos hay que dejar preparada la tarjeta *microSD* donde se va a realizar la grabación de la imagen.

Hay que asegurarse que la imagen cabe en nuestra tarjeta *microSD*. Para dejarla bien configurada seguimos los siguientes pasos:

1. Abrimos el CMD del sistema con permisos de administrador.
2. Lanzamos el siguiente comando para entrar en DISKPART ²⁴.

```
C:\Windows\system32>diskpart

Microsoft DiskPart versión 10.0.18362.1

Copyright (C) Microsoft Corporation.
En el equipo: DESKTOP-U0DFPMK

DISKPART>
```

²⁴ Diskpart es una utilidad de línea de comandos que permite administrar discos.

3. Listamos los discos que tenemos disponibles y seleccionamos el deseado.

```
DISKPART> list disk
```

Núm Disco	Estado	Tamaño	Disp	Din	Gpt
Disco 0	En línea	447 GB		0 B	
Disco 1	En línea	14 GB		0 B	

```
DISKPART> select disk 1
```

El disco 1 es ahora el disco seleccionado.

```
DISKPART>
```

4. Ejecutamos el siguiente comando para limpiar el disco.

```
DISKPART> clean
```

5. Creamos una partición primaria con sistema de archivos NTFS ²⁵ y le asignamos letra.

```
DISKPART> create partition primary
```

El disco 1 es ahora el disco seleccionado.

Y ya tenemos nuestro volumen listo para poder grabar en la imagen que hemos descargado previamente.

CONFIGURACIÓN SERVIDOR UBUNTU

Una vez se haya instalado correctamente la imagen en el servidor y tengamos el S.O funcionando, comenzamos cambiando el *hostname* ²⁶ del servidor configurando el siguiente archivo y cambiando el valor que viene por defecto por el valor deseado:

```
root@ubuntu:~# nano /etc/hostname
```

```
root@server:~# reboot
```

A continuación, vamos a configurar la IP del servidor para que pase de ser una IP dinámica (asignada por DHCP ²⁷) a una IP estática.

²⁵ Es un sistema de archivos desarrollado por Microsoft

²⁶ Hace referencia a un nombre de equipo (único) y relativamente informal que se le da a un dispositivo conectado a una red informática.

²⁷ Protocolo DHCP sirve principalmente para distribuir direcciones IP en una red.

Ejecutamos el siguiente comando para ver las interfaces de red disponibles en el servidor:

```
root@server:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether b8:27:eb:05:fd:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.21/24 brd 192.168.0.255 scope global dynamic eth0
        valid_lft 85299sec preferred_lft 85299sec
    inet6 fe80::ba27:ebff:fe05:fdc3/64 scope link
        valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether b8:27:eb:50:a8:96 brd ff:ff:ff:ff:ff:ff
```

Mediante la información proporcionada por el comando anterior, sabemos que el nombre de nuestra interfaz de red es *eth0*.

Editamos el archivo de configuración de las interfaces de red que se encuentra ubicado en la siguiente ruta y añadimos los valores necesarios:

```
root@server:~# nano /etc/network

auto eth0
iface eth0 inet static
address 192.168.0.27
netmask 255.255.255.0
network 192.168.0.0
gateway 192.168.168.0.1
```

- ADDRESS es la dirección IP que queremos que tenga la máquina.
- NETMASK es la máscara de subred de esa dirección IP.
- NETWORK es la red a la que pertenece esa dirección IP.
- GATEWAY es la dirección IP de la puerta de enlace predeterminada.

Para guardar cambios reiniciamos las interfaces de red de la siguiente manera:

```
root@server:~# ifconfig eth0 down
root@server:~# ifconfig eth0 up
```

Y ya tenemos el servidor preparado para comenzar con la instalación de *Ansible*.

INSTALACIÓN ANSIBLE

Para comenzar a utilizar *Ansible* es necesario instalar el *software* en la máquina que actuará como nodo de control.

El primer paso a seguir lanzar el siguiente comando:

```
root@server:~# apt-get install software-properties-common
```

Este *software* proporciona una abstracción de los repositorios apt utilizados. Permite administrar fácilmente la distribución y las fuentes de *software* de proveedores de software independientes.

Ejecutamos el siguiente comando para incluir el *PPA*²⁸ en la lista de fuentes del sistema:

```
root@server:~# apt-add-repository ppa:ansible/ansible
```

Y realizamos un *update* para que actualice el índice de paquetes del sistema para que tenga conocimiento de los paquetes disponibles que acabamos de incluir.

```
root@server:~# apt update
```

Y finalmente podemos instalar el *software* de *Ansible*:

```
root@server:~# apt-get install Ansible
```

²⁸ Archivo de paquetes personal.

CONFIGURACIÓN HOST REMOTO


Para realizar la configuración del cliente se hace uso de un sistema operativo *Debian 9* que podemos descargar la imagen desde el siguiente *link*.

Importamos la imagen desde *VirtualBox* siguiendo los siguientes pasos en orden:

1. Nueva → especificamos nombre, la ubicación de la carpeta donde se van a almacenar los datos de la máquina, el tipo de S.O que vamos a implantar y su versión.

Nombre:

Carpeta de máquina:

Tipo: 

Versión:

2. Especificamos el tamaño de RAM ²⁹, creamos un disco duro virtual con reservado dinámico y asignamos cuanto memoria necesitamos que tenga y ya tenemos el equipo que actuará de servidor LAMP configurado.

Tamaño de memoria

Seleccione la cantidad de memoria (RAM) en megabytes a ser reservada para la máquina virtual.

El tamaño de memoria recomendado es **1024 MB**.

MB

4 MB

Escriba el nombre del archivo de unidad de disco duro virtual en el campo debajo o haga clic en el icono de carpeta para seleccionar una carpeta diferente donde crear el archivo.

Seleccione el tamaño de disco duro virtual en megabytes. Este tamaño es el límite para el archivo de datos que una máquina virtual podrá almacenar en el disco duro.

4,00 MB 2,00 TB

²⁹ La memoria RAM es la memoria principal de un dispositivo donde se almacena programas y datos informativos.

