Cristian Pérez Gómez

Del Código a la Seguridad: Despliegue, Cumplimiento y Detección de Amenazas en AWS

Índice

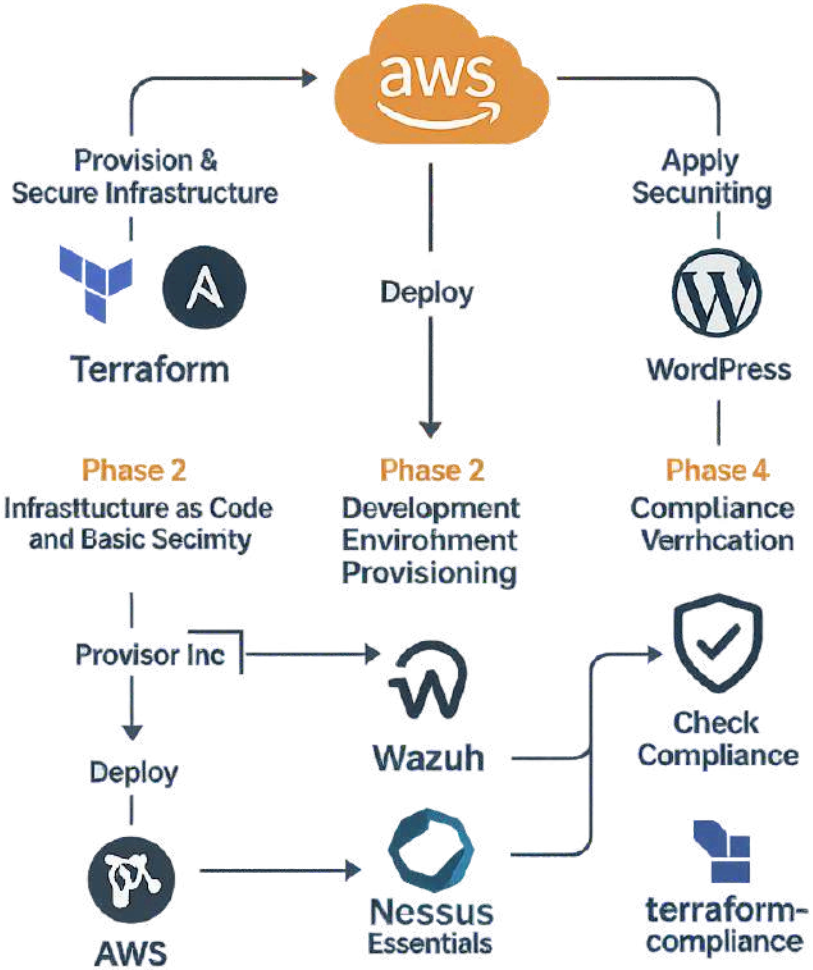
Aclaraciones técnicas..... Páginas 2 - 6
Objetivos del proyecto..... Páginas 2 - 3
Descripción técnica estructura AWS..... Páginas 4 - 6
Fase 1: Fundamentos de IaC y Seguridad Básica..... Página 7
¿Qué es Infrastructure as Code (IaC)?..... Página 7
Seguridad en IaC..... Página 7
Fase 2: Aprovisionamiento de entorno de desarrollo..... Páginas 8 - 13
Instalación de Ansible..... Página 8
Instalación de Terraform..... Página 9
Instalación de AWS Client..... Página 10
Configuración de AWS Client..... Páginas 11 - 13
Fase 3: Despliegue de infraestructura..... Páginas 14 - 32
Infraestructura en AWS: Descripción Técnica y Funcional..... Página 14
Introducción al código..... Páginas 15 - 25
Subida a producción y testeo de la infraestructura..... Páginas 26 - 32
Fase 4: Cumplimiento normativo..... Páginas 33 - 39
¿Qué es el cumplimiento normativo?..... Página 33
¿Qué implica el cumplimiento normativo en infraestructura?..... Página 33
Terraform Compliance..... Páginas 34 - 37
AWS Config..... Páginas 38 - 39
Fase 5: Gestión de Parches y Configuración Segura..... Páginas 40 - 46
Gestión de parches..... Página 41
Configuración segura (hardering)..... Páginas 42 - 45
Pruebas en el entorno de producción..... Páginas 46
Fase 6: Escaneo de Vulnerabilidades..... Páginas 47 - 58
Despliegue de Nessus..... Páginas 47 - 55
Pruebas de Seguridad de la Infraestructura..... Páginas 56 - 58
Fase 7: Monitoreo y Detección de Amenazas..... Páginas 59 - 68
Despliegue de Wazuh..... Páginas 59 - 65
Agente de Wazuh..... Páginas 66 - 68
Fase 8: Supervisión, Seguridad, Cumplimineto y Detección Continua de Amenazas..... Páginas 69 - 74
Monitorizar archivos y detectar cambios (FIM)..... Páginas 69 - 70
Detección de intrusiones (HIDS)..... Página 71
Configuration Assessment..... Páginas 72 - 73
Malware Detection..... Página 73
Cloud Security de Wazuh para AWS..... Página 74
Resultados: Alertas de Seguridad Detectadas en el Primer Mes de Operación..... Página 75
Webgrafía..... Página 76

Aclaraciones técnicas

Objetivos del proyecto

Descripción de objetivos

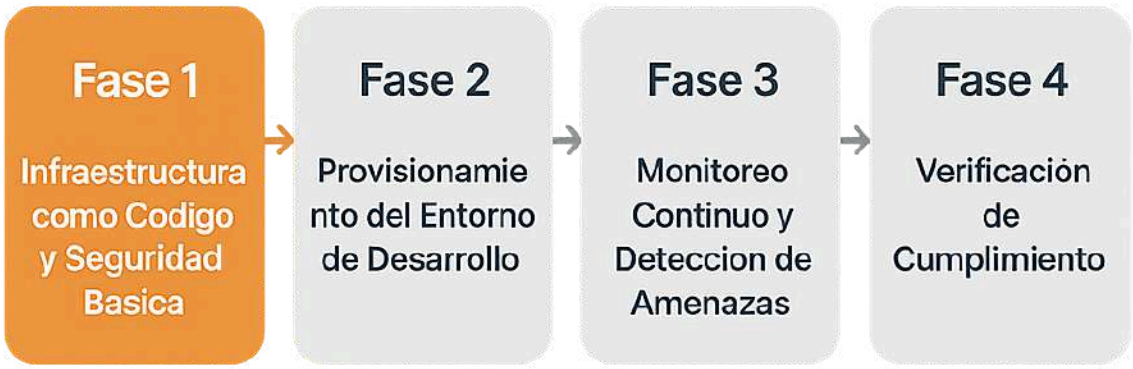
El objetivo principal de este proyecto es **diseñar, desplegar y asegurar una infraestructura completa en Amazon Web Services** (AWS) utilizando metodologías modernas de *Infrastructure as Code* (IaC), combinando las capacidades de Terraform para el aprovisionamiento y Ansible para la configuración post-despliegue.



A través de este enfoque, se busca:

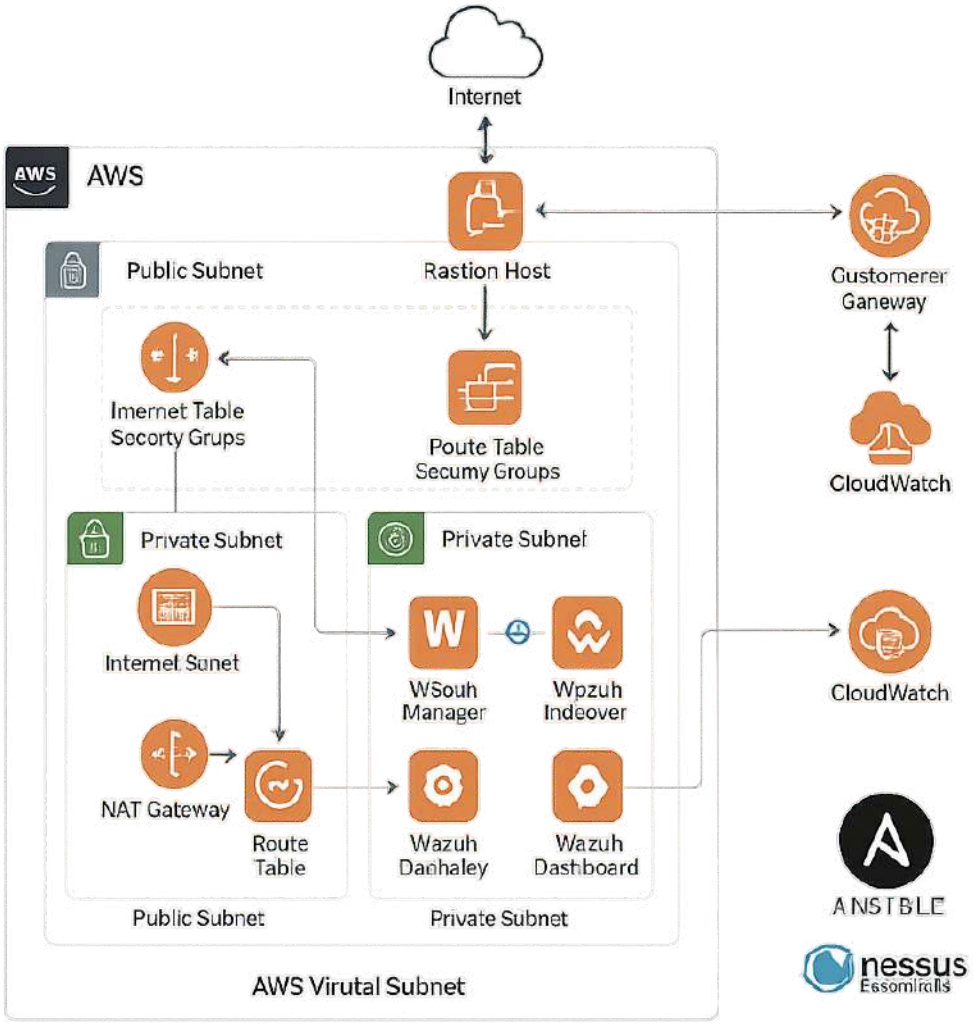
- **Automatizar el ciclo de vida completo de la infraestructura**, desde su creación hasta su configuración segura.
- **Garantizar el cumplimiento normativo** mediante auditorías automatizadas y reglas de seguridad (AWS Config, Terraform Compliance).
- **Monitorear y proteger activamente el entorno** frente a amenazas, vulnerabilidades y cambios no autorizados mediante herramientas como Wazuh y Nessus.
- **Aplicar principios de defensa en profundidad**, con separación de subredes públicas y privadas, uso de bastion hosts, control granular de accesos y monitoreo continuo.
- **Demostrar la trazabilidad, reproducibilidad y seguridad** de los despliegues cloud mediante código versionado, validado y auditable.
- **Construir una arquitectura escalable y segura**, adecuada para entornos productivos, laboratorios o proyectos reales con requisitos técnicos y normativos exigentes.

Este proyecto no solo tiene un propósito técnico, sino también pedagógico y estratégico, ya que representa una solución realista, modular y extensible para organizaciones que buscan adoptar IaC con un enfoque de seguridad desde el diseño.



Descripción técnica estructura AWS

La infraestructura desplegada en AWS se organiza bajo una arquitectura segmentada y segura, basada en las buenas prácticas del **AWS Well-Architected Framework**. Fue creada íntegramente con Terraform y configurada mediante Ansible, lo que permite un control total sobre cada recurso.



Red y Acceso

- **VPC principal** (30.0.0.0/16) con separación en:
    - Subredes públicas (acceso externo): Bastion Host, NAT Gateway.
    - Subredes privadas (acceso interno): WordPress, Wazuh, Nessus, RDS.
- **Internet Gateway y NAT Gateway** → permiten comunicación controlada según la zona.
- **Tablas de rutas y asociaciones a subredes** para direccionamiento interno segmentado.

Recursos de Cómputo

- **EC2 Bastion Host** → Único punto de entrada pública vía SSH. Desde aquí se gestionan los nodos privados.
- **EC2 WordPress Server** → Desplegado en subnet privada, accede al backend RDS.
- **EC2 Nessus Scanner** → Analizador de vulnerabilidades, con puerto 8834 habilitado.
- **EC2 Wazuh Server** → Plataforma SIEM compuesta por Wazuh Manager, API y Dashboard.

Seguridad

- **Grupos de Seguridad personalizados** → Para bastión, Wazuh, WordPress y RDS. Reglas explícitas de tráfico entrante y saliente.
- **Par de claves SSH** → Para acceso seguro a instancias.
- **Reglas de AWS Config** → Controlan cumplimiento en puertos restringidos y estructura VPC.

Almacenamiento y Base de Datos

- **Amazon RDS MySQL:**
    - Aislado en subred privada, acceso solo desde WordPress.
    - Backups automáticos y configuración sin exposición pública.

Supervisión y Cumplimiento

- **Wazuh SIEM:**
    - Integración completa de agentes EC2 (con Ansible).
    - Monitoreo FIM, HIDS, malware, cambios en archivos, cumplimiento (CIS).
- **Nessus Essentials:**
    - Escaneo activo de vulnerabilidades con reporte detallado.
    - Reglas sobre puertos, servicios obsoletos, y hardening.

Gestión de configuración

- **Ansible:**
    - Aplicación de parches, configuración segura (hardening), instalación automatizada de Wazuh y Nessus.
    - Uso de inventory.ini para conexión por grupos a nodos bastión y privados.

Este entorno representa un **modelo de seguridad en múltiples capas**, completamente automatizado, auditable y escalable. La combinación de Terraform + Ansible + Wazuh + Nessus forma un ecosistema integral de despliegue, configuración, monitoreo y cumplimiento.

Fase 1: Fundamentos de IaC y Seguridad Básica

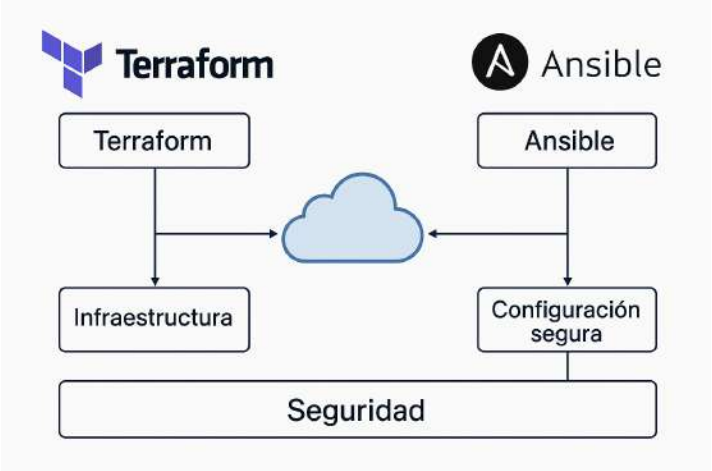
¿Qué es Infrastructure as Code (IaC)?

IaC (Infrastructure as Code) es una metodología que permite describir y gestionar la infraestructura mediante archivos de código, permitiendo la automatización, trazabilidad y reproducibilidad de entornos. Esta práctica elimina la configuración manual, reduce errores humanos y asegura entornos consistentes.

En este proyecto, se utilizan dos herramientas complementarias:

- **Terraform (HashiCorp)** → Aprovisionamiento declarativo de recursos cloud **(AWS)**, mediante bloques HCL **(HashiCorp Configuration Language)**.

- **Ansible (Red Hat)** → Gestión de configuración y automatización de tareas a nivel de sistema operativo, utilizando YAML **(playbooks)** y conexión remota por SSH.



Beneficios de IaC

- Declarativo y sencillo.
- Reutilizable mediante módulos.
- Compatible con control de versiones.
- Creación de planes de ejecución.
- Escalabilidad eficiente y programada.
- Control de políticas.
- Velocidad en despliegues.
- Integración CI/CD.

Seguridad en IaC

La seguridad en IaC implica **asegurar desde el código** los recursos que se van a desplegar. Esto incluye:

- **Restringir acceso de red** (grupos de seguridad, firewalls).

- **Validar configuraciones** con herramientas como terraform-compliance.

Fase 2: Aprovisionamiento de entorno de desarrollo

Antes de proceder con el despliegue de infraestructura mediante herramientas de Infrastructure as Code **(IaC)**, es fundamental **preparar adecuadamente el entorno de desarrollo local**.

Esta fase inicial garantiza que el entorno cuente con todas las herramientas, configuraciones y permisos necesarios para interactuar de forma segura y eficiente con los servicios en la nube.

La preparación del entorno incluye los siguientes aspectos clave:

Instalación de Ansible

Dependiendo del sistema operativo, **Ansible puede instalarse mediante el gestor de paquetes correspondiente**. En sistemas basados en Debian o Ubuntu, por ejemplo, puede utilizarse el siguiente comando.

```
devsecops@PC-Cristian:~$ sudo apt install ansible
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  ansible-core ieee-data python-babel-localedata python3-anyio python3-arg
  python3-httpx python3-hyperframe python3-jinja2 python3-jmespath python3
  python3-packaging python3-passlib python3-pygments python3-requests-kerb
  python3-tz python3-winrm python3-xmltodict python3-yaml
```

Una vez completada la instalación, se debe verificar que Ansible esté correctamente instalado y disponible en el sistema. Para ello, se ejecuta el siguiente comando.

```
devsecops@PC-Cristian:~$ ansible --version
ansible [core 2.14.18]
  config file = None
  configured module search path = ['/home/devsecops/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/devsecops/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.11.2 (main, Nov 30 2024, 21:22:50) [GCC 12.2.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

Instalación de Terraform

Para poder utilizar **Terraform** como herramienta de aprovisionamiento de infraestructura, es necesario realizar su instalación en el sistema operativo anfitrión. En este caso, el sistema base es **Debian 12**, por lo que se deben seguir los pasos adecuados para garantizar una instalación correcta y segura.

Antes de instalar Terraform, se deben **instalar ciertas dependencias básicas requeridas para el correcto funcionamiento del sistema de gestión de paquetes** y para facilitar la conexión con el repositorio oficial de HashiCorp.

```
devsecops@PC-Cristian:~$ sudo apt install gnupg software-properties-common curl
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
gnupg ya está en su versión más reciente (2.2.40-1.1).
fijado gnupg como instalado manualmente.
software-properties-common ya está en su versión más reciente (0.99.30-4.1~deb12u1).
curl ya está en su versión más reciente (7.88.1-10+deb12u12).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
```

Con las dependencias ya disponibles, se procede a instalar Terraform utilizando el **repositorio oficial de HashiCorp**. Para ello, se deben seguir los siguientes pasos.

```
devsecops@PC-Cristian:~$ sudo apt install terraform
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  terraform
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
```

Una vez completada la instalación, se debe verificar que Terraform esté correctamente instalado y disponible en el sistema. Para ello, se ejecuta el siguiente comando.

```
devsecops@PC-Cristian:~$ terraform -version
Terraform v1.11.3
on linux_amd64
```

Instalación de ASW Client

La **AWS Command Line Interface (AWS CLI)** es la herramienta oficial de Amazon Web Services que permite interactuar con sus servicios desde la línea de comandos. Su instalación es fundamental para autenticar herramientas como **Terraform** y **Ansible**, ya que les permite acceder y gestionar recursos en la nube de forma segura y automatizada.

El primer paso consiste en **descargar el paquete oficial de instalación** desde el sitio web de Amazon, utilizando curl o wget. A continuación se muestra un ejemplo utilizando curl.

