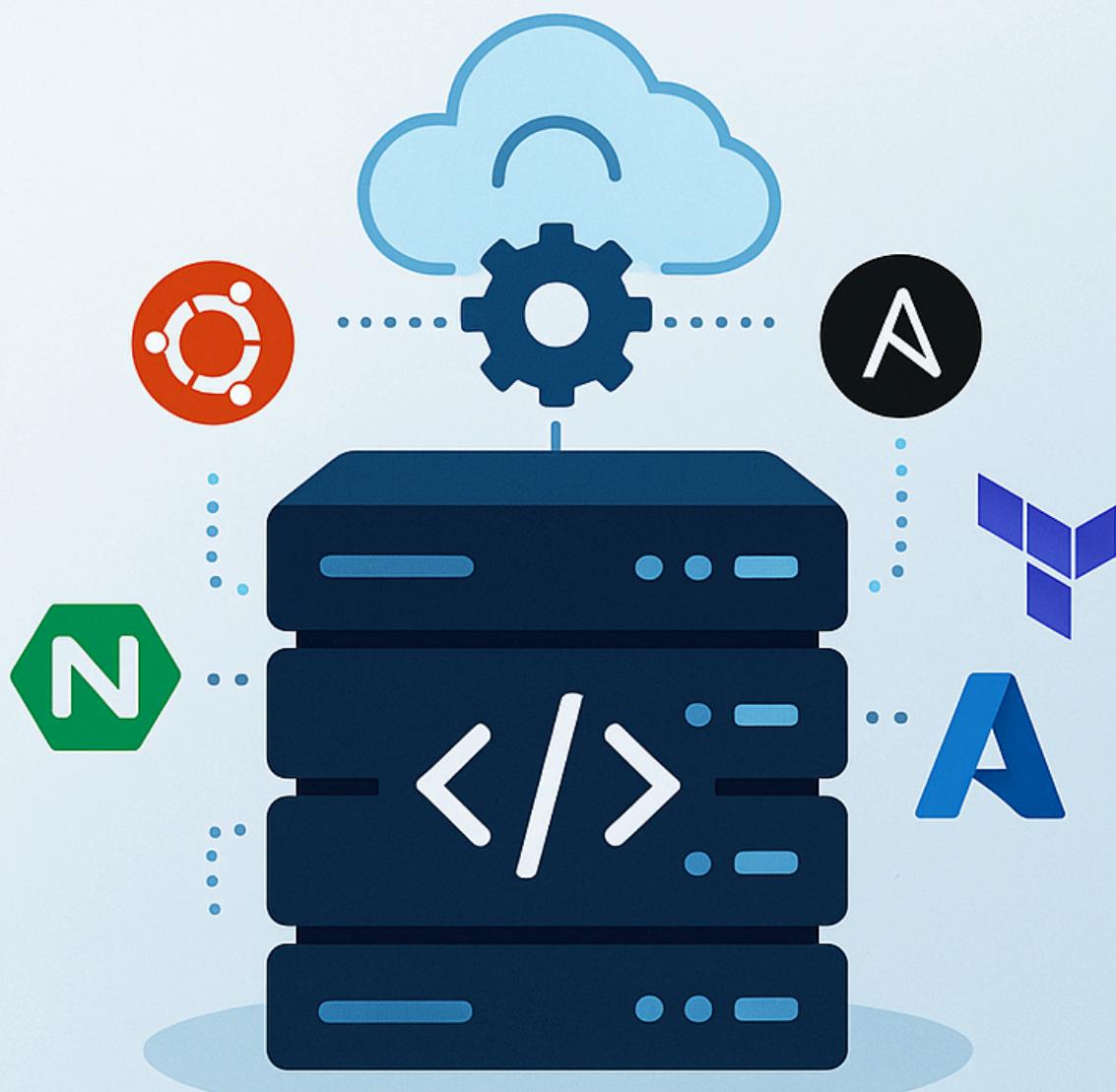


# Infraestructura como código: automatización y despliegue de un servidor web local y en la nube

Iñaki Arévalo Peña

Curso académico 2024-2025

Ciclo Superior ASIR



08/12/2025

# Índice

---

<b>Introducción .....</b>	<b>4</b>
<b>Contexto y Justificación (RA1) .....</b>	<b>5</b>
<b>Objetivos, Metodología y Recursos (RA2) .....</b>	<b>12</b>
<b>Implantación y ejecución del proyecto (RA3) .....</b>	<b>18</b>
<b>    1. Fase 1 - Ubuntu server, Nginx y configuraciones básicas .....</b>	<b>18</b>
1.1. Instalación Ubuntu Server en local .....	18
1.2. Preparación básica del servidor .....	19
1.2.1. Puesta a punto del entorno linux .....	19
1.2.2. Crear nuevo usuario y cambio de usuario principal .....	21
1.2.3. Configuración de firewall básico (UFW) .....	22
1.2.4. Instalar herramientas útiles.....	23
1.3. Nginx: Configuración del servidor web .....	23
1.3.1. Instalar Nginx: Comprobación del servidor web .....	23
1.3.2. Configuración básica de Nginx .....	25
1.3.2.1. Crear el archivo de configuración del sitio .....	26
1.3.2.2. Crear el enlace simbólico .....	27
1.3.2.3. Crear el archivo index.html .....	27
1.3.2.4. Creación de la página web en Nginx .....	28
<b>    2. Fase 2 - Configuración avanzada del servidor .....</b>	<b>30</b>
2.1. Usuarios, grupos y permisos .....	31
2.2. Configuración de acceso SSH con llaves públicas .....	34
2.3. Scripts .....	35
2.3.1. Script: Reiniciar Nginx .....	35
2.3.2. Script: Backup del sitio web y de configuración de Nginx .....	38
2.3.3. Script: Restaurar backup .....	40
2.4. Creación de otro sitio web .....	44
2.4.1. Configuración del segundo sitio web .....	44
2.4.2. Añadir alias en la VM cliente .....	45
2.5. Habilitar HTTPS con certificado autofirmado .....	46
2.6. Automatización parcial mediante CRON .....	50
2.6.1. Directorio de logs centralizado .....	50

2.6.2. Script Maestro .....	52
2.6.3. Automatización con CRON. Configuración del cron job .....	54
<b>3. Fase 3 - Automatización con Ansible .....</b>	<b>55</b>
3.1. Ventajas de Ansible .....	56
3.2. Instalación y configuración inicial de Ansible .....	57
3.2.1. Instalación en el host .....	57
3.2.2. Configuración del inventario e intercambio de llaves SSH .....	58
3.2.3. Prueba de conectividad .....	59
3.3. Estructura del proyecto Ansible .....	60
3.3.1. Archivos principales (ansible.cfg, hosts, playbooks/) .....	60
3.3.2. Organización de roles y tareas .....	62
3.4. Playbook principal: configuración automática del servidor .....	63
3.4.1. Rol entorno_basico .....	63
3.4.2. Rol usuarios .....	68
3.4.3. Rol nginx .....	73
3.4.4. Rol mantenimiento .....	80
3.5. Despliegue completo en una nueva máquina virtual con Ansible .....	85
3.5.1. Preparación de la nueva máquina virtual .....	85
3.5.2. Actualización del inventario de Ansible .....	86
3.5.3. Despliegue automático completo. Ejecución de playbooks .....	87
3.6. Verificación tras ejecución de playbooks .....	88
3.7. Conclusiones de la fase .....	92
3.7.1. Ventajas obtenidas .....	92
3.7.2. Preparación para la siguiente fase (Terraform y Azure) .....	93
<b>4. Fase 4 - Despliegue en la nube con Terraform y Azure .....</b>	<b>94</b>
4.1 Estructura y objetivos de la fase .....	95
4.2. Configuración del entorno de Terraform en el host de control .....	96
4.2.1 Creación de una cuenta Free Tier en Azure .....	96
4.2.2. Instalación de Terraform en el host de control .....	97
4.2.3. Configuración del proveedor de Azure y autenticación .....	98
4.2.4. Estructura del entorno de Terraform en el host de control .....	98
4.2.5. Validación del entorno de Terraform .....	99
4.3. Creación del archivo de configuración principal (main.tf) .....	100
4.3.1. Proveedor de Azure .....	101

4.3.2. Grupo de recursos .....	<b>101</b>
4.3.3. Red virtual y subred .....	<b>102</b>
4.3.4. IP Pública .....	<b>103</b>
4.3.5. Interfaz de red (NIC) .....	<b>103</b>
4.3.6. Grupo de seguridad de red (NSG) .....	<b>104</b>
4.3.7. Máquina virtual .....	<b>105</b>
4.4. Gestión de variables y salida de datos .....	<b>106</b>
4.4.1. Outputs.tf .....	<b>106</b>
4.5. Despliegue y validación de recursos con Terraform .....	<b>107</b>
4.6. Logros alcanzados con Terraform y Azure .....	<b>111</b>
4.7. Despliegue de configuración con Ansible sobre la VM de Azure .....	<b>112</b>
4.7.1. Configuración del inventario de Ansible .....	<b>112</b>
4.7.2. Ejecución de los playbooks .....	<b>112</b>
4.7.3. Comprobaciones finales .....	<b>113</b>
<b>Calidad y Evaluación del producto (RA4)</b> .....	<b>114</b>
<b>Conclusión del proyecto</b> .....	<b>120</b>
<b>Bibliografía y referencias</b> .....	<b>122</b>
<b>Anexos/Otros</b> .....	<b>124</b>

# Introducción

---

El proyecto trata la **infraestructura como código (IaC)** y la **automatización** como necesidades de gestionar entornos de TI de manera rápida, estable y segura. En muchas ocasiones, depender de procesos manuales para gestionar recursos, tanto en entornos **locales** como en plataformas **cloud computing** como **Azure**, puede no ser la opción más eficiente en cuanto a velocidad de despliegue y consistencia de la configuración.

El objetivo central del proyecto es demostrar la automatización completa de un servidor **Ubuntu**, incluyendo configuraciones avanzadas y gestión de servicios como un servidor web (**Nginx**) utilizando **Terraform** para el despliegue de máquinas virtuales en la nube y **Ansible** para la gestión de su configuración interna.

Este proyecto certifica que es posible desplegar un servidor idéntico en minutos en diferentes entornos (local y cloud)

El trabajo está organizado en cuatro bloques principales, estructurados en función de las diferentes partes a tratar: la **justificación empresarial**, los **objetivos y métodos**, la **ejecución técnica** del proyecto y la **evaluación**:

**Bloque 1: Contexto y justificación:** Contiene la propuesta de valor, el análisis DAFO y la definición del marco legal y financiero de la empresa.

**Bloque 2: Objetivos, metodología y recursos:** Se centra en los objetivos específicos del proyecto; IaC, automatización y metodología de desarrollo.

**Bloque 3: Implementación y ejecución del proyecto:** Es la parte más extensa del proyecto. Presentará todo el proceso desde la creación de un servidor local hasta la automatización y despliegue de un servidor idéntico en la nube.

**Bloque 4: Calidad y evaluación del producto:** Analizará la eficiencia del despliegue automatizado frente al manual, la calidad del proyecto, y las conclusiones finales sobre la viabilidad técnica del proyecto.

# Contexto y Justificación (RA1)

---

## Propuesta de valor

El proyecto consiste en la posibilidad de **automatizar entornos** locales o en la nube (**Ansible**) y desplegar máquinas virtuales (**Terraform**).

Está dirigido a empresas de tamaño medio que necesitan una gestión rápida y replicable de servidores sin depender de procedimientos manuales. La propuesta de valor se centra en agilizar el aprovisionamiento de la infraestructura, conseguir consistencia y seguridad de los servidores, y permitir la replicación del entorno en distintos ámbitos de forma automatizada, reduciendo errores humanos y tiempos de despliegue.

Aspecto	Propuesta
Actividad	<b>IaC y automatización</b> completa para gestionar el despliegue de servidores. El flujo de trabajo sería la combinación de Terraform y Ansible
¿A quién va dirigido?	Empresas de tamaño medio y PYMES, departamentos de TI y administradores de sistemas. Se dirige a aquellos que desean gestionar servidores de manera eficiente.
¿Por qué se venderá?	<b>Eficacia y velocidad:</b> Este método garantiza el aprovisionamiento de infraestructura en minutos, permitiendo la replicación de entornos idénticos y seguros de forma automatizada. La clave es agilizar el despliegue, reducir errores humanos y ahorrar tiempo.
Objetivos	<b>Consistencia</b> (garantiza la seguridad y configuración de servidores), <b>eficiencia</b> (reduce tiempos de despliegue y configuración) y <b>replicación</b> (permite hacer copias de manera rápida de entornos en distintos ámbitos)

El **entorno específico** serían los departamentos de TI y los administradores de sistemas, que necesitan gestionar sus servidores de manera centralizada y eficiente.

El **entorno general** serían los sectores que se basen en la tecnología de la información, caracterizados por la digitalización, el uso de recursos en la nube y de la automatización de procesos. La evolución tecnológica, convierte este tipo de soluciones en una necesidad.

### Análisis DAFO



#### Fortalezas:

- **Experiencia en IaC y automatización:** Dominio de las herramientas: Terraform para el despliegue de infraestructura en Azure. Ansible para la gestión de configuración.
- **Conocimiento en Azure:** Capacidad para integrar el despliegue en una plataforma de cloud que es líder en la industria. Aprovechamiento de sus servicios.
- **Eficiencia en despliegues:** La metodología implementada permite aprovisionar entornos completos en minutos, no en días, lo que es un punto de venta clave.
- **Consistencia garantizada:** Eliminar errores manuales, y asegurar servidores idénticos y seguros, de forma rápida y fiable.

### **Debilidades:**

- **Aprendizaje previo y dependencia del equipo:** Terraform y Ansible pueden necesitar de un aprendizaje inicial. Para garantizar el éxito hace falta experiencia y conocimiento del equipo en estas herramientas.
- **Falta de historial:** El proyecto es completamente funcional. Pero al ser un proyecto nuevo no existe un historial de casos de éxito que lo pueda abalar.
- **Recursos limitados:** Sin tener más capacidad económica o de un equipo humano más grande, escalar el proyecto rápidamente sería más difícil. El proyecto inicial podría no incluir un plan de mantenimiento “post-Implementación” para los clientes.

### **Oportunidades:**

- **Crecimiento del Cloud Computing:** La tendencia imparable de las empresas a migrar a la nube (Azure, AWS, GCP) aumenta la demanda de IaC y automatización.
- **Expansión geográfica:** El proyecto se podría enfocar en mercados o regiones con menor competencia en soluciones IaC altamente especializadas.
- **Demandas de eficiencia y reducción de costes:** Las empresas cada vez más buscan optimizar recursos y reducir gasto. Esto se consigue con la automatización. Todas las empresas quieren más por menos.
- **Seguridad:** El aumento de las amenazas a la seguridad de los sistemas y servidores crea en las empresas la necesidad de entornos predecibles y seguros.

### **Amenazas:**

- **Competencia:** Otras empresas ya ofrecen soluciones similares con más recursos.
- **Evolución de herramientas:** La constante evolución de Terraform, Ansible o Azure podría requerir una adaptación continua que el equipo debe gestionar.
- **Resistencia al cambio:** Departamentos de TI más tradicionales se resisten a abandonar procesos manuales o a invertir en nuevos métodos por miedo al cambio.

- **Cambios en precios de Azure:** Aumentos inesperados en los costes de Azure en el futuro, podrían hacer la solución menos atractiva para el cliente final.

## Forma jurídica

La forma jurídica elegida es la **sociedad de responsabilidad limitada (S.L.)**. Esta opción es la más adecuada para una empresa de servicios tecnológicos que necesita limitar la responsabilidad de sus fundadores al capital aportado.

## Detalles de la Constitución

Aspecto	Detalle
Forma jurídica	Sociedad de responsabilidad limitada (S.L)
Nº Socios	2 Socios fundadores
Aportación mínima	3000€ (Capital Social Mínimo)

## Estructura de capital social

Socio	Aportación	Distribución de capital
Socio 1 (fundador/director técnico)	1500€	50% de participaciones
Socio 2 (cofundador/director comercial)	1500€	50% de participaciones
Total capital social	3000€	100% de participaciones

**Participaciones sociales:** El capital se divide en 3.000 participaciones sociales, con un valor nominal de 1 € cada una.

**Suscripción y desembolso íntegro :** El 100% de las participaciones están abonadas en efectivo por los socios fundadores íntegramente en el momento de la constitución.

### **Trabajadores**

Para asegurar la escalabilidad del modelo de negocio, la calidad en la ejecución técnica de los proyectos y la efectividad comercial, la S.L. requiere una plantilla de **cuatro personas**.

Esta estructura se justifica en la necesidad de separar las funciones directivas y comerciales de las funciones técnicas y de ejecución, permitiendo un enfoque especializado en cada área. La plantilla se divide legalmente en **socios trabajadores y personal contratado**.

#### **Estructura de personal y régimen de cotización:**

Puesto	Régimen de cotización	Justificación de la función	Salario mensual
Socio 1	RETA (autónomo)	Dirección/líder ejecución técnica	1000€
Socio 2	RETA (autónomo)	Dirección comercial/estratégica.	1000€
Técnico Junior	Régimen general	Ejecución técnica	1400€
Comercial Junior	Régimen general	Captación y soporte al Socio 2	1200€

**Gasto laboral total fijo:** El coste fijo de personal sería de 6.500 € mensuales (incluyendo salarios netos y todas las cotizaciones a la Seguridad Social).

### **Obligaciones**

La S.L. debe cumplir con las siguientes obligaciones laborales:

#### **Contratos y cotizaciones (régimen general):**

- Los dos empleados serán contratados con **contrato indefinido** a tiempo completo.
- La S.L. tiene la obligación de ingresar las cuotas correspondientes a la Seguridad Social (empresa y trabajador). Supondría un gasto estimado de **1.300 € mensuales** a cargo de la sociedad. Este gasto se integra en las **salidas** del plan de tesorería.

#### **Cotizaciones de socios (RETA):**

Los Socios, al ejercer funciones de control y dirección, deben cotizar en el **Régimen Especial de Trabajadores Autónomos (RETA)**, lo que implica un gasto total de **600 € mensuales** para la sociedad que se integra en las **salidas** del plan de tesorería.

#### **Prevención de Riesgos Laborales (PRL)**

Dada la actividad principal de la empresa (desarrollo tecnológico), la PRL se enfoca en riesgos asociados al trabajo de oficina y en exigencias en relación al sector.

La S.L. utilizará un **servicio de prevención ajeno (externo)**. Esta es la opción más habitual para PYMEs y asegura el cumplimiento normativo sin sobrecargar la estructura interna.

#### **Riesgos laborales y medidas Preventivas:**

Riesgos laborales	Consecuencia	Medida preventiva
<b>Ergonómicos</b>	Uso intensivo y prolongado de pantallas.	Suministro de mobiliario ergonómico certificado y formación en higiene postural.
<b>Psicosociales</b>	Estrés por gestión de proyectos y plazos ajustados.	Implementación de plan de desconexión digital y fomento de pausas activas.
<b>Seguridad</b>	Riesgos básicos de oficina (incendios, eléctricos).	Instalación de extintores homologados e impartición de planes de evacuación.

## Plan de tesorería

Concepto	E.	F.	M.	A.	M.	J.	J.	A.	S.	O.	N.	D.	TOTAL
ENTRADAS													
Aportaciones socios	3.000	0	0	0	0	0	0	0	0	0	0	0	3.000
Préstamo bancario	15.000	0	0	0	0	0	0	0	0	0	0	0	15.000
Subvenciones	3.000	0	0	0	0	0	0	0	0	0	0	0	3.000
Ventas	0	3.000	7.000	9.000	11.000	12.000	13.000	13.000	14.000	15.000	15.000	16.000	128.000
TOTAL ENTRADAS	21.000	3.000	7.000	9.000	11.000	12.000	13.000	13.000	14.000	15.000	15.000	16.000	149.000
SALIDAS													
Gastos de constitución (S.L.)	600	0	0	0	0	0	0	0	0	0	0	0	600
Salarios netos (4 personas)	4.600	4.600	4.600	4.600	4.600	4.600	4.600	4.600	4.600	4.600	4.600	4.600	55.200
Seg. Social (SS/RETA)	1.900	1.900	1.900	1.900	1.900	1.900	1.900	1.900	1.900	1.900	1.900	1.900	22.800
Alquiler/Cloud/Oficina	1.500	1.500	1.500	1.500	1.500	1.500	1.500	1.500	1.500	1.500	1.500	1.500	18.000
Equipos/Licencias	3.000	0	0	0	0	0	0	0	0	0	0	0	3.000
Devolución préstamo	400	400	400	400	400	400	400	400	400	400	400	400	4.800
Suministros/Marketing/Otros	500	500	500	500	500	500	500	500	500	500	500	500	6.000
TOTAL SALIDAS	12.500	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	8.900	112.500
FLUJO NETO (Entradas - Salidas)	8.500	-5.900	-1.900	100	2.100	3.100	4.100	4.100	5.100	6.100	6.100	7.100	38.500
SALDO ACUMULADO	8.500	2.600	700	800	2.900	6.000	10.100	14.200	19.300	25.400	31.500	38.600	38.600

El plan de tesorería demuestra que el proyecto es viable a corto plazo (1 año) y posee la liquidez necesaria para sostener una estructura de 4 personas.

## Viabilidad y liquidez

- **Financiación inicial:** La inyección de 21.000 € cubre todos los gastos.
- **Punto de equilibrio:** El coste fijo mensual es de 8.900 € (incluyendo 6.500 € de gastos de personal). El flujo de caja neto se vuelve positivo a partir de Abril.
- **Riesgo controlado:** El saldo acumulado nunca cae a niveles negativos.
- **Resultado final:** El año concluye con un saldo acumulado de **38.600 €**, confirmando la solidez financiera y la capacidad de generar beneficios.

## Gestión de la deuda

- **Deuda total:** El préstamo inicial fue de 15.000 €.
- **Pagado en el primer año:** La empresa paga 4.800 € durante el primer año.
- **Deuda pendiente:** Al inicio del segundo año, la deuda se reduce a 10.200 €.
- **Capacidad de pago:** La alta liquidez final de 38.600 € garantiza que la empresa puede afrontar fácilmente los pagos restantes del préstamo en los años posteriores.

# Objetivos, Metodología y Recursos (RA2)

---

## Objetivos

### Objetivo general

Diseñar, automatizar y desplegar un entorno de infraestructura reproducible basado en servidores Linux, utilizando tecnologías estándar del sector (Ansible, Terraform y Azure), con el fin de demostrar un proceso completo de automatización IT y despliegue que pueda ser aplicado en entornos empresariales reales.

### Objetivos específicos

- **Automatizar la configuración inicial de servidores Linux**, incluyendo gestión de usuarios, permisos, firewall, servicios y alguna otra configuración avanzada.
- **Instalar y configurar un servidor web Nginx** completamente automatizado a través de roles y playbooks reproducibles.
- **Diseñar un entorno de scripts de mantenimiento programado**, que permita ejecutar tareas mediante scripts y de forma periódica con CRON.
- **Construir una infraestructura cloud mínima funcional** Azure mediante Terraform, incluyendo grupo de recursos, red virtual, subred, NSG, IP pública, NIC y VM.
- **Integrar Terraform y Ansible**, de forma que la máquina virtual desplegada automáticamente sea configurada mediante los playbooks previos.
- **Aplicar buenas prácticas de IaC** para garantizar coherencia y trazabilidad.
- **Demostrar que todo el proceso puede ejecutarse sin costes**, o con un coste mínimo. Utilizando únicamente recursos incluidos dentro del Azure Free Tier.

## Metodología

El desarrollo del proyecto se basa en una **metodología práctica DevOps**, centrada en reproducibilidad y automatización:

## Enfoque técnico

- **Infraestructura como código:** Describir y configurar todos los recursos de infraestructura mediante archivos de código.
- **Automatización de configuración:** Implementar tareas repetibles en los servidores.
- **Despliegue cloud reproducible:** Se combina Terraform para el aprovisionamiento y Ansible para la configuración.

## Ciclo de trabajo aplicado

La siguiente tabla describe el proceso para configurar y desplegar una infraestructura híbrida, comenzando con la configuración manual en un entorno local (VirtualBox/Ubuntu), seguido de la automatización con Ansible y finalizando con el despliegue en Azure usando Terraform:

<b>Preparación y validación local</b>	<b>Preparación del entorno local:</b> Despliegue inicial de una VM con Ubuntu Server utilizando VirtualBox.
	<b>Configuración inicial del servidor:</b> Realización manual de las configuraciones.
	<b>Configuración de servicios:</b> Configuración del servidor web Nginx y otras configuraciones avanzadas.
<b>Automatización de la configuración</b>	<b>Creación de playbooks y roles (Ansible):</b> Codificar el proceso en roles y playbooks de Ansible.
	<b>Aplicación y validación local (Ansible):</b> Ejecución de los playbooks de Ansible sobre una nueva VM.
<b>Despliegue y validación híbrida (IaC)</b>	<b>Desarrollo del código (Terraform):</b> Creación de los scripts de Terraform para definir el entorno en Azure.
	<b>Despliegue híbrido (Terraform + Ansible):</b> Desplegar la infraestructura en Azure y aplicar los playbooks.
	<b>Validación final:</b> Pruebas de acceso en ambos entornos (local y Azure) y verificación del proceso.

## Recursos necesarios

### Recursos materiales

- **PC principal (host):** Equipo físico donde se levantarán las máquinas virtuales y se realizará todo el proceso de pruebas y despliegue. (Procesador: CPU Intel Core i7-8700 K de 6 núcleos / 12 hilos, memoria RAM: 32 GB, almacenamiento: SSD 500 GB y HDD 1TB y sistema operativo base: Windows 10)
- **Conexión a Internet:** Requisito esencial para la instalación de paquetes, actualizaciones, certificados SSL y despliegue en la nube.
- **Cuenta gratuita Azure Free Tier:** Para poder levantar máquinas en la nube.

### Recursos software

- **VirtualBox** (levantar VM's)
- **Ubuntu 24.04** (sistema operativo principal)
- **Ansible** (automatización de configuración)
- **Terraform** (gestión de infraestructura cloud)
- **Azure CLI** (autenticación con *az login*)
- **OpenSSH** (claves públicas/privadas para acceso seguro)
- **Editor de código:** Visual Studio Code
- **Nginx** (servidor web desplegado)

### Recursos humanos

El proyecto está diseñado para que la parte técnica puede ser desarrollada por una única persona, cubriendo todos los roles:

**Analista:** Fase de planificación y diseño. Tareas que incluyen la definición de requisitos técnicos y la creación de la arquitectura híbrida del despliegue.

**Especialista en configuración y seguridad:** Se enfoca en la capa del sistema operativo. Es el encargado de realizar las pruebas manuales, la instalación y configuración avanzada del servidor Nginx y de codificar dichas configuraciones en los playbooks de Ansible.

**Ingeniero IaC / automatización:** Es el rol responsable de la integración y el despliegue. Sus tareas se centran en el desarrollo de los scripts de Terraform, en la integración de Ansible, los entornos de Azure y local, y la validación técnica del despliegue.

