Universidad de Buenos Aires

Facultades de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería


---

# Carrera de Especialización en Seguridad Informática

## Trabajo Final

### Herramientas para el aprovisionamiento automático de infraestructuras aplicadas a la seguridad

AUTOR: PABLO EMANUEL REYES Y ESCURRA

TUTOR: JAVIER J. VALLEJOS MARTINEZ

2025

Cohorte 2023

# Declaración Jurada de origen de los contenidos:

“Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual”.

Nombre: Pablo Emanuel Reyes y Escurra

DNI: 33019276

FIRMADO

# Resumen

El aumento desmedido de los sistemas informáticos y, con él, el de la ciberdelincuencia, hace cada vez más necesaria la implementación de herramientas de automatización en cuestiones de seguridad de la infraestructura. Estas herramientas proporcionan una forma más rápida, eficiente y precisa de prevenir como así también a responder a los ataques cibernéticos. Además, ayudan a gestionar y monitorear de manera más eficiente los sistemas, reduciendo el riesgo de errores humanos.

En un mundo donde la seguridad informática es cada vez más importante, la automatización se presenta como una entidad indispensable para proteger de manera efectiva la información y los sistemas.

El objetivo de este trabajo es introducir el concepto automatización de la seguridad. Se va a definir lo que es IaC (Infraestructura como código) analizando la herramienta Ansible pero centrándonos principalmente en la seguridad de los servicios y servidores. Se realizará un estudio de tipo explicativo y descriptivo, en donde se analizarán y detallarán las características principales de las herramientas de automatización y sus funcionalidades.

*Palabras clave: automatización, DevOps, DevSecOps, ansible, seguridad*

# Índice

# Introducción

A medida que aumenta el tamaño y la complejidad de la infraestructura y las redes, es cada vez más difícil gestionar la seguridad y el cumplimiento de forma manual. Las operaciones manuales pueden ralentizar el proceso de detección de problemas, la resolución de los mismos, dar lugar a errores en la configuración de los recursos e impedir que haya uniformidad en la aplicación de las políticas, lo cual expone a los sistemas a problemas de cumplimiento y ataques. Esto puede generar tiempos de inactividad imprevistos, altos costos y la reducción general del funcionamiento. La automatización puede ayudar a optimizar las operaciones cotidianas y a integrar la seguridad con la infraestructura de TI (Tecnologías de la información), los procesos, las arquitecturas de nube híbrida y las aplicaciones desde el comienzo.

# Objetivos

- Desarrollar el concepto de Automatización de la Seguridad.
- Explorar y analizar las distintas herramientas para esta disciplina.
- El objetivo principal de este trabajo es analizar la utilización de Ansible como herramienta dentro de la cultura de *DevSecOps* para las actividades de implementación continua, la puesta a punto y el *hardening* de servidores.

# Alcance

Toda la investigación se hará con una base abierta a cualquier sistema operativo, pero siempre pensando en la implementación a futuro sobre sistemas GNU/Linux.

Se analizarán las distintas herramientas de código abierto que se encuentran disponibles a la fecha para el aprovisionamiento automático de servicios.
Se realizarán ejemplos prácticos para mostrar la utilización de la herramienta como pueden ser: la generación de inventarios, la ejecución de algunos comandos, la instalación y configuración simple de algún servicio en particular, etc.

# 1. Automatización de la seguridad

En este capítulo se abordará el primer objetivo de este trabajo. Para ello es fundamental conocer o reforzar algunos conceptos básicos. Se indagará sobre qué es DevOps, cuáles son sus ventajas, características y su evolución a DevSecOps. Finalmente, se definirá el concepto de automatización de la seguridad y su importancia dentro de este enfoque de trabajo.

## 1.1. Conceptos básicos

Para poder definir a qué se llama automatización, y más específicamente aplicada a la seguridad informática, se debe realizar una rápida introducción por lo que se llama el Ciclo de Vida del Desarrollo de Software (SDLC) y cómo ha ido evolucionando a lo largo del tiempo. Luego de esto se llegará a la cultura DevOps, o ampliada a DevSecOps y cómo esta se relaciona con la automatización. Finalmente se puede mencionar de qué forma Ansible ayuda dentro de este modelo.
La evolución del ciclo de vida del desarrollo de software ha sido un proceso continuo y constante en busca de mejorar la eficiencia y la calidad en la entrega de software. A lo largo de los años, distintas metodologías han surgido, cada una con su enfoque particular.


**Imagen 1: Fases del ciclo de vida del desarrollo del software**

Inicialmente, el desarrollo de software seguía un enfoque en cascada, donde cada etapa del proceso era llevada a cabo de manera secuencial y se realizaba una entrega final del software. Sin embargo, este enfoque presentaba limitaciones, ya que los cambios tardíos en los requisitos del cliente eran difíciles de incorporar y los errores se detectaban y corregían en etapas tardías del proceso, lo que resultaba en un ciclo de desarrollo lento y costoso.

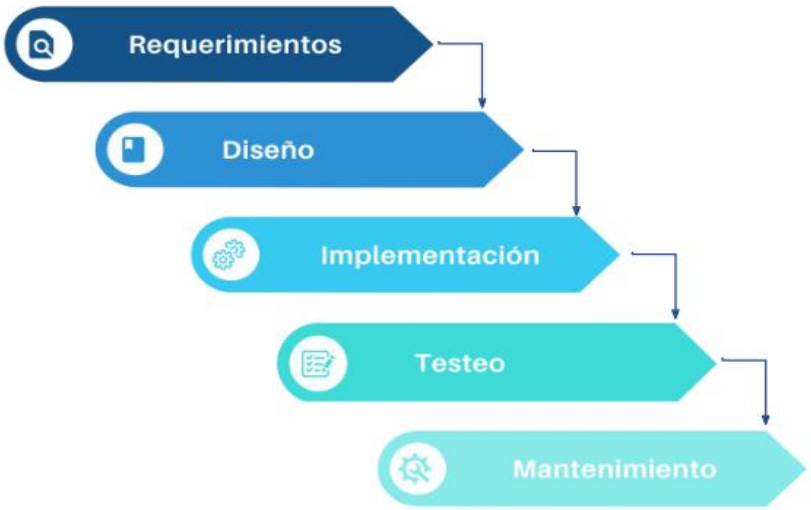


**Imagen 2: Ciclo de vida en cascada**

Con el tiempo, surgió la metodología ágil, que permitía una mayor flexibilidad al adaptarse rápidamente a los cambios en los requisitos y centrarse en entregas parciales y frecuentes. Esto permitió un mayor grado de colaboración y comunicación entre los equipos de desarrollo y los clientes, lo que condujo a una mejora en la calidad del software y una reducción en los tiempos de entrega.

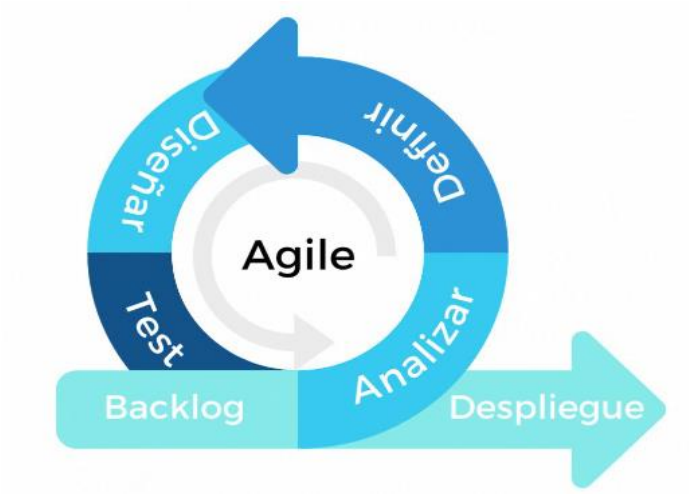


**Imagen 3: Ciclo de vida en metodologías ágiles**

Sin embargo, a pesar de los avances logrados con las metodologías ágiles, surgieron nuevos desafíos a medida que los sistemas y las aplicaciones se volvían más complejos. Ahí es donde entra en juego DevOps, una práctica que combina el desarrollo (Development) y las operaciones (Operations) en un enfoque unificado y colaborativo.

DevOps busca eliminar las barreras entre los equipos de desarrollo y operaciones, fomentando una cultura de colaboración y automatización. Se basa en la integración continua, la entrega continua y la implementación continua para lograr un ciclo de vida de desarrollo de software más ágil, eficiente y confiable.

Mediante la adopción de DevOps, las organizaciones pueden lograr un despliegue más rápido y seguro de software, minimizar los riesgos asociados con los cambios y mejorar la calidad del software. Además, esto permite una mejor alineación entre los equipos de desarrollo y operaciones, lo que resulta en una mayor eficiencia y un mejor aprovechamiento de los recursos.

## 1.2. DevOps: Transformando el proceso de desarrollo de software

Entre los numerosos términos nuevos que surgen asociados a la transformación digital y al entorno IT en los últimos tiempos tenemos que añadir el concepto DevOps.

Se trata de una forma de desarrollar software que favorece la colaboración entre el equipo de desarrolladores y el departamento de operaciones encargado de mantenerlo durante todo su ciclo de vida. En las siguientes líneas se profundizará sobre su verdadero significado, ventajas y aplicaciones.

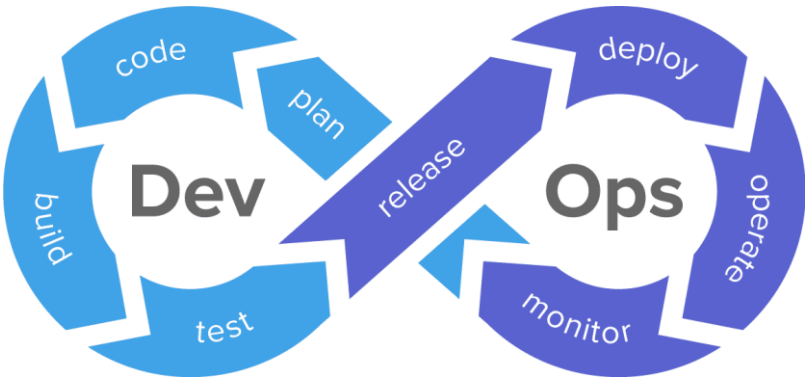


**Imagen 4: Fases en DevOps**

### 1.2.1. ¿Qué es DevOps? Definiciones

- Es un movimiento profesional emergente que promueve la colaboración, el monitoreo y la automatización en el proceso de desarrollo de software, desde las primeras etapas de desarrollo, integración, pruebas y liberación de nuevas versiones hasta la implementación y administración de la infraestructura de sistemas.
- Es un modo de desarrollar software. Aúna los esfuerzos de los desarrolladores (Dev) y de los encargados de mantenerlo en producción (Ops).