```
devsecops@PC-Cristian:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  5 65.0M    5 3334k    0     0  3357k      0  0:00:19 --:--:--  0:00:19 3358k
```

Una vez finalizada la descarga, se debe descomprimir el archivo .zip utilizando la herramienta unzip. Si unzip no está instalado, puede agregarse mediante apt:

```
devsecops@PC-Cristian:~$ unzip awscliv2.zip
Archive:  awscliv2.zip
   creating: aws/
   creating: aws/dist/
  inflating: aws/THIRD_PARTY_LICENSES
  inflating: aws/README.md
  inflating: aws/install
   creating: aws/dist/awscli/
   creating: aws/dist/cryptography/
```

A continuación, se ejecuta el script de instalación proporcionado por Amazon, Este comando instalará AWS CLI en el sistema, normalmente en **/usr/local/bin/aws**.

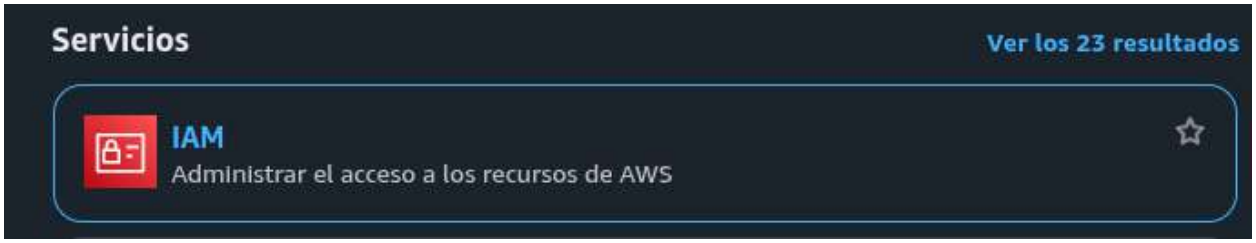
Para comprobar que AWS CLI se ha instalado correctamente y está disponible en el sistema, se puede ejecutar el siguiente comando:

```
devsecops@PC-Cristian:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
devsecops@PC-Cristian:~$ aws --version
aws-cli/2.25.6 Python/3.12.9 Linux/6.1.0-32-amd64 exe/x86_64.debian.12
```

Configuración de AWS Client

Para que la herramienta **AWS CLI** pueda interactuar correctamente con los servicios de AWS, es necesario autenticarla mediante un **usuario de servicio**, conocido como **usuario IAM (Identity and Access Management)**. Este usuario representa una identidad con permisos específicos y credenciales asociadas, necesarias para ejecutar acciones programáticas sobre la infraestructura en la nube.

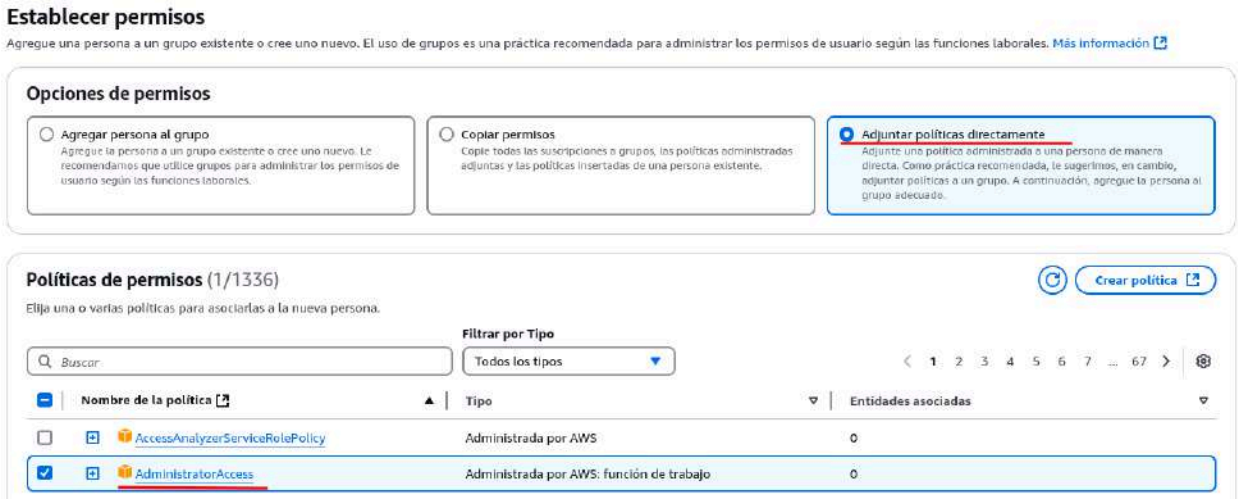
Para ello, en el buscador del panel principal, escribiremos **IAM** y accederemos al servicio correspondiente.



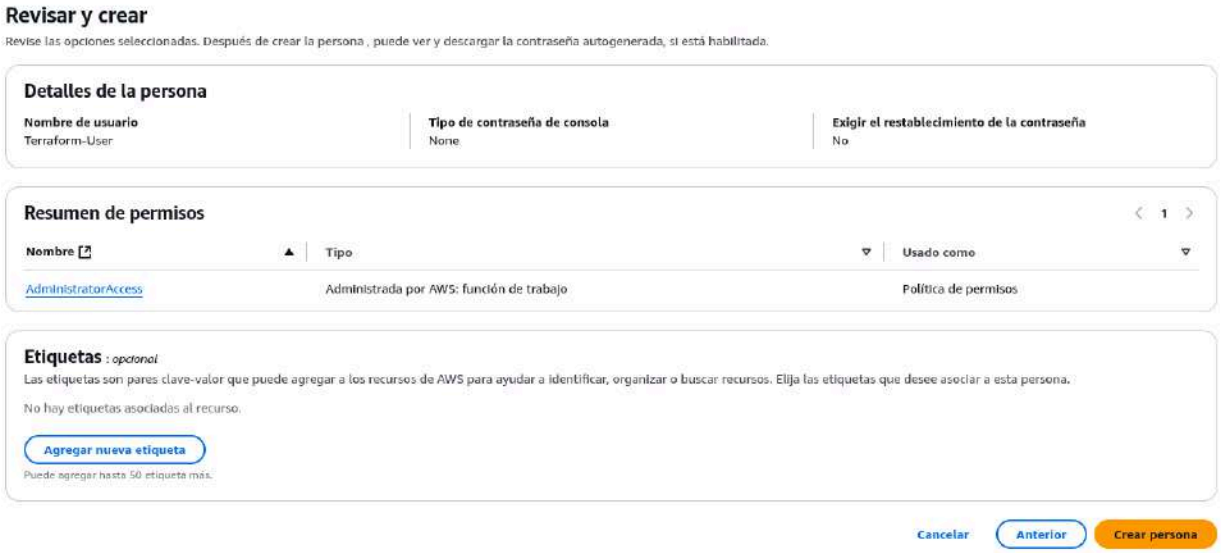
Posteriormente, haremos clic en "**Agregar usuario**". Durante el proceso, se debe completar el nombre de usuario.

En la sección de permisos, se debe adjuntar una **política que determine el nivel de acceso del usuario**. Para fines de prueba, laboratorio o despliegue inicial, se puede asignar la política predefinida:

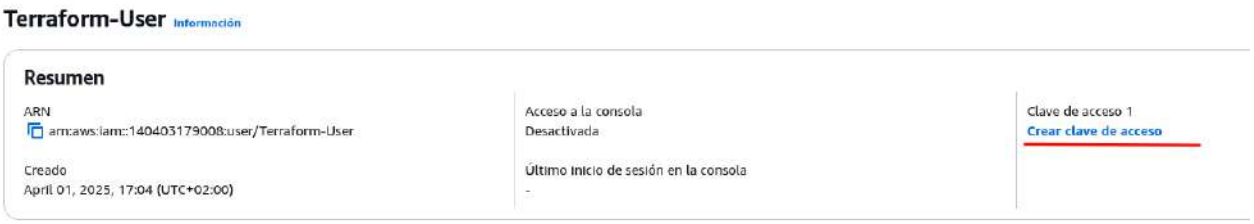
Esta política otorga acceso completo a todos los recursos y servicios de AWS. **Se recomienda usarla solo en entornos de desarrollo o bajo entornos controlados**.



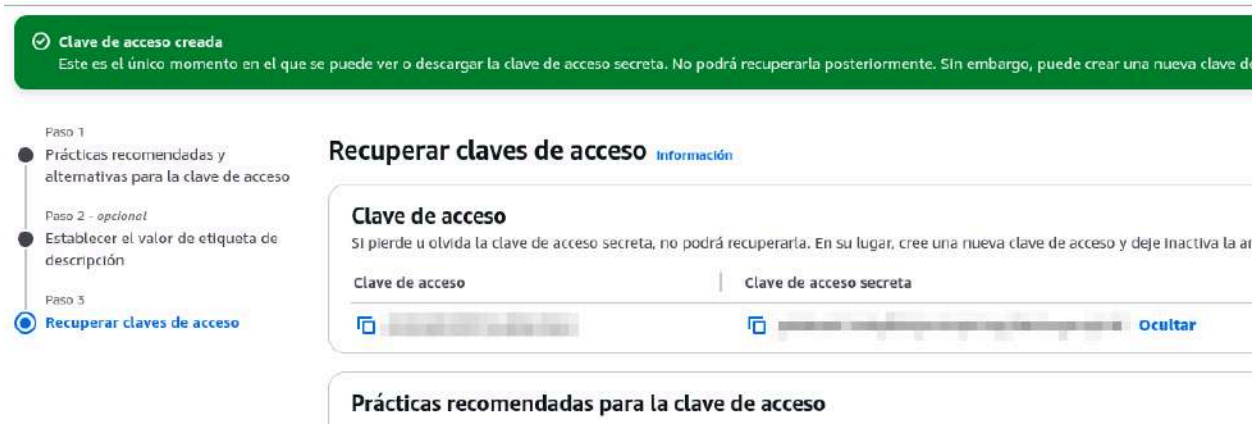
Una vez creado el usuario IAM, la consola de AWS proporciona un resumen de su configuración.



El siguiente paso consiste en generar las **credenciales de acceso programático** necesarias para que herramientas como AWS CLI puedan autenticarse y operar con los servicios de AWS en nombre del usuario.



La clave secreta **solo se muestra una vez** en pantalla. Se recomienda copiarla inmediatamente y almacenarla en un lugar seguro.



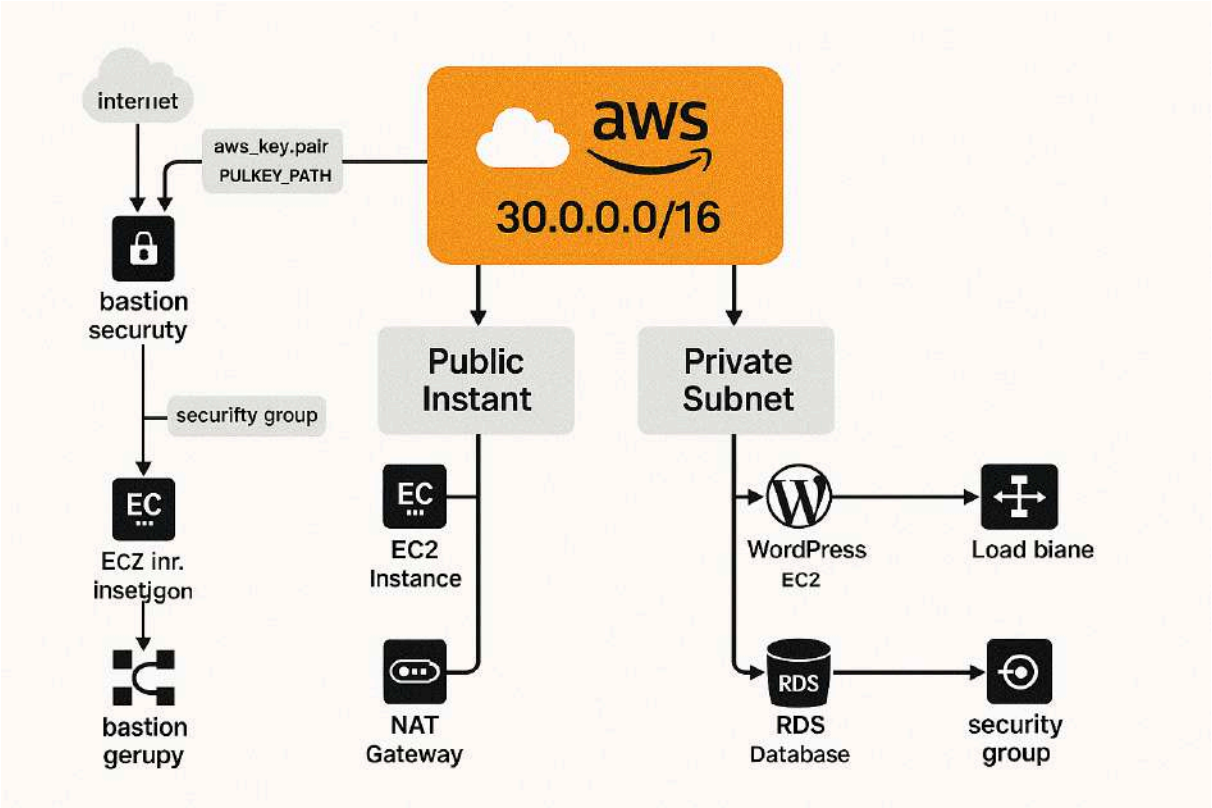
Con las claves generadas, es necesario **configurar AWS CLI** para que utilice dichas credenciales al interactuar con AWS. Esto se realiza ejecutando el siguiente comando:

```
devsecops@PC-Cristian:~$ aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]: json
```

Fase 3: Despliegue de infraestructura

Infraestructura en AWS: Descripción Técnica y Funcional

La infraestructura representada en el siguiente diagrama está diseñada bajo una arquitectura de red segmentada, que implementa **buenas prácticas de seguridad en AWS** y facilita el despliegue de aplicaciones web escalables.



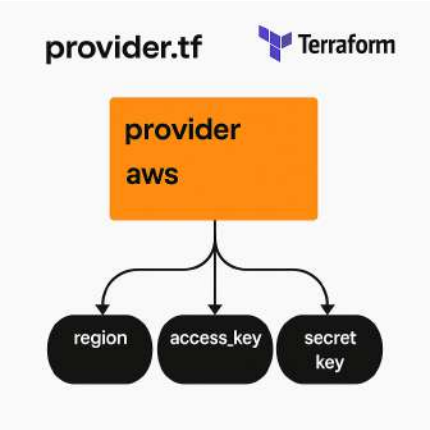
Esta arquitectura hace uso de **subredes públicas y privadas**, servidores bastión, balanceadores de carga, y servicios gestionados como **RDS** (Relational Database Service), todo ello definido bajo el paradigma de **Infrastructure as Code (IaC)**, empleando herramientas como **Terraform** para el aprovisionamiento y **Ansible** para la configuración post-despliegue.

Esta arquitectura es un ejemplo clásico de **entorno seguro y escalable en AWS**, ideal para aplicaciones web que requieren separación de capas, acceso controlado a la base de datos y alta disponibilidad.

Introducción al código

Terraform organiza la infraestructura en **bloques reutilizables llamados recursos**. Cada archivo `.tf` agrupa recursos relacionados. A continuación, se explican cada uno de los archivos utilizados en el proyecto

Archivo: provider.tf



Este archivo tiene como propósito principal la **declaración del proveedor de infraestructura en la nube**, que en este caso es **Amazon Web Services (AWS)**. En Terraform, los proveedores son responsables de traducir los bloques de configuración en llamadas a las API de los distintos servicios, permitiendo el aprovisionamiento, modificación y destrucción de recursos en la plataforma seleccionada.

Dentro de este archivo se especifican dos elementos clave:

Proveedor (provider) → Se utiliza el bloque provider **"aws"** para indicar que se desea trabajar con recursos de AWS. **Este bloque también puede incluir configuraciones específicas** como perfiles de credenciales, acceso mediante variables de entorno, y configuraciones avanzadas como timeouts o endpoints personalizados.

Región (region) → Se establece una región por defecto **(por ejemplo, eu-west-1, us-east-1, etc.)**, lo que define la zona geográfica de los data centers donde se desplegarán los recursos de infraestructura. Esta configuración es esencial para garantizar baja latencia, **cumplimiento normativo y disponibilidad** según las necesidades del proyecto.

provider.tf ×

Terraform_Project_Resources > terraform > provider.tf

```
# --Definimos el proveedor (AWS)--

# Este bloque configura el proveedor de Terraform para AWS.
# Utiliza variables para definir las credenciales y la región, lo que permite flexibilidad y seguridad.

provider "aws" {
  region     = var.region     // Región donde se desplegarán los recursos (por ejemplo, us-east-1).
  access_key = var.access_key // Clave de acceso pública para autenticar en la cuenta de AWS.
  secret_key = var.secret_key // Clave de acceso privada para autenticar en la cuenta de AWS.
}
```

Archivo: versions.tf

Este bloque de configuración tiene como objetivo establecer los requisitos mínimos de versión tanto para la herramienta principal **—Terraform—** como para los proveedores **(providers)** utilizados dentro del proyecto. Esta práctica es esencial para garantizar la **compatibilidad, estabilidad y reproducibilidad** de los despliegues en distintos entornos.

Este archivo actúa como **mecanismo de control de versiones**, y es una parte fundamental de cualquier infraestructura definida como código

Dentro de este archivo se especifican dos elementos clave:

Requisito de versión de Terraform → Se utiliza el bloque terraform con la directiva **required_version** para definir la versión mínima **(o el rango permitido)** de Terraform que debe utilizarse. Esto previene la ejecución accidental del proyecto con versiones incompatibles, lo que podría generar errores inesperados en tiempo de ejecución.

Requisitos de los proveedores → Dentro del mismo bloque terraform, se define también el subbloque **required_providers,** que especifica qué proveedores se utilizarán **(por ejemplo, aws, random, null, etc.)** y qué versiones mínimas deben cumplir. Esto asegura que los módulos y recursos declarados se comporten conforme a las versiones validadas durante el desarrollo.