CONEXIÓN CLIENTE-SERVIDOR

Vamos a usar el método de autenticación de clave privada y clave pública para evitar tener que introducir la contraseña cada vez que queramos conectarnos y así hacer las tareas lo más automatizadas posible.

Desde el servidor exportamos la clave pública al equipo cliente con el comando *ssh-copy-id*:

```
root@server:~# ssh-copy-id root@192.168.0.26
/usr/bin/ssh-copy-
id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'root@192.168.0.26'"
and check to make sure that only the key(s) you wanted were added.
```

Comprobamos si podemos acceder correctamente al *host* remoto lanzando el siguiente comando:

```
root@server:~# ssh root@192.168.0.26
Linux cliente 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat May 16 19:24:02 2020 from 192.168.0.27
root@cliente:~#
```

Y como podemos observar al final, hemos iniciado sesión con el usuario de la máquina cliente.

DISEÑO E IMPLEMENTACIÓN DEL CÓDIGO

Árbol con la estructura de directorios y archivos necesario para la implementación y configuración del servidor LAMP.

```
.
├── inventories
│   └── virtualbox
│       └── hosts
├── playbook.yml
├── README.md
├── roles
│   ├── apache2
│   │   ├── defaults
│   │   │   └── main.yml
│   │   ├── files
│   │   │   ├── CSS
│   │   │   │   ├── estilo_formulario.css
│   │   │   │   ├── estilo_index.css
│   │   │   │   ├── estilo_inicio.css
│   │   │   │   └── estilo_registro.css
│   │   │   ├── formulario.php
│   │   │   ├── index.html
│   │   │   ├── inicio.php
│   │   │   ├── registro.php
│   │   │   └── ssl-params.conf.j2
│   │   ├── handlers
│   │   │   └── main.yml
│   │   ├── tasks
│   │   │   └── main.yml
│   │   ├── templates
│   │   │   └── virtualhost.conf.j2
│   │   └── vars
│   │       └── main.yml
│   ├── common
│   │   └── tasks
│   │       └── main.yml
│   └── mariadb
│       ├── defaults
│       │   └── main.yml
│       ├── files
│       │   ├── backup.sh
│       │   └── bdd.sql
│       ├── handlers
│       │   └── main.yml
│       └── tasks
│           └── main.yml
18 directories, 23 file
```

INVENTORIES

En esta ubicación encontramos otro directorio con el fichero que contiene la información acerca de los *hosts* y los grupos a los que pertenecen.

El valor proporcionado entre corchetes es el nombre con el que nos vamos a referir al *host* remoto.

```
[lamp]
192.168.0.26 ansible_user=root
```

Especificamos la IP y con el parámetro `ANSIBLE_USER` le indicamos el nombre del usuario para realizar las tareas previstas dentro del nodo a controlar.

Tenemos a nuestra disposición multitud de parámetros más para especificar en este fichero, pero en este caso solo son necesarios hacer uso de estos dos.

Otra manera de especificar la información relativa a los *hosts* es editar el fichero que viene por defecto en la ruta de `/ETC/ANSIBLE/HOSTS`.

PLAYBOOK

Como hemos definido anteriormente en el apartado de conceptos básicos, sabemos que un *playbook* es un libro de jugadas cuya función es realizar un conjunto de tareas en un nodo específico a controlar.

```
- hosts: lamp
  roles:
    - common
    - mariadb
    - apache2
```

HOSTS: En este apartado hacemos referencia al *host* o al grupo de *hosts* que queremos administrar.

BECOME: el valor proporcionado a este parámetro indica que *Ansible* realizará todas las tareas previstas en el *host* como usuario *root*.

ROLES: en este apartado incluimos la lista de roles a ejecutar.

COMMON

En este directorio se encuentran especificadas las tareas que son comunes a ambos roles.

TASKS

Dentro del directorio nos encontramos con la subcarpeta *tasks* donde se ubica el fichero *main.yml* con las tareas a ejecutar.

MAIN.YML

```
- name: Update repositories
  apt:
    update_cache: yes
    force_apt_get: yes
    cache_valid_time: 3600
```

En este fichero ya empezamos a hacer uso de los módulos que provee *Ansible*.

Los módulos nos ofrecen funciones que podemos ejecutar con tareas.

MÓDULO: APT

Con este módulo se hace uso del administrador de paquetes *apt*, que es el predeterminado en las distribuciones *Linux* basadas en *Debian/Ubuntu*.

UPDATE_CACHE: Mediante este parámetro, al instalar o actualizar paquetes, lo que hace es obligar a *apt* a verificar si el caché del paquete está desactualizado y lo actualiza si es necesario.

FORCE_APT_GET: Este parámetro fuerza a usar *apt-get install* en lugar de *aptitude*.

CACHE_VALID_TIME: Especificamos el tiempo que dura la caché.

```
- name: Upgrading all the apt packages
  apt:
    upgrade: dist
    force_apt_get: yes
```

UPGRADE: El valor que le estamos proporcionando a este parámetro (*dist*) es el equivalente a realizar un *apt dist-upgrade*.

La principal diferencia entre un *apt upgrade* y el que hemos especificado, es que cuando únicamente ejecutamos un *apt upgrade*, solo se actualiza cuando hay una nueva versión disponible tal y como se define en */etc/apt/sources.list*.

Sin embargo, cuando ejecutamos un *apt dist-upgrade*, se realiza la instalación o eliminación de los paquetes de forma inteligente, según sea necesario para completar la actualización. Cuenta con un sistema inteligente de resolución de conflictos, por lo que intentará actualizar los paquetes más importantes.

```
- name: Install common packages
  apt:
    name: ufw
    state: present
```

En esta tarea instalamos el paquete *ufw*.

Hace referencia al cortafuegos diseñado para ser manejado a través de línea de comando en distribuciones *Linux*. Es necesario para tareas futuras donde haremos uso de este programa.