Ahora, si se piensa en la verdadera misión de este movimiento dentro de una empresa, se pueden señalar los siguientes puntos:

- Promueve la creación de una cultura de responsabilidad compartida, transparente y ofrece una respuesta más rápida en el resultado de los procesos y operaciones.

- Hace que los procesos se vuelvan más simples, más programables y dinámicos, minimizando los trabajos manuales.

### 1.2.2. Características principales

No se trata de tener un perfil laboral especializado que desarrolle este trabajo dentro de la organización, sino más bien es una especie de "*cultura*", es decir, una forma de trabajar y de aplicar una serie de principios, herramientas y prácticas ágiles, que pretende romper las barreras entre los diferentes departamentos de una misma organización, creando un entorno de colaboración entre ellos.

- Este nuevo concepto, va muy de la mano del Cloud. Gracias a la gestión de infraestructuras como servicio, los equipos de desarrollo pueden obtener, de forma muy ágil, todos los entornos necesarios para poder realizar su trabajo. Además, se adaptan de forma muy eficiente y rápida a cualquier cambio que pueda ser necesario, por la aparición de nuevos requisitos o necesidades en los proyectos

- Es un complemento al proceso de desarrollo de software ágil. Lo extiende y lo complementa a través de la automatización de procesos de desarrollo, pruebas y el despliegue de aplicaciones, asegurando así, una entrega continua sin errores y aportando valor al cliente en un flujo constante.

### 1.2.3. ¿Qué ventajas tiene?

1. Fomenta la colaboración, la comunicación y la confianza dentro de los diferentes equipos de Desarrollo y Operación IT

2. Ahorra tiempo y hace un uso eficiente de los recursos, dado que DevOps automatiza los procesos de pruebas y despliegues de aplicaciones. Es decir, define una estrategia de pruebas y despliegues automatizados que garantizan una entrega continua y libre de errores a los clientes.

3. Permite tener publicaciones más rápidas de nuevas versiones de producto con una mayor calidad y estabilidad y todo ello, de forma automatizada. Por lo tanto, ayuda a llegar antes al mercado, acortando el tiempo de llegada a producción y mejorando la experiencia de los clientes sobre tu trabajo y su producto.

4. Monitoriza de forma continua la "salud" de aplicaciones e infraestructuras, facilitando la toma de decisiones.

5. Permite eliminar todo lo que no es útil para de esta manera, invertir más tiempo en innovar dentro de las organizaciones.

## 1.3. Evolución a DevSecOps

DevOps (*Development and Operations*) es una metodología que busca integrar el desarrollo de software con la operación de infraestructura, con el objetivo de mejorar la colaboración y la eficiencia en el ciclo de vida del software.

DevOps es un enfoque relativamente nuevo que pretende mejorar la colaboración y comunicación entre los equipos de desarrollo de software y operaciones, con el fin de acelerar la entrega de software de alta calidad. Los orígenes de DevOps se pueden rastrear hasta principios de la década del 2000, cuando los desarrolladores comenzaron a darse cuenta de que las prácticas tradicionales de desarrollo de software no estaban funcionando.

Los antecedentes de DevOps se encuentran en la cultura Agile y Lean, que se enfocan en la entrega de valor al cliente de manera rápida y continua, y en la mejora continua del proceso de desarrollo. En particular, el movimiento Agile impulsó la creación de equipos multidisciplinarios y autoorganizados que trabajan de manera colaborativa y que están en contacto constante con el cliente.

Adicionalmente, y a medida que la seguridad se ha convertido en una preocupación cada vez más importante en el desarrollo de software, ha surgido una nueva variante de DevOps: DevSecOps.

DevSecOps se enfoca en integrar la seguridad en todo el ciclo de vida del software, desde el diseño hasta la implementación y el mantenimiento. La idea es que la seguridad no sea vista como una responsabilidad aislada del equipo de seguridad, sino que se integre en el proceso de desarrollo y operaciones en todas las etapas.

### 1.3.1. De DevOps a DevSecOps

La evolución de DevOps a DevSecOps ha sido impulsada por varios factores, entre ellos:

- Aumento de las amenazas de seguridad: con la creciente sofisticación de los ataques informáticos, la seguridad se ha convertido en una preocupación cada vez más importante para las empresas.
- Mayores exigencias normativas: muchas empresas están sujetas a regulaciones cada vez más estrictas en cuanto a la protección de datos y la privacidad.
- Mayor adopción de la nube: con la creciente adopción de la nube, la seguridad se ha convertido en un aspecto crítico en la protección de datos y en la gestión de riesgos.

Para implementar DevSecOps, es necesario que todos los equipos involucrados, incluyendo los equipos de seguridad, adopten prácticas y herramientas que permitan la integración de la seguridad en el proceso de desarrollo y operaciones.

DevSecOps se enfoca en integrar la seguridad en todo el ciclo de vida del software

Algunas de estas prácticas y herramientas incluyen:

- Evaluación continua de riesgos y vulnerabilidades.
- Pruebas de seguridad automatizadas.
- Monitoreo continuo de la seguridad en la producción.
- Implementación de controles de seguridad en el proceso de implementación.
- Capacitación en seguridad para todo el equipo.

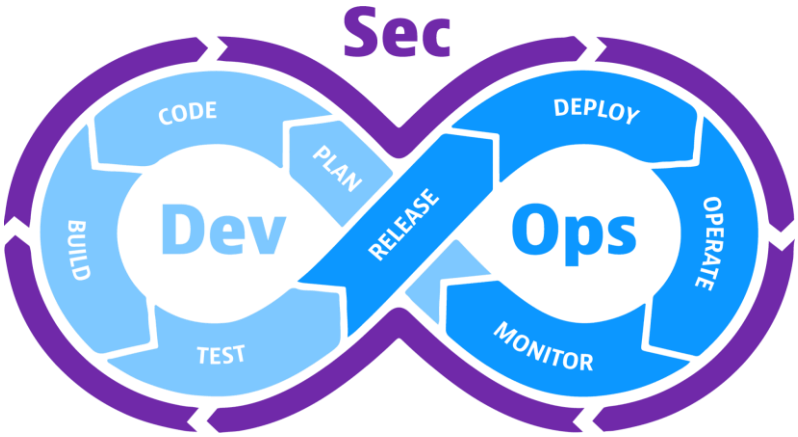


**Imagen 5: Fases en DevSecOps**

Dentro del ámbito de la infraestructura y los sistemas de la información la implementación de DevOps puede ser un desafío debido a las particularidades del entorno.

## 1.4. Herramientas DevOps

Algunas herramientas que pueden ayudar a llevar a buen puerto una estrategia de DevOps (También aplican para DevSecOps) son:

- Jenkins: Es una herramienta de automatización de integración continua que permite la integración de código fuente, la compilación, las pruebas automatizadas y la entrega continua de software.
- GitLab: Es una plataforma integral de DevOps que incluye herramientas de control de versiones de código fuente, integración y entrega continua, pruebas y monitoreo de aplicaciones.
- Docker: Es una herramienta de virtualización de contenedores que permite la creación, distribución y ejecución de aplicaciones en un entorno aislado y portátil.
- Ansible: Es una herramienta de automatización de configuración y orquestación que permite la gestión de la configuración de infraestructura y aplicaciones en múltiples plataformas.
- Kubernetes: Es una plataforma de orquestación de contenedores que permite la gestión y escalado de aplicaciones en contenedores.
- Puppet: Es una herramienta de automatización de configuración que permite la gestión de la configuración de infraestructura y aplicaciones en múltiples plataformas.
- Terraform: Es una herramienta de automatización de infraestructura que permite la definición de infraestructura como código y la gestión de recursos en múltiples proveedores de nube.
- Grafana: Es una plataforma de monitoreo y análisis de métricas y logs que permite la visualización y análisis de datos de aplicaciones y sistemas.

Con un claro enfoque hacia la seguridad y por tanto a transformar la cultura corporativa hacia un enfoque DevSecOps, se puede considerar la evaluación

de las siguientes herramientas adicionales y complementarias a las anteriores:

- SonarQube: Es una herramienta de análisis de código fuente que permite identificar y corregir vulnerabilidades y debilidades en el código. Proporciona informes detallados sobre la calidad del código, la seguridad y el cumplimiento de normas.

- OWASP ZAP: es una herramienta de prueba de penetración automatizada para aplicaciones web. Permite encontrar vulnerabilidades y debilidades en las aplicaciones web mediante la simulación de ataques.

- Qualys: es una plataforma de seguridad en la nube que ofrece una amplia gama de herramientas de seguridad, incluyendo análisis de vulnerabilidades, evaluación de riesgos, cumplimiento normativo y monitorización continua.

- Aqua Security: es una plataforma de seguridad de contenedores que incluye herramientas para la evaluación de vulnerabilidades, la gestión de políticas y la monitorización de eventos de seguridad.

- Twistlock: es una plataforma de seguridad de contenedores que incluye herramientas para la evaluación de vulnerabilidades, la gestión de políticas y la monitorización de eventos de seguridad.

Estas son solo algunas de las herramientas DevOps más populares en el mercado, y existen muchas otras herramientas disponibles que pueden ser adecuadas para las necesidades y objetivos específicos de cada organización.

## 1.5. ¿Qué es la automatización de la seguridad?

En primer lugar, se debe definir qué se entiende por automatización. La automatización, en este caso, consiste en usar la tecnología para realizar tareas con la menor intervención humana posible y, por ende, reducir los

errores que pueden derivarse de ella. Permite optimizar los procesos, ampliar los entornos y diseñar flujos de trabajo de integración, distribución e implementación continuas (CI/CD). Hay muchos tipos de automatización, como la automatización de la TI, la automatización de la infraestructura, la automatización empresarial, la automatización robótica de los procesos, la automatización industrial, la inteligencia artificial, el aprendizaje automático y el aprendizaje profundo.