```
versions.tf ×
Terraform_Project_Resources > terraform > versions.tf
1   # --Configuración Terraform--
2
3   terraform {
4     required_version = ">= 1.0" // Indicamos la versión mínima requerida de Terraform.
5     required_providers {
6       aws = {
7         source  = "hashicorp/aws" // Especifica el proveedor de AWS.
8         version = "~> 3.0"        // Especifica la versión mínima del proveedor de AWS.
9       }
10    }
11  }
```

Archivo: variables.tf

variables.tf Terraform

variable
instance_type
description = "Tipo de instancia para Wordpress"
default = t2.micro

Este archivo tiene la función principal de **declarar todas las variables reutilizables que serán utilizadas a lo largo del proyecto Terraform**. Las variables permiten parametrizar configuraciones, separar datos de implementación del código de infraestructura y facilitar la reutilización y personalización de los despliegues en distintos entornos.

Dentro de este archivo se especifican dos elementos clave:

Finalidad de las variables → Al declarar variables en Terraform, se define un conjunto de entradas dinámicas que pueden ser utilizadas en cualquier parte del proyecto para sustituir valores estáticos.

Ejemplos típicos de variables → Entre las variables más comunes que suelen declararse en este archivo se incluyen:

variables.tf ×

Terraform_Project_Resources > terraform > variables.tf

```
1   # -- Variables --
2
3   # Variable que define el nombre de la base de datos.
4   variable "database_name" {
5     description = "Nombre de la base de datos"
6     default     =
7   }
8
9   # Variable que define la contraseña para acceder a la base de datos.
10  variable "database_password" {
11    description = "Contraseña de la base de datos"
12    default     =
13  }
```

Archivo: Vpc subnets

Este archivo tiene como objetivo principal la definición de la **red base del entorno en la nube**, la cual se construye sobre una **VPC (Virtual Private Cloud)** y sus respectivas **subredes**. Esta capa constituye el **fundamento lógico de conectividad** sobre el cual se desplegarán todos los demás recursos de la infraestructura (instancias EC2, bases de datos, balanceadores, etc.).

Dentro de este archivo se especifican dos elementos clave:

¿Qué es una VPC? → Una VPC en AWS es una red virtual privada dentro de la infraestructura de la nube, que permite al usuario definir su propio espacio de red. Al crear una VPC mediante Terraform, se especifican estos elementos de forma declarativa y reproducible.

Subredes → Dentro de la VPC, se definen subredes que pueden clasificarse en:

- **Subredes públicas** → Conectadas directamente a internet mediante un **Internet Gateway**, típicamente usadas para balanceadores o instancias bastión.

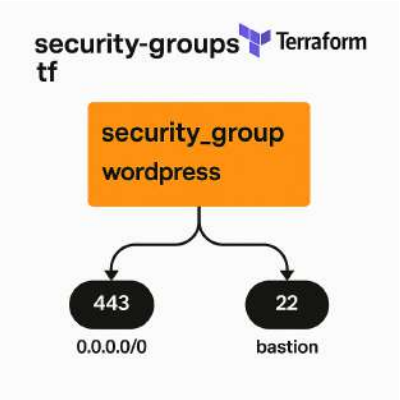
- **Subredes privadas** → No tienen acceso directo desde internet y se utilizan para recursos internos como bases de datos, servidores de aplicación, etc.

vpc-subnets.tf ×

Terraform_Project_Resources > terraform > vpc-subnets.tf

```tf
# --VPC & Subredes--

# 1. Creamos la VPC (Virtual Private Cloud)
# Este recurso define una red privada virtual (VPC) con soporte para DNS y hostnames internos.
resource "aws_vpc" "vpc" {
  cidr_block           = "30.0.0.0/16"                  // Rango de direcciones IPv4 para la VPC.
  enable_dns_support   = true                           // Habilita soporte para DNS interno.
  enable_dns_hostnames = true                           // Habilita nombres de host internos.
  instance_tenancy     = "default"                      // Tenencia predeterminada para las instancias.

  tags = {
    Name = "VPC"                                        // Etiqueta descriptiva para la VPC.
  }
}
```

Archivo: security-groups



Este archivo tiene como finalidad establecer las **reglas de control de tráfico** que protegen los recursos desplegados dentro de la infraestructura de red en AWS. En términos prácticos, se definen los **Security Groups**, que actúan como un **firewall virtual a nivel de instancia**, permitiendo o denegando conexiones **entrantes (inbound)** y **salientes (outbound)** según criterios predefinidos.

Dentro de este archivo se especifican dos elementos clave:

¿Qué es un Security Group? → Un Security Group **(SG)** en AWS es un conjunto de reglas que se asocia a recursos como instancias EC2, balanceadores de carga o bases de datos. A diferencia de los firewalls tradicionales, los SG son stateful, es decir, **si se permite una conexión entrante, la respuesta automática de salida también está permitida** sin necesidad de una regla adicional.

Tipos comunes de reglas → Las configuraciones típicas que se definen mediante Security Groups incluyen:

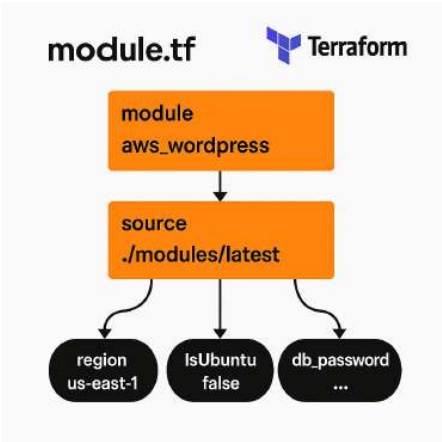
- **Acceso SSH limitado** → Permitir el puerto 22 solo desde IPs de administración específicas.
- **Acceso HTTP/HTTPS** → Habilitar los puertos 80 y 443 para tráfico web desde cualquier fuente (0.0.0.0/0).
- **Comunicación interna segura** → Permitir tráfico sólo entre ciertos grupos (por ejemplo, entre servidor de aplicación y base de datos).

security-groups.tf ×

Terraform_Project_Resources > terraform > security-groups.tf

```
# --Grupos de Seguridad--

# 1. Grupo de Seguridad para Bastion
# Este grupo de seguridad permite todo el tráfico de entrada y salida para propósitos de demostración.
resource "aws_security_group" "bastion" {
  vpc_id = aws_vpc.vpc.id

  ingress {
    description = "Allow all traffic (for demo purposes)" // Permite todo el tráfico entrante.
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]                           // Permite acceso desde cualquier IP.
  }
```

Archivo: module.tf



Este archivo tiene como finalidad principal la **importación, configuración y reutilización de módulos** dentro de un proyecto Terraform. Los módulos permiten encapsular bloques de configuración reutilizables, facilitando la **organización, escalabilidad y mantenimiento** del código de infraestructura.

Dentro de este archivo se especifican dos elementos clave:

¿Qué es un módulo en Terraform? → Un módulo en Terraform es un conjunto lógico de recursos que se agrupan bajo una misma unidad reutilizable. Puede ser:

- **Módulo local** → Definido dentro del mismo repositorio o estructura de carpetas.
- **Módulo remoto** → Publicado en un registro (como Terraform Registry) o alojado en Git, S3, etc.

Los módulos permiten aplicar el principio de **"infraestructura DRY" (Don't Repeat Yourself)**, evitando repetir código y mejorando la mantenibilidad.

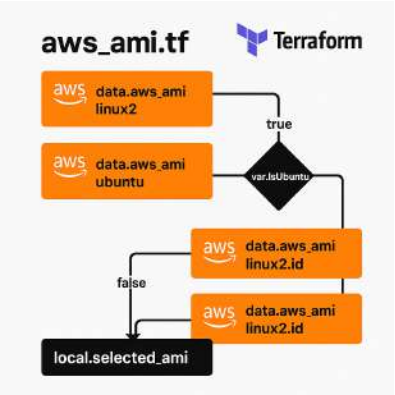
Propósito del archivo module.tf → Este archivo centraliza la invocación de uno o varios módulos y define los valores que serán pasados como entrada **(variables)** y las salidas que se utilizarán **(outputs)**. Es un punto clave en la composición de infraestructuras modulares.

```
module.tf ×
Terraform_Project_Resources > terraform > module.tf
1  # --Parámetros corregidos del módulo--
2
3  # Este bloque define un módulo de Terraform para implementar una solución de WordPress en AWS.
4  # El módulo utiliza configuraciones predefinidas y variables para personalizar los recursos.
5
6  module "aws_wordpress" {
7    source = "./modules/latest"                          // Ruta al módulo que contiene la configuración de WordPress.
8
9    # Requeridos por el módulo
10   rds_sg_id            = aws_security_group.rds.id       // ID del grupo de seguridad asociado a la base de datos RDS.
11   wordpress_sg_id      = aws_security_group.wordpress.id // ID del grupo de seguridad asociado a la instancia de WordPress.
12   selected_ami         = local.selected_ami              // AMI seleccionada para las instancias EC2.
13   db_subnet_group_name = aws_db_subnet_group.RDS_subnet_grp.name // Nombre del grupo de subredes para la base de datos.
14   subnet3_id           = aws_subnet.subnet3.id           // ID de la subred donde se lanzarán las instancias EC2
```

```
26
27   # Instancias EC2
28   instance_type    = var.instance_type                          // Tipo de instancia EC2 (por ejemplo, t2.micro).
29   root_volume_size = var.root_volume_size                       // Tamaño del volumen raíz de la instancia EC2 en GB.
30   key_name         = var.key_name                               // Nombre del par de claves para acceso SSH.
31 }
32
```

Archivo: aws_ami.tf

Este archivo tiene como propósito obtener de forma dinámica la **Amazon Machine Image (AMI)** más reciente de **Amazon Linux 2**, utilizando un **bloque de datos (`data`)** en Terraform. Esta práctica permite que las instancias EC2 se desplieguen siempre con una imagen actualizada, confiable y mantenida por Amazon, sin necesidad de definir manualmente un ID de AMI que podría volverse obsoleto.



Dentro de este archivo se especifican dos elementos clave:

¿Qué es un bloque `data` en Terraform? → El bloque `data` se utiliza para consultar información externa o ya existente en lugar de crear nuevos recursos. En este caso, permite buscar en tiempo de ejecución la AMI más reciente publicada por un proveedor específico.

Ejemplo típico de uso → A continuación se muestra un ejemplo representativo del uso del bloque `data` para obtener la AMI oficial más reciente de Amazon Linux 2.

```
instances.tf ×
Terraform_Project_Resources > terraform > instances.tf
1  # --Instancias--
2
3  # 1. Importamos la clave para las instancias
4  # Este recurso crea un par de claves en AWS para permitir el acceso SSH a las instancias EC2.
5  resource "aws_key_pair" "mykey-pair" {
6    key_name   = var.key_name                          // Nombre del par de claves que se registrará en AWS.
7    public_key = file(var.PUBLIC_KEY_PATH)            // Ruta al archivo que contiene la clave pública que se usará.
8  }
9
```

Ventajas del enfoque dinámico

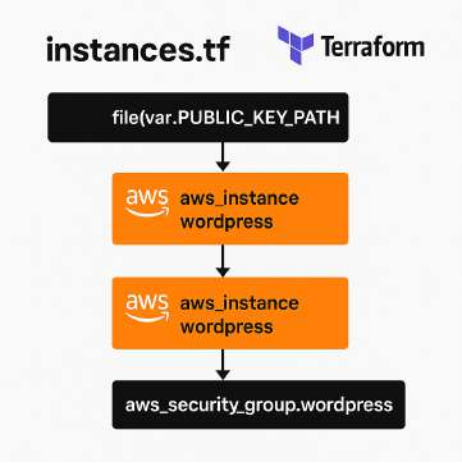
Actualización automática → Siempre se selecciona la versión más reciente, sin necesidad de modificar el código manualmente.

Portabilidad → Facilita reutilizar el código en diferentes regiones o entornos, sin hardcodear IDs específicos.

En resumen, este archivo proporciona una forma **segura, flexible y automatizada** de seleccionar la AMI base para las instancias del entorno.

Archivo: instances.tf

Este archivo tiene como objetivo declarar y aprovisionar una o más **instancias EC2** dentro del entorno definido en AWS, utilizando recursos previamente configurados como la **VPC**, **subredes**, **grupos de seguridad** y la **AMI más reciente**, obtenida dinámicamente desde un bloque `data` en el archivo.



En resumen, este archivo representa la **unidad de cómputo principal del entorno** y se basa en recursos definidos previamente para integrarse de forma segura, organizada y eficiente en la arquitectura general.

<u>Dentro de este archivo se especifican dos elementos clave:</u>

¿Qué es una instancia EC2? → Una instancia EC2 **(Elastic Compute Cloud)** es un servidor virtual en la nube de AWS, configurable en términos de sistema operativo, CPU, memoria, red, almacenamiento y permisos. **Es el componente principal** para ejecutar aplicaciones, servicios, contenedores, bases de datos, entre otros.

Terraform permite automatizar el despliegue de estas instancias, asegurando consistencia y control en la configuración.

Recursos utilizados en este archivo → El bloque resource **"aws_instance"** se utiliza para definir la creación de instancias. Dentro del archivo se referencian recursos ya definidos en otros archivos del proyecto.

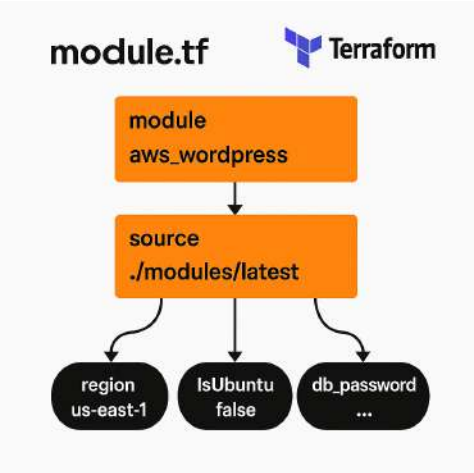
aws_ami.tf

Terraform_Project_Resources > terraform > aws_ami.tf

```
# --Parámetros AMI--

# 1. AMI Amazon Linux 2
# Este bloque obtiene la AMI más reciente de Amazon Linux 2.
data "aws_ami" "linux2" {
  most_recent = true                                    // Selecciona la AMI más reciente disponible.

  filter {
    name   = "name"                                     // Filtra por el nombre de la AMI.
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]            // Patrón para identificar AMIs de Amazon Linux 2.
  }
```

Archivo: main.tf (/modules)

Este archivo contiene la **declaración explícita de los recursos gestionados por el módulo**, es decir, los elementos que serán aprovisionados cuando el módulo sea invocado desde un archivo superior, como module.tf. Dentro de este archivo se define la lógica principal del módulo, utilizando variables de entrada para parametrizar los recursos y valores de salida para exponer resultados relevantes.



Dentro de este archivo se especifican dos elementos clave:

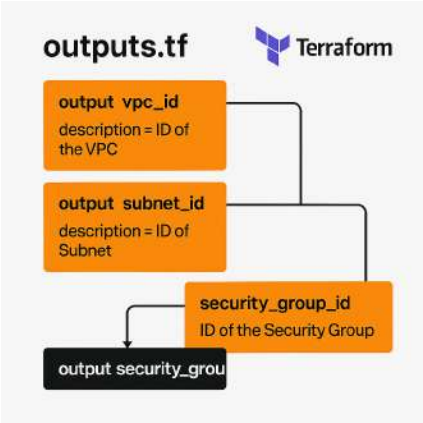
¿Qué es main.tf dentro de un módulo? → El archivo main.tf en un módulo es **el núcleo funcional del mismo.** Aquí se definen los recursos que el módulo se encargará de desplegar, como instancias EC2, subredes, VPCs, balanceadores de carga, claves SSH, etc. El contenido de este archivo es reutilizable y se abstrae de valores específicos, que son proporcionados externamente al momento de invocar el módulo.

Ejemplos de recursos típicos definidos en main.tf →Algunos ejemplos comunes de recursos que podrían definirse dentro de este archivo son:

```
main.tf
Terraform_Project_Resources > terraform > modules > latest > main.tf
1  // Este recurso define una instancia de AWS EC2 para alojar un servidor de WordPress.
2  // Utiliza variables para configurar sus atributos, lo que permite flexibilidad y reutilización.
3  resource "aws_instance" "wordpress_instance" {
4    ami                    = var.selected_ami                // ID de la Amazon Machine Image (AMI) seleccionada.
5    instance_type          = var.instance_type               // Tipo de instancia EC2 (por ejemplo, t2.micro, t3.medium).
6    subnet_id              = var.subnet3_id                  // ID de la subred donde se lanzará la instancia.
7    vpc_security_group_ids = [var.wordpress_sg_id]           // Grupo(s) de seguridad asociado(s) a la instancia.
8    key_name               = var.key_name                    // Nombre del par de claves para acceso SSH.
9
10   // Configuración del dispositivo de almacenamiento raíz de la instancia.
11   root_block_device {
12     volume_size = var.root_volume_size                     // Tamaño del volumen raíz en GB.
13   }
14
15   // Etiquetas para identificar la instancia en la consola de AWS.
16   tags = {
17     Name = "WordPress Server"                              // Nombre descriptivo para la instancia.
18   }
19 }
20
21 // Este recurso define una instancia de base de datos AWS RDS para la aplicación de WordPress.
```

Archivo: outputs.tf (/modules)

El archivo outputs.tf tiene como propósito declarar los **valores de salida (outputs)** de un módulo, es decir, los datos que serán **expuestos y devueltos al archivo principal** que invoque dicho módulo. Estas salidas permiten que otros bloques de configuración accedan a información generada dentro del módulo, promoviendo la reutilización y la interconexión entre componentes.

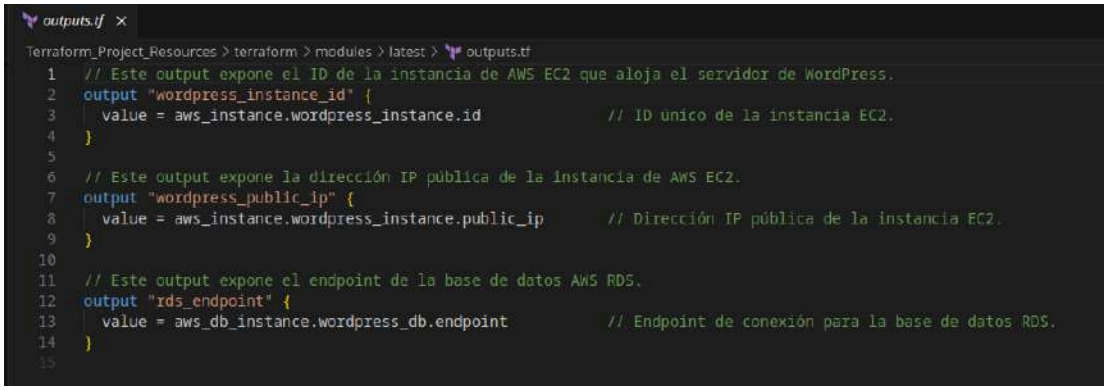


Dentro de este archivo se especifican dos elementos clave:

¿Qué es una salida (output) en Terraform? → Una salida es un valor que Terraform devuelve una vez que termina de aplicar un plan de infraestructura. En el contexto de un módulo, los outputs permiten **exponer recursos clave o resultados intermedios**, para que puedan ser utilizados desde otros módulos o desde el archivo raíz del proyecto **(main.tf o module.tf)**.

Beneficios de usar salidas

- **Reutilización de información** → Entre módulos sin acoplamiento directo.
- **Desacoplamiento lógico** → El módulo se comporta como una caja negra, y solo revela los datos que realmente se necesitan.
- **Interoperabilidad** → Con flujos de automatización, pipelines CI/CD o scripts auxiliares.
- **Claridad en la ejecución** → Al aplicar el plan, Terraform muestra los valores clave de manera explícita.



```
outputs.tf ×
Terraform_Project_Resources > terraform > modules > latest > outputs.tf
1  // Este output expone el ID de la instancia de AWS EC2 que aloja el servidor de WordPress.
2  output "wordpress_instance_id" {
3    value = aws_instance.wordpress_instance.id              // ID único de la instancia EC2.
4  }
5
6  // Este output expone la dirección IP pública de la instancia de AWS EC2.
7  output "wordpress_public_ip" {
8    value = aws_instance.wordpress_instance.public_ip       // Dirección IP pública de la instancia EC2.
9  }
10
11 // Este output expone el endpoint de la base de datos AWS RDS.
12 output "rds_endpoint" {
13   value = aws_db_instance.wordpress_db.endpoint           // Endpoint de conexión para la base de datos RDS.
14 }
15
```

Archivo: variables.tf (/modules)

El archivo variables.tf tiene como objetivo principal definir todas las **variables de entrada** que el módulo acepta desde el archivo principal que lo invoca. Estas variables permiten **parametrizar el comportamiento interno del módulo**, otorgando flexibilidad, reutilización y desacoplamiento en el diseño de la infraestructura como código.

En resumen, el archivo variables.tf actúa como la **interfaz de configuración del módulo**, permitiendo que los usuarios externos lo adapten a sus necesidades sin modificar su lógica interna.

Dentro de este archivo se especifican dos elementos clave:

¿Qué es una variable en un módulo de Terraform? → Las **variables de entrada** son parámetros definidos dentro del módulo que permiten que el usuario (es decir, el archivo o proyecto que utiliza el módulo) le proporcione valores externos. Esto convierte al módulo en una **unidad reutilizable** y configurable, evitando valores codificados de forma rígida.

Beneficios de declarar variables correctamente

- **Reutilización** → El mismo módulo puede usarse en diferentes entornos (dev, test, prod) con distintos valores.

- **Claridad** → El archivo documenta claramente los parámetros requeridos por el módulo.

- **Control de errores** → Terraform validará que se provean los tipos y valores correctos.

variables.tf ×

Terraform_Project_Resources > terraform > modules > latest > variables.tf

```
1  // Variable que define el ID de la Amazon Machine Image (AMI) seleccionada para la instancia EC2.
2  variable "selected_ami" {}
3
4  // Variable que define el tipo de instancia EC2 (por ejemplo, t2.micro, t3.medium).
5  variable "instance_type" {}
6
7  // Variable que define el ID de la subred donde se lanzará la instancia EC2.
8  variable "subnet3_id" {}
9
```

Subida a producción y testeo de la infraestructura

Una vez comprendido el código que se utilizará para levantar la infraestructura, se procede a ejecutar su despliegue en el entorno de producción y realizar un **testeo manual** de los recursos implementados.

Para ello, se sigue una secuencia de comandos que aseguran la validez y consistencia del entorno antes de aplicar cualquier cambio de forma definitiva.

El primer paso consiste en **verificar que la sintaxis del código Terraform sea correcta**. Esto se logra mediante el siguiente comando, este comando analiza la estructura de los archivos **.tf** y confirma que su contenido sea válido. Una salida esperada será similar a:

```
devsecops@PC-Cristian:~/Project_Resources$ terraform validate
Success! The configuration is valid.
```

Una vez validado el código, se procede a **generar el plan de ejecución**, es decir, una vista previa de los cambios que se van a aplicar en la infraestructura. Para ello, se ejecuta:

```
devsecops@PC-Cristian:~/Project_Resources$ terraform plan
data.aws_ami.ubuntu: Reading...
data.aws_ami.linux2: Reading...
data.aws_ami.ubuntu: Read complete after 1s [id=ami-03c1aa37835df06a1]
data.aws_ami.linux2: Read complete after 1s [id=ami-072e42fd77921edac]
```

Este comando muestra de forma detallada:

- Los recursos que serán creados.
- Los que serán modificados **(si existieran)**.
- Los que serán destruidos.

Debemos esperar esta salida, la cual indica que Terraform ha detectado **27 nuevos recursos** que deben ser desplegados y que no habrá modificaciones ni eliminaciones.

