Implementación de nube privada para pequeña empresa

Antonio Javier Cordero Maestre

Grado de Ingeniería Informática

Área: GNU/Linux

Tutor: Álvaro del Álamo Cortés

05 / Enero / 2024

FICHA DEL TRABAJO FINAL

Título del trabajo:	Implementación de nube privada para pequeña empresa
Nombre del autor:	*Antonio Javier Cordero Maestre*
Nombre del consultor/a:	Joaquín López Sánchez-Montañés
Nombre del PRA:	Montse Serra Vizern
Fecha de entrega (mm/aaa):	01/2024
Titulación:	Grado Ingeniería Informática
Área del Trabajo Final:	*GNU/Linux*
Idioma del trabajo:	*Castellano*
Palabras clave	Ansible, Terraform, Proxmox

Resumen del Trabajo

El objetivo de este trabajo de evaluación continua es utilizar el hipervisor Proxmox en un servidor HP Proliant DL360 G7 para automatizar tareas y crear una plataforma completa de servicios orientada a la pequeña empresa. Se busca lograr que, a través de herramientas como Ansible, Terraform, entre otras, sea posible definir y mantener toda la arquitectura de la plataforma en ficheros, permitiendo tener un estado actualizado y real de la misma haciendo uso, además, de lo que en la actualidad se puede considerar equipo hardware desfasado.

Se incluirán servicios como base de datos, web, etc. dentro de la plataforma, además de las configuraciones de seguridad necesarias para obtener un entorno de trabajo seguro y asequible, capaz de conseguir un rendimiento de trabajo adecuado en el entorno empresarial planteado.

Para ello, se hará uso, tal y como hemos referido anteriormente, del mayor número de elementos de **software libre** disponibles lo cual se traduce en un ahorro económico para la pequeña empresa sin afectar a la productividad de los servicios entregados.

Abstract

The objective of this continuous assessment work is to use the Proxmox hypervisor on an HP Proliant DL360 G7 server to automate tasks and create a complete service platform oriented towards small businesses. The aim is to achieve, through tools like Ansible, Terraform, among others, the ability to define and maintain the entire platform architecture in files, allowing for an updated and real-time state of the platform, even using what can currently be considered outdated hardware equipment.

Services such as databases, web, etc. will be included within the platform, along with the necessary security configurations to obtain a secure and affordable working environment capable of achieving suitable performance in the proposed business environment.

To accomplish this, as previously mentioned, the maximum number of available open-source software elements will be used, resulting in cost savings for the small business without affecting the productivity of the delivered services.

Índice

1. Introducción

1.1 Contexto y justificación del Trabajo

La motivación de realizar este TFG es verificar que, haciendo uso de herramientas de **software** libre, se pueden lograr los mismos o mejores resultados que obtienen las grandes corporaciones a partir de unas inversiones económicas realmente reducidas.

Además, se aprovecharán elementos de **hardware** que están en el mercado disponibles y que son considerados temporalmente desfasados, pero cuyo aprovechamiento puede beneficiar no sólo al pequeño empresario, también al entorno, ya que se reduce la generación de residuos electrónicos debido a la obsolescencia con el consiguiente beneficio para la huella de carbono.

1.2 Objetivos del Trabajo

El objetivo de este trabajo de evaluación continua es utilizar el hipervisor Proxmox en un servidor HP Proliant DL360 G7 para automatizar tareas y crear una plataforma completa de servicios orientada a la pequeña empresa. Se busca lograr que, a través de herramientas como Ansible, Terraform, entre otras, sea posible definir y mantener toda la arquitectura de la plataforma en ficheros, permitiendo tener un estado actualizado y real de la misma haciendo uso, además, de lo que en la actualidad se puede considerar equipo hardware desfasado.

Se incluirán servicios como base de datos, web, correo, etc. dentro de la plataforma, además de las configuraciones de seguridad necesarias para obtener un entorno de trabajo seguro y asequible, capaz de conseguir un rendimiento de trabajo adecuado en el entorno empresarial planteado.

Para ello, se hará uso, tal y como hemos referido anteriormente, del mayor número de elementos de **software libre** disponibles lo cual se traduce en un ahorro económico para la pequeña empresa sin afectar a la productividad de los

servicios entregados.

1.3 Enfoque y método seguido

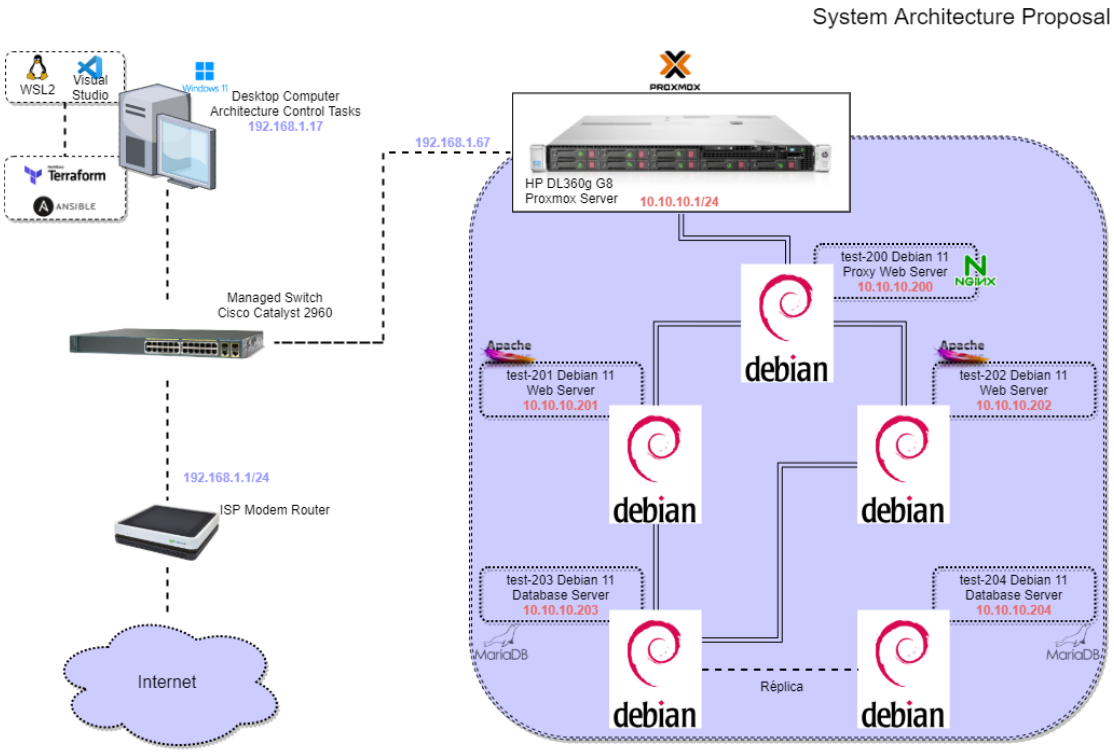
Para llevar a cabo este proyecto, se utilizará una metodología basada en **DevOps**. Se seguirán los principios de integración y entrega continua, lo que permitirá automatizar el despliegue y la configuración de los servicios en la plataforma.

Se utilizarán herramientas de automatización, como Ansible y Terraform, para gestionar la infraestructura y la configuración de los servicios de **manera eficiente** y **reproducible**.

Además, se hará uso de herramientas de monitorización de sistemas de software libre.

1.4 Arquitectura Propuesta

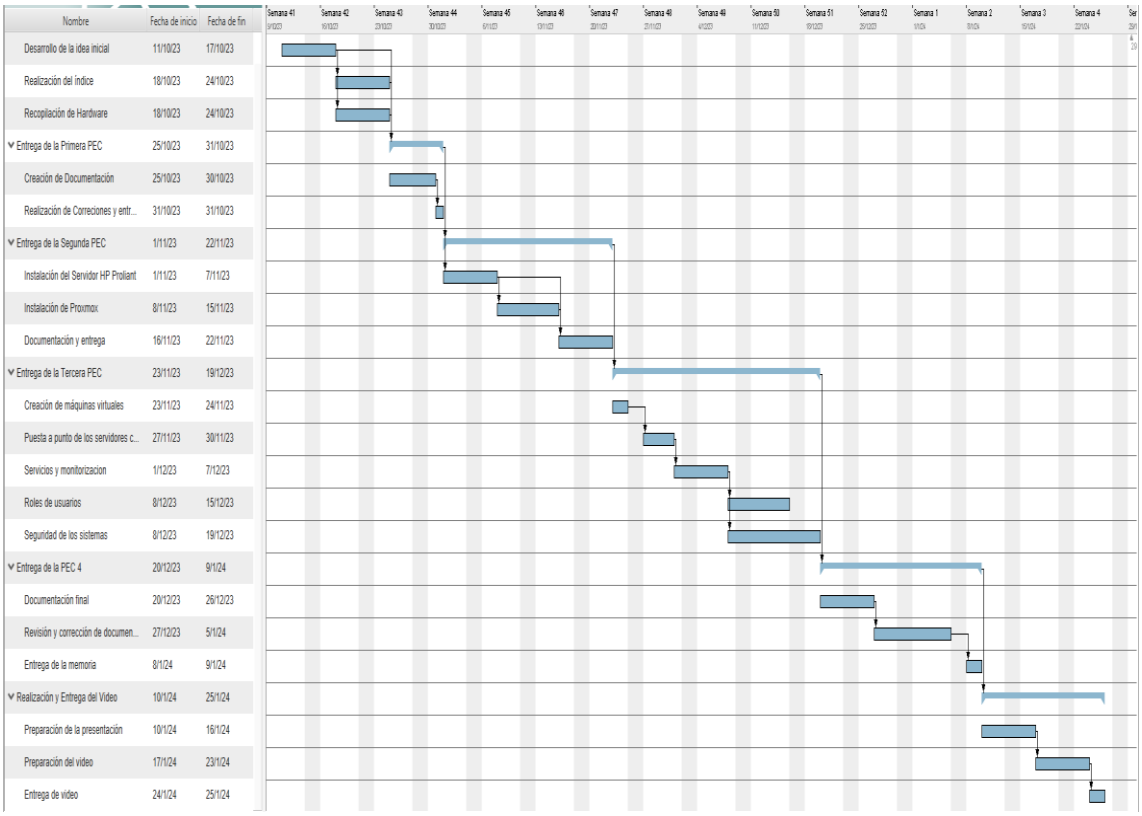
Partiendo que la finalidad del proyecto será dotar a una hipotética pequeña o mediana empresa de los recursos necesarios para poder tener presencia en el mundo digital de Internet, aprovechando equipos cuyo ciclo de vida está consumido y herramientas de software libre, se propone la siguiente arquitectura:



1.5 Planificación del Trabajo

Nombre	Fecha de inicio	Fecha de fin
Desarrollo de la idea inicial	11/10/23	17/10/23
Realización del índice	18/10/23	24/10/23
Recopilación de Hardware	18/10/23	24/10/23
⌄ Entrega de la Primera PEC	25/10/23	31/10/23
Creación de Documentación	25/10/23	30/10/23
Realización de Correciones y entr...	31/10/23	31/10/23
⌄ Entrega de la Segunda PEC	1/11/23	22/11/23
Instalación del Servidor HP Proliant	1/11/23	7/11/23
Instalación de Proxmox	8/11/23	15/11/23
Documentación y entrega	16/11/23	22/11/23
⌄ Entrega de la Tercera PEC	23/11/23	19/12/23
Creación de máquinas virtuales	23/11/23	24/11/23
Puesta a punto de los servidores c...	27/11/23	30/11/23
Servicios y monitorizacion	1/12/23	7/12/23
Roles de usuarios	8/12/23	15/12/23
Seguridad de los sistemas	8/12/23	19/12/23
⌄ Entrega de la PEC 4	20/12/23	9/1/24
Documentación final	20/12/23	26/12/23
Revisión y corrección de documen...	27/12/23	5/1/24
Entrega de la memoria	8/1/24	9/1/24
⌄ Realización y Entrega del Video	10/1/24	25/1/24
Preparación de la presentación	10/1/24	16/1/24
Preparación del video	17/1/24	23/1/24
Entrega de video	24/1/24	25/1/24

El diagrama de Gantt de la realización del proyecto ha sido el siguiente:



1.6 Breve sumario de productos obtenidos

Utilizando Terraform se han generado una serie de máquinas virtuales de manera automatizada para su paso a producción dentro de la empresa.

Además, utilizando ansible se han definido los roles de servidor de base de datos y servidor de web, con los cuales se han definido las máquinas previamente creadas mediante ansible.

1.7 Breve descripción de los otros capítulos que componen la memoria

Capítulos	Descripción
2	Elementos de Hardware principales
3	Instalación de Proxmox
4, 5, 6	Herramientas de automatización, configuración e instalación
7,8	Proxmox desde su GUI, creación de máquinas virtuales
9, 10	Primeros playbook con Ansible
11,12	Template en Terraform y su uso.
13,14	Roles lógicos y Roles en Ansible

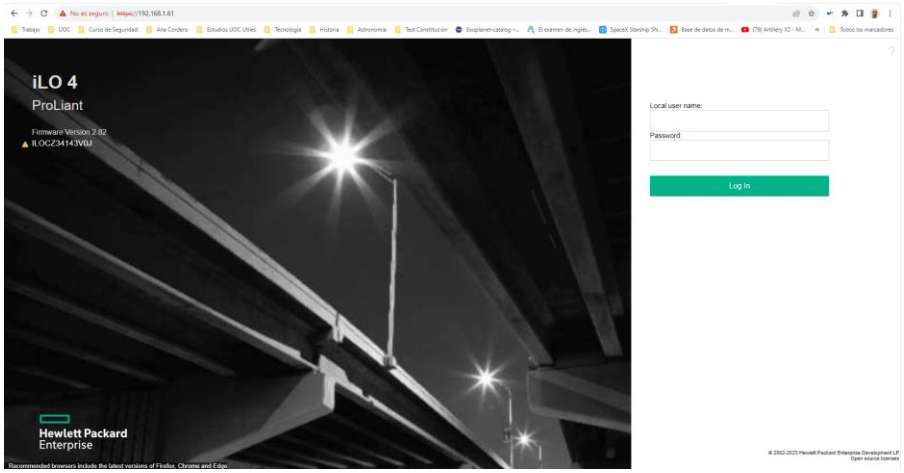
2. Hardware Utilizado para el desarrollo del proyecto.

El servidor **HP Proliant DL360 G7** es una solución versátil y segura, que ofrece un rendimiento excepcional para diversas aplicaciones empresariales. Está diseñado para satisfacer las necesidades de empresas de todos los tamaños y sectores, brindando una plataforma sólida para la gestión de datos y el procesamiento de cargas de trabajo intensivas.

La iLO4 (Integrated Lights-Out versión 4) que incluye dicho servidor, es una interfaz de administración que remota que nos permite controlar y gestionar el servidor a través de una conexión de red.

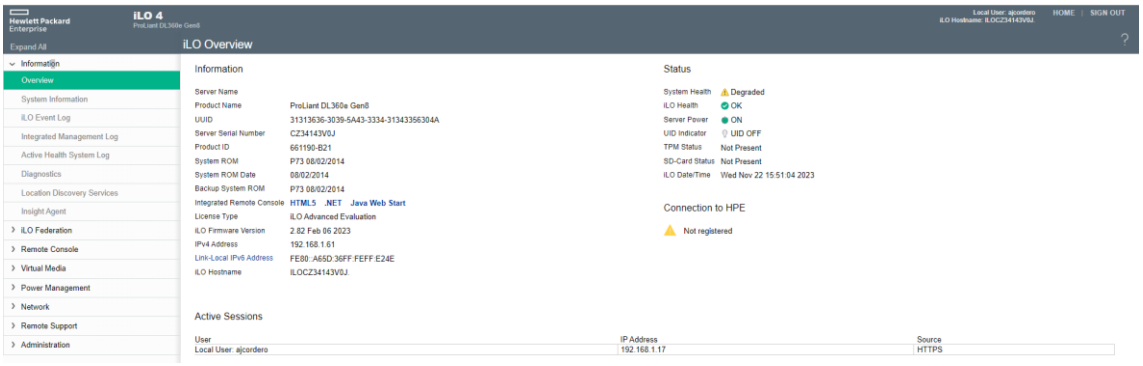
Para ello se configura con una dirección IPv4 propia, a través de la cual procederemos a realizar los accesos, teniendo así un control en todo momento del estado del hardware de nuestro servidor. Además, en el caso de este TFG nos va a permitir conocer de una manera más directa, cuáles son las características específicas del hardware que hemos utilizado para su realización.

Tal y como se indica anteriormente, accederemos a la iLO del servidor mediante la IP configurada en el mismo. En el caso de este TFG, cuando se configuró el servidor, se le asignó la IP 192.168.1.61 para la iLO. Por tanto, los accesos se realizarán a través de dicha dirección IP tal y como se puede ver en la siguiente imagen:



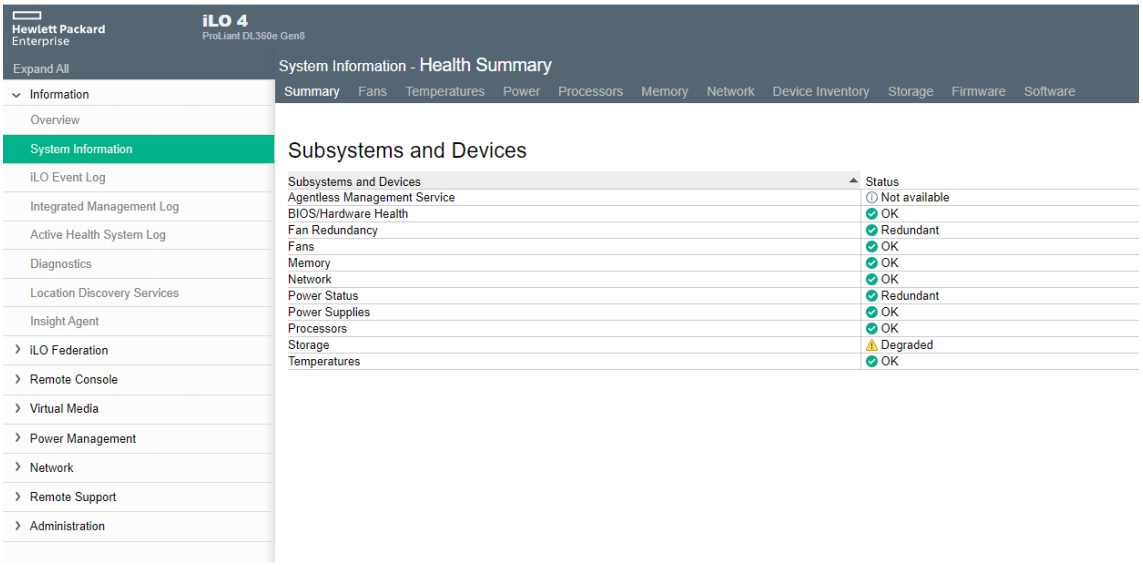
Una vez dentro de la interfaz que la iLO nos ofrece, tras introducir el usuario y contraseña por defecto que se suministra al adquirir dicho equipo, se accede a un resumen del estado del sistema, además de una serie de opciones para poder conocer de

manera más concreta cada uno de los subsistemas y posibilidades de configuraciones que éste nos da.



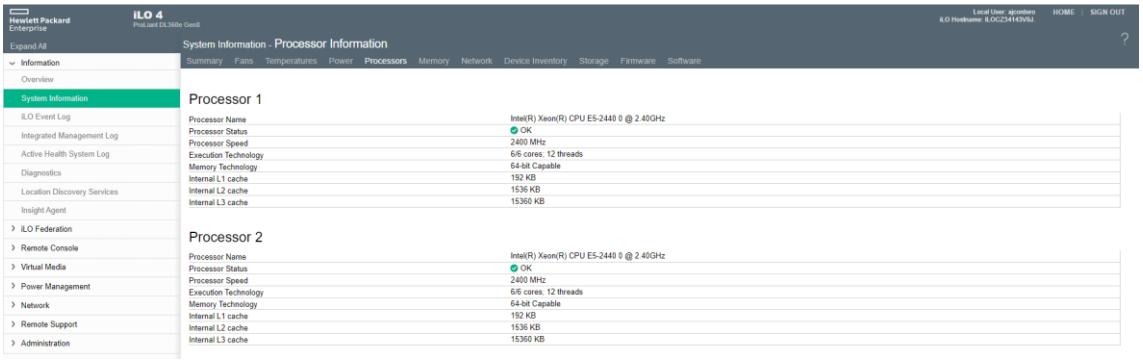
iLO mostrando el estado del sistema

Si deseamos obtener una información más avanzada, seleccionaremos la opción System Information del menú de iLO4. A partir de dicha opción podremos ver las características más específicas de dicho servidor.



System Information

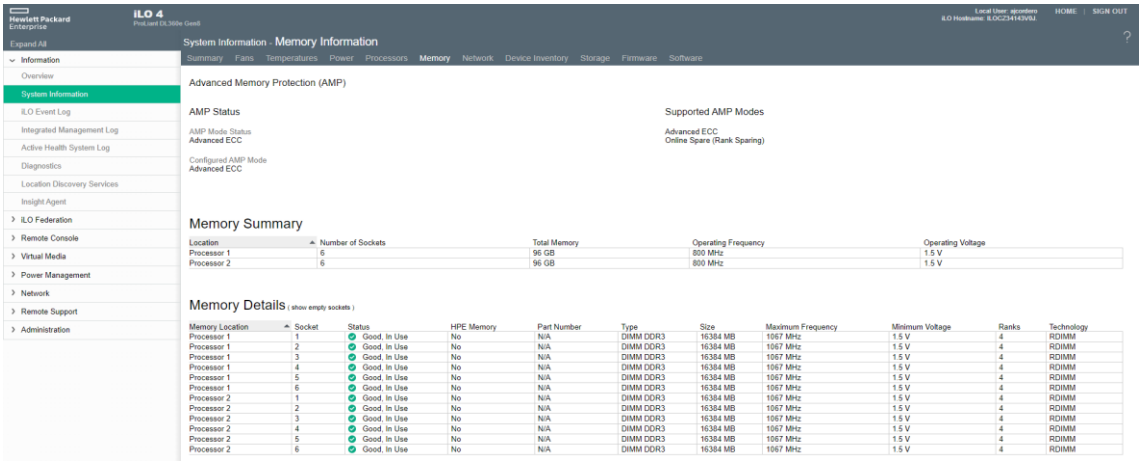
Una de las características más destacables de este servidor es su **potencia de proceso**. Está equipado con procesadores Intel Xeon E5-2440 de 64 bits a 2.40Ghz, lo que garantiza un rendimiento eficiente y rápido.



1Información de los procesadores

Como podemos ver, cada uno de los dos procesadores Intel Xeon E5-2440 que monta el servidor nos ofrece 6 cores y es capaz de trabajar con 12 threads por procesador, lo que nos ofrece un total de **24 threads** de trabajo de manera simultánea.

En términos de memoria, cuenta con una **arquitectura de memoria avanzada** que permite una mayor capacidad de almacenamiento y una mejor administración de la memoria. Tal y como vemos en la siguiente imagen, se ha dotado al sistema de 192 GB de memoria RAM para la realización de este trabajo de fin de grado. Memoria más que suficiente para poder realizar las tareas diseñadas en función de la carga de una pequeña empresa:



2Información referente a la memoria del sistema

La capacidad de **almacenamiento** es otro aspecto destacado del servidor. El DL360 G7 admite múltiples opciones de almacenamiento, incluyendo discos duros SAS y SATA, así como unidades de estado sólido (SSD). Esto proporciona a los usuarios una flexibilidad excepcional para adaptarse a diferentes requisitos de almacenamiento.