La automatización de la seguridad, por ende, consiste en usar las tecnologías que ejecutan tareas con la menor intervención humana para integrar los procesos, las aplicaciones y la infraestructura de seguridad.

La seguridad de la TI protege la integridad de las tecnologías de la información (como los sistemas informáticos, las redes y los datos) de los ataques, los daños o el acceso no autorizado. Es un concepto general que abarca la seguridad de las redes, del Internet, de los extremos, de las interfaces de programación de aplicaciones (API), de la nube, de las aplicaciones, de los contenedores y mucho más. Se trata de establecer un conjunto de estrategias de seguridad que funcionan juntas para proteger los datos.

### 1.5.1. Rol de la automatización de la seguridad en DevSecOps

La automatización de la seguridad y DevSecOps representan una evolución significativa en las prácticas de desarrollo de software, al integrar de manera proactiva la seguridad en todo el ciclo de vida del desarrollo.

Como se dijo anteriormente, la automatización de la seguridad consiste en la implementación de tecnologías y procesos para automatizar tareas de seguridad, como la detección de vulnerabilidades, la respuesta a incidentes y la aplicación de parches. Esta automatización reduce la latencia en la detección y respuesta a amenazas, minimizando el riesgo de explotación.

Mientras que DevSecOps es una metodología que fusiona las disciplinas de desarrollo, seguridad y operaciones, promoviendo una cultura de colaboración y responsabilidad compartida en materia de seguridad. DevSecOps busca integrar la seguridad desde las primeras etapas del desarrollo, asegurando que el software sea seguro desde su concepción.

Cuando relacionamos ambos conceptos obtenemos:

- Integración temprana de la seguridad: La automatización permite incorporar pruebas de seguridad de manera temprana en el ciclo de desarrollo, facilitando la identificación y mitigación de vulnerabilidades.
- Aceleración de la salida a producción: Al automatizar tareas repetitivas, se reducen los tiempos de desarrollo y despliegue, sin comprometer la seguridad.
- Consistencia y escalabilidad: La automatización garantiza la aplicación uniforme de políticas de seguridad en todos los entornos, facilitando la gestión de infraestructuras complejas.
- Resiliencia: La automatización de la respuesta a incidentes permite una recuperación más rápida ante ataques cibernéticos.

### 1.5.2. Beneficios de automatizar los procesos de seguridad

A medida que aumenta el tamaño y la complejidad de la infraestructura y las redes, es cada vez más difícil gestionar la seguridad y el cumplimiento de forma manual. Las operaciones manuales pueden ralentizar el proceso de detección y resolución de los problemas, dar lugar a errores en la configuración de los recursos e impedir que haya uniformidad en la aplicación de las políticas, lo cual expone a los sistemas a problemas de cumplimiento y ataques. Esto puede generar tiempo de inactividad imprevisto y la reducción general del funcionamiento, lo cual puede ser muy costoso para las organizaciones. La automatización puede ayudar a optimizar las operaciones cotidianas y a integrar la seguridad con la infraestructura de TI, los procesos, las arquitecturas de nube híbrida y las aplicaciones desde el comienzo.

Ventajas:

- Mejora de la postura de seguridad: Al integrar la seguridad en todo el ciclo de vida del desarrollo, se reduce significativamente la superficie de ataque.
- Aumento de la eficiencia: La automatización libera a los equipos de seguridad para que se enfoquen en tareas estratégicas.
- Mayor agilidad: La integración continua de la seguridad permite adaptarse rápidamente a las amenazas emergentes.

Algunas de las tecnologías y herramientas que nos pueden ayudar a aplicar la automatización de la seguridad en todo el proceso de DevSecOps son:

- Plataformas SOAR: Orquestan y automatizan los flujos de trabajo de seguridad.
- Herramientas de análisis de código estático y dinámico: Identifican vulnerabilidades en el código.
- Contenedores y orquestación: Facilitan el despliegue seguro de aplicaciones en entornos dinámicos.
- CI/CD: Integra la seguridad en los pipelines de integración y entrega continua.

De esta forma se afirma que la automatización de la seguridad y DevSecOps son elementos complementarios que permiten a las organizaciones construir sistemas más seguros y resilientes. Al adoptar estas prácticas, las empresas pueden reducir el riesgo de sufrir incidentes de seguridad, proteger sus activos digitales y cumplir con los requisitos regulatorios.

# 2. Herramientas de automatización

Como se vio en el capítulo anterior, la gestión de infraestructuras informáticas ha evolucionado significativamente con la adopción de prácticas que priorizan la automatización y la definición declarativa de los entornos. En este contexto, las herramientas de automatización y el paradigma de Infraestructura como Código (IaC) desempeñan un papel fundamental.

Pero primero se debe definir a qué se llama IaC. La infraestructura como Código (IaC) es una metodología que trata la infraestructura de TI como un recurso de software, definido a través de código en lugar de configuraciones manuales. Esto permite versionar, automatizar y escalar la infraestructura de manera más eficiente.

Por su lado, las herramientas de automatización son el motor que impulsa la implementación de IaC. Permiten definir el estado deseado de la infraestructura en archivos de configuración (manifiestos o *playbooks*) y luego aplicar esos cambios de manera repetible y confiable en múltiples sistemas.

Utilizando las herramientas de automatización bajo el concepto de IaC se va a obtener:

- Definición declarativa: Las herramientas de automatización permiten expresar el estado deseado de la infraestructura utilizando un lenguaje declarativo, es decir, describiendo qué se quiere lograr en lugar de como hacerlo.
- Automatización de aprovisionamiento: Facilitan la creación y configuración automática de recursos de infraestructura, como servidores virtuales, redes y almacenamiento.
- Gestión de configuración: Permiten mantener la configuración de los sistemas de manera consistente y actualizada.
- Orquestación: Coordinan la ejecución de múltiples tareas en diferentes sistemas de forma secuencial o paralela.

Principales beneficios:

- Mayor eficiencia: Reducción del tiempo de implementación y menor intervención manual.

- Consistencia: Aseguran que todos los entornos estén configurados de forma idéntica.
- Repetibilidad: Facilitan la recreación de entornos y la realización de pruebas.
- Escalabilidad: Permiten gestionar infraestructuras de gran tamaño de manera ágil.
- Colaboración: Facilitan el trabajo en equipo al proporcionar una única fuente de verdad para la infraestructura.

La combinación de IaC y herramientas de automatización es una práctica esencial para cualquier organización que busque agilizar sus procesos de desarrollo, mejorar la fiabilidad de sus sistemas y reducir los costos operativos. Al adoptar este enfoque, las empresas pueden responder de manera más rápida y efectiva a las demandas del negocio.

Ahora se hará un pequeño detalle de las principales herramientas de automatización actuales.

## 2.1. Puppet

Puppet es una herramienta de automatización de la configuración y administración de sistemas informáticos. Funciona mediante la definición y aplicación de reglas de configuración predefinidas, conocidas como "recursos", en los sistemas objetivo. Estas reglas se definen en archivos llamados "manifiestos" que describen cómo debe estar configurado un sistema para cumplir con los requisitos especificados.

Una de las principales ventajas de Puppet es la capacidad de automatizar tareas repetitivas y complejas en entornos de infraestructura de TI. Esto permite a los administradores de sistemas gestionar de manera eficiente y consistente grandes cantidades de servidores y dispositivos.

Además, Puppet ofrece una interfaz visual y una API que permiten supervisar y gestionar de forma centralizada la configuración de los sistemas, así como la capacidad de realizar cambios en tiempo real en función de eventos específicos. También cuenta con una extensa biblioteca de recursos predefinidos y una activa comunidad de usuarios que comparten y colaboran en la creación de nuevos recursos y módulos.

Sin embargo, Puppet también tiene algunas limitaciones, como su curva de aprendizaje inicial, que puede resultar pronunciada para usuarios sin experiencia en la herramienta. Además, la implementación y el mantenimiento de Puppet pueden requerir una inversión de tiempo y recursos significativa.

## 2.2. Chef

Es una plataforma de automatización de infraestructuras que permite a los equipos de operaciones de TI administrar y configurar su infraestructura de forma rápida y consistente. Utilizando una arquitectura de código abierto, Chef permite a los usuarios automatizar la creación, implementación y administración de servidores y aplicaciones en cualquier entorno, ya sea en la nube, en instalaciones locales o en entornos híbridos.

Chef funciona utilizando recetas y cookbooks, que son scripts escritos en un lenguaje de dominio específico llamado DSL (Domain Specific Language). Estas recetas y cookbooks se utilizan para definir la configuración deseada del sistema, que luego se implementa en los servidores a través de un proceso de convergencia, donde se compara el estado actual de la infraestructura con la configuración definida y realiza los cambios necesarios para que coincidan.

Uno de los principales beneficios de Chef es la capacidad de escalar de manera eficiente y administrar infraestructuras complejas con miles de servidores. También ayuda a garantizar la consistencia y la conformidad en toda la infraestructura, lo que mejora la seguridad y reduce el riesgo de errores humanos.

Sin embargo, Chef también tiene algunas desventajas. La curva de aprendizaje puede resultar empinada para aquellos que no están familiarizados con la programación o la automatización de infraestructuras. Además, la arquitectura distribuida de Chef puede resultar difícil de mantener y depurar en entornos complejos.

## 2.3. SaltStack

SaltStack es una herramienta de automatización de infraestructura y configuración de sistemas que permite gestionar y mantener de manera eficiente un gran número de servidores de forma remota.

Su funcionamiento se basa en un sistema de comunicación entre un servidor central llamado "Master" y múltiples servidores remotos llamados "Minions". El Master envía comandos a los Minions para realizar tareas específicas como instalación de software, actualizaciones de configuración, monitoreo y mantenimiento de sistemas, entre otros.

Algunos de los pros de SaltStack incluyen su rapidez y escalabilidad, ya que permite gestionar fácilmente miles de servidores de forma eficiente. Además, su arquitectura basada en Python facilita la creación y personalización de módulos y fórmulas para adaptarse a las necesidades específicas de cada infraestructura.