### **Viability técnica del proyecto**

En cuanto a la **justificación técnica** cabe decir que todas las tecnologías utilizadas son:

- Gratuitas
- Open source
- Estándar en la industria

En cuanto a los **riesgos y mitigación**:

Riesgo	Mitigación
Consumo excesivo del crédito Azure	Uso de tamaños gratuitos y apagado manual de VM cuando no se utilice.
Error en playbooks	Uso de --check y ejecución repetible.
Fallos de SSH	Regeneración de claves
Dependencia del cloud	Documentación clara para reproducir el trabajo en local.

### **Conclusión de viabilidad:**

El proyecto demuestra que es **viable** tanto a nivel técnico como económico, ajustándose a lo que se espera en un perfil de administración de sistemas avanzado. Técnicamente la infraestructura como código (IaC) es viable para resolver estas necesidades, logrando la automatización completa del despliegue de un servidor web Nginx sobre Ubuntu 24.04 en entornos híbridos (Azure y local) mediante la integración efectiva de Terraform y Ansible .

Esta metodología no solo garantiza un proyecto consistente y reproducible, sino que reduce el tiempo de aprovisionamiento significativamente.

Las **tecnologías** son gratuitas, open source y estándares en la industria. Por lo que hay documentación de sobra y son soportadas por cualquier sistema.

Por otro lado, los **riesgos** que puedan ocurrir en el proyecto se pueden mitigar de forma consistente.

Económicamente, el **plan de negocio** es sólido: la financiación inicial de 21.000€ soporta todas las salidas y acaba cerrando el ejercicio con una liquidez de 38.600€.

El diseño es modular, lo que me permite añadir o cambiar servicios, herramientas o configuraciones según vaya avanzando el proyecto. Esto asegura que la infraestructura pueda crecer y adaptarse sin romper lo que ya está hecho.

En resumen, el proyecto es totalmente factible con los recursos y el tiempo que tengo, y me permite alcanzar los objetivos de aprendizaje y desarrollo de la infraestructura, dejando además margen para ampliaciones o mejoras futuras.

## **Fases del proyecto y temporización**

El proyecto se desarrollará siguiendo un enfoque por fases, combinando la creación de la infraestructura local con la automatización progresiva y su despliegue en la nube.

A continuación detallo la planificación temporal y las tareas principales de cada fase:

Semana	Fase	Actividades
1-2	<b>Fase 1: Infraestructura local básica</b>	Instalación de Ubuntu Server
		Configuración de Nginx
		Sitio Web (HTML/CSS)
		Permisos y accesos
3-4	<b>Fase 2: Preparación para automatización</b>	Usuarios, grupos y permisos
		Acceso SSH seguro
		Scripts de mantenimiento

		Segundo sitio web
		Habilitación HTTPS
		Automatización parcial con CRON
5-6	<b>Fase 3: Automatización con Ansible</b>	Creación de playbooks y roles
		Despliegue automático del servidor
		Pruebas y ajustes de configuración
7-9	<b>Fase 4: Despliegue en la nube con Terraform y Azure</b>	Definición de IaC con Terraform
		Configuración con Azure
		Replicación de infraestructura en la nube
		Verificación de escalabilidad y funcionalidad
10	<b>Fase 5: Documentación y pruebas finales</b>	Redacción de documentación técnica
		Registro de incidencias y soluciones
		Presentación del proyecto

# Implantación y ejecución del proyecto (RA3)

## 1. Fase 1 - Ubuntu server, Nginx y configuraciones básicas

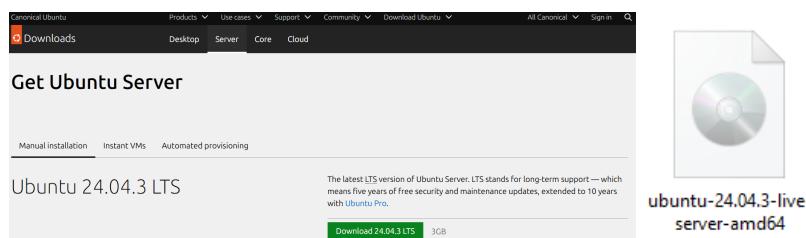
### 1.1. Instalación Ubuntu Server en local

En primer lugar, he instalado el software **VirtualBox** de Oracle.

Es un software que permite crear máquinas virtuales (VMs) sobre mi sistema operativo anfitrión, donde cada VM funciona como un sistema independiente. Lo usaré para crear una VM Ubuntu **Server 22.04**, instalar **Nginx** y practicar configuración de usuarios, firewall y redes sin afectar a mi PC real. Esto me permite probar servicios, automatización con **Ansible** o **Terraform**, simulando un entorno de producción, antes de desplegar en **Azure**.

#### Descargar Ubuntu Server

Desde el link <https://ubuntu.com/download/server> voy a descargar la ISO de **Ubuntu Server 22.04 LTS**. Guardaré el archivo en mi PC y lo usaré para crear mi máquina virtual.

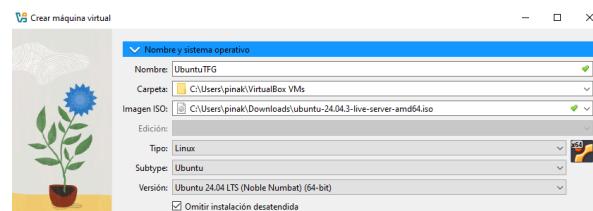


#### Crear mi primera máquina virtual

En primer lugar, abro VirtualBox y hago clic en “**Nueva**” y le pongo el nombre de: **UbuntuTFG**. Tipo (LINUX) Subtype (Ubuntu) Versión (Ubuntu 24.04 Noble Numbat 64-bit).

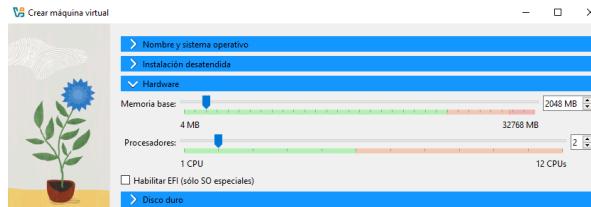
Imagen ISO:

La misma que acabo de descargar.



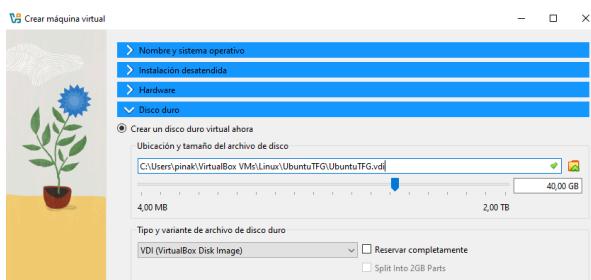
En cuanto a memoria:

- **RAM:** 2048 MB (2 GB)
- **CPUs:** 2 núcleos lógicos



En cuanto a Disco duro:

- **20 GB** tipo VDI, dinámico



Ya tengo las especificaciones de nuestra máquina asignadas. Ahora, Arranco la VM.



### - Instalar Ubuntu Server en la VM

Hay que seguir los pasos de instalación y una vez haya terminado, reinicio:

```
Instalando el sistema [ Help ]  
configuring lvm_vggroup-0  
configuring lvm_partition: lvm_partition-0  
configuring mount: mount-0  
configuring mount: mount-1  
executing /usr/share/vmware-installer/extrac_step  
curlin command: install  
writing install sources to disk  
curlin command: extract  
curlin command: extracting image from cpi:///tmp/tmp189048/m/mount  
configuring keyboard  
curlin command: in-target  
executing /usr/share/vmware-installer/carthooks_step  
curlin command: install  
configuring install system  
configuring carthooks  
curlin command: carthooks  
curlin command: in-target  
installing missing apt  
installing missing packages  
Installing packages on target system: ['grub-pc']  
configuring raid (mdadm) service  
configuring network over TCP  
installing kernel  
setting up swap  
setting up swap  
writing etcfstab  
configuring multilib  
installing packages on target system  
configuring pollinate user-agent on target  
configuring target system  
configuring target system bootloader  
installing grub to target devices  
configuring target system rootfs  
final system configuration  
calculating extra packages to install  
installing packages  
retrieving openssh-server  
locating openssh-server  
unpacking openssh-server  
curlin command: system-install -
```

## 1.2. Preparación básica del servidor

### 1.2.1. Puesta a punto del entorno Linux

En este punto, voy a preparar la máquina virtual Ubuntu Server para la instalación de servicios, asegurando que el sistema esté actualizado, seguro y correctamente configurado. Esto incluye: actualización de paquetes, configuración de zona horaria, nombre del servidor, red, firewall y utilidades básicas.

En primer lugar, hago el login con el usuario creado en la instalación:

```
Ubuntu 24.04.3 LTS webserver01 tty1
webserver01 login: inaki
Password:
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-83-generic x86_64)
```

Una vez iniciado el servidor, compruebo versión:

```
inaki@webserver01:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.3 LTS
Release:        24.04
Codename:       noble
```

#### - Actualización del Sistema:

Realizaré una actualización de la lista de paquetes y aplicaré las actualizaciones disponibles.

```
inaki@webserver01:~$ sudo apt update && sudo apt upgrade -y
```

#### - Configuración de zona horaria:

Voy a ajustar la hora del servidor a la zona horaria local. Es importante para la correcta gestión de logs y horarios de servicios.

```
inaki@webserver01:~$ sudo timedatectl set-timezone Europe/Madrid
inaki@webserver01:~$ timedatectl
          Local time: mié 2025-09-17 12:51:50 CEST
              Universal time: mié 2025-09-17 10:51:50 UTC
                  RTC time: mié 2025-09-17 10:51:51
                    Time zone: Europe/Madrid (CEST, +0200)
System clock synchronized: yes
          NTP service: active
    RTC in local TZ: no
```

#### - Configuración del nombre del servidor (hostname)

```
inaki@webserver01:~$ sudo hostnamectl set-hostname servidor01
[sudo] password for inaki:
inaki@webserver01:~$ hostname
servidor01
```

#### - Verificación de red y conectividad

Con **ip a** vemos las interfaces de red y IP

```
inaki@servidor01:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:36:03:d1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 86334sec preferred_lft 86334sec
    inet6 fd00::a00:27ff:fe36:3d1/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 86337sec preferred_lft 14337sec
    inet6 fe80::a00:27ff:fe36:3d1/64 scope link
        valid_lft forever preferred_lft forever
```

Comprobar la conexión a internet y garantizar que la VM puede conectarse a Internet para instalar paquetes y recibir actualizaciones.:

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=5.04 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=4.39 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=4.50 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=255 time=4.99 ms
```

Comprobar que hace resolución DNS:

```
sysadmin01@servidor01:~$ ping google.com
PING google.com (212.166.161.5) 56(84) bytes of data.
64 bytes from 212-166-161-5.red-acceso.airtel.net (212.166.161.5): icmp_seq=1 ttl=255 time=4.60 ms
64 bytes from 212-166-161-5.red-acceso.airtel.net (212.166.161.5): icmp_seq=2 ttl=255 time=4.47 ms
64 bytes from 212-166-161-5.red-acceso.airtel.net (212.166.161.5): icmp_seq=3 ttl=255 time=5.01 ms
64 bytes from 212-166-161-5.red-acceso.airtel.net (212.166.161.5): icmp_seq=4 ttl=255 time=5.47 ms
```

### 1.2.2. Crear nuevo usuario y cambio de usuario principal

Para el servidor definimos dos usuarios administrativos siguiendo buenas prácticas de Linux:

```
inaki@servidor01:~$ sudo adduser sysadmin02
```

Añadimos a este usuario al grupo sudo para evitar el uso directo de root y mejorar la seguridad:

```
inaki@servidor01:~$ sudo usermod -aG sudo sysadmin02
```

Iniciaré sesión con **sysadmin02** para cambiar el nombre del usuario principal (de **inaki** a **sysadmin01**).

- Modifico la cuenta de usuario con sysadmin01 como nuevo nombre de login.

```
sysadmin02@servidor01:~$ sudo usermod -l sysadmin01 inaki
```

- Cambio el directorio home

```
sysadmin02@servidor01:~$ sudo usermod -d /home/sysadmin01 -m sysadmin01
```

- Voy a renombrar el grupo principal para mantener la coherencia entre nombre de usuario y grupo principal:

```
sysadmin02@servidor01:~$ sudo groupmod -n sysadmin01 inaki
```

### 1.2.3. Configuración de firewall básico (UFW)

Para proteger el servidor Linux bloqueando accesos no deseados y permitiendo solo los servicios necesarios (SSH, HTTP/HTTPS) antes de instalar Nginx y otros servicios

Primero lo tengo inactivo:

```
sysadmin01@servidor01:~$ sudo ufw status  
Status: inactive
```

Voy a permitir conexiones SSH para administración remota (puerto 22):

```
sysadmin01@servidor01:~$ sudo ufw allow ssh  
Rules updated  
Rules updated (v6)
```

Permito http para tráfico web no seguro (puerto 80) y https tráfico web seguro (puerto 443):

```
sysadmin01@servidor01:~$ sudo ufw allow http  
Rules updated  
Rules updated (v6)  
sysadmin01@servidor01:~$ sudo ufw allow https  
Rules updated  
Rules updated (v6)
```

Activo UFW y compruebo estado:

To	Action	From
--	---	---
22/tcp	ALLOW IN	Anywhere
80/tcp	ALLOW IN	Anywhere
443	ALLOW IN	Anywhere
22/tcp (v6)	ALLOW IN	Anywhere (v6)
80/tcp (v6)	ALLOW IN	Anywhere (v6)
443 (v6)	ALLOW IN	Anywhere (v6)

#### 1.2.4. Instalar herramientas útiles

Vamos a instalar herramientas como **curl** (Probar URLs, descargar archivos desde terminal), **wget** (Descargas de archivos), **vim / nano** (Editores de texto), **htop** (Monitorizar procesos de forma interactiva), **tree** (ver la estructura de directorios en forma de árbol) y **net-tools / iputils-ping** (herramientas de red (ifconfig, ping)):

```
sysadmin01@servidor01:~$ sudo apt install -y curl wget vim htop tree net-tools iputils-ping
```

### 1.3. Nginx: Configuración del servidor web

#### 1.3.1. Instalar Nginx: Comprobación del servidor web

Antes de instalar el servidor web actualizamos la lista de paquetes del sistema:

```
sysadmin01@servidor01:~$ sudo apt update
```

Posteriormente instalaremos Nginx:

```
sysadmin01@servidor01:~$ sudo apt install -y nginx
```

- **Comprobar estado:**

Una vez hecha la instalación vamos a comprobar si está Nginx está corriendo:

```
sysadmin01@servidor01:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
  Active: active (running) since Wed 2025-09-17 18:36:47 CEST; 1min 22s ago
    Docs: man:nginx(8)
 Process: 1538 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
 Process: 1539 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
 Main PID: 1570 (nginx)
   Tasks: 3 (limit: 2267)
   Memory: 2.4M (peak: 5.3M)
     CPU: 19ms
    CGroup: /system.slice/nginx.service
            ├─1570 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
            ├─1572 "nginx: worker process"
            ├─1573 "nginx: worker process"

sep 17 18:36:47 servidor01 systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server
sep 17 18:36:47 servidor01 systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.
```

Para poder hacer una comprobación de que Nginx está activo:

- Accedemos al archivo de configuración de red para editarlo:

```
sysadmin01@servidor01:~$ sudo nano /etc/netplan/50-cloud-init.yaml
```

- He configurado el archivo de configuración de red para que la interfaz use una IP estática en lugar de DHCP, asignando como puerta de enlace la del router y los servidores DNS públicos 8.8.8.8 y 8.8.4.4.

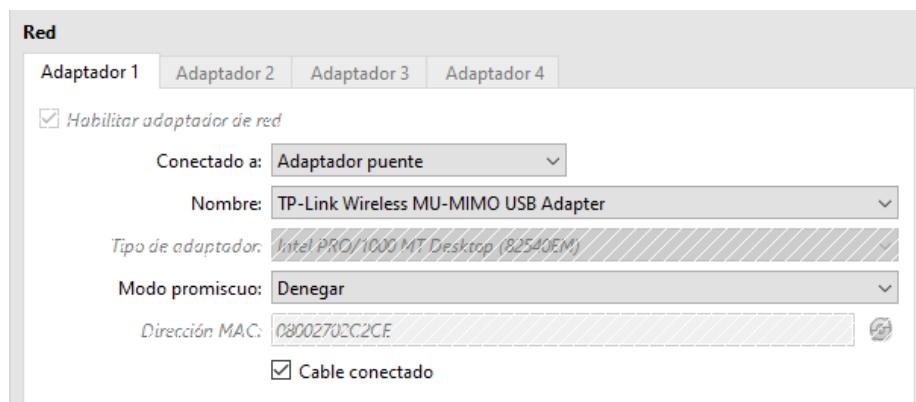
```
GNU nano 7.2
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.0.2/24
      gateway4: 192.168.0.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

- Para aplicar esta configuración de red:

```
sysadmin01@servidor01:~$ sudo netplan apply
```

Para la comprobación final, encenderemos una máquina virtual **host** (VM con un **Linux Mint**) y configuraremos la red de manera que ambas máquinas estén en la misma red, permitiendo que se vean y se comuniquen entre sí. Para ello:

- He cambiado la configuración de red de VirtualBox en ambas máquinas de NAT a **Adaptador Puente**, para que las máquinas se vean entre sí, pero también puedan acceder a la red local y a Internet.



- Ahora enciendo mi máquina virtual host y cambiamos la configuración de la red para que comparta red con el servidor:



- Hago ping entre ambas máquinas para ver si se ven entre sí:

```
sysadmin@servidor01:~$ ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=0.699 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=0.323 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=64 time=0.435 ms
64 bytes from 192.168.0.3: icmp_seq=4 ttl=64 time=0.359 ms
```

```
inaki@inaki-VirtualBox:~$ ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.381 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.281 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.359 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.473 ms
```

- Para acabar vamos al navegador de nuestra VM Host y en el buscador escribimos la IP del servidor. Ahora puedo ver la página por defecto de Nginx para la comprobación final de que el servidor web está corriendo:



### 1.3.2. Configuración básica de Nginx

Antes de configurar Nginx para servir mi propio sitio, necesito un directorio en el servidor donde alojar los archivos de la web (HTML, CSS, imágenes...).

En Linux, los sitios web se guardan dentro de la carpeta **/var/www**.

He creado y preparado la carpeta **/var/www/infra-cloud** con los siguientes pasos:

1. Crear el directorio donde irá mi web y todos los archivos de la misma:

```
sysadmin01@servidor01:~$ sudo mkdir -p /var/www/infra-cloud  
[sudo] password for sysadmin01:
```

2. Cambiar el propietario del directorio para que el usuario de trabajo (**sysadmin01**) puede editar los archivos sin necesidad de usar **sudo** constantemente:

```
sysadmin01@servidor01:~$ sudo chown -R sysadmin01:sysadmin01 /var/www/infra-cloud/
```

3. Cambiar los permisos del directorio, que permite que Nginx (y cualquier cliente web) pueda leer y ejecutar los archivos, manteniendo la seguridad. Y que el propietario pueda leer, escribir y ejecutar.

```
sysadmin01@servidor01:~$ sudo chmod -R 755 /var/www/infra-cloud/
```

Con esto, el servidor queda listo para alojar el contenido que servirá Nginx.

```
sysadmin01@servidor01:/var/www/infra-cloud$ ls -ld  
drwxr-xr-x 2 sysadmin01 sysadmin01 4096 sep 17 19:14
```

### 1.3.2.1. Crear el archivo de configuración del sitio

Nginx guarda las configuraciones de cada “sitio” en la carpeta **sites-available**.

Cada archivo describe cómo debe responder Nginx cuando alguien visita una IP o dominio.

Creo el archivo para nuestro sitio: infra-cloud para que Nginx sepa que debe servir nuestro sitio, no la página de prueba.

```
sysadmin01@servidor01:~$ sudo nano /etc/nginx/sites-available/infra-cloud
```

Dentro del archivo escribimos:

```
GNU nano 7.2                                         /etc/nginx/sites-available/infra-cloud *
```

```
server {  
    listen 80;                                     # Puerto donde escucha (HTTP)  
    server_name _;                                # Acepta cualquier nombre o IP  
    root /var/www/infra-cloud;                      # Carpeta donde están los archivos de la web  
    index index.html;                             # Archivo principal que se mostrara  
    location / {  
        try_files $uri $uri/ =404;      # Si el archivo no existe devuelve error 404  
    }  
}
```

**listen 80**: define el puerto de escucha para peticiones web normales.

**server\_name** : el guión bajo indica “cualquier nombre o IP”

**root**: la ruta real de los archivos de la web.

**index**: el archivo que Nginx mostrará por defecto (p. ej. index.html).

**location**: indica cómo manejar las peticiones; **try\_files** busca el archivo solicitado y, si no lo encuentra, devuelve un 404.

### 1.3.2.2. Crear el enlace simbólico

Para que Nginx cargue mi sitio, necesito activar la configuración con un enlace simbólico:

```
sudo ln -s /etc/nginx/sites-available/infra-cloud /etc/nginx/sites-enabled/
```

Esto hace que Nginx “vea” mi archivo **infra-cloud** y lo cargue al iniciar o recarga:

Para comprobar la sintaxis, ejecuto el comando **nginx -t** y comprobamos que nos aparece el mensaje **syntax is ok** y **test is successful**:

```
sysadmin01@servidor01:~$ sudo nginx -t
2025/09/23 13:29:20 [warn] 1175#1175: conflicting server name "_" on 0.0.0.0:80, ignored
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Ahora recargamos nginx para aplicar los cambios sin reiniciar el servidor:

```
sudo systemctl reload nginx
```

### 1.3.2.3. Crear el archivo index.html

Antes de configurar Nginx para servir nuestro sitio, necesito un archivo principal que actúe como página de inicio. En HTML, este archivo por defecto se llama **index.html**. Nginx busca automáticamente este archivo al acceder a un sitio web, así cuando un cliente visita la IP del servidor, Nginx sirve este archivo como primer contenido visible.

Para ello voy a crear el archivo donde he alojado la web y escribo un contenido básico de prueba:

```
sudo nano /var/www/infra-cloud/index.html
```

```
GNU nano 7.2                                     /var/www/infra-cloud/index.html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Infra-Cloud</title>
</head>
<body>
    <h1>Infraestructura en la nube con automatización</h1>
    <p>Iñaki Arévalo Peña</p>
</body>
</html>
```

Para hacer la comprobación, antes me aseguro de que el propietario y permisos del archivo permiten que Nginx lo lea:

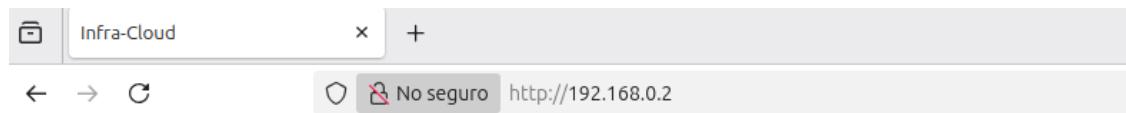
```
sudo chown sysadmin01:sysadmin01 /var/www/infra-cloud/index.html
sudo chmod 644 /var/www/infra-cloud/index.html
```

Y para acabar voy a desactivar los archivos de prueba (**default**) que están alojados en los directorios tanto **sites-enabled** como **sites-available**, para que no entre en conflicto con mi archivo **infra-cloud**:

```
sudo rm /etc/nginx/sites-enabled/default
```

```
sudo rm /etc/nginx/sites-available/default
```

Después de recargar nginx, abro mi máquina virtual host y accedo mediante el buscador de mi navegador a la IP de mi servidor. Ahora aparece mi página de prueba:



## Infraestructura en la nube con automatización

Iñaki Arévalo Peña

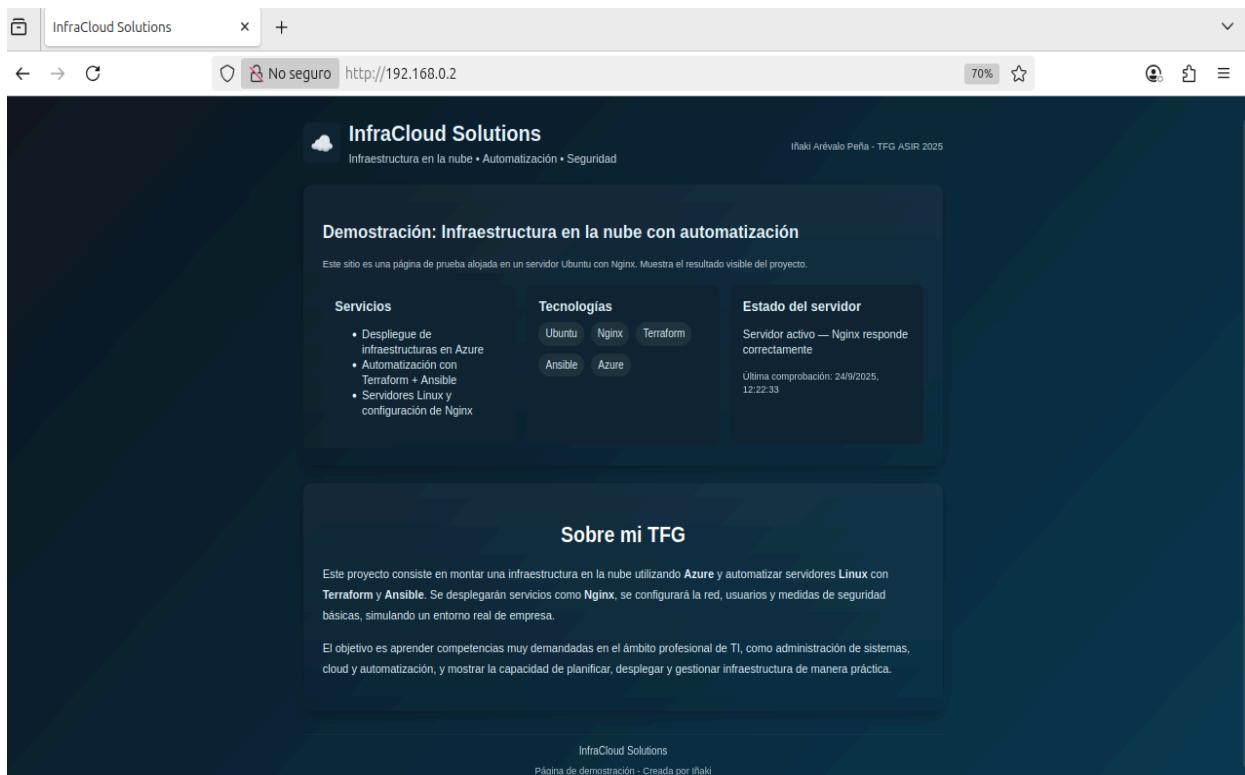
### 1.3.2.4. Creación de la página web en Nginx

Para poder probar y visualizar los servicios de mi infraestructura, he creado una página web de demostración que sirve como **index.html** de nuestro servidor Nginx.

Los archivos se encuentran dentro del directorio `/var/www/infra-cloud`, preparado previamente con permisos adecuados para el usuario de trabajo (sysadmin01) y lectura por parte de Nginx.

- Archivo ***index.html***: contiene la estructura básica de la página, incluyendo títulos, párrafos y secciones que describen el proyecto. Se han utilizado etiquetas HTML semánticas para mantener la organización y accesibilidad del contenido.
- Archivo ***style.css***: define los estilos de la página, como colores de fondo y texto, tipografía, márgenes y paddings. Se ha mantenido consistencia con el resto de la web para que todas las secciones tengan un diseño uniforme y profesional.

Con esta configuración, Nginx sirve la página correctamente y permite que cualquier cliente de la red acceda al contenido introduciendo la dirección IP del servidor. Esta página tiene principalmente función demostrativa, mostrando que el servidor web está operativo y listo para alojar servicios más complejos o integrarse con aplicaciones de infraestructura automatizada en fases posteriores.



## **2. Fase 2 - Configuración avanzada del servidor**

Antes de iniciar la fase de automatización con Ansible, es importante identificar qué tareas de administración y configuración del servidor se pueden automatizar de momento.

En esta etapa, lo que he realizado de manera manual (instalación de Ubuntu Server, configuración de red, creación de usuarios, instalación de Nginx, despliegue de la página web de prueba y configuración básica de firewall) se convierte en el punto de partida para que Ansible pueda replicar estas acciones de manera rápida, segura y repetible en múltiples servidores.

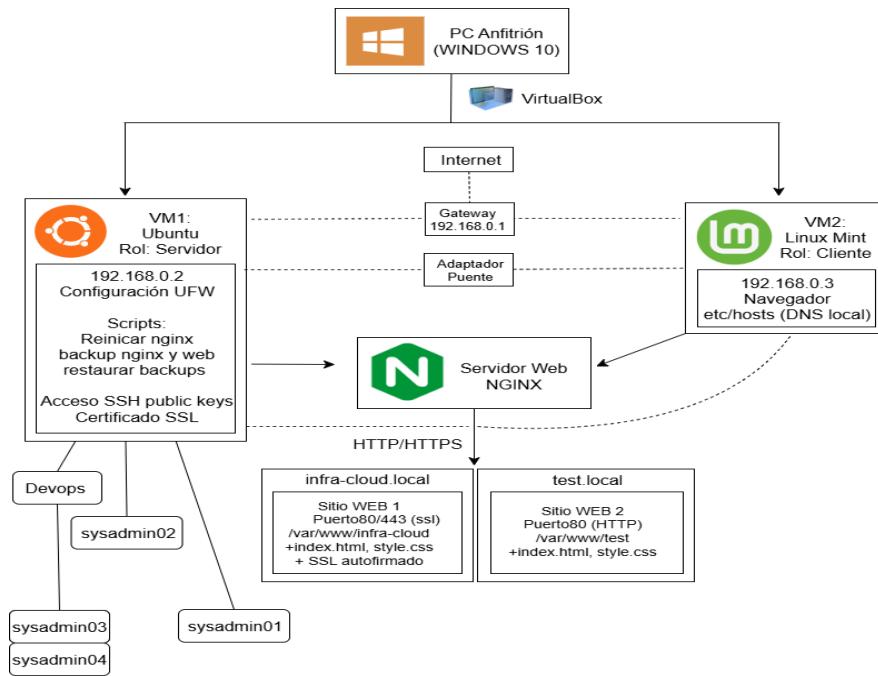
De momento, las tareas que podríamos automatizar con Ansible incluyen:

- **Gestión de usuarios:** creación de cuentas administrativas, asignación de grupos y permisos.
- **Configuración de red:** asignación de IP estática, puerta de enlace y servidores DNS.
- **Instalación y configuración de software:** Nginx, herramientas útiles de administración (curl, wget, vim, htop, tree) y actualizaciones de sistema.
- **Seguridad básica:** configuración de firewall con UFW y apertura de puertos necesarios.
- **Despliegue de la página web:** creación de directorios, permisos, copiado de archivos HTML y CSS, activación de la configuración de Nginx y recarga del servicio.

Estas tareas representan acciones repetitivas y críticas que, automatizadas, permiten ahorrar tiempo, reducir errores humanos y mantener consistencia en entornos de producción.

Antes de pasar a Ansible hay algunos apartados que se pueden ampliar en el servidor previos a automatizarlos.

La idea es crear una arquitectura local como la que se representa en el siguiente diagrama:



## 2.1. Usuarios, Grupos y Permisos

Antes de empezar a automatizar con Ansible, necesito tener el servidor configurado con usuarios, grupos y permisos de manera que la automatización tenga sentido y sea replicable en cualquier máquina.

- **Creación de usuarios:**

voy a crear dos usuarios administrativos más:

```
sudo adduser sysadmin03
```

```
sudo adduser sysadmin04
```

Se añaden al grupo ***sudo*** para que tengan privilegios administrativos:

```
sudo usermod -aG sudo sysadmin03
sudo usermod -aG sudo sysadmin04
```

- **Crear el grupo DevOps:**

Simulo un entorno Development Operations creando un grupo de usuarios donde todos los miembros de este grupo tendrán permisos especiales sobre directorios

importantes (por ejemplo /var/www/infra-cloud o /etc/nginx), que es donde gestionan la infraestructura.

```
sudo groupadd devops
```

- **Añadir usuarios al grupo:**

Los usuarios **sysadmin03** y **sysadmin04** se añaden al grupo **devops**, lo que les permite gestionar archivos y servicios de manera controlada, siguiendo buenas prácticas de seguridad y colaboración en un entorno DevOps.

```
sudo usermod -aG devops sysadmin03  
sudo usermod -aG devops sysadmin04
```

- **Asignar permisos y propietarios a directorios clave:**

Asigno permisos y propietarios sobre los directorio **/var/www/infra-cloud** y **/etc/nginx/**, de forma que los usuarios del grupo puedan modificar archivos web sin comprometer la seguridad del resto del sistema. Esto simula cómo en entornos profesionales se delegan responsabilidades y se evita el uso de root para tareas habituales.

```
sudo chown -R sysadmin01:devops /var/www/infra-cloud/  
sudo chown -R root:devops /etc/nginx/
```

En **/var/www/infra-cloud/**: Asigna el propietario **sysadmin01** y el grupo **devops** al directorio y todo su contenido.

En **/etc/nginx/**: Asigna el propietario **root** y el grupo **devops** al directorio y todo su contenido.

```
sudo find /var/www/infra-cloud/ -type d -exec chmod 750 {} \;  
sudo find /var/www/infra-cloud/ -type f -exec chmod 640 {} \;  
sudo find /etc/nginx/ -type d -exec chmod 750 {} \;  
sudo find /etc/nginx/ -type f -exec chmod 640 {} \;
```

#### **En el directorio /var/www/infra-cloud/:**

Lectura, escritura y ejecución al propietario (sysadmin01). Lectura y ejecución al grupo (devops). Sin permisos para otros usuarios.

#### **En los archivos del directorio /var/www/infra-cloud/:**

Lectura y escritura al propietario (sysadmin01). Lectura al grupo (devops). Sin permisos para otros usuarios.

#### **En el directorio /etc/nginx/:**

Lectura, escritura y ejecución al propietario (root). Lectura y ejecución al grupo (devops). Sin permisos para otros usuarios.

#### **En los archivos del directorio /etc/nginx/:**

Lectura y escritura al propietario (root). Lectura al grupo (devops). Sin permisos para otros usuarios.

#### **- Diferenciación de usuarios y grupos:**

Para mantener una correcta gestión de la infraestructura y reflejar prácticas profesionales, los usuarios y grupos se diferencian de la siguiente manera:

- sysadmin01** (propietario principal):

Usuario con control total sobre los archivos y directorios críticos de la infraestructura web. Es el responsable de la administración completa de /var/www/infra-cloud/, pudiendo leer, escribir y ejecutar todo el contenido sin restricciones.

- Grupo devops** (usuarios sysadmin03 y sysadmin04):

Usuarios que forman parte del grupo devops tienen permisos limitados sobre ciertos recursos. Pueden leer y ejecutar archivos necesarios para trabajar en la infraestructura web y realizar tareas operativas delegadas, pero no tienen control total sobre los archivos propiedad de sysadmin01.

- sysadmin02** (usuario administrativo auxiliar):

Usuario administrativo con permisos generales de sudo, pero sin acceso directo a los archivos y directorios de la web crítica. Este usuario simula un

rol de soporte o gestión general, sin interferir en la operación diaria del sitio ni en los scripts automatizados que se implementarán posteriormente.

## 2.2. Configuración de acceso SSH con llaves públicas

El acceso SSH mediante llaves públicas permite que los usuarios puedan conectarse de forma segura al servidor sin usar contraseñas. Esto es fundamental para entornos profesionales y para que Ansible pueda ejecutar tareas automáticamente.

- **Generar la llave SSH en el cliente:**

Desde mi VM cliente, ejecuto el siguiente comando para crear un par de llaves (pública y privada):

```
inaki@inaki-VirtualBox:~$ ssh-keygen -t rsa -b 4096 -C "sysadmin01"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/inaki/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/inaki/.ssh/id_rsa
Your public key has been saved in /home/inaki/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:EQUpLlur7b3FB5UM5df+ZgQXcNsFAVi0ubAu2afL78 sysadmin01
The key's randomart image is:
+---[RSA 4096]----+
| .+=* =*B+ |
| ..0+=0000 |
| . . . . . 0 . |
| . o S . |
| . o . . o+ |
| . o . oo . . |
| . . o o o o |
| . . . . Eo |
+---[SHA256]-----+
```

Donde **-t rsa** es el tipo de llave (RSA). **-b 4096** es el tamaño de la llave en bits. **-C "sysadmin01"** es el comentario para identificar la llave.

Se generan dos archivos en **~/.ssh/**: **id\_rsa** (llave privada) y **id\_rsa.pub** (llave pública, que se copia al servidor)

- **Copiar la llave pública al servidor:**

Desde mi Linux Mint (cliente), copio la llave pública al servidor:

```
inaki@inaki-VirtualBox:~$ ssh-copy-id sysadmin01@192.168.0.2
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
sysadmin01@192.168.0.2's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'sysadmin01@192.168.0.2'"
and check to make sure that only the key(s) you wanted were added.
```

La llave se copia a `~/.ssh/authorized_keys` en el servidor.

- **Probar la conexión SSH sin contraseña:**

Ahora puedo conectarme al servidor con mi usuario sysadmin01 desde mi cliente sin usar contraseña. Esto simula cómo en empresas profesionales se da acceso seguro a servidores sin usar root ni compartir contraseñas.

```
inaki@inaki-VirtualBox:~$ ssh sysadmin01@192.168.0.2
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-83-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
```

Ahora hago lo mismo con sysadmin02, sysadmin03 y sysadmin04.

```
ssh-copy-id sysadmin02@192.168.0.2 ssh-copy-id sysadmin03@192.168.0.2
ssh-copy-id sysadmin04@192.168.0.2
```

## 2.3. Scripts

Voy a realizar unos scripts de mantenimiento para mejorar la seguridad del servidor web.

### 2.3.1. Script: Reiniciar Nginx

Durante la administración de un servidor web, es normal realizar cambios en la configuración de Nginx. Para que estos cambios surtan efecto, el servicio debe reiniciarse de manera segura. Crear un script para esta tarea permite automatizar, reducir errores humanos y documentar la acción de forma reproducible, un enfoque profesional y adecuado para entornos empresariales.

- **La carpeta bin en Linux**

**Bin**, viene de binaries, es decir, “programas ejecutables”. Este directorio es un lugar personal y seguro para guardar tus scripts ejecutables. Tiene la ventaja de no tener que poner toda la ruta del script para ejecutarlo.

Para ello, primero voy a crear la carpeta **bin** dentro de **/home/sysadmin01**:

```
sysadmin01@servidor01:~$ mkdir -p ~/bin
```

Dentro de bin, he creado el siguiente script:

```
sysadmin01@servidor01:~/bin$ ls  
restart_nginx.sh
```

```
GNU nano 7.2                                         bin/restart_nginx.sh  
#!/bin/bash  
  
# Archivo de log donde se registran todas las ejecuciones  
LOG_FILE="/home/sysadmin01/restart_nginx.log"  
  
# Mostrar fecha y hora del inicio del script  
echo "$(date '+%Y-%m-%d %H:%M:%S') - Inicio del script de reinicio de Nginx" >> "$LOG_FILE"  
  
# Comprobar si nginx está activo  
if systemctl is-active --quiet nginx; then  
    echo "El servicio está activo. REINICIANDO..." | tee -a "$LOG_FILE"  
else  
    echo "El servicio no está activo. INICIANDO..." | tee -a "$LOG_FILE"  
fi  
  
# Reinicio de Nginx  
if sudo systemctl restart nginx; then  
    echo "Reinicio completado con éxito." | tee -a "$LOG_FILE"  
else  
    echo "Error al reiniciar Nginx. Revisa el estado del servicio." | tee -a "$LOG_FILE"  
fi  
  
# Fin de script  
echo "Fin del script de reinicio de Nginx" >> "$LOG_FILE"  
  
# Estado del servicio  
STATUS=$(systemctl is-active nginx)  
if [ "$STATUS" = "active" ]; then  
    echo -e "\e[32m• Nginx ACTIVO\e[0m"  
else  
    echo -e "\e[31m• Nginx NO ACTIVO\e[0m"  
fi
```

**#!/bin/bash:** Indica que el script se ejecutará con Bash, el intérprete de comandos por defecto en muchas distribuciones Linux. Esto asegura compatibilidad con los comandos utilizados en el script.

**LOG\_FILE:** Archivo log donde se registra todo, como haría un administrador profesional.

**\$(date '+%Y-%m-%d %H:%M:%S')**: Genera la fecha y hora en ese formato.

**systemctl is-active --quiet nginx:** Comprueba si Nginx está activo antes de reiniciar.

**sudo systemctl restart nginx:** Reinicia el servicio de manera segura.

**tee -a "\$LOG\_FILE"**: Permite mostrar el mensaje por pantalla y guardar lo en el log al mismo tiempo.

**STATUS=\$(systemctl is-active nginx)**: Guarda en la variable si Nginx está activo.

**\e[32m** y **\e[31m**: Son códigos ANSI para color verde y rojo respectivamente.

#### - Permisos de ejecución

Añado permiso de ejecución al archivo ejecutable que se encuentra en

**/bin/restart\_nginx.sh**: `sudo chmod +x ~/bin/restart_nginx.sh`

#### - Añadir bin al PATH

Esto permite ejecutar los scripts directamente por su nombre, sin `./`.

Primero, escribo la linea **export PATH="\$HOME/bin:\$PATH"** al final del archivo **.bashrc**

```
sysadmin01@servidor01:~$ nano /home/sysadmin01/.bashrc
export PATH="$HOME/bin:$PATH"
```

Y con **source ~/.bashrc** hago recargar la configuración en la terminal actual.

```
sysadmin01@servidor01:~$ source ~/.bashrc
```

Ya estaría añadido bin al PATH. Podría ejecutar el script directamente desde cualquier directorio.

#### - Ejecución del script:

Ahora simplemente con hacer **restart\_nginx.sh** podría ejecutar el script y reiniciar el servicio de nginx. Esto es lo que se vería tras ejecutarlo:

```
sysadmin01@servidor01:~$ restart_nginx.sh
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
• Nginx ACTIVO
```

#### - Archivo con los log:

Cada vez que ejecute este script se irá registrando todo en el archivo log que hemos añadido en el script. Tanto las veces que se ha hecho un restart, como la hora y día a la que se ha hecho:

```
sysadmin01@servidor01:~$ ls
backups bin restart_nginx.log
sysadmin01@servidor01:~$ cat restart_nginx.log
2025-09-30 18:30:52 - Inicio del script de reinicio de Nginx
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
Fin del script de reinicio de Nginx
2025-09-30 18:31:18 - Inicio del script de reinicio de Nginx
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
Fin del script de reinicio de Nginx
2025-09-30 18:33:57 - Inicio del script de reinicio de Nginx
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
Fin del script de reinicio de Nginx
```

### 2.3.2. Script: Backup del sitio web y de configuración de Nginx

Para proteger los archivos críticos del proyecto y poder restaurarlos en caso de fallo, he creado un script de backup automático.

Este script realiza una copia comprimida tanto del sitio web principal (**/var/www/infra-cloud**) como de la configuración de Nginx (**/etc/nginx**). Así, mantengo copias de seguridad actualizadas, puedo restaurar rápidamente el servicio en caso de error, pérdida de datos y automatizo una tarea repetitiva que, de otro modo, sería manual y propensa a errores.

- **Creación del archivo del script:**

Voy a crear el archivo del script en **/usr/local/bin/**, una ubicación habitual para scripts de administración:

```
sudo nano /usr/local/bin/backup_infra-cloud.sh
```

```

GNU nano 7.2                                         backup_infra-cloud.sh
#!/bin/bash

# Carpeta donde se guardarán los backups
BACKUP_DIR="/home/sysadmin01/backups"
mkdir -p "$BACKUP_DIR"

# Nombre del archivo de backup con fecha y hora
DATE=$(date +%Y%m%d_%H%M%S)

# Directorios a respaldar
WEB_DIR="/var/www/infra-cloud"
NGINX_DIR="/etc/nginx"

# Crear copia comprimida del sitio web
sudo tar -czf "$BACKUP_DIR/infra-cloud_$DATE.tar.gz" -C / "${WEB_DIR:1}"

# Crear copia comprimida de la configuración de Nginx
sudo tar -czf "$BACKUP_DIR/nginx_$DATE.tar.gz" -C / "${NGINX_DIR:1}"

echo "Backups creados en $BACKUP_DIR"

```

**#!/bin/bash:** indica que el script se ejecutará con el intérprete de comandos Bash.

**BACKUP\_DIR="/home/sysadmin01/backups":** define la carpeta donde se almacenarán las copias.

**mkdir -p "\$BACKUP\_DIR":** crea la carpeta si no existe.

**DATE=\$(date +%Y%m%d\_%H%M%S):** genera una marca de tiempo para diferenciar cada copia.

**sudo tar -czf:** crea un archivo comprimido en formato **.tar.gz** con el contenido del directorio.

**\$BACKUP\_DIR/infra-cloud\_\$DATE.tar.gz:** Es dónde se guardará la copia de seguridad, con nombre que incluye la fecha y hora (**\$DATE**) para no sobrescribir otros backups.

**\${WEB\_DIR:1}:** quita el primer carácter “/” y deja **var/www/infra-cloud**.

Combinado con **-C /**, esto hace que dentro del **.tar.gz** las rutas sean relativas:

**var/www/infra-cloud/...** y no **/var/www/infra-cloud/**.

**echo:** muestra un mensaje de confirmación al finalizar.

- **Permisos de ejecución:**

Para que el archivo sea ejecutable, se otorgan permisos de ejecución:

```
sudo chmod +x /usr/local/bin/backup_infra-cloud.sh
```

- **Ejecución del script:**

Bastaría con escribir ***backup\_infra-cloud.sh*** desde cualquier directorio del servidor, para hacer los backups del sitio web y de la configuración de Nginx:

```
backup_infra-cloud.sh
```

Ya tendríamos creados los backups, solo ejecutando el script. Para comprobar que se han creado correctamente vamos a ***/home/sysadmin01/backups*** y listamos los archivos del directorio para confirmar que están ahí.

```
sysadmin01@servidor01:~$ cd /home/sysadmin01/backups/
sysadmin01@servidor01:~/backups$ ls
infra-cloud_20250925_123334.tar.gz  nginx_20250925_123334.tar.gz
```

Si quisiera restaurar un backup por algún error, pérdida de archivos o fallo, descomprimo el gzip en la ruta deseada y ya tendríamos, por ejemplo, los archivos del sitio web restaurado.

```
sudo tar -xzf /home/sysadmin01/backups/infra-cloud_20250925_123334.tar.gz -C /var/www/infra-cloud/
```

### **2.3.3. Script: Restaurar backup**

Para mantener la integridad del sitio web y la configuración del servidor, he desarrollado un script de restauración de los backups creados en el punto anterior. Este permite recuperar los archivos del sitio web principal (***/var/www/infra-cloud***) y la configuración de Nginx (***/etc/nginx***) asegurando así la continuidad del servicio con tan solo la ejecución de un archivo desde cualquier directorio.

- **Creación del archivo del script**

El archivo del script se guarda en la carpeta ***/usr/local/bin/*** para poder ejecutarlo fácilmente desde cualquier ubicación. Se crea con el siguiente contenido:

```
sudo nano /usr/local/bin/restaurar_backup_infra-cloud.sh
```

```

GNU nano 7.2                               /usr/local/bin/restaurar_backup_infra-cloud.sh *
#!/bin/bash

# Carpeta donde están los backups
BACKUP_DIR="/home/sysadmin01/backups"

# Comprobación de argumento
if [ -z "$1" ]; then
    echo "Uso: $0 <nombre_del_backup_sin_extension>"
    echo "Ejemplo: $0 infra-cloud_20251001_153000"
    echo "Últimos backups disponibles:"
    sudo ls -1t /home/sysadmin01/backups/ | head -n 10
    exit 1
fi

BACKUP_NAME="$1"

# Directorios originales
WEB_DIR="/var/www/infra-cloud"
NGINX_DIR="/etc/nginx"

# Restaurar sitio web/configuración nginx
echo "Restaurando desde ${BACKUP_DIR}/${BACKUP_NAME}.tar.gz..."
sudo tar -xzf "${BACKUP_DIR}/${BACKUP_NAME}.tar.gz" -C /
echo "Backup restaurado con éxito"

```

**`#!/bin/bash`**: Indica que el script se ejecutará con Bash, el intérprete de comandos por defecto en muchas distribuciones Linux.

**`BACKUP_DIR`**: Se establece la ruta donde se encuentran los backups realizados previamente. Esto permite que el script busque y descomprima los archivos correctos.

```

if [ -z "$1" ]; then

echo "Uso: $0 <nombre_del_backup_sin_extension>"

echo "Ejemplo: $0 infra-cloud_20251001_153000"

echo "Últimos backups disponibles:"

sudo ls -1t /home/sysadmin01/backups | head -n 10

exit 1

fi

```

Esta parte verifica mediante una estructura condicional “IF” si el usuario ha proporcionado un argumento, que debe ser el nombre del backup que desea restaurar. (sin la extensión .tar.gz).

**"-z \$1"**: comprueba si la variable está vacía y representa el primer argumento pasado al script.

Si no se proporciona un argumento: Se muestra un mensaje de uso correcto. Se listan los backups disponibles con **sudo ls** para que el usuario pueda elegir uno válido.

**exit 1**: detiene la ejecución del script para evitar errores posteriores.

**BACKUP\_DIR="/home/sysadmin01/backups"**: Variable para la carpeta donde se guardan los backups.

**WEB\_DIR="/var/www/infra-cloud"**: Variable para la ruta del sitio web principal.

**NGINX\_DIR="/etc/nginx"**: Variable para la ruta de la configuración de Nginx.

**BACKUP\_NAME="\$1"**: Variable para el nombre del backup a restaurar.

**tar -xzf**: descomprime el archivo .tar.gz.

- **Permisos de ejecución:**

Para que el archivo sea ejecutable, se otorgan permisos de ejecución:

```
sysadmin01@servidor01:/usr/local/bin$ sudo chmod +x restaurar_backup_infra-cloud.sh
```

- **Ejecución y demostración del script:**

Bastaría con escribir **restaurar\_backup\_infra-cloud.sh** desde cualquier directorio del servidor, para restaurar los backups del sitio web y de la configuración de Nginx.

Si al escribir el comando para ejecutar el script, no añadimos como argumento después del archivo del script, el backup que queremos restaurar nos sale el siguiente mensaje:

```
sysadmin01@servidor01:~$ restaurar_backup_infra-cloud.sh
Uso: /usr/local/bin/restaurar_backup_infra-cloud.sh <nombre_del_backup_sin_extension>
Ejemplo: /usr/local/bin/restaurar_backup_infra-cloud.sh infra-cloud_20251001_153000
Últimos backups disponibles:
nginx_20251001_192651.tar.gz
infra-cloud_20251001_192651.tar.gz
nginx_20251001_175302.tar.gz
infra-cloud_20251001_175302.tar.gz
nginx_20251001_175027.tar.gz
infra-cloud_20251001_175027.tar.gz
nginx_20251001_174858.tar.gz
infra-cloud_20251001_174858.tar.gz
nginx_20251001_174018.tar.gz
infra-cloud_20251001_174018.tar.gz
```

Para ejecutar bien el script, hay que añadir un backup disponible. Pero antes voy a hacer cambios en el HTML o CSS de la página infra-cloud para luego, restaurar el backup y volver a como estaba.

Antes de probar la restauración, es necesario contar con un backup válido. Para ello, genero una copia actual del sitio:

```
sysadmin01@servidor01:~$ backup_infra-cloud.sh
[sudo] password for sysadmin01:
Backups creados en /home/sysadmin01/backups
```

A continuación, modiflico los archivos del sitio web para simular un cambio. En mi caso, edito el archivo CSS y voy a cambiar el color de fondo a naranja.