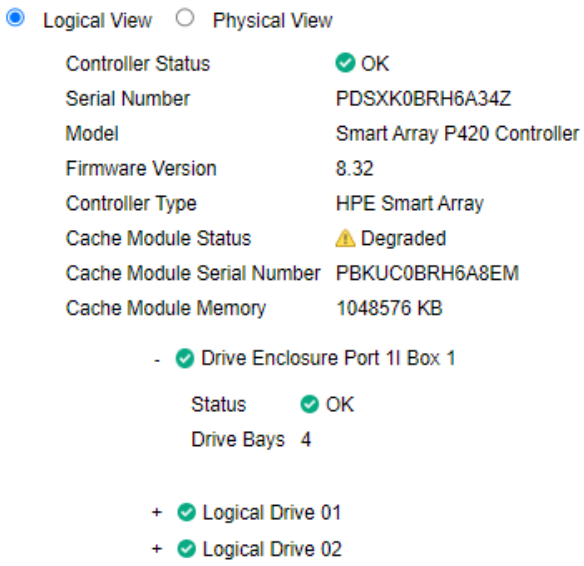
Para la realización de este proyecto, se han montado 4 discos HP 693959 B21 en cada bahía libre que el servidor contiene:

3Disco Duro-SATA HP 653959

Las características técnicas de dichos discos son:

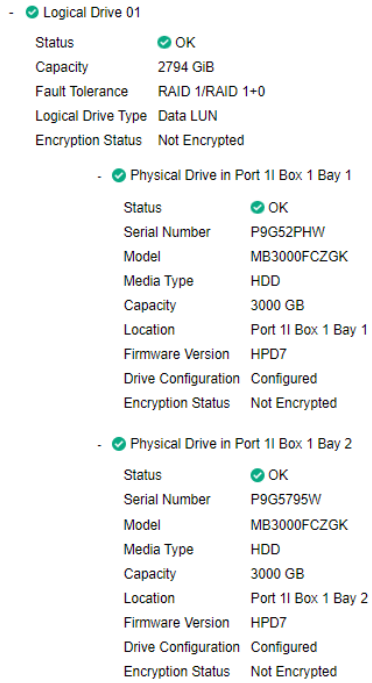
Descripción del disco	
Capacidad	3 TB
Transferencia de datos por segundo	600 MB
Revoluciones por minuto	7200

Por lo tanto, nuestro sistema contará con 12 TB de almacenamiento. En este punto , se debe optar entre dotar a nuestra infraestructura de una capacidad mayor de almacenamiento, o por el contrario de una mayor seguridad de la información contenida en el mismo. Independientemente de las soluciones de copias de seguridad de la información que puedan ser introducidas posteriormente a nivel de máquinas virtuales, sistema operativo y aplicaciones instaladas, se ha optado por asegurar la información de este, creando dos particiones de discos de 3 GB + 3 GB cada uno y dejando los otros dos discos como RAID o espejo de los anteriores. Por lo tanto, la capacidad de almacenamiento efectiva del sistema será de 6 TB.
Podemos ver en la información de la iLO como queda nuestro sistema de almacenamiento a nivel de servidor configurado desde un punto de vista lógico:



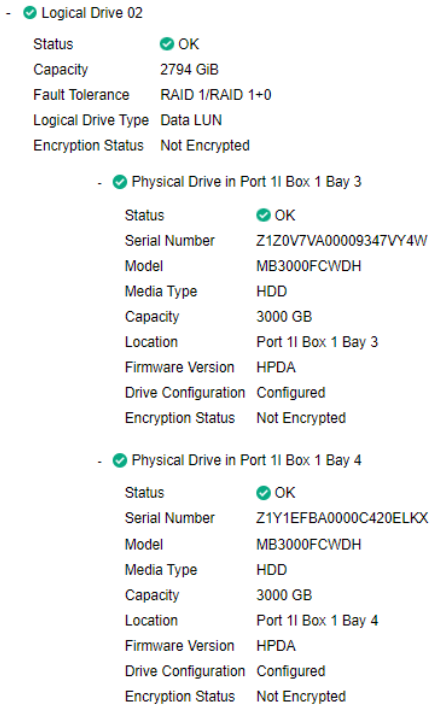
4 Almacenamiento lógico del sistema

Tendremos dos discos presentados al sistema, que serán:



5Primer disco lógico

Y el segundo disco lógico:

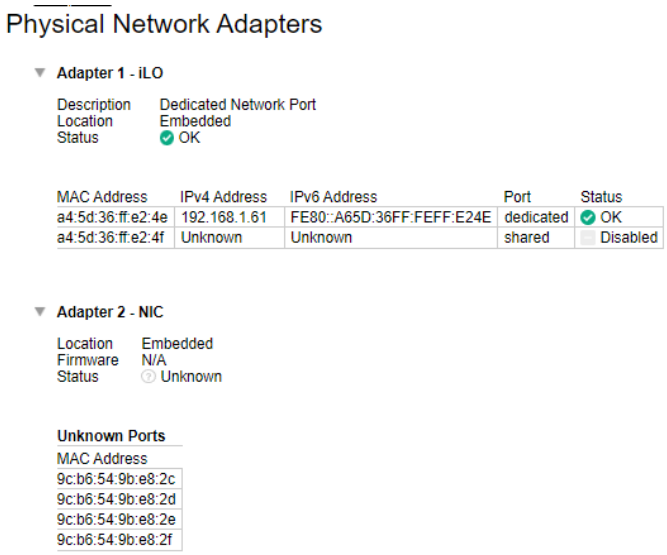


6 - Segundo disco lógico

Desde el punto de vista de la **conectividad**, el servidor HP Proliant DL360 G7 ofrece una amplia gama de opciones. Cuenta con múltiples puertos USB, puertos Ethernet y ranuras PCIe para permitir la conexión de periféricos y la expansión de capacidades. Además, ofrece capacidades de red avanzadas, como la compatibilidad con tecnologías de virtualización y la posibilidad de configurar enlaces de alta velocidad para mejorar la velocidad de transferencia de datos.

Centrándonos en la conexión del servidor a la red, podemos ver desde la iLO en la imagen a continuación, que presenta dos adaptadores de red. Uno para la propia iLO el cual está configurado y funcional en el puerto dedicado. Se le asigna por nuestra parte la IPv4 192.168.1.61 y que como hemos podido verificar anteriormente es la dirección IP de acceso a la iLO del servidor.

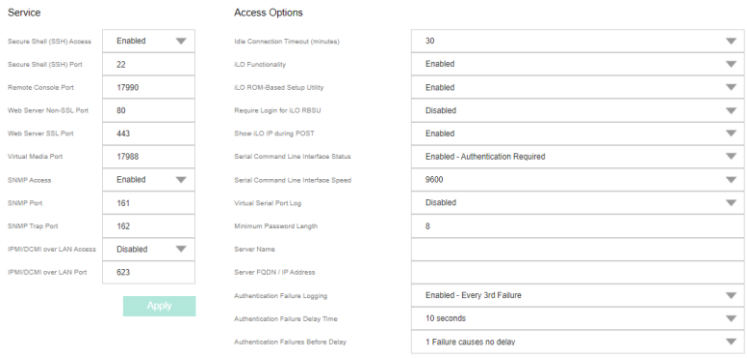
El segundo adaptador, el cual posee cuatro interfaces de red, no está configurado, ya que de ello se encarga el sistema operativo, y será el que comunicará nuestro servidor con el exterior.



7Conexiones de red del servidor

La **seguridad** es otro aspecto crucial que el servidor aborda eficientemente. Cuenta con características de seguridad integradas, como el módulo de plataforma segura (TPM) y la función de bloqueo del servidor. Estas medidas ayudan a proteger los datos sensibles y a prevenir accesos no autorizados.

Además, podemos definir el comportamiento del servidor desde la iLO, abriendo o cerrando o cambiando los valores por defecto de los distintos servicios. Para la realización del TFG , su configuración a nivel de servicios es la siguiente:



8Configuración de los Servicios del Servidor

Podemos ver que tenemos permitidos los accesos por web, web segura , ssh y snmp. Los dos primeros son utilizados para poder acceder a la interfaz de la iLO. SSH lo tenemos habilitado por si es necesario logarse en el interfaz texto del servidor para ejecutar algún comando más avanzado que los que nos permite el interfaz web. Por

último, el servicio SNMP, tal y como veremos a lo largo del TFG, nos permitirá acceder a distintas métricas del sistema y así poder configurar un sistema de monitorización de este así como de las máquinas virtuales y servicios que correrán en él.

El servidor DL360 G7 también se destaca por su capacidad de administración de los **accesos** a usuarios. Incorpora herramientas de gestión intuitivas que permiten a los administradores supervisar y controlar el servidor de manera efectiva. Podremos realizar la gestión de los usuarios que pueden acceder a la iLO, realizar una gestión por ROLES, configurar los tipos de acceso mediante pares de claves, etc.

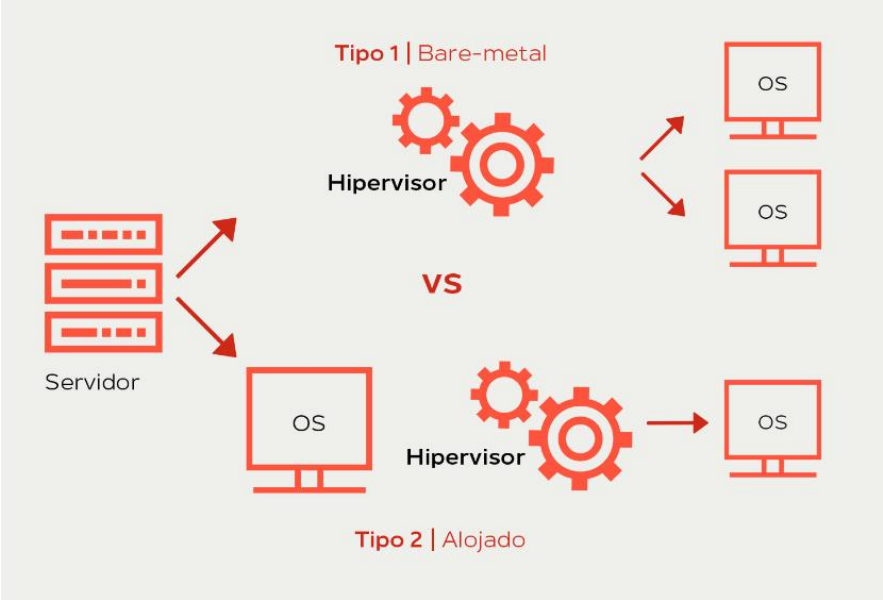
En términos de **eficiencia energética**, el servidor HP Proliant DL360 G7 es altamente eficiente en el consumo de energía. Está diseñado para reducir el consumo de energía sin comprometer el rendimiento, lo que ayuda a las organizaciones a ahorrar costos y reducir su huella de carbono.

En conclusión, el servidor HP Proliant DL360 G7 es una solución de **alto rendimiento** y **confiable** para la pequeña y mediana empresa a pesar de los años que han pasado desde su salida al mercado. Este servidor es una opción sólida para aquellas que buscan una plataforma robusta y versátil para gestionar sus cargas de trabajo empresariales a un bajo precio y gestionando de una manera eficiente la huella ecológica al aprovechar tecnología destinada a ser dada de baja.

3. Instalación y configuración de Proxmox.

Proxmox es una **plataforma de virtualización de servidores de código abierto** que permite a los usuarios crear y gestionar entornos virtuales. Con Proxmox, las organizaciones pueden aprovechar al máximo los recursos hardware y maximizar la utilización de servidores mediante la virtualización, lo que resulta en ahorro de costos y una mayor flexibilidad en la gestión de su infraestructura.

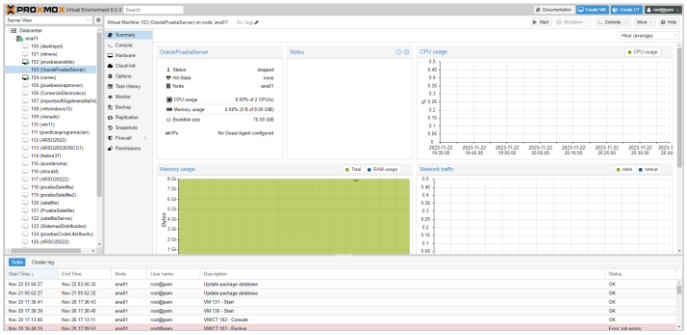
Proxmox se basa en el uso del **hipervisor** de código abierto **KVM** (Kernel-based Virtual Machine), que proporciona una virtualización completa y de alto rendimiento para distintos sistemas operativos (GNU/Linux, Windows, Oracle, etc.) . Además, Proxmox también hace uso de contenedores basados en tecnología de virtualización de Linux llamada **LXC** (Linux Containers), que ofrecen una virtualización ligera y rápida para aplicaciones y servicios. La diferencia entre ambas tecnologías es el aprovechamiento que la segunda hace del Kernel del propio Proxmox, ya que será el propio Kernel de Proxmox el que utilizarán los contenedores, mientras que para las máquinas basadas en KVM será el propio Kernel de la máquina virtual el que deba de instalarse.



9 Tipos de Hipervisores

Una de las principales ventajas de Proxmox es su **interfaz de usuario** intuitiva y fácil de usar. Proporciona un panel de control web llamado Proxmox VE (Virtual Environment) que permite a los administradores gestionar y monitorizar sus entornos

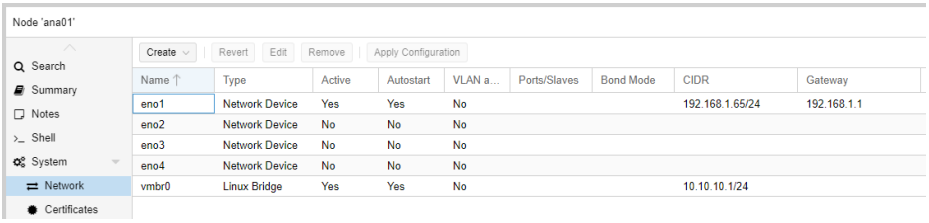
virtuales de manera centralizada. Desde el panel de control, los usuarios pueden crear, clonar y migrar máquinas virtuales, así como administrar redes y almacenamiento.



10 Interfaz de usuario de Proxmox

Proxmox también ofrece herramientas de **gestión de almacenamiento** integradas las cuales nos proporcionan un sistema de archivos robusto y escalable con capacidades de snapshots y replicación. Esto permitirá realizar copias de seguridad y restauraciones rápidas de sus datos y garantizar la integridad de la información almacenada.

Desde el punto de vista de **la Red y conectividad**, Proxmox nos permitirá realizar las configuraciones necesarias para dotar de una salida a la red a las máquinas virtuales y contenedores que ejecuta, así como realizar una gestión eficiente de la seguridad a nivel de red de los mismos. Así , en la siguiente imagen, podemos ver un ejemplo de configuración de la red que el hipervisor, que se está ejecutando en el nodo, compartirá con sus máquinas virtuales y contenedores:



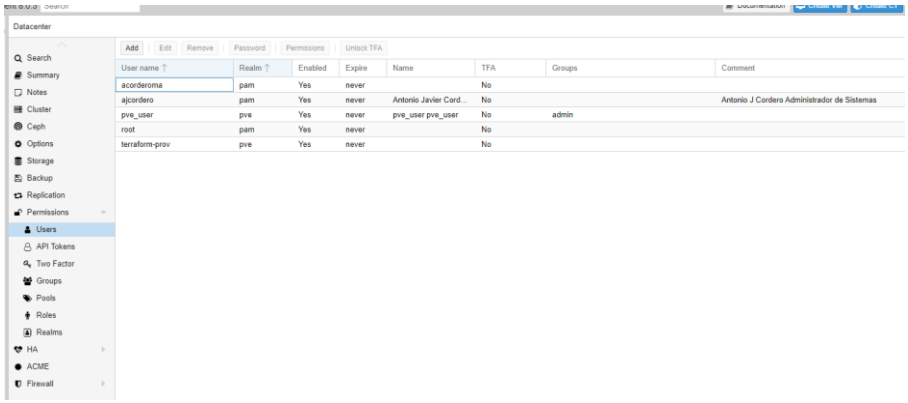
11Configuración de los interfaces de red de un nodo en Proxmox

Además, Proxmox ofrece una funcionalidad avanzada de **alta disponibilidad (HA)** que permite a los usuarios configurar clústeres de servidores para garantizar la continuidad del servicio en caso de fallos. Con la función de alta disponibilidad, en el caso de existir problemas en uno de los nodos que componen el clúster, las máquinas virtuales se migrarán automáticamente a otros nodos disponibles, lo que minimiza el tiempo de inactividad y asegura la disponibilidad de los servicios. Aunque se aparta de nuestro

TFG, en los Anexos será introducida la gestión de HA mediante Proxmox.

Otra característica destacada de Proxmox es la capacidad de **gestionar y monitorizar recursos a través de API (Application Programming Interface)**. Esto permite a los usuarios **automatizar** tareas y realizar integraciones personalizadas con otras herramientas y sistemas de gestión.

Desde el lado de la seguridad, tenemos garantizado el **acceso seguro** de los usuarios y roles configurados mediante la interfaz de gestión de Proxmox, que nos permitirá definir el quién y el qué puede realizar determinado individuo accediendo al hipervisor, tal y como podemos ver en la imagen a continuación:



12Gestión de Roles y Usuarios utilizando Proxmox

Proxmox es por tanto, una **plataforma de virtualización de servidores** de código abierto que ofrece una solución completa y eficiente para la creación y gestión de entornos virtuales, lo que nos permitirá dotar de distintos servicios a la pequeña empresa ejecutándose en un único servidor inicialmente y realizar su escalado en el caso que lo consideremos necesario.

4. Herramientas de automatización.

Para el desarrollo de la practica vamos a utilizar, tal y como anteriormente hemos referido, las herramientas **Terraform** y **Ansible**.

4.1 Terraform

Terraform es una herramienta de código abierto desarrollada por HashiCorp que permite **la creación, configuración y gestión de infraestructuras de manera automatizada**. Con Terraform, los desarrolladores y administradores de sistemas podrán describir su infraestructura como código, y por lo tanto, podrán definir y desplegar recursos de infraestructura utilizando un lenguaje descriptivo y reutilizable.

La principal ventaja de utilizar Terraform radica en su enfoque basado en la infraestructura como código (IaC). Esto significa que, en lugar de configurar manualmente los recursos de infraestructura, como servidores, redes o bases de datos, se pueden escribir archivos de configuración en un lenguaje específico de Terraform llamado **HCL** (HashiCorp Configuration Language).

Al utilizar archivos de configuración en lugar de realizar configuraciones manuales, Terraform ofrece una serie de beneficios. En primer lugar, proporciona una forma fácil de mantener y versionar la infraestructura, ya que los cambios se pueden realizar en los archivos de configuración y controlarse mediante sistemas de control de versiones como Git. Esto permite un control más preciso sobre los cambios y facilita la **colaboración** en equipos.

Además, Terraform es una herramienta multiplataforma, lo que significa que se puede utilizar en diferentes proveedores de infraestructura en la nube, como Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) y muchos otros. Esto brinda flexibilidad a los equipos de desarrollo y administración de sistemas para trabajar con diferentes proveedores o incluso con una combinación de ellos.

Otro aspecto destacado de Terraform es su capacidad para **realizar despliegues de infraestructura de manera incremental y controlada**. Con su sistema de planificación y ejecución, Terraform muestra una vista previa de los cambios que se realizarán antes de aplicarlos, lo que permite a los usuarios revisar y validar la infraestructura propuesta antes de llevarla a cabo. Esto es especialmente útil para evitar **cambios no deseados** o errores costosos.

Además de su funcionalidad principal, Terraform también cuenta con una amplia comunidad de usuarios y una gran cantidad de módulos y proveedores de terceros. Estos módulos y proveedores permiten extender las capacidades de Terraform para integrarse con diferentes servicios y componentes de infraestructura, lo que brinda aún más flexibilidad y opciones a los usuarios.

4.2 Ansible

Ansible, al igual que Terraform, es una herramienta de automatización de TI de código abierto, pero que se utiliza para simplificar y agilizar la **gestión de la configuración**, implementación de aplicaciones y orquestación de tareas en entornos de infraestructura. Diseñado para ser simple, potente y escalable, Ansible permite a los administradores de sistemas y desarrolladores automatizar de manera eficiente tareas repetitivas, lo que mejora la productividad y reduce los errores humanos.

En el núcleo de Ansible se encuentra un lenguaje de automatización sencillo y legible llamado **YAML** (Yet Another Markup Language). Con archivos YAML, los usuarios pueden describir y definir la configuración deseada de los sistemas, servicios y aplicaciones en un formato estructurado y fácil de entender. Esto permite a los equipos de TI definir su infraestructura y políticas operativas como código, lo que facilita la colaboración y el seguimiento de cambios.

Una de las características clave de Ansible es su enfoque "**sin agente**". Esto significa que no requiere la instalación de software adicional en los nodos objetivo. En cambio, se basa en SSH (Secure Shell) y utiliza conexiones seguras para comunicarse y ejecutar tareas en los sistemas remotos. Este diseño que presenta Ansible nos simplificará la configuración y el mantenimiento de nuestros sistemas, ya que no es necesario instalar y mantener agentes en cada nodo.

Ansible se basa en el concepto de "**playbooks**", que son archivos de configuración que describen el estado deseado del sistema y las tareas que deben ejecutarse para lograrlo. Los playbooks están escritos en YAML y se pueden utilizar para configurar servidores, desplegar aplicaciones, realizar actualizaciones de software y mucho más. Esto permite a los administradores de sistemas automatizar de forma declarativa una amplia gama de tareas, lo que ahorrará tiempo y evitará la posibilidad de introducir errores dentro de nuestros sistemas.

Además de la gestión de la configuración, Ansible también es conocido por su capacidad de **orquestación**. Con Ansible, los usuarios pueden coordinar y ejecutar tareas secuenciales o paralelas en diferentes nodos o grupos de nodos. Esto es especialmente útil en entornos de infraestructura complejos, donde se requiere la coordinación de múltiples sistemas para realizar acciones específicas.

Otra característica destacada de Ansible es su amplia comunidad y ecosistema. Ansible cuenta con una gran cantidad de roles y módulos desarrollados por la comunidad, que permiten a los usuarios extender y personalizar la funcionalidad de Ansible para adaptarse a sus necesidades específicas. Esto hace que Ansible sea altamente flexible y adaptable a diferentes entornos y casos de uso.

5. Instalación de ansible en el equipo de desarrollo.

Tal y como se definió en el diagrama de arquitectura, para poder trabajar con tanto con Ansible como con Terraform, y realizar el código necesario para cualquier tipo de configuración o modificación, necesitaremos una máquina para poder trabajar desde ella. A esta máquina que puede ser cualquier tipo de ordenador de sobremesa, y que lógicamente no es imprescindible, la he denominado equipo de desarrollo.

En nuestro caso concreto será un PC de 16 Gigas de memoria, corriendo el sistema operativo Windows 11, y del cual vamos a aprovechar el **Windows Subsystem for Linux 2** o WSL2 que nos permite ejecutar un entorno **Ubuntu Linux** directamente, sin necesidad de una máquina virtual independiente.

Por tanto, todas las referencias que se realizan a partir de ahora en este proyecto referentes a la ejecución de comandos GNU/Linux dentro del equipo de desarrollo se realizarán en dicho equipo en la terminal Linux que nos ofrece.

Además, el desarrollo del código podrá realizarse bien usando el editor vi de Ubuntu si estamos dentro de WSL2 o bien usando Visual Studio Code para Windows en el caso de estar operando con el escritorio de Windows.