Por otro lado, algunos de los contras de SaltStack incluyen su curva de aprendizaje pronunciada, ya que su configuración y uso pueden ser algo complejos para usuarios novatos. Además, a pesar de contar con una documentación extensa, a veces puede resultar difícil encontrar soluciones a problemas específicos.

## 2.4. Terraform

Terraform es una herramienta de código abierto desarrollada por HashiCorp que permite la automatización de la infraestructura mediante la creación y gestión de recursos en la nube de forma programática.

El funcionamiento de Terraform se basa en la definición de la infraestructura como código, lo que significa que se describe la configuración deseada de los recursos en un lenguaje específico (HCL) y Terraform se encarga de aprovisionarlos y gestionar su estado. Esto permite una automatización completa del ciclo de vida de los recursos, desde su creación hasta su actualización o eliminación.

Entre los pros de Terraform como herramienta de automatización se encuentran la simplificación de la gestión de la infraestructura, la capacidad de definir y mantener la configuración de forma sencilla, el soporte multi-cloud

que permite trabajar con múltiples proveedores de nube y la integración con otras herramientas de DevOps.
Sin embargo, también existen algunos contras a tener en cuenta al utilizar Terraform. Entre ellos se encuentran la curva de aprendizaje para dominar el lenguaje de definición de infraestructura, la posibilidad de errores al aplicar cambios en la configuración y la dependencia de terceros proveedores como HashiCorp.

## 2.5. Pulumi

Pulumi es una herramienta de código abierto que se utiliza para la automatización de la infraestructura en la nube. Permite a los equipos de desarrollo y operaciones definir, implementar y gestionar la infraestructura como código, lo que facilita la creación y actualización de recursos en la nube de manera eficiente y escalable.
Pulumi funciona mediante la definición de la infraestructura a través de código en lenguajes de programación populares como TypeScript, Python, Go y C#. Esta característica permite a los equipos de desarrollo utilizar sus habilidades existentes en lugar de aprender una nueva sintaxis específica de una herramienta de automatización.
Entre los principales beneficios de utilizar Pulumi se encuentran la capacidad de gestionar múltiples proveedores de nube (como AWS, Azure, Google Cloud, etc.) desde un solo lugar, la posibilidad de definir la infraestructura de forma declarativa y la integración con herramientas de control de versiones como Git.
Sin embargo, como cualquier herramienta, Pulumi también tiene sus desventajas. Algunas de ellas incluyen la curva de aprendizaje que puede resultar empinada para aquellos que no están familiarizados con la automatización de la infraestructura, la necesidad de mantener el código de infraestructura actualizado y la posibilidad de errores que puedan surgir al definir la infraestructura a través de código.

## 2.6. Comparación entre las principales herramientas

Como hemos visto, existe una gran variedad de herramientas de automatización que nos pueden ayudar para mejorar la seguridad. Cada una de ellas tiene sus fortalezas y debilidades. La elección ideal dependerá de las necesidades específicas, del equipo de trabajo y de la complejidad de la infraestructura.

En la siguiente tabla se mencionan las principales herramientas de automatización con algunas de sus características más importantes para poder compararlos más claramente. Si bien todavía no se habló de Ansible, se definió agregar sus características en la tabla y detallar en el próximo capítulo toda su funcionalidad.

| Característica | Ansible | Puppet | Chef | SaltStack | Terraform | Pulumi |
|---|---|---|---|---|---|---|
| **Enfoque** | Sin agente | Sin agente | Con agente | Sin agente | Multinube[1] | Multinube[2] |
| **Lenguaje** | YAML | DSL propio | DSL propio | Python | HCL | Python, JS, Go |
| **Comunidad** | Grande y activa | Grande y madura | Grande y madura | Grande y activa | Enorme y creciente | Creciente rápidamente |
| **Curva de aprendizaje** | Fácil | Moderada | Moderada | Moderada | Fácil | Fácil |
| **Uso típico** | Config. de servidores, despliegue de aplicaciones | Config. de estado a gran escala, cumplimiento | Config. de estado a gran escala, cloud computing | Gestión de config. distribuida, orquestación de servicios | Provisión de infraestructura en la nube | Provisión de infraestructura en la nube |

[1] Opera a un nivel más alto, gestionando la creación y configuración de recursos en la nube. No requiere agentes porque interactúa directamente con las APIs de los proveedores de nube.
[2] Idem anterior.

### 2.6.1. Otras opciones a considerar:

- Rundeck: Ideal para ejecutar comandos en múltiples servidores de forma paralela, con una interfaz web intuitiva.
- Jenkins: Principalmente conocido por la integración continua, pero también puede utilizarse para automatización de tareas.
- GitLab CI/CD: Plataforma completa de DevOps que incluye herramientas de CI/CD y automatización.
- Ansible Tower: Interfaz gráfica y funcionalidades adicionales para Ansible, ideal para equipos que necesitan una gestión más visual.
- CloudFormation (AWS), Azure Resource Manager: Herramientas específicas para gestionar la infraestructura en AWS y Azure, respectivamente.

### 2.6.2. Factores influyentes al elegir

- Tamaño y complejidad de la infraestructura: Para entornos pequeños Rundeck puede ser suficiente. Para grandes infraestructuras, Ansible, Puppet, Chef o Terraform son más adecuados.
- Experiencia del equipo: Si el equipo está familiarizado con un lenguaje de programación específico, Pulumi puede ser una buena opción.
- Necesidades de integración: Asegurarse de que la herramienta se integre bien con las otras herramientas que administran (por ejemplo, sistemas de control de versiones, contenedores).
- Presupuesto: Algunas herramientas tienen versiones comerciales con características adicionales.
- Seguridad: Evaluar las medidas de seguridad de cada herramienta y cómo se ajustan a las políticas de seguridad.

### 2.6.3. Ejemplos de uso

- Ansible: Configurar un servidor web Apache en múltiples máquinas, desplegar una aplicación en un entorno de producción.
- Puppet: Asegurar el cumplimiento de políticas de seguridad en toda la infraestructura, gestionar la configuración de cientos de servidores.

- Chef: Automatizar el despliegue de aplicaciones en múltiples entornos, gestionar la configuración de nodos en una nube privada.
- SaltStack: Gestionar la configuración de miles de servidores en tiempo real, orquestar servicios de microservicios.
- Terraform: Proveer infraestructura en múltiples nubes (AWS, Azure, GCP), gestionar la configuración de redes virtuales y subredes.
- Pulumi: Desarrollar infraestructura como código utilizando lenguajes de programación conocidos, gestionar la configuración de contenedores en Kubernetes.

La elección de la herramienta de automatización adecuada es una decisión importante que puede tener un impacto significativo en la eficiencia y la fiabilidad de la infraestructura. Al evaluar las diferentes opciones, se deben considerar las necesidades específicas de cada caso, la experiencia del equipo y los factores mencionados anteriormente.

# 3. Ansible

Ansible es un motor *open source* que automatiza una gran cantidad de procesos informáticos, como la preparación de la infraestructura, la gestión de la configuración, la implementación de las aplicaciones y la organización de los sistemas.

Con la automatización que ofrece, se pueden instalar los sistemas de software, automatizar las tareas diarias, preparar la infraestructura, mejorar la seguridad y el cumplimiento normativo, ejecutar parches en los sistemas y trasladar la automatización a toda la empresa.

Entre las características principales de Ansible se puede mencionar que es una herramienta de automatización sin agentes, usa un lenguaje muy amigable que las personas pueden aprender rápidamente, utiliza el protocolo SSH para organizar la gestión de la configuración, la implementación de las aplicaciones y la preparación en un entorno con uno o varios niveles. Es una de las tecnologías de automatización de TI *open source* más conocidas en todo el mundo.

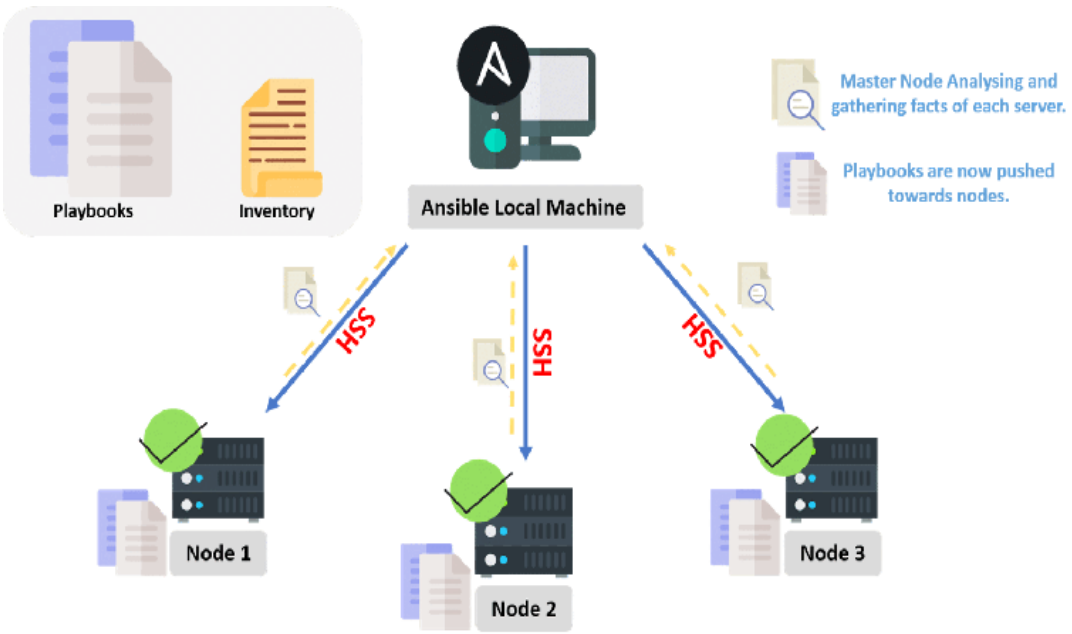


**Imagen 6: Funcionamiento e interconexión en Ansible**

## 3.1. Conceptos básicos de Ansible

Antes de continuar, hay que definir la terminología para entender mejor los conceptos:

Nodo controlador (Ansible local machine): el host en el cual se instala Ansible para ejecutar tareas en los nodos gestionados.

Nodos gestionados: los hosts que configura el nodo controlador.