MÓDULO: UFW

Módulo que se encarga de administrar el firewall con *UFW*.

```
- name: Enable firewall
  ufw:
    state: enabled
```

STATE: Mediante este parámetro recargamos el firewall y lo habilitamos para que inicie en el arranque del sistema.

MARIADB

En el siguiente apartado vamos a comenzar con los pasos necesarios para la instalación y configuración de un sistema gestor de bases de datos que, posteriormente, se conectará con apache para poder organizar y recolectar la información procedente de la aplicación web implantada.

MariaDB se considera básicamente un reemplazo de *MySQL*, que además de aportar un mayor rendimiento también proporciona nuevas funcionalidades.

De hecho, es un [fork](#)³⁰ de *MySQL* ya que ha sido creado a través de él. Se ha escogido la implementación de este software porque permite una mayor organización de las bases de datos y provee de un mayor crecimiento y progresión que el propio *MySQL*.

DEFAULTS

Dentro de este directorio vamos a ubicar las variables predeterminadas para el rol de *MariaDB*.

Estas variables tendrán la prioridad más baja de cualquier variable disponible y pueden ser anuladas fácilmente por cualquier otra variable.

MAIN.YML

```
---
mysql_bdd: incidencias
mysql_user: admin_1
mysql_password: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    64353238343131313963643237303966666533343433323833373633306539383333393939653434
    3737363935333336303263383538313562383933656364610a626439396435313161356336623938
    61646434306330363934376164636235323031663836626634333163353039303964386666373362
```

MYSQL_BDD: Esta variable hace referencia al nombre de la base de datos que vamos a insertar en *MariaDB*.

MYSQL_USER: Nombre que vamos a proporcionar al usuario administrador de la base de datos.

MYSQL_PASSWORD: Contraseña del usuario administrador.

Se ha encriptado mediante una característica de Ansible denominada **VAULT** que permite mantener datos confidenciales, en este caso contraseñas, de forma cifrada en lugar de mostrar el texto plano.

Para encriptar un valor específico del archivo lanzamos el siguiente comando:

```
ansible-vault encrypt_string --name 'mysql_password'
```

³⁰ Bifurcación.

FILES

En este directorio se ubican los archivos que necesitamos copiar desde el nodo de control al *host* remoto.

BACKUP.SH

Este fichero es un script realizado en `bash` ³¹ con el fin de la realización de copias de seguridad de las bases de datos existentes en nuestro *host*.

En el primer apartado hacemos uso de variables para almacenar datos necesarios para la ejecución de las copias de seguridad.

Después creamos la siguiente ruta `/etc/mysql/backup/test-backup` para almacenar las copias de seguridad, dentro de ese directorio se crearán subdirectorios con la fecha específica del día en el que se ejecute el *script*.

A continuación, en la variable `databases` guardamos el nombre de todas las bases de datos que tenemos en el *host* realizando la conexión a la base de datos mediante el usuario y la contraseña y haciendo uso del comando `grep` para eliminar las bases de datos que no nos interese incluir en el *backup*.

Mediante una estructura repetitiva `for` ³², recorreremos la lista generada anteriormente y generamos la copia de seguridad en la ruta que se ha detallado en el apartado anterior.

Finalmente, la última instrucción se encarga de encontrar y eliminar copias de seguridad que lleven más de 15 días en el sistema.

Este script se ejecutará periódicamente con una tarea programada en `crontab` ³³ para que se realice cada 15 días. Esta tarea estará definida en el `main.yml` principal ubicado en la carpeta de *tasks*.

```
#!/bin/bash
## Variables
DATE=$(date +%d-%m-%Y)
BACKUP_DIR="/backup/test-backup"
MYSQL_USER="admin_1"
MYSQL_PASSWORD="Admin12345"

mkdir -p /etc/mysql/$BACKUP_DIR/$DATE

## Generate a list with the name of the databases and cut the ones that you do not want to back up
databases=`mysql -u$MYSQL_USER -p$MYSQL_PASSWORD -e "SHOW DATABASES;" | grep -Ev "(information_schema|performance_schema)"`

## Backup for each database
for db in $databases; do
```

³¹ Lenguaje de comandos y shell de Unix

³² Es una sentencia que implementa un bucle, es decir, que es capaz de repetir un grupo de sentencias un número determinado de veces.

³³ Es un administrador regular de procesos en segundo plano que ejecuta procesos o guiones a intervalos regulares.

```
mysqldump --force --opt --skip-lock-tables -u $MYSQL_USER -p$MYSQL_PASSWORD -
-databases $db > "/etc/mysql/$BACKUP_DIR/$DATE/$db.sql"
done

# Delete files older than 15 days
find /etc/mysql/$BACKUP_DIR/* -mtime +15 -exec rm {} \;
```

BDD.SQL

En el siguiente fichero nos encontramos con las sentencias precisas para la implementación de la base de datos incidencias. Esta base de datos está creada para recoger información de un formulario que posteriormente se desplegará mediante *Apache*. Se ha establecido como clave primaria el correo electrónico del usuario que rellena el informe. Todas las incidencias que vaya realizando el usuario se almacenarán en la tabla datos, que tiene como clave primaria el código de incidencia.

```
DROP DATABASE IF EXISTS incidencias;

CREATE DATABASE incidencias;

USE incidencias;

CREATE TABLE info(
  Correo          VARCHAR(30)      NOT NULL,
  Num_equipo      INT UNSIGNED    NOT NULL,

  CONSTRAINT pk_Correo PRIMARY KEY (Correo)
);

CREATE TABLE datos(
  CodIncidencia   SMALLINT(6)     AUTO_INCREMENT,
  Correo          VARCHAR(30)      NOT NULL,
  Departamento    VARCHAR(10)     NOT NULL,
  Asunto          VARCHAR(15)     NOT NULL,
  Descripcion     VARCHAR(150)    NOT NULL,

  CONSTRAINT pk_CodIncidencia PRIMARY KEY (CodIncidencia),
  CONSTRAINT fk_Correo FOREIGN KEY (Correo) REFERENCES info (Correo)
);
```

HANDLERS

Como ya se detalló en el apartado de conceptos básicos de *Ansible*, los *handlers* son tareas referenciadas por un nombre global único que son alertadas por otras tareas. Si no se notifica al handler, no se ejecuta.