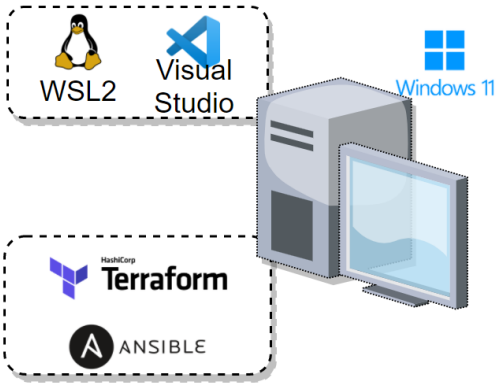
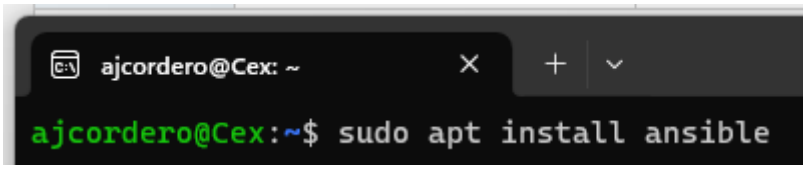


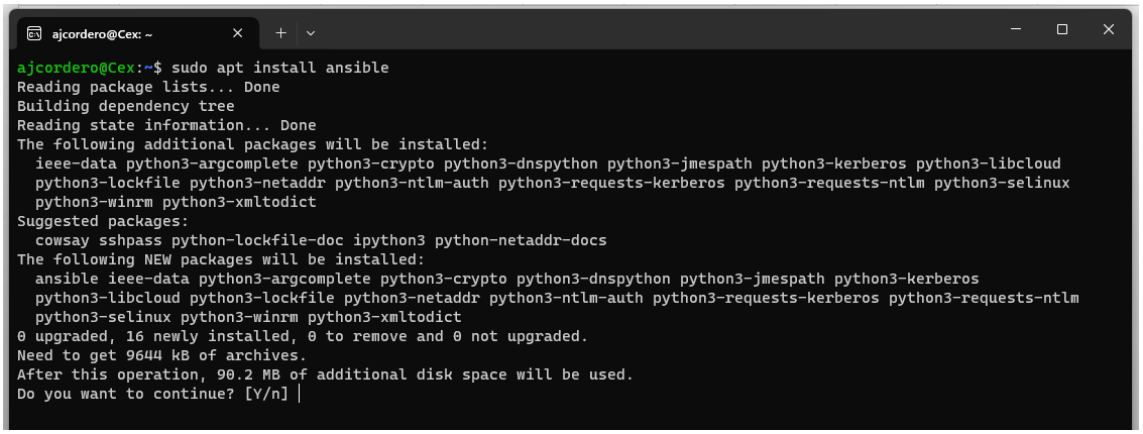
Ilustración 13 Entorno de desarrollo

Abriremos por tanto la sesión de Ubuntu en Windows 11:

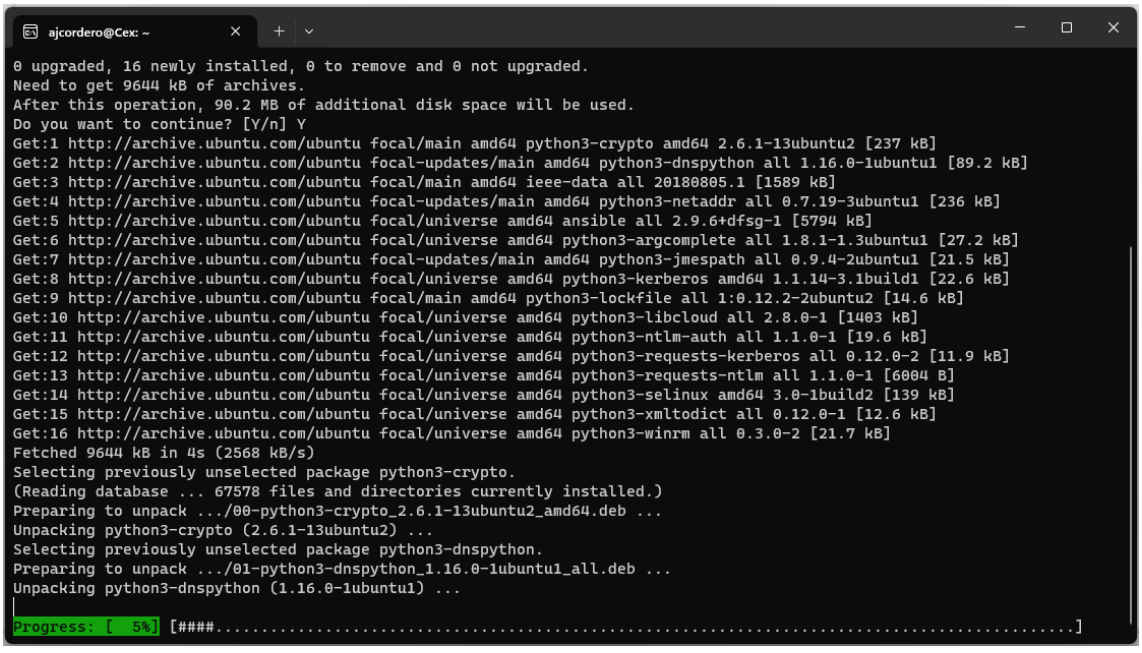
A continuación, procederemos a instalar ansible en el equipo de desarrollo. Para ello se ejecuta el comando:



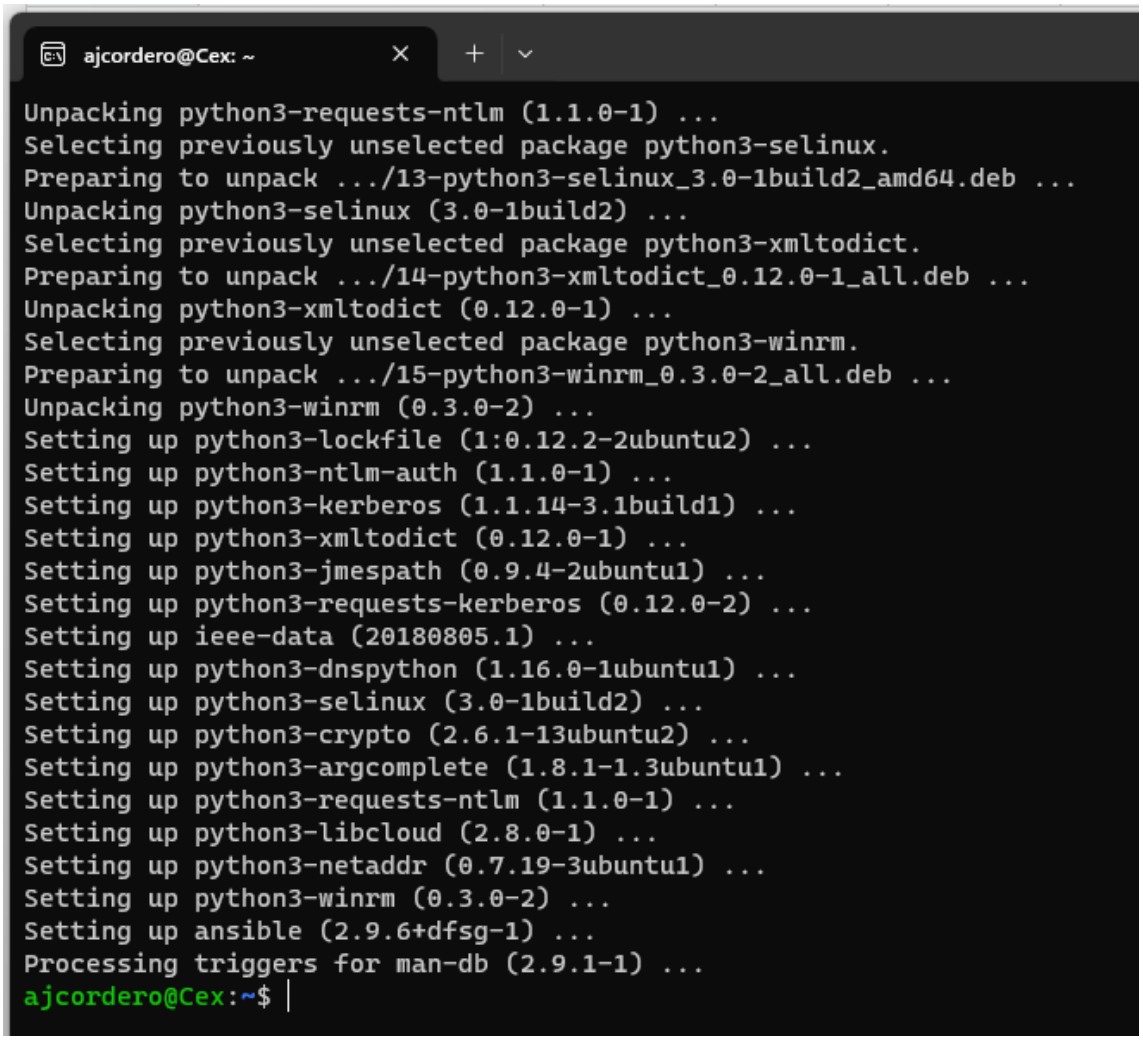
Se nos pedirá confirmación para la instalación de los paquetes necesarios:



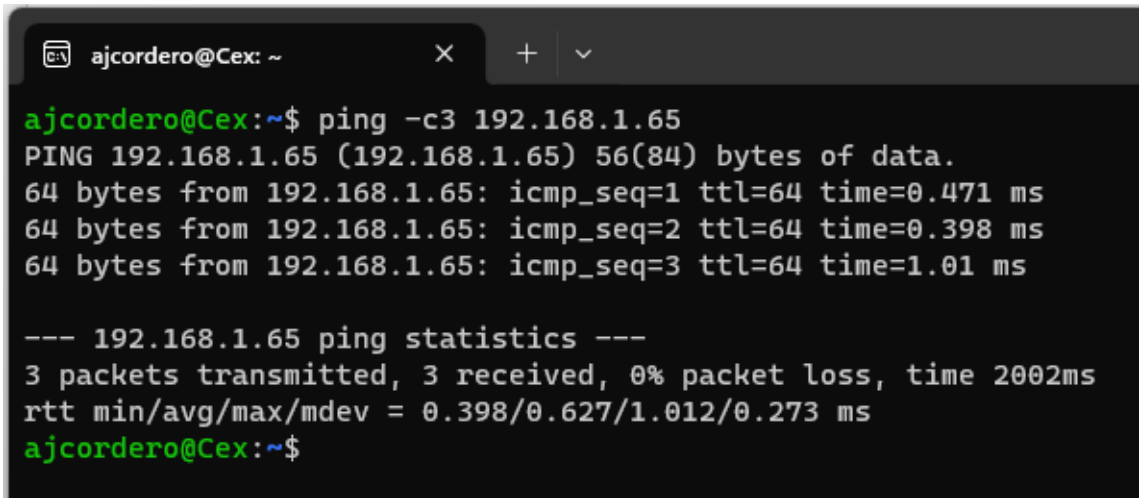
Confirmamos la instalación pulsando Y:



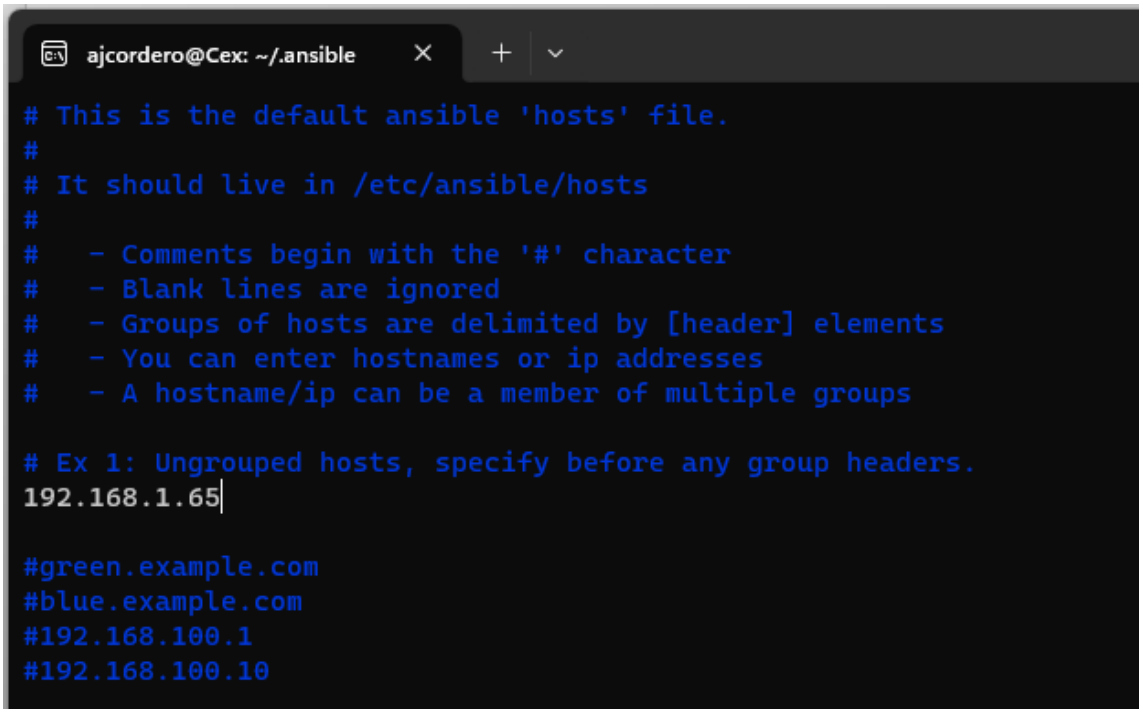
Esperamos al final del proceso de instalación de ansible en nuestro servidor:



Una vez finalizado el proceso de instalación de ansible en nuestro servidor, realizamos la prueba de conexión con el servidor Proxmox ejecutando un ping contra el servidor de proxmox:



Tras finalizar la instalación , en /etc/ansible , tendremos el fichero de configuración de ansible que es ansible.cfg. Editaremos dicho fichero y añadiremos la dirección ip de nuestro servidor de proxmox en él:



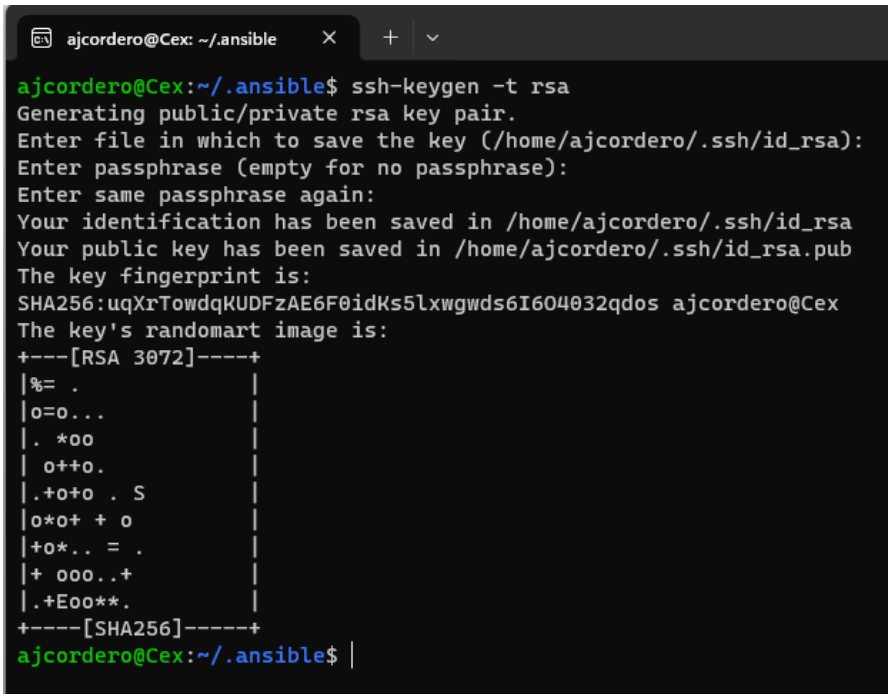
Comprobamos que ansible puede trabajar con el servidor de proxmox ejecutando el comando:

#ansible -m ping all

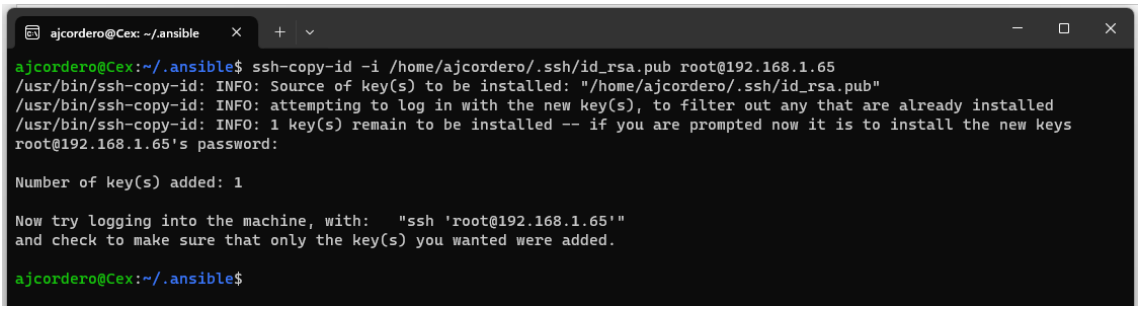
Podemos comprobar que, aunque no se ejecuta el ping, hemos llegado al servidor de Proxmox, pero al no tener configurado el acceso mediante ssh para nuestro usuario , el servidor de destino rechaza la conexión.

Para realizar dicha configuración , realizamos lo siguiente:

1. Generamos un par de claves SSH en el host local con el comando ssh -keygen.

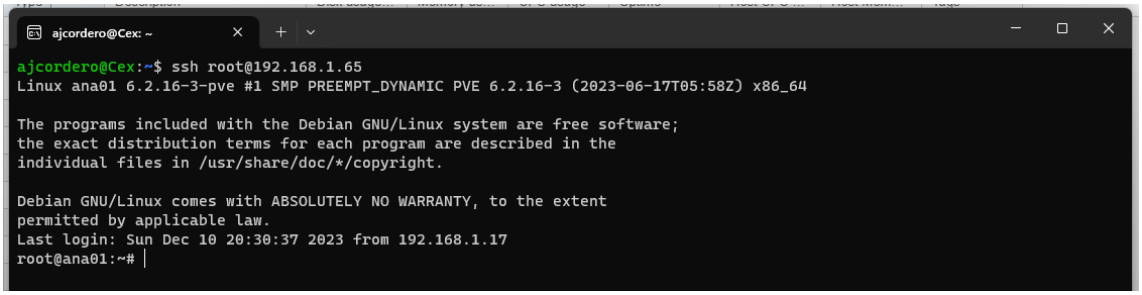


2. Posteriormente, copiamos la clave pública que se ha generado en el directorio /home/ajcordero/.ssh de nuestro host de “control” al servidor Promox:

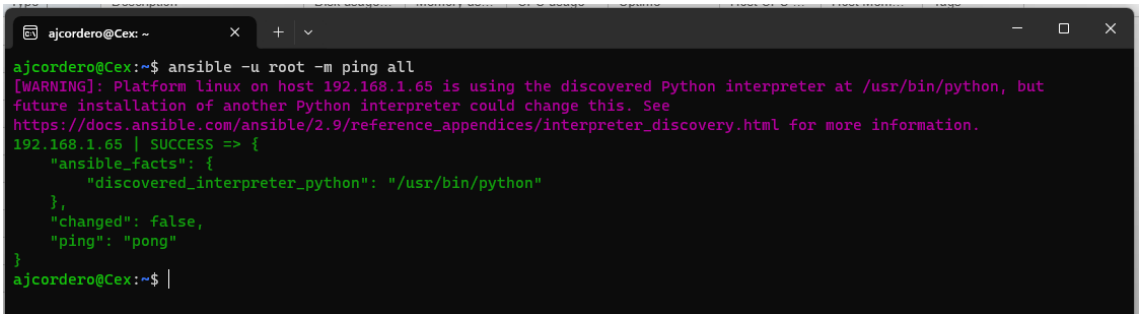


3. Verificamos que está correctamente añadida en el archivo autorized_keys del host proxmox. Este lo podemos hacer verificando el contenido del fichero /root/.ssh/authorized_keys.

4. Iniciamos sesión en el servidor remoto usando ssh y verificamos que todo es correcto:



5. Ya podemos cerrar la sesión ssh que tenemos abierta en el servidor proxmox, volver a nuestro servidor de control y ejecutar nuevamente ansible para ver el funcionamiento:



6. Configuración básica de Ansible.

En el equipo de control o desarrollo, editamos el fichero /etc/ansible/ansible.cfg y añadimos un nuevo grupo de hosts. En este caso, crearemos un grupo de hosts que corren proxmox , y lo haremos con el siguiente código:

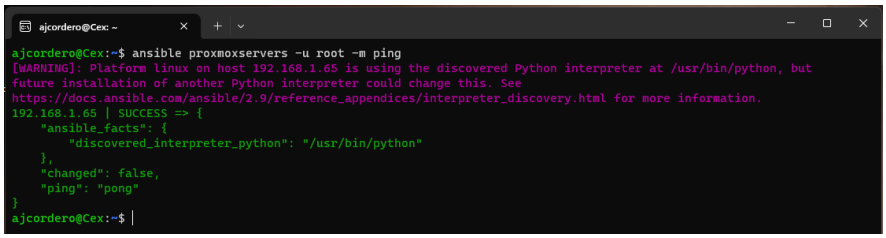
```
[proxmoxservers]
192.168.1.65
```

Con ello estamos creando un nuevo grupo de hosts, compuesto únicamente por nuestro servidor de Proxmox, y por lo tanto , podremos referenciarlos directamente desde el intérprete de comandos. Si anteriormente, para poder atacar al servidor lo hacíamos desde su dirección IPv4 con el comando:

```
#ansible 192.168.1.65 -u root -m ping
```

Ahora podremos hacerlo ejecutando:

```
#ansible proxmoxservers -u root -m ping
```



Volvemos a editar el fichero hosts, y añadimos el usuario con el que trabajaremos sobre nuestro host, así como su dirección IPv4 , y a partir de ahora podremos referirnos a él directamente por la denominación que le damos. En este caso proxmox.

Además, indicamos que el usuario con el que atacaremos al servidor será el usuario root. Todo esto queda referenciando en la siguiente línea:

```
Proxmox ansible_host=192.168.1.65 ansible_user=root
```

Y verificamos su funcionamiento con:

```
ajcordero@Cex:~$ ansible proxmox -m ping
[WARNING]: Platform linux on host proxmox is using the discovered Python interpreter at /usr/bin/python, but future
installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information.
proxmox | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
ajcordero@Cex:~$
```

Hemos visto, por tanto, que el fichero /etc/ansible/hosts o inventario por defecto una vez que hemos instalado ansible en nuestra máquina de desarrollo, es el fichero que nos permite para definir los hosts o nodos en los que se desea administrar la configuración utilizando Ansible. En dicho fichero se especifican los nombres y direcciones IP de dichos hosts, y nos ofrece la posibilidad de incluir algunas configuraciones específicas que se pueden pasar en cada línea de dicho fichero.

Además, nos permitirá organizar los hosts de destino de una manera **lógica**, tal y como vemos en el fichero de inventario creado para trabajar con la arquitectura propuesta en este proyecto:

```
[proxmoxservers]
ana01 ansible_host=192.168.1.65 ansible_user=root
inspiration04 ansible_host=192.168.1.67 ansible_user=root

[frontend]
test-200 ansible_host=10.10.10.200 ansible_user=acorderoma

[backend]
test-201 ansible_host=10.10.10.201 ansible_user=acorderoma
test-202 ansible_host=10.10.10.202 ansible_user=acorderoma

[mysqlservers]
test-203 ansible_host=10.10.10.203 ansible_user=acorderoma
test-205 ansible_host=10.10.10.205 ansible_user=acorderoma

[mailservers]
test-204 ansible_host=10.10.10.204 ansible_user=acorderoma
```

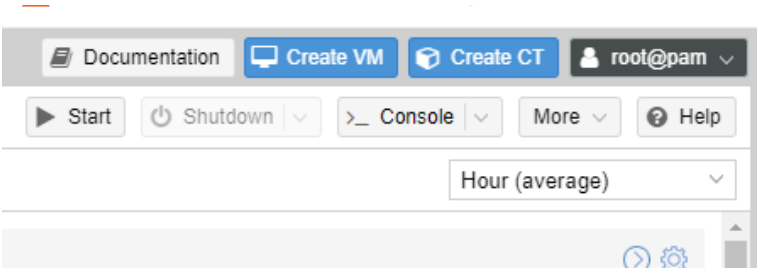
Ilustración 14 Organización Lógica de Ansible para la arquitectura propuesta.