Inventario de hosts: es la lista de los nodos gestionados.

Comando específico: tarea sencilla y puntual.

Idempotencia: las operaciones idempotentes producen los mismos resultados si se ejecutan una o varias veces sin ninguna intervención.

Esta herramienta es independiente de la arquitectura. Esto significa que Ansible puede instalarse en ambientes físicos, virtualizados, nubes públicas y privadas. Puede integrarse con muchísimas otras aplicaciones que amplían significativamente las posibilidades de ejecución y parametrización.

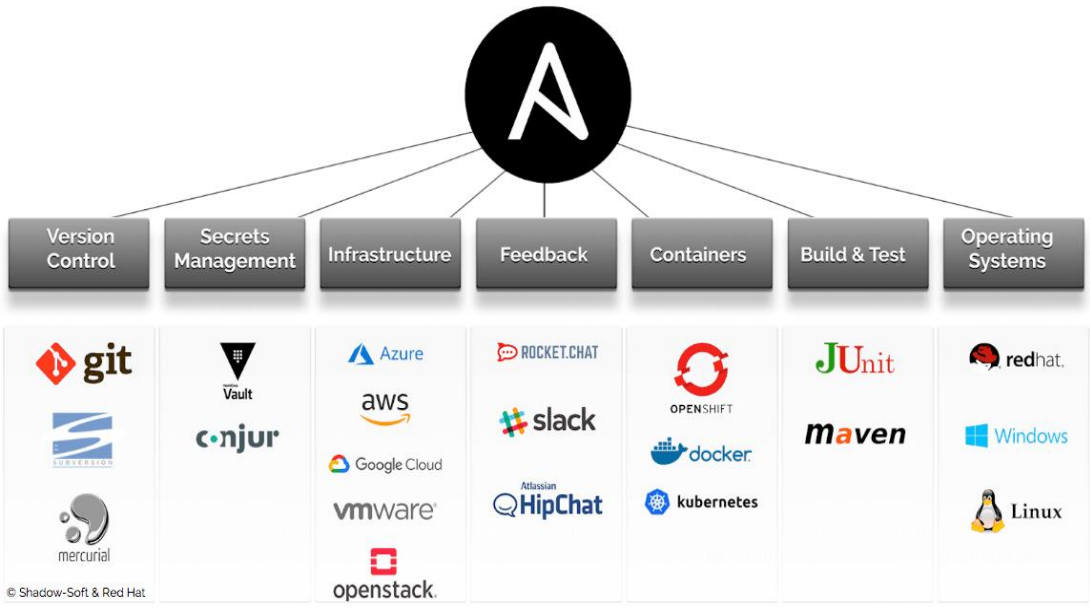


**Imagen 7: Integración de Ansible con distintas herramientas**

## 3.2. Funcionamiento de Ansible

Se mencionó anteriormente que con Ansible no se necesitan agentes, esto quiere decir que no se requiere instalar ningún software en los nodos que se gestionan. La plataforma lee la información del inventario para saber qué máquinas se van a gestionar. Aunque cuenta con un archivo de inventario predeterminado, se puede crear uno propio y definir los servidores que se quieran administrar.

La plataforma utiliza el protocolo SSH para conectarse a los servidores y ejecutar las tareas. De forma predeterminada, Ansible utiliza claves y un agente SSH y se conecta a las máquinas remotas con el nombre de usuario que ejecutó la tarea. No es necesario que se inicie sesión como súper usuario; se puede iniciar como cualquier otro y luego utilizar los comandos su o sudo para adquirir privilegios superiores.

Una vez que Ansible se conecta, transfiere los módulos que requiere el comando o el *playbook* a la máquina remota para que los ejecute. La plataforma utiliza las plantillas de YAML, las cuales son fácilmente comprensibles para las personas, de manera que los usuarios puedan programar la ejecución automática de las tareas repetitivas sin necesidad de aprender un lenguaje de programación avanzado.

### 3.2.1. Instalación, configuración y primeros pasos

Como requisitos previos hay que tener instalado Python y se debe tener acceso por SSH a los nodos que van a ser gestionados.
La forma más común de instalar Ansible es usando el paquete *pip* de Python. Para esto primero hay que asegurarse de que esté instalado, con el comando:

*$ python3 -m pip -V*

Si no lo está, se realiza la instalación del paquete *python3-pip* para luego terminar instalando Ansible con el comando:

*$ python3 -m pip install --user ansible*

Para la configuración básica tenemos 2 archivos:

- Archivo de configuración: Ansible utiliza un archivo de configuración (por defecto, /etc/ansible/ansible.cfg) para almacenar opciones globales. Se puede personalizar este archivo según las necesidades de cada uno. También se puede hacer una copia específica para un directorio en particular.
- Archivo de inventario: Un archivo de inventario define los hosts que Ansible administrará.

Lo primero que se va a probar luego de instalar y hacer una configuración mínima de Ansible son los llamados comandos ad-hoc. Estos comandos se ejecutan de forma inmediata y no requieren un *playbook*. Por ejemplo, para ejecutar un comando en todos los hosts del grupo ServidoresWeb del archivo de inventario personalizado, vamos a ejecutar:

```
ansible ServideoresWeb -m ping
```

Con el parámetro -m se indica el comando que se va a ejecutar.

### 3.2.2. Creación de *Playbooks*

Los *playbooks* son el corazón de Ansible. Son archivos de texto, generalmente escritos en YAML, que describen la configuración deseada de uno o más hosts. En otras palabras, son recetas para automatizar tareas en múltiples sistemas.

Un *playbook* se compone de una o más *plays*. Cada *play* define un grupo de hosts y un conjunto de tareas a ejecutar en esos hosts.

Estructura básica de un *playbook* en formato YAML:

```
- name: Mi primer playbook
  hosts: all
  tasks:
    - name: Actualizar el sistema
      apt:
        update_cache: yes
        upgrade: dist-upgrade
```

Definición de los parámetros:

*name*: Un nombre descriptivo para el *play*.

*hosts*: Una lista de hosts o grupos de hosts definidos en el inventario.
*tasks*: Una lista de tareas a ejecutar. Cada tarea utiliza un módulo de Ansible. En este caso se utilizó solo el módulo *apt*.
Si guardamos ese contenido en el archivo de texto mi-playbook.yml, luego podemos ejecutarlo con el comando:

*$ ansible-playbook -i inventory.ini mi-playbook.yml*

**Módulos de Ansible**

Ansible se conecta a los nodos y les inserta pequeños programas denominados módulos. Los módulos son los bloques constructores de Ansible. Cada uno realiza una acción específica, como instalar paquetes, copiar archivos, iniciar servicios, etc.

Puede verse una lista de todos los módulos instalados en el sistema con el comando:

*$ ansible-doc -l*

Ejemplo de uso de un módulo en formato YAML:

```
- name: Copiar un archivo
  copy:
    src: /local/file.txt
    dest: /remote/file.txt
```

### 3.2.3. Roles

Un rol en Ansible es una colección de tareas, archivos y variables que se organizan de manera lógica para realizar una función específica en un sistema o aplicación.

Un rol puede ser visto como un paquete que encapsula todo lo necesario para realizar una tarea específica como, por ejemplo, instalar y configurar un servidor web, configurar un cortafuegos, configurar un servicio de base de datos, etc.

Un rol se compone de los siguientes elementos:

- Estructura de directorios estandarizada para organizar los archivos y tareas que conforman el rol.
- Archivos YAML para definir sus variables y configuraciones específicas.
- Tareas escritas en lenguaje YAML.
- Plantillas Jinja2 para generar archivos de configuración dinámicos.
- Handlers, que son tareas especiales que se ejecutan en respuesta a eventos específicos.

Los roles en Ansible se pueden descargar y utilizar desde el repositorio oficial de roles en Ansible Galaxy (Que veremos más adelante) o de otros repositorios públicos o privados.

Un ejemplo sencillo de rol puede ser el de instalación y configuración de un servidor web Apache.

El mismo se puede estructurar de la siguiente manera:

```
apache/
|– tasks/
        |-- main.yml
        |-- install.yml
        |-- configure.yml
|– files/
        |-- httpd.conf
|– templates/
        |-- index.html.j2
|– vars/
        |-- main.yml
|– handlers/
        |-- main.yml
```

El rol se puede invocar desde un *playbook* de la siguiente manera:

```
- name: Instalar y configurar Apache
  hosts: webservers
  roles:
    - apache
```

Este sería el contenido típico de un *playbook*, en el cual se puede agregar otros roles para por ejemplo la instalación de postgres, la configuración de SSH y la administración de reglas de firewall.

Ahora, la pregunta que surge es qué ventajas tiene utilizar roles frente a *playbooks*. Los roles son una forma de reutilizar código y organizar mejor la estructura de los *playbooks* en proyectos de Ansible más grandes y complejos. Al utilizar roles, podemos separar las tareas y variables en módulos más pequeños y especializados que pueden ser reutilizados en varios *playbooks*. Esto tiene varias ventajas:

- Reutilización de código: Los roles permiten reutilizar tareas y variables comunes en varios *playbooks*.
- Modularidad: Los roles permiten organizar las tareas en módulos más pequeños y especializados.
- Facilidad de mantenimiento: Los roles permiten separar las tareas y variables en módulos más pequeños y especializados, lo que facilita la actualización y el mantenimiento del código a largo plazo.
- Facilidad de colaboración: Los roles permiten que los miembros del equipo colaboren en proyectos de Ansible más grandes y complejos al separar las tareas en módulos más pequeños y especializados.

### 3.2.4. Gestión de inventario

Un inventario en Ansible es un archivo que contiene una lista de los hosts que se desean administrar. Estos hosts pueden ser agrupados según características comunes (por ejemplo, servidores web, bases de datos) para facilitar la gestión.
Un ejemplo sencillo de un archivo inventario.ini sería:

*[webservers]*
*server1*
*server2*
*server3*
*[databases]*
*db1*

*db2*

Existen dos tipos de inventarios: los estáticos y los dinámicos. Los primeros son los más habituales. Estos son un fichero de texto donde uno detalla a mano los servidores, sus agrupaciones y hasta, si quiere, las variables que los configuran.