Independientemente de cuantas veces sea notificado, solo se ejecutará al concluir la ejecución de todas las tareas.

MAIN.YML

En este caso creamos el handler con el propósito de reiniciar *mariadb*.

```
---
- name: Restart mariadb
  service:
    name: mysql
    state: restarted
```

MODULO: SERVICE

Módulo encargado de controlar los servicios en *hosts* remotos.

TASKS

MAIN.YML

```
---
- name: Install package MariaDB with APT
  apt:
    name: mariadb-server
    state: present
```

En esta tarea se procede a la instalación del paquete principal de *MariaDB*.

Se separa de la siguiente tarea y no se introduce en el *loop* ³⁴ con el resto de paquetes porque al realizarse la instalación con el módulo *package* y al ser tan común a todas las distribuciones Linux, no instala el fichero de configuración principal de *MariaDB* en la ruta adecuada en *Debian*. Por eso se hace uso del módulo *apt*, específico para la distribución que estamos utilizando.

```
- name: Install required packages for MariaDB
  package:
    name: "{{ item }}"
    state: present
  with_items:
    - python-mysqldb
    - default-libmysqlclient-dev
    - php-mysql
```

En la siguiente tarea se procede a la instalación del resto de paquetes necesarios para el funcionamiento correcto de *MariaDB*.

- PYTHON-MYSQLDB: es una interfaz para conectarse a un servidor de base de datos *MySQL* desde Python. Este paquete es requerido por el propio Ansible si no se especifica.

³⁴ Es una secuencia que ejecuta repetidas veces un trozo de código, hasta que la condición asignada a dicho bucle deja de cumplirse.

- `DEFAULT-LIBMYSQCLIENT-DEV`: el paquete de instalación principal depende de la implantación predeterminada de bibliotecas de desarrollo del cliente.
- `PHP-MYSQL`: este paquete proporciona un módulo *MySQL* para *PHP*.

MODULO: PACKAGE

Instala, actualiza y elimina paquetes usando el administrador de paquetes subyacente.

```
- name: Enable firewall rules for MySQL
  ufw:
    rule: allow
    port: '3306'
```

Hacemos uso del módulo visto anteriormente para configurar el *firewall* del host y permitir una regla por el puerto 3306, que es el puerto de *MySQL*.

```
- name: Start MySQL service and enable it
  service:
    name: mysql
    state: started
    enabled: yes
```

Después iniciamos el servicio de *MySQL* y lo habilitamos para que inicie en el arranque del sistema.

```
- name: Create user with privileges
  mysql_user:
    name: "{{ mysql_user }}"
    password: "{{ mysql_password }}"
    priv: ' *.*:ALL '
    host: localhost
    state: present
    no_log: True
```

Creamos un Usuario con privilegios y hacemos uso de las variables especificadas en el directorio *defaults* para hacer referencia al nombre del usuario que vamos a crear y la contraseña que va a tener asignada.

Para evitar exponer datos secretos (contraseñas) hacemos uso del atributo `NO_LOG`, para mantener los valores confidenciales fuera de los registros.

MODULO: MYSQL_USER

Es un módulo de *Ansible* cuya funcionalidad consiste en agregar o eliminar un usuario de una base de datos *MySQL*.

NAME: parámetro para proporcionar el nombre que se le asignará al usuario en cuestión.

PASSWORD: parámetro para proporcionar la contraseña para dicho usuario.

PRIV: hace referencia a los privilegios que se le puede asignar a un usuario. Con el valor que se le ha especificado al parámetro se le asignan todos los permisos al usuario. En este caso será un usuario administrador.

HOST: con este parámetro creamos el usuario a nivel local.

STATE: por defecto el valor predeterminado es *present*, para crear el usuario. En caso de querer eliminarlo se especificaría como valor *absent*.

```
- name: Create database
  mysql_db:
    name: "{{ mysql_bdd }}"
    state: present
    register: db_created
```

MODULO: MYSQL_DB

Módulo encargado de agregar o eliminar bases de datos en *MySQL* de un *host* remoto.

REGISTER: cuando se ejecute la tarea en cuestión, se guarda el valor de retorno en una variable para ser usada en tareas posteriores.

```
- name: Copy database
  copy:
    src: bdd.sql
    dest: /tmp/bdd.sql
```

Esta tarea se encarga de copiar el fichero que contiene las instrucciones para la creación de la base de datos en el directorio */tmp*

MODULO: COPY

Módulo cuya finalidad consiste en copiar un archivo desde la máquina local o remota a una ubicación en la máquina remota.

SRC: ruta local a un archivo para copiar en el servidor remoto.

DEST: ruta absoluta remota donde se debe copiar el archivo.

```
- name: Insert database
  mysql_db:
    name: incidencias
    state: import
    target: /tmp/bdd.sql
    when: db_created.changed
```

Mediante el módulo *mysql_db* insertamos la base de datos que hemos copiado en el *host* remoto con la última tarea que hemos visto.

Hacemos uso del valor **WHEN** para condicionar esta tarea. Únicamente se insertará la base de datos cuando en la tarea anterior se haya producido algún cambio.

NAME: con este parámetro se asigna el nombre a la base de datos.

STATE: este parámetro tiene 4 posibles valores: *present*, *absent*, *dump* e *import*, en este caso se especifica que queremos insertar la base de datos.

TARGET: ubicación, en el *host* remoto del archivo de volcado para leer o escribir.

```
- name: Copy shell script for backup
```

```
copy:
  src: backup.sh
  dest: /etc/backup/
```

Haciendo uso del módulo *copy*, ya explicado anteriormente, copiamos el script que realiza la copia de seguridad a la base de datos en la ruta especificada en el parámetro *dest*.

```
- name: Generate a cron task to run the sh
cron:
  name: "Backup"
  minute: "0"
  hour: "12"
  day: "15"
  job: "sh /etc/backup/backup.sh"
```

La última tarea especificada en este rol se encarga de crear una tarea programada en cron.