```
sudo nano /var/www/infra-cloud/style.css
```

Cambio el color de fondo a naranja:

```
:root {
    --bg1:#eb9826; --bg2:#0b3a53; --accent:#00d4ff; --card:#0f2733;
    --text:#e6f7ff;
```

Al acceder al sitio desde el host mediante el navegador (*infra-cloud.local*), se aprecia el cambio en la apariencia de la página.



Una vez comprobado, ejecuto el script de restauración (***restaurar\_backup\_infra-cloud.sh***) indicando como argumento el último backup disponible (***infra-cloud\_20251002\_134008***):

```
sysadmin01@servidor01:~$ restaurar_backup_infra-cloud.sh infra-cloud_20251002_134008
Restaurando desde /home/sysadmin01/backups/infra-cloud_20251002_134008.tar.gz...
Backup restaurado con éxito
```

El script descomprime y restaura los archivos en sus rutas originales.

Tras recargar la página en el navegador, el color vuelve al estado que tenía en el momento de hacer el backup, verificando que el proceso de restauración ha funcionado correctamente.



## 2.4. Creación de otro sitio web básico

### 2.4.1. Configuración del segundo sitio web

- **Creación de carpeta, permisos y propietarios:**

```
sysadmin01@servidor01:~$ sudo mkdir -p /var/www/test
[sudo] password for sysadmin01:
sysadmin01@servidor01:~$ sudo chown -R sysadmin01:devops /var/www/test/
sysadmin01@servidor01:~$ sudo find /var/www/test/ -type d -exec chmod 750 {} \;
sysadmin01@servidor01:~$ sudo find /var/www/test/ -type f -exec chmod 640 {} \;
```

- **Crear index.html**

```
sysadmin01@servidor01:~$ nano /var/www/test/index.html
```

```
sudo nano /var/www/test/style.css
```

- **Crear configuración de Nginx (virtual host)**

```
sudo nano /etc/nginx/sites-available/test
```

```
GNU nano 7.2                                     /etc/nginx/sites-available/test
server {
    listen 80;
    server_name test.local;

    root /var/www/test;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    access_log /var/log/nginx/test_access.log;
    error_log  /var/log/nginx/test_error.log;
}
```

- **Crear enlace simbólico y habilitar sitio:**

```
sudo ln -s /etc/nginx/sites-available/test /etc/nginx/sites-enabled/test
```

```
sysadmin01@servidor01:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

```
sudo systemctl reload nginx
```

#### 2.4.2. Añadir alias en la VM cliente

Para añadir un alias en el host:

```
inaki@inaki-VirtualBox:~$ sudo nano /etc/hosts
```

Ahora escribimos la IP del servidor y el alias correspondiente de ambos sitios:

```
GNU nano 7.2                                     /etc/hosts *
127.0.0.1      localhost
127.0.1.1      inaki-VirtualBox
192.168.0.2     infracloud.local
192.168.0.2     test.local
# The following lines are desirable for IPv6 capable hosts
::1            ip6-localhost ip6-loopback
fe00::0         ip6-localnet
ff00::0         ip6-mcastprefix
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
```

```
inaki@inaki-VirtualBox:~$ ping infracloud.local
PING infracloud.local (192.168.0.2) 56(84) bytes of data.
64 bytes from infracloud.local (192.168.0.2): icmp_seq=1 ttl=64 time=0.498 ms
64 bytes from infracloud.local (192.168.0.2): icmp_seq=2 ttl=64 time=0.407 ms
^C
--- infracloud.local ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.407/0.452/0.498/0.045 ms
inaki@inaki-VirtualBox:~$ ping test.local
PING test.local (192.168.0.2) 56(84) bytes of data.
64 bytes from infracloud.local (192.168.0.2): icmp_seq=1 ttl=64 time=0.321 ms
64 bytes from infracloud.local (192.168.0.2): icmp_seq=2 ttl=64 time=0.397 ms
```

Ahora si escribimos el alias en el buscador del navegador accedemos sin escribir manualmente la IP de cada uno de los sitios web:

The image contains two screenshots of a web browser window. The top screenshot shows a site titled "Test Site — InfraCloud" with the URL "http://test.local". The content includes a heading "Comprobación" with a list of items: "Servidor: InfraCloud Test" and "Directorio: /var/www/test". The bottom screenshot shows a site titled "InfraCloud Solutions" with the URL "http://infracloud.local". This site has a heading "Demostración: Infraestructura en la nube con automatización" and a note stating it's a test page on Ubuntu with Nginx. It features sections for "Servicios" (with a list of automation steps), "Tecnologías" (Ubuntu, Nginx, Terraform, Ansible, Azure), and "Estado del servidor" (Server active — Nginx responds correctly, last check 25/9/2025, 13:58:11).

Para facilitar el acceso a los sitios web alojados en el servidor y simular un entorno profesional, he añadido alias locales en **/etc/hosts**. Esto permite utilizar nombres de dominio propios (**infracloud.local**, **test.local**) en lugar de la IP del servidor.

## 2.5. Habilitar HTTPS con certificado autofirmado

Antes de iniciar con este apartado, hay que instalar el paquete para certificados:

```
sudo apt update
sudo apt install openssl
```

Para asegurar la comunicación cifrada entre cliente y servidor he configurado Nginx con un certificado autofirmado. Este método es habitual en entornos de pruebas o desarrollo, ya que permite disponer de HTTPS sin depender de una autoridad certificadora.

#### - Creación del directorio de certificados:

En primer lugar he creado una carpeta, donde se guardarán el certificado y la clave privada:`sudo mkdir -p /etc/nginx/ssl`

Voy a darle permiso exclusivo de escritura, lectura y ejecución al propietario en este directorio: `sudo chmod 700 /etc/nginx/ssl`

#### - Generación del certificado y la clave:

Mediante OpenSSL voy a generar una clave RSA de 2048 bits y un certificado válido durante 365 días:

```
sudo openssl req -x509 -nodes -days \
```

Con este comando, “**req**” indica que voy a generar una petición de certificado o un certificado autofirmado. “**-x509**” pide a openssl que genere un certificado X.509 autofirmado, es decir crea directamente el certificado que Nginx usará. “**-nodes**” significa “no DES” (no cifrar la clave privada). Con esto la clave privada se guarda sin passphrase. “**-days 365**” es la duración en días del certificado (en este caso 1 año).

Sigo con “**-newkey rsa:2048**”, que genera una nueva clave privada RSA de 2048 bits al mismo tiempo que crea el certificado. 2048 bits es seguro y suficiente para pruebas y entornos reales básicos.

```
> -keyout /etc/nginx/ssl/infra-cloud.key \
> -out /etc/nginx/ssl/infra-cloud.crt
```

“**-keyout /etc/nginx/ssl/infra-cloud.key**” es la ruta donde se escribirá la clave privada. Y “**-out /etc/nginx/ssl/infra-cloud.crt**” es la ruta donde se escribirá el certificado X.509 (archivo público que se entrega a los clientes).

Ahora contesto interactivamente al ejecutar el comando (Country, State, Locality, Organization, Organizational Unit y CN= Common Name). El **CN** es lo más importante, es el nombre que usaré en **/etc/hosts** y en **server\_name** de Nginx). El navegador/cliente validará que el CN coincida con lo que pongo en la URL.

- **Seguridad: permisos recomendados para los certificados SSL**

Una vez que he generado la clave privada y el certificado autofirmado, es fundamental protegerlos correctamente, porque contienen información sensible (especialmente la clave privada). Los permisos recomendados son los siguientes:

```
sudo chown root:root /etc/nginx/ssl/infra-cloud.key /etc/nginx/ssl/infra-cloud.crt
sudo chmod 600 /etc/nginx/ssl/infra-cloud.key
sudo chmod 644 /etc/nginx/ssl/infra-cloud.crt
sudo chmod 700 /etc/nginx/ssl
```

Se asegura la protección de los certificados asignando propietarios y permisos adecuados:

- **Propietario y grupo (root:root)**: solo root puede modificar o eliminar los archivos, protegiendo la integridad.
- **Clave privada (chmod 600)**: lectura y escritura solo para root, evitando accesos no autorizados.
- **Certificado público (chmod 644)**: root puede escribir, otros solo leer; seguro y accesible para Nginx.
- **Directorio (chmod 700)**: solo root puede listar o modificar su contenido, protegiendo la ubicación de los archivos sensibles.

Estas configuraciones garantizan que la clave privada permanezca segura y el certificado público accesible, siguiendo buenas prácticas profesionales.

- **Configurar el Virtual Host para HTTPS:**

Abro el archivo de configuración de mi sitio:

```
sudo nano /etc/nginx/sites-available/infra-cloud
```

Le indico a Nginx que escuche en el puerto 443 (HTTPS) y use los certificados que he generado:

```
# Bloque HTTPS
server {
    listen 443 ssl;
    server_name _;
    root /var/www/infra-cloud;
    index index.html;

    ssl_certificate /etc/nginx/ssl/infra-cloud.crt;
    ssl_certificate_key /etc/nginx/ssl/infra-cloud.key;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Ahora compruebo la sintaxis para ver si hay algún fallo y hago un restart al servicio de Nginx:

```
sysadmin01@servidor01:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
sysadmin01@servidor01:~$ sudo systemctl restart nginx
```

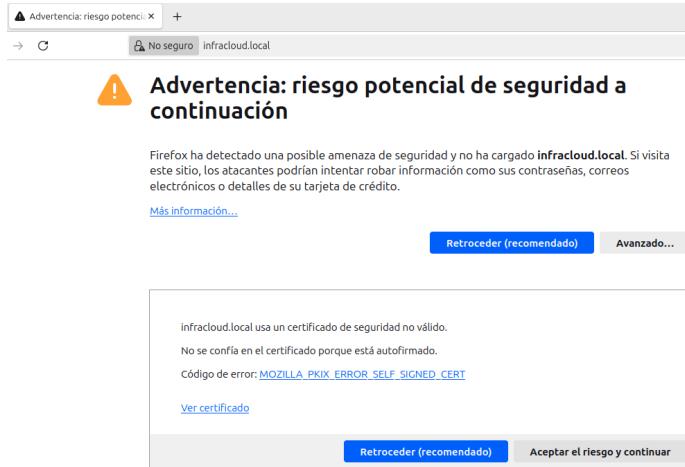
Ahora si, se ha configurado Nginx para servir contenido mediante HTTPS usando un certificado autofirmado. Esto asegura que las conexiones entre clientes y servidor estén cifradas, siguiendo buenas prácticas profesionales.

Ahora el servidor sirve mi web con HTTPS, y puedo acceder a ella de forma segura.

- **Prueba desde la terminal:**

```
sysadmin01@servidor01:~$ curl -Ik https://192.168.0.2
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Mon, 29 Sep 2025 19:26:10 GMT
Content-Type: text/html
Content-Length: 2936
Last-Modified: Thu, 25 Sep 2025 11:03:30 GMT
Connection: keep-alive
ETag: "68d52182-b78"
Accept-Ranges: bytes
```

- **Prueba desde el host:**



Antes de configurar HTTPS, todas las comunicaciones entre el cliente (navegador o máquina host) y el servidor web se transmitían en texto plano usando HTTP. Esto implicaba que cualquier usuario malintencionado en la misma red podría interceptar credenciales, modificar tráfico o espiar información enviada, representando un riesgo de seguridad.

Con la configuración actual de HTTPS, incluso con un certificado autofirmado, el tráfico queda cifrado, reduciendo significativamente este riesgo, aunque los navegadores mostrarán una advertencia sobre la confianza del certificado, ya que no está firmado por una autoridad reconocida.

## 2.6. Automatización parcial mediante CRON

Para reducir la intervención manual y garantizar que las tareas críticas del servidor se realicen de forma periódica y segura, he configurado **CRON**, la herramienta de programación de tareas de Linux. Con ello se automatizan los backups del sitio web y de la configuración de Nginx, se permite el reinicio controlado del servicio y se genera un registro de logs centralizado, siguiendo buenas prácticas profesionales de administración de sistemas.

### 2.6.1. Directorio de logs centralizado

Antes de programar las tareas, voy a crear un directorio dedicado para almacenar los registros (**logs**) de todas las operaciones automáticas (***sudo mkdir -p /var/log/server-logs/***):

```
sudo mkdir -p /var/log/server-logs
```

Una vez creado el directorio, es necesario establecer una política de permisos y propiedad adecuada para garantizar la seguridad de los registros generados por los scripts automatizados.

Por defecto, el sistema asigna la propiedad del directorio al usuario que lo crea, pero en entornos profesionales se recomienda definir de forma explícita qué usuarios y grupos pueden acceder a los logs.

En este caso, la infraestructura cuenta con los siguientes usuarios y grupos administrativos:

Sysadmin01: Administrador principal de la infraestructura.

Sysadmin02: Administrador auxiliar sin acceso a logs sensibles

Sysadmin03: Colaborador técnico (DevOps)

Sysadmin04: Colaborador técnico (DevOps)

- **Permisos y control de acceso al directorio de logs:**

El administrador (**sysadmin01**) mantiene la capacidad de gestión global, pero el grupo **devops** puede administrar los registros de manera más flexible.

```
sudo chown sysadmin01:devops /var/log/server-logs/
```

Se definen permisos que otorgan al grupo **devops** un rol operativo superior al del resto de usuarios, sin comprometer la integridad del sistema:

```
sudo chmod 770 /var/log/server-logs/
```

- **Seguridad y trazabilidad**

Esta estructura de permisos ofrece un equilibrio entre seguridad y operatividad:

- **Delegación de responsabilidades:** el grupo **devops** puede intervenir directamente sobre los registros sin necesidad de acceder como root.

- **Aislamiento de roles:** **sysadmin02** y otros usuarios del sistema no tienen acceso a información sensible.
- **Transparencia y control:** cada archivo de log queda vinculado al usuario que ejecuta los scripts, facilitando auditorías y seguimiento de actividad.

### 2.6.2. Script maestro

Para centralizar la gestión de tareas repetitivas del servidor, he desarrollado un script maestro denominado ***server\_maintenance.sh***.

- **Script para automatizar backup de Infra-cloud y reinicio de Nginx**

Este script se encarga de coordinar la ejecución de los scripts existentes de backup y reinicio de Nginx, asegurando que todas las acciones queden registradas en un log centralizado, sin interferir con el funcionamiento de los scripts originales.

```
sudo nano /home/sysadmin01/bin/server_maintenance.sh
```

Escribo lo siguiente:

```
GNU nano 7.2                                         /home/sysadmin01/bin/server_maintenance.sh *
#!/bin/bash

# Archivo de log centralizado
LOG_DIR="/var/log/server-logs"
LOG_FILE="$LOG_DIR/server_maintenance_$(date '+%Y%m%d').log"

# Crear log si no existe
mkdir -p "$LOG_DIR"
touch "$LOG_FILE"

# Función para registrar mensajes
log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

log "==== Inicio del script de mantenimiento ==="

# Ejecutar backup del sitio web y configuración de nginx
log "Ejecutando backup del sitio web y configuración de Nginx..."
backup_infra-cloud.sh >> "$LOG_FILE" 2>&1
log "Backup completado"

# Reiniciar nginx
log "Reiniciando Nginx..."
restart_nginx.sh >> "$LOG_FILE" 2>&1
log "Nginx reiniciado correctamente"

log "==== Fin del script de mantenimiento===="
```

- **Permisos de ejecución:**

Para que el archivo sea ejecutable, se otorgan permisos de ejecución:

```
sudo chmod 755 /home/sysadmin01/bin/server_maintenance.sh
```

- Funcionalidad del script maestro:

Ejecuto el script maestro, que hará un backup del sitio y la configuración de nginx, además de reiniciar nginx:

```
sysadmin01@servidor01:~$ server_maintenance.sh
2025-10-06 13:30:42 - === Inicio del script de mantenimiento ===
2025-10-06 13:30:42 - Ejecutando backup del sitio web y configuración de Nginx...
2025-10-06 13:30:42 - Backup completado
2025-10-06 13:30:42 - Reiniciando Nginx...
2025-10-06 13:30:43 - Nginx reinicado correctamente
2025-10-06 13:30:43 - === Fin del script de mantenimiento==
```

Para comprobar que se ha creado un archivo de log en la carpeta **server-logs**, listo los archivos y aquí aparecerán los logs creados al ejecutar el script maestro:

```
sysadmin01@servidor01:~$ ls -l /var/log/server-logs/
total 4
-rw-rw-r-- 1 sysadmin01 sysadmin01 488 oct  6 13:30 server_maintenance_20251006.log
```

Hago **cat** al archivo **.log** y compruebo el registro:

```
sysadmin01@servidor01:/var/log/server-logs$ cat server_maintenance_20251006.log
2025-10-06 13:30:42 - === Inicio del script de mantenimiento ===
2025-10-06 13:30:42 - Ejecutando backup del sitio web y configuración de Nginx...
Backups creados en /home/sysadmin01/backups
2025-10-06 13:30:42 - Backup completado
2025-10-06 13:30:42 - Reiniciando Nginx...
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
• Nginx ACTIVO
2025-10-06 13:30:43 - Nginx reinicado correctamente
2025-10-06 13:30:43 - === Fin del script de mantenimiento==
```

**Registro de ejecución:** Cada vez que se ejecuta el script, se crea un registro en **/var/log/server-logs/** con fecha y hora, detallando las acciones realizadas y el resultado de cada una. Esto facilita auditorías y seguimiento de la actividad del servidor.

**Ejecución de backups:** Se llama al script **backup\_infra-cloud.sh** para generar copias de seguridad del sitio web y de la configuración de Nginx, garantizando que siempre exista una versión reciente en caso de fallo o necesidad de restauración manual.

**Reinicio controlado de Nginx:** Tras realizar el backup, se ejecuta el script ***restart\_nginx.sh*** para reiniciar el servicio de Nginx de forma segura, aplicando cambios de configuración si los hubiera y manteniendo el servicio web operativo.

**No incluye restauración automática:** La restauración de backups se mantiene como tarea manual a través de ***restaurar\_backup\_infra-cloud.sh***, evitando sobrescribir datos en cada ejecución y preservando la integridad del sitio web y la configuración de Nginx.

### 2.6.3. Automatización con CRON. Configuración del cron job:

Ahora voy a automatizar el mantenimiento del servidor para que se realice de forma regular sin intervención manual. CRON permite ejecutar comandos o scripts en momentos específicos.

Antes, vamos a añadir una última línea al script ***server\_maintenance.sh***, para dejar constancia en ***server.log*** (otro registro que irá en la carpeta server-logs).

```
# Mensaje de mantenimiento ejecutado
echo "[${date}] Mantenimiento ejecutado correctamente." >> /var/log/server-logs/server.log
```

Ahora cada vez que ejecutemos el script nos creará otro registro con la fecha.

```
sysadmin01@servidor01:~$ cat /var/log/server-logs/server.log
[mié 22 oct 2025 19:44:55 CEST] Mantenimiento ejecutado correctamente.
[mié 22 oct 2025 19:46:29 CEST] Mantenimiento ejecutado correctamente.
[mié 22 oct 2025 19:46:30 CEST] Mantenimiento ejecutado correctamente.
```

El script ***server\_maintenance.sh*** se ha programado para ejecutarse cada minuto (solo como prueba). Para ello ejecuto el comando:

```
sysadmin01@servidor01:~$ sudo crontab -e
```

En la última línea voy a escribir lo siguiente:

```
* * * * * /home/sysadmin01/bin/server_maintenance.sh >> /var/log/server-logs/cron.log 2>&1
```

\* \* \* \* \* indica que se programa una tarea para que se ejecute cada minuto. Cada uno de los cinco asteriscos representan: minuto, hora, día del mes, mes y día de la semana.

**>> /var/log/server-logs/cron.log** registra la salida de CRON en un log aparte.

**2>&1** asegura que los errores también se registren.

Si quisiera que se ejecutará cada vez que inició el servidor usaría:

```
@reboot /home/sysadmin01/bin/server_maintenance.sh >> /var/log/server-logs/cron.log 2>&1
```

Para ejecutarse automáticamente cada día a las 03:00 h, momento en el que la carga del servidor es mínima, reduciendo el riesgo de afectar al servicio web. Para ello, se edita el archivo y queda así:

```
0 3 * * * /home/sysadmin01/bin/server_maintenance.sh >> /var/log/server-logs/cron.log 2>&1
```

### - Gestión de logs

En **/var/log/server-logs** se manejan tres tipos de registros:

```
sysadmin01@servidor01:/var/log/server-logs$ ls
cron.log  server.log  server_maintenance_20251006.log
```

**server.log:** Se actualiza con cada ejecución manual del script maestro `server_maintenance.sh` y acumula todas las acciones continuas en un solo archivo.

**cron.log:** Se genera únicamente cuando el script se ejecuta automáticamente mediante CRON. Contiene todas las ejecuciones automáticas en un único archivo.

**server\_maintenance\_YYYY-MM-DD.log:** Se crea un archivo independiente por cada día de ejecución, ya sea manual o automática y registra de forma detallada todas las acciones realizadas ese día.

Esta estructura permite diferenciar fácilmente la ejecución manual de la automática, mantener un historial diario completo y evitar la pérdida de información relevante.

## **3. Fase 3 - Automatización con Ansible**

En esta fase el proyecto da un salto importante, pasando de la configuración manual o semiautomatizada (CRON) a un entorno gestionado completamente mediante Ansible.

El objetivo es que toda la configuración del servidor en local (instalación de paquetes, creación de usuarios, permisos, servicios y archivos) se pueda reproducir automáticamente, sin



intervención manual y con los mismos resultados en cualquier entorno.

Con Ansible, todas las tareas que antes se realizaban paso a paso se agrupan y ejecutan desde un único archivo llamado **playbook**, escrito en formato YAML. Esto permite definir de forma clara el estado deseado del sistema y aplicarlo sobre los equipos definidos (en este caso, la máquina virtual del servidor) mediante conexión SSH.

Una de las principales ventajas de Ansible es que no requiere instalar agentes en las máquinas gestionadas. El nodo de control (host principal) se conecta de forma remota mediante autenticación por clave pública, lo que simplifica la gestión y mejora la seguridad.

El objetivo de esta fase es conseguir que el servidor web pueda desplegarse desde cero, de manera completamente automatizada, incluyendo la instalación y configuración de Nginx, creación y despliegue de sitios web, gestión de usuarios y permisos, configuración de acceso SSH, ejecución de tareas de mantenimiento y backup, entre otras tareas.

Este paso marca la transición hacia una infraestructura totalmente automatizada y reproducible, que no solo facilita la administración del servidor, sino que también sirve como base para la siguiente fase del proyecto, donde se integrará Terraform para desplegar esta misma infraestructura en Microsoft Azure.

### **3.1 Ventajas de Ansible**

El uso de Ansible en el proyecto proporciona varias ventajas clave:

**Reproducibilidad:** Las configuraciones y tareas que antes se realizaban manualmente ahora se pueden aplicar automáticamente, asegurando que cualquier servidor configurado sea idéntico al original.

**Eficiencia y ahorro de tiempo:** Reduce considerablemente el tiempo requerido para instalar paquetes, configurar servicios, gestionar usuarios o aplicar cambios en múltiples equipos.

**Escalabilidad y modularidad:** Permite añadir nuevos servicios, scripts o configuraciones de manera ordenada y flexible, adaptándose al crecimiento del proyecto sin afectar la estabilidad de lo implementado.

**Base para la nube:** Al centralizar y automatizar la infraestructura local, se sienta la base para su posterior despliegue en la nube mediante Terraform y Microsoft Azure.

### 3.2 Instalación y configuración inicial de Ansible

Antes de poder automatizar cualquier tarea, es necesario preparar el entorno de Ansible en el host de control y establecer la comunicación segura con las máquinas gestionadas.

#### 3.2.1 Instalación en el host

Ansible se instalará en la **VM Linux Mint**, que funcionará como **host de control**. No es necesario instalar Ansible en la máquina que se desea replicar (Ubuntu Server), ya que Ansible opera de manera **remota vía SSH**.

Los pasos principales para la instalación en Linux Mint son:

- **Actualizar repositorios:**

```
inaki@inaki-VirtualBox:~$ sudo apt update
```

```
inaki@inaki-VirtualBox:~$ sudo apt upgrade -y
```

- **Instalar Ansible:**

```
inaki@inaki-VirtualBox:~$ sudo apt install ansible -y
```

- **Verificar versión:**

```
inaki@inaki-VirtualBox:~$ ansible --version
ansible [core 2.16.3]
  config file = None
  configured module search path = ['/home/inaki/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/inaki/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

Esto garantiza que el host de control está preparado para ejecutar playbooks y gestionar los nodos de manera remota.

### 3.2.2. Configuración del inventario e intercambio de llaves SSH

Para que Ansible pueda comunicarse con el servidor y ejecutar tareas de forma remota, es necesario configurar dos elementos fundamentales:

- **Archivo de inventario**, que indica qué servidores se van a gestionar,
- **Autenticación SSH**, que permite la conexión sin contraseñas mediante claves públicas.

El host de control es la máquina Linux Mint, y el nodo gestionado es el Ubuntu Server, al que se accede mediante el usuario sysadmin01.

Cabe destacar que la autenticación SSH mediante llaves públicas ya fue configurada en una fase anterior (**apartado 4.2**), por lo que el acceso remoto sin contraseña está completamente operativo. Desde el host de control, se puede acceder al servidor Ubuntu simplemente ejecutando:

```
inaki@inaki-VirtualBox:~$ ssh sysadmin01@192.168.0.2
```

Esto confirma que el intercambio de claves se realizó correctamente y que Ansible podrá conectarse sin requerir credenciales interactivas.

#### Configuración del inventario

En el host de control (Linux Mint), donde he instalado Ansible, creo una carpeta de trabajo para el proyecto:

```
inaki@inaki-VirtualBox:~$ mkdir -p ~/ansible  
inaki@inaki-VirtualBox:~$ cd ~/ansible
```

Dentro de ella creo un archivo de inventario denominado **inventory.ini**, donde se definen los nodos gestionados.

```
inaki@inaki-VirtualBox:~/ansible$ nano inventory.ini
```

Dentro del archivo añado el siguiente contenido:

```
|_ GNU nano 7.2                                         inventory.ini *
[servidores]
ubuntu_server ansible_host=192.168.0.2 ansible_user=sysadmin01
```

Donde:

**[servidores]** Es el nombre del grupo que agrupa los nodos gestionados.

**ubuntu\_server** Es el nombre identificativo del nodo dentro del grupo.

**ansible\_host** Es la dirección IP del servidor Ubuntu

**ansible\_user** Indica el usuario remoto con el que se establecerá la conexión SSH

De este modo, Ansible sabrá que el nodo **ubuntu\_server** pertenece al grupo **servidores** y podrá ejecutar sobre él los distintos playbooks o comandos que se definan.

### 3.2.3 prueba de conectividad

En este paso voy a verificar la conexión entre el host de control (Linux Mint) y el servidor gestionado (Ubuntu Server).

Esto se hace mediante el módulo ping de Ansible, que comprueba si es posible establecer comunicación SSH y ejecutar comandos de forma remota.

#### Ejecución del módulo de prueba

El intercambio de llaves públicas ya se realizó en fases anteriores, pero es importante verificar que Ansible puede comunicarse correctamente con el servidor a través de SSH.

Desde el **host de control**, voy a hacer una prueba simple:

```
inaki@inaki-VirtualBox:~$ ansible all -i /home/inaki/ansible/inventory.ini -m ping
ubuntu_server | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

### 3.3. Estructura del proyecto Ansible

Para organizar correctamente la automatización del servidor, se ha diseñado una estructura de proyecto ordenada y modular. Esta organización permite separar claramente los archivos de configuración global, los inventarios, los *playbooks* y los *roles*, facilitando la reutilización del código y el mantenimiento del sistema.

El proyecto de Ansible se ubica en el host de control (Linux Mint) dentro del directorio **/home/inaki/ansible/**, que constituye la raíz de trabajo del entorno automatizado.

Dentro de esta carpeta se irán incluyendo todos los elementos necesarios para definir, ejecutar y mantener la infraestructura gestionada.

#### 3.3.1. Archivos principales (`ansible.cfg`, `hosts`, `playbooks/`)

Se procede a crear la estructura base del proyecto de Ansible.

Esta estructura permitirá centralizar la configuración, definir los nodos gestionados y almacenar los *playbooks* y roles que compondrán la automatización completa del servidor.

#### Crear el directorio principal del proyecto

Desde el **host de control**, he creado la carpeta principal donde residirá toda la configuración de Ansible:

```
inaki@inaki-VirtualBox:~$ mkdir -p ~/ansible/playbooks/roles
inaki@inaki-VirtualBox:~$ cd ~/ansible/
```

Con esto se genera la siguiente estructura inicial:

- Directorio general **/home/inaki/ansible**: Donde irán carpetas y archivos de ansible.

- Directorio **/home/inaki/ansible/playbooks**: Donde irán los roles.
- Directorio **/home/inaki/ansible/playbooks/roles**: Donde irá cada uno de los roles a automatizar mediante playbooks.

### Crear el archivo de configuración global “ansible.cfg”

Dentro del directorio raíz del proyecto (**~/ansible/**), he creado el archivo principal de configuración de Ansible:

```
inaki@inaki-VirtualBox:~/ansible$ nano ansible.cfg
```

Y dentro escribo lo siguiente:

```
GNU nano 7.2                                         ansible.cfg *
[defaults]
inventory = /home/inaki/ansible/inventory.ini
remote_user = sysadmin01
host_key_checking = False
retry_files_enabled = False
roles_path = /home/inaki/ansible/playbooks/roles
```

- **inventory** indica la ubicación del archivo de inventario donde se definirán los nodos gestionados.
- **remote\_user** usuario por defecto con el que se conectará Ansible (en este caso, sysadmin01).
- **host\_key\_checking = False** desactiva la comprobación de autenticidad de las claves SSH, evitando confirmaciones manuales.
- **retry\_files\_enabled = False** desactiva la creación de archivos de reintento tras fallos de conexión.
- **roles\_path** especifica el directorio donde estarán los roles personalizados del proyecto.

### Crear el archivo de inventario “inventory.ini”

El archivo **inventory.ini** es el que define los hosts o grupos de hosts sobre los que se ejecutarán los *playbooks*. Ya lo he creado en el punto 5.2.2.