En algunas ocasiones se necesita que este inventario no venga de un archivo estático, por ejemplo, con la tecnología en la nube y con los contenedores o máquinas volátiles que pueden desaparecer y aparecer en cualquier momento. En estos casos es más fácil crear un script que lea dinámicamente la configuración.

Se pueden crear estos scripts en cualquier lenguaje (bash, python, javascript, etc.) y debe devolver un archivo en formato JSON donde tengamos las distintas máquinas con la misma información que un inventario estático. Después se pueden utilizar de la misma forma que estos para ejecutar *playbooks* de la siguiente forma:

*$ ansible-playbook -i inventory.sh mi-playbook.yml*

### 3.2.5. Variables

Las variables en Ansible son contenedores que almacenan datos que pueden ser utilizados en los *playbooks*. Estos datos pueden ser nombres de hosts, rutas de archivos, valores de configuración, etc. Al utilizar variables se puede hacer que los *playbooks* sean más flexibles y reutilizables.

Tipos de Variables

- Variables de host: Se definen directamente en el inventario y son específicas para cada *host*.
- Variables de grupo: Se definen para un grupo de *hosts* y son heredadas por todos los miembros del grupo.
- Variables globales: Se definen a nivel global y están disponibles para todos los *hosts*.
- Variables de registro: Se crean durante la ejecución del *playbook* y almacenan información sobre el estado de las tareas.
- Variables especiales: Variables predefinidas por Ansible que proporcionan información sobre el entorno de ejecución.

Alcance y Prioridad de las Variables

El alcance de una variable determina dónde puede ser utilizada. La prioridad de una variable determina qué valor se utiliza cuando hay múltiples variables con el mismo nombre.

La prioridad que van a tener las variables está definida por:

1. Variables de registro: Tienen la mayor prioridad y se sobrescriben si hay conflictos.
2. Variables de tarea: Definidas dentro de una tarea, tienen prioridad sobre las variables de grupo y globales.
3. Variables de grupo: Tienen prioridad sobre las variables globales.
4. Variables globales: Tienen la menor prioridad.

Ejemplo para personalizar la instalación de software:

```
- name: Instalar paquetes
  apt:
    name: "{{ packages }}"
     - state: present
```

La variable *packages* se define en el inventario o en el *playbook* y contiene una lista de paquetes a instalar.

Como buenas prácticas para el uso de variables hay que mencionar:

- Organización: Utilizar archivos de variables separados para mejorar la legibilidad y la mantenibilidad.
- Utilizar variables de registro: Capturar información útil durante la ejecución de *playbooks*.
- Evitar la duplicación: Definir las variables en el nivel más alto posible para evitar la redundancia.
- Documentación: Incluir comentarios en los *playbooks* para explicar el propósito de cada variable.

### 3.2.6. Facts

Los *facts* en Ansible son variables que contienen información sobre el estado del sistema, como la distribución del sistema operativo, la versión del kernel,

la cantidad de memoria RAM, el espacio en disco, entre otros. Esta información se recopila automáticamente antes de ejecutar un *playbook* y puede ser utilizada para personalizar las tareas en función de las características de cada nodo. Estas variables se pueden consultar a través del módulo setup:

*$ ansible <hostname> -m setup*

Los *facts* se utilizan en los *playbooks* de Ansible de la misma manera que cualquier otra variable. Se encierran entre dobles llaves {{ }} y pueden ser utilizados en cualquier parte del *playbook* donde se necesite una variable.

Los *facts* permiten crear *playbooks* altamente flexibles y adaptables. Aquí algunos ejemplos de cómo utilizar los facts para automatizar tareas:

- Instalar paquetes específicos para cada distribución: Utilizando el fact *ansible_distribution*.
- Configurar servicios en función del sistema operativo: Utilizando el fact *ansible_os_family*.
- Aplicar políticas de seguridad basadas en el nivel de criticidad del host: Utilizando etiquetas o variables personalizadas.
- Optimizar la configuración de aplicaciones en función de los recursos del sistema: Utilizando facts como *ansible_memtotal_mb* o *ansible_processor_cores*.

Por ejemplo, podemos crear una tarea para cargar distintos tipos de variables según el sistema operativo en el cual ejecutemos el *playbook*:

Contenido del archivo playbook.yml:

```
- name: “Incluir variables específicas del SO”
  include_vars: "{{ ansible_os_family }}.yml"
```

Contenido de los archivos según la familia:

RedHat.yml

```
apache_packages:
 - httpd
```

Debian.yml

```
apache_packages:
 - apache2
```

## 3.3. Gestión de la configuración con Ansible

La gestión de la configuración es un proceso que busca mantener los sistemas informáticos, los servidores y el software en un estado deseado y uniforme. Permite garantizar que un sistema funcione como se espera a medida que se realizan cambios. Esto era algo que los administradores de sistemas solían hacer manualmente o con scripts personalizados.

Cuando se utiliza Ansible como una herramienta para la gestión de la configuración, su función es almacenar el estado actual de los sistemas y ayudarnos a mantenerlo.

Con las herramientas de gestión de la configuración, se agilizan los cambios y las implementaciones, se elimina la posibilidad de que se cometan errores humanos, y la gestión del sistema se torna predecible y flexible. También permiten realizar un seguimiento del estado de los recursos y evitar repetir ciertas tareas, como instalar el mismo paquete dos veces.

Si se automatiza la gestión de la configuración con Ansible, se puede mejorar la capacidad de recuperación del sistema tras un evento importante. Si un servidor falla por alguna razón desconocida, se puede implementar uno nuevo rápidamente y obtener un registro de las modificaciones o las actualizaciones efectuadas para identificar el origen del problema.

## 3.4. Preparación de los sistemas con Ansible

En la actualidad, la infraestructura suele definirse en el software. La virtualización y los contenedores agilizaron los procesos de preparación y eliminaron la necesidad de preparar y gestionar sistemas de hardware con frecuencia.

Cuando la preparación de las implementaciones se realiza de forma manual, resulta difícil hacer un seguimiento de los cambios y controlar las versiones, así como evitar los errores y las faltas de uniformidad. Al automatizar la preparación de la infraestructura se simplifica mucho el proceso, por lo que

debería ser el primer paso en la automatización del ciclo de vida operativo de nuestras aplicaciones.

Ansible se utiliza para preparar la infraestructura subyacente del entorno, los hosts virtualizados y los hipervisores, los dispositivos de red y los servidores dedicados (*bare metal*). También permite instalar servicios; agregar hosts informáticos, y preparar recursos, servicios y aplicaciones dentro de la nube.

Se puede utilizar un *playbook* de Ansible para describir el estado deseado de la infraestructura, y la plataforma se encargará de prepararlo. Si se utiliza un *playbook* para codificar la infraestructura, se asegura de preparar siempre el mismo entorno.

## 3.5. Automatización de la implementación

Al automatizar la implementación, se puede trasladar el software del entorno de prueba al de producción sin utilizar procesos manuales. Como resultado, se obtiene una implementación confiable y repetible en todo el ciclo de distribución del software. La automatización de la implementación es importante a la hora de habilitar las prácticas de DevOps y gestionar un canal de CI/CD.

Este proceso no funciona si el equipo de desarrollo implementa las aplicaciones o configura los entornos de una manera, y los equipos de operaciones lo hacen de otra. El entorno debe ser uniforme para poder automatizarse. Es por eso que se debe utilizar el mismo proceso de implementación para todos los entornos, incluso para el de producción.

Por lo general, un proceso de implementación sigue tres pasos fundamentales (aunque puede incluir más): el diseño, la prueba y la implementación. Este proceso respalda la capacidad de automatizar el proceso de implementación y garantiza que el código pase rápidamente de la etapa de revisión y confirmación de cambios a la de implementación. Ansible nos permite implementar aplicaciones de varios niveles de manera confiable y uniforme utilizando un marco común. Se pueden usar los *playbooks* para configurar los

servicios necesarios (como se puede observar en el capítulo 4: *Hardening* del servicio SSH) e insertar los grandes paquetes binarios de las aplicaciones.

## 3.6. Ansible Galaxy

Es un sitio en línea donde se comparte contenido de Ansible. Desde allí se puede buscar, descargar, calificar y revisar todo tipo de contenido desarrollado por la comunidad, como si fuera una biblioteca de Ansible. Se pueden crear roles propios o utilizar roles ya existentes en Ansible Galaxy. Esta biblioteca se nutre con los roles y colecciones (conjunto de roles agrupados según algún criterio) que sube la comunidad. Todo este contenido es de acceso público, probado y calificado por los miembros de la misma.

Entre los beneficios de esta herramienta se puede mencionar que ayuda a ahorrar tiempo al reutilizar código existente. Por otro lado, aumenta la calidad de un *playbook* gracias al aporte de la comunidad. Por último, facilita la colaboración y el intercambio de conocimientos.

El link de Ansible Galaxy es el siguiente: https://galaxy.ansible.com/

### 3.6.1. Uso de Ansible Galaxy

El proceso se divide en 3 etapas: búsqueda, selección e instalación.
La búsqueda tiene 2 métodos. Se puede realizar a través de la interfaz web: Galaxy tiene una interfaz web intuitiva con un potente motor de búsqueda. Se puede buscar por nombre de rol, autor, etiquetas, y otros criterios.
El segundo método es a través de la línea de comandos: Se puede usar el comando *ansible-galaxy search* para buscar roles desde la terminal.
Para la etapa de selección del rol que vamos a instalar se deben tener en cuenta los siguientes aspectos:

- Popularidad: Fijarse en la cantidad de estrellas y descargas que tiene un rol.
- Mantenimiento: Verifica si el rol está siendo actualizado regularmente.
- Documentación: Revisar la documentación del rol para entender cómo funciona y cuáles son sus requisitos.

- Licencia: Asegurarse de que la licencia del rol sea compatible con las necesidades de quien va a utilizarla.

La última etapa es la de instalación. Para ello simplemente hay que ejecutar por línea de comando:

*$ ansible-galaxy install nombre_del_rol*

### 3.6.2. Ejemplo práctico

Como ejemplo se puede instalar un rol que realice la instalación y configuración de nginx en Linux.

Para esto debo ingresar por la web y directamente busco el nombre del servicio o simplemente por línea de comandos ejecutar:

*$ ansible-galaxy search nginx*

Para esta fecha (08/2024) la búsqueda devuelve 691 roles.