Cron es un administrador regular de procesos en segundo plano, o también conocido como demonio, que ejecuta procesos o guiones a intervalos regulares. Por ejemplo, cada minuto, día, semana o es.

La copia de seguridad se realizará a las doce de la noche cada 15 días todos los meses.

MODULO: CRON

Módulo encargado de administrar el demonio³⁵ cron.

NAME: nombre que especificamos a la tarea cron que vamos a crear.

MINUTE: minuto en el que debe ejecutarse el trabajo (0-59).

HOURL: hora en la que debe ser ejecutada (0-23).

DAY: día del mes en que debe ejecutarse la tarea (1-31).

JOB: en este parámetro especificamos el comando que queremos que ejecute.

APACHE

En el apartado final realizamos los pasos necesarios para la instalación y configuración de Apache, donde vamos a implementar nuestro servicio web y así dar por finalizado la instalación e implementación completa del *LAMP*.

DEFAULTS

MAIN.YML

```
---
port_http: '80'
port_https: '443'
domain: www.cesa.com
```

³⁵ Un demonio o servicio es un programa que se ejecuta en segundo plano, fuera del control interactivo de los usuarios del sistema

Dentro de este directorio vamos a ubicar las variables predeterminadas para el rol de *Apache*.

PORT_HTTP: mediante este parámetro indicamos el puerto de escucha del servidor *Apache*.

Es el puerto al que por defecto un servidor HTTP ³⁶ “escucha” la petición hecha por un cliente. (Puerto no seguro, no cifra las comunicaciones).

PORT_HTTPS: el puerto 443 o también conocido como protocolo de transmisión (TCP) ³⁷ es el predeterminado que utiliza el HTTPS ³⁸, utiliza la combinación de dos protocolos HTTP + SSL/TLS ³⁹ que hace que cualquier tipo de información que se transmita en la red sea cifrada.

DOMAIN: parámetro que hace referencia al nombre que se le asigna a la página web.

FILES

Directorio donde se ubican los archivos que necesitamos copiar desde el nodo de control al *host* remoto.

INDEX.HTML

Es el archivo que el servidor ejecuta por defecto cuando se carga el dominio en cuestión, dicho de otro modo, el punto de entrada a nuestro sitio web.

Es el archivo que se va a visualizar al entrar a la raíz del sitio web. Si no existiera este archivo, mostraría el listado de todos los archivos que tengamos en la raíz.

Visualizamos dos botones con las siguientes opciones a elegir que se detallan en los siguientes archivos.

Estos archivos se combinan con el *index.html* con la finalidad de establecer una conexión entre el servidor *Apache* y la base de datos.

REGISTRO.PHP

En el caso de elegir el botón de registro nos redirige al siguiente archivo PHP.

Se compone de dos campos los cuales vamos a rellenar en el caso de registrarnos introduciendo el correo electrónico y el ID del equipo.

En el apartado de PHP se compone de varios pasos.

El primer paso que se realiza es establecer la conexión a la base de datos. Si se produce algún error a la hora de establecer la conexión, dicho error se controla y se informa al usuario mediante la función específica de PHP *mysqli_connect_error()*.

A continuación, se recogen los datos del formulario *HTML* mediante el método *POST* y cada dato se asigna a una variable.

Existen dos métodos con los que el navegador puede enviar información al servidor. El método *GET* envía los datos usando la *URL*, el método *POST* los envía de forma que no podemos verlos, es decir, en segundo

³⁶ HTTP es el acrónimo de Hypertext Transfer Protocol.

³⁷ Protocolo de Control de Transmisión

³⁸ Protocolo seguro de transferencia de hipertexto.

³⁹ Capa de Conexiones Seguras. Protocolo que hace uso de certificados digitales para establecer comunicaciones seguras a través de Internet.

plano u ocultos para el usuario. Se ha escogido el método *POST* debido a la seguridad que proporciona en cuanto a la privacidad de datos.

Se realiza una consulta para comprobar si en la base de datos ya existe un registro que coincida con el correo electrónico, en el caso de que sea así, redirige de nuevo al archivo *registro.php* para darse de alta con un correo que no existe y lanza un *alert* mediante *javascript* que informa del error.

Si se rellena el formulario con un correo que no existe en la base de datos, realiza el *insert* y redirige al archivo *inicio.php* para que se inicie sesión mediante un script introducido en el php.

INICIO.PHP

Este archivo sirve para poder iniciar sesión y acceder al formulario de incidencias.

Al inicio se abre una conexión con la base de datos controlando los posibles errores como en el anterior fichero.

Guardamos el valor del campo email en una variable y realizamos una consulta donde busque en la base de datos si existe un correo que coincida con el introducido. Si es así nos redirige al *formulario.php* mediante la función especificada en la etiqueta script.

En el caso de que no exista el correo introducido nos redirige al fichero de registro para darnos de alta.

FORMULARIO.PHP

Este archivo se compone de un formulario para rellenar las incidencias que surjan.

Del mismo modo que en los ficheros anteriores, abrimos la conexión con la base de datos y mediante variables recogemos todos los datos necesarios para insertarlos en la base de datos.

Una vez se hayan insertado los datos correctamente, mediante un *alert* se informa de que todo ha ido bien.

CSS

Directorio donde se ubica el fichero con los estilos que se aplican a los ficheros *HTML*.

SSL-PARAMS.CONF.J2

Realizamos la implementación de este fichero en el *host* remoto para completar la configuración respecto a la seguridad del servicio web que se va a implantar.

HANDLERS

MAIN.YML

```
---
- name: Restart apache2
  service:
    name: "{{ service }}"
    state: restarted
```

Haciendo uso del módulo *service*, creamos un *handler* para reiniciar el servicio de *Apache* cada vez que sea necesario.

TASKS

MAIN.YML

```
- name: Install required packages (for apache and php)
  package:
    name: "{{ item }}"
    state: present
  with_items:
    - apache2
    - php
    - libapache2-mod-php
```

Utilizando un bucle, instalamos los paquetes necesarios para la instalación de *Apache* y *PHP*.