```
Plan: 27 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

Para proceder con el despliegue real de la infraestructura, se ejecuta el siguiente comando:

```
devsecops@PC-Cristian:~/Project_Resources$ terraform apply
data.aws_ami.ubuntu: Reading...
data.aws_ami.linux2: Reading...
data.aws_ami.ubuntu: Read complete after 1s [id=ami-03c1aa37835df06a1]
data.aws_ami.linux2: Read complete after 1s [id=ami-072e42fd77921edac]
```

En este punto, se debe escribir **"yes"** para continuar con la creación de los recursos.

```
Plan: 27 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Si no se presentan errores, Terraform mostrará un mensaje indicando que el despliegue ha finalizado correctamente. Una salida típica podría ser:

```
module.aws_wordpress.aws_db_instance.wordpress_db: Still creating... [6m0
module.aws_wordpress.aws_db_instance.wordpress_db: Still creating... [6m1
module.aws_wordpress.aws_db_instance.wordpress_db: Still creating... [6m2
module.aws_wordpress.aws_db_instance.wordpress_db: Still creating... [6m3
module.aws_wordpress.aws_db_instance.wordpress_db: Creation complete afte

Apply complete! Resources: 27 added, 0 changed, 0 destroyed.
```

Esto confirma que la infraestructura ha sido levantada exitosamente, quedando lista para ser validada manualmente o mediante herramientas automatizadas.

Una vez completado el despliegue de la infraestructura con Terraform, es fundamental realizar una **verificación manual de los componentes críticos**, en especial aquellos que forman parte del núcleo funcional de la solución: la **instancia EC2** que aloja el servicio WordPress y la **base de datos RDS**.

Antes de proceder con la validación manual, es recomendable **inspeccionar los valores de salida generados por Terraform**, los cuales proporcionan información clave sobre los recursos desplegados, como direcciones IP, nombres de instancias, identificadores de red y endpoints de bases de datos.

Para listar todos los outputs definidos, se ejecuta.

```
Outputs:

rds_endpoint = "terraform-20250408144354551200000005.cota62seis3c.us-east-1.rds.amazonaws.com:3306"
wordpress_private_ip = "30.0.3.146"
```

Como las instancias EC2 (incluyendo la de WordPress) están ubicadas dentro de **subredes privadas**, no es posible acceder directamente a ellas desde internet. Para ello, se implementa un **servidor bastión** (jump host), ubicado en una subred pública, que permite realizar conexiones seguras vía SSH a través de una sola puerta de entrada.

```
bastion.tf ×
Terraform_Project_Resources > terraform > bastion.tf
1   // --Instancia Bastion Host--
2
3   # Este recurso crea una instancia EC2 que actúa como Bastion Host.
4   # El Bastion Host se utiliza para acceder de forma segura a recursos privados dentro de la VPC.
5
6   resource "aws_instance" "bastion_host" {
7     ami           = "ami-0a38b8c18f189761a" # Amazon Linux 2 (AMI predefinida).
8     instance_type = "t2.micro"              # Tipo de instancia EC2 (pequeña y económica).
9     subnet_id     = aws_subnet.subnet1.id   # Subred pública donde se lanzará la instancia.
10    key_name      = aws_key_pair.mykey-pair.key_name # Par de claves para acceso SSH.
11
12    associate_public_ip_address = true     # Asocia una IP pública para acceso desde Internet.
13
14    vpc_security_group_ids = [aws_security_group.bastion.id] # Grupo de seguridad asociado al Bastion Host.
15
16    tags = {
17      Name = "Bastion Host"                # Etiqueta descriptiva para identificar la instancia.
18    }
19  }
20
```

Se agrega un nuevo recurso al proyecto Terraform para desplegar el bastión. Este se define con una instancia EC2, un grupo de seguridad específico y una IP pública asociada. Una vez añadido el código, se aplica con.

```
devsecops@PC-Cristian:~/Terraform_Project_Resources/terraform$ terraform apply -target=aws_instance.bastion_host
aws_key_pair.mykey-pair: Refreshing state... [id=my-keypair]
aws_vpc.vpc: Refreshing state... [id=vpc-061ad9872e3c79495]
```

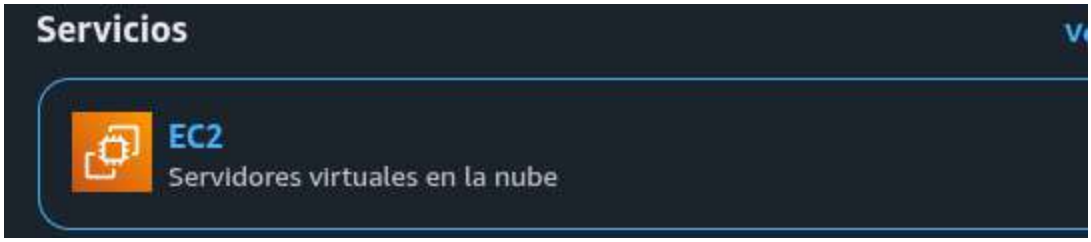
Tras el despliegue, se puede consultar la IP pública del bastión con.

```
● devsecops@PC-Cristian:~/Terraform_Project_Resources/terraform$ terraform show | grep public_ip

    public_ip                = "52.73.92.72"
    public_ipv4_pool         = "amazon"
    public_ip                = "52.7.96.157"
    public_ipv4_pool         = "amazon"
    associate_public_ip_address          = true
    public_ip                            = "100.29.7.201"
    public_ip            = "52.73.92.72"
    public_ip            = "52.7.96.157"
```

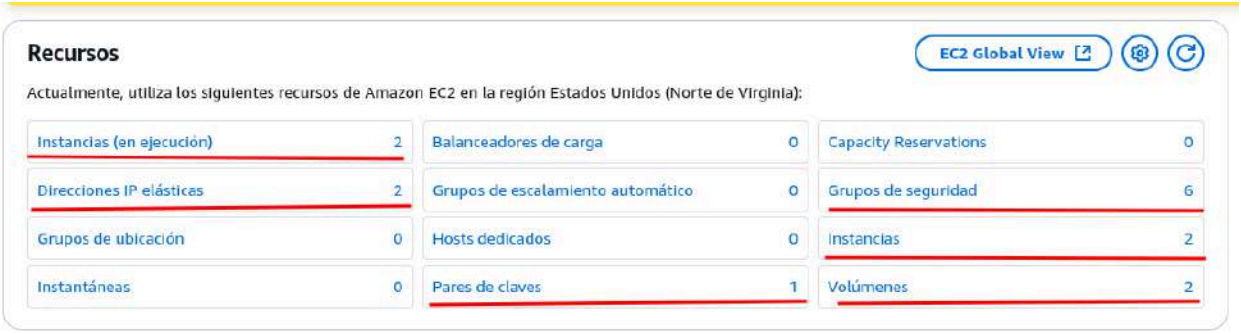
Una vez obtenida la IP del bastión, nos conectamos a él mediante SSH.

```
○ devsecops@PC-Cristian:~/Terraform_Project_Resources/terraform$ ssh -i ./mykey-pair ec2-user@100.29.7.201
The authenticity of host '100.29.7.201 (100.29.7.201)' can't be established.
ED25519 key fingerprint is SHA256:G5VAhBnSroJzPez+BEx4+bS5O5sko0TvyAgvp2EqUqw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '100.29.7.201' (ED25519) to the list of known hosts.
       ,     #_
       ~\_  ####_        Amazon Linux 2
      ~~  \_#####\
      ~~     \###|       AL2 End of Life is 2026-06-30.
      ~~       \#/ ___
       ~~       V~' '->
        ~~~         /    A newer version of Amazon Linux is available!
          ~~._.   _/
             _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
           _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-30-0-1-203 ~]$
```

También es posible verificar la existencia y estado de las instancias EC2 desde la consola web de AWS, para ello, realizamos la siguiente búsqueda en **AWS CLI** .



Una vez obtengamos acceso podemos observar todas los recursos activos, entre los que vemos **instancias, direcciones IP elásticas, pares de claves, grupos de seguridad, volúmenes etc.**



Posteriormente podremos observar que tenemos dos instancias levantadas, una dedicada al bastión y otra del propio EC2 de Wordpress.

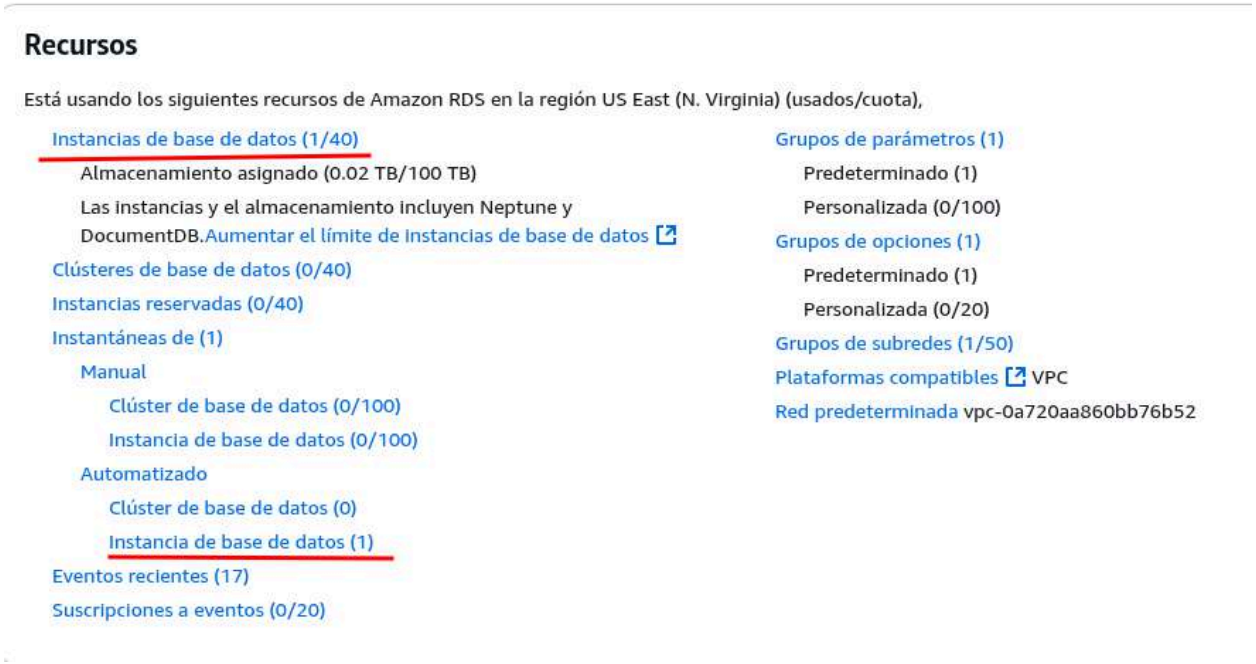
Al acceder a los detalles de la instancia WordPress, se podrá validar que está en estado **running** y revisar sus características: tipo de instancia, volúmenes asociados, SGs, zona de disponibilidad, etc.

Una vez desplegada la infraestructura, es crucial verificar que la **base de datos asociada al servicio WordPress** "provisionada a través de **Amazon RDS**" se encuentra correctamente configurada, en estado activo y lista para aceptar conexiones.

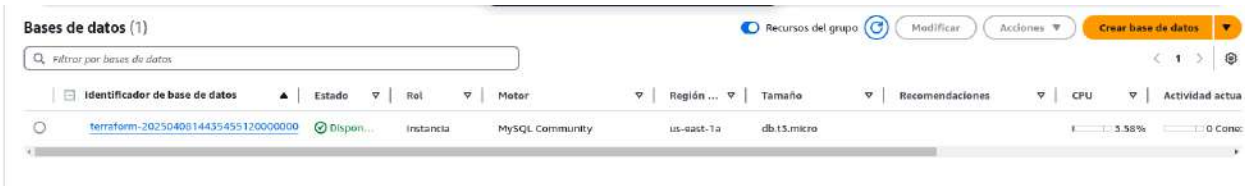
Para comprobar el estado de la base de datos RDS de **forma gráfica**, se accede a la consola web de AWS siguiendo estos pasos:

En el buscador superior, escribir y seleccionar **"RDS"** o **"Amazon RDS & Aurora"**.

Una vez entremos en detalle podremos comprobar todos los recursos usados entre los que se encuentra la base de datos.



Si nos adentramos más se nos mostrará una lista con todas las instancias de base de datos gestionadas en el entorno. Entre ellas debe encontrarse la instancia correspondiente al proyecto, por ejemplo **wordpress-db.**



También es posible comprobar el estado de la base de datos desde la línea de comandos, utilizando **AWS CLI**. Para ello, se ejecuta el siguiente comando.

```
devsecops@PC-Cristian:~/Terraform_Project_Resources/terraform$ aws rds describe-db-instances --region us-east-1
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "terraform-20250408144354551200000005",
            "DBInstanceClass": "db.t3.micro",
            "Engine": "mysql",
            "DBInstanceStatus": "available",
            "MasterUsername": [redacted],
            "DBName": [redacted],
            "Endpoint": {
                "Address": "terraform-20250408144354551200000005.cota62seis3c.us-east-1.rds.amazonaws.com",
                "Port": 3306,
                "HostedZoneId": "Z2R2ITUGPM61AM"
            },
            "AllocatedStorage": 20,
            "InstanceCreateTime": "2025-04-08T14:47:32.082000+00:00",
            "PreferredBackupWindow": "08:35-09:05",
```

Este comando devolverá un resumen con el identificador y el estado de cada base de datos en la región actual. La salida esperada es similar a la anterior.

A través de esta doble verificación —visual en la consola web y técnica mediante AWS CLI— se garantiza que la **base de datos RDS está correctamente desplegada, configurada y activa**, y por tanto, apta para integrarse con el resto de la infraestructura (como el servidor de aplicación WordPress).

Fase 4: Cumplimiento normativo

¿Qué es el cumplimiento normativo?

El **cumplimiento normativo** es un componente fundamental en el diseño, despliegue y mantenimiento de entornos de infraestructura tecnológica, especialmente cuando se trabaja en entornos productivos, en la nube o con datos sensibles. En el contexto de **Infraestructura como Código (IaC)**, el cumplimiento normativo implica asegurarse de que toda la infraestructura desplegada cumpla con las políticas, leyes, normativas técnicas y estándares de seguridad requeridos por el entorno regulatorio correspondiente.

La implementación de IaC con herramientas como **Terraform** (para aprovisionamiento) y **Ansible** (para configuración del sistema) permite **automatizar, auditar y estandarizar** la infraestructura, lo que facilita significativamente el cumplimiento de dichos requisitos.

¿Qué implica el cumplimiento normativo en infraestructura?

A nivel técnico, el cumplimiento normativo implica aplicar controles y medidas en diversas áreas, tales como:

- Seguridad de la información.
- Gestión de identidades y accesos.
- Protección de datos personales.
- Registro y trazabilidad de cambios.
- Reducción de la superficie de exposición.
- Control de configuraciones y vulnerabilidades.

Estas medidas se alinean con marcos como:

- **ISO/IEC 27001** → Seguridad de la información.
- **GDPR** → Reglamento General de Protección de Datos.
- **NIST 800-53 / NIST CSF** → Controles de ciberseguridad.
- **PCI-DSS** → Requisitos para manejo de datos de tarjetas.
- **CIS Benchmarks** → Guías de configuración segura.
- **AWS Well-Architected Framework** → Pilar de Seguridad.

Terraform Compliance

Terraform Compliance es una herramienta de auditoría que permite validar que las configuraciones definidas en Terraform se ajustan a un conjunto de políticas preestablecidas. Su funcionamiento se basa en ejecutar pruebas de cumplimiento sobre el archivo de plan **(terraform plan)** exportado en formato **JSON**.

Las reglas se escriben en un lenguaje estructurado y fácil de entender, conocido como **Gherkin** (similar al usado en pruebas BDD), lo cual facilita su lectura y mantenimiento incluso por parte de equipos no técnicos.

La herramienta puede instalarse fácilmente utilizando **pipx**, un entorno aislado recomendado para utilidades CLI en Python.

```
devsecops@PC-Cristian:~/Project_Resources$ pipx install terraform-compliance
  installed package terraform-compliance 1.3.50, installed using Python 3.11.2
  These apps are now globally available
    - terraform-compliance
⚠ Note: '/home/devsecops/.local/bin' is not on your PATH environment variable. These apps will not be glob
    add it, or manually modify your PATH in your shell's config file (i.e. ~/.bashrc).
done! ✨ 🌟 ✨
```

Primero se debe ejecutar el plan habitual de Terraform, pero guardando la salida en un archivo binario **(plan.out)**.

```
devsecops@PC-Cristian:~/Project_Resources$ terraform plan -out plan.out
aws_key_pair.mykey-pair: Refreshing state... [id=my-keypair]
aws_vpc.vpc: Refreshing state... [id=vpc-0e4405e2735fcad2c]
data.aws_ami.ubuntu: Reading...
data.aws_ami.linux2: Reading...
```

A continuación, se debe transformar el plan binario en un archivo **JSON** legible por Terraform Compliance.

```
devsecops@PC-Cristian:~/Project_Resources$ terraform show -json plan.out > plan.json
```

Este archivo **plan.json** será la entrada sobre la cual se ejecutarán las validaciones.

Una vez que se ha generado el archivo **plan.json**, se puede ejecutar la herramienta apuntando al directorio que contiene las reglas escritas en Gherkin:

```
● devsecops@PC-Cristian:~/Project_test_code/terraform$ terraform-compliance -p plan.json -f ../tests/rules
  terraform-compliance v1.3.50 initiated

  🚩 Features     : /home/devsecops/Project_test_code/tests/rules/
  🚩 Plan File    : /home/devsecops/Project_test_code/terraform/plan.json

  🚩 Running tests. 🎉
```

Regla: vpc.feature

Su objetivo es garantizar que se está creando **al menos una VPC**, lo cual es esencial para encapsular la infraestructura en una red privada definida. Esta regla es útil en entornos que deben cumplir con segmentación de red lógica, control de tráfico interno, y aislamiento entre entornos **(por ejemplo, dev/prod)**.