7. Creación tradicional de una MV en Proxmox.

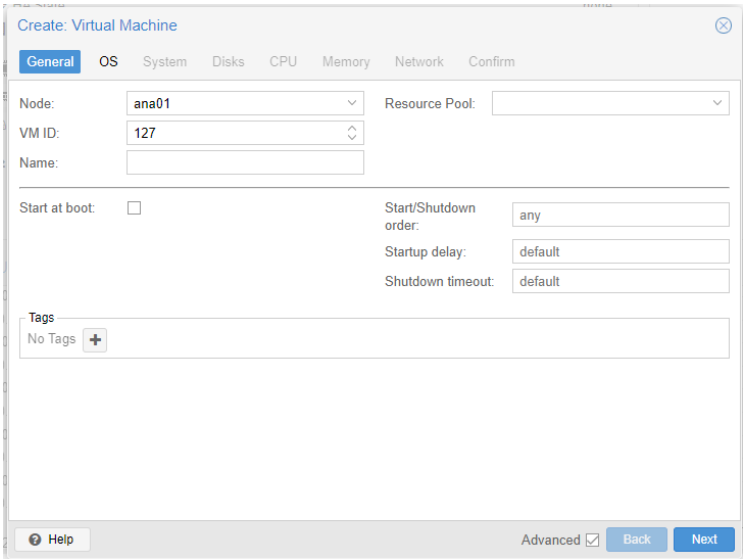
Una vez que hemos configurado ansible en nuestro equipo de desarrollo, y antes de comenzar a trabajar con terraform para la creación de la infraestructura virtual que será configurada utilizando playbooks de ansible, se ha preferido realizar en los siguientes puntos de esta memoria (puntos 6, 7, 8 y 9) la creación y configuración de una máquina virtual en Proxmox aprovechando el interfaz web del mismo, para así poder **comparar** las diferencias entre la implementación tradicional de nodos virtuales y la que nos permiten los generadores de infraestructura como código (Terraform y Ansible).

El proceso de creación de una máquina virtual en Proxmox desde la interfaz web que nos proporciona, se realiza de la siguiente manera:

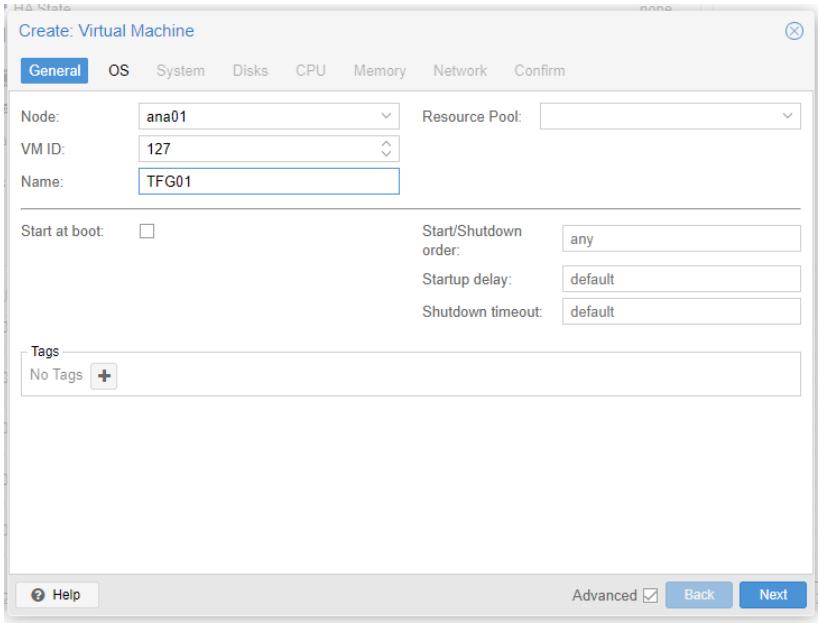
- Accedemos a la función de menú de crear una nueva máquina virtual.



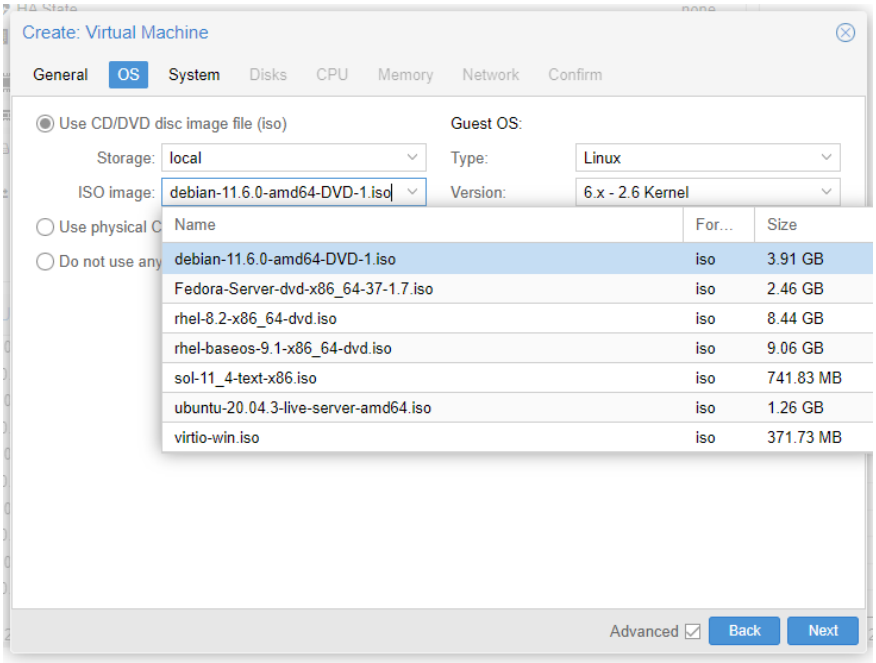
- Se nos presenta por parte de proxmox las opciones de configuración disponibles para la máquina virtual que vamos a crear.



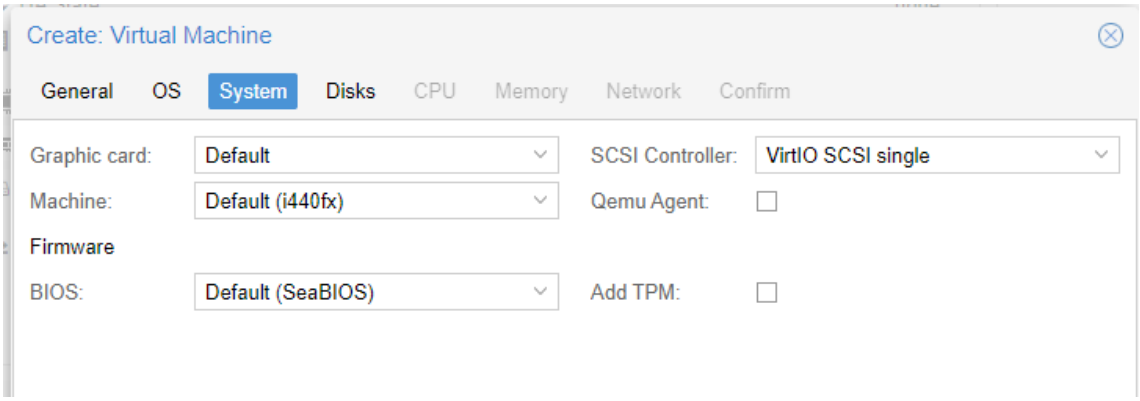
- En nuestro caso, la máquina que vamos a crear de manera general la denominaremos (propiedad Name) TFG01.



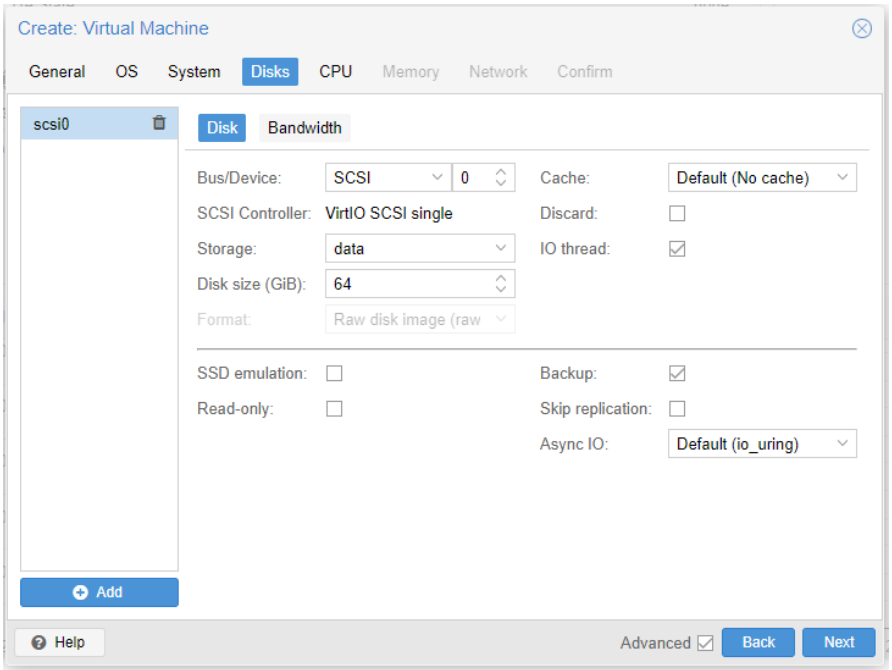
- A continuación, configuraremos el sistema operativo que deseamos correr en el host. Dicho sistema operativo, dependerá de las imágenes de distintos sistemas que tengamos almacenados en nuestro servidor de Proxmox. Para la realización del Trabajo de Fin de Grado, nos decantaremos por seleccionar la distribución debian-11.6.0 de entre las distintas que podríamos haber tomado.



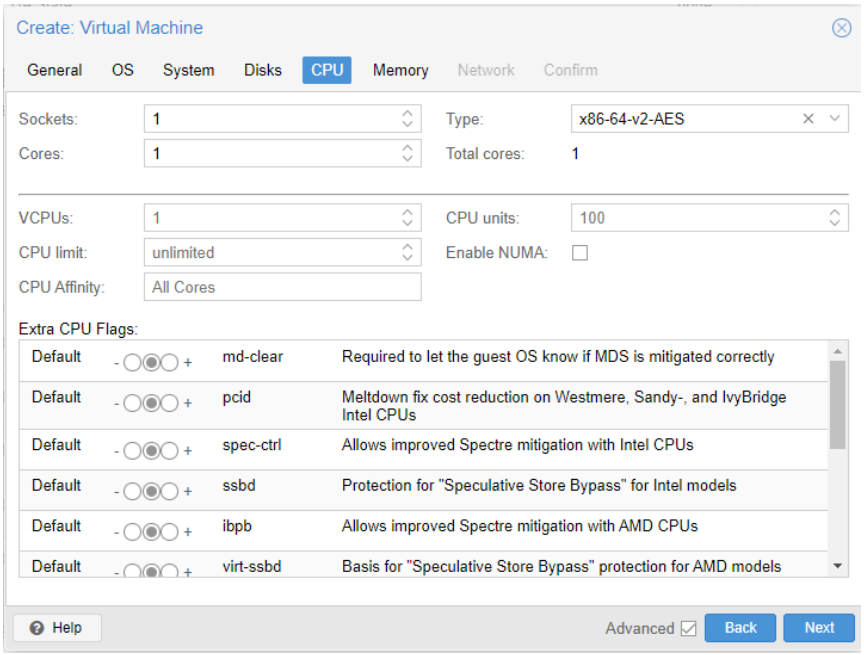
- Al pulsar el botón de Next, pasaremos a seleccionar distintos parámetros del sistema que estamos creando. Concretamente la tarjeta gráfica, el tipo de máquina, el tipo de BIOS y el modelo de controladora de discos deseada. Para la realización del Trabajo de Fin de Grado , dejaremos los valores por defecto.



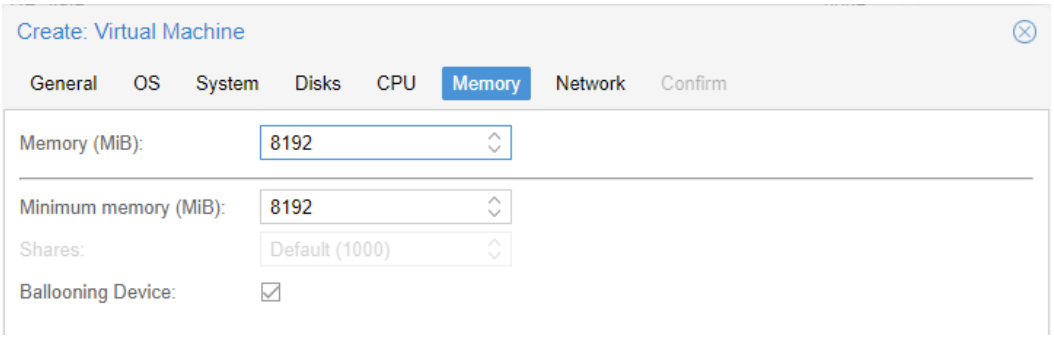
- Se entra a continuación en la selección del almacenamiento para nuestra máquina virtual. A efectos del Trabajo Final de Grado, elegiremos añadir un único disco de 50GB para la máquina virtual. Por tanto, sólo realizaremos un cambio de tamaño de este, ya que por defecto se ofrece por parte de Proxmox un disco de 32 GB.



- En el paso del tipo de CPU de la máquina virtual , se dejan los valores por defecto , ya que no será necesario el incremento de estos para poder desarrollar el trabajo. Una única CPU por máquina nos valdrá a efectos demostrativos.

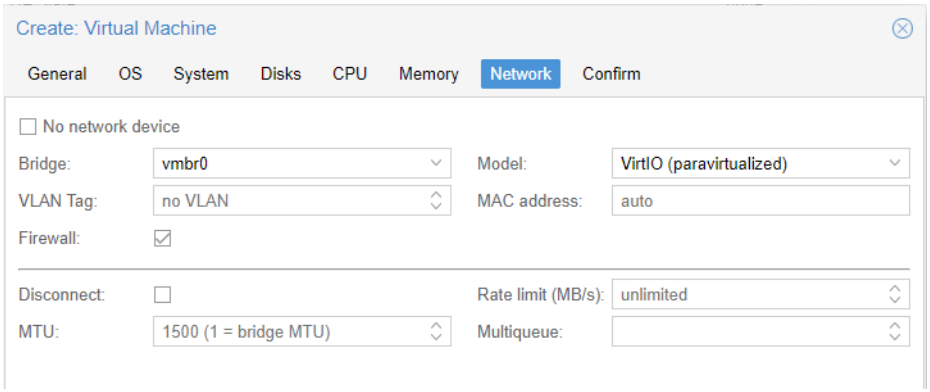


- En cuanto a memoria, se procederá a incrementar el tamaño de la misma. Concretamente, trabajaremos con máquinas de 8GB de memoria, que es un tamaño suficiente para poder desarrollar las distintas tareas que se desean. En el caso de necesitar más rendimiento , se procederá al cambio de dicho tamaño. Este ocurrirá sólo para tareas muy específicas.

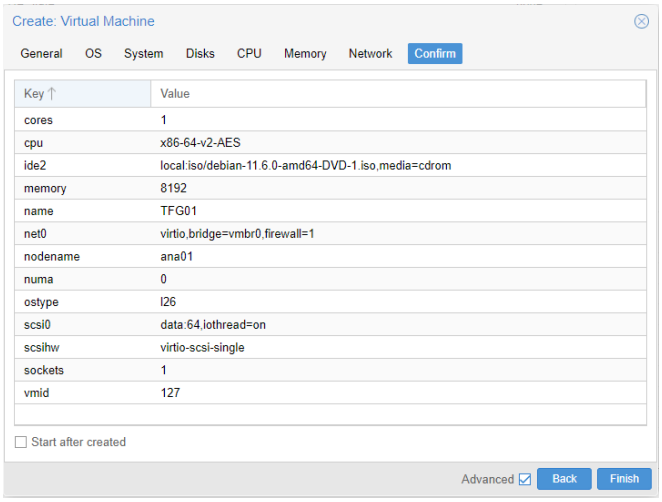


- Y por último, a nivel de red, se añadirá una única interfaz. Ciertamente, el diseño podría haber sido más extenso a nivel de comunicación, diseñando

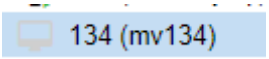
una red más estructurada y trabajando con distintas interfaces según su cometido (backup, gestión, producción, etc.) pero dicha segregación de redes se escapa del cometido del Trabajo de Fin de Grado. Por tanto, se admiten los valores por defecto que nos ofrece Proxmox.



- Sólo nos queda la creación de la máquina virtual y estará disponible para poder trabajar con ella.



- Nuestra máquina virtual, preparada para ser arrancada.



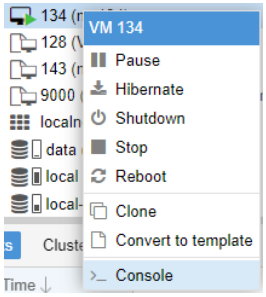
8. Instalación del Sistema Operativo en la máquina virtual.

Una vez creada la máquina virtual , el siguiente paso es la instalación del sistema operativo en la misma.

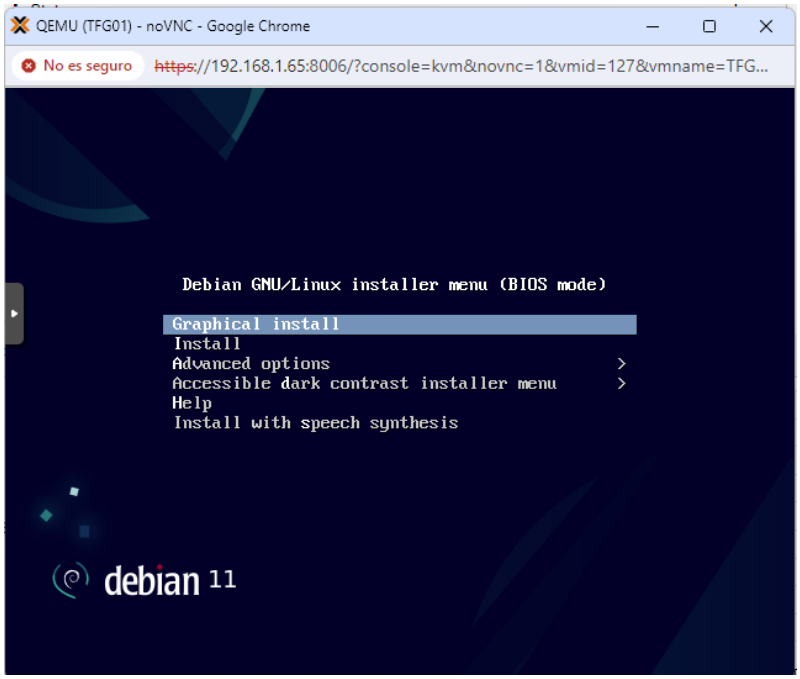
Para ello, deberemos encender la máquina, la cual arrancará el proceso de instalación de la imagen que quedó asociada a la misma durante el proceso de creación:

Instalación del Sistema Operativo.

Una vez arrancada la máquina virtual, se procede al acceso por consola de la misma, a fin de configurar el sistema operativo seleccionado y realizar su instalación.



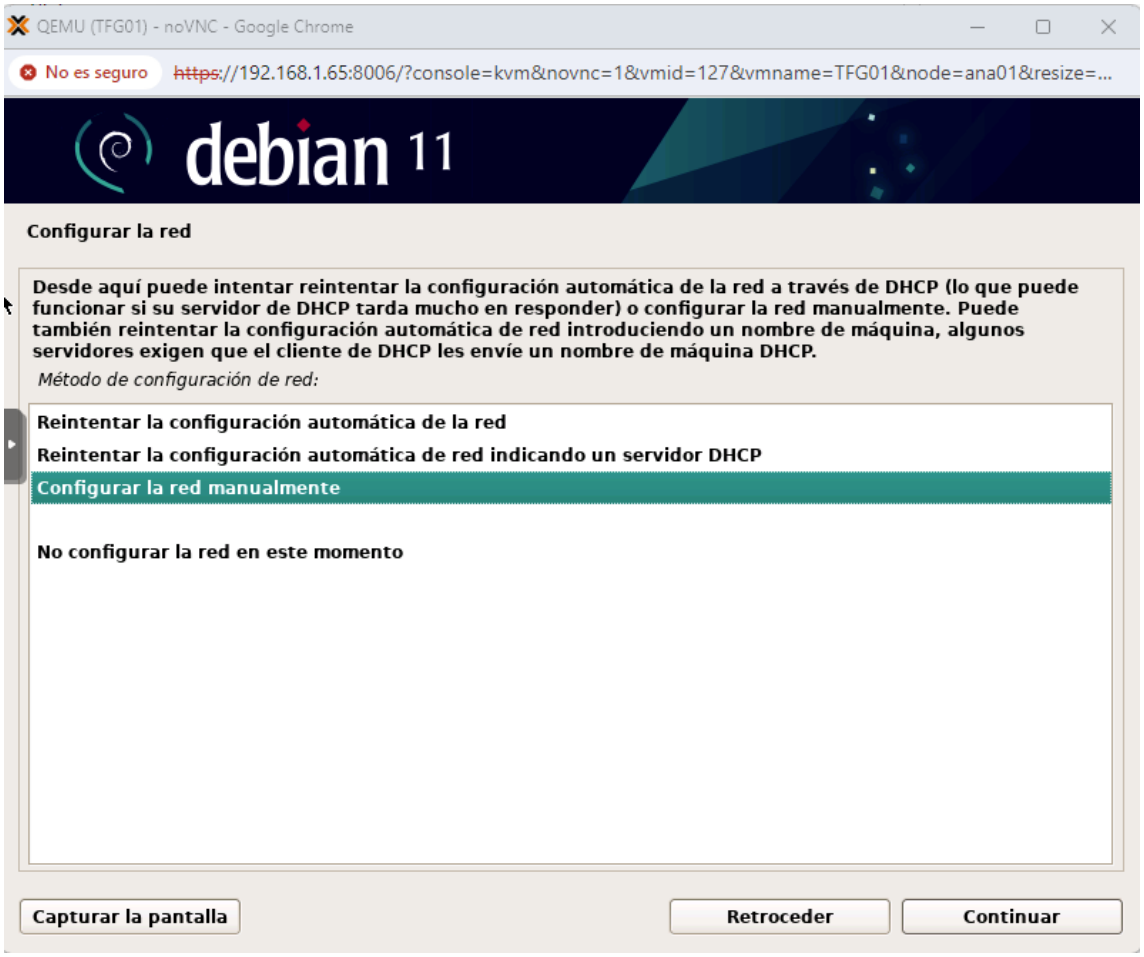
Como vemos, nos ofrece las típicas opciones de instalación de la distribución Debian.



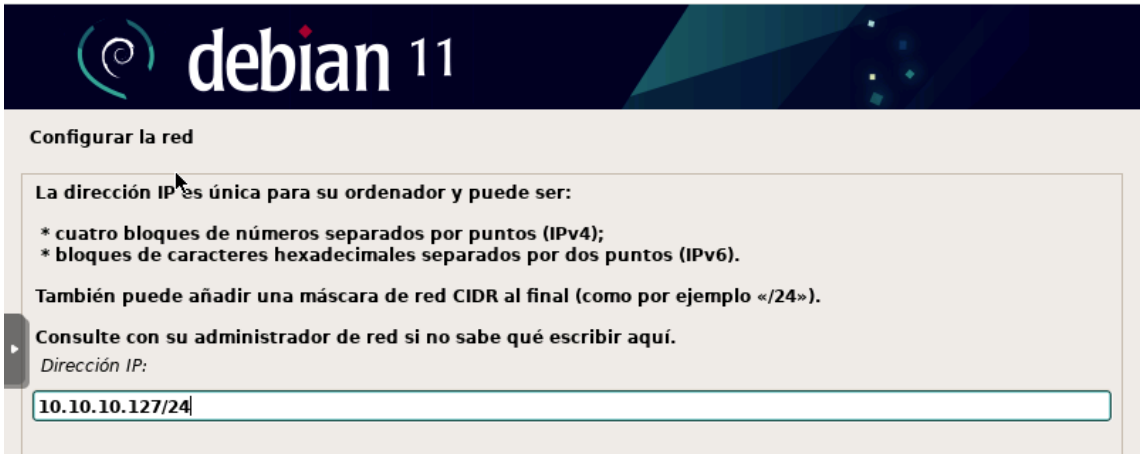
Seleccionamos la instalación gráfica, ya que nos será más sencillo así realizar una

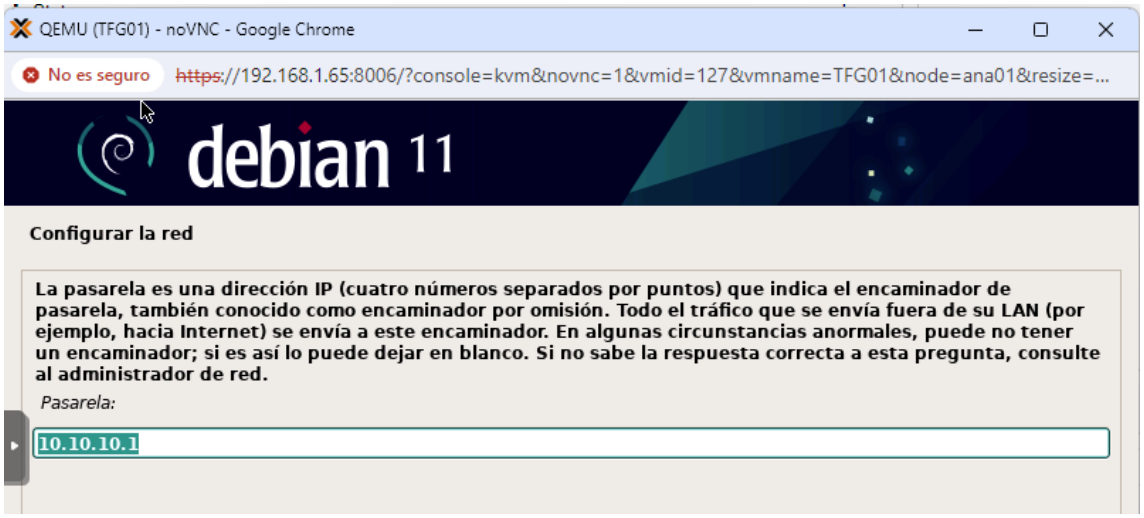
instalación básica del sistema. Seleccionaremos el idioma, la ubicación del sistema y la configuración del teclado.