```
[servidores]
ubuntu_server ansible_host=192.168.0.2 ansible_user=sysadmin01
```

## Crear el playbook principal “site.yml”

El playbook principal sirve como punto de entrada para la automatización del servidor. Desde este archivo se llaman los diferentes roles que gestionan tareas específicas, garantizando que la configuración se aplique de forma ordenada y reproducible.

Creo el archivo desde el directorio playbooks:

```
inaki@inaki-VirtualBox:~/ansible/playbooks$ nano site.yml
```

Esta serían las líneas que habría que añadir para todos los roles:

```
GNU nano 7.2                                         site.yml *
-----
- name: Configuración completa del servidor web
  hosts: servidores
  become: yes
  roles:
    - entorno_basico
    - usuarios
    - nginx
    - mantenimiento
```

Aunque de momento escribo solo el primer rol para que no haya conflictos:

```
-----
- name: Configuración completa del servidor web
  hosts: servidores
  become: yes
  roles:
    - entorno_basico
```

- **hosts: servidores**: indica el grupo de nodos sobre los que se ejecutará el playbook, definido en inventory.ini.
- **become: yes**: permite elevar privilegios (sudo) cuando sea necesario.
- **roles**: lista los roles que se aplicarán, de forma modular. Cada rol corresponde a un conjunto de tareas que automatizan parte de la configuración del servidor.

### 3.3.2. Organización de roles y tareas

Los roles permiten agrupar tareas relacionadas y mantener la automatización modular y reutilizable. Para este proyecto se han definido cuatro roles principales:

- **entorno\_basico**: preparación inicial del servidor, instalación de herramientas y configuración de firewall.
- **usuarios**: creación de usuarios, grupos y permisos, así como configuración de acceso SSH.
- **nginx**: instalación y configuración del servidor web, creación de sitios y gestión de permisos.
- **mantenimiento**: scripts de backup, reinicio de servicios y gestión de logs.

Cada rol contiene subdirectorios estándar de Ansible (**tasks**, **vars**, **templates**, **files**) para organizar de forma ordenada las tareas, variables, plantillas y archivos necesarios para su ejecución.

Esta estructura modular permite añadir fácilmente nuevos roles en el futuro (por ejemplo, SSL o monitorización) sin afectar la configuración existente.

### **3.4 Playbook principal: configuración automática del servidor**

El playbook principal (**site.yml**) es el núcleo del proyecto Ansible.

Desde este archivo se presenta la ejecución de los distintos roles, garantizando que cada parte del sistema se configure de manera automática, ordenada y reproducible.

Cada rol representa un conjunto lógico de tareas relacionadas, lo que facilita la modularidad y el mantenimiento del proyecto.

#### **3.4.1. Rol entorno\_basico:**

Este rol tiene como objetivo dejar el sistema operativo preparado para el resto de configuraciones.

Las tareas incluidas son **actualización del sistema, configuración de zona horaria, hostname, firewall y paquetes básicos**.

#### **Creación de la estructura del rol:**

Desde el directorio raíz del proyecto (**~/ansible/playbooks/roles**), ejecuto lo siguiente:

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ ansible-galaxy init entorno_basico
- Role entorno_basico was created successfully
```

Esto genera la siguiente estructura:

```
entorno_basico/
├── defaults/
│   └── main.yml (Variables por defecto)
├── files/ (Archivos estáticos)
├── handlers/
│   └── main.yml (Manejadores de eventos)
├── meta/
│   └── main.yml (Metadatos)
├── tasks/
│   └── main.yml (Tareas principales)
├── templates/ (Plantillas Jinja2)
├── tests/
│   ├── inventory
│   └── test.yml
└── vars/
    └── main.yml (Variables específicas del rol)
```

Si hacemos **ls** sobre **entorno\_basico**:

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles/entorno_basico$ ls
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

**Edición de archivos:**

Voy a empezar a editar las tareas principales del rol. En el caso de **entorno\_basico** voy a editar únicamente el **main.yml** de los directorios **task/**, **vars/** y **handlers/**. Los archivos de los demás directorios no va a hacer falta editarlos para este rol en concreto:

**Rol entorno\_basico: “tasks/main.yml”:**

- Contienen la lista de acciones que Ansible ejecuta sobre los nodos gestionados.
- Cada tarea tiene un nombre descriptivo **name** y un módulo (como **apt**, **ufw**, **hostname**).
- Puede usar variables, loops y notificar handlers.

Dentro del directorio **task** edito el archivo de configuración:

```
~/ansible/playbooks/roles$ nano entorno_basico/tasks/main.yml
```

Y escribo lo siguiente:

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/entorno_basico/tasks
GNU nano 7.2                      main.yml
- name: Actualizar el sistema
  apt:
    update_cache: yes
    upgrade: dist
  become: yes

- name: Configurar zona horaria
  timezone:
    name: "{{ timezone }}"
  become: yes

- name: Configurar hostname
  hostname:
    name: "{{ server_hostname }}"
  become: yes
  notify: Reiniciar hostname

- name: Instalar herramientas básicas
  apt:
    name: "{{ basic_tools }}"
    state: present
    update_cache: yes
  become: yes

- name: Configuración básica de firewall (UFW)
  ufw:
    rule: allow
    port: "{{ item }}"
    proto: tcp
  loop: "{{ ufw_allowed_ports }}"
  become: yes
  notify: Reiniciar UFW

- name: Habilitar UFW
  ufw:
    state: enabled
  become: yes
```

**Rol entorno\_basico: “vars/main.yml”:**

- Archivo donde se declaran variables reutilizables.
- Se usan en **tasks** con la sintaxis **{{ variable }}**.
- Permite modificar configuraciones sin tocar directamente el playbook o la tarea.

Dentro del directorio **vars** edito el archivo de configuración:

```
~/ansible/playbooks/roles$ nano entorno_basico/vars/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles
GNU nano 7.2                                entorno_basico/vars/main.yml
---
# vars file for entorno_basico

server_hostname: "servidor01"
timezone: "Europe/Madrid"
basic_tools:
  - curl
  - wget
  - vim
  - nano
  - htop
  - tree
  - net-tools
  - iputils-ping
ufw_allowed_ports:
  - 22
  - 80
  - 443
```

Rol **entorno\_basico**: “**handlers/main.yml**”:

- Son tareas especiales que esperan ser notificadas por **notify**.
- Solo se ejecutan si una tarea que lo llama cambia algo.
- Evita reinicios innecesarios: si el hostname ya es correcto, el handler no se ejecuta.

Dentro del directorio **handlers** edito el archivo de configuración:

```
~/ansible/playbooks/roles$ nano entorno_basico/handlers/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles
GNU nano 7.2                                entorno_basico/handlers/main.yml *
---
# handlers file for entorno_basico

- name: Reiniciar hostname
  command: hostnamectl set-hostname "{{ server_hostname }}"
  become: yes

- name: Reiniciar UFW
  command: ufw reload
  become: yes
```

## Explicación de palabras y módulos:

- **Become**: Significa que la tarea se ejecuta con privilegios elevados. Es equivalente a hacer sudo <comando> en la terminal.
- **Notify**: No ejecuta la acción inmediatamente. Envía una notificación al handler y solo se ejecuta al final del playbook si alguna tarea lo notifica. Sirve para acciones que solo deben ejecutarse cuando hay un cambio, como reiniciar un servicio.
- **Loop**: Permite ejecutar la misma tarea varias veces con diferentes elementos de una lista. Ejemplo: abrir varios puertos con UFW, instalar varios paquetes, crear varios usuarios, etc.
- **{{ item }}**: Es una variable especial de Ansible que se usa dentro de bucle. Como por ejemplo para hacer referencia a los puertos en el ufw.
- **Command**: Hace referencia al propio comando a usar.

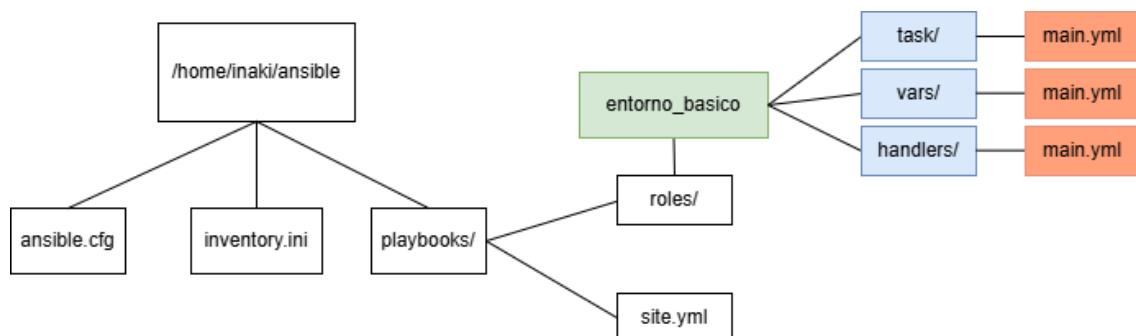
## Flujo lógico de ejecución:

Una vez se ejecute “**ansible-playbook -i inventory.ini playbooks/site.yml**”

- Ansible lee el playbook principal **site.yml**
- En site.yml ve que debe aplicar el rol **entorno\_basico**.
- Carga las variables desde **vars/main.yml**.
- Ejecuta las tareas de **tasks/main.yml**, usando esas variables.
- Si alguna tarea usa **notify**, activa el **handler** correspondiente.
- Al final, ejecuta los handlers (solo si fueron notificados).

## Estructura del proyecto:

La estructura de momento se vería así:



## Comprobaciones:

Para comprobar la sintaxis:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --syntax-check
playbook: playbooks/site.yml
```

Este comando **no ejecuta nada**. Solo revisa que todo esté bien escrito y enlazado (YAML, rutas, roles, variables...).

Para hacer una comprobación general del rol, al comando “**ansible-playbook -i inventory.ini playbooks/site.yml**”, añado “**–tags entorno\_basico**” (para especificar el rol), **–check** (para especificar que es una prueba) y **“–ask-become-pass”** (para especificar la contraseña para hacer cambios):

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --tags entorno_basico --check
--ask-become-pass
BECOME password:

PLAY [Configuración completa del servidor web] ****
TASK [Gathering Facts] ****
ok: [ubuntu_server]

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

De momento estaría todo correcto. Si hubiera errores o aparecerán en respuesta al comando escrito.

### 3.4.2. Rol usuarios

Este rol se encarga de la creación, configuración y gestión de usuarios y grupos en el servidor. Su objetivo es garantizar que la administración del sistema se realice de forma segura y consistente, siguiendo buenas prácticas de organización de permisos y grupos en un entorno DevOps.

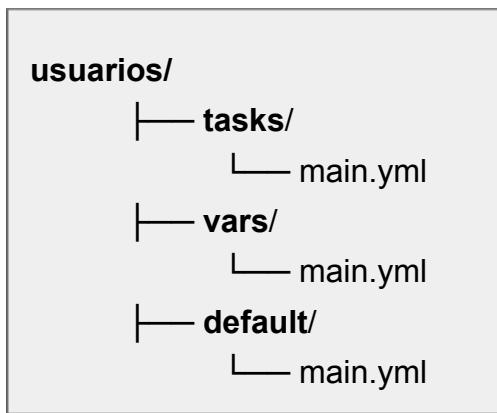
Las tareas incluidas son **crear usuarios administrativos y estándar con privilegios** definidos, **añadir los usuarios a los grupos correspondientes (sudo, devops)** según sus funciones y **configurar acceso SSH seguro mediante llaves públicas**

## Creación de la estructura del rol:

Desde el directorio raíz del proyecto (`~/ansible/playbooks/roles`), ejecuto lo siguiente:

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ ansible-galaxy init usuarios
- Role usuarios was created successfully
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ cd usuarios/
inaki@inaki-VirtualBox:~/ansible/playbooks/roles/usuarios$ ls
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

La estructura que usaremos será la siguiente:



### Edición de archivos:

En primer lugar añadimos la línea **usuarios** en `/playbooks/site.yml`:

```
GNU nano 7.2                                         site.yml
...
- name: Configuración completa del servidor web
  hosts: servidores
  become: yes
  roles:
    - entorno_basico
    - usuarios
```

Voy a empezar a editar las tareas principales del rol. En el caso de **usuarios** voy a editar únicamente el **main.yml** de los directorios **task/**, **vars/** y **default/**. Los archivos de los demás directorios no va a hacer falta editarlos para este rol en concreto.

#### Rol usuarios: “tasks/main.yml”:

Dentro del directorio **task** edito el archivo de configuración:

```
~/ansible/playbooks/roles$ nano usuarios/tasks/main.yml
```

Y escribo lo siguiente:

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/usuarios/tasks
GNU nano 7.2                                         main.yml *
```

```
- name: Crear grupo devops
  group:
    name: devops
    state: present

- name: Crear usuarios administrativos
  user:
    name: "{{ item }}"
    group: sudo
    shell: /bin/bash
    state: present
    password: "{{ user_password_hash }}"
  loop: "{{ admin_users }}"

- name: Crear usuarios DevOps
  user:
    name: "{{ item }}"
    group: devops
    shell: /bin/bash
    state: present
    password: "{{ user_password_hash }}"
  loop: "{{ devops_users }}"

- name: Crear directorio .ssh para cada usuario
  file:
    path: "/home/{{ item }}/.ssh"
    state: directory
    owner: "{{ item }}"
    group: sudo
    mode: '0700'
  loop: "{{ admin_users + devops_users }}"

- name: Copiar clave publica del host de control a cada usuario
  authorized_key:
    user: "{{ item }}"
    state: present
    key: "{{ lookup('file', ssh_public_key_path) }}"
  loop: "{{ admin_users + devops_users }}"
```

En este archivo se definen las tareas que ejecuta el rol usuarios:

- Se crea el grupo devops.
- Se crean los usuarios administrativos, incluidos en el grupo sudo, y los usuarios DevOps, incluidos en el grupo devops.
- Se genera de forma automática el directorio **.ssh** en los home de cada usuario, asegurando que los permisos sean correctos (0700).
- Mediante el módulo **authorized\_key**, se copia la clave pública SSH del host de control a cada cuenta, permitiendo la autenticación sin contraseña.
- Gracias al uso de loops, cada tarea se repite dinámicamente para los usuarios.

**Rol usuarios: “vars/main.yml”:**

Antes de nada, para generar la contraseña que será igual para todos los usuarios (**Holaquetal123**):

```
python3 -c 'import crypt; print(crypt.crypt("Holaquetal123", crypt.mksalt(crypt.METHOD_SHA512)))'
```

Este comando genera el texto cifrado (hash) que será el que usaremos para la variable “**user\_password\_hash**”.

Dentro del directorio **vars** edito el archivo de configuración:

```
~/ansible/playbooks/roles/usuarios$ nano vars/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/usuarios/vars
GNU nano 7.2                                     main.yml *
---
# vars file for usuarios

admin_users:
  - sysadmin01
  - sysadmin02

devops_users:
  - sysadmin03
  - sysadmin04

user_password_hash: $6$mWT5tjJvTUd18tui$8G1jGYyJcXhQTQuXnfEblgYAxD4ebSLbtn6vGC1dMuqZz>
```

Este archivo define las variables específicas del rol, que determinan los usuarios que se crearán durante la automatización.

- El grupo **admin\_users** incluye las cuentas con privilegios administrativos (sudo).
- El grupo **devops\_users** agrupa a los usuarios técnicos encargados de la gestión operativa del servidor.

**Rol usuarios: “default/main.yml”:**

Dentro del directorio **default** edito el archivo de configuración:

```
~/ansible/playbooks/roles/usuarios$ nano defaults/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/usuarios
GNU nano 7.2                                     defaults/main.yml *
---
# defaults file for usuarios

# Variables por defecto del rol usuarios
ssh_public_key_path: "~/.ssh/id_rsa.pub"
```

- En este archivo se definen las variables por defecto del rol.
- La variable ***ssh\_public\_key\_path*** indica la ruta donde se encuentra la clave pública SSH en el equipo de control

### Explicación de palabras y módulos:

- **group**: Módulo que gestiona grupos de usuarios (crear, modificar o borrar).
- **user**: Módulo que crea y configura cuentas de usuario
- **state: present**: Significa que el recurso (usuario, grupo o archivo) debe existir.
- **loop**: Sirve para repetir una tarea varias veces.
- **file**: Módulo que crea, borra o modifica archivos y directorios.
- **path**: Ruta del archivo o carpeta que se crea o modifica.
- **owner / group**: Dueño y grupo propietario del archivo o carpeta.
- **mode**: Permisos del archivo (**0700** = solo el dueño puede leer, escribir y ejecutar).
- **authorized\_key**: Módulo que añade una clave pública SSH al archivo `~/.ssh/authorized_keys` del usuario, para que pueda conectarse sin contraseña.
- **lookup('file', ...)**: Función que permite leer el contenido de un archivo local.
- **{{ item }}**: Variable temporal que representa cada elemento del loop (por ejemplo, cada usuario).

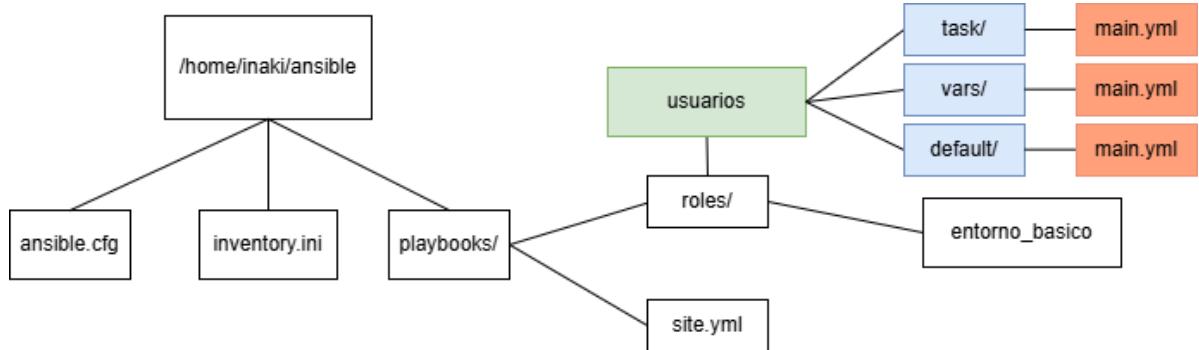
### Flujo lógico de ejecución:

Una vez se ejecute “***ansible-playbook -i inventory.ini playbooks/site.yml***”:

- Ansible lee el playbook principal ***site.yml***.
- En *site.yml* ve que debe aplicar el rol ***usuarios***.
- Carga las variables desde ***vars/main.yml*** y ***default/main.yml***
- Ejecuta las tareas de ***tasks/main.yml***, usando esas variables.

### Estructura del proyecto:

La estructura de momento se vería así:



## Comprobaciones:

Para comprobar la sintaxis:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --syntax-check
playbook: playbooks/site.yml
```

Para hacer una comprobación general:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --tags usuario --check --ask-become-pass
BECOME password:

PLAY [Configuración completa del servidor web] ****
TASK [Gathering Facts] ****
ok: [ubuntu_server]

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

De momento estaría todo correcto.

### 3.4.3. Rol nginx

El rol `nginx` tiene como objetivo automatizar la instalación, configuración y despliegue del sitio principal ***infra-cloud***. Esto permite replicar la infraestructura de manera segura y consistente, garantizando que los permisos, propietarios y configuración sean adecuados.

Las tareas incluidas son **instalar nginx, asegurar que el servicio nginx está habilitado y activo, crear directorio del sitio web, copiar archivos estáticos del sitio web (index.html y style.css), configurar archivo de nginx, crear enlace simbólico en sites-enabled, desactivar sitios por defecto de nginx y reiniciar el servicio.**

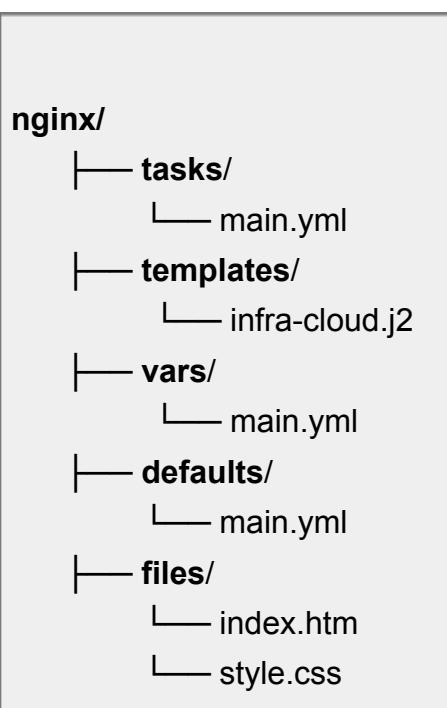
## Creación de la estructura del rol:

Desde el directorio raíz del proyecto (`~/ansible/playbooks/roles`), ejecuto lo siguiente:

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ ansible-galaxy init nginx
- Role nginx was created successfully
```

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles/nginx$ ls
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
inaki@inaki-VirtualBox:~/ansible/playbooks/roles/nginx$
```

La estructura que usaremos será la siguiente:



## Edición de archivos:

En primer lugar añadimos la línea `nginx` en `/playbooks/site.yml`:

```
GNU nano 7.2                                         site.yml *
---
- name: Configuración completa del servidor web
  hosts: servidores
  become: yes
  roles:
    - entorno_basico
    - usuarios
    - nginx
```

Voy a empezar a editar las tareas principales del rol. En el caso del rol ***nginx*** voy a editar el ***main.yml*** de los directorio ***task/***, ***vars/*** y ***default/***. Por otro lado crearemos el archivo ***infra-cloud.j2*** en ***templates/***. En ***files/*** crearemos ***index.html*** y ***style.css***.

### Rol ***nginx***: “***tasks/main.yml***”:

Dentro del directorio ***task*** edito el archivo de configuración:

```
~/ansible/playbooks/roles$ nano nginx/tasks/main.yml
```

```
GNU nano 7.2                                     main.yml
- name: Instalar Nginx
  apt:
    name: nginx
    state: present
    update_cache: yes
  become: yes
  tags: nginx

- name: Asegurar que el servicio Nginx está habilitado y activo
  service:
    name: nginx
    state: started
    enabled: yes
  become: yes
  tags: nginx

- name: Crear directorio del sitio web
  file:
    path: "{{ site_root }}"
    state: directory
    owner: "{{ nginx_user }}"
    group: "{{ nginx_group }}"
    mode: '0755'
  become: yes
  tags: nginx

- name: Copiar archivos estáticos del sitio web (index.html y style.css)
  copy:
    src: "{{ item }}"
    dest: "{{ site_root }}/"
    owner: "{{ nginx_user }}"
    group: "{{ nginx_group }}"
    mode: '0644'
  with_items:
    - files/index.html
    - files/style.css
  become: yes
  tags: nginx
```

```

- name: Configurar archivo de Nginx para el sitio infra-cloud
  template:
    src: "{{ template_src }}"
    dest: "{{ nginx_config_path }}/{{ site_name }}"
    owner: root
    group: "{{ nginx_group }}"
    mode: '0640'
  become: yes
  tags: nginx

- name: Crear enlace simbólico en sites-enabled
  file:
    src: "{{ nginx_config_path }}/{{ site_name }}"
    dest: "{{ nginx_enabled_path }}/{{ site_name }}"
    state: link
  become: yes
  tags: nginx

- name: Desactivar sitios por defecto de Nginx
  file:
    path: "{{ nginx_enabled_path }}/{{ item }}"
    state: absent
  loop: "{{ default_sites }}"
  become: yes
  tags: nginx

- name: Reiniciar Nginx para aplicar cambios
  service:
    name: nginx
    state: restarted
  become: yes
  tags: nginx

```

En este archivo se definen las tareas que ejecuta el rol usuarios:

Primero se instala nginx. Se asegura que el servicio está activo y habilitado. Después crea el directorio de infra-cloud. Copia los archivos del sitio web (html y css). Configura /etc/nginx con la configuración del sitio. Crea enlace simbólico de sites-available a sites-enabled. Desactiva los sitios default. Para acabar reinicia el servicio.

- ***update\_cache***: refresca la lista de paquetes.
- ***state***: estado deseado
- ***become***: eleva privilegios a sudo cuando es necesario.
- ***service***: módulo para gestionar servicios (iniciar, detener, reiniciar, habilitar).
- ***file***: módulo para crear directorios, enlaces simbólicos o eliminar archivos.
- ***copy***: módulo para copiar archivos estáticos al nodo gestionado.
- ***template***: copia archivos **.j2** y reemplaza variables de ansible.
- ***with\_items / loop***: permite iterar sobre listas de elementos.
- ***tags***: etiqueta para ejecutar tareas específicas con --tags.

## Rol nginx: “vars/main.yml”:

Dentro del directorio **vars** edito el archivo de configuración:

```
~/ansible/playbooks/roles/nginx$ nano vars/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/nginx
GNU nano 7.2                                         vars/main.yml *
---
# vars file for nginx

nginx_config_path: /etc/nginx/sites-available
nginx_enabled_path: /etc/nginx/sites-enabled
default_sites:
  - default
```

Este archivo define las variables específicas del rol: los archivos **sites-available** y **sites-enabled** y el sitio **default**.

- **nginx\_config\_path**: ubicación de los archivos de configuración de sitios en Nginx.
- **nginx\_enabled\_path**: ubicación de los enlaces simbólicos activos.
- **default\_sites**: lista de sitios por defecto que se eliminarán para evitar conflictos.

## Rol nginx: “defaults/main.yml”:

Dentro del directorio **defaults** edito el archivo de configuración:

```
~/ansible/playbooks/roles/nginx$ nano defaults/main.yml
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/nginx
GNU nano 7.2                                         defaults/main.yml *
---
# defaults file for nginx

site_name: infra-cloud
site_root: /var/www/infra-cloud
nginx_user: sysadmin01
nginx_group: devops
nginx_port: 80
template_src: infra-cloud.j2
```

- **site\_name**: nombre identificador del sitio.
- **site\_root**: ruta donde se alojarán los archivos del sitio web.