```
vpc.feature ×
Terraform_Project_Resources > tests > rules > vpc.feature
1   Feature: Red y subredes
2
3     Scenario: VPC CIDR debe ser 30.0.0.0/16
4       Given I have aws_vpc defined
5       Then it must contain cidr_block
6       And its value must be "30.0.0.0/16"
7
```

Esta regla está alineada con controles de **segmentación de red segura**, **reducción de superficie de ataque**, y con normas como:

- NIST 800-53 AC-4 (Information Flow Enforcement).
- ISO 27001 A.13.1.1 (Segregation in networks).

Tras ejecutar el test debemos esperar esta salida que indica que todos los pasos que contiene la regla han sido superados con éxito.

```
Feature: Validar red VPC  # /home/devsecops/Project_test_code/tests/rules/vpc_cidr.feature

    Scenario: La VPC tiene el bloque CIDR correcto
        Given I have aws_vpc defined
        Then it must contain cidr_block
        And its value must be "30.0.0.0/16"
```

Regla: vpc_cidr.feature

Esta regla garantiza que el bloque CIDR de la VPC cumpla con un rango específico de red privada, siguiendo buenas prácticas de direccionamiento.

```
vpc_cidr.feature ×
Terraform_Project_Resources > tests > rules > vpc_cidr.feature
1   Feature: Validar red VPC
2
3     Scenario: La VPC tiene el bloque CIDR correcto
4       Given I have aws_vpc defined
5       Then it must contain cidr_block
6       And its value must be "30.0.0.0/16"
7
```

Los rangos privados definidos por RFC 1918 son estándar en la industria para evitar el uso de direcciones públicas, previniendo colisiones y garantizando el aislamiento. Esta validación responde a controles como:

- ISO 27001 A.13.1.3 (Segregation in networks).
- CIS AWS Benchmark v1.4 - Section 2.1.

Tras ejecutar el test debemos esperar esta salida que indica que todos los pasos que contiene la regla han sido superados con éxito.

```
Feature: Red y subredes  # /home/devsecops/Project_test_code/tests/rules/vpc.feature

    Scenario: VPC CIDR debe ser 30.0.0.0/16
        Given I have aws_vpc defined
        Then it must contain cidr_block
        And its value must be "30.0.0.0/16"
```

Regla: sg.feature

Esta regla tiene como objetivo evitar que **puertos sensibles (como SSH, 22)** estén abiertos a cualquier IP (0.0.0.0/0), ya que esto representa una vulnerabilidad crítica en la seguridad perimetral del sistema. Esta regla previene accesos no autorizados desde internet.

```
sg.feature ×
Terraform_Project_Resources > tests > rules > sg.feature
1   Feature: Grupos de Seguridad
2
3     Scenario: Permitir tráfico desde cualquier IP
4       Given I have aws_security_group defined
5       When it has cidr_blocks
6       Then its value must contain "0.0.0.0/0"
7
```

Tras ejecutar el test debemos esperar esta salida que indica que todos los pasos que contiene la regla han sido superados con éxito.

```
Feature: Grupos de Seguridad  # /home/devsecops/Project_test_code/tests/rules/sg.feature

    Scenario: Permitir tráfico desde cualquier IP
        Given I have aws_security_group defined
        When it has cidr_blocks
        Then its value must contain "0.0.0.0/0"
```

Esta regla es clave para aplicar el **principio de mínimo privilegio** y evitar exposiciones innecesarias. Su cumplimiento está alineado con:

- CIS AWS Benchmark v1.4 - Section 4.1.
- NIST 800-53 SC-7 (Boundary Protection).
- ISO/IEC 27001 A.13.1.1.

Tras la ejecución completa del conjunto de reglas, Terraform Compliance mostrará un **resumen detallado del resultado de cada test**, señalando si todas las políticas se cumplen.

```
3 features (3 passed)
3 scenarios (3 passed)
9 steps (9 passed)
Run 1743612877 finished within a moment
```

AWS Config

AWS Config es un servicio gestionado de AWS que permite **auditar, monitorizar y evaluar de forma continua las configuraciones** de los recursos dentro de una cuenta AWS. Su objetivo principal es facilitar el **cumplimiento normativo**, la **gestión de cambios** y la **detección de desviaciones** con respecto a configuraciones deseadas o políticas de seguridad definidas por la organización.

En este proyecto, se ha implementado el servicio **AWS Config** utilizando **Infrastructure as Code (IaC)** con Terraform. Esta implementación permite llevar a cabo un **seguimiento continuo y automatizado del estado de configuración** de los recursos desplegados en la cuenta de AWS, lo cual es fundamental para garantizar el **cumplimiento normativo, la seguridad operativa y la trazabilidad** de todos los cambios en infraestructura.

La configuración está dividida en dos archivos principales:

- **aws_config.tf** → Define los recursos base del servicio AWS Config, necesarios para registrar, almacenar y activar la auditoría.

- **aws_config_rules.tf** → Contiene las reglas de cumplimiento que evaluarán si los recursos siguen las políticas de seguridad y buenas prácticas establecidas.

Esta estructura modular facilita la **organización y mantenimiento** del código, separando la configuración base del servicio de las reglas que lo gobiernan.

La implementación de **AWS Config con reglas gestionadas mediante Terraform** ofrece una solución robusta y escalable para cumplir con estándares de seguridad, auditoría y gobernanza en entornos de nube. Gracias a esta arquitectura, es posible mantener un monitoreo continuo y automatizado de la infraestructura, alineándose con buenas prácticas corporativas y requisitos regulatorios modernos.

A continuación, se destacan partes del código de ambos archivos.

Archivo: aws_config.tf

aws_config_configuration_recorder → Activa el componente que se encarga de **registrar cada cambio** que ocurre sobre los recursos AWS.

```
resource "aws_config_configuration_recorder" "recorder" {
  name     = "default" // Nombre del grabador de configuración.
  role_arn = aws_iam_role.config_role.arn // ARN del rol IAM que AWS Config utilizará.

  recording_group {
    all_supported                 = true // Graba todos los tipos de recursos compatibles.
    include_global_resource_types = true // Incluye recursos globales como IAM.
  }

  depends_on = [aws_s3_bucket_policy.config_logs_policy] // Garantiza que la política del bucket esté configurada antes.
}
```

aws_config_delivery_channel → Define el medio por el cual AWS Config **envía los registros de configuración**, en este caso a un bucket de S3.

```
// --Canal de entrega para AWS Config--
// Este recurso configura un canal de entrega para enviar los datos de AWS Config al bucket S3.
resource "aws_config_delivery_channel" "channel" {
  name           = "default" // Nombre del canal de entrega.
  s3_bucket_name = aws_s3_bucket.config_logs.bucket // Nombre del bucket S3 donde se almacenarán los datos.
  depends_on     = [aws_config_configuration_recorder.recorder] // Garantiza que el grabador esté configurado antes.
}
```

Archivo: aws_config_rules.tf

restricted_ports → Esta regla verifica que no se permite tráfico SSH entrante por SSH.

```
# 2. Regla para puertos restringidos
# Esta regla verifica que no se permita tráfico SSH entrante (puerto 22).
resource "aws_config_config_rule" "restricted_ports" {
  name = "restricted-ssh" // Nombre de la regla.

  source {
    owner             = "AWS"                       // AWS es el propietario de la regla.
    source_identifier = "INCOMING_SSH_DISABLED"     // Identificador de la regla predefinida.
  }

  depends_on = [aws_config_configuration_recorder_status.recorder_status] // Garantiza que el grabador esté habilitado antes.
}
```

Una vez desplegados los recursos definidos en los archivos **aws_config.tf** y **aws_config_rules.tf,** es **AWS Config quien se encarga de ejecutar las reglas automáticamente**, sin necesidad de intervención manual.

Este servicio funciona de forma **gestionada y continua**, es decir, no necesita un servidor dedicado para su operación y no requiere la ejecución de scripts adicionales por parte del usuario.

Fase 5: Gestión de Parches y Configuración Segura

La Fase 5 se centra en garantizar que los servidores desplegados como parte de la infraestructura estén **debidamente actualizados y configurados de forma segura**, en línea con estándares de seguridad y cumplimiento. Para esta tarea se utiliza **Ansible**, una herramienta de automatización que permite definir políticas de seguridad y mantenimiento mediante **playbooks** escritos en YAML.

Ansible es especialmente adecuado para esta fase porque permite aplicar medidas de seguridad de manera **declarativa, reproducible y escalable**, sin necesidad de agentes, aprovechando el acceso remoto por SSH.

Para ello ansible utiliza el archivo inventory (también conocido como **"archivo de inventario"**) es un **componente esencial en Ansible**, ya que define **los hosts o grupos de hosts** sobre los que se van a ejecutar los playbooks. Es, por decirlo de forma simple, **el "catálogo de máquinas gestionadas"**.

Ansible necesita saber **a qué nodos conectarse y cómo**, y eso lo define el archivo de inventario.

```
inventory.ini ×
Ansible > inventory.ini
1  [bastion]
2  [redacted] ansible_user=ec2-user ansible_ssh_private_key_file=~/Project_Resources/terraform/mykey-pair
3
```

El archivo inventory es una **pieza central de la arquitectura de Ansible**: organiza, clasifica y parametriza la ejecución de tareas sobre los servidores. En la Fase 5 de Gestión de Parches y Configuración Segura, el inventario garantiza que las políticas de seguridad se apliquen de forma **ordenada, selectiva y repetible**, especialmente en entornos donde existen múltiples tipos de nodos (EC2 públicos, privados, bastiones, etc.).

Gestión de parches

Mantener los servidores actualizados es un componente **esencial en cualquier estrategia de seguridad y mantenimiento de infraestructura**, especialmente en entornos críticos o expuestos a Internet. La existencia de vulnerabilidades conocidas en versiones antiguas del sistema operativo o en paquetes instalados puede convertirse rápidamente en un vector de ataque si no se gestionan adecuadamente.

Por este motivo, se implementa un **playbook de Ansible dedicado a la actualización automatizada del sistema operativo**, como parte de la Gestión de Parches y Configuración Segura.

```
---
# Este archivo es un playbook de Ansible diseñado para aplicar parches de seguridad
# y reiniciar un servidor bastion si es necesario.

- name: Aplicar parches al bastion
  hosts: bastion  # Define que este playbook se ejecutará en los hosts del grupo 'bastion'.
  become: yes     # Escala privilegios para ejecutar las tareas como usuario root.

```

El objetivo principal es garantizar que **todos los servidores gestionados mantengan su sistema operativo y paquetes al día**, reduciendo así el riesgo de explotación de vulnerabilidades conocidas y asegurando una base confiable sobre la cual operar servicios y aplicaciones.

Este proceso está completamente automatizado y puede ejecutarse de forma periódica.

```
tasks:
  # Primera tarea: Actualizar todos los paquetes en el sistema operativo.
  - name: Actualizar todos los paquetes (Amazon Linux)
    yum:
      name: "*"          # Indica que se deben actualizar todos los paquetes instalados.
      state: latest      # Asegura que los paquetes estén en su última versión disponible.

  # Segunda tarea: Reiniciar el sistema si es necesario.
  - name: Reiniciar si es necesario
    reboot:
      msg: "Reinicio automático tras aplicar parches de seguridad"  # Mensaje que se mostrará durante el reinicio.
      reboot_timeout: 600  # Tiempo máximo de espera (en segundos) para que el sistema se reinicie.
```

Esta tarea asegura que cualquier vulnerabilidad publicada y corregida por el fabricante del sistema sea mitigada de forma inmediata.

Configuración segura (hardering)

El **hardening** (o "configuración segura") es el proceso de **reforzar la seguridad de un sistema operativo o servicio**, mediante la aplicación de una serie de configuraciones técnicas que reducen su superficie de ataque. El objetivo es **minimizar los riesgos** de seguridad, eliminar servicios innecesarios, deshabilitar accesos inseguros y aplicar políticas que dificulten la explotación de vulnerabilidades.

En entornos modernos, este proceso debe ser **automatizado, auditable y reproducible**. Para ello, Ansible se convierte en una herramienta ideal, permitiendo aplicar políticas de seguridad mediante **playbooks estructurados**, de forma consistente y a escala.

```
---
# Este playbook de Ansible implementa medidas de hardening en un servidor bastion basado en Amazon Linux 2.

- name: Hardening de servidor bastion - Amazon Linux 2
  hosts: bastion  # Define que las tareas se ejecutarán en los hosts del grupo 'bastion'.
  become: yes     # Escala privilegios para ejecutar las tareas como usuario root.
```

Con el objetivo de reforzar la seguridad de los servidores recién desplegados, se implementará un **playbook de Ansible** que aplique una serie de medidas fundamentales de **configuración segura (hardening)**. Estas medidas están orientadas a reducir la superficie de ataque, proteger los accesos remotos, y establecer una línea base de cumplimiento en el sistema operativo.

A continuación se detallan las tareas que compondrán este playbook, ordenadas según su ejecución lógica y agrupadas por funcionalidad:

Una de las primeras acciones de endurecimiento consiste en **prohibir el acceso remoto como superusuario (root)** a través del servicio SSH. Esta medida evita ataques de fuerza bruta dirigidos directamente a la cuenta con máximos privilegios.

```
tasks:
  # Primera tarea: Deshabilitar el login SSH como root.
  - name: Deshabilitar login SSH como root
    lineinfile:
      path: /etc/ssh/sshd_config  # Archivo de configuración de SSH.
      regexp: '^PermitRootLogin' # Busca la línea que comienza con 'PermitRootLogin'.
      line: 'PermitRootLogin no' # Establece que el login como root no está permitido.
      create: yes                # Crea la línea si no existe.
    notify: Reiniciar SSH        # Notifica al handler para reiniciar el servicio SSH.

  # Segunda tarea: Instalar firewalld.
  - name: Instalar firewalld
    yum:
      name: firewalld            # Nombre del paquete a instalar.
      state: present             # Asegura que el paquete esté instalado.
```

Se instalará el servicio firewalld, que permite definir reglas de firewall de forma dinámica, persistente y gestionable con Ansible y se añadirá una regla para **permitir el tráfico SSH (puerto 22)**, asegurando que las conexiones de administración remota no se vean interrumpidas:

```
  # Tercera tarea: Habilitar e iniciar firewalld.
  - name: Habilitar e iniciar firewalld
    systemd:
      name: firewalld            # Nombre del servicio.
      enabled: true              # Habilita el servicio para que inicie automáticamente.
      state: started             # Asegura que el servicio esté en ejecución.

  # Cuarta tarea: Permitir tráfico SSH en firewalld.
  - name: Permitir tráfico SSH en firewalld
    firewalld:
      service: ssh               # Habilita el servicio SSH en el firewall.
      permanent: true            # Hace que la regla sea persistente tras reinicios.
      state: enabled             # Asegura que la regla esté habilitada.
      immediate: true            # Aplica la regla inmediatamente.
```

Para fortalecer la protección contra ataques de fuerza bruta en servicios como SSH, se instalará el repositorio **EPEL** (Extra Packages for Enterprise Linux), necesario para disponer de ciertos paquetes de seguridad, posteriormente se instalará **Fail2Ban**, una herramienta que bloquea direcciones IP tras múltiples intentos fallidos de autenticación:

```
# Quinta tarea: Instalar el repositorio EPEL para paquetes adicionales.
- name: Instalar fail2ban
  yum:
    name: epel-release          # Instala el repositorio EPEL.
    state: present              # Asegura que esté instalado.

# Sexta tarea: Instalar fail2ban.
- name: Instalar fail2ban core
  yum:
    name: fail2ban              # Nombre del paquete fail2ban.
    state: present              # Asegura que esté instalado.
```

Se activará y habilitará el servicio, ademas, se procederá a desinstalar software considerado **inseguro o innecesario** en servidores de producción, como herramientas de administración remota no cifradas (telnet, ftp, etc.):

```
# Séptima tarea: Activar fail2ban.
- name: Activar fail2ban
  systemd:
    name: fail2ban              # Nombre del servicio.
    enabled: true               # Habilita el servicio para que inicie automáticamente.
    state: started              # Asegura que el servicio esté en ejecución.

# Octava tarea: Eliminar paquetes innecesarios (ejemplo: telnet).
- name: Eliminar paquetes innecesarios (telnet como ejemplo)
  yum:
    name: telnet                # Nombre del paquete a eliminar.
    state: absent               # Asegura que el paquete esté desinstalado.
```

Por motivos de cumplimiento y trazabilidad, se establecerá un **banner legal** que se mostrará a cualquier usuario que intente iniciar sesión en el servidor vía SSH. Este banner actúa como advertencia formal de uso autorizado.

```
# Novena tarea: Configurar un banner legal.
- name: Configurar banner legal
  copy:
    content: |                  # Contenido del banner.
      ¡ATENCIÓN! Acceso restringido. Toda actividad será registrada y supervisada.
    dest: /etc/issue.net        # Archivo donde se guardará el banner.
    owner: root                 # Propietario del archivo.
    group: root                 # Grupo del archivo.
    mode: '0644'                # Permisos del archivo.

# Décima tarea: Activar el uso del banner en SSH.
- name: Activar uso del banner en SSH
  lineinfile:
    path: /etc/ssh/sshd_config # Archivo de configuración de SSH.
    regexp: '^Banner'          # Busca la línea que comienza con 'Banner'.
    line: 'Banner /etc/issue.net' # Establece la ruta del banner.
    create: yes                # Crea la línea si no existe.
  notify: Reiniciar SSH        # Notifica al handler para reiniciar el servicio SSH.
```

Para aplicar los cambios relacionados con el acceso remoto y el banner legal, se reiniciará el servicio **sshd**. Esta acción se definirá como un **handler** invocado por las tareas que modifican la configuración de SSH:

```
handlers:
  # Handler para reiniciar el servicio SSH cuando sea notificado.
  - name: Reiniciar SSH
    systemd:
      name: sshd                  # Nombre del servicio SSH.
      state: restarted            # Reinicia el servicio.
```

Pruebas en el entorno de producción

Una vez desplegada la infraestructura base y definidos los correspondientes playbooks de Ansible que incluyen las tareas de **gestión de parches, configuración segura (hardening)** y medidas de refuerzo de seguridad, el siguiente paso consiste en **aprovisionar los servidores** aplicando dichas configuraciones de manera automatizada.

```
devsecops@PC-Cristian:~/Project_Resources/Ansible$ ansible-playbook -i inventory.ini patch-bastion.yml

PLAY [Aplicar parches al bastion] ***********************************************************************
```

Este proceso garantiza que todos los sistemas inicien su ciclo de vida operativo con un **estado base seguro, actualizado y homogéneo**, cumpliendo con los estándares técnicos y normativos establecidos.

Para ejecutar los playbooks definidos previamente, se utiliza el comando **ansible-playbook**, indicando el archivo de inventario **(inventory)** que contiene los grupos de hosts objetivo y el playbook específico que se desea aplicar.

```
devsecops@PC-Cristian:~/Project_Resources/Ansible$ ansible-playbook -i inventory.ini secure-bastion.yml

PLAY [Hardening de servidor bastion - Amazon Linux 2] ***************************************************

TASK [Gathering Facts] **********************************************************************************
```

Si el playbook se ejecuta correctamente, Ansible mostrará un resumen al final de la salida que indica el estado de cada host y de cada tarea. Un resultado esperado típico podría ser el siguiente:

```
ok: [ ]

TASK [Actualizar todos los paquetes (Amazon Linux)] *****************************************************
ok: [ ]

TASK [Reiniciar si es necesario] ************************************************************************
changed: [ ]

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

Fase 6: Escaneo de Vulnerabilidades

Despliegue de Nessus

¿Qué es Nessus?

Nessus es una herramienta de **escaneo de vulnerabilidades** desarrollada por la empresa **Tenable**. Su propósito principal es **identificar fallos de seguridad** en sistemas, redes y aplicaciones **antes de que puedan ser explotados** por atacantes.