- LIBAPACHE2-MOD-PHP: este paquete proporciona el módulo PHP para el servidor web *Apache2*.

```
- name: Enable firewall rules | HTTP | HTTPS
  ufw:
    rule: allow
    port: "{{ item }}"
    proto: tcp
  with_items:
    - "{{ port_http }}"
    - "{{ port_https }}"
```

Utilizando el módulo *ufw* y un bucle, habilitamos los puertos del HTTP y el HTTPS mediante las variables ubicadas en *defaults*.

```
- name: Enable module
  apache2_module:
    name: rewrite
    state: present
  notify:
    - Restart apache2
```

Se habilita el módulo *rewrite* de *Apache*. Este módulo provisto por Apache que permite crear URLs alternativas a las URLs dinámicas generadas por la forma en que están programadas nuestras aplicaciones web. Se hace uso de este módulo para redirigir el tráfico de http a https.

MODULO: APACHE2_MODULE

Se encarga de habilitar o deshabilitar un módulo específico del servidor web de *Apache2*.

```
- name: Enable apache service
  service:
    name: "{{ service }}"
    enabled: yes
```

Mediante esta tarea habilitamos el servicio de *Apache* para que inicie en el arranque del sistema.

```
- name: Hosts | File configuration
  lineinfile:
    path: /etc/hosts
    line: "{{ inventory_hostname }}" "{{ domain }}"
```

HOSTS: archivo que almacena nombres de *hosts* con sus correspondientes direcciones IP.

Editamos el fichero especificado para añadir la siguiente línea mediante variables: la IP de nuestro *host* y el nombre del dominio.

Gracias a esto podremos acceder a nuestro servicio web implantado mediante el nombre del dominio sin tener que acceder mediante la IP o poniendo *localhost*.

MODULO: LINEINFILE

Este módulo asegura que una línea en particular este en un archivo o reemplaza una línea existente usando una expresión regular referenciada.

PATH: se especifica el archivo a modificar.

LINE: parámetro donde se especifica la línea para insertar o reemplazar en el archivo de destino.

Las siguientes tareas que se detallan a continuación son necesarias para la creación de un certificado SSL autofirmado para *Apache*. Un certificado SSL ⁴⁰ es un título digital que autentifica la identidad de un sitio web y cifra con tecnología SSL la información que se envía al servidor.

Que sea autofirmado implica que no hay una autoridad certificadora, es decir, puede ser construido por cualquier persona.

Para generar un certificado CSR en el servidor HTTP Apache, tenemos que realizar las siguientes 3 tareas que se detallan a continuación.

```
- name: Generate key pair
  openssl_privatekey:
    path: "/etc/ssl/private/{{ domain }}.key"
```

Generamos una clave privada y la guardamos en la ruta especificada con el nombre del dominio.

MODULO: OPENSSL_PRIVATEKEY

Módulo encargado de generar claves privadas OpenSSL.

```
- name: Create directory for CSR files
  file:
    path: /etc/ssl/csr
    state: directory
    mode: '0755'
```

Creamos el directorio donde se va a almacenar los ficheros CSR y le asignamos permisos de lectura y ejecución a todos los usuarios excepto al propietario que lo tiene al completo (lectura, escritura y ejecución).

⁴⁰ Secure sockets layer.

MODULO: FILE

Este módulo es capaz de:

- Crear y eliminar archivos, directorios y enlaces simbólicos.
- Modificar permisos y propiedades de archivos y directorios.

```
- name: Generate CSR
  openssl_csr:
    path: "/etc/ssl/csr/{{ domain }}.csr"
    privatekey_path: "/etc/ssl/private/{{ domain }}.key"
    country_name: ES
    organization_name: Cesa
    email_address: cesa@admin.com
    common_name: "{{ domain }}"
```

Generamos el CSR haciendo uso del propio modulo que ofrece Ansible.

MODULO: OPENSSL_CSR

Este módulo permite generar solicitudes de firma de certificado *OpenSSL*. Hace uso de la biblioteca *pyOpenSSL* de *Python* para interactuar con *OpenSSL*.

PATH: Nombre que se le asigna al CSR que se va a generar.

PRIVATEKEY_PATH: parámetro que hace referencia a la ruta de la clave privada que se utilizará al firmar el CSR.

COUNTRY_NAME: ciudad donde se encuentra la organización.

ORGANIZATION_NAME: nombre del departamento dentro de la organización.

EMAIL_ADDRESS: correo electrónico al que queremos que se asigne el certificado.

COMMON_NAME: se especifica el dominio al que va a ser el certificado. Importante indicar el nombre completo de la web.

```
- name: Generate cert
  openssl_certificate:
    path: "/etc/ssl/certs/{{ domain }}.crt"
    privatekey_path: "/etc/ssl/private/{{ domain }}.key"
    csr_path: "/etc/ssl/csr/{{ domain }}.csr"
    provider: selfsigned
```

Una vez tengamos el CSR y la clave privada, generamos el certificado.

MODULO: OPENSSL_CERTIFICATE

Módulo que permite generar certificados *OpenSSL*.

PATH: ruta donde queremos que se genere el certificado.

PRIVATEKEY_PATH: ruta para especificar la llave privada.

CSR_PATH: ruta para especificar el archivo CSR.

PROVIDER: nombre del proveedor a utilizar para generar el certificado OpenSSL.

```
- name: Create root directory structure | Assign permissions to apache user
  file:
    path: "/var/www/{{ domain }}/public_html"
    state: directory
    mode: '0755'
    owner: www-data
    group: www-data
```

Generamos el directorio donde se van a almacenar los ficheros del sitio web a mostrar.

Por defecto, el directorio que de forma predeterminada crea apache para alojar un sitio web se encuentra en la siguiente ruta: */var/www/html*.

En nuestro caso generamos un directorio especializado para nuestro *host*. Le asignamos permisos de lectura y ejecución a todos los usuarios excepto al propietario que tiene los permisos al completo.