- ***nginx\_user*** y ***nginx\_group***: propietario y grupo del directorio del sitio, garantizando permisos correctos.
- ***nginx\_port***: puerto en el que escuchará nginx.
- ***template\_src***: nombre del archivo de plantilla en *templates/*

### Rol nginx: “*templates/infra-cloud.j2*”:

Dentro del directorio ***templates/*** voy a crear el siguiente archivo ***j2***:

```
~/ansible/playbooks/roles/nginx/templates$ nano infra-cloud.j2
```

```
inaki@inaki-VirtualBox: ~/ansible/playbooks/roles/nginx/templates
GNU nano 7.2                                     infra-cloud.j2 *
server {
    listen {{ nginx_port }};
    server_name _;

    root {{ site_root }};
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    error_log /var/log/nginx/{{ site_name }}_error.log;
    access_log /var/log/nginx/{{ site_name }}_access.log;
}
```

Esto sería la copia literal del archivo *infra-cloud* que se ubica en mi servidor original en los directorios *sites-available* y *sites-enabled*.

### Rol nginx: “*files/*”:

Dentro del directorio ***files/*** voy a crear los siguientes archivos ***html*** y ***css***:

```
~/ansible/playbooks/roles/nginx/files$ nano index.html
```

```
~/ansible/playbooks/roles/nginx/files$ nano style.css
```

Voy a copiar los archivos de mi servidor original para que sirva exactamente la misma página.

### Flujo lógico de ejecución:

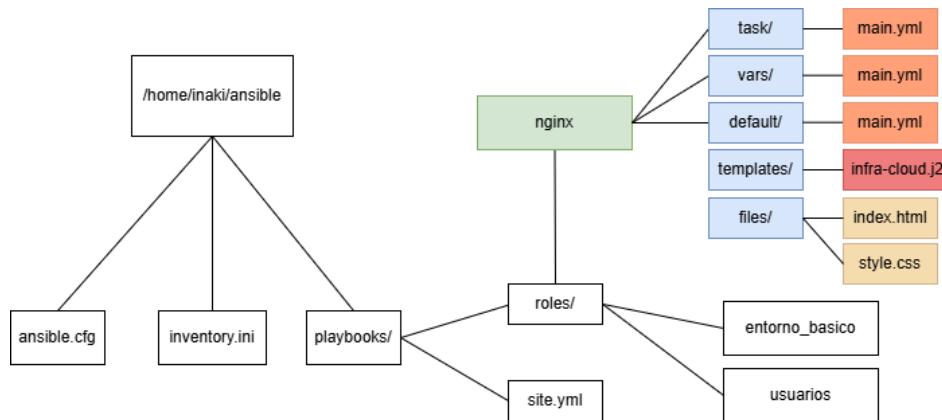
Una vez se ejecute “***ansible-playbook -i inventory.ini playbooks/site.yml***”

- Ansible lee el playbook principal ***site.yml***.
- En *site.yml* ve que debe aplicar el rol ***nginx***.
- Carga las variables desde ***vars/main.yml*** y ***default/main.yml***

- Carga los archivos de **files/** (**index.html** y **style.css**)
- Usa el archivo **infra-cloud.j2** de **templates/**
- Ejecuta las tareas de **tasks/main.yml**, usando esas variables.

## Estructura del proyecto:

La estructura de momento se vería así:



## Comprobaciones:

Para comprobar la sintaxis:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --syntax-check
playbook: playbooks/site.yml
```

Para hacer una comprobación general:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --tags nginx --check --ask-bec
ome-pass
BECOME password:
PLAY [Configuración completa del servidor web] ****
TASK [Gathering Facts] ****
ok: [ubuntu_server]
TASK [nginx : Instalar Nginx] ****
ok: [ubuntu_server]
TASK [nginx : Asegurar que el servicio Nginx está habilitado y activo] ****
ok: [ubuntu_server]
TASK [nginx : Crear directorio del sitio web] ****
ok: [ubuntu_server]
TASK [nginx : Copiar archivos estáticos del sitio web (index.html y style.css)] ****
changed: [ubuntu_server] => (item=files/index.html)
changed: [ubuntu_server] => (item=files/style.css)
TASK [nginx : Configurar archivo de Nginx para el sitio infra-cloud] ****
changed: [ubuntu_server]
TASK [nginx : Crear enlace simbólico en sites-enabled] ****
ok: [ubuntu_server]
TASK [nginx : Desactivar sitios por defecto de Nginx] ****
ok: [ubuntu_server] => (item=default)
TASK [nginx : Reiniciar Nginx para aplicar cambios] ****
changed: [ubuntu_server]
PLAY RECAP ****
ubuntu_server      : ok=9   changed=3   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0
```

De momento estaría todo correcto.

#### 3.4.4. Rol mantenimiento

El rol **mantenimiento** se encarga de automatizar las tareas básicas de administración y conservación del servidor, garantizando la disponibilidad del servicio y la seguridad de los datos.

Su objetivo es **disponer de scripts reutilizables** que ejecuten copias de seguridad, restauraciones y reinicios de servicio, y además **automatizar su ejecución periódica** de forma controlada.

#### Creación de la estructura del rol:

Desde el directorio raíz del proyecto (**~/ansible/playbooks/roles**), ejecuto lo siguiente:

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ ansible-galaxy init mantenimiento
- Role mantenimiento was created successfully
```

```
inaki@inaki-VirtualBox:~/ansible/playbooks/roles$ cd mantenimiento/
inaki@inaki-VirtualBox:~/ansible/playbooks/roles/mantenimiento$ ls
defaults files handlers meta README.md tasks templates tests vars
```

La estructura que usaremos será la siguiente:



#### Edición de archivos:

En primer lugar añadimos la línea **mantenimiento** en **/playbooks/site.yml**:

```

GNU nano 7.2                                         ansible/playbooks/site.yml *
---
- name: Configuración completa del servidor web
  hosts: servidores
  become: yes
  roles:
    - entorno_basico
    - usuarios
    - nginx
    - mantenimiento

```

Voy a empezar a editar las tareas principales del rol. En el caso del rol ***mantenimiento*** voy a editar el ***main.yml*** de los directorios ***task/*** y ***default/***. Por otro lado, en ***files/*** crearemos ***restart\_nginx.sh***, ***backup\_infra-cloud.sh*** y ***restaurar\_backup\_infra-cloud.sh***.

#### Rol mantenimiento: “tasks/main.yml”:

Dentro del directorio ***task*** edito el archivo de configuración:

```
~/ansible/playbooks/roles/mantenimiento/tasks$ nano main.yml
```

```

- name: Copiar scripts de mantenimiento al servidor
  copy:
    src: "{{ item }}"
    dest: "{{ mantenimiento_dir }}/"
    owner: root
    group: root
    mode: '0755'
  loop:
    - restart_nginx.sh
    - backup_infra-cloud.sh
    - restaurar_backup_infra-cloud.sh
  become: yes

- name: Programar tarea automática diaria (backup + reinicio)
  cron:
    name: "Ejecución diaria de mantenimiento"
    minute: "0"
    hour: "3"
    user: root
    job: "{{ mantenimiento_dir }}/backup_infra-cloud.sh && {{ mantenimiento_dir }}/restart_nginx.sh"
  become: yes

```

Con esta configuración, el sistema copiará scripts de mantenimiento y ejecutará automáticamente una rutina diaria de mantenimiento que realiza una copia de seguridad de los datos del servidor y reinicia los servicios web, garantizando estabilidad y disponibilidad sin intervención manual.

- ***copy***: copia los tres scripts desde el directorio ***files/*** del rol hasta el servidor.

- **owner** y **group**: asignan la propiedad a **root** por seguridad.
- **mode: '0755'**: otorga permisos de lectura y ejecución global, y escritura sólo al propietario.
- **loop**: ejecuta la tarea en loop para cada script indicado.
- **become: yes**: permite ejecutar la tarea con privilegios administrativos (**sudo**).
- **cron**: crea una tarea programada en el sistema.
  - **name**: descripción de la tarea.
  - **minute y hour**: especifica el horario de ejecución.
  - **user: root**: ejecuta los scripts con permisos administrativos.
  - **job**: orden que ejecutará Ansible cada día: se realiza una copia de seguridad y posteriormente se reinicia el servicio nginx.

### Rol mantenimiento: “default/main.yml”:

Dentro del directorio **default** edito el archivo de configuración:

```
nano ansible/playbooks/roles/mantenimiento/defaults/main.yml

inaki@inaki-VirtualBox: ~
GNU nano 7.2                               ansible/playbooks/roles/mantenimiento/defaults/main.yml
---
# defaults file for mantenimiento
# Ruta donde se copiarán los scripts de mantenimiento
mantenimiento_dir: /usr/local/bin
```

Esta variable define la ubicación global donde se instalarán los scripts (**/usr/local/bin**). Este directorio se encuentra incluido en la variable de entorno **\$PATH**, lo que permite ejecutar los scripts desde cualquier ubicación del sistema, sin necesidad de indicar su ruta completa.

### Rol mantenimiento: “files/”:

Dentro del directorio **files/** voy a crear los siguientes archivos **.sh**:

- **restart\_nginx.sh**:

```
nano ansible/playbooks/roles/mantenimiento/files/restart_nginx.sh
```

```

#!/bin/bash

# Archivo de log donde se registran todas las ejecuciones
LOG_FILE="/home/sysadmin01/restart_nginx.log"

# Mostrar fecha y hora del inicio del script
echo "$(date '+%Y-%m-%d %H:%M:%S') - Inicio del script de reinicio de Nginx" >> "$LOG_FILE"

# Comprobar si nginx está activo
if systemctl is-active --quiet nginx; then
    echo "El servicio está activo. REINICIANDO..." | tee -a "$LOG_FILE"
else
    echo "El servicio no está activo. INICIANDO..." | tee -a "$LOG_FILE"
fi

# Reinicio de Nginx
if sudo systemctl restart nginx; then
    echo "Reinicio completado con éxito." | tee -a "$LOG_FILE"
else
    echo "Error al reiniciar Nginx. Revisa el estado del servicio." | tee -a "$LOG_FILE"
fi

# Fin de script
echo "Fin del script de reinicio de Nginx" >> "$LOG_FILE"

# Estado del servicio
STATUS=$(systemctl is-active nginx)
if [ "$STATUS" = "active" ]; then
    echo -e "\e[32m● Nginx ACTIVO\e[0m"
else
    echo -e "\e[31m● Nginx NO ACTIVO\e[0m"
fi

```

- ***backup\_infra-cloud.sh:***

```
nano ansible/playbooks/roles/mantenimiento/files/backup_infra-cloud.sh
```

```

GNU nano 7.2                               ansible/playbooks/roles/mantenimiento/files/backup_infra-cloud.sh
#!/bin/bash

# Carpeta donde se guardarán los backups
BACKUP_DIR="/home/sysadmin01/backups"
mkdir -p "$BACKUP_DIR"

# Nombre del archivo de backup con fecha y hora
DATE=$(date +%Y%m%d_%H%M%S)

# Directorios a respaldar
WEB_DIR="/var/www/infra-cloud"
NGINX_DIR="/etc/nginx"

# Crear copia comprimida del sitio web
sudo tar -czf "$BACKUP_DIR/infra-cloud_$DATE.tar.gz" -C / "${WEB_DIR:1}"

# Crear copia comprimida de la configuración de Nginx
sudo tar -czf "$BACKUP_DIR/nginx_$DATE.tar.gz" -C / "${NGINX_DIR:1}"

echo "Backups creados en $BACKUP_DIR"

```

- ***restaurar\_backup\_infra-cloud.sh:***

```
nano ansible/playbooks/roles/mantenimiento/files/restaurar_backup_infra-cloud.sh
```

```

GNU nano 7.2          ansible/playbooks/roles/mantenimiento/files/restaurar backup infra-cloud.sh
#!/bin/bash

# Carpeta donde están los backups
BACKUP_DIR="/home/sysadmin01/backups"
mkdir -p "$BACKUP_DIR"

# Comprobación de argumento
if [ -z "$1" ]; then
    echo "Uso: $0 <nombre_del_backup_sin_extension>"
    echo "Ejemplo: $0 infra-cloud_20251001_153000"
    echo "Últimos backups disponibles:"
    sudo ls -1t /home/sysadmin01/backups/ | head -n 10
    exit 1
fi

BACKUP_NAME="$1"

# Directorios originales
WEB_DIR="/var/www/infra-cloud"
NGINX_DIR="/etc/nginx"

# Restaurar sitio web/configuración nginx
echo "Restaurando desde $BACKUP_DIR/${BACKUP_NAME}.tar.gz..."
sudo tar -xzf "$BACKUP_DIR/${BACKUP_NAME}.tar.gz" -C /

# Mensaje de éxito
echo "Backup restaurado con éxito"

```

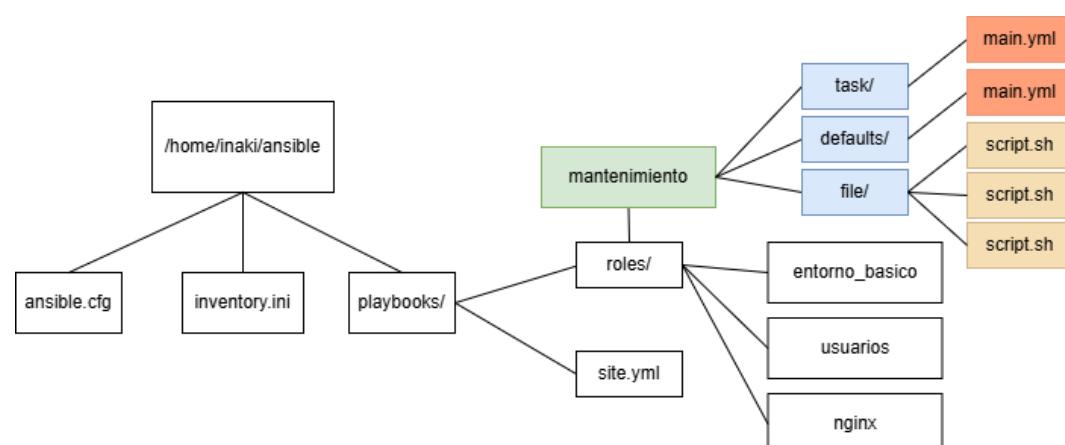
## Flujo lógico de ejecución:

Una vez se ejecute “**ansible-playbook -i inventory.ini playbooks/site.yml**”

- Ansible lee el playbook principal **site.yml**.
- En site.yml ve que debe aplicar el rol **mantenimiento**.
- Carga la variable **mantenimiento\_dir** desde **default/main.yml**
- Carga los archivos de **files/ (.sh)**
- Ejecuta las tareas de **tasks/main.yml**

## Estructura del proyecto:

La estructura de momento se vería así:



## Comprobaciones:

Para comprobar la sintaxis:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --syntax-check
playbook: playbooks/site.yml
```

Para hacer una comprobación general:

```
inaki@inaki-VirtualBox:~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --tags mantenimiento --check -
-ask-become-pass
BECOME password:

PLAY [Configuración completa del servidor web] ****
TASK [Gathering Facts] ****
ok: [ubuntu_server]

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

## 3.5. Despliegue completo en una nueva máquina virtual con Ansible

Una vez desarrollados y probados los distintos roles Ansible (**entorno\_basico**, **usuarios**, **nginx** y **mantenimiento**) sobre el servidor principal, el siguiente paso consiste en replicar todo el entorno de forma automatizada en una nueva máquina virtual Ubuntu Server.

El objetivo es garantizar que cualquier sistema recién desplegado pueda quedar totalmente configurado y operativo con una sola orden, sin intervención manual.

### 3.5.1 Preparación de la nueva máquina virtual

**Creación de la VM:** Se crea una nueva máquina virtual con **Ubuntu Server 22.04 LTS**, utilizando VirtualBox:

```
Ubuntu 24.04.3 LTS webserver02 tty1
webserver02 login: inaki
Password:
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-87-generic x86_64)
```

**Configuración básica inicial:** Asigno una IP estática de la misma red que el host y compruebo conexión server-host mediante ping:

```
GNU nano 7.2                                         /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.0.4/24
      gateway4: 192.168.0.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

```
inaki@inaki-VirtualBox:~$ ping 192.168.0.4
PING 192.168.0.4 (192.168.0.4) 56(84) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=64 time=0.462 ms
```

```
inaki@webserver02:~$ ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=0.390 ms
```

**Se actualiza el sistema base:** aunque con el playbook entorno\_basico tambien lo hace. `sudo apt update && sudo apt upgrade -y`

**Se instala el paquete SSH:**

```
sudo apt install openssh-server -y
```

**Comprobación de acceso desde el host de control (Linux Mint):** Esto es necesario e imprescindible porque es la única forma por la cual ansible se puede ejecutar desde el host de control para replicar el servidor.

```
inaki@inaki-VirtualBox:~$ ssh inaki@192.168.0.4
The authenticity of host '192.168.0.4 (192.168.0.4)' can't be established.
ED25519 key fingerprint is SHA256:nYCWNc9IxdrP8CpQ645rkWBEIvQdXDAa1rWGjZMeHwY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.0.4' (ED25519) to the list of known hosts.
inaki@192.168.0.4's password:
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-87-generic x86_64)
```

También necesito el programa ***sshpass***, que es el que Ansible usa para poder pasar la contraseña SSH de forma automática.

```
inaki@inaki-VirtualBox:~$ sudo apt install sshpass -y
```

### 3.5.2 Actualización del inventario de Ansible

En el host de control, dentro del proyecto, edito el archivo ***inventory.ini*** para añadir la nueva máquina virtual:

```
inaki@inaki-VirtualBox:~/ansible$ nano inventory.ini
  GNU nano 7.2                                inventory.ini *
[servidores]
#ubuntu_server ansible_host=192.168.0.2 ansible_user=sysadmin01
ubuntu_server02 ansible_host=192.168.0.4 ansible_user=inaki
```

### 3.5.3 Despliegue automático completo. Ejecución de playbooks.

Una vez añadida la nueva máquina, se ejecuta el playbook principal, que combina todos los roles definidos en el proyecto (además añado **--ask-pass** porque no tengo conexión con clave pública de momento, solo con contraseña y **--ask-become-pass** para que me pida la contraseña de **sudo**):

```
~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --ask-pass --ask-become-pass
```

Este comando:

- Conecta por SSH a la nueva máquina virtual.
- Aplica el playbook del rol **entorno\_basico** (actualización del sistema, configuración de zona horaria, hostname, firewall y paquetes básicos)
- Aplica el playbook del rol **usuarios** (crear usuarios administrativos y estándar con privilegios, añadir los usuarios a los grupos correspondientes y configurar **acceso SSH**)
- Aplica el playbook del rol **nginx** (instalar nginx, asegurar que el servicio nginx está habilitado y activo, crear directorio del sitio web, copiar archivos estáticos del sitio web (index.html y style.css), configurar archivo de nginx, crear enlace simbólico en sites-enabled, desactivar sitios por defecto de nginx y reiniciar el servicio)
- Aplica el playbook del rol **mantenimiento** (disponer de scripts reutilizables y automatizar su ejecución periódica)

Tras ejecutarlo, va ir tarea a tarea en cada uno de los roles ejecutando sus playbooks correspondientes. Hay tareas que las dará como **ok** y otras como **changed**. Si hay errores los marca como **failed** o **unreachable**.

Ejemplo de tareas:

```

TASK [nginx : Desactivar sitios por defecto de Nginx] ****
ok: [ubuntu_server02] => (item=default)

TASK [nginx : Reiniciar Nginx para aplicar cambios] ****
changed: [ubuntu_server02]

TASK [mantenimiento : Copiar scripts de mantenimiento al servidor] ****
ok: [ubuntu_server02] => (item=restart_nginx.sh)
ok: [ubuntu_server02] => (item=backup_infra-cloud.sh)
ok: [ubuntu_server02] => (item=restaurar_backup_infra-cloud.sh)

```

La automatización con ansible en el nuevo servidor estaría correcta.

```

PLAY RECAP ****
ubuntu_server02 : ok=22   changed=2    unreachable=0   failed=0    skipped=0   rescued=0   ignored=0

```

### 3.6. Verificación tras ejecución de playbooks.

Después de la ejecución del playbook, se realiza una comprobación manual y automática de los cambios:

#### Entorno básico:

- Verificación actualización del sistema:

```

Calculando la actualización... Hecho
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.

```

- Verificación zona horaria:

```

sysadmin01@servidor01:~$ timedatectl
          Local time: lun 2025-11-03 13:47:04 CET
          Universal time: lun 2025-11-03 12:47:04 UTC
                  RTC time: lun 2025-11-03 12:47:04
                 Time zone: Europe/Madrid (CET, +0100)
System clock synchronized: yes
          NTP service: active
      RTC in local TZ: no

```

- Verificación hostname:

```

sysadmin01@servidor01:~$ hostname
servidor01

```

- Verificación firewall (UFW):

```
sysadmin01@servidor01:~$ sudo ufw status
Status: active

To                         Action      From
--                         ----       --
22/tcp                      ALLOW       Anywhere
80/tcp                      ALLOW       Anywhere
443/tcp                     ALLOW       Anywhere
22/tcp (v6)                 ALLOW       Anywhere (v6)
80/tcp (v6)                 ALLOW       Anywhere (v6)
443/tcp (v6)                ALLOW       Anywhere (v6)
```

- Verificación paquetes básicos:

```
sysadmin01@servidor01:~$ which curl
/usr/bin/curl
sysadmin01@servidor01:~$ which wget
/usr/bin/wget
sysadmin01@servidor01:~$ which vim
/usr/bin/vim
sysadmin01@servidor01:~$ which htop
/usr/bin/htop
sysadmin01@servidor01:~$ which tree
/usr/bin/tree
```

## Usuarios y grupos:

- Verificación usuarios y grupos:

```
sysadmin01@servidor01:~$ id sysadmin01
uid=1001(sysadmin01) gid=27(sudo) groups=27(sudo)
sysadmin01@servidor01:~$ id sysadmin02
uid=1002(sysadmin02) gid=27(sudo) groups=27(sudo)
sysadmin01@servidor01:~$ id sysadmin03
uid=1003(sysadmin03) gid=1001(devops) groups=1001(devops)
sysadmin01@servidor01:~$ id sysadmin04
uid=1004(sysadmin04) gid=1001(devops) groups=1001(devops)
```

- Verificación acceso SSH sin contraseña:

```
inaki@inaki-VirtualBox:~$ ssh sysadmin01@192.168.0.4
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-87-generic x86_64)
```

- Verificación directorio .ssh:

```
root@servidor01:/home/sysadmin01/.ssh# ls -l
total 4
-rw----- 1 sysadmin01 sudo 736 nov  2 22:36 authorized_keys
```

### Servidor web Nginx:

- Verificación servicio Nginx instalado, habilitado y activo: systemctl status nginx muestra que el servicio está activo y habilitado:

```
sysadmin01@servidor01:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
  Active: active (running) since Thu 2025-11-06 12:36:57 CET; 3min 2s ago
```

- Verificación directorio del sitio web: /var/www/infra-cloud y index.html y style.css con permisos correctos:

```
sysadmin01@servidor01:/var/www/infra-cloud$ ls -l
total 8
-rw-r--r-- 1 sysadmin01 devops 2509 nov  2 22:36 index.html
-rw-r--r-- 1 sysadmin01 devops 1764 nov  2 22:36 style.css
```

- Verificación archivo de configuración de Nginx:  
/etc/nginx/sites-available/infra-cloud:

```
sysadmin01@servidor01:/etc/nginx/sites-available$ ls -l
total 8
-rw-r--r-- 1 root root  2412 nov 30 2023 default
-rw-r----- 1 root devops 265 nov  2 22:36 infra-cloud
```

- Verificación enlace simbólico: /etc/nginx/sites-enabled/infra-cloud y sitios default desactivados:

```
sysadmin01@servidor01:/etc/nginx/sites-enabled$ ls -l
total 0
lrwxrwxrwx 1 root root 38 nov  2 22:36 infra-cloud -> /etc/nginx/sites-available/infra-cloud
```

- Verificación del sitio web desde el host:

Antes, añadimos el alias desde el Linux Mint a la IP del nuevo servidor para acceder mediante IP y mediante alias:

```
GNU nano 7.2                               /etc/hosts
127.0.0.1      localhost
127.0.1.1      inaki-VirtualBox
#192.168.0.2   infracloud.local
192.168.0.4    infracloud.local
192.168.0.2    test.local
```

Desde el host de control accedemos al sitio web:



### Scripts de mantenimiento:

- Verificación de scripts backup.sh, restaurar\_backup.sh y restart\_nginx.sh en /usr/local/bin y son ejecutables:

```
sysadmin01@servidor01:/usr/local/bin$ ls -l
total 12
-rwxr-xr-x 1 root root 551 nov  2 22:36 backup_infra-cloud.sh
-rwxr-xr-x 1 root root 967 nov  2 22:36 restart_nginx.sh
-rwxr-xr-x 1 root root 685 nov  2 22:36 restaurar_backup_infra-cloud.sh
```

- Verificación ejecución scripts:

```
sysadmin01@servidor01:~$ restart_nginx.sh
El servicio está activo. REINICIANDO...
Reinicio completado con éxito.
• Nginx ACTIVO
sysadmin01@servidor01:~$ backup_infra-cloud.sh
Backups creados en /home/sysadmin01/backups
sysadmin01@servidor01:~$ restaurar_backup_infra-cloud.sh
Uso: /usr/local/bin/restaurar_backup_infra-cloud.sh <nombre_del_backup_sin_extension>
Ejemplo: /usr/local/bin/restaurar_backup_infra-cloud.sh infra-cloud_20251001_153000
Últimos backups disponibles:
nginx_20251106_125329.tar.gz
infra-cloud_20251106_125329.tar.gz
```

- Verificación carpeta backups y archivos de backup:

```
sysadmin01@servidor01:~/backups$ ls
infra-cloud_20251106_125329.tar.gz  nginx_20251106_125329.tar.gz
```

- Verificación ejecución periódica:

```
sysadmin01@servidor01:~$ sudo crontab -l
#Ansible: Ejecución diaria de mantenimiento
0 3 * * * /usr/local/bin/backup_infra-cloud.sh && /usr/local/bin/restart_nginx.sh
```

## 3.7. Conclusiones de la fase

La fase de automatización mediante **Ansible** supone un punto de inflexión en la gestión del entorno de sistemas. A través del desarrollo e implementación de roles específicos, se ha conseguido automatizar todos los procesos de configuración y mantenimiento del servidor.

El uso de **infraestructura como código (IaC)** ha permitido transformar tareas tradicionalmente manuales en procesos reproducibles, verificables y escalables. Mejorando la eficiencia, la seguridad, la trazabilidad y la capacidad de mantenimiento del sistema

### 3.7.1. Ventajas obtenidas

El uso de Ansible ha aportado una serie de ventajas a nivel técnico y organizativo:

**Estandarización de la configuración:** todos los servidores compartirán una estructura homogénea y coherente, con los mismos servicios, permisos y configuraciones.