Nessus se basa en una enorme cantidad de **plugins**, que son scripts especializados que simulan ataques, interrogan sistemas, detectan configuraciones peligrosas, verifican versiones de software vulnerable.

Estos plugins se actualizan casi **diariamente** por Tenable, permitiendo a Nessus estar siempre al día frente a nuevas amenazas.

¿Para qué sirve Nessus?

- **Detectar vulnerabilidades** → En sistemas operativos, servicios, puertos abiertos, software desactualizado, configuraciones inseguras, etc.
- **Escanear redes completas** → Entornos locales o en la nube.
- **Auditorías de cumplimiento** → (CIS Benchmarks, PCI DSS, HIPAA, etc.).
- **Identificar malware** → troyanos, puertas traseras y contraseñas débiles.
  Evaluar la seguridad de dispositivos IoT, routers, impresoras, servidores, etc.

Uso de Nessus en el proyecto

Como parte de las tareas de análisis y aseguramiento de la infraestructura, **utilizaremos la herramienta Nessus Essentials** con el objetivo de identificar y evaluar las posibles **vulnerabilidades presentes en los sistemas desplegados**.

El proceso de escaneo se realizará una vez finalizado el despliegue de los servidores mediante Terraform y su posterior configuración con Ansible, lo que nos permitirá auditar el estado de seguridad **antes y después de aplicar las políticas de hardening**.

Introducción al código: Terraform.

Como primer paso del proceso, es necesario **provisionar una nueva instancia virtual** que alojará la herramienta de análisis de vulnerabilidades. Se creará un nuevo archivo de configuración **".tf"** que describe los recursos necesarios para desplegar esta máquina en la nube.

```
nessus_instance.tf ×   inventory.ini   key_pair.tf   nessus-install.yml
terraform > nessus_instance.tf
1   # Configuración de la instancia EC2 para Nessus Essentials
2   # Este archivo define los recursos necesarios para desplegar una instancia EC2
3   # que ejecutará Nessus Essentials en AWS.
4
5   # Grupo de seguridad para la instancia de Nessus
6   resource "aws_security_group" "nessus" {
7     name        = "nessus-sg"  # Nombre del grupo de seguridad
8     description = "Security group for Nessus Essentials"  # Descripción del grupo
9     vpc_id      = aws_vpc.main.id  # VPC donde se creará el grupo
10
11    # Regla de entrada para SSH (puerto 22)
```

Un aspecto clave al diseñar el archivo de configuración es **la habilitación del tráfico de red necesario** para que la herramienta sea accesible y operativa. En particular, es imprescindible **permitir el tráfico entrante (ingress)** hacia el puerto web utilizado por la interfaz gráfica (por defecto, el puerto **8834** en el caso de Nessus), así como **el tráfico saliente (egress)** para que el sistema pueda conectarse con los servidores de Tenable y descargar actualizaciones de plugins y feeds de vulnerabilidades.

```
20    # Regla de entrada para Nessus (puerto 8834)
21    ingress {
22      from_port   = 8834
23      to_port     = 8834
24      protocol    = "tcp"
25      cidr_blocks = ["0.0.0.0/0"]  # Permite acceso a Nessus desde cualquier I
26      description = "Nessus web interface"
27    }
28
29    # Regla de salida para todo el tráfico
30    egress {
31      from_port   = 0
32      to_port     = 0
33      protocol    = "-1"  # Todos los protocolos
34      cidr_blocks = ["0.0.0.0/0"]  # Permite todo el tráfico saliente
35      description = "Allow all outbound traffic"
36    }
```

Una vez que la instancia ha sido levantada correctamente, se realizarán una serie de **ajustes de configuración inicial** que aseguren su correcto funcionamiento dentro del entorno del proyecto. Entre ellos se incluyen:

- El **cambio del hostname** para facilitar su identificación dentro de la infraestructura.

- La **instalación de dependencias básicas y utilidades del sistema**, incluyendo paquetes necesarios para el funcionamiento del escáner y sus procesos asociados.

- El **reinicio del servicio de red**, con el objetivo de aplicar correctamente los cambios de nombre de host y ajustes de conectividad.

```
# Script de inicialización que se ejecuta al crear la instancia
user_data = <<-EOF
            #!/bin/bash
            # Actualiza el sistema
            apt-get update
            apt-get upgrade -y

            # Instala dependencias básicas
            apt-get install -y python3-pip

            # Configura el hostname
            hostnamectl set-hostname nessus-scanner

            # Reinicia el servicio de red
            systemctl restart systemd-networkd
            EOF
```

Introducción al código:Ansible.

Una vez que la instancia ha sido correctamente desplegada y configurada a nivel de red, el siguiente paso consiste en **automatizar la instalación y puesta en marcha de Nessus,** para ello, se desarrollará un nuevo *playbook* en Ansible, una herramienta de automatización de configuración que nos permitirá gestionar esta instalación de manera repetible, coherente y **sin intervención manual.**

```
Ansible > ! nessus-install.yml
9   - name: Instalar y configurar Nessus Essentials
10    hosts: nessus-scanner  # Grupo de hosts donde se instalará Nessus
11    become: yes  # Ejecuta las tareas con privilegios de root
12    vars:
13      # Variables para la instalación
14      nessus_version: "10.8.4"  # Versión de Nessus a instalar
15      nessus_deb: "Nessus-{{ nessus_version }}-ubuntu1604_amd64.deb"  # Nombre del archivo .deb
16
```

Una de las primeras tareas definidas en el *playbook* será la **instalación de las dependencias necesarias** para el correcto funcionamiento de Nessus.

```
- name: Instalar dependencias necesarias
  apt:
    name:
      - curl  # Para descargar archivos
      - ufw   # Para configurar el firewall
      - default-jre  # Java Runtime Environment para Nessus
      - default-jdk  # Java Development Kit (por si es necesario)
    state: present
    update_cache: yes  # Actualiza la caché de paquetes
```

Posteriormente, el *playbook* procederá a **descargar el instalador oficial de Nessus Essentials**.

```
# Descarga el instalador de Nessus desde el sitio oficial
- name: Descargar Nessus Essentials
  get_url:
    url: "https://www.tenable.com/downloads/api/v2/pages/nessus/files/{{ nessus_deb }}"
    dest: "/tmp/{{ nessus_deb }}"
    mode: '0644'  # Permisos del archivo

# Instala Nessus usando dpkg con opciones forzadas
- name: Instalar Nessus usando dpkg con opciones forzadas
  shell: |
    # DEBIAN_FRONTEND=noninteractive evita prompts durante la instalación
    # --force-confdef --force-confold --force-remove-reinstreq fuerzan la instalación
    DEBIAN_FRONTEND=noninteractive dpkg --force-confdef --force-confold --force-remove-reinstreq -i /tmp/{{ nessus
  args:
    creates: /opt/nessus  # Evita reinstalación si el directorio existe
  register: dpkg_result
  ignore_errors: yes
```

Como buena práctica en la automatización de infraestructura, el *playbook* también incluirá **outputs al finalizar su ejecución**, que permitirán al usuario obtener información útil sin necesidad de inspeccionar manualmente la máquina.

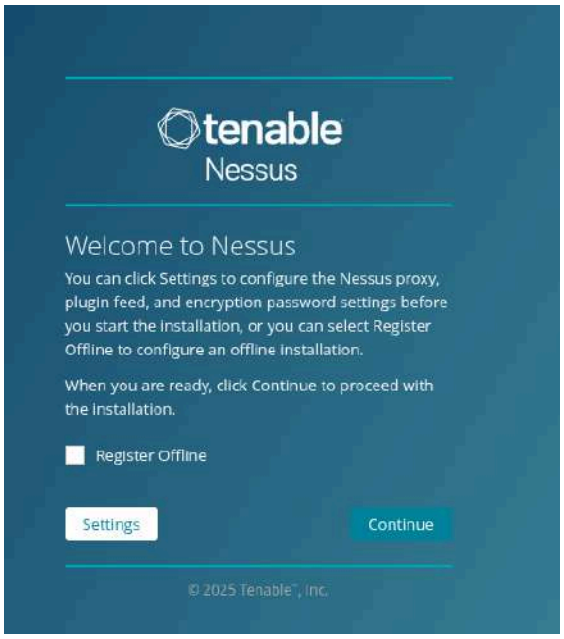
```
# Muestra instrucciones de acceso
- name: Mostrar instrucciones de acceso
  debug:
    msg: |
      Nessus Essentials ha sido instalado correctamente.

      Para acceder a Nessus:
      1. Abre tu navegador y ve a https://{{ ansible_host }}:8834
      2. Acepta el certificado autofirmado
      3. Crea una cuenta de administrador
      4. Activa Nessus Essentials con tu clave gratuita de Tenable

      Nota: La primera inicialización puede tardar varios minutos.
```

Finalización del Proceso de Instalación

Una vez completado el despliegue de la instancia y ejecutado el *playbook* de instalación, **Nessus Essentials estará disponible a través de su interfaz web**, accesible mediante el puerto seguro **8834**.

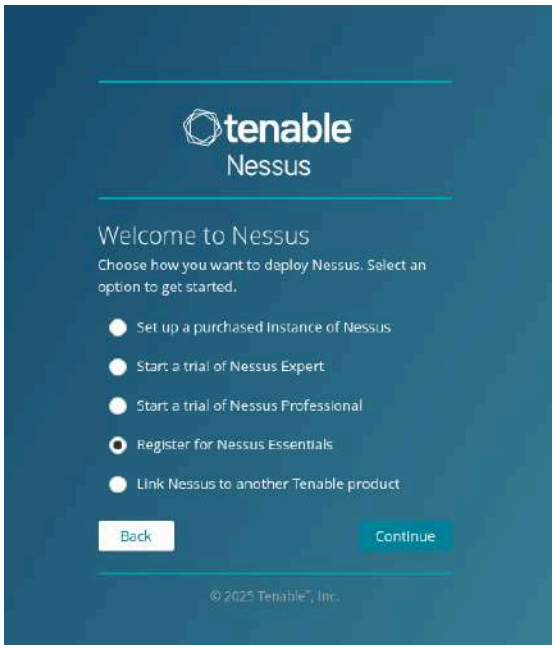
Es normal que en este momento el sistema muestre una pantalla indicando que **la instalación sigue en proceso,** este proceso puede tardar varios minutos, y es i**mportante esperar a que finalice completamente antes de continuar**.

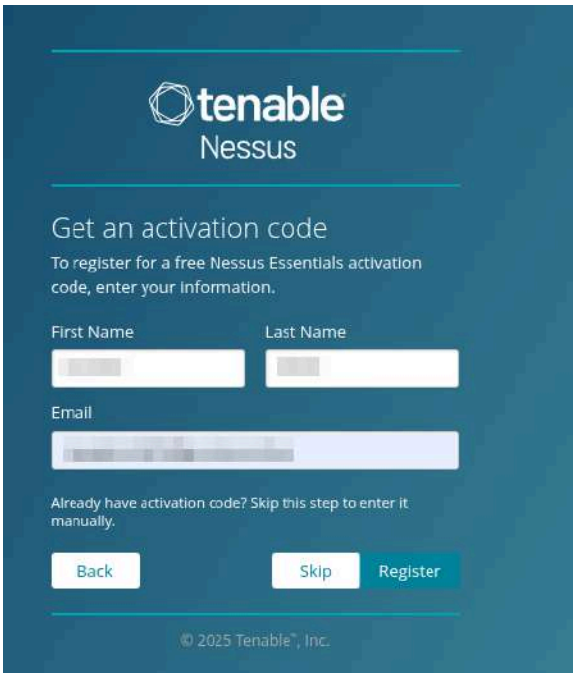
Una vez que la herramienta esté lista, se mostrará el **asistente de bienvenida de Nessus**. Este asistente nos guiará a través de los siguientes pasos:



1. Selección del tipo de licencia

Deberemos elegir el tipo de uso deseado para el entorno. En nuestro caso seleccionaremos la opción **Nessus Essentials,** esta versión gratuita es suficiente para el entorno de pruebas del proyecto, permitiendo el escaneo de hasta 16 hosts.





2. Inicio de sesión en el portal de Tenable

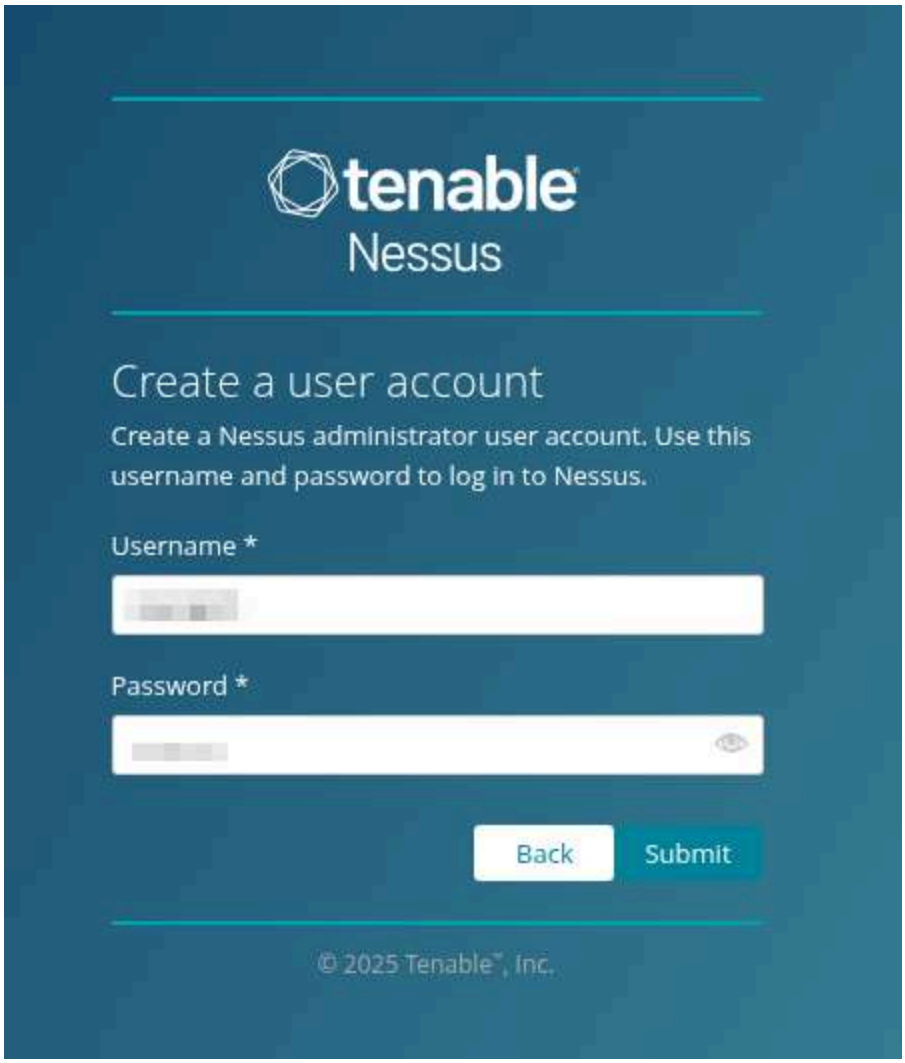
A continuación, será necesario **autenticarse en el portal de Tenable** utilizando una cuenta previamente registrada. Esto permite validar el uso de la herramienta y vincular la instalación con una clave de activación.

3. Introducción del código de licencia

Se solicitará el **código de activación** (Activation Code) proporcionado al registrarse en el sitio oficial de Nessus Essentials. Este código debe introducirse exactamente tal como fue recibido por correo electrónico.

4. Creación de un usuario administrador

En la siguiente pantalla se solicitará la creación de una cuenta de acceso. Este será el **usuario administrador local** que se usará para ingresar al panel de Nessus en adelante. Deberán establecerse el nombre de usuario y contraseña, los cuales podrán ser modificados posteriormente si es necesario.



Descarga y compilación del plugin feed

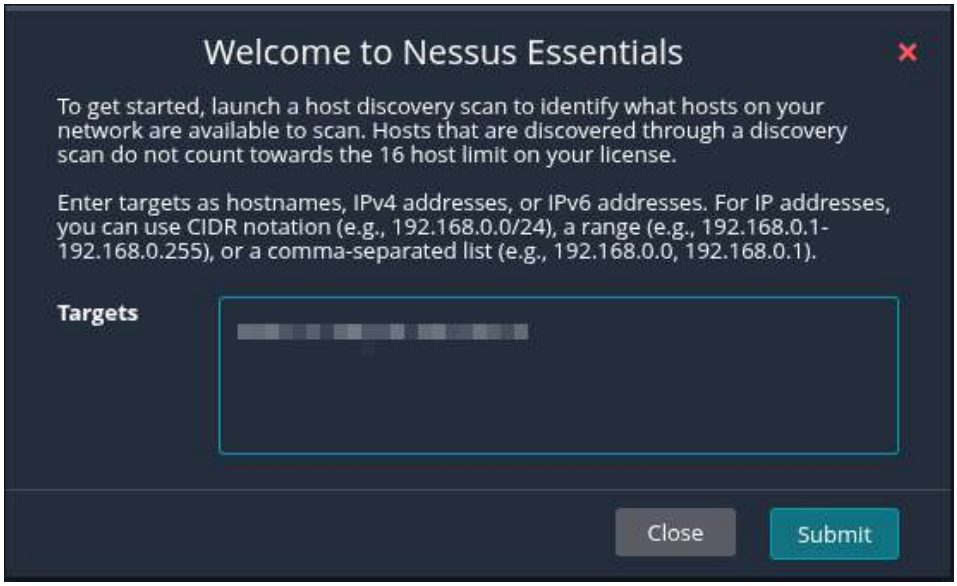
Una vez completados estos pasos, Nessus procederá automáticamente a:

- **Descargar el plugin feed completo**, que contiene miles de definiciones de vulnerabilidades.

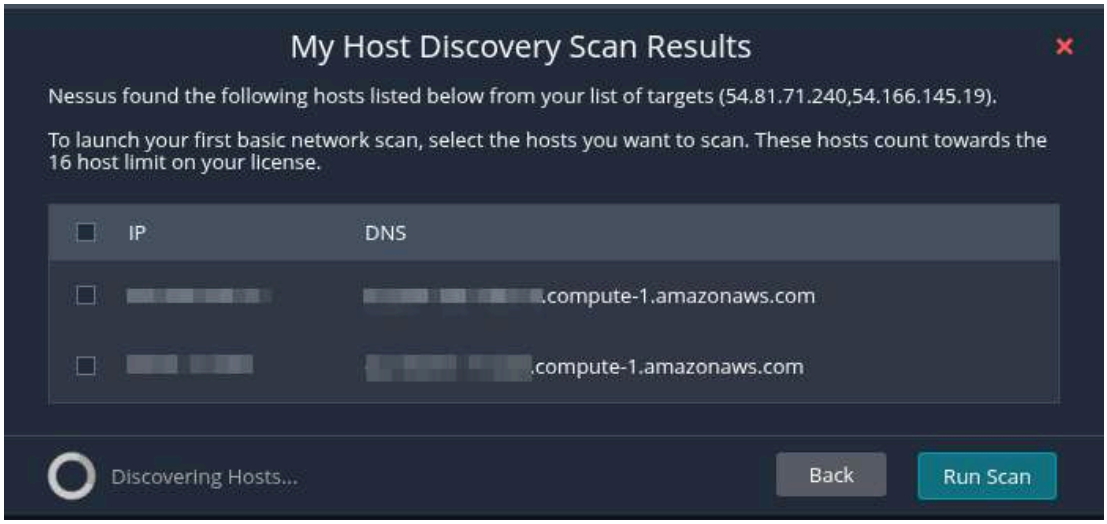
- **Compilar los plugins localmente**, proceso que puede durar varios minutos dependiendo de los recursos de la instancia.

Pruebas de Seguridad de la Infraestructura

El primer paso en el proceso consiste en añadir manualmente las direcciones IP de las demás instancias al escáner Nessus. Esta configuración inicial es fundamental para que la **herramienta pueda identificar los dispositivos** que serán objeto de análisis dentro de la infraestructura desplegada.