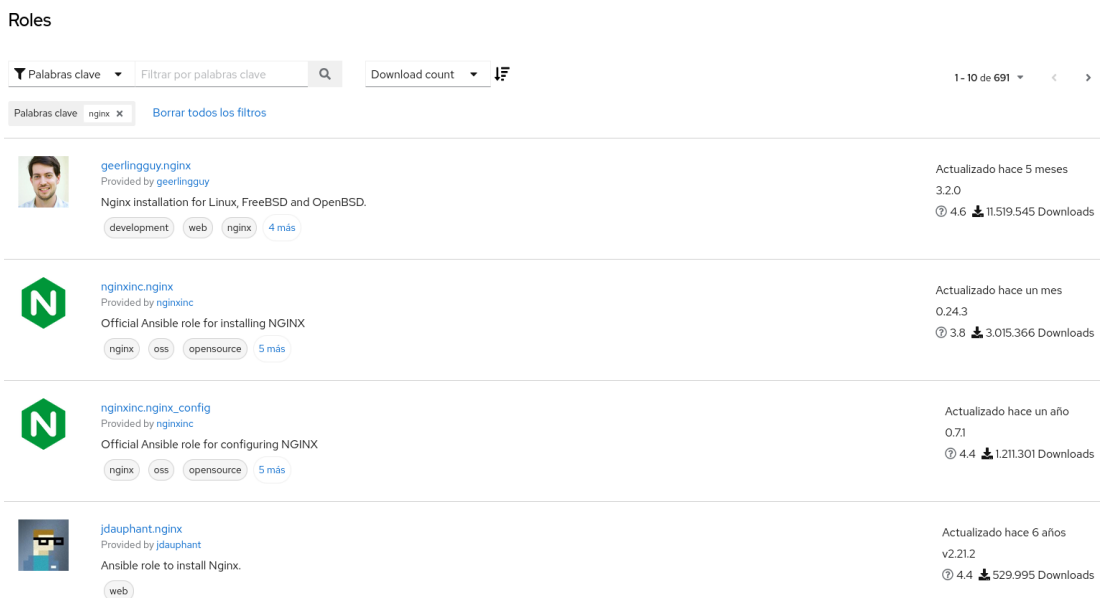


**Imagen 8: Buscador de roles de Ansible Galaxy**

En la imagen ordené los roles por la cantidad de descargas. Luego de estar seguro del que voy a seleccionar, por ejemplo el rol geerlingguy.nginx, se procede a la instalación simplemente ejecutando:

*$ ansible-galaxy install geerlingguy.nginx*

# 4. *Hardening* del servicio SSH

En esta sección se presentará una demostración concisa de las capacidades de Ansible para mejorar la seguridad informática. A través de un ejemplo práctico, se evidenciará cómo esta herramienta permite configurar servicios de manera ágil y escalable, reduciendo significativamente el tiempo y esfuerzo requeridos en la gestión de múltiples equipos. Se ilustrará cómo un *playbook* sencillo puede garantizar la configuración consistente de servicios en una flota de equipos, minimizando el riesgo de errores manuales y fortaleciendo la postura de seguridad de la organización.

SSH (*Secure Shell*) es uno de los protocolos más utilizados para la administración remota de sistemas. Sin embargo, es también un objetivo frecuente para ataques. Ansible, una herramienta de automatización, nos permite configurar de manera consistente y eficiente múltiples servidores, garantizando que SSH esté configurado de manera segura.

El *hardening* de SSH consiste en aplicar una serie de configuraciones para minimizar la superficie de ataque y dificultar la explotación de vulnerabilidades. Algunas de las medidas más comunes incluyen:

- **Deshabilitar el acceso root directo:** Evitar que los usuarios se conecten directamente como *root*.
- **Forzar la autenticación basada en claves:** Eliminar la autenticación por contraseña, que es más vulnerable a ataques de fuerza bruta.
- **Limitar las conexiones desde direcciones IP específicas:** Permitir conexiones solo desde las redes confiables.
- **Configurar timeouts:** Desconectar sesiones inactivas después de un tiempo determinado.
- **Deshabilitar servicios innecesarios:** Eliminar servicios que no son necesarios para el funcionamiento del sistema.
- **Mantener el software actualizado:** Aplicar parches de seguridad de forma regular.

## 4.1. Implementación

Como se vio en el punto 3.2 “Funcionamiento de Ansible”, podríamos ejecutar las tareas directo en un *playbook*, pero crear roles en Ansible es una práctica recomendada para organizar, reutilizar y escalar las automatizaciones. Al utilizar roles se puede mejorar la eficiencia, la mantenibilidad y la colaboración en los proyectos. Entonces la mejor manera de trabajar no es crear las tareas directamente en el *playbook*, sino crear un *playbook* que llame a la ejecución de los roles que van a ejecutar dichas tareas.

### 4.1.1. Crear un *playbook* de Ansible

Crear un archivo llamado UBA_MSI_Prueba.yml con el siguiente contenido:

```
---
- name: “Playbook para la instalacion del servidor UBA_MSI_Prueba”
  hosts: UBA_MSI_Prueba
  become: true
  become_method: sudo
  roles:
  - { role: reyespab.ssh }
...
```

### 4.1.2. Explicación del *playbook*

- hosts: Con esta directiva se define a qué equipos aplica el *playbook*. Puede ser a un host en particular, a un grupo o a todos los definidos en el inventario con la palabra reservada “all”.
- become: Si se necesita escalar privilegios para realizar el *playbook*.
- become_method: Qué método de escalado de privilegios se va a utilizar.
- roles: Para listar todos los roles que vamos a aplicar sobre los hosts definidos. Se van agregando uno por línea.

### 4.1.3. Creación del rol

En primer lugar, se debe crear la estructura del rol. Para esto se crea un directorio con el nombre del rol (por ejemplo, ssh). Dentro de este directorio, se crean las siguientes subcarpetas:

- tasks: Aquí se colocarán los *playbooks* con las tareas específicas.
- vars: Aquí se definirán las variables que se utilizarán en las tareas.
- defaults: Aquí se definirán los valores por defecto para las variables.
- handlers: Aquí se colocarán los *handlers*, que son tareas que se ejecutan sólo cuando se produce un cambio en una tarea anterior.
- templates: Aquí se colocarán las plantillas de archivos que se copiarán a los hosts.

### 4.1.4. Creación del archivo de tareas principal

El *playbook tasks/main.yml* contiene las tareas principales para configurar SSH. Este archivo se genera dentro de la carpeta *tasks* con el siguiente contenido de ejemplo:

```
---
- name: Crear Directorio de Llaves
  file:
    path: "{{ reyespab_ssh_sshd_authorized_keys }}"
    state: directory
    owner: root
    group: root
    mode: 0755

- name: Manejo de llaves ssh.
  authorized_key:
    user: "{{ item.0.remote_user }}"
    key: "{{ lookup('file', item.1) }}"
    state: "{{ item.0.state }}"
  path:      "{{      reyespab_ssh_sshd_authorized_keys      }}/{{
item.0.remote_user}}"
  with_subelements:
    - '{{ reyespab_ssh_users }}'
    - keys

- name: "Modificar permisos de llaves"
```

```
  file:
    path:     "{{     reyespab_ssh_sshd_authorized_keys     }}/{{
item.0.remote_user}}"
    owner: "{{ item.0.remote_user }}"
    group: "{{ item.0.remote_user }}"
    mode: "0600"
  with_subelements:
    - '{{ reyespab_ssh_users }}'
    - keys
 when: "item.0.state != 'absent'"

- name: Copiar configuracion del servicio SSH
  template:
    src: templates/sshd_config.j2
    dest: "{{ reyespab_ssh_sshd_config }}"
    owner: root
    group: root
    mode: "u=rw,g=r,o=r"
  notify:
    - restart ssh
...
```

### 4.1.5. Explicación de las tareas

En este ejemplo se usaron los módulos *file, authorized_key y template.* En la primera utilización del comando file se usa para crear el directorio destino de las llaves y asignarle los permisos necesarios.

La segunda tarea usa el módulo *authorized_key* para pasar las llaves públicas de los usuarios desde nuestro equipo controlador de Ansible hacia los equipos destino. Se tuvo en cuenta poder cargar varias llaves por cada usuario, en el caso de que quiera acceder desde varios equipos.

En la tercera tarea volvemos a usar el comando *file* pero esta vez para cambiar los permisos de las llaves.

Y en la última tarea utilizamos el comando *template* para modificar el contenido del archivo sshd_config con los datos que queremos configurar.

### 4.1.6. Contenido de la carpeta *vars*

Las variables definidas en vars (generalmente en el archivo main.yml dentro de esta carpeta) son variables de rol. Esto significa que su valor se aplica a todos los hosts a los que se aplica el rol, a menos que sean sobrescritas por variables de mayor prioridad.

Propósito: Se utilizan para definir valores por defecto que serán comunes a todos los hosts que utilicen el rol. Por ejemplo, si todos los servidores web en tu infraestructura utilizan el mismo puerto para HTTP, puedes definir esa variable en vars.

Prioridad: Las variables de vars tienen una prioridad intermedia. Pueden ser sobrescritas por variables definidas en el inventario, en group_vars o en la línea de comandos.

Por ejemplo, podríamos tener:

```
reyespab_ssh_port: 22
```

### 4.1.7. Contenido de la carpeta *defaults*

La carpeta *defaults* dentro de un rol de Ansible es un lugar especial para almacenar variables predeterminadas. Estas variables sirven como valores iniciales para diversas configuraciones y pueden ser sobrescritas por otras variables de mayor prioridad, como las definidas en *vars* o en el inventario.

En esta carpeta se almacenan:

- Valores por defecto: Como su nombre lo indica, aquí se colocan los valores iniciales que se espera que tengan las variables en la mayoría de los casos.
- Valores generales: Se utilizan para definir valores que son aplicables a la mayoría de los hosts a los que se aplicará el rol.
- Estructura: Se puede establecer una estructura para las variables, facilitando su organización y comprensión.