Mediante los parámetros `OWNER` y `GROUP` especificamos el propietario y el grupo al que pertenece el directorio.

```
- name: Create ssl-params.conf
  copy:
    src: ssl-params.conf.j2
    dest: /etc/apache2/conf-available/ssl-params.conf
```

En esta tarea copiamos el fichero *ssl-params-conf.j2* al *host* remoto en la ruta especificada en el parámetro *dest*.

```
- name: Enable apache changes | SSL
  apache2_module:
    name: ssl
    state: present
  notify:
    - Restart apache2
```

Mediante el módulo de *Apache* específico para *Ansible* habilitamos los cambios que hemos configurado respecto al SSL y llamamos al *handler* que se encarga de reiniciar el servicio de *Apache*.

```
- name: Enable apache changes | Headers
  apache2_module:
    name: headers
    state: present
  notify:
    - Restart apache2
```

En la siguiente tarea habilitamos otro módulo necesario para la configuración de *Apache*. Este módulo proporciona directivas para controlar y modificar encabezados de solicitud y respuesta HTTP. Y volvemos a hacer uso del *handler* para reiniciar el servicio.

```
- name: Enable ssl-params
  command: a2enconf ssl-params
  notify:
    - Restart apache2
```

El comando *a2enconf* habilita o deshabilita un archivo de configuración en *Apache*. Mediante este comando habilitamos las configuraciones que se han realizado respecto a los ficheros SSL y finalmente llamamos al *handler* para el reinicio de *Apache*.

MODULO: COMMAND

Como no disponemos de ningún módulo de *Ansible* que se encargue de realizar dicho comando tenemos que utilizar el módulo *command*, se encarga de ejecutar un comando en el nodo seleccionado. Especificamos el comando seguido de una lista de argumentos.

```
- name: Configure virtualhost file
  template:
    src: "virtualhost.conf.j2"
    dest: "/etc/apache2/sites-available/10-{{ domain }}.conf"
    backup: yes
```

En la siguiente tarea copiamos la plantilla que tenemos con

Los archivos *virtual host*, son archivos que donde se especifica la configuración actual de un *virtual host* y se indica como el servidor *Apache* va a responder a varias solicitudes de dominio. Posteriormente se explicarán los componentes del fichero y como funciona.

MODULO: TEMPLATE

La función de este módulo consiste en procesar plantillas mediante el lenguaje de plantillas *jinja2*. Estas plantillas contienen variables y/o expresiones que se reemplazan con valores.

SRC: ruta a la plantilla que se encuentra ubicada en la carpeta *templetes* dentro del rol de *Apache*.

DEST: parámetro que hace referencia a la ubicación donde se va a implementar la plantilla.

BACKUP: realiza una copia de seguridad del archivo para poder recuperarlo en el caso de que se haya producido algún incidente.

```
- name: Copy index.html
  copy:
    src: index.html
    dest: "/var/www/{{ domain }}/public_html/"
    mode: '0664'

- name: Copy files for web

  copy:
    src: "{{ item }}"
```

```

dest: "/var/www/{{ domain }}/public_html/"
mode: '0664'
with_items:
  - index.html
  - inicio.php
  - registro.php
  - formulario.php
  - CSS

```

Copiamos todos los archivos necesarios para el despliegue de la página web .

```

- name: Enable changes
  command: "a2ensite 10-{{ domain }}"
  notify:
    - Restart apache2

```

En la última tarea, volvemos a hacer uso del módulo command para activar nuestro sitio web virtual en el servidor de apache con el comando a2ensite.

TEMPLATES

VIRTUALHOST.CONF.J2

```

<Virtualhost *:{{ port_http }}>
  ServerAdmin admin@{{ domain }}
  ServerName {{ domain }}
  Redirect / https://{{ domain }}
</Virtualhost>
<VirtualHost _default_:{{ port_https }}>
  ServerAdmin admin@{{ domain }}
  ServerName {{ domain }}

  DocumentRoot /var/www/{{ domain }}/public_html

  <Directory /var/www/{{ domain }}/public_html>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

  SSLEngine on
  SSLCertificateFile      /etc/ssl/certs/{{ domain }}.crt
  SSLCertificateKeyFile /etc/ssl/private/{{ domain }}.key
</VirtualHost>

```

En este archivo podemos personalizar los datos para el dominio agregando directivas.

Se han establecido dos secciones, la principal función de la primera sección consiste en que cualquier solicitud que se realice por el puerto 80 (HTTP) va a ser redirigida por el puerto seguro 443 al HTTPS.

Editamos algunas directivas como:

- `SERVERAMDIN`: para establecer un correo electrónico donde el administrador pueda recibir correos.
- `SERVERNAME`: que establece el dominio base que debe coincidir para la definición del virtual *host*.
- `SERVERALIAS`: define nombres alternativos, también llamados alias, al servidor virtual (solo funciona dentro de una directiva *VirtualHost*)
- `DOCUMENTROOT`: indica la raíz de los documentos de apache.

Mediante el parámetro `ERRORLOG` indicamos la ruta al archivo log de errores y mediante `CUSTOMLOG` creamos en los logs personalizados.

VARs

MAIN.YML

```
---  
service: apache2
```

En este fichero especificamos las variables que se van a utilizar en este rol. En este caso establecemos el nombre del servicio de apache para la distribución *Linux* con la que trabajamos.

JUEGO DE PRUEBAS

A continuación, se procederá a detallar los pasos a seguir para la implementación de toda la estructura detallada anteriormente.

El primer paso es comprobar que tenemos conexión desde el servidor de *Ansible* al nodo que queremos controlar que en este caso es nuestra máquina cliente implantada con *VirtualBox*.