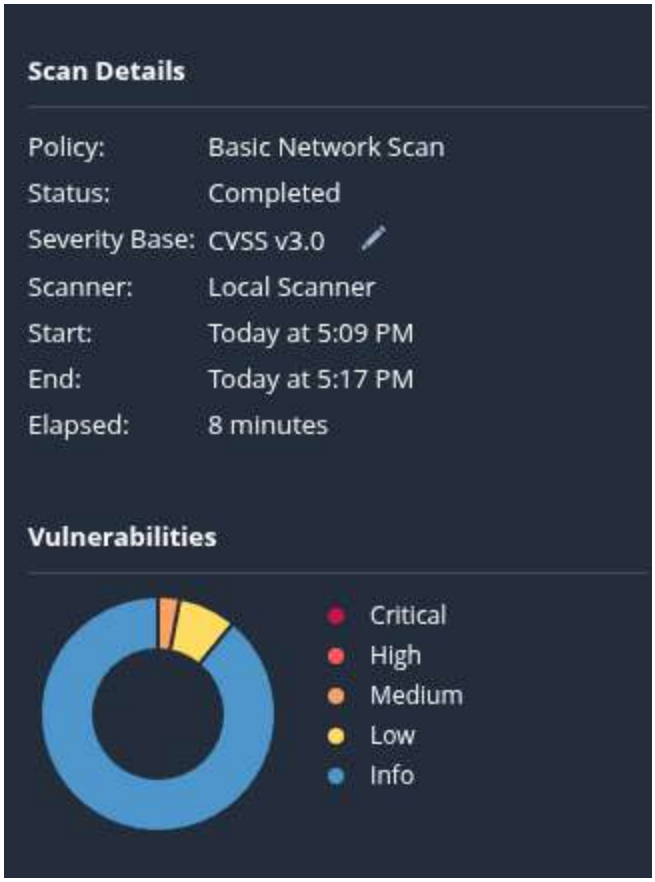


Una vez que se han añadido las IPs correspondientes, Nessus comenzará a **detectar automáticamente las instancias asociadas**. Esta detección se realiza a medida que la herramienta escanea el entorno de red, identificando los servicios activos, los puertos abiertos y otros elementos relevantes para la evaluación de seguridad.



Posteriormente, se procederá a realizar un **análisis de vulnerabilidades exhaustivo sobre todas las instancias detectadas**. A medida que se descubran nuevas instancias o servicios dentro del entorno, Nessus las irá categorizando automáticamente, asignándoles una evaluación específica según el tipo y criticidad de las vulnerabilidades encontradas.

Al finalizar el escaneo, Nessus generará una **representación gráfica que resume el nivel de peligrosidad de las vulnerabilidades detectadas**. Esta gráfica proporciona una visión global del estado de seguridad de la infraestructura, permitiendo identificar rápidamente los activos más comprometidos.



Además de la visión general, el usuario podrá **examinar cada vulnerabilidad en detalle**. La herramienta ofrece información ampliada sobre cada hallazgo, incluyendo su descripción, clasificación **(por ejemplo, según el CVSS)**, impacto potencial, vectores de ataque asociados y recomendaciones para su remediación.

Sev	CVSS	VPR	EPSS	Name
MIXED	...	...	...	SSL (Multiple Issues)
INFO	...	...	...	HTTP (Multiple Issues)
INFO	...	...	...	SSH (Multiple Issues)
INFO	...	...	...	SSH (Multiple Issues)
INFO	...	...	...	TLS (Multiple Issues)
INFO				Service Detection
INFO				Nessus SYN scanner
INFO				Additional DNS Hostnames
INFO				Common Platform Enumeration (CPE)
INFO				Device Type
INFO				Host Fully Qualified Domain Name (FQDN) Resolution

En aquellos casos en los que se detecten múltiples vulnerabilidades relacionadas con un mismo servicio o componente, Nessus las agrupará automáticamente dentro de una carpeta o sección específica. Esta funcionalidad facilita el análisis y la priorización de las acciones correctivas, permitiendo abordar de manera eficiente los problemas de seguridad más relevantes.

Vulnerabilities 19

Search Vulnerabilities 4 Vulnerabilities

Sev	CVSS	VPR	EPSS	Name
LOW	3.7	1.4	0.0307	SSH Server CBC Mode Ciphers Enabled
LOW	3.7			SSH Weak Key Exchange Algorithms Enabled
INFO				SSH Algorithms and Languages Supported
INFO				SSH SHA-1 HMAC Algorithms Enabled

Fase 7: Monitoreo y Detección de Amenazas

Despliegue de Wazuh

¿Qué es Wazuh?

Wazuh es una plataforma de código abierto para la detección de amenazas, monitoreo de integridad, respuesta ante incidentes y cumplimiento normativo.

Se basa en una arquitectura cliente-servidor que permite recopilar, **analizar y correlacionar información de seguridad** proveniente de diversos endpoints y dispositivos dentro de una red.

¿Para qué sirve Wazuh?

Wazuh tiene múltiples funcionalidades orientadas a mejorar la postura de seguridad de una organización, entre las que destacan:

- **Gestión de logs** → Recopila y analiza registros del sistema y de aplicaciones para identificar comportamientos anómalos.

- **Gestión de vulnerabilidades** →Identifica posibles fallos de seguridad en los sistemas mediante análisis periódicos.

- **Cumplimiento normativo** → Proporciona informes y auditorías orientadas a estándares como GDPR, PCI DSS, HIPAA, entre otros.

- **Respuesta ante incidentes** → Permite configurar alertas, acciones automatizadas y flujos de trabajo ante eventos de seguridad.

Uso de Wazuh en el proyecto

En el contexto del proyecto, Wazuh se implementa como una herramienta central para la **monitorización y protección de los sistemas desplegados**.

En resumen, Wazuh se convierte en un componente clave dentro de la arquitectura de seguridad del proyecto, proporcionando capacidades de detección, análisis y respuesta automática.

Introducción al código: Terraform.

Como primer paso, se procederá al despliegue de una nueva instancia destinada a alojar la solución de monitorización y detección de amenazas **Wazuh**. Es fundamental que esta instancia tenga conectividad tanto hacia la red externa como hacia la red interna:

- **Acceso a la red externa** → necesario para permitir el acceso al **dashboard web** de Wazuh desde el exterior, facilitando así la administración remota y la visualización de los datos recopilados.

- **Acceso a la red interna** → indispensable para que el sistema pueda comunicarse con el resto de las instancias desplegadas, lo que le permitirá recopilar logs, detectar anomalías y realizar un análisis centralizado de seguridad.

Esta configuración de red híbrida asegura que Wazuh pueda operar como un nodo central de supervisión, tanto para los administradores como para los dispositivos monitorizados.

wazuh_instance.tf × ! wazuh-install.yml inventory.ini

terraform > wazuh_instance.tf

```
1  # Instancia EC2 para Wazuh
2  resource "aws_instance" "wazuh" {
3    ami           = data.aws_ami.ubuntu.id
4    instance_type = "t3.medium"  # Recomendado para Wazuh
5    subnet_id     = aws_subnet.subnet1.id  # Subnet en la VPC correcta
6    key_name      = aws_key_pair.mykey-pair.key_name
7
8    root_block_device {
9      volume_size = 30
10     volume_type = "gp3"
11   }
12
13   tags = {
14     Name = "Wazuh-Server"
15   }
```

La arquitectura de Wazuh se compone principalmente de los siguientes tres módulos:

- **Wazuh Manager** → Núcleo del sistema, encargado de recibir, procesar y almacenar los datos de los agentes.
- **Wazuh API** → Interfaz de comunicación que permite la interacción entre el Manager y el Dashboard, así como con herramientas externas.
- **Wazuh Dashboard** → Entorno gráfico basado en Kibana que ofrece visualización avanzada de los eventos y alertas.

```
# Wazuh Manager
ingress {
  from_port   = 1514
  to_port     = 1514
  protocol    = "tcp"
  cidr_blocks = ["10.0.0.0/8"]  # Red interna
}

# Wazuh API
ingress {
  from_port   = 55000
  to_port     = 55000
  protocol    = "tcp"
  cidr_blocks = ["10.0.0.0/8"]  # Red interna
}

# Wazuh Dashboard
ingress {
  from_port   = 5601
  to_port     = 5601
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

Cada uno de estos componentes debe ser desplegado con las correspondientes **reglas de *ingress* y *egress*** en los grupos de seguridad o *security groups* asociados. Esto implica:

- **Reglas de ingress** → Definir qué tráfico entrante está permitido hacia cada componente, especificando protocolos, puertos y direcciones IP de origen autorizadas.

- **Reglas de egress** → Establecer qué tipo de tráfico saliente puede emitir cada componente, garantizando la conectividad necesaria sin comprometer la seguridad.

Introducción al código: Ansible.

Una vez desplegada y configurada la instancia destinada a alojar Wazuh, el siguiente paso consiste en instalar la herramienta propiamente dicha. Para llevar a cabo esta tarea de forma automatizada y reproducible, se desarrollará un **nuevo *playbook* de Ansible**, específicamente diseñado para gestionar todo el proceso de instalación y configuración.

Este **playbook** facilitará la ejecución ordenada de los distintos pasos necesarios, asegurando consistencia en los entornos y minimizando la posibilidad de errores manuales.

```
Ansible > ! wazuh-install.yml
1   ---
2   # Playbook para instalar y configurar Wazuh, Wazuh Indexer, Wazuh D
3   # Este playbook realiza la instalación completa de la plataforma Wa
4   # y la configuración de los servicios necesarios para el funcionami
5
6   - name: Instalar Wazuh
7     hosts: wazuh
8     become: yes
9     vars:
10      # Versiones de los componentes de Wazuh
11      wazuh_version: "4.7.0"
12      wazuh_indexer_version: "4.7.0"
13      wazuh_dashboard_version: "4.7.0"
```

Antes de proceder con la instalación de los componentes de Wazuh, es imprescindible asegurarse de que la instancia cuente con todas las **dependencias necesarias**. Estas incluyen:

```
# Instalación de dependencias necesarias para Wazuh
- name: Instalar dependencias
  apt:
    name:
      - apt-transport-https
      - curl
      - gnupg2
      - lsb-release
      - python3-pip
      - default-jre
      - openssl
    state: present
    update_cache: yes
```

La correcta instalación y configuración de estas dependencias es **crucial** para garantizar que los componentes de Wazuh se instalen de forma exitosa y funcionen adecuadamente.

Con las dependencias ya instaladas, se procederá a desplegar los componentes fundamentales del sistema Wazuh:

1. **Wazuh Manager** → Se instalará como el núcleo del sistema, encargado de recibir los datos de los agentes y generar alertas en base a reglas de correlación.

2. **Wazuh Indexer** → Motor de almacenamiento basado en OpenSearch, responsable de indexar los datos generados y permitir búsquedas eficientes.

3. **Wazuh Dashboard** → Entorno gráfico que se conectará al Indexer y al Manager para ofrecer una vista centralizada de los eventos de seguridad.

```
# Instalación del Wazuh Manager
- name: Instalar Wazuh Manager
  apt:
    name: wazuh-manager
    state: present
    update_cache: yes

# Instalación del Wazuh Indexer
- name: Instalar Wazuh Indexer
  apt:
    name: wazuh-indexer
    state: present
    update_cache: yes

# Instalación del Wazuh Dashboard
- name: Instalar Wazuh Dashboard
  apt:
    name: wazuh-dashboard
    state: present
    update_cache: yes
```

Cada uno de estos pasos estará reflejado en el *playbook*, asegurando que la instalación se realice de forma ordenada y con las configuraciones recomendadas por el fabricante.

Finalización del Proceso de Instalación

Una vez completado correctamente el proceso de instalación del servidor Wazuh mediante la ejecución del correspondiente playbook de Ansible, el sistema queda totalmente aprovisionado con todos los componentes necesarios: **Wazuh Manager, Wazuh API, Kibana (interfaz gráfica) y el stack de visualización basado en OpenSearch**.

El siguiente paso consiste en acceder al panel web de administración para verificar el funcionamiento inicial del sistema.

La interfaz presentará una pantalla de inicio de sesión, donde se deberán introducir las credenciales del usuario administrador, que por defecto suelen ser:

- **Usuario** → admin

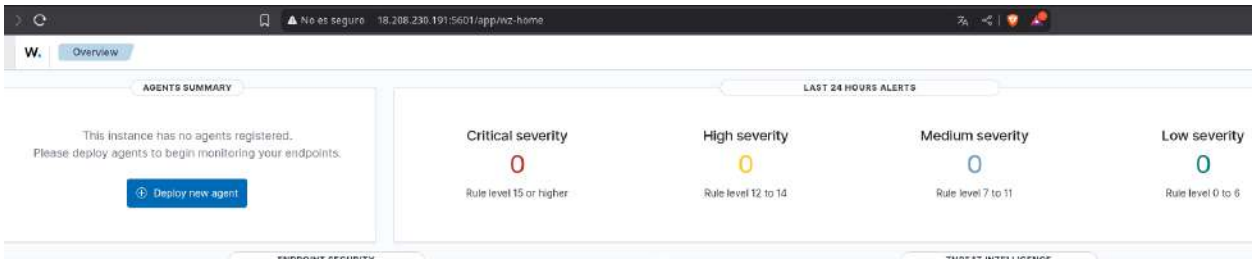
- **Contraseña** → Definida durante la instalación, o generada automáticamente **(se puede consultar en el archivo de variables o en la salida del playbook)**.

Una vez autenticado correctamente, se accede al dashboard principal de Wazuh.

La primera vez que se accede al panel, la interfaz gráfica estará completamente operativa, mostrando todos los elementos de control, pero **sin datos de actividad visibles aú**n. Esto se debe a que, en este punto, **ningún agente Wazuh está conectado o reportando información** al servidor.

<u>En este estado inicial se pueden observar:</u>

- El panel principal (Dashboard)
- La sección de agentes (Agents), donde aparecerá "0 connected"
- Secciones como Rules, Security Events, Logs, Vulnerabilities y Settings estarán vacías o sin datos.



El acceso exitoso al panel de Wazuh tras la instalación mediante Ansible confirma que el **stack de monitorización de seguridad se ha desplegado correctamente**.

Aunque en esta etapa inicial no se visualicen eventos, el sistema se encuentra completamente preparado para comenzar a recibir y analizar información, habilitando capacidades como detección de intrusiones, gestión de vulnerabilidades, correlación de eventos y cumplimiento normativo centralizado.

Agente de Wazuh

El **agente Wazuh** es un componente ligero que se instala en sistemas operativos **Linux, Windows o macOS**, cuyo propósito es **monitorear la actividad local del host**, recopilar datos de seguridad, integridad, configuración y comportamiento, y **enviar esa información al servidor Wazuh Manager** para su análisis centralizado.

```
Ansible > ! wazuh-agent-install.yml
  # 3. Descarga e instala el agente Wazuh
  # 4. Configura y inicia el servicio

  - name: Instalar agente Wazuh en todos los servidores
    hosts: all
    become: yes
    tasks:
      # Tarea 1: Mostrar información del sistema
      - name: Mostrar información del sistema
        debug:
          msg:
            - "Sistema operativo: {{ ansible_distribution }} {{ ansible_distribution_version }}"
            - "Familia: {{ ansible_os_family }}"
            - "Gestor de paquetes: {{ ansible_pkg_mgr }}"

```

El agente es fundamental para que el servidor Wazuh pueda recibir información relevante desde cada equipo o servidor bajo supervisión.

Para simplificar y estandarizar el proceso de instalación, se ha desarrollado un **playbook de Ansible que automatiza completamente la instalación del agente Wazuh**. Este playbook es capaz de detectar automáticamente el sistema operativo del host objetivo y, en función de ello, aplica el procedimiento de instalación adecuado.

```
# Tarea 3: Instalar dependencias en sistemas Debian/Ubuntu
- name: Instalar dependencias en Debian/Ubuntu
  apt:
    name:
      - curl
      - apt-transport-https
      - lsb-release
    state: present
    update_cache: yes
  when: ansible_os_family == "Debian" and "not_manager" in manager_check_debian.stdout

# Tarea 4: Instalar dependencias en sistemas RedHat/CentOS
- name: Instalar dependencias en RedHat/CentOS
  yum:
    name:
      - curl
      - wget
    state: present
  when: ansible_os_family == "RedHat" and "not_manager" in manager_check_redhat.stdout
```

El proceso de instalación del agente Wazuh está diseñado para ejecutarse en múltiples servidores con distintos sistemas operativos. Para ello, el playbook de Ansible implementa **estructuras condicionales (when) que detectan automáticamente la distribución del sistema operativo** del host remoto, y en función de ello, aplican las tareas correspondientes.

```
# Tarea 8: Verificar el estado del agente
- name: Verificar el estado del agente
  shell: systemctl status wazuh-agent
  register: agent_status
  changed_when: false
  when: ansible_os_family == "Debian" and "not_manager" in manager_check_debian.stdout or ansible_os_family == "RedHa

# Tarea 9: Mostrar el estado del agente
- name: Mostrar el estado del agente
  debug:
    var: agent_status.stdout_lines
  when: ansible_os_family == "Debian" and "not_manager" in manager_check_debian.stdout or ansible_os_family == "RedHa
```

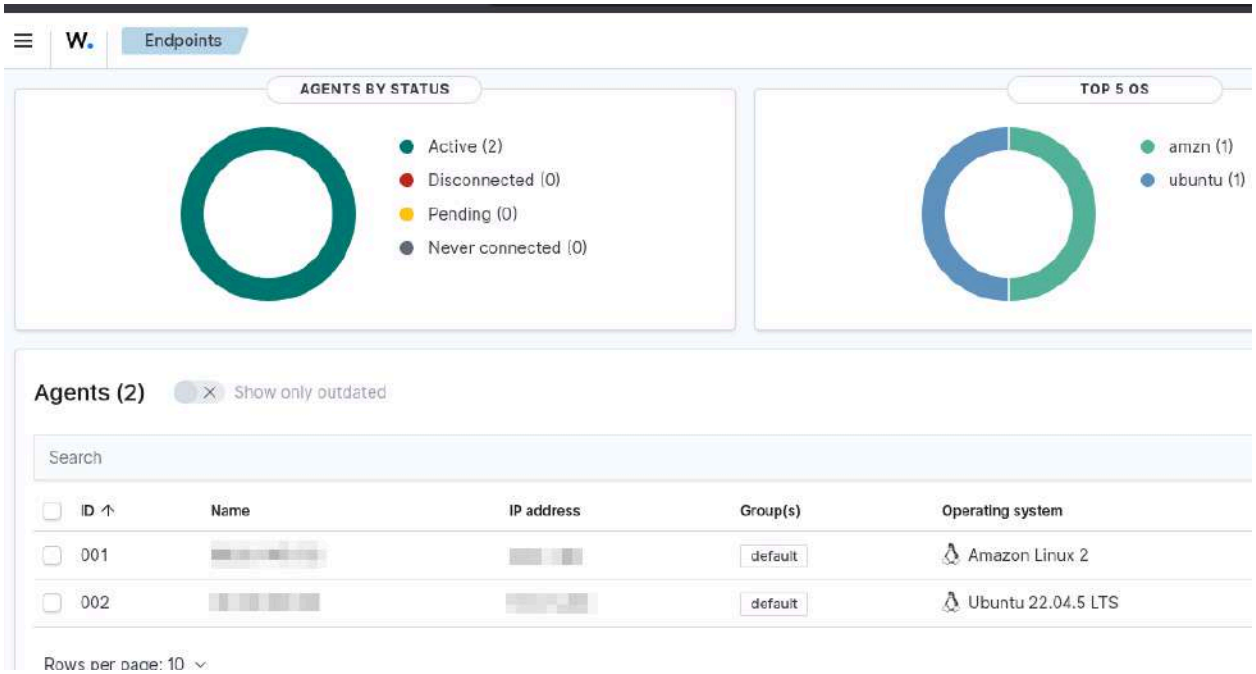
Para proceder con la instalación del agente en todos los servidores definidos en el inventario de Ansible, se ejecuta el siguiente comando desde la máquina de control:

```
devsecops@PC-Cristian:~/Project_Resources/Ansible$  ansible-playbook -i inventory.ini wazuh-agent-install.yml
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

PLAY [Instalar agente Wazuh en todos los servidores] ***************************************************************

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

Si la ejecución del playbook se completa correctamente, Ansible mostrará un resumen indicando el estado de cada máquina gestionada. Un ejemplo típico sería:

```
PLAY RECAP *********************************************************************
                 : ok=9    changed=0    unreachable=0    failed=0
                 : ok=3    changed=0    unreachable=0    failed=0
                 : ok=11   changed=0    unreachable=0    failed=0
```

Una vez completada la ejecución del playbook en todos los servidores, se procede a verificar el resultado de la instalación accediendo al **dashboard web de Wazuh**.