Entonces, dentro de la carpeta defaults, se puede crear el archivo main.yml con el siguiente contenido:

```
reyespab_ssh_PermitRootLogin: "yes"
reyespab_ssh_LoginGraceTime: "10"
reyespab_ssh_MaxAuthTries: "3"
```

```
reyespab_ssh_MaxStartups: "2"
reyespab_ssh_AllowUsers: "ansible root" #lista de usarios
separados por un espacio.
reyespab_ssh_DenyUsers: [] #lista de usarios separados por un
espacio.
reyespab_ssh_PasswordAuthentication: "yes"
reyespab_ssh_PubkeyAuthentication: "yes"
reyespab_ssh_ChallengeResponseAuthentication: "yes"
reyespab_ssh_AuthorizedKeysFile: "/etc/ssh/authorized_keys/%u"
reyespab_ssh_UsePAM: "yes"
reyespab_ssh_users:
  - remote_user: mlorenzo
    state: present #Define si la key ssh debe estar o no
(absent|present)
    keys:  # key(s) a agregar al usuario remoto
     - files/mlorenzo_1.pub
     - files/mlorenzo_2.pub
```

### 4.1.8. Creación del archivo de *handlers*

Los *handlers* en Ansible son tareas especiales que se ejecutan únicamente cuando son "notificadas" por otras tareas. Esto significa que no se ejecutan de forma secuencial como las tareas regulares, sino que esperan a recibir una señal para activarse.

Se utilizan para:

- Optimización de tareas: Evitan ejecutar tareas innecesariamente. Por ejemplo, si se modifica un archivo de configuración, no es necesario reiniciar un servicio a menos que el archivo haya cambiado realmente.
- Organización de *playbooks*: Ayudan a mantener los *playbooks* más limpios y legibles al separar las tareas de configuración de las tareas de "post-configuración" (como reiniciar servicios).
- Mayor control: Permiten un control más granular sobre cuándo se ejecutan ciertas acciones.

Una tarea puede notificar a un *handler* utilizando la palabra clave *notify*. Esto indica que, si la tarea realiza algún cambio, el *handler* asociado debe ejecutarse.

Al final de la ejecución del *playbook*, Ansible verifica qué *handlers* han sido notificados y los ejecuta en el orden en que fueron notificados.
En nuestras tareas de ejemplo, la cuarta tarea lleva agregado el *notify.*
Entonces dentro de la carpeta *handlers* creamos el archivo main.yml:

```
---
- name: restart sshd
  service:
    name: sshd
    state: restarted

- name: restart ssh
  service:
    name: ssh
    state: restarted
...
```

### 4.1.9. Creación del *template*

Los *templates* o plantillas de Ansible son archivos que combinan texto estático con variables dinámicas para generar contenido personalizado. Esto es especialmente útil cuando se necesita crear archivos de configuración, scripts o cualquier otro contenido que varíe entre diferentes equipos o según otros parámetros.
Ansible utiliza el motor de plantillas Jinja2, un lenguaje de plantillas muy popular en Python. Jinja2 también permite utilizar estructuras de control como if, for y filtros para crear lógicas más complejas.
Dentro de las plantillas, se pueden utilizar variables definidas en los *playbooks* o inventarios para personalizar el contenido.
El módulo *template* de Ansible se encarga de renderizar las plantillas y escribir el resultado en el archivo que se indique en los hosts remotos.
Para crear la plantilla hay que tomar el archivo de configuración original, en este caso el *sshd_config*, y parametrizar todas las variables que se van a modificar de la siguiente forma. Entonces se crea el archivo *sshd_config.j2*:

```
Port {{ reyespab_ssh_port }}
#AddressFamily any
```

```
#ListenAddress 0.0.0.0
#ListenAddress ::

HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
SyslogFacility AUTHPRIV
#LogLevel INFO

# Authentication:

LoginGraceTime {{ reyespab_ssh_LoginGraceTime }}
PermitRootLogin {{ reyespab_ssh_PermitRootLogin }}
#StrictModes yes
MaxAuthTries {{ reyespab_ssh_MaxAuthTries }}
MaxStartups {{ reyespab_ssh_MaxStartups }}
#MaxSessions 10
AllowUsers {{ reyespab_ssh_AllowUsers }}
DenyUsers {{ reyespab_ssh_DenyUsers }}


PubkeyAuthentication {{ reyespab_ssh_PubkeyAuthentication }}

# The default is to check both .ssh/authorized_keys and # Kerberos
options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#  but  this  is  overridden  so  installations  will  only  check
.ssh/authorized_keys2
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication {{ reyespab_ssh_PasswordAuthentication }}
#PermitEmptyPasswords no
PasswordAuthentication yes
```

```
# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes
ChallengeResponseAuthentication                                    {{
reyespab_ssh_ChallengeResponseAuthentication }}

.ssh/authorized_keys
AuthorizedKeysFile     {{ reyespab_ssh_AuthorizedKeysFile }}

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

#  For  this  to  work  you  will  also  need  host  keys  in
/etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
#KerberosUseKuserok yes

# GSSAPI options
GSSAPIAuthentication yes
GSSAPICleanupCredentials no
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
#GSSAPIEnablek5users no

#  Set  this  to  'yes'  to  enable  PAM  authentication,  account
processing,
# and session processing. If this is enabled, PAM authentication
will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication.  Depending on your PAM configuration,
#  PAM  authentication  via  ChallengeResponseAuthentication  may
bypass
# the setting of "PermitRootLogin without-password".
# If you just want the PAM account and session checks to run without
#    PAM    authentication,    then    enable    this    but    set
PasswordAuthentication
```

```
# and ChallengeResponseAuthentication to 'no'.
# WARNING: 'UsePAM no' is not supported in Red Hat Enterprise Linux
and may cause several
# problems.
UsePAM {{ reyespab_ssh_UsePAM }}

#AllowAgentForwarding yes
#AllowTcpForwarding yes
#GatewayPorts no
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
#PrintMotd yes
#PrintLastLog yes
#TCPKeepAlive yes
#UseLogin no
#UsePrivilegeSeparation sandbox
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#ShowPatchLevel no
#UseDNS yes
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
# no default banner path
#Banner none

# Accept locale-related environment variables
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS

# override default of no subsystems
Subsystem       sftp  /usr/libexec/openssh/sftp-server
```

Este es un pequeño ejemplo, pero en este archivo se pueden parametrizar todas las variables que se quieran configurar en los entornos.

### 4.1.10. Creación del archivo de variables del *host*

Como se vio anteriormente, las variables de *host* tienen más prioridad que las variables del rol, por lo cual todo lo que se defina en estos archivos son los valores que finalmente van a tomar las variables. Esto facilita la personalización y aplicación de excepciones sobre cada *host* administrado con Ansible.

De esta forma vamos a crear el archivo host_vars/UBA_MSI_Prueba/reyespab.ssh.yml con el siguiente contenido:

```
---
reyespab_ssh_port: "22"
reyespab_ssh_PermitRootLogin: "no"
reyespab_ssh_LoginGraceTime: "10"
reyespab_ssh_MaxAuthTries: "3"
reyespab_ssh_MaxStartups: "2"
reyespab_ssh_AllowUsers: "ansible preyes"
reyespab_ssh_DenyUsers: "root"
reyespab_ssh_PasswordAuthentication: "yes"
reyespab_ssh_PubkeyAuthentication: "yes"
reyespab_ssh_ChallengeResponseAuthentication: "yes"
reyespab_ssh_AuthorizedKeysFile: "/etc/ssh/authorized_keys/%u"
reyespab_ssh_UsePAM: "yes"
reyespab_ssh_users:
  - remote_user: ansible
    state: present
    keys:
      - /etc/ansible/files/LlavesPublicas/ansible.pub
  - remote_user: preyes
    state: present
    keys:
      - /etc/ansible/files/LlavesPublicas/preyes.pub
      - /etc/ansible/files/LlavesPublicas/preyes_home.pub
 ...
```

De esta manera, el árbol de directorios y archivos nos quedaría de la siguiente manera:

*ansible.cfg*
*inventario.ini*
*UBA_MSI_Prueba.yml*
*host_vars/*
|-- *UBA_MSI_Prueba/*
        |-- *reyespab.ssh.yml*
        |-- *vars_otro_rol.yml*
*roles/*
|-- *reyespab.ssh/*
        |-- *tasks/*
                |-- *main.yml*
        |-- *defaults/*
                |-- *main.yml*
        |-- *templates/*
                |-- *sshd_config.j2*
        |-- *vars/*
                |-- *main.yml*
        |-- *handlers/*
                |-- *main.yml*

Por último, para realizar la ejecución del *playbook* hay que ejecutar por la línea de comandos:

*ansible-playbook UBA_MSI_Prueba.yml -i inventario.ini -vvv*

# Conclusiones

Luego del análisis de la utilización de Ansible para automatizar los procesos de seguridad informática dentro de la cultura de DevSecOps se puede llegar a las siguientes conclusiones:

En primer lugar, mejora de la eficiencia: Ansible permite automatizar tareas repetitivas y tediosas de seguridad informática, lo que conlleva a un aumento significativo en la eficiencia. Los procesos que antes llevaban mucho tiempo y esfuerzo manual ahora pueden ejecutarse de manera rápida y precisa mediante la automatización con Ansible.

En segundo lugar, reducción de errores humanos: Al automatizar los procesos, se minimiza el riesgo de cometer errores humanos. Al no depender de la intervención manual en cada paso del proceso, se eliminan los errores asociados a omisiones, malentendidos o cansancio. Esto resulta en una mayor precisión y disminución de los riesgos en los sistemas informáticos. También vamos a tener consistencia en la implementación: Ansible garantiza que los procesos de seguridad se implementen de manera coherente en todos los sistemas. Al definir y codificar las tareas de seguridad en un *playbook*, se asegura que todos los sistemas sigan las mismas pautas de seguridad. Esto es especialmente beneficioso en entornos empresariales con múltiples máquinas y sistemas distribuidos.

Otro punto importante que destaco es la flexibilidad y escalabilidad: Ansible se adapta a diferentes entornos y es altamente escalable. Esto significa que puede utilizarse en proyectos de cualquier tamaño, desde pequeñas empresas hasta grandes organizaciones. Puede aplicarse a una amplia gama de tareas de seguridad, como la instalación de parches, actualización de software, configuraciones de *firewall*, entre otras. Además, Ansible permite fácilmente agregar nuevos sistemas a medida que la infraestructura crece