Debian pasará entonces a realizar una detección de los dispositivos instalados en la máquina virtual, tras lo cual nos ofrecerá realizar la configuración de los parámetros de red. En nuestro caso , se configurará la máquina virtual de manera manual, ya que no se desea realizar la configuración automática por DHCP:

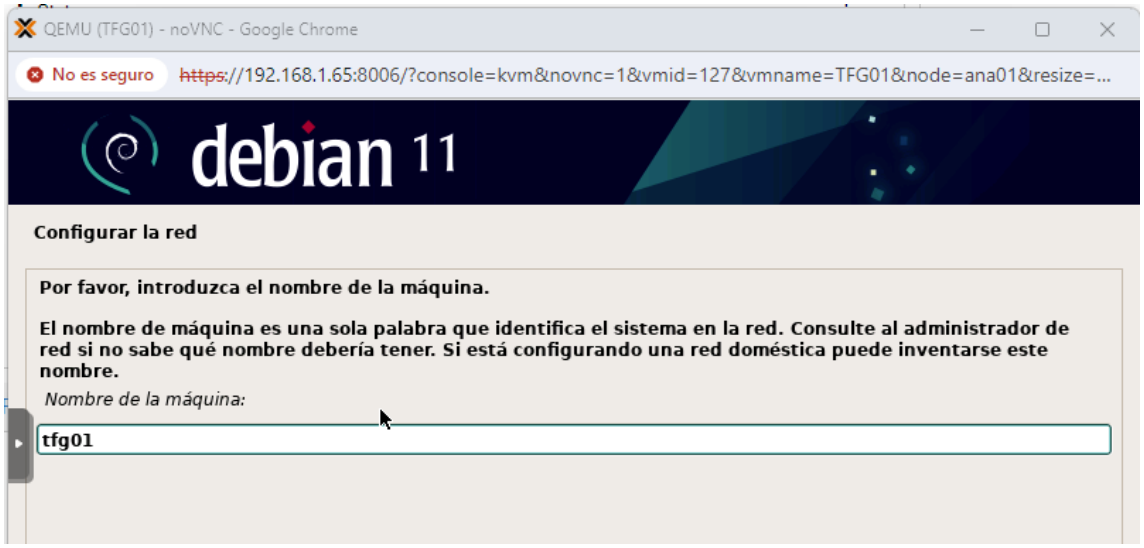


En nuestro sistema, la red con la que Proxmox trabajará internamente para las máquinas virtuales la hemos configurado como 10.10.10.0/24. Al tener que asignar por tanto a una máquina virtual una IP , la haremos coincidir con su Identificador de Máquina virtual. Por tanto , si nuestra máquina virtual es la que identifica el sistema como 127, asignaremos dicho valor a la IPv4 y su dirección será 10.10.10.127/24, siendo el Gateway la dirección del servidor de Proxmox a nivel de red interna , que es 10.10.10.1/24.



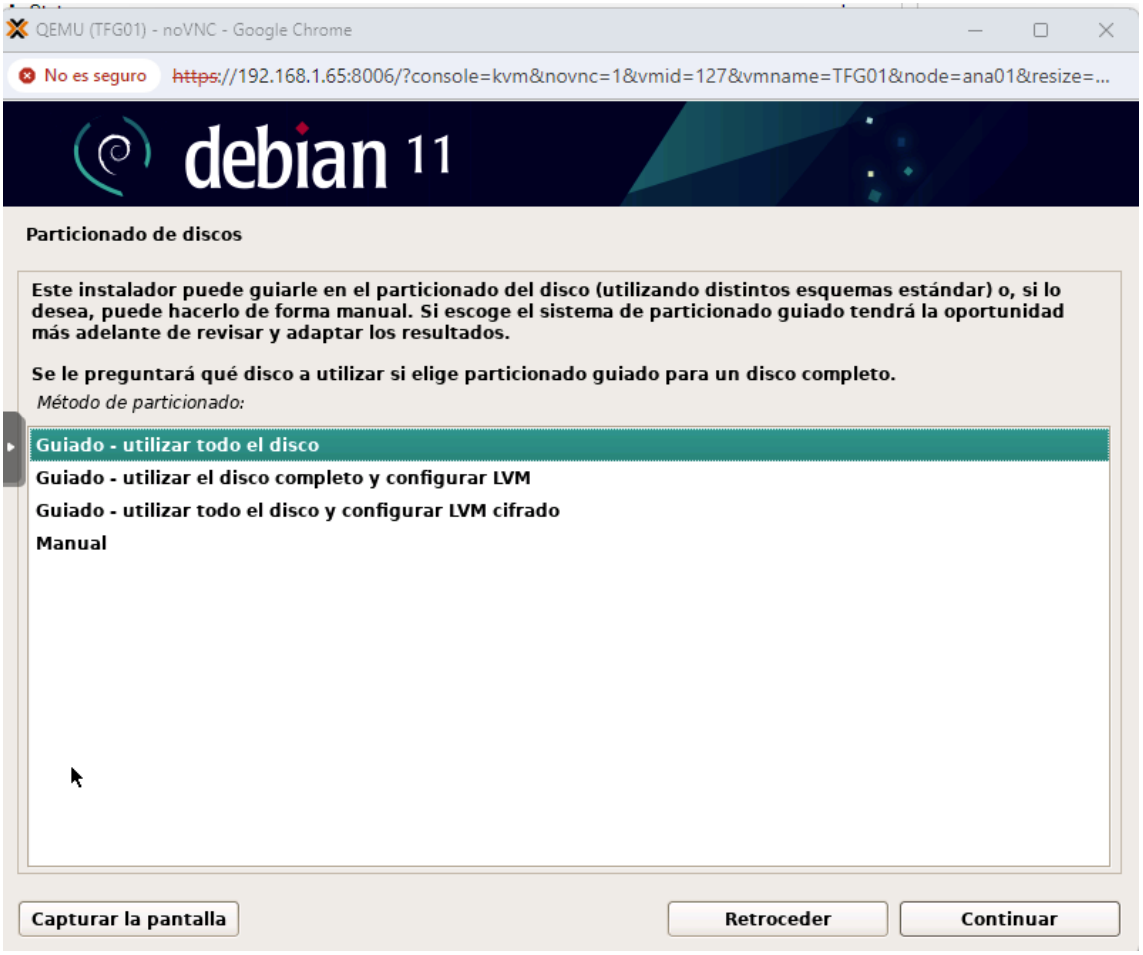


Como nombre del host, asignaremos tfg01, que es además el nombre que a nivel de Proxmox le hemos dicho que iba a tener. Además, indicaremos que el dominio de la máquina es cordero.eu

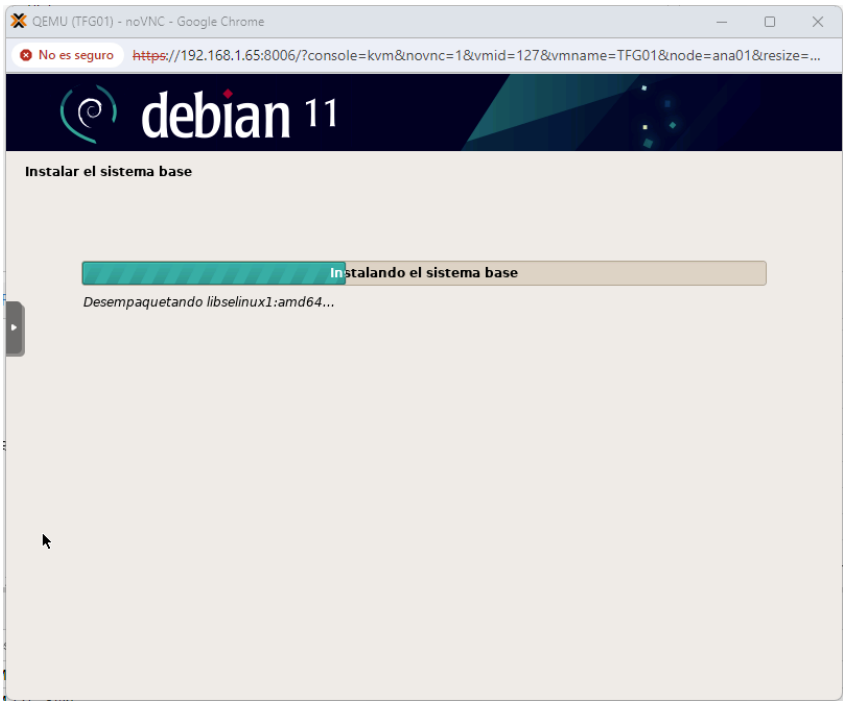


Finalmente, introduciremos la password de superusuario para nuestra máquina virtual, en nuestro caso será root, y, además , crearemos un usuario para poder trabajar con ella. Lógicamente, el usuario creado es el mismo que coincide con la cuenta de UOC, **acorderoma**.

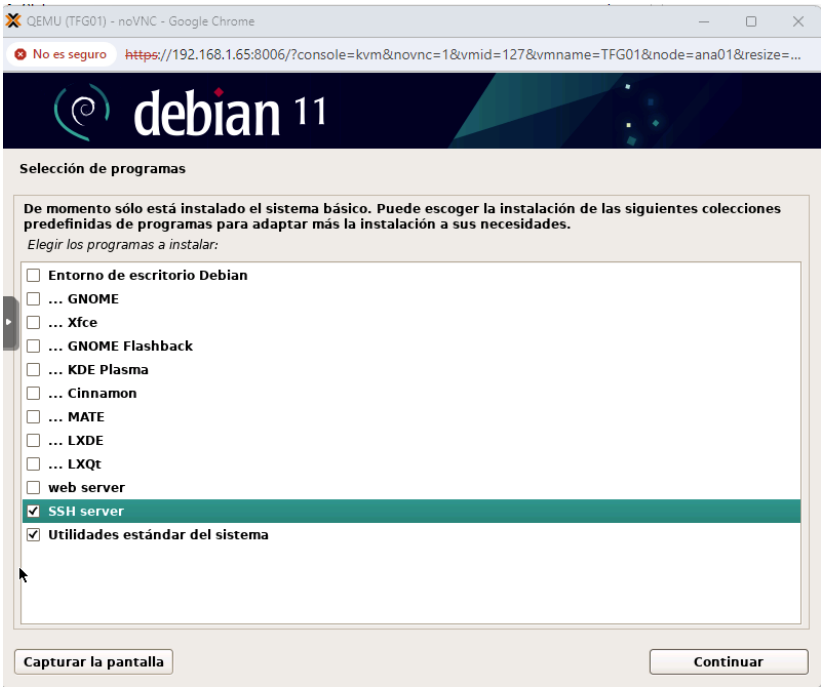
A nivel de almacenamiento , aunque se podría realizar su particionado de manera más eficiente, seleccionamos “utilizar todo el disco” ya que realizar configuraciones avanzadas de almacenamiento se aleja del objetivo de este TFG.



Por lo tanto, nuestra máquina virtual presentará un disco de 64 GB, en el cual comenzará a instalarse el sistema operativo.

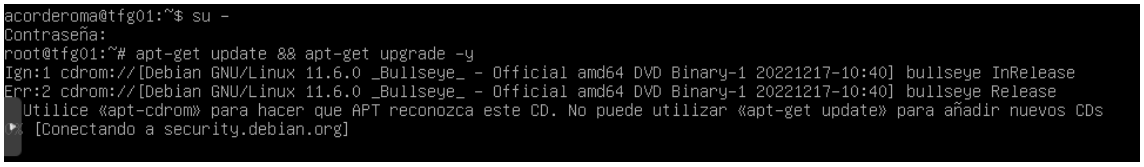


En la selección de programas solo seleccionaremos as utilidades estándar del sistema y el servidor SSH para poder trabajar con la máquina, ya que el resto no las necesitamos para poder trabajar.

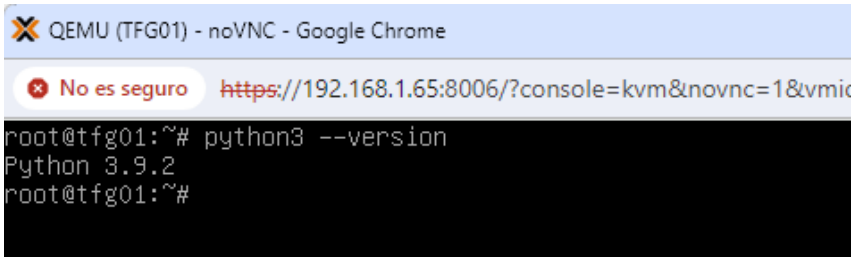


Con ello finalizamos la instalación. Hemos creado una máquina virtual con un sistema operativo, en nuestro caso Debian, totalmente operativa.

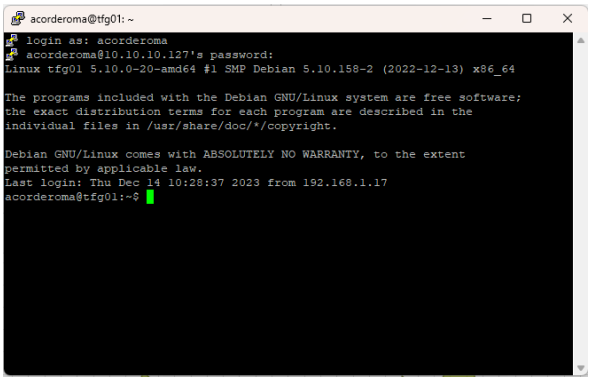
El siguiente paso será actualizar la paquetería del sistema operativo:



Para poder trabajar dicha máquina desde Ansible, necesitamos que tenga instalado OpenSSH, el cual ha quedado instalado en el proceso de configuración del sistema operativo y Python. Por tanto , tendremos que verificar que Python está corriendo en nuestra máquina virtual:



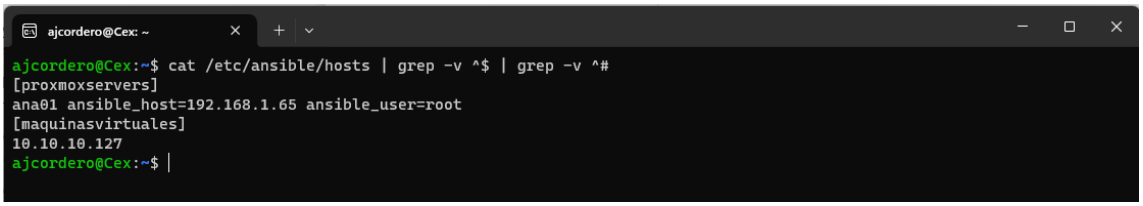
Para comprobar que el servidor ssh está funcionando correctamente, se procede a realizar el acceso a la máquina virtual mediante putty:



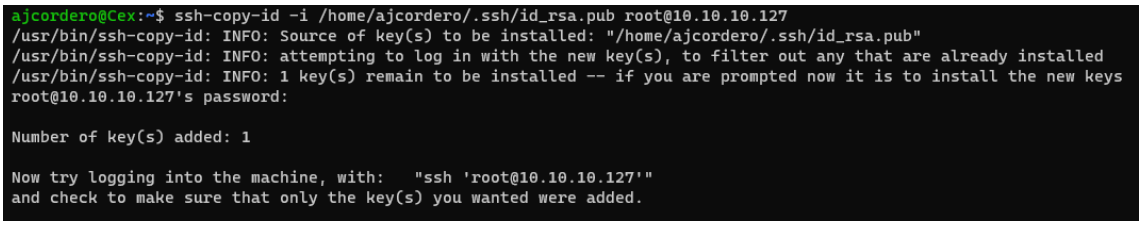
Como vemos, nuestra máquina está funcionando correctamente. Ahora podría ser configurada a nivel de programas utilizando **Ansible**.

9. Interactuando con Ansible.

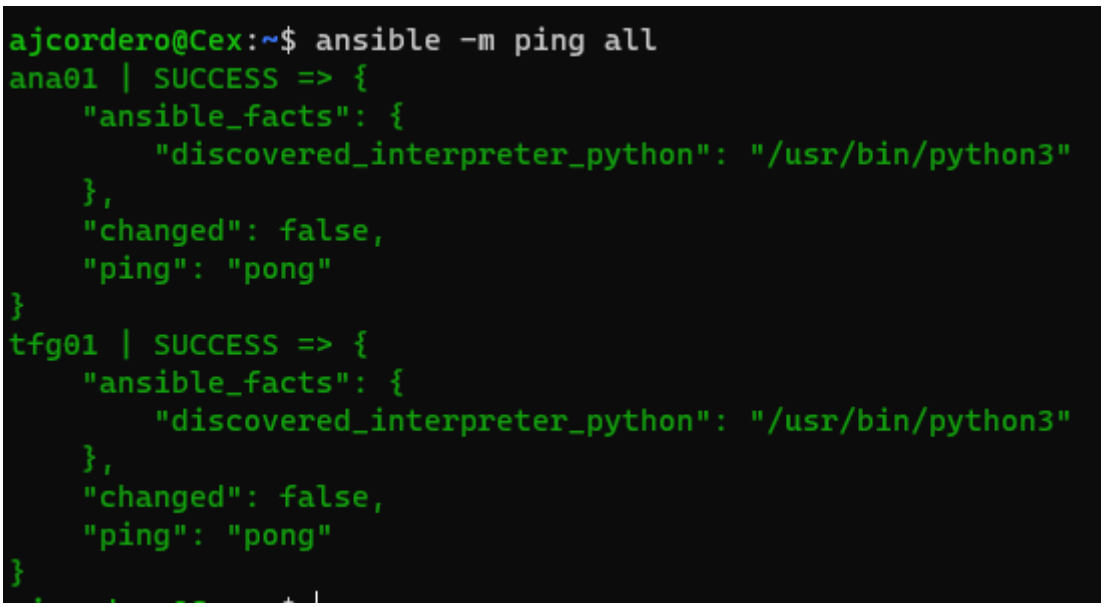
Añadimos nuestro nuevo host al inventario de la máquina:



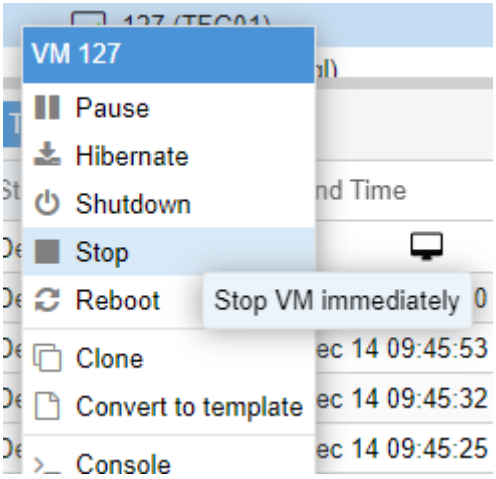
Copiamos la clave publica de nuestro usuario, así ansible podrá acceder a la máquina sin errores:



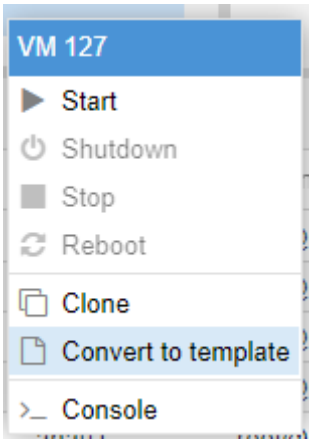
Ejecutamos ping sobre los servidores que tenemos en nuestro inventario para ver que no existen errores:



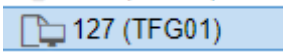
Una vez que hemos actualizado nuestra máquina virtual y vemos que podemos acceder sin problema a ella, vamos a convertirla en una imagen desde Proxmox a fin de poder trabajar con ella y generar nuevas máquinas virtuales a partir de dicho template. Para ello, lo primero que tendremos que hacer es apagarla.



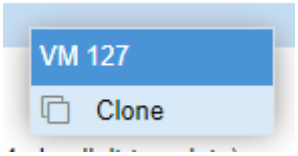
Una vez que la máquina está apagada, seleccionamos la opción de convertirla en template desde el menú de Proxmox:



Nos pedirá confirmación para dicha acción , le indicamos que sí , y comienza el proceso de conversión. Finalmente , nuestra máquina virtual 127 , se ha transformado en un template, con lo cual podremos clonarla directamente para arrancar nuevas máquinas:



Dichas máquinas , serán un clon de nuestra máquina de origen, y utilizaremos ansible para instalar los paquetes necesarios en ella dependiendo del **role** que le queramos dar a cada maquina generada.



Una vez que tenemos nuestra máquina clonada, deberemos comenzar cambiando su direccionamiento ip, ya que posee la dirección de la máquina virtual con la cual creamos el template.

Pare ello , hemos creado un playbook de ansible , que nos permitirá realizar dicho proceso:

```
---
- name: Cambiar la dirección IP de la máquina
  hosts: tfg01
  become: yes
  vars_files:
    - ./roles/vars/main.yml
    - ./roles/proxmox_deploy/defaults/main.yml

  tasks:
    - name: Cambiar la dirección ip del template por la verdadera ip
fija.
      lineinfile:
        dest: '/etc/network/interfaces'
        line: "address 10.10.10.129/24"
        state: present
        regexp: "address"
      notify:
      - Reinicio de la interfaz de red

  handlers:
    - name: Reinicio de la interfaz de red
      service:
        name: networking
        state: restarted
```

Además, se necesita cambiar el nombre del host y tanto en /etc/hosts, como en /etc/hostname. También generamos código Ansible para ello , ya que son tareas repetitivas que hay que realizar cada vez que clonemos una nueva máquina.

```
---
- name: Cambiar el valor del hostname de la máquina
  hosts: tfg01
  become: yes
  vars_files:
    - ./roles/vars/main.yml
    - ./roles/proxmox_deploy/defaults/main.yml

  tasks:
    - name: Actualizar el archivo /etc/hostname
      lineinfile:
        path: /etc/hostname
        regexp: '^tfg01'
        line: "correo"
      notify: Reiniciar el Servicio
```

```
    - name: Actualizar el /etc/hosts
      lineinfile:
        path: /etc/hosts
        regexp: '^10.10.10.127'
        line: '10.10.10.129 correo.cordero.eu correo'

  handlers:
    - name: Reiniciar el Servicio
      systemd:
        name: systemd-hostnamed
        state: restarted
```

Dependiendo del Role que le asignemos a la máquina , tendremos preparado playbooks de ansible que se encargarán de actualizar los paquetes de la distribución de manera adecuada, y realizar la instalación y configuración de los mismos.
Así , en el caso que el Role de la máquina sea “Servidor Web” tendremos un playbook que deje configurada a la máquina para tal propósito, si es un “Servidor de Correo” se tendrá otro playbook distinto, etc.

10. Roles de los distintos servicios.

Los distintos roles de servicio que se han generado hasta el momento son los siguientes:

- Servidor de Correo

```yaml
---
- name: Cambiar el valor del hostname de la máquina
  hosts: tfg01
  become: yes
  vars_files:
    - ./roles/vars/main.yml
    - ./roles/proxmox_deploy/defaults/main.yml

  tasks:
    - name: Actualizar cache de paquetes
      apt:
        update_cache: yes

    - name: Instalar Exim
      apt:
        name: exim4
        state: present

    - name: Configurar Exim
      template:
        src: exim.conf.j2
        dest: /etc/exim4/exim4.conf

    - name: Reiniciar Exim
      service:
        name: exim4
        state: restarted
```

- Servidor de Web

```yaml
---
- name: Instalar Apache en el Role Servidor de Web
  hosts: tfg01
  become: yes
  vars_files:
    - ./roles/vars/main.yml
    - ./roles/proxmox_deploy/defaults/main.yml

  tasks:
    - name: Actualizar cache de paquetes
      apt:
        update_cache: yes

```

```
      - name: Instalar Apache
        apt:
          name: apache2
          state: present

      - name: Levantar Apache
        service:
          name: apache2
          state: started
          enabled: yes
```

11. Creación eficiente de un template para Terraform.

Para poder realizar la creación de máquinas virtuales de una más manera más eficiente que la que hemos visto anteriormente y poder trabajar con **Terraform**, procedemos a crear un Template a partir de una imagen Debian disponible en la web. Concretamente, descargaré la última versión disponible de Debian 11.

Se accede a la misma realizando un wget desde nuestro servidor Proxmox:

```
root@ana01:~# wget https://cloud.debian.org/images/cloud/bullseye/latest/debian-11-generic-amd64.qcow2
--2024-01-05 11:17:30--  https://cloud.debian.org/images/cloud/bullseye/latest/debian-11-generic-amd64.qcow2
Resolving cloud.debian.org (cloud.debian.org)... 194.71.11.163, 194.71.11.173, 194.71.11.165, ...
Connecting to cloud.debian.org (cloud.debian.org)|194.71.11.163|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://laotzu.ftp.acc.umu.se/images/cloud/bullseye/latest/debian-11-generic-amd64.qcow2 [following]
--2024-01-05 11:17:31--  https://laotzu.ftp.acc.umu.se/images/cloud/bullseye/latest/debian-11-generic-amd64.qcow2
Resolving laotzu.ftp.acc.umu.se (laotzu.ftp.acc.umu.se)... 194.71.11.166, 2001:6b0:19::166
Connecting to laotzu.ftp.acc.umu.se (laotzu.ftp.acc.umu.se)|194.71.11.166|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 334096896 (319M)
Saving to: ‘debian-11-generic-amd64.qcow2’

debian-11-generic-amd64.qcow2 100%[=================================================>] 318.62M  4.11MB/s    in 73s

2024-01-05 11:18:45 (4.34 MB/s) - ‘debian-11-generic-amd64.qcow2’ saved [334096896/334096896]

root@ana01:~#
```

Verificamos la instalación de la librería libguestfs-tools. Dicha librería nos permitirá trabajar con las imágenes de las máquinas virtuales, permitiéndonos acceder y modificar el sistema de archivos, crear y clonar imágenes, etc.

```
root@ana01:~# apt-get update -y && apt install libguestfs-tools -y
Get:1 http://security.debian.org bookworm-security InRelease [48.0 kB]
Hit:2 http://ftp.es.debian.org/debian bookworm InRelease
Hit:3 http://download.proxmox.com/debian/pve bookworm InRelease
Hit:4 http://download.proxmox.com/debian/ceph-quincy bookworm InRelease
Get:5 http://ftp.es.debian.org/debian bookworm-updates InRelease [52.1 kB]
Fetched 100 kB in 1s (113 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libguestfs-tools is already the newest version (1:1.48.6-2).
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
root@ana01:~#
```

Se customiza la imagen:

```
root@ana01:~# virt-customize -a debian-11-generic-amd64.qcow2 --install qemu-guest-agent
[   0.0] Examining the guest ...
[   6.5] Setting a random seed
virt-customize: warning: random seed could not be set for this type of
guest
[   6.5] Setting the machine ID in /etc/machine-id
[   6.5] Installing packages: qemu-guest-agent
[  23.4] Finishing off
root@ana01:~#
```

Por último, pasamos a crear el template, que llamaremos 9001:

```
root@ana01:~# qm create 9001 --name "debian-11-cloudinit-template" --memory 4096 --cores 2 --net0 virtio,bridge=vmbr0
root@ana01:~#
```

Importamos el disco:

```
root@ana01:~# qm create 9001 --name "debian-11-cloudinit-template" --memory 4096 --cores 2 --net0 virtio,bridge=vmbr0
```

```
root@Cex: /home/ajcordero/
transferred 1.5 GiB of 2.0 GiB (76.30%)
transferred 1.5 GiB of 2.0 GiB (77.30%)
transferred 1.6 GiB of 2.0 GiB (78.30%)
transferred 1.6 GiB of 2.0 GiB (79.30%)
transferred 1.6 GiB of 2.0 GiB (80.30%)
transferred 1.6 GiB of 2.0 GiB (81.30%)
transferred 1.6 GiB of 2.0 GiB (82.30%)
transferred 1.7 GiB of 2.0 GiB (83.30%)
transferred 1.7 GiB of 2.0 GiB (84.30%)
transferred 1.7 GiB of 2.0 GiB (85.30%)
transferred 1.7 GiB of 2.0 GiB (86.30%)
transferred 1.7 GiB of 2.0 GiB (87.30%)
transferred 1.8 GiB of 2.0 GiB (88.30%)
transferred 1.8 GiB of 2.0 GiB (89.30%)
transferred 1.8 GiB of 2.0 GiB (90.30%)
transferred 1.8 GiB of 2.0 GiB (91.30%)
transferred 1.8 GiB of 2.0 GiB (92.30%)
transferred 1.9 GiB of 2.0 GiB (93.30%)
transferred 1.9 GiB of 2.0 GiB (94.30%)
transferred 1.9 GiB of 2.0 GiB (95.30%)
transferred 1.9 GiB of 2.0 GiB (96.30%)
transferred 1.9 GiB of 2.0 GiB (97.30%)
transferred 2.0 GiB of 2.0 GiB (98.30%)
transferred 2.0 GiB of 2.0 GiB (99.30%)
transferred 2.0 GiB of 2.0 GiB (100.00%)
transferred 2.0 GiB of 2.0 GiB (100.00%)

Successfully imported disk as 'unused0:data:vm-9001-disk-0'
root@ana01:~#
root@ana01:~#
```

Asignamos sus características generales de hardware:

```
root@ana01:~# qm set 9001 --scsihw virtio-scsi-pci --scsi0 data:vm-9001-disk-0
update VM 9001: -scsi0 data:vm-9001-disk-0 -scsihw virtio-scsi-pci
root@ana01:~#
```

Definimos el orden de arranque de la máquina:

```
root@ana01:~# qm set 9001 --boot c --bootdisk scsi0
update VM 9001: -boot c -bootdisk scsi0
root@ana01:~#
```

A su vez, para poder trabajar con cloud-init , le asignamos un disco ide.

```
root@Cex: /home/ajcordero/
root@ana01:~# qm set 9001 --ide2 data:cloudinit
update VM 9001: -ide2 data:cloudinit
  Logical volume "vm-9001-cloudinit" created.
ide2: successfully created disk 'data:vm-9001-cloudinit,media=cdrom'
generating cloud-init ISO
root@ana01:~#
```

Se añade un puerto serial para generar la consola:

```
root@ana01:~# qm set 9001 --serial0 socket --vga serial0
update VM 9001: -serial0 socket -vga serial0
root@ana01:~#
```

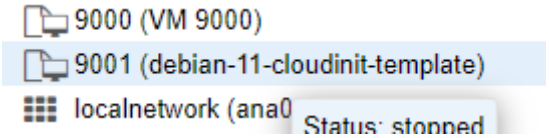
Indicamos que el agente esté levantado para la comunicación con la MV:

```
root@ana01:~# qm set 9001 --agent enabled=1
update VM 9001: -agent enabled=1
root@ana01:~#
```

Finalmente creamos el template:

```
root@ana01:~# qm template 9001
  Renamed "vm-9001-disk-0" to "base-9001-disk-0" in volume group "data"
  Logical volume data/base-9001-disk-0 changed.
  WARNING: Combining activation change with other commands is not advised.
root@ana01:~#
```

Volviendo al interfaz de Proxmox, podremos verificar la existencia del nuevo template que acabamos de generar, el 9001:

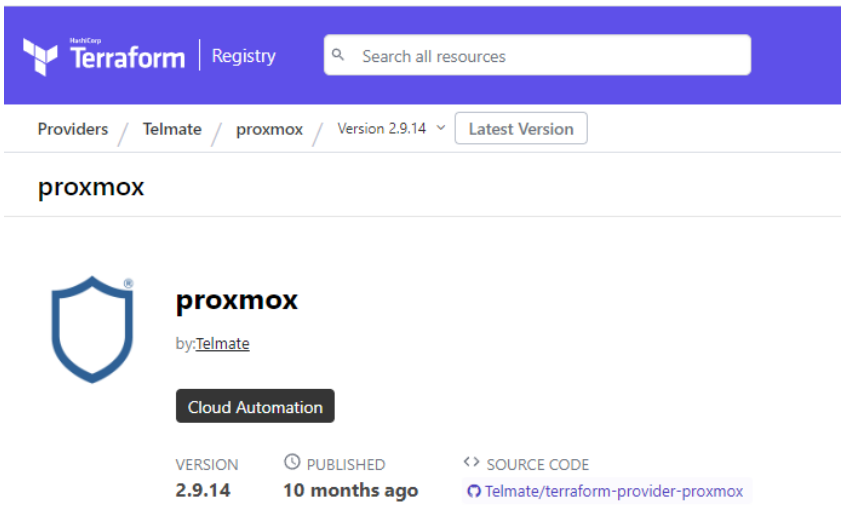


12. Usando Terraform con Proxmox.

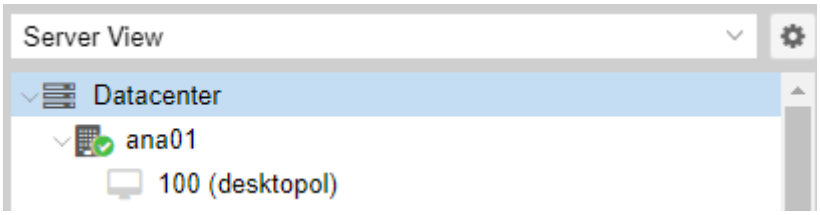
Para poder actuar con Terraform en nuestro servidor Proxmox necesitaremos un **provider**. En Terraform se denomina así a un componente de software que permite interactuar con una plataforma o servicio específico, ya sea AWS, Google Cloud, o en nuestro caso Proxmox. Cada provider de Terraform proporciona los recursos y datos necesarios para crear, modificar y eliminar los recursos en la plataforma o servicio elegido para trabajar.

Cuando indicamos un proveedor concreto en un archivo de configuración de Terraform, estamos realizando una conexión con la infraestructura que poseemos, indicando que recursos serán administrados.

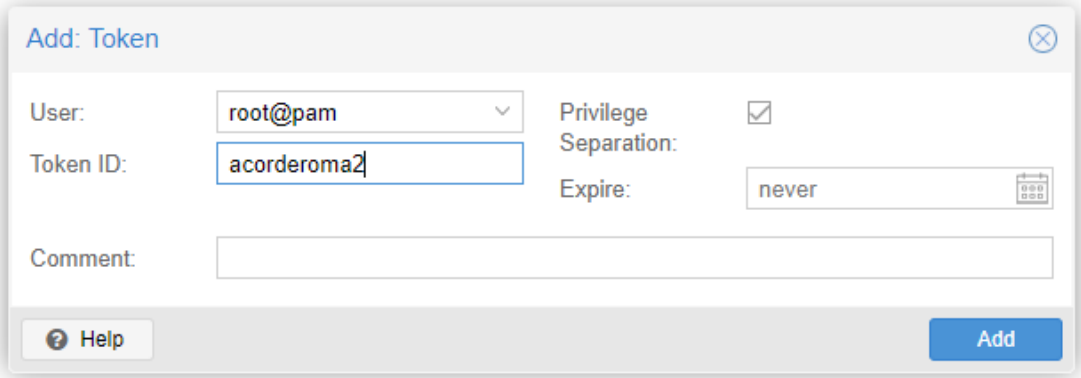
En nuestro caso, como Terraform provider usaremos **telmate** , que nos permitirá conectar Terraform con Proxmox. Verificamos lógicamente que dicho provider se encuentra disponible en la web de Terraform, tal y como podemos ver en la imagen siguiente:



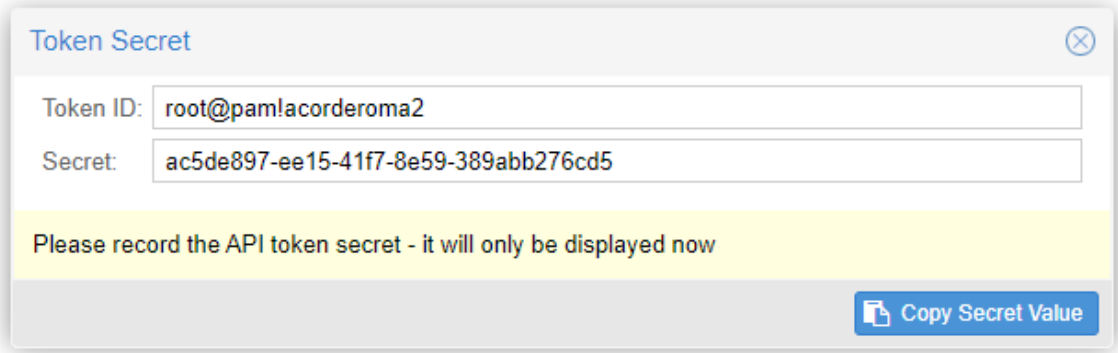
Por otro lado , para poder trabajar con Terraform, necesitamos crear un Token para que nuestro usuario pueda acceder al API. Ello lo realizamos desde la GUI del servidor de Proxmox, concretamente en el Datacenter:



Generamos un TokenID para nuestro usuario, en este caso root@pam:



Y podremos copiarlo, ya que una vez cerrada dicha ventana, no podemos conocer nuevamente el token de acceso al API:



A continuación, se generará el fichero provider.tf , el cual contendrá una serie de variables con los valores deseados para poder ejecutar nuestros ficheros en Terraform:

configuracion-mv > terraform > dev > provider.tf

```
terraform {

  required_version = ">= 0.13.0"

  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "2.9.11"
    }
  }
}

variable "proxmox_api_url" {
  type = string
}

variable "proxmox_api_token_id" {
  type = string
  sensitive = true
}

variable "proxmox_api_token_secret" {
  type = string
  sensitive = true
}

provider "proxmox" {

  pm_api_url = var.proxmox_api_url
  pm_api_token_id = var.proxmox_api_token_id
  pm_api_token_secret = var.proxmox_api_token_secret

  pm_tls_insecure = true
}

```

Además, procederemos a crear un fichero , el cual no se compartirá, en el cual se incluirán los datos que por seguridad no pueden ser compartidos, o incluidos en ningún repositorio , pero a los cuales Terraform sí podrá acceder y trabajar con ellos mediante las variables declaradas previamente.

El contenido de dicho fichero de credenciales será el siguiente:

configuracion-mv > terraform > dev > credentials.auto.tfvars

```
proxmox_api_url = "https://192.168.1.65:8006/api2/json"
proxmox_api_token_id = "root@pam!acorderoma2"
proxmox_api_token_secret = "a8c40619-414c-4d9d-88a1-c3e8bc805af9"

```

Ejecutamos **terraform init** para verificar que todo está correcto:

```
root@Cex:/home/ajcordero/TFG/configuracion-mv/terraform/dev# terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of telmate/proxmox from the dependency lock file
- Using previously-installed telmate/proxmox v2.9.11

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
root@Cex:/home/ajcordero/TFG/configuracion-mv/terraform/dev#
```

Vamos ahora a definir nuestras máquinas virtuales a partir del template que hemos creado en el proceso anterior, para ello creamos un fichero , que en este caso llamaré main.tf , en el cual se incluyen las instrucciones para la creación de 5 máquinas virtuales en nuestro servidor de Proxmox:

configuracion-mv > terraform > dev > main.tf

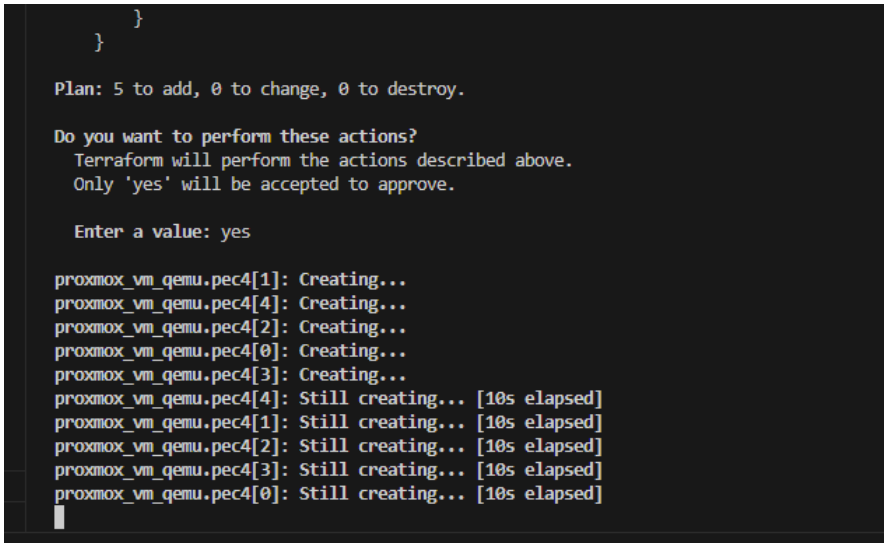
```
variable "proxmox_node" {
  type = string
}

variable "template_name" {
  type = string
}

variable "ssh_key" {
  type = string
  sensitive = true
}

resource "proxmox_vm_qemu" "pec4" {
  count = 5
  name = "tfg20${count.index + 1}"
  target_node = var.proxmox_node
  clone = var.template_name
  agent = 1
  os_type = "cloud-init"
  cores = 2
  sockets = 1
  cpu = "host"
  memory = 4096
  scsihw = "virtio-scsi-pci"
  bootdisk = "scsi0"

```

```
34
35    network {
36      model = "virtio"
37      bridge = "vmbr2"
38    }
39
40    lifecycle {
41      ignore_changes = [
42        network,
43      ]
44    }
45
46    ipconfig0 = "ip=10.10.10.20${count.index + 1}/24,gw=10.10.10.1"
47    nameserver = "8.8.8.8"
48    sshkeys = <<EOF
49    ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDW9jLknSvrCw1kY6Pp9VX/8zzHsQhy7TоozU3F
50    EOF
51
52
53  }
```

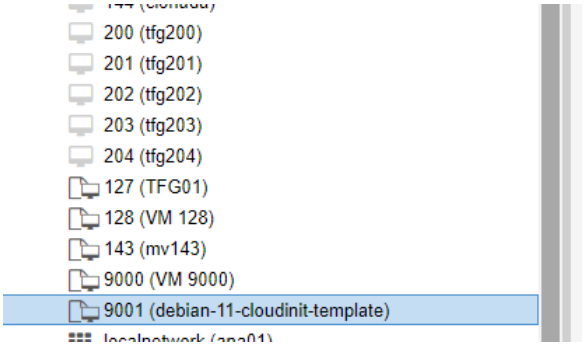
Como vemos una vez ejecutado terraform plan , nos indica que se crearán 5 nuevas máquinas virtuales en nuestro servidor de Proxmox:

```
32      storage = "data"
33    }

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS 4

      + mbps_wr          = 0
      + mbps_wr_max      = 0
      + media            = (known after apply)
      + replicate        = 0
      + size             = "2G"
      + slot             = 0
      + ssd              = 0
      + storage          = "data"
      + storage_type     = (known after apply)
      + type             = "scsi"
      + volume           = (known after apply)
    }

  + network {
      + bridge    = "vmbr2"
      + firewall  = false
      + link_down = false
      + macaddr   = (known after apply)
      + model     = "virtio"
      + queues    = (known after apply)
      + rate      = (known after apply)
      + tag       = -1
    }
  }

Plan: 5 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
root@Cex:/home/ajcordero/TFG/configuracion-mv/terraform/dev
```

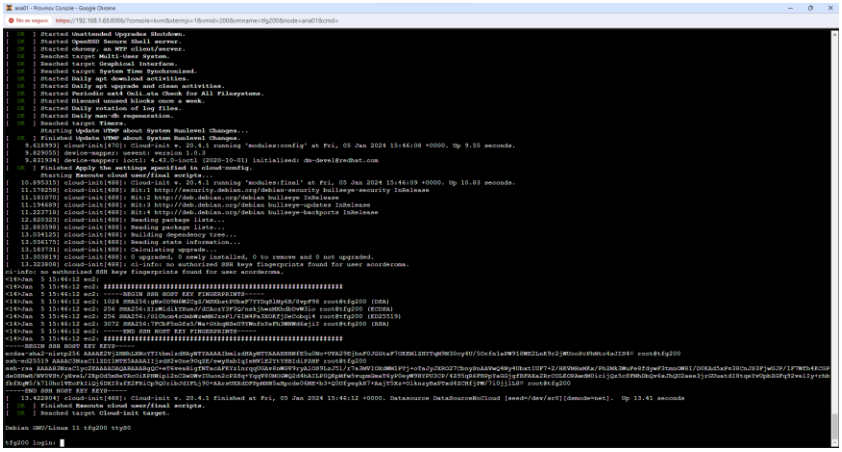
Por último, ejecutamos terraform apply:



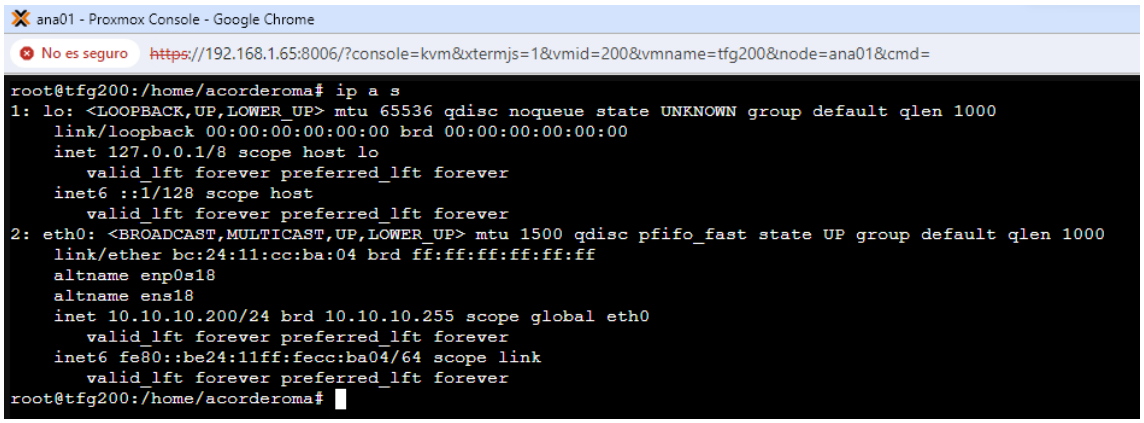
Después de unos minutos, podemos ver nuestras nuevas máquinas virtuales creadas en el interfaz de Proxmox:



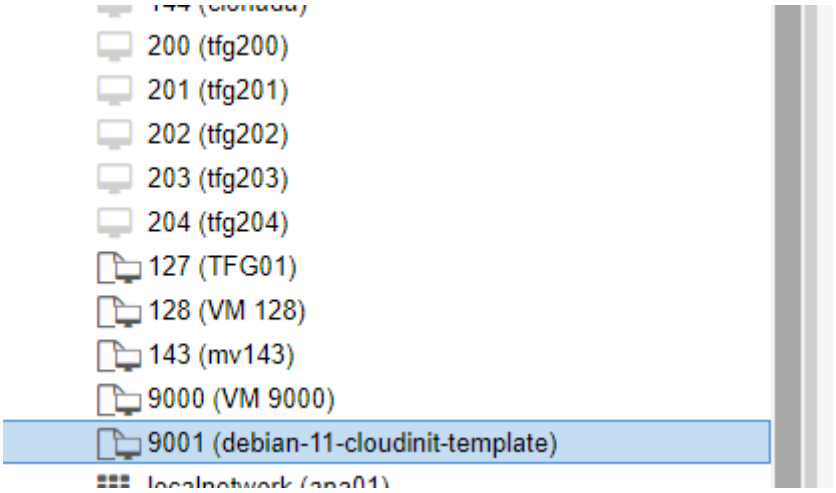
Si arrancamos nuestras máquinas podremos acceder a ellas. Sin lanzamos la terminal de la máquina tfg200 arrancado:



Configuración de red:



Como resultado final del proceso tenemos cinco nuevos servidores, llamados tfg200, tfg201, tfg202, tfg203, tfg204 creados mediante **Terraform** a partir de un clonado total del template debian-11-cloud-init que habíamos generado anteriormente en el host Proxmox:

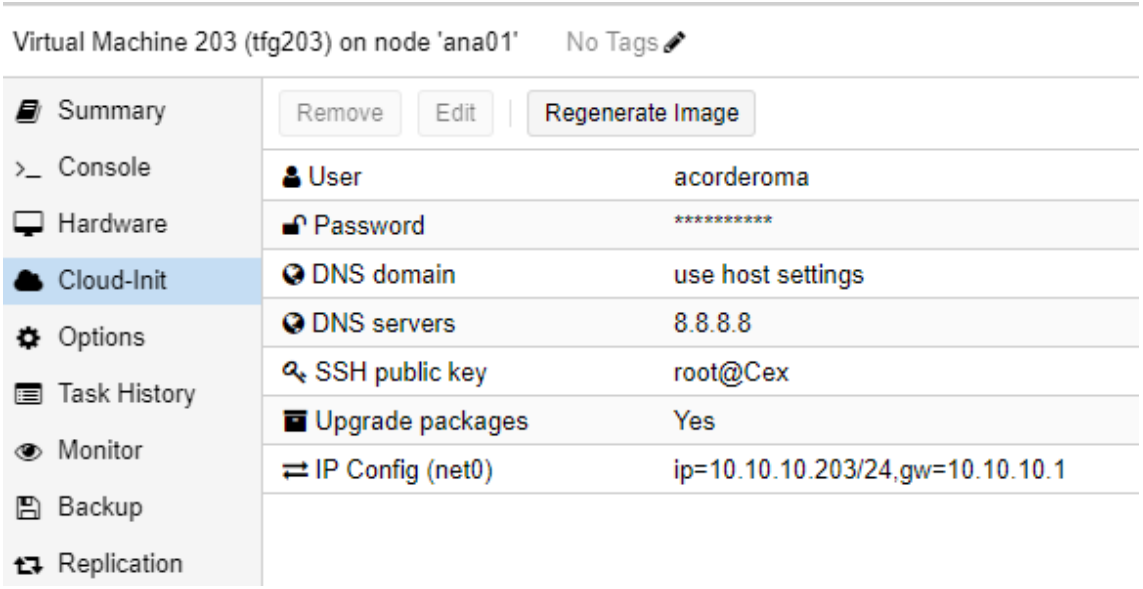


13. Asignación de Roles Lógicos con Ansible.

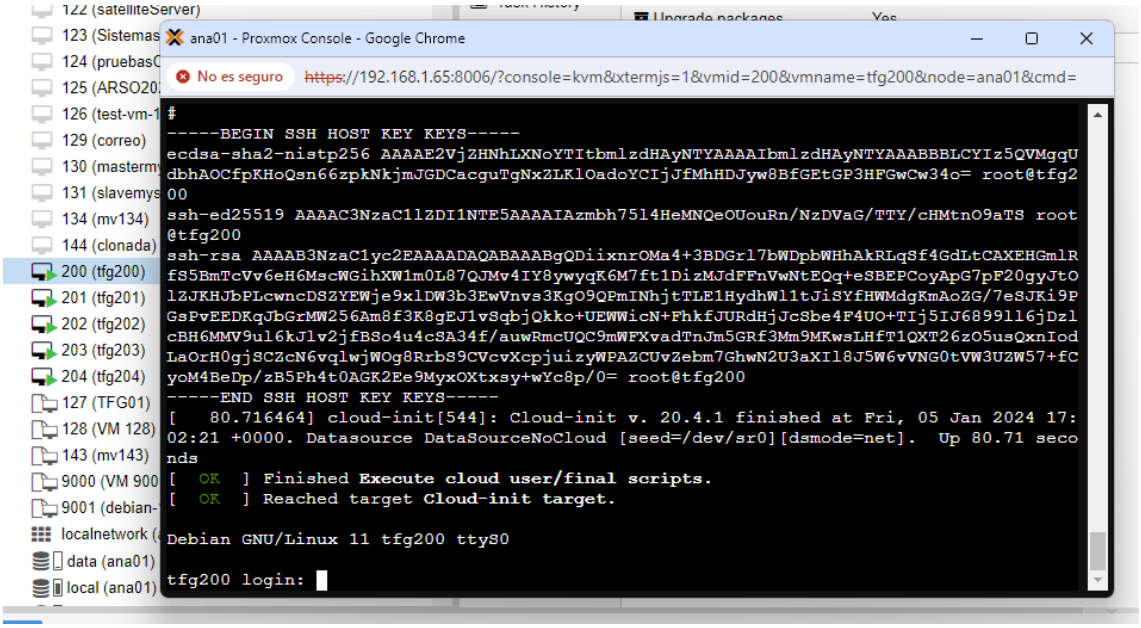
El role de los cuatro primeros servidores va a ser de Servidor de Web, mientras que el servidor restante (tfg204) tendrá role de Servidor de Base de Datos.
Para asignar dichos roles , primero tenemos que indicar en nuestro fichero de inventario de ansible (en este caso /etc/ansible/hosts en nuestro equipo de escritorio) la existencia de dichas máquinas y el role que les corresponde de la siguiente manera:

```
[proxmoxservers]
ana01 ansible_host=192.168.1.65 ansible_user=root

[webservers]
tfg200 ansible_host=10.10.10.200 ansible_user=acorderoma
tfg201 ansible_host=10.10.10.201 ansible_user=acorderoma
tfg202 ansible_host=10.10.10.202 ansible_user=acorderoma
tfg203 ansible_host=10.10.10.203 ansible_user=acorderoma

[mysqlservers]
tfg204 ansible_host=10.10.10.204 ansible_user=acorderoma
```

Procedemos a verificar que efectivamente, nuestras máquinas virtuales creadas tienen los datos que habíamos incluido en el fichero Terraform de generación de máquinas, verificando concretamente los valores de red deseados, para poder esperar un buen resultado en el arranque de cloud init. Por ejemplo , si vemos la configuración de cloud-init para la máquina virtual tfg203, podemos observar que los datos son los correctos:

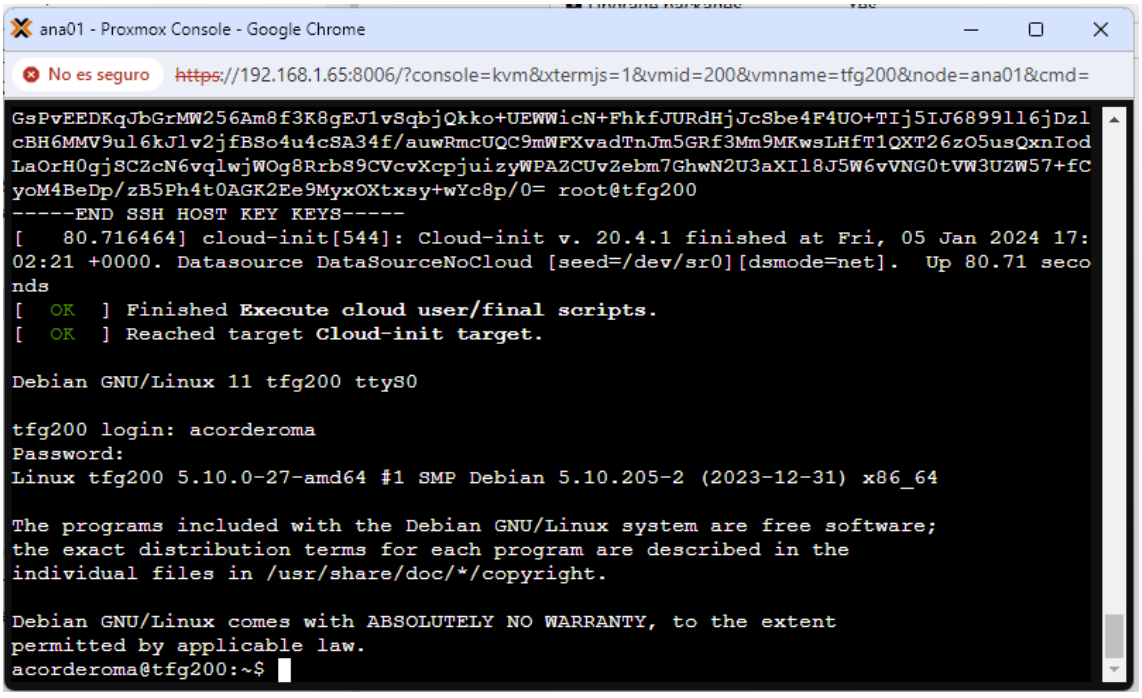


Por tanto, ya podemos arrancar nuestras máquinas y comenzar a configurarlas mediante ansible.

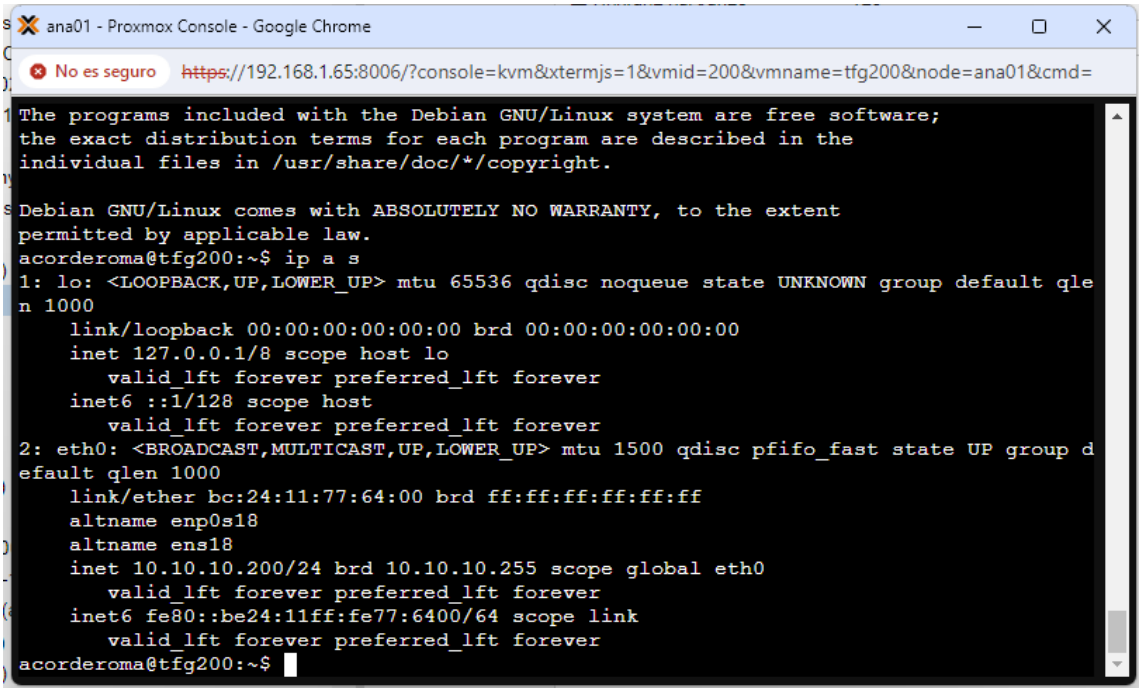


Como podemos observar en la imagen anterior, nuestra máquina tfg es accesible por consola, ha realizado correctamente el arranque, y está levantada esperando que el usuario abra una sesión en ella.

El usuario que hemos indicado en el cloud-init es acorderoma, por tanto, vamos a introducir dicho nombre de usuario y su password:



Vemos que el login es correcto. Por tanto, de manera preventiva, pasamos a comprobar la IP con la que la máquina levantó , que se tendría que corresponder con la 10.10.10.200, ya que la asignación de ips, se ha de corresponder con el número de id de la máquina:



De la misma manera, podemos verificar que el resto de las máquinas se ha levantado de manera correcta.

13.1 Asignación de Roles a cada servidor Mediante Ansible.

13.1.1 Asignación del Role Servidor de Web.

Para asignar los roles a cada uno de los hosts, se han creado dos playbooks simples de ansible. Uno realizará la instalación de apache en cada uno de los servidores que corresponden al role Servidor de Web (webserver) y el otro , instalar el servidor de base de datos en la máquina que posee el role de Servidor de Base de Datos (mysqlserver).
El primer playbook llamado instalar_apache.yaml , actualizará la paquetería de debian, e instalará apache en el caso de que éste no se encuentre en el servidor de destino. El contenido de dicho playbook es el siguiente:

Proxmox_Container_Con_Ansible > instalar_apache.yaml

```yaml
---
- hosts: webservers
  become: true
  tasks:
    - name: Actualizar paquetería mediante apt
      apt:
        upgrade: yes
        update_cache: yes
        cache_valid_time: 86400

    - name: Verificar si Apache está instalado
      stat:
        path: /usr/sbin/apache2
      register: apache_check

    - name: Instalar Apache
      apt:
        name: apache2
        state: present
      when: apache_check.stat.exists == false
      become: true
```

Procedemos a ejecutarlo:

```
ajcordero@Cex: ~/Proxmox_C
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ansible-playbook instalar_apache.yaml
```

Vemos durante la ejecución que:

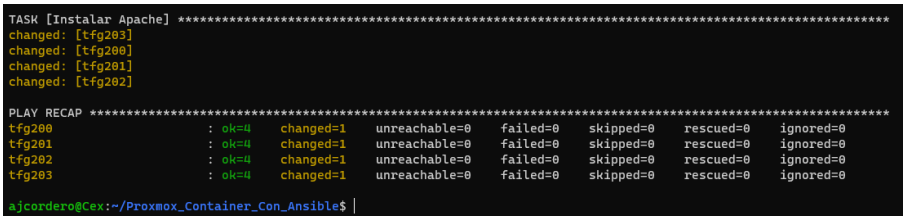
- Actualiza la paquetería en los cuatro servidores web.

```
ok: [tfg200]
ok: [tfg203]

TASK [Actualizar paquetería mediante apt] ********
ok: [tfg201]
ok: [tfg200]
ok: [tfg203]
ok: [tfg202]
```

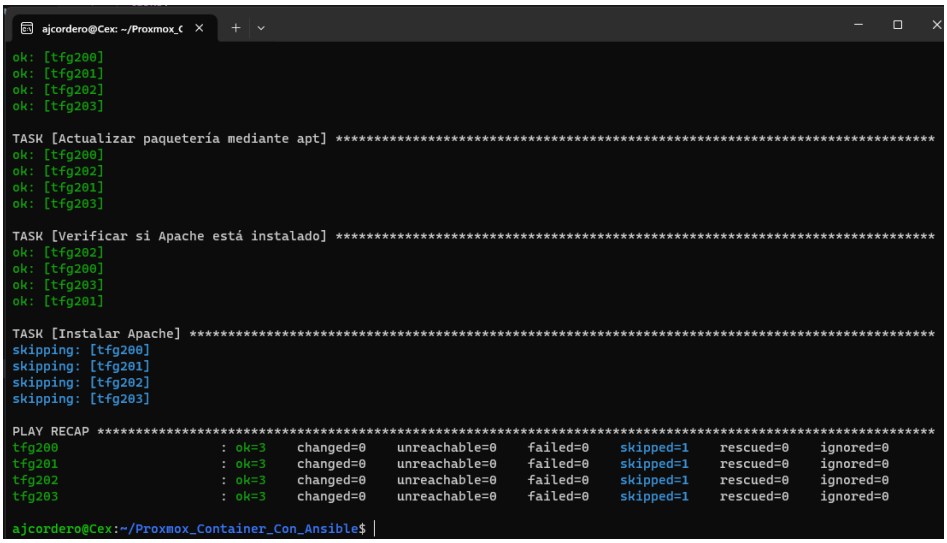
- Verifica si apache ya se encuentra instalado:

```
TASK [Verificar si Apache está instalado] ****
ok: [tfg202]
ok: [tfg201]
ok: [tfg203]
ok: [tfg200]
```

- Y finalmente , al no encontrarse en ellos , lo instala con resultado exitoso:

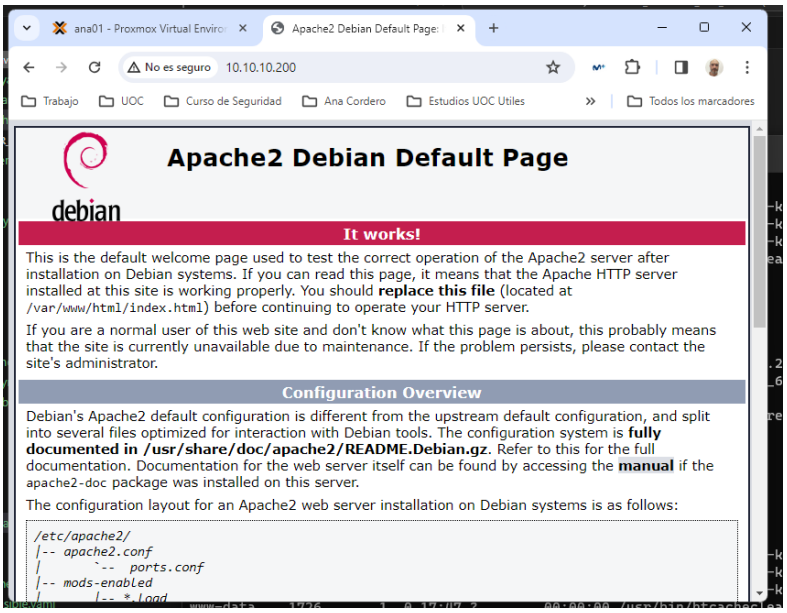


En el caso de haber existido una instalación de apache en los servidores, no se hubieran realizado cambios , tal y como podemos observar en una nueva ejecución del playbook de instalación en la siguiente imagen:



Podemos verificar en cada uno de los servidores que apache se encuentra levantado y esperando tráfico de red:

Si abrimos un navegador web, podremos acceder a cada uno de los servidores de role Servidor Web mediante el uso de este:



Por lo tanto , cada uno de los servidores frontales de web, queda configurado con apache instalado y funcionando.

13.1.2 Asignación del Role Servidor de Base de Datos.

Tal y como se ha referido anteriormente, se ha creado otro playbook de Ansible para la asignación del Role de servidor de base de datos en este caso sólo a una de las máquinas virtuales creadas, la tfg204, que es la única que hemos indicado en nuestro fichero de configuración de ansible que va a cumplir dicho Role.

Veamos nuevamente que mediante un sencillo playbook , procederemos a actualizar todas las máquinas que posean dicho role a nivel de paquetería. Posteriormente se realizará la instalación del servidor MySQL en la misma, ya que en nuestro caso es solamente una de ellas.

Proxmox_Container_Con_Ansible > instalar_mysql.yaml

```yaml
---
- name: Actualizar paquetería del Sistema Operativo y configurar mysql en SBD
  hosts: mysqlservers
  become: true
  tasks:
    - name: Actualizacíon de paquetes
      apt:
        upgrade: yes
        update_cache: yes
        cache_valid_time: 86400

    - name: Instalar mysql
      apt:
        name: mysql-server
        state: present
```

Ejecutamos en nuestro equipo de escritorio el playbook:

```
ajcordero@Cex: ~/Proxmox_C
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ansible-playbook instalar_mysql.yaml
```

Como en el caso anterior, vemos que primeramente realiza la actualización de la paquetería del sistema operativo. En nuestro caso la distribución Debian 11:

```
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ansible-playbook instalar_mysql.yaml

PLAY [Actualizar paquetería del Sistema Operativo y configurar mysql en SBD] ***************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [tfg204]

TASK [Actualizacíon de paquetes] ***********************************************************************************
ok: [tfg204]
```

Y después de instalar los paquetes y mysql , vemos que el resultado ha sido correcto:

```
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ansible-playbook instalar_mysql.yaml

PLAY [Actualizar paquetería del Sistema Operativo y configurar mysql en SBD] ***************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [tfg204]

TASK [Actualizacion de paquetes] ***********************************************************************************
ok: [tfg204]

TASK [Instalar mysql] **********************************************************************************************
changed: [tfg204]

PLAY RECAP *********************************************************************************************************
tfg204                     : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ajcordero@Cex:~/Proxmox_Container_Con_Ansible$
```

Para verificar que se ha instalado correctamente, accedemos al servidor tfg204, y vemos que el proceso mysql está lanzado. Concretamente, lanzará **maríadb** , ya que es el servidor de base de datos que por **defecto** provee esta distribución de Linux:

```
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ssh acorderoma@10.10.10.204
Linux tfg204 5.10.0-27-amd64 #1 SMP Debian 5.10.205-2 (2023-12-31) 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: Fri Jan  5 18:13:32 2024 from 192.168.1.17
acorderoma@tfg204:~$ ps -fe | grep mysql
mysql       2032       1  0 18:14 ?        00:00:00 /usr/sbin/mariadbd
acorder+    3014    3008  0 18:16 pts/0    00:00:00 grep mysql
acorderoma@tfg204:~$ netstat -putan
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0    360 10.10.10.204:22         192.168.1.17:10300      ESTABLISHED -
tcp6       0      0 :::22                   :::*                    LISTEN      -
udp        0      0 0.0.0.0:68              0.0.0.0:*                           -
udp        0      0 127.0.0.1:323           0.0.0.0:*                           -
udp6       0      0 ::1:323                 :::*                                -
acorderoma@tfg204:~$
```

Como podemos ver está escuchando en el puerto 3306. Y podemos verificar que está levantado mediante systemctl:

```
acorderoma@tfg204:~$ systemctl status mysql.service
● mariadb.service - MariaDB 10.5.21 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2024-01-05 18:14:18 UTC; 4min 33s ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
    Process: 1982 ExecStartPre=/usr/bin/install -m 755 -o mysql -g root -d /var/run/mysqld (code=exited, status=0/SUCCE>
    Process: 1983 ExecStartPre=/bin/sh -c systemctl unset-environment _WSREP_START_POSITION (code=exited, status=0/SUCC>
    Process: 1985 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/galera_recovery ] && VAR= ||   VAR=`cd /usr/bin/..; /usr/bin/>
    Process: 2049 ExecStartPost=/bin/sh -c systemctl unset-environment _WSREP_START_POSITION (code=exited, status=0/SUC>
    Process: 2051 ExecStartPost=/etc/mysql/debian-start (code=exited, status=0/SUCCESS)
   Main PID: 2032 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 9 (limit: 4660)
     Memory: 69.3M
        CPU: 493ms
     CGroup: /system.slice/mariadb.service
             └─2032 /usr/sbin/mariadbd

Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] InnoDB: 10.5.21 started; log sequence number 45079;>
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] Plugin 'FEEDBACK' is disabled.
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/>
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] InnoDB: Buffer pool(s) load completed at 240105 18:>
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] Server socket created on IP: '127.0.0.1'.
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] Reading of all Master_info entries succeeded
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] Added new Master_info '' to hash table
Jan 05 18:14:18 tfg204 mariadbd[2032]: 2024-01-05 18:14:18 0 [Note] /usr/sbin/mariadbd: ready for connections.
Jan 05 18:14:18 tfg204 mariadbd[2032]: Version: '10.5.21-MariaDB-0+deb11u1'  socket: '/run/mysqld/mysqld.sock'  port: 3>
Jan 05 18:14:18 tfg204 systemd[1]: Started MariaDB 10.5.21 database server.
lines 1-28/28 (END)
```

Como vemos, el servidor mysql se encuentra arrancado y disponible. Aún así , aún habrá que ejecutar el proceso de configuración del servidor. Para evitar esto , podemos crear un nuevo playbook que lo haga por nosotros aprovechando la potencia de ansible.