Inicialmente, los agentes aparecerán con estado **"Pending"** o **"Disconnected"** durante los primeros segundos. Sin embargo, una vez se establece la comunicación con el Wazuh Manager, comenzarán a mostrarse como **"Active"**.

La ejecución del playbook para instalar el agente Wazuh sobre múltiples servicios y servidores es una tarea **totalmente automatizada, controlada y auditable** gracias a Ansible. Gracias a la detección automática del sistema operativo y a la configuración dinámica, esta operación puede realizarse de forma masiva y segura, reduciendo el riesgo de errores humanos y asegurando una integración completa con el sistema de monitoreo centralizado.

Fase 8: Supervisión, Seguridad, Cumplimineto y Detección Continua de Amenazas

Monitorizar archivos y detectar cambios (FIM)

El módulo **FIM (File Integrity Monitoring)** es una de las funcionalidades más importantes de Wazuh para la detección temprana de amenazas y alteraciones no autorizadas en archivos críticos del sistema. Su función principal es **vigilar continuamente archivos y directorios específicos**, generando alertas cuando se detectan modificaciones, eliminaciones o creaciones de ficheros.

El módulo FIM debe estar **activado explícitamente** en la configuración del agente. Esto se realiza editando el archivo de configuración del mismo ubicado en **"/var/ossec/etc/ossec.conf"**.

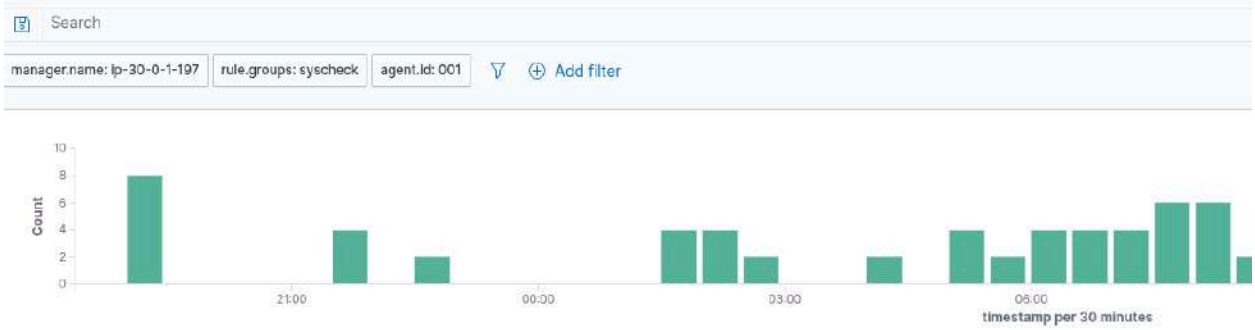
```
<!-- File integrity monitoring -->
<syscheck>
  <disabled>no</disabled>

  <!-- Frequency that syscheck is executed default every 12 hours -->
  <frequency>43200</frequency>

  <scan_on_start>yes</scan_on_start>

  <!-- Directories to check  (perform all possible verifications) -->
  <directories>/etc,/usr/bin,/usr/sbin</directories>
  <directories>/bin,/sbin,/boot</directories>
```

Con el módulo FIM activado, el siguiente paso es **verificar su funcionamiento mediante una prueba controlada**. Para ello, se procederá a **simular un cambio agresivo en un archivo monitorizado**, con el fin de generar una alerta que pueda ser visualizada en el panel de administración de Wazuh.

```
[ec2-user@ip-30-0-1-80 ~]$ echo "cambio agresivo $(date)" | sudo tee -a /etc/wazuh_fim_test > /dev/null
[ec2-user@ip-30-0-1-80 ~]$ sudo chmod 000 /etc/wazuh_fim_test
[ec2-user@ip-30-0-1-80 ~]$ sudo touch -d "yesterday" /etc/wazuh_fim_test
```

Una vez realizado el cambio, se puede acceder al **panel de administración web de Wazuh** para validar que el evento ha sido detectado correctamente.

El módulo FIM de Wazuh ofrece una capa crítica de detección ante manipulaciones no autorizadas de archivos, reforzando la postura de seguridad del sistema operativo. Su configuración mediante el agente es sencilla y, una vez activado, permite **monitorear la integridad del sistema en tiempo real**, alertando sobre eventos que podrían estar asociados a ataques, errores operativos o violaciones de políticas.



La prueba controlada de modificación demuestra que el sistema está funcionando correctamente y permite validar que la infraestructura está preparada para responder ante eventos reales.

timestamp	agent.name	syscheck.path	syscheck.event	rule.description
May 20, 2025 @ 18:17:16.9...	RHEL7	/tmp/wazuh-config	modified	Integrity checksum changed.
May 20, 2025 @ 18:17:16.9...	RHEL7	/tmp/wazuh-config	modified	Integrity checksum changed.
May 20, 2025 @ 18:17:15.9...	RHEL7	/var/wazuh/queue/fim/db/fim.db	modified	Integrity checksum changed.
May 20, 2025 @ 18:17:15.9...	RHEL7	/var/wazuh/queue/fim/db/fim.db	modified	Integrity checksum changed.
May 20, 2025 @ 18:11:25.5...	RHEL7	/var/osquery/osquery.db/CURRENT	modified	Integrity checksum changed.

Detección de intrusiones (HIDS)

Wazuh actúa como un **HIDS (Host-based Intrusion Detection System)**, es decir, un sistema de detección de intrusos a nivel de host. A diferencia de los sistemas de detección basados en red (NIDS), el HIDS funciona **desde dentro del sistema operativo**, permitiendo supervisar eventos y comportamientos internos.

```
<localfile>
  <log_format>audit</log_format>
  <location>/var/log/audit/audit.log</location>
</localfile>
```

Para validar la capacidad del módulo HIDS de Wazuh, se puede simular un **ataque típico basado en fuerza bruta**, consistente en múltiples intentos fallidos de autenticación SSH.

```
devsecops@PC-Cristian:~/Project_Resources/Ansible$ for i in {1..6}; do ssh usuarioinvalido@5[redacted]; done
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
usuarioinvalido@[redacted]: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

Esto se puede lograr ejecutando un **bucle de conexiones SSH con credenciales incorrectas**, generando así actividad anómala que debe ser detectada por el agente.

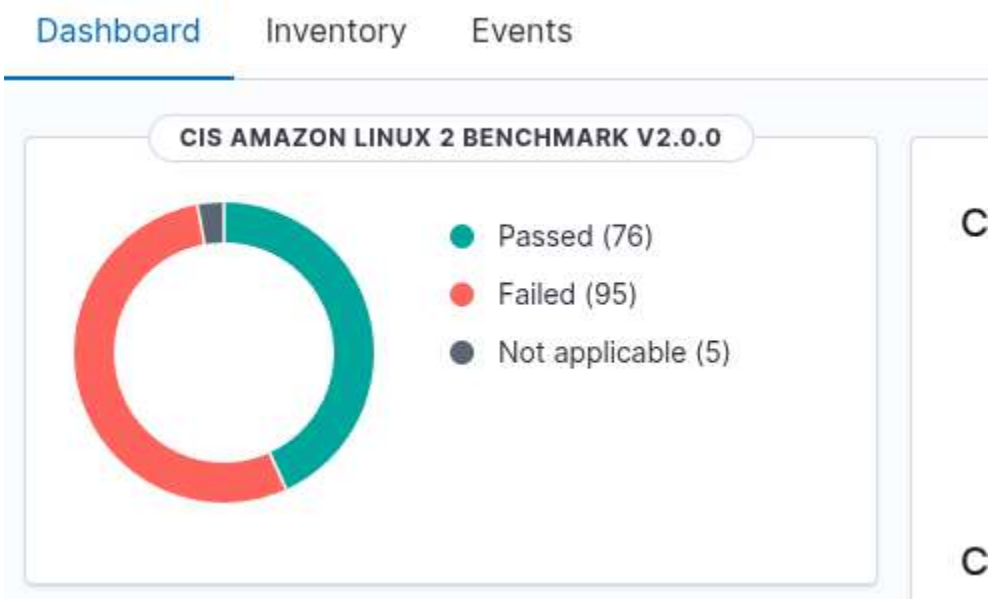
Una vez realizados los intentos de conexión fallidos, el agente Wazuh recopila los registros generados y los envía al **Wazuh Manager**, donde se evalúan contra la **base de reglas de correlación**.

timestamp	_source
May 20, 2025 @ 17:40	analysisd.total_events_decoded: 6,124 analysisd.sshd_bruteforce_detected: 2

Wazuh detectará la actividad anómala como un posible intento de intrusión y generará una alerta que se visualizará en el **panel de administración web**.

Configuration Assessment

El **Configuration Assessment** (también conocido como **compliance monitoring** o **security configuration assessment**) es un módulo de Wazuh diseñado para **evaluar la configuración de los sistemas monitoreados**, comparándola con políticas, normativas y buenas prácticas de seguridad establecida.



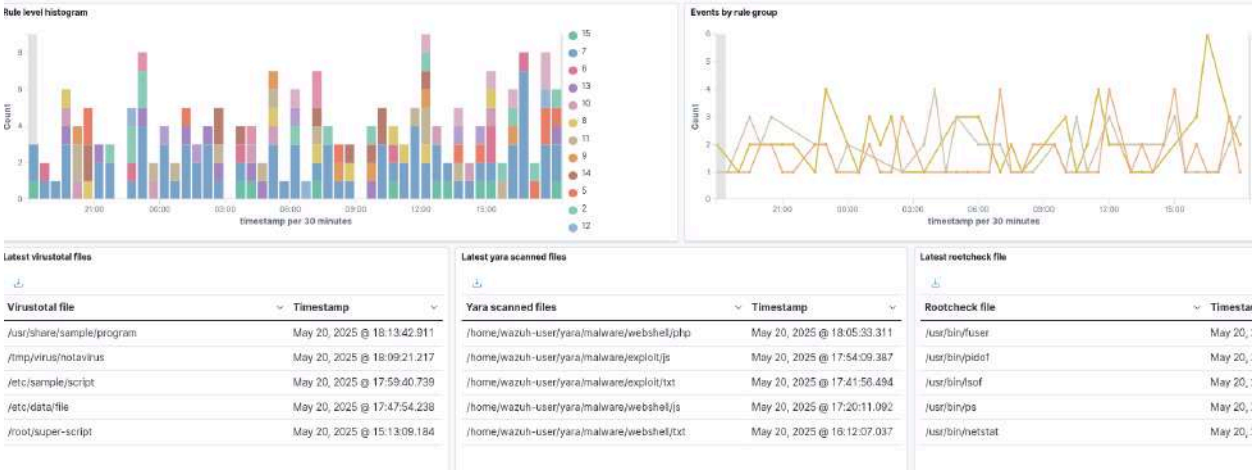
Una vez activado el módulo en el agente, Wazuh realiza escaneos regulares utilizando scripts de comprobación basados en el sistema operativo y servicios detectados. Estos escaneos generan resultados que son enviados al Wazuh Manager, donde son **correlacionados, almacenados y visualizados** a través del panel de control.

Los resultados de las evaluaciones realizadas por el módulo **Configuration Assessment** pueden visualizarse directamente desde el **panel de administración web de Wazuh**, en la sección específica dedicada al cumplimiento.

ID ↑	Title	Target	Result
20500	Ensure mounting of cramfs filesystems is disabled.	**Command:** modprobe -n -v cramfs	Failed
20501	Ensure mounting of squashfs filesystems is disabled.	**Command:** modprobe -n -v squashfs	Failed
20502	Ensure mounting of udf filesystems is disabled.	**Command:** modprobe -n -v udf	Failed
20503	Ensure /tmp is configured.	**File:** /etc/fstab	Failed
20504	Ensure noexec option set on /tmp partition.	**Command:** findmnt -n /tmp	Passed
20505	Ensure nodev option set on /tmp partition.	**Command:** findmnt -n /tmp	Passed

Malware Detection

El módulo de **Malware Detection** en Wazuh es una funcionalidad integrada que permite a los agentes instalados en los sistemas supervisar, identificar y reportar **presencia o indicios de malware** en los archivos y procesos locales.

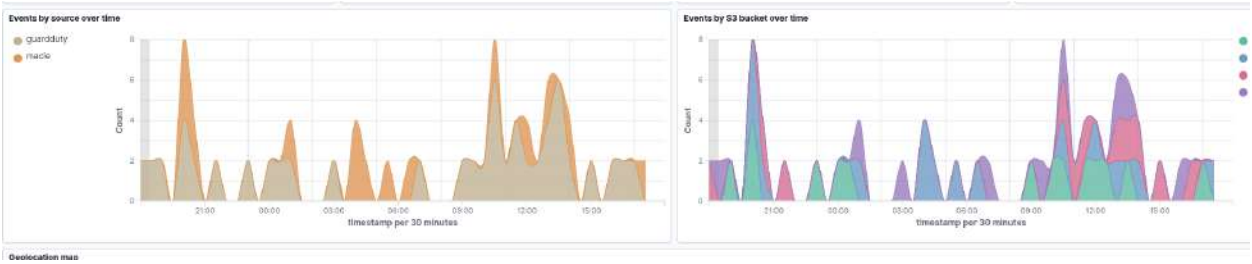


Una vez habilitado el módulo y completados los escaneos en los distintos agentes, los **resultados se presentan de forma centralizada en el panel de administración de Wazuh**, dentro del módulo.

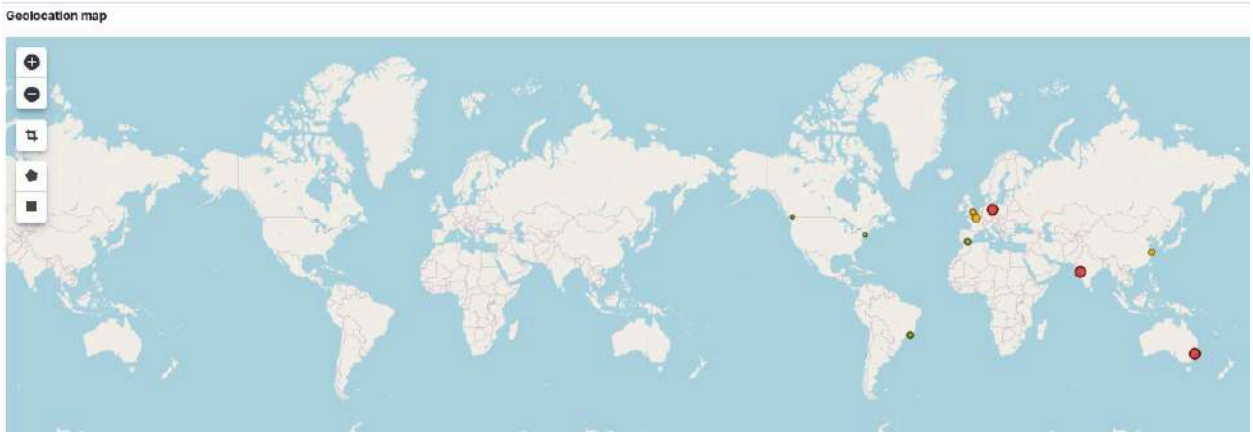
El módulo de **detección de malware de Wazuh** representa una **capa adicional de defensa proactiva**, que permite detectar infecciones o amenazas en los endpoints antes de que generen impacto.

Cloud Security de Wazuh para AWS

El módulo **Cloud Security de Wazuh para AWS** permite integrar, monitorizar y auditar la actividad de una cuenta de Amazon Web Services, proporcionando una **visión de seguridad completa de los recursos desplegados en la nube**. Esta funcionalidad se basa en el análisis de los logs nativos que ofrece AWS para la detección de anomalías, accesos no autorizados, errores de configuración y eventos sospechosos en servicios críticos.



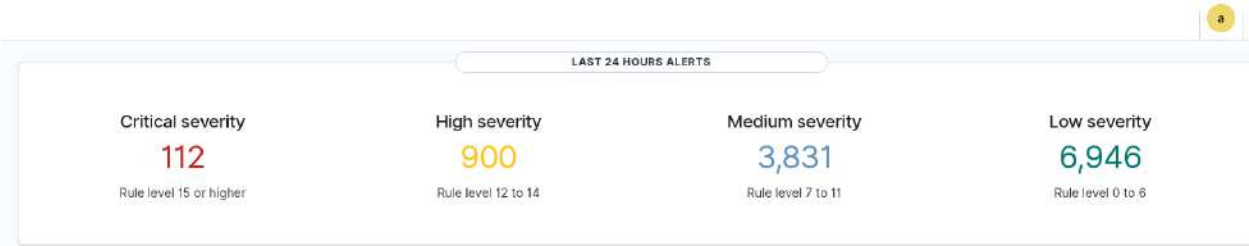
Una vez habilitada la integración con AWS y configurado el módulo correspondiente, los **eventos de seguridad cloud se presentan visualmente en el panel web de Wazuh**, de forma centralizada y organizada por cuentas, servicios y tipo de evento.



Wazuh también proporciona un **mapa geográfico interactivo**, que permite visualizar la **ubicación de los servidores, agentes y eventos registrados**, incluyendo aquellos generados por instancias EC2 u otros servicios de AWS.

Resultados: Alertas de Seguridad Detectadas en el Primer Mes de Operación

Tras un mes completo de funcionamiento en producción, la infraestructura desplegada ha sido capaz de **detectar y clasificar de forma efectiva múltiples eventos de seguridad**, sin que se haya registrado ningún impacto negativo en los sistemas. Esto ha sido posible gracias a la integración de mecanismos avanzados de **seguridad activa, automatización y cumplimiento normativo**, diseñados e implementados desde las primeras fases del proyecto.



Gracias a la configuración segura aplicada con herramientas como **Ansible**, al uso de **Infraestructura como Código con Terraform** y a la integración de los módulos de **detección, control de integridad y cumplimiento normativo** de Wazuh, el sistema ha logrado:

- Detectar intentos de intrusión y accesos no autorizados
- Identificar configuraciones inseguras
- Monitorizar cambios en archivos sensibles
- Supervisar la actividad en la nube (AWS)
- Prevenir amenazas asociadas a malware

Lo más importante es que **ninguno de estos eventos ha producido daños en los sistemas, interrupciones de servicio ni accesos comprometidos**, lo cual **demuestra la solidez de las medidas de defensa en profundidad** diseñadas para el entorno.

Webgrafía

Inteligencias artificiales usadas para el proyecto.

Chat GPT

Gemini.

Copilot.

Páginas Webs usadas en el proyecto.

Documentación oficial de AWS.

Referencias técnicas sobre infraestructuras.

Documentación oficial Wazuh.

Documentación oficial Ansible.

Documentación oficial Nessus.

API reference Nessus.

Repositorio Wazuh.

Ansible AWS Guide.

Proveedores de servicios usados para el proyecto.

AWS.

Google Drive

Contacto con el creador del proyecto.