**Reducción del tiempo de despliegue:** los procesos de instalación y configuración se realizan en cuestión de segundos mediante la ejecución de los playbooks correspondientes.

**Eliminación de errores humanos:** la automatización sustituye la intervención manual por código, reduciendo el riesgo de fallos operativos.

**Mantenimiento centralizado y simplificado:** las modificaciones en roles o variables se gestionan automáticamente a todos los servidores.

**Seguridad reforzada:** se automatiza la creación de usuarios, grupos, permisos, acceso SSH y políticas básicas de firewall, minimizando vulnerabilidades.

**Escalabilidad y reproducibilidad:** añadir nuevos servidores o reinstalar entornos se convierte en un proceso predecible y controlado.

**Gestión proactiva del sistema:** el rol de mantenimiento permite programar tareas periódicas (como copias de seguridad y reinicios de servicios), asegurando la continuidad operativa.

En conjunto, la utilización de Ansible ha permitido establecer un entorno automatizado, seguro y fácilmente replicable, cumpliendo con los objetivos de esta fase y proporcionando una infraestructura más eficiente y sostenible.

### **3.7.2. Preparación para la siguiente fase (Terraform y Azure)**

Una vez consolidada la automatización local con Ansible, el proyecto se encuentra preparado para evolucionar hacia una infraestructura gestionada en la nube mediante **Terraform** y **Azure**.

El trabajo realizado en esta fase proporciona una base sólida para la integración entre herramientas, ya que Terraform se encargará de la provisión de recursos, mientras que Ansible continuará desempeñando el rol de configuración y administración de los mismos.

Entre los principales aspectos que facilitarán esta transición destacan:

- La estructura modular de roles y variables facilita la **adaptación directa** a máquinas virtuales y servicios desplegados en Azure.
- La lógica de configuración establecida mediante Ansible puede reutilizarse para **automatizar entornos cloud** sin necesidad de redefinir procesos.
- La documentación obtenida permite una **trazabilidad completa** de la infraestructura, manteniendo la coherencia entre entornos locales y remotos.
- Se establecen las bases de una arquitectura **IaC integral**, en la que Terraform gestionará el ciclo de vida de los recursos y Ansible su configuración detallada.

En definitiva, la finalización de esta fase marca el cierre del despliegue automatizado en entorno local y abre el camino hacia la **infraestructura híbrida y escalable en la nube**, manteniendo como eje principal la automatización, la estandarización y la eficiencia operativa.

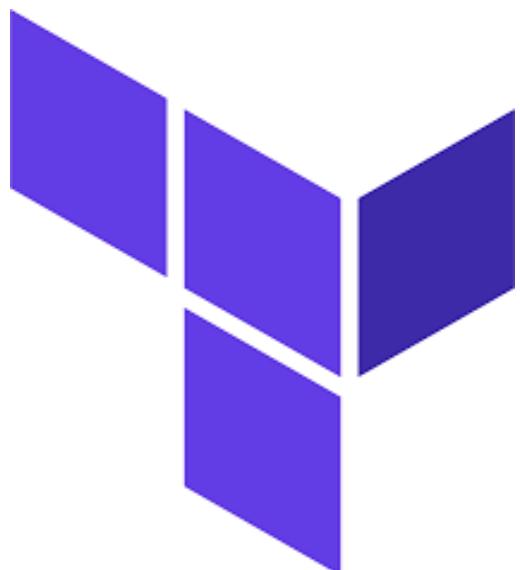
#### **4. Fase 4 - Despliegue en la nube con Terraform y Azure**

Tras la completa automatización del entorno local mediante Ansible, el siguiente paso lógico del proyecto consiste en extender la infraestructura al entorno cloud, adoptando un enfoque de Infraestructura como Código (**IaC**).

Para ello, se empleará **Terraform** como herramienta principal y **Microsoft Azure** como plataforma de servicios en la nube.

El objetivo principal de esta fase es **replicar y desplegar automáticamente** la infraestructura local en un **entorno cloud**, garantizando la portabilidad, escalabilidad y consistencia del sistema.

De este modo, los recursos gestionados se transforman en **recursos virtuales**



**definidos mediante código**, lo que facilita su creación, modificación y eliminación de forma controlada.

Terraform permite describir la infraestructura deseada mediante archivos de configuración declarativos escritos en **HCL (HashiCorp Configuration Language)**. Estos archivos definen los componentes necesarios (redes virtuales, máquinas, grupos de seguridad, discos o direcciones IP) y Terraform se encarga de desplegarlos automáticamente en Azure, asegurando que el estado real coincida con el deseado.

Esta fase, por tanto, no solo implica un cambio de entorno tecnológico, sino también un salto conceptual hacia la **gestión automatizada de la infraestructura en la nube**, reforzando los principios de reproducibilidad, escalabilidad y eficiencia operativa establecidos en fases anteriores.

Entre los objetivos específicos de esta fase se destacan los siguientes:

- Diseñar la estructura base de la infraestructura en Azure (red, subred, grupo de recursos y máquina virtual).
- Implementar los archivos de configuración de Terraform para definir, desplegar y gestionar los recursos cloud.
- Integrar la ejecución de Ansible sobre la máquina virtual creada, asegurando la coherencia con la configuración local.
- Validar la correcta conexión entre el host de control local y la instancia en la nube.
- Documentar el flujo de trabajo completo y los beneficios del despliegue automatizado en la nube.

#### **4.1. Estructura y objetivos de la fase**

La fase de despliegue en la nube se estructura en una serie de pasos progresivos que permiten replicar el entorno automatizado local dentro de **Microsoft Azure**,

utilizando **Terraform** como herramienta principal de gestión de infraestructura.

El propósito general es **definir, crear y configurar todos los recursos cloud de forma automática**, asegurando la trazabilidad y reproducibilidad del entorno.

La estructura de trabajo de esta fase se compone de los siguientes elementos principales:



1. **Configuración** del entorno de **Terraform** en el host de control
2. Creación del **archivo de configuración principal** (main.tf)
3. **Gestión de variables y salida de datos** (variables.tf y outputs.tf)
4. **Archivo de estado** (terraform.tfstate)
5. **Despliegue y validación de recursos**
6. **Integración con Ansible**

## 4.2. Configuración del entorno de Terraform en el host de control

El primer paso para el despliegue de la infraestructura en la nube consiste en preparar el **host de control** desde el cual se gestionan los recursos de Azure mediante Terraform.

### 4.2.1 Creación de una cuenta Free Tier en Azure

Antes de iniciar cualquier configuración de Terraform, es necesario disponer de una cuenta válida en Azure. Voy a usar una cuenta **Free Tier**, la cual permite probar y desplegar recursos sin pagar nada.

## Servicios de Azure



Una vez creada, se obtiene acceso a la suscripción gratuita, con la que Terraform podrá interactuar posteriormente mediante la API de Azure.

### 4.2.2 Instalación de Terraform en el host de control

Con la cuenta de Azure preparada, el siguiente paso consiste en instalar Terraform en el host de control. Terraform es la herramienta de infraestructura como código (IaC) que permitirá definir y desplegar los recursos en Azure.

Primero voy a actualizar el sistema:

```
sudo apt update && sudo apt upgrade -y
```

Después, desde la página oficial de terraform, nos brindan un comando para copiar en nuestra terminal e instalar terraform:

```
inaki@inaki-VirtualBox:~$ wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(grep -oP '(?=<UBUNTU_CODENAME=).*/' /etc/os-release || lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform
```

Este comando se encarga de:

- **Clave GPG (Seguridad)**: Descarga y guarda la clave pública de verificación de HashiCorp
- **Repositorio (Ubicación)**: Crea un archivo que le dice al sistema dónde encontrar los paquetes de Terraform.

- **Ejecutable (Programa):** Descarga e instala el binario terraform, colocándolo en un directorio estándar del sistema (usualmente /usr/bin/), dejándolo listo para ejecutar sin moverlo manualmente.

Ahora voy a verificar la instalación para confirmar que Terraform está listo para usar.

```
inaki@inaki-VirtualBox:~$ terraform -version
Terraform v1.13.5
on linux_amd64
```

#### 4.2.3 Configuración del proveedor de Azure y autenticación

Terraform necesita autenticarse en Azure para poder gestionar recursos.

##### Autenticación interactiva mediante Azure CLI:

Primero, voy a instalar la **Azure CLI**:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Ahora hago inicio de sesión con la cuenta Free Tier:

```
inaki@inaki-VirtualBox:~$ az login
```

Esto abrirá una ventana del navegador donde se debe iniciar sesión con la cuenta creada. Tras autenticarse, la CLI guarda un token válido en el host de control.

```
Retrieving tenants and subscriptions for the selection...
[Tenant and subscription selection]
No   Subscription name   Subscription ID           Tenant
-----
[1] * Azure subscription 1  9747e312-3f8b-4de2-af52-b1f56c1d3eaf  Directorio predeterminado

The default is marked with an *; the default tenant is 'Directorio predeterminado' and subscription is 'Azure subscription 1' (9747e312-3f8b-4de2-af52-b1f56c1d3eaf).
```

#### 4.2.4 Estructura del entorno de Terraform en el host de control

Voy a crear un directorio dentro del host de control para centralizar toda la configuración de Terraform:

```
inaki@inaki-VirtualBox:~$ mkdir -p ~/terraform/azure
```

Dentro de este directorio se organizan los archivos principales:

- **main.tf**: define los recursos de infraestructura (VM's, redes, grupos de seguridad).
- **variables.tf**: contiene las variables configurables de la infraestructura.
- **outputs.tf**: define las salidas de Terraform tras aplicar los cambios.

```
inaki@inaki-VirtualBox:~/terraform/azure$ touch main.tf variables.tf outputs.tf
```

- **terraform.tfstate**: archivo que mantiene el estado de la infraestructura.

Esta estructura facilita la modularidad, trazabilidad y reutilización de la configuración, permitiendo que se mantenga un flujo de trabajo ordenado y reproducible.

```
inaki@inaki-VirtualBox:~/terraform/azure$ tree
.
├── main.tf
└── outputs.tf
    └── variables.tf
1 directory, 3 files
```

**terraform.tfstate** se genera automáticamente.

#### 4.2.5 Validación del entorno de Terraform

Antes de desplegar recursos, es recomendable validar que Terraform reconoce correctamente el proveedor de Azure y la autenticación:

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform init
Initializing the backend...
Initializing provider plugins...
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.
```

**terraform init**, inicia el directorio de trabajo, descarga los plugins necesarios (en este caso, el proveedor de Azure) y verifica la conectividad con la API de Azure.

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform plan
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes
are needed.
```

**terraform plan**, muestra el plan de ejecución de Terraform, indicando los recursos que se crearán, modificarán o eliminarán. Permite comprobar que la sintaxis de los archivos **.tf** es correcta y que la autenticación con Azure funciona correctamente. De momento al no haber nada

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform apply
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
inaki@inaki-VirtualBox:~/terraform/azure$ ls
main.tf  outputs.tf  terraform.tfstate  variables.tf
```

**terraform apply**, crea los recursos en la nube y luego crea/actualiza el archivo `terraform.tfstate`.

En este punto, el host de control está preparado para:

- **Ejecutar Terraform** en Linux Mint.
- **Autenticarse** correctamente con la cuenta Free Tier de Azure.
- **Mantener una estructura organizada** de archivos de configuración.
- **Validar la conectividad** con Azure antes de desplegar cualquier recurso.

Con esto se garantiza que la fase de despliegue de infraestructura en la nube pueda iniciarse sin problemas y con total trazabilidad de cambios.

#### 4.3 Creación del archivo de configuración principal (`main.tf`)

El archivo **main.tf** es el núcleo de Terraform, donde definimos todos los recursos que queremos desplegar en Azure. Para nuestro caso, vamos a desplegar una máquina virtual Ubuntu 24.04 LTS, que será posteriormente gestionada mediante Ansible.

En el host de control, dentro del directorio de Terraform que creamos, voy a escribir todo el contenido de la infraestructura:

```
~/terraform/azure$ nano main.tf
```

#### 4.3.1. Proveedor de Azure

Para usar terraform, voy a usar **Virtual Studio Code** como editor de texto.

Primero debemos indicar a Terraform que usaremos **Azure** como proveedor:

```
# 1. Configuración del proveedor de Azure
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.107.0"
    }
  }

  required_version = ">= 1.5.0"
}

provider "azurerm" {
  features {}
  subscription_id = "9747e312-3f8b-4de2-af52-b1f56c1d3eaf"
}
```

**Explicación:**

Bloque **terraform**:

- Le indica a Terraform que necesita descargar el plugin (**azurerm**) de HashiCorp.
- Asegura que se use una versión compatible con la serie **3.107.0**.

Bloque **provider "azurerm"**:

- Es la instancia de conexión real a Azure.
- **features {}** activa el uso de las últimas características del proveedor Azure.
- **subscription\_id** es el numero de mi suscripción en azure.

#### 4.3.2. Grupo de recursos

En Azure, todos los recursos deben estar dentro de un grupo de recursos:

```
# 2. Grupo de recursos (contenedor de todos los elementos)
resource "azurerm_resource_group" "rg_tfg" {
    name      = "rg-tfg"
    location  = "West Europe"
}
```

#### Explicación:

Este bloque crea un contenedor en Azure llamado **rg-tfg** en la región **West Europe**. Este Grupo de Recursos es el lugar lógico donde se almacenarán la máquina virtual y todos sus componentes de red.

#### 4.3.3. Red virtual y subred

Necesitamos una **red interna** para la VM:

```
# 3. Red virtual (VNet)
resource "azurerm_virtual_network" "vnet_tfg" {
    name          = "vnet-tfg"
    address_space = ["10.0.0.0/16"]
    location      = azurerm_resource_group.rg_tfg.location
    resource_group_name = azurerm_resource_group.rg_tfg.name
}

# 4. Subred dentro de la red virtual
resource "azurerm_subnet" "subnet_tfg" {
    name          = "subnet-tfg"
    resource_group_name = azurerm_resource_group.rg_tfg.name
    virtual_network_name = azurerm_virtual_network.vnet_tfg.name
    address_prefixes   = ["10.0.1.0/24"]
}
```

#### Explicación:

Este código se encarga de definir la red privada de tu infraestructura en Azure.

Red Virtual (**azurerm\_virtual\_network**): Crea el contenedor de red llamado **vnet-tfg**. Es el límite de aislamiento para toda tu red. Se le asigna un gran espacio de direcciones IP privadas: **10.0.0.0/16**. La ubicación se hereda del Grupo de Recursos (**rg\_tfg**).

Subred (**azurerm\_subnet**): Crea una división interna dentro de la VNet, llamada **subnet-tfg**. A esta subred se le asigna un rango más pequeño de direcciones IP privadas: **10.0.1.0/24**.

#### 4.3.4. IP Pública

```
# 5. IP pública para la máquina virtual
resource "azurerm_public_ip" "ip_publica_tfg" {
  name          = "ip-publica-tfg"
  location      = azurerm_resource_group.rg_tfg.location
  resource_group_name = azurerm_resource_group.rg_tfg.name
  allocation_method  = "Static"
  sku            = "Standard"
}
```

#### Explicación:

Este bloque crea un recurso de dirección IP pública en Azure llamado **ip-publica-tfg**.

- **allocation\_method = "Static"**: La dirección IP real se asignará fija cuando se despliegue.
- **sku = "Standard"**: Especifica el nivel de IP pública.

#### 4.3.5. Interfaz de red (NIC)

```
# 6. Interfaz de red (NIC)
resource "azurerm_network_interface" "nic_tfg" {
  name          = 'nic-tfg'
  location      = azurerm_resource_group.rg_tfg.location
  resource_group_name = azurerm_resource_group.rg_tfg.name

  ip_configuration {
    name                = "ipconfig-tfg"
    subnet_id          = azurerm_subnet.subnet_tfg.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id       = azurerm_public_ip.ip_publica_tfg.id
  }
}

# Asociación NSG a la NIC
resource "azurerm_network_interface_security_group_association" "nic_nsg_assoc_tfg" {
  network_interface_id    = azurerm_network_interface.nic_tfg.id
  network_security_group_id = azurerm_network_security_group.nsg_tfg.id
}
```

#### Explicación:

Este bloque crea la tarjeta de red virtual (**nic-tfg**) que la máquina virtual usará para conectarse a la red.

La clave está en el bloque **ip\_configuration**, que enlaza la NIC a:

- Subred: Se conecta a la subred previamente creada (**subnet-tfg**).
- IP Privada: Se le asigna una IP privada dinámica dentro de esa subred.
- IP Pública: Se le adjunta la IP pública del bloque anterior (**ip-publica-tfg**), haciendo que la VM sea accesible desde Internet.

#### 4.3.6. Grupo de seguridad de red (NSG)

```
# 7. Grupo de seguridad de red (NSG) con reglas SSH y HTTP
resource "azurerm_network_security_group" "nsg_tfg" {
  name          = "nsg-tfg"
  location      = azurerm_resource_group.rg_tfg.location
  resource_group_name = azurerm_resource_group.rg_tfg.name

  security_rule {
    name          = "SSH"
    priority      = 1001
    direction     = "Inbound"
    access        = "Allow"
    protocol      = "Tcp"
    source_port_range = "*"
    destination_port_range = "22"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name          = "HTTP"
    priority      = 1002
    direction     = "Inbound"
    access        = "Allow"
    protocol      = "Tcp"
    source_port_range = "*"
    destination_port_range = "80"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }
}
```

```
# Asociar NSG a la subred
resource "azurerm_subnet_network_security_group_association" "nsg_assoc_tfg" {
  subnet_id      = azurerm_subnet.subnet_tfg.id
  network_security_group_id = azurerm_network_security_group.nsg_tfg.id
}
```

#### Explicación:

Este código crea y aplica un firewall para proteger tu máquina virtual en Azure.

- ***azurerm\_network\_security\_group (nsg\_tfg)***: Define el firewall. Contiene dos reglas clave:
  - Regla **SSH** (Puerto 22): Permite la conexión remota segura (administración) desde cualquier lugar.
  - Regla **HTTP** (Puerto 80): Permite el tráfico web (servidor HTTP) desde cualquier lugar.
- ***azurerm\_subnet\_network\_security\_group\_association***: Aplica este firewall a la subred, asegurando que tu VM use estas reglas de seguridad.

Estas líneas abren los puertos 22 y 80 al público para la VM y cierran el resto por defecto.

#### 4.3.7. Máquina virtual

```
# 8. Máquina virtual Ubuntu 24.04
resource "azurerm_linux_virtual_machine" "vm_tfg" {
  name          = "vm-tfg"
  resource_group_name = azurerm_resource_group.rg_tfg.name
  location       = azurerm_resource_group.rg_tfg.location
  size          = "Standard_B2s"
  admin_username = "inaki"

  network_interface_ids = [
    azurerm_network_interface.nic_tfg.id
  ]

  admin_ssh_key {
    username  = "inaki"
    public_key = file("~/ssh/id_rsa.pub")
  }

  os_disk {
    caching           = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "ubuntu-24_04-lts"
    sku       = "server"
    version   = "latest"
  }

  disable_password_authentication = true
}
```

**Explicación:**

Este código define una VM Ubuntu 24.04 con las siguientes características:

- Nombre/Tamaño: Se llama **vm-tfg** y usa el tamaño económico **Standard\_B1s**.
- Red: La conecta a la Interfaz de Red (**nic\_tfg**).
- SO y Disco: Especifica que el sistema operativo es **Ubuntu 20.04 LTS** y que usará un disco estándar (**Standard\_LRS**).
- Acceso: Configura el usuario administrador **inaki** y habilita el acceso exclusivamente por clave **SSH**, desactivando las contraseñas para mayor seguridad.

En resumen, es la instrucción para Azure para crear el servidor Ubuntu funcional dentro de mi red, listo para ser usado con mi clave SSH.

#### 4.4 Gestión de variables y salida de datos

No vamos a usar **variables.tf**, ya que mi **main.tf** usa valores fijos.

Sí crearé un archivo **outputs.tf**, para ver los datos clave cuando despleguemos (como la IP pública, el nombre de la VM, etc.). Así podré usar esos datos después para conectarme por SSH y ejecutar playbooks de Ansible.

##### 4.4.1. Outputs.tf

Ese bloque de código define los output (salidas) de Terraform. Su única función es mostrar información importante en la terminal una vez que la infraestructura en Azure se haya creado o modificado exitosamente con **terraform apply**.

```

# Descripción: Muestra la información clave del despliegue
# tras ejecutar Terraform.

# Dirección IP pública de la máquina virtual
output "public_ip_address" {
  description = "Dirección IP pública de la máquina virtual"
  value       = azurerm_public_ip.ip_publica_tfg.ip_address
}

# Nombre de la máquina virtual
output "vm_name" {
  description = "Nombre de la máquina virtual creada"
  value       = azurerm_linux_virtual_machine.vm_tfg.name
}

# Nombre del grupo de recursos
output "resource_group_name" {
  description = "Nombre del grupo de recursos"
  value       = azurerm_resource_group.rg_tfg.name
}

```

## Explicación

Los bloques output son esenciales para que sepa cómo acceder a los recursos creados.

- **output "public\_ip\_address"**: Muestra la dirección IP pública de la máquina virtual. Es la IP que usaré para conectarme por SSH (o HTTP).
- **output "vm\_name"**: Muestra el nombre de la máquina virtual creada (**vm-tfg**).
- **output "resource\_group\_name"**: Muestra el nombre del grupo de recursos que contiene la infraestructura (**rg-tfg**).

## 4.5 Despliegue y validación de recursos con Terraform

Voy a crear toda la infraestructura definida en **main.tf** usando Terraform desde mi host de control Linux Mint.

Primero hago el login:

```

inaki@inaki-VirtualBox:~/terraform/azure$ az login
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser. If no web browser is available or if the web browser fails to open, use device code flow with `az login --use-device-code`.

Retrieving tenants and subscriptions for the selection...

[Tenant and subscription selection]

No   Subscription name   Subscription ID           Tenant
----- *   Azure subscription 1   9747e312-3f8b-4de2-af52-b1f56c1d3eaf   Directorio predeterminado

The default is marked with an *, the default tenant is 'Directorio predeterminado' and subscription is 'Azure subscription 1' (9747e312-3f8b-4de2-af52-b1f56c1d3eaf).

Select a subscription and tenant (Type a number or Enter for no changes):

Tenant: Directorio predeterminado
Subscription: Azure subscription 1 (9747e312-3f8b-4de2-af52-b1f56c1d3eaf)

[Announcements]
With the new Azure CLI login experience, you can select the subscription you want to use more easily. Learn more about it and its configuration at https://go.microsoft.com/fwlink/?linkid=2271236

If you encounter any problem, please open an issue at https://aka.ms/azclibug

[Warning] The login output has been updated. Please be aware that it no longer displays the full list of available subscriptions by default.

```

## Inicializar el entorno de Terraform (terraform init)

Desde el directorio donde está mi proyecto:

```

inaki@inaki-VirtualBox:~/terraform/azure$ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Using previously-installed hashicorp/azurerm v4.52.0

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.

```

Con este comando, se descarga el proveedor de Azure (azurerm), prepara el directorio para poder ejecutar plan y apply y crea un archivo oculto con la configuración del proveedor.

## Comprobar el plan de ejecución

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
+ create

Terraform will perform the following actions:

# azurerm_linux_virtual_machine.vm_tfg will be created
+ resource "azurerm_linux_virtual_machine" "vm_tfg" {
    + admin_username = "inaki"
    + allow_extension_operations = (known after apply)
    + bypass_platform_safety_checks_on_user_schedule_enabled = false
    + computer_name = (known after apply)
    + disable_password_authentication = true
    + disk_controller_type = (known after apply)
    + extensions_time_budget = "PT1H30M"
    + id = (known after apply)
    + location = "eastus"
```

(en la foto solo he puesto lo primero que devuelve el prompt. Los recursos ocupan mucho más.)

Con este comando, analiza **main.tf** y **outputs.tf** y Muestra qué recursos se van a crear, modificar o eliminar.

### Aplicar los cambios (crear la infraestructura)

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform apply
```

Con este comando, Terraform mostrará un resumen del plan y pedirá confirmación. Terraform empezará a crear todo en Azure. Se generará automáticamente el archivo **terraform.tfstate** con el estado actual de la infraestructura. Al finalizar, se mostrarán los outputs.