Mediante la opción `-m` indicamos que haga uso del módulo *ping* para comprobar si tenemos conexión entre ambas máquinas. Si todo funciona correctamente nos devuelve el siguiente mensaje:

```
root@server:/home/server/git/tfg-lamp-ansible# ansible -m ping lamp
192.168.0.26 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Una vez hemos comprobado tenemos *ping* con el host remoto, procedemos a lanzar el *playbook* principal haciendo uso del parámetro `--ask-vault-pass` para introducir la contraseña del archivo que ciframos anteriormente y comenzará la ejecución de todas las tareas especificadas:

INICIO

```
root@server:/home/server/git/tfg-lamp-ansible# ansible-
playbook playbook.yml --ask-vault-pass -I /inventories/virtualbox/hosts
Vault password:

PLAY [lamp] *****

TASK [Gathering Facts] *****
ok: [192.168.0.26]

TASK [common : Update repositories] *****
ok: [192.168.0.26]

TASK [common : Upgrading all the apt packages] *****
changed: [192.168.0.26]

TASK [common : Install common packages] *****
changed: [192.168.0.26]

TASK [common : Enable firewall] *****
changed: [192.168.0.26]
```

FIN

```
TASK [apache2 : Copy index.html] *****
changed: [192.168.0.26]

TASK [apache2 : Copy php for DB] *****
changed: [192.168.0.26]

TASK [apache2 : Enable changes] *****
changed: [192.168.0.26]

RUNNING HANDLER [apache2 : Restart apache2] *****
changed: [192.168.0.26]

PLAY RECAP *****
192.168.0.26      : ok=33    changed=30    unreachable=0    failed=0
skipped=0      rescued=0    ignored=0
```

Al final de la ejecución del playbook Ansible nos ofrece un resumen informativo sobre cómo se han ejecutado las tareas. En este caso nos muestra que se han ejecutado todas las tareas correctamente.

ANÁLISIS DE LOS RESULTADOS OBTENIDOS PUNTOS DE MEJORA.

Los resultados obtenidos durante el desarrollo de este trabajo corresponden con los objetivos fijados al inicio.

Se ha completado la instalación y configuración de las herramientas necesarias para llevar a cabo la automatización de una infraestructura tecnológica mediante entornos virtualizados.

También se ha completado satisfactoriamente el aprendizaje básico sobre *Ansible* y a su vez se han reforzados los conocimientos ya sabidos en cuando al manejo y configuración de un servidor de aplicaciones web de una forma distinta y más funcional.

En este proyecto se ha cubierto la automatización de servidores mediante tecnologías de virtualización, pero como propuesta de futuro se ha pensado el hecho de intentar automatizar más aun en la medida de lo posible dicha estructura y hacer uso de *vagrant* (es una herramienta de código abierto para la creación de ambientes virtuales de desarrollo) para el despliegue de las máquinas virtuales mediante código.

También se intentaría mejorar la integración continua de software haciendo uso de otra herramienta denominado *jenkins*. es una herramienta que permite ejecutar de forma inmediata pruebas unitarias y ofrece una monitorización continua de las métricas de calidad del proyecto.

ANÁLISIS DE LAS COMPETENCIAS ADQUIRIDAS

Se ha adquirido la capacidad para seleccionar, diseñar, desplegar e integrar infraestructuras tecnológicas de una forma óptima.

La infraestructura que se ha diseñado permite automatizar un nodo o un grupo de nodos y tras diversas pruebas de distintas herramientas finalmente fueron seleccionadas las que permiten una mejor optimización del sistema.

Así mismo, también se han adquirido las capacidades necesarias para comprender, aplicar y gestionar la seguridad en las infraestructuras creadas.

CONCLUSIÓN

Poniendo fin a este documento, se procede a realizar las principales conclusiones que se han sacado a lo largo de su desarrollo.

Teniendo en cuenta que he partido de una tecnología completamente desconocida como lo es *Ansible* he sido capaz de implantar un servidor LAMP de forma automatizada gracias a las tecnologías ya conocidas y las aprendidas.

Como bien se ha comentado ya en numerosas ocasiones, el tema principal sobre el que abarca este TFG es el campo de la automatización, y después de finalizar mi proyecto puedo comprender que resulta de gran importancia en el mundo laboral, pues el hecho de integrar una herramienta como la que se ha estado utilizando supone una gran ventaja respecto al ahorro de tiempo y esfuerzo, reduciendo y evitando los posibles errores humanos.

Creo que es una buena apuesta continuar con el estudio de la automatización, ya que estos servicios cada vez están más demandados por las empresas y creo, personalmente, que es un campo que actualmente se encuentra en auge en el sector de la informática.

Este proyecto pone fin a una gran etapa de mi vida. Durante mi recorrido como estudiante he tenido la fortuna de adquirir nuevos conocimientos de todas las asignaturas impartidas y, sobre todo, he aprendido que con paciencia, trabajo y esfuerzo todo es posible.

BIBLIOGRAFÍA

<https://www.udemy.com/course/ansible-automatizacion-de-principiante-a-experto/>

<https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html>

https://docs.ansible.com/ansible/latest/modules/apt_module.html

https://docs.ansible.com/ansible/latest/modules/package_module.html

<https://docs.ansible.com/ansible/latest/plugins/lookup/items.html>

https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

https://docs.ansible.com/ansible/latest/modules/ufw_module.html

https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

https://docs.ansible.com/ansible/latest/modules/service_module.html

https://docs.ansible.com/ansible/latest/modules/mysql_user_module.html

https://docs.ansible.com/ansible/latest/reference_appendices/logging.html

https://docs.ansible.com/ansible/latest/modules/mysql_db_module.html

https://docs.ansible.com/ansible/latest/modules/copy_module.html

https://docs.ansible.com/ansible/latest/modules/cron_module.html

https://docs.ansible.com/ansible/latest/modules/apache2_module_module.html

https://docs.ansible.com/ansible/latest/modules/lineinfile_module.html

https://docs.ansible.com/ansible/latest/modules/openssl_privatekey_module.html

https://docs.ansible.com/ansible/latest/modules/openssl_certificate_module.html

https://docs.ansible.com/ansible/latest/modules/openssl_csr_module.html

https://docs.ansible.com/ansible/latest/modules/template_module.html

https://docs.ansible.com/ansible/latest/modules/command_module.html

ANEXO

- [LINK](#) carpeta google drive máquinas virtuales y readme.txt con pasos a seguir.
- [LINK](#) repositorio github.