Como ejemplo, creo una nueva máquina virtual (tfg205) , se añade al fichero de hosts la máquina virtual y ejecutamos el siguiente playbook , que instalará MaríaDB en dicha máquina y además realizará el proceso de Secure Installation de manera automática , pudiendo acceder a ella como root una vez finalizado:

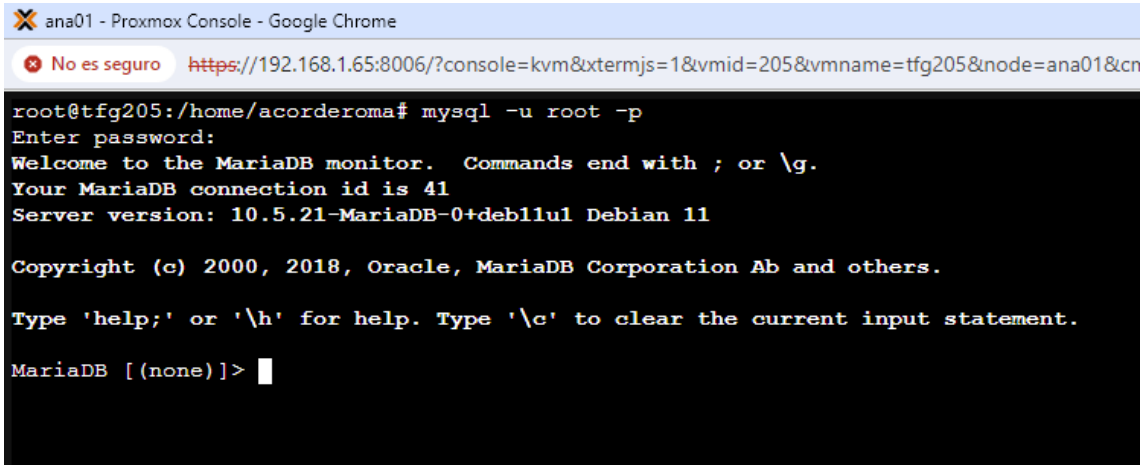
Proxmox_Container_Con_Ansible > instalar_mysql2.yaml

```yaml
---
- name: Instalar MariaDB y ejecutar secure installation
  hosts: tfg205
  become: yes

  tasks:
    - name: Actualizar paqueteria
      apt:
        upgrade: yes
        update_cache: yes
        cache_valid_time: 86400

    - name: Instalar paquetes del servidor de BBDD
      apt:
        name:
          - default-mysql-server
          - python3-pymysql

    - name: Ejecutar secure installation
      shell: |
        mysql_secure_installation <<EOF

        y
        # Establecer contraseña de root aquí
        acorderoma
        acorderoma
        y
        y
        y
        y
        EOF
      args:
        executable: /bin/bash

```

Procedemos a la ejecución del playbook:

```
ajcordero@Cex:~/Proxmox_Container_Con_Ansible$ ansible-playbook instalar_mysql2.yaml

PLAY [Instalar MariaDB y ejecutar secure installation] *************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [tfg205]

TASK [Actualizar paqueteria] ***************************************************************************************
ok: [tfg205]

TASK [Instalar paquetes del servidor de BBDD] **********************************************************************
ok: [tfg205]

TASK [Ejecutar secure installation] ********************************************************************************
changed: [tfg205]

PLAY RECAP *********************************************************************************************************
tfg205                     : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ajcordero@Cex:~/Proxmox_Container_Con_Ansible$
```

Accedemos al servidor MySQL:



14. Aprovechando los Roles en Ansible.

Hasta este punto hemos estado trabajando los Roles de ansible de una manera lógica. Es decir, el administrador del sistema sabe el role que ejecutará cada una de las máquinas que ha creado y en función de ello, ejecuta una serie de playbooks que tendrá preparados para asignar dicha función al nodo.

Ahora bien, esta manera no es la más eficiente. Cuando trabamos con Ansible, un "role" es una unidad de organización y reutilización de configuraciones y tareas en un proyecto. Son una colección de tareas, archivos de configuración, variables y plantillas agrupados para cumplir un propósito específico.

Pasamos a ver el concepto de Role de Ansible de manera más detallada en los siguientes puntos.

14.1 Role nginx y descripción genérica del procedimiento.

Para la creación de los Roles, usaremos el comando ansible-galaxy. Con ello creamos la estructura de directorios necesaria para poder definir el role, en concreto , vamos a definir un nuevo role que llamaremos nginx:

```
root@Cex:/home/ajcordero/TFG_ansible# ansible-galaxy init ./roles/nginx --offline
- Role ./roles/nginx was created successfully
root@Cex:/home/ajcordero/TFG_ansible#
```

El resultado de la ejecución del comando ansible-galaxy con el parámetro init es una estructura de subdirectorios dentro del directorio indicado (./roles/nginx) que es la siguiente:

```
root@Cex:/home/ajcordero/TFG_ansible# tree ./roles/nginx
./roles/nginx
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
root@Cex:/home/ajcordero/TFG_ansible#
```

En la siguiente tabla se refleja la función de cada uno de los subdirectorios creados dentro del directorio indicado, que hemos pasado como parámetro en la ejecución del comando ansible-galaxy:

Subdirectorio	Función
Defaults	Variables por defecto que se usarán en el role.
Files	Contendrá los archivos necesarios para la asignación del role a la máquina de destino.
Handlers	Contiene las funciones de control , para ser utilizadas por el role.
Meta	Los datos de información del Role. Autor, versionado, etc.
Tasks	Lista principal de tareas que debe ejecutar el role.
Templates	Las plantillas que se pueden implementar a través de este Role.
Tests	
Vars	Otras variables para el role.

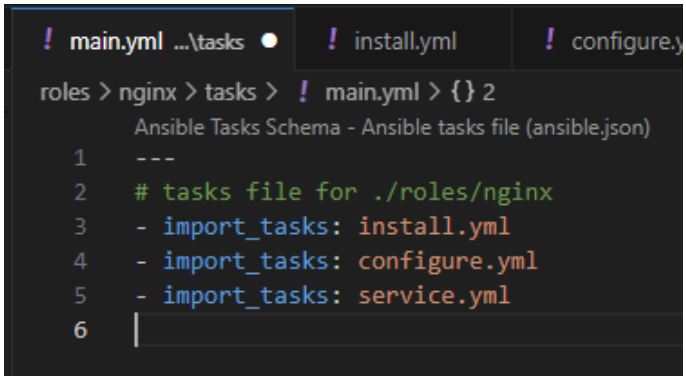
Dentro de directorio tasks, tenemos un fichero llamado main.yml. En dicho fichero , se definirán las tareas que se van a ejecutar dentro del role.

Lógicamente, para este role de ejemplo que estamos examinando , las tareas básicas serán la instalación de nginx dentro de la máquina, configurar nginx y reiniciar el servicio una vez configurado para que se puedan aplicar los cambios indicados. Para ello, aprovechando la facilidad que nos da ansible de simplificación de estructura de código, crearemos una serie de ficheros individuales que serán llamados desde task/main.yml para su ejecución paso a paso.

Por tanto, el contenido del fichero task/main.yml para el role nginx en un principio es el siguiente:

```
root@Cex:/home/ajcordero/TFG_ansible# cat ./roles/nginx/tasks/main.yml
---
# tasks file for ./roles/nginx
root@Cex:/home/ajcordero/TFG_ansible#
```

Editamos el fichero para realizar las acciones definidas anteriormente, de manera secuencial:



Como vemos, se han definido tres playbooks sencillos que realizarán las tareas descritas anteriormente. Realizando un examen del contenido de cada uno de dichos ficheros es el siguiente:

Una vez creado todo el contenido necesario que define a nuestro role, procederemos a realizar un chequeo de la sintaxis de creación del role:

```
root@Cex:/home/ajcordero/TFG_ansible# ansible-playbook instalar_nginx.yml --syntax-check

playbook: instalar_nginx.yml
root@Cex:/home/ajcordero/TFG_ansible#
```

Vemos que es correcto y no depara ningún error, por lo tanto , podemos ejecutarlo:

14.2 Role apache.

Utilizando la misma estructura que anteriormente es ejecutó para la creación del role nginx, pasamos a realizar los procesos necesarios para la creación del role de los tres servidores de web que componen el Backend.

El procedimiento, tal y como se comenta anteriormente, será el mismo que se ha seguido para la creación del role de nginx, por lo que no se incidirá en el comentario de cada uno de los pasos más que de manera esquemática.

En este caso se genera un role apache, que incluirá la instalación, configuración y puesta en marcha del servidor web en cada una de las instancias de máquina que cumplan dicho role (para este proyecto serán las máquinas test-201, test-202 y test-203).

En un primer paso , se crea la estructura del role, utilizando el comando ansible-galaxy:

```
root@Cex:/home/ajcordero/TFG_ansible# ansible-galaxy init ./roles/apache --offline
- Role ./roles/apache was created successfully
root@Cex:/home/ajcordero/TFG_ansible#
```

Verificamos la correcta creación de la estructura de directorios:

```
root@Cex:/home/ajcordero/TFG_ansible# tree ./roles/apache
./roles/apache
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
root@Cex:/home/ajcordero/TFG_ansible#
```

A continuación, vamos a estructurar el fichero ./task/main.yml para la configuración del role, que consistirá en la instalación, configuración y ejecución de apache dentro del servidor de cumpla dicho role:

roles > apache > tasks > ! main.yml > {} 2

Ansible Tasks Schema - Ansible tasks file (ansible.json)

```
1  ---
2  # tasks file for ./roles/apache
3  - import_tasks: install.yml
4  - import_tasks: configure.yml
5  - import_tasks: service.yml
```

Finalmente, incluimos el código ansible necesario en cada una de las tareas, para realizar la instalación de apache y su ejecución. Así, los ficheros , tendrán el siguiente contenido:

- install.yml

roles > apache > tasks > ! install.yml > {} 0

Ansible Tasks Schema - Ansible tasks file (ansible.json)

```
1  ---
2  - name: Instala apache2 en su ultima versión
3    become: true
4    apt: name=apache2 state=latest
5
```

- configure.yml

roles > apache > tasks > ! configure.yml > ...

Ansible Tasks Schema - Ansible tasks file (ansible.json)

```
1  ---
2  - name: copia el fichero index.html al nuevo servidor de web
3    become: true
4    copy: src=files/index.html dest=/var/www/html
5    notify:
6    - restart apache
7
```

- service.yml

roles > apache > tasks > ! service.yml > {} 0 > become

Ansible Tasks Schema - Ansible tasks file (ansible.json)

```
1  ---
2  - name: Inicia apache
3    become: true
4    service: name=apache2 state=restarted enabled=yes
5
```

Por último, editamos el fichero handlers/main.yml y añadimos el servicio de reinicio de apache:

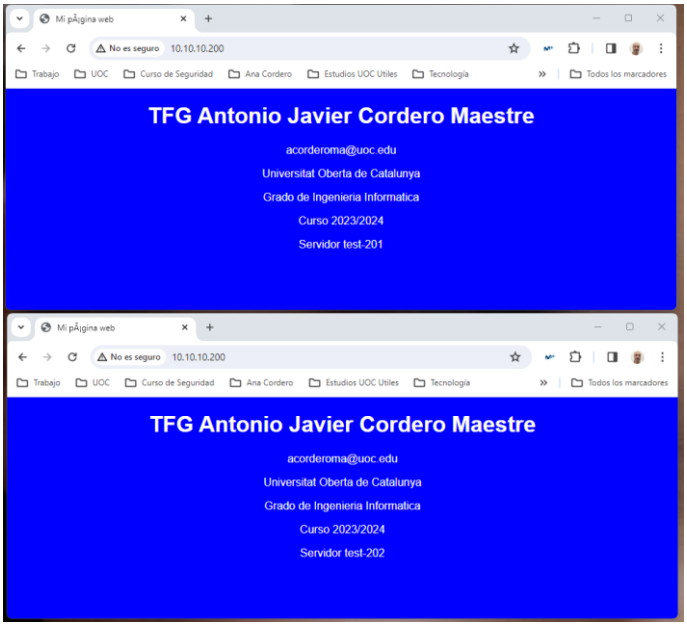
```
roles > apache > handlers > ! main.yml > {} 0
      Ansible Tasks Schema - Ansible tasks file (ansible.json)
1     ---
2     # handlers file for ./roles/apache
3     - name: restart apache
4       become: true
5       service: name=apache2 state=restarted
6
```

Como siguiente paso se procede a comprobar que no existen problemas de sintaxis:

```
root@Cex:/home/ajcordero/TFG_ansible# ansible-playbook instalar_www.yml --syntax-check

playbook: instalar_www.yml
root@Cex:/home/ajcordero/TFG_ansible#
```

Finalmente, ejecutamos el playbook:

Tenemos a los servidores funcionando:



14.3 Role DataBaseServer.

Con este role se pretende instalar en el host de destino un servidor de base de datos SQL. En este caso, el servidor que trae la distribución por defecto es maríadb 10.5, el cual será instado una vez se invoque este Role.

Como ocurría en el Role anterior, la estructura que se utilizará será la mismas que en los roles anteriores, es decir, que la operativa no variará más que para la parte específica de instalación del servidor de base de datos y es por lo que ahorraremos extensión en el documento y pasaremos directamente a describir su instalación.

Este Role, será aplicado a la máquina test-203 para que pueda ejecutar el servidor de base de datos.

Comenzamos con la creación de la estructura de directorios del Role:

```
root@Cex:/home/ajcordero/TFG_ansible# ansible-galaxy init ./roles/databaseserver --offline
- Role ./roles/databaseserver was created successfully
root@Cex:/home/ajcordero/TFG_ansible#
```

Nuevamente , revisamos la estructura para ver que todo está correcto a nivel de ficheros y directorios:

```
root@Cex:/home/ajcordero/TFG_ansible# tree roles/databaseserver/
roles/databaseserver/
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
root@Cex:/home/ajcordero/TFG_ansible#
```

Se detallan nuevamente las tareas que se van a ejecutar en roles/databaseserver/tasks/main.yml:

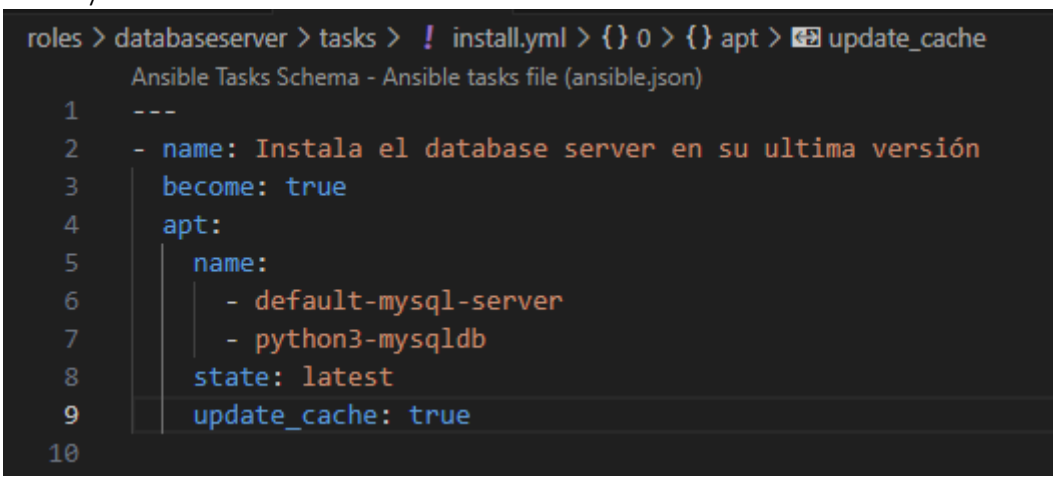
roles > databaseserver > tasks > ! main.yml > {} 2

Ansible Tasks Schema - Ansible tasks file (ansible.json)

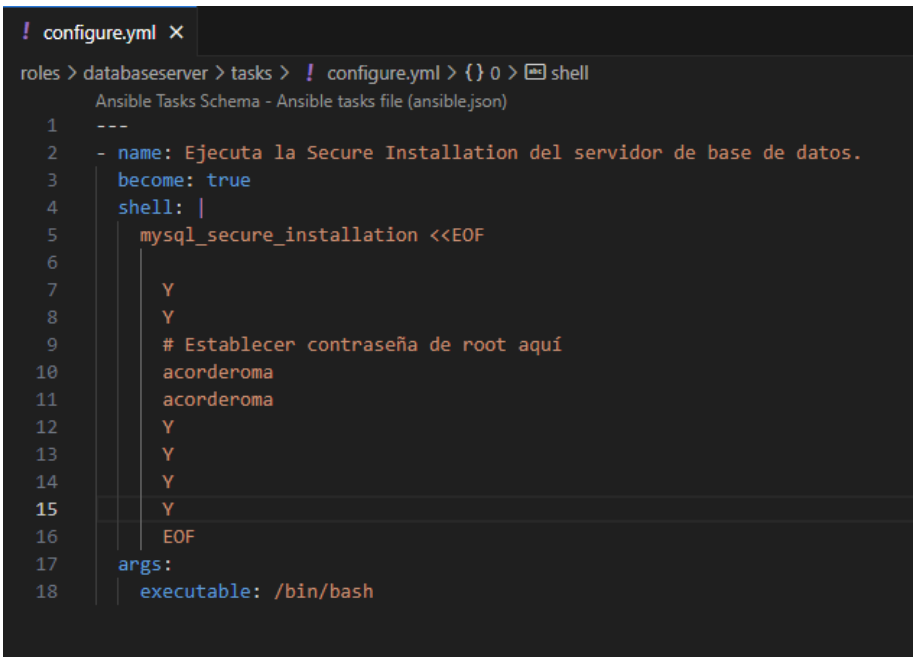
```yaml
---
# tasks file for ./roles/databaseserver
- import_tasks: install.yml
- import_tasks: configure.yml
- import_tasks: service.yml
```

Y se crea cada uno de los ficheros vistos anteriormente para que cumplan su función en la instalación del Role databaseserver.

- install.yml



- configure.yml
  En este caso, se está simulando la introducción por teclado de la ejecución del script mysql_secure_instalation. Como vemos, simplemente damos Yes a todas las opciones y además , introducimos la nueva clave de root.



- service.yml
  Por último, editamos el fichero handlers/main.yml y añadimos el servicio de reinicio del servidor SQL:

```
roles > databaseserver > handlers > ! main.yml > {} 0
      Ansible Tasks Schema - Ansible tasks file (ansible.json)
1     ---
2     # handlers file for ./roles/databaseserver
3     - name: restart sql server
4       become: true
5       service: name=mysql state=restarted
6
```

Se realiza el chequeo de sintaxis:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

ajcordero@Cex:~/TFG_ansible$ ansible-playbook instalar_basededatos.yml --syntax-check

playbook: instalar_basededatos.yml
ajcordero@Cex:~/TFG_ansible$
```

Se ejecuta el playbook:

Y quedan los servidores test-203 y test-204 con su nuevo Role de servidor de base de datos configurado.

15. Conclusión

Durante la realización de este proyecto, hemos podido verificar, que a partir de software libre y aprovechando una serie de recursos desfasados temporalmente , es posible entregar en producción elementos que pertenecen al concepto de nube privada, sin necesidad de realizar un gran desembolso económico.

Lógicamente, como suele ocurrir en el desarrollo de proyectos, las expectativas son mayores que el producto final entregado. La realización de este, de manera individual, hace que los deseos iniciales no puedan ser plasmados en el tiempo deseado, ya que hay que estar entre la vida laboral , los estudios, y por supuesto la vida familiar.

Además, la planificación inicial no fue la adecuada, tomando como partida unos tiempos estimados que no se han podido reflejar en la realización final del mismo.

A pesar de seguir de una manera metódica la planificación, los retrasos en la misma han sido incrementales, entregando un producto final operativo, pero bastante reducido en la potencia que la plataforma y las herramientas nos permitían.

Por otro lado, el embarcarme en la realización a modo de reto de un proyecto sobre una temática que anteriormente no había trabajado, ansible / terraform, me ha permitido crecer en el mundo de la automatización, en las tecnologías actuales y en comprender el nuevo enfoque DevOps que está moviendo de posición al Administrador de Sistemas clásico, siendo esta última posición indicada en la cual me encuentro en la actualidad.

Me apena no haber podido avanzar más , ya que las posibilidades que las herramientas utilizadas ofrecen son infinitas, no sólo a nivel de los servicios entregados, también en el diseño de la red, seguridad, etc. que no han podido ser abordados, tal y como me gustaría haber hecho.

Por tanto, no me queda más que concluir que he aprendido a que hay que ser más realista en los planteamientos iniciales y en el desarrollo de los mismos, desde un punto de vista de la temporalidad, además de ser consciente que las cargas de trabajo que se pueden generar a partir de un proyecto pueden ser una rémora si no se trabaja con el equipo adecuado.

Aún así , me quedo con todo lo aprendido, que servirá de referencia para introducirme en el mundo de la automatización de sistemas de una manera eficiente.

16. Glosario

- Ansible: Es una herramienta de automatización de TI que permite configurar y administrar sistemas de manera eficiente. Permite definir y desplegar configuraciones en múltiples servidores de forma simultánea.

- Ansible-Galaxy: Es una herramienta de Ansible que permite compartir y reutilizar roles de configuración y módulos creados por la comunidad.

- Debian: Es una distribución de Linux estable, popular y de código abierto. Se caracteriza por su enfoque en la estabilidad y la seguridad.

- Máquina virtual: Es una representación virtual de un ordenador o servidor físico que puede ejecutar sistemas operativos y aplicaciones como si fuera una computadora independiente. Permite compartir los recursos de hardware de un servidor físico entre múltiples máquinas virtuales.

- Inventario de Ansible: Es un archivo en el que se especifican los servidores o dispositivos en los que se ejecutarán las tareas de Ansible. Contiene información como las direcciones IP, nombres de host y grupos de hosts.

- Ordenador: Es un dispositivo electrónico capaz de procesar datos y realizar diversas tareas. También conocido como computadora o computador, es utilizado para realizar cálculos, almacenar información, ejecutar programas y conectarse a redes.

- Orquestación: Es el proceso de automatizar y coordinar la ejecución de tareas en múltiples sistemas o dispositivos de forma secuencial o en paralelo. Ansible es una herramienta popular para la orquestación de infraestructuras.

- Playbook de Ansible: Es un archivo en formato YAML que contiene una serie de instrucciones o tareas que Ansible debe ejecutar en los sistemas gestionados. Los playbooks definen el estado deseado del sistema y las acciones necesarias para alcanzarlo.

- Proxmox: Plataforma de virtualización de código abierto que permite administrar y virtualizar servidores. Proporciona una interfaz web para gestionar máquinas virtuales y contenedores, así como otras funcionalidades como el almacenamiento y la alta disponibilidad.

- Servidor: Ordenador o sistema informático que proporciona servicios, recursos o funcionalidades a otros dispositivos o usuarios en una red. Los servidores están diseñados para gestionar y responder a las solicitudes de otros equipos, como el almacenamiento de archivos, el hosting de sitios web o el procesamiento de datos.

- Terraform: Es una herramienta de infraestructura como código, que permite crear y gestionar la infraestructura de manera automatizada. Permite definir la infraestructura deseada en un archivo de configuración y desplegarla en diferentes recursos.

- Windows 11: Última versión del sistema operativo de Microsoft lanzada en 2021.

- WSL2: Windows Subsystem for Linux 2 (Subsistema de Windows para Linux 2) es una capa de compatibilidad de Windows que permite ejecutar un entorno Linux dentro de Windows.

- Ubuntu: Es una distribución de Linux basada en Debian y ampliamente utilizada. Su atractivo principal es la facilidad de uso y su enfoque en la usabilidad y accesibilidad.

17. Bibliografía

- Autor: David González
  Título: Ansible: Automatización para todos
  Edición: 1ª edición
  Editorial: RA-MA Editorial
  Ciudad: Madrid, España
  Año: 2020

- Autor: Marcos Gómez Hidalgo
  Título: Aprendiendo Ansible: Automatiza la configuración de tu entorno
  Edición: 1ª edición
  Editorial: Marcombo
  Ciudad: Barcelona, España
  Año: 2019

- Autor: Juan Manuel Rey
  Título: Aprendiendo Terraform: Construye y administra tu infraestructura como código
  Edición: 1ª edición
  Editorial: Marcombo
  Ciudad: Barcelona, España
  Año: 2020

18. Anexos

18.1 Diagrama de la Arquitectura.