```
inaki@inaki-VirtualBox:~/terraform/azure$ terraform apply
azurerm_resource_group.rg_tfg: Refreshing state... [id=/subscriptions/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg]
azurerm_network_security_group.nsg_tfg: Refreshing state... [id=/subscriptions/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/networkSecurityGroups/nsg-tfg]
azurerm_public_ip.ip_publica_tfg: Refreshing state... [id=/subscriptions/9747e312-3fb8-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/publicIPAddresses/ip-publica-tfg]
azurerm_virtual_network.vnet_tfg: Refreshing state... [id=/subscriptions/9747e312-3fb8-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/virtualNetworks/vnet-tfg]
azurerm_subnet.subnet_tfg: Refreshing state... [id=/subscriptions/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/virtualNetworks/subnets/subnet-tfg]
azurerm_network_interface.nic_tfg: Refreshing state... [id=/subscriptions/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/networkInterfaces/nic-tfg]
azurerm_subnet_network_security_group_association.nsg_assoc_tfg: Refreshing state... [id=/subscriptions/9747e312-3fb8-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/virtualNetworks/vnet-tfg/subnets/subnet-tfg]
azurerm_linux_virtual_machine.vm_tfg: Refreshing state... [id=/subscriptions/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Compute/virtualMachines/vm-tfg]
azurerm_network_interface_security_group_association.nic_nsg_assoc_tfg: Refreshing state... [id=/subscriptions/9747e312-3fb8-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/networkInterfaces/nic-tfg/resourceGroups/9747e312-3f8b-4de2-af52-b1f56c1d3eaf/resourceGroups/rg-tfg/providers/Microsoft.Network/networkSecurityGroups/nsg-tfg]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

resource_group_name = "rg-tfg"
vm_name = "vm-tfg"
vm_public_ip = "4.180.56.170"
```

### Validación de recursos en Azure

Aunque todo se haya creado con Terraform, es recomendable verificar en la interfaz gráfica:

Voy a abrir el portal de Azure.

The screenshot shows the Azure Resource Groups blade. At the top, there are buttons for 'Exportar grupos de recursos con Bicep o Terraform' and 'Resumir mis costos por servicio'. Below that, it says 'Directorio predeterminado (pinaki96hotmail.onmicrosoft.com)'. There are buttons for '+ Crear', 'Administrar vista', 'Actualizar', 'Exportar a CSV', 'Abrir consulta', and 'Asignar etiquetas'. A message box says 'Está viendo una nueva versión de la experiencia de exploración. Haga clic aquí para acceder a la experiencia anterior.' Below the message, there are filters: 'Filtrar por cualquier ca...', 'Suscripción es igual que todo', 'Ubicación es igual que todo', and '+ Agregar filtro'. The main table lists two resource groups:

	Nombre	Suscripción	Ubicación
<input type="checkbox"/>	NetworkWatcherRG	... Azure subscription 1	East US
<input type="checkbox"/>	rg-tfg	... Azure subscription 1	West Europe

En grupos de recursos compruebo que está: **rg-tfg** y los demás recursos:

- **vm-tfg**: máquina virtual Ubuntu 24.04.
- **ip-publica-tfg**: dirección IP pública.
- **vnet-tfg**: red virtual.
- **nsg-tfg**: grupo de seguridad con reglas SSH y HTTP.
- **nic-tfg**: interfaz de red asociada.

The screenshot shows the 'rg-tfg' resource group blade. On the left, there's a sidebar with 'Información general', 'Registro de actividad', 'Control de acceso (IAM)', 'Etiquetas', 'Visualizador de recursos', 'Eventos', 'Configuración', 'Administración de costos', 'Supervisión', 'Automation', and 'Ayuda'. The main area shows 'Essentials' information: Suscripción (mover) to 'Azure subscription 1', Implementaciones 'Sin implementaciones', Id. de suscripción '9747e312-3fb4-4de2-af52-b1f56c1d3eaf', Ubicación 'West Europe', and Etiquetas (editar). Below that is the 'Recursos' tab, which lists the resources created by Terraform:

	Nombre	Tipo	Ubicación
<input type="checkbox"/>	ip-publica-tfg	Dirección IP pública	West Europe
<input type="checkbox"/>	nic-tfg	Interfaz de red	West Europe
<input type="checkbox"/>	nsg-tfg	Grupo de seguridad	West Europe
<input type="checkbox"/>	vm-tfg	Máquina virtual	West Europe
<input type="checkbox"/>	vm-tfg_disk1_308ca562d6c946c1bea066	Disco	West Europe
<input type="checkbox"/>	vnet-tfg	Red virtual	West Europe

**Verificar IP pública accesible:**

Ahora desde mi host de control voy a acceder a la máquina virtual en la nube que acabo de crear mediante SSH (La IP es la que nos devolvió Azure tras ejecutar el terraform apply):

```
inaki@inaki-VirtualBox:~/terraform/azure$ ssh inaki@4.180.56.170
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1012-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
```

Ya estaría todo creado y comprobado el acceso SSH desde el host de control.

#### 4.6 Logros alcanzados con Terraform y Azure

Con la ejecución de esta fase se ha conseguido implementar un entorno de infraestructura automatizado completamente funcional en la nube, utilizando **Terraform** y **Azure Free Tier**. La principal consecución ha sido la creación reproducible de una VM Linux Ubuntu 24.04.3, junto con todos los recursos de red necesarios para su operación, incluyendo red virtual, subred, IP pública y grupo de seguridad con reglas básicas de acceso (SSH y HTTP). Esto permite que la máquina esté accesible de manera segura desde el host de control, facilitando la ejecución de tareas de administración y despliegue mediante Ansible.

A esta VM en azure, se le puede aplicar automáticamente la configuración completa de usuarios, permisos, servidor web Nginx, archivos del sitio, scripts de mantenimiento y políticas de seguridad, sin necesidad de intervención manual, con **Ansible**. En otras palabras, todo el entorno, desde la creación de recursos hasta la puesta en marcha del servidor de aplicaciones, puede ser reproducido de manera automática en cualquier momento, garantizando consistencia y reducción de errores humanos.

Este enfoque tiene aplicaciones muy amplias en entornos profesionales. La combinación de Terraform y Ansible permite separar el levantar la **infraestructura** de la **configuración**, siguiendo buenas prácticas de DevOps y automatización.

En términos de innovación y relevancia, el trabajo realizado demuestra cómo se puede lograr un **despliegue completo y reproducible de infraestructura en la**

**nube** utilizando herramientas modernas, evitando configuraciones manuales, mejorando la trazabilidad de cambios y facilitando la escalabilidad.

## 4.7. Despliegue de configuración con Ansible sobre la VM de Azure

Una vez provisionada la infraestructura en Azure mediante Terraform, el siguiente paso consiste en aplicar la configuración automatizada previamente desarrollada con Ansible sobre la máquina virtual recién creada. Esto permite llevar la VM desde un estado básico a un entorno operativo completo, tal como se definió en las fases anteriores del proyecto.

### 4.7.1 Configuración del inventario de Ansible

Para que Ansible pueda gestionar la VM en Azure, se actualiza el archivo **inventory.ini** en el host de control. Se incluye la dirección IP pública asignada por Terraform y el usuario configurado durante el despliegue:

```
GNU nano 7.2                                         inventory.ini *
[servidores]
#ubuntu_server ansible_host=192.168.0.2 ansible_user=sysadmin01
#ubuntu_server02 ansible_host=192.168.0.4 ansible_user=inaki
azure_ubuntu ansible_host=4.180.56.170 ansible_user=inaki ansible_ssh_private_key_file=~/ssh/id_rsa
```

**ansible\_ssh\_private\_key\_file**: es ruta a la clave privada que corresponde a la clave pública utilizada para SSH en Terraform.

### 4.7.2 Ejecución de los playbooks

Con el inventario configurado, se pueden ejecutar los playbooks replicando exactamente el entorno que se montó en las VMs locales:

```
~/ansible$ ansible-playbook -i inventory.ini playbooks/site.yml --ask-become-pass
```

Este comando conecta al servidor remoto mediante SSH. Aplica todos los roles: **entorno\_basico, usuarios, nginx y mantenimiento**.

Es importante destacar que, gracias a la **clave SSH configurada durante Terraform**, no es necesario introducir contraseñas manualmente, garantizando un

despliegue seguro.

```
PLAY RECAP ****
azure_ubuntu      : ok=24   changed=22   unreachable=0   failed=0    skipped=0   rescued=0    ignored=0
```

**Beneficios obtenidos:** Al aplicar Ansible sobre la VM de Azure se consiguen varias ventajas:

1. **Reproducibilidad:** cualquier cambio en los playbooks puede aplicarse a nuevas VMs sin intervención manual.
2. **Consistencia:** se asegura que la configuración del servidor, Nginx, scripts, permisos y resto de configuraciones es idéntica a la del entorno local.
3. **Automatización completa:** la infraestructura y la configuración se gestionan de manera centralizada desde el host de control.
4. **Preparación para escalabilidad:** nuevas máquinas virtuales pueden añadirse simplemente actualizando el inventario y aplicando los mismos playbooks.

#### 4.7.3. Comprobaciones finales\*

\* En la presentación del proyecto voy a proyectar un video con comprobaciones finales. La VM creada con Azure es una réplica exacta en la nube de las que teníamos en local.

# Calidad y Evaluación del producto (RA4)

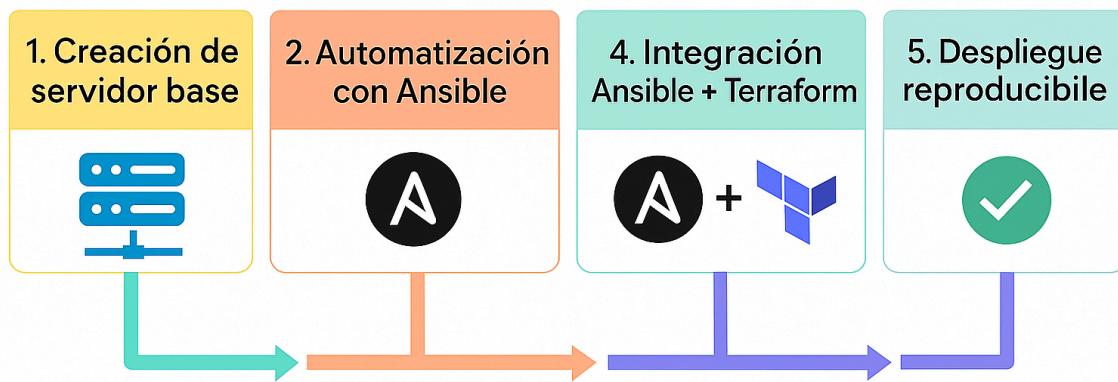
---

## Propuesta de soluciones y justificación

Los objetivos centrales del proyecto han sido resueltos mediante la integración de **Terraform** y **Ansible** en un flujo de trabajo DevOps, creando una solución única que trata tanto el aprovisionamiento de infraestructura como su configuración interna.

Objetivo	Justificación	Aplicación
<b>Creación de servidor reproducible</b>	Permite tener una configuración estándar. Así se consigue que todo sea uniforme y sirve de referencia práctica.	Configuración manual de Ubuntu Server y Nginx en VirtualBox para validar la configuración final requerida antes de automatizarla.
<b>Automatización con Ansible</b>	Configura múltiples servidores de forma reproducible, segura y automática, minimizando errores y garantizando coherencia.	Configuración de usuarios, grupos y permisos, firewall, servicios web, scripts de mantenimiento y otras configuraciones avanzadas.
<b>Infraestructura como código con Terraform (azure)</b>	Permite declarar la infraestructura de Azure en forma de código, evitando errores de aprovisionamiento manual y facilitando la trazabilidad.	Creación de grupo de recursos, red virtual, subred, NSG, IP pública, NIC y VM Ubuntu.
<b>Integración Ansible + Terraform</b>	Terraform crea la infraestructura y Ansible la configura, asegurando que el despliegue es de inicio a final reproducible.	Después de desplegar la VM en Azure, Ansible configura automáticamente el entorno básico, Nginx, usuarios y scripts de mantenimiento.

La combinación de estas fases permite cumplir todos los objetivos con un enfoque profesional y replicable:



### **Definición del procedimiento de evaluación del proyecto**

La evaluación del proyecto se hará siguiendo un proceso que permita verificar la eficacia y la correcta implementación de las soluciones. El objetivo principal es asegurar que cada fase del proyecto cumple con los objetivos y los resultados son reproducibles y fiables.

El procedimiento de evaluación se desarrollará en varias etapas:

#### **1. Verificación de la infraestructura local:**

- Se comprobará la **correcta instalación de Ubuntu Server y Nginx**
- Se comprobará que los **servicios básicos y del servidor web funcionen** correctamente.
- Se comprobará la **existencia de los usuarios, grupos y permisos** definidos.

#### **2. Prueba de automatización con Ansible:**

- Se **ejecutarán los playbooks** desarrollados para la configuración de los servidores.

- Se evaluará que **todos los roles** (entorno básico, usuarios, Nginx, mantenimiento) **se aplican correctamente**, que los **archivos y directorios** tienen los **permisos adecuados** y que los **servicios se encuentran activos**.
- Se comprobará la **capacidad del servidor de ser reproducible**, verificando que la configuración aplicada en diferentes servidores genera entornos idénticos.

### **3. Evaluación de la infraestructura como código con Terraform:**

- Se desplegará una **máquina virtual en Azure siguiendo la configuración declarada en los archivos de Terraform** (main.tf, variables.tf, outputs.tf).
- Se **comprobará la creación correcta de todos los recursos**: grupo de recursos, red virtual, subred, NSG, IP pública, NIC y máquina virtual.
- Se verificará que la **infraestructura desplegada permite la integración con Ansible**, asegurando la automatización completa del entorno.

### **4. Integración Ansible + Terraform:**

- Se ejecuta **Ansible sobre la infraestructura desplegada en Azure** para garantizar que la configuración, los usuarios, servicios y scripts de mantenimiento se apliquen correctamente.
- Se **validará que los cambios se realizan sin errores** y que la infraestructura y la configuración cumplen con los objetivos definidos.

De esta forma, el procedimiento de evaluación asegura que el proyecto se puede validar de manera objetiva, garantizando la calidad del proceso y del producto final, y proporciona un control que permite detectar y corregir errores antes de su implementación definitiva en entornos de producción.

#### **Determinación de las variables susceptibles de evaluación**

Para que la evaluación del proyecto sea objetiva y sistemática, he definido una serie de variables que ayudarán a medir la eficacia, la reproducibilidad y la calidad de lo que he hecho:

### 1. Variables de infraestructura y despliegue:

- **Disponibilidad de recursos en Azure:** Comprobación de los elementos de la infraestructura (*grupo de recursos, red virtual, subred, NSG, IP pública, NIC y VM*).
- **Configuración de la máquina virtual:** Verificación de que el sistema operativo, tamaño y configuración de la VM son correctos según lo definido en *main.tf*.
- **Integridad de la infraestructura como código:** Confirmación de que los archivos de Terraform no presentan errores de sintaxis y que *terraform plan* muestra los cambios esperados sin generar modificaciones inesperadas.

### 2. Variables de configuración del sistema con Ansible:

- **Usuarios y permisos:** Existencia de los usuarios y grupos configurados, correcta asignación de permisos y pertenencia a grupos correspondientes.
- **Servicios activos:** Estado de los servicios después de ejecutar los *playbooks*.
- **Directorio y archivos web:** Comprobación de que los directorios del sitio web y los archivos estáticos (*index.html, style.css*) se encuentran correctamente ubicados con permisos adecuados.
- **Scripts de mantenimiento:** Disponibilidad y funcionalidad de los scripts de backup, restauración y reinicio, así como su ejecución periódica programada.

### 3. Variables de automatización y reproducibilidad:

- **Ejecución sin errores:** Registro de tareas Ansible y Terraform completadas correctamente (*ok/changed*) en diferentes entornos.

- **Uniformidad entre entornos:** Comparación de configuraciones entre distintos servidores desplegados para garantizar que los entornos son idénticos.
- **Tiempo de despliegue:** Medición del tiempo requerido para aprovisionar la infraestructura y aplicar la configuración completa.

#### **4. Variables de documentación y trazabilidad:**

- **Registro de resultados:** Disponibilidad de *logs*, capturas de pantalla y archivos de salida que permitan verificar la ejecución correcta de cada fase.
- **Cumplimiento de objetivos:** Evaluación de si cada solución implementada cumple con los objetivos definidos en la fase de planificación del proyecto.

#### **Metodología para la documentación del proyecto:**

Para evaluar de manera objetiva la ejecución y resultados del proyecto, he documentado todo con una metodología sistemática basada en “**captura tras comando**”.

Entre la documentación que se ha aportado junto al proyecto puedo destacar:

- **Terraform:** Archivos de configuración (*main.tf*, *variables.tf*, *outputs.tf*), estado de recursos (*terraform.tfstate*) y registros de ejecución (*init*, *plan*, *apply*), junto con capturas del portal de Azure que muestran la VM y sus recursos.
- **Ansible:** Playbooks y roles completos, registros de ejecución de tareas, configuración de usuarios, servicios y Nginx, scripts de mantenimiento funcionando.
- **Resultados finales:** (*video en la presentación*) Comprobación del acceso web al sitio desplegado, verificación de directorios y permisos, y consistencia entre entornos local y en la nube

Reitero que la fase de **planificación de la ejecución del proyecto**, es básicamente una guía completa y exhaustiva diseñada para que cualquier persona con nociones

de informática en cloud computing, devOps o administración de sistemas, pueda replicar el proyecto fielmente gracias a esta **documentación de alta fidelidad**.

### **Control de calidad de proceso y producto final**

Para garantizar la correcta ejecución del proyecto y la calidad del producto final, destaco los siguientes criterios de control:

#### **Proceso de desarrollo y despliegue:**

- Validación mediante comprobaciones (*ansible-playbook --check*, *terraform plan*).
- *Logs* y registros de ejecución para detectar errores durante la automatización.

#### **Producto final:**

- Verificación de que los servidores se despliegan correctamente tanto en el entorno local como en Azure.
- Comprobación de roles y de la configuración que estos mismos han realizado.
- Confirmación de que los scripts de mantenimiento funcionan y se ejecutan.
- Acceso web a los sitios desplegados (disponibilidad y contenido correcto).

#### **Trazabilidad y consistencia:**

- Todos los cambios en Terraform y Ansible quedan documentados.
- Comparación entre configuraciones locales y en la nube para asegurar uniformidad y reproducibilidad.

El proyecto se considera satisfactorio cuando los servidores se despliegan sin errores, la configuración es coherente y las pruebas funcionales se completan correctamente.

# Conclusión del proyecto

---

El proyecto desarrollado demuestra la **viabilidad técnica y operativa de construir una infraestructura completa mediante herramientas de automatización e infraestructura como código**. A lo largo del trabajo se ha conseguido desplegar, configurar y validar un entorno reproducible tanto en local como en la nube, siguiendo prácticas profesionales utilizadas actualmente en el sector IT.

En primer lugar, la creación de un **servidor base en VirtualBox** permitió definir y documentar los requisitos funcionales del sistema de manera controlada. Este entorno inicial sirvió como referencia para elaborar los **playbooks de Ansible**, asegurando que la configuración fuera coherente y estable.

La posterior **automatización** demostró que es posible levantar un **servidor completamente operativo sin intervención manual**, reduciendo errores y aumentando la eficiencia en los despliegues, además de reducir considerablemente los tiempos de despliegue, pasando de horas, si hablamos de configuración manual, a minutos si tenemos todo automatizado de una manera correcta.

Posteriormente, **Terraform permitió trasladar todo este entorno a la nube de Azure**, garantizando que la infraestructura se declarara como código y pudiera reproducirse de forma segura y verificable. La integración entre **Terraform (aprovisionamiento) y Ansible (configuración)** completó el ciclo de **automatización, obteniendo un sistema totalmente desplegable desde cero mediante comandos simples y predecibles**.

El proyecto valida que los métodos utilizados (**virtualización, automatización de configuración, infraestructura como código y despliegue en nube**) no sólo son técnicamente factibles si se usan en conjunto a la hora de levantar servidores completamente estables, sino que representan **la transformación digital** y se están consolidando rápidamente como el estándar actual que definirá el futuro inmediato en las empresas del sector.

La infraestructura creada cumple con los requisitos de **estabilidad, escalabilidad, control de calidad y trazabilidad** marcados inicialmente.

Finalmente, el conjunto de pruebas realizadas confirma que **el sistema es funcional, seguro y repetible**. Por tanto, el proyecto cumple adecuadamente con los objetivos planteados y constituye una base sólida para futuras ampliaciones, convirtiéndose en un proyecto modular que siempre puede ser mejorado o incluir más servicios y configuraciones más complejas.

# Bibliografía y referencias

---

Para la realización de este proyecto, se ha tenido en cuenta la documentación técnica y los recursos oficiales que se listan a continuación. Estos materiales han constituido la **principal referencia para la implementación del proyecto**. La consulta de esta documentación se ha realizado de forma continuada a lo largo de la ejecución del proyecto (de septiembre a noviembre de 2025). Para su presentación, las referencias se han organizado alfabéticamente y según las directrices del formato APA.

**Ansible (Red Hat).** (2025). *Ansible Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://docs.ansible.com/>

**Canonical Ltd.** (2025). *Ubuntu Server 24.04 LTS Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://ubuntu.com/server/docs>

**HashiCorp.** (2025). *Terraform Azure Provider Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de  
<https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs>

**HashiCorp.** (2025). *Terraform Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://developer.hashicorp.com/terraform/docs>

**Microsoft Azure.** (2025). *Azure Free Tier – Free Account and Included Services*. Recuperado entre Septiembre y Noviembre de 2025, de  
<https://azure.microsoft.com/free/>

**Microsoft Azure.** (2025). *Azure Resource Manager (ARM) Overview*. Recuperado entre Septiembre y Noviembre de 2025, de  
<https://learn.microsoft.com/azure/azure-resource-manager/>

**Microsoft Azure.** (2025). *Azure Virtual Machines Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de  
<https://learn.microsoft.com/azure/virtual-machines/>

**Microsoft Azure.** (2025). *Azure Virtual Network Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://learn.microsoft.com/azure/virtual-network/>

**NGINX Inc.** (2025). *NGINX Web Server Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://nginx.org/en/docs/>

**OpenSSH (OpenBSD Project).** (2025). *OpenSSH Manual Pages*. Recuperado entre Septiembre y Noviembre de 2025, de <https://www.openssh.com/manual.html>

**VirtualBox – Oracle Corporation.** (2025). *VirtualBox Documentation*. Recuperado entre Septiembre y Noviembre de 2025, de <https://www.virtualbox.org/wiki/Documentation>

**VirtualBox – Oracle Corporation.** (2025). *VirtualBox User Manual*. Recuperado entre Septiembre y Noviembre de 2025, de <https://www.virtualbox.org/manual/>

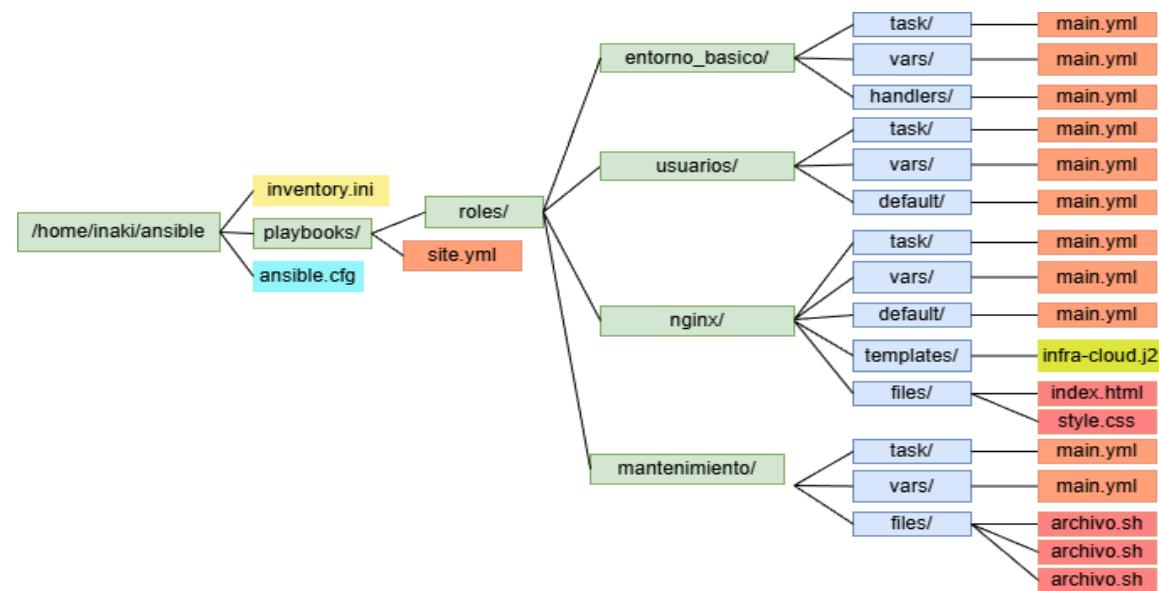
# Anexos/Otros

Dado que la totalidad del proyecto está implementada como **infraestructura como código (IaC)** y **automatización**, el código completo y los recursos **se entregan al tribunal en el medio digital adjunto (pendrive)** para facilitar su revisión, verificación y replicación del entorno.

Esta sección detalla la **estructura de directorios y archivos** del material entregado, sirviendo para demostrar la organización profesional del proyecto.

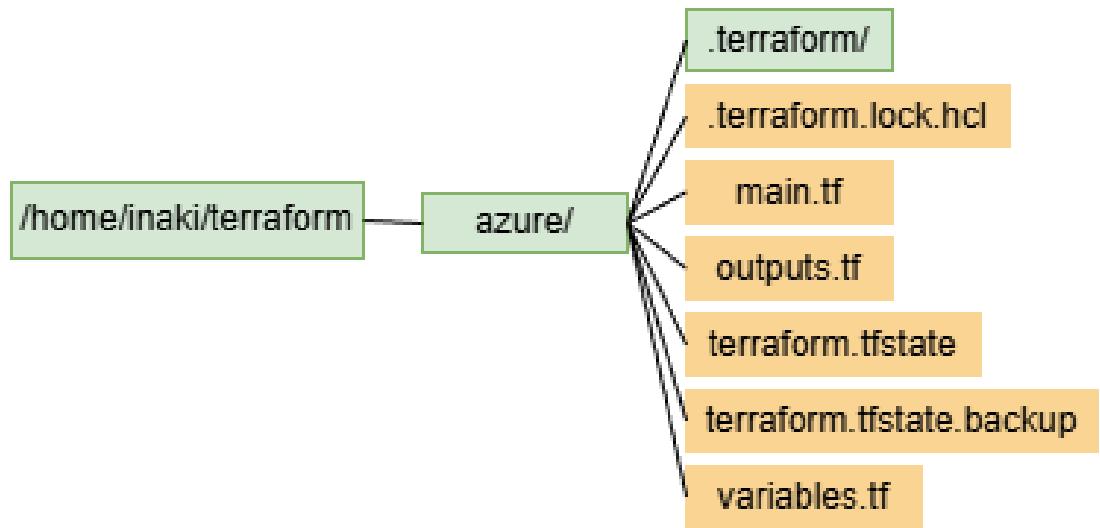
## Anexo I: Automatización (Estructura de directorios de Ansible)

Este anexo reproduce la estructura del directorio de automatización (*ansible/*) en el Host de Control. Esta organización modular facilita la ejecución y configuración del servidor.



## Anexo II: Infraestructura como Código (Estructura de directorios de Terraform)

Este anexo reproduce la estructura del directorio de aprovisionamiento (*terraform/*) en el Host de Control. Estos archivos definen la creación de todos los recursos necesarios en Microsoft Azure.



### Anexo III: Archivos de contenido y funcionalidad del servidor

Este anexo lista los archivos estáticos (.html y .css) y scripts (.sh) que son desplegados por Ansible en la VM de Azure. Estos elementos son cruciales para la validación funcional del servidor web Nginx y para la demostración de las capacidades de mantenimiento.

Archivo	Función
<b>index.html / style.css</b> (sitio <b>infra-cloud</b> )	Contenido web que confirma que Nginx está sirviendo el sitio Infra-cloud
<b>index.html / style.css</b> (sitio <b>test</b> )	Contenido web que confirma que Nginx está sirviendo el sitio Test
<b>restart_nginx.sh</b>	Script de mantenimiento de reinicio del servidor web Nginx
<b>backup_infra-cloud.sh</b>	Script de mantenimiento de backup de configuración de Nginx
<b>restaurar_backup_infra-cloud.sh</b>	Script de mantenimiento de recuperación de backup de configuración de Nginx

#### Anexo IV: Autenticación, configuración fuente y trazabilidad

Este anexo contiene los archivos que demuestran el método de conexión segura, la configuración final aplicada por la automatización y la evidencia de ejecución necesaria para validar la estabilidad del sistema.

Archivo o carpeta	Función
.ssh/	<b>Autenticación (Seguridad).</b> Directorio que contiene el archivo de clave privada <b>id_rsa</b> , esencial para que el Host de Control pueda establecer sesiones seguras en la Máquina Virtual de Azure.
infra-cloud.j2	<b>Configuración fuente.</b> Plantilla <b>Jinja2</b> que contienen la lógica para generar el <i>server block</i> de Nginx para infra-cloud.
ansible_full_log.txt	<b>Evidencia de idempotencia.</b> Registro completo de la ejecución del <i>playbook</i> de <b>Ansible</b>
deploy_log.txt	<b>Evidencia de Estabilidad IaC.</b> Registro de la ejecución de <b>terraform plan</b> , que confirma que la infraestructura en <b>Azure</b> se encuentra en un <b>estado estable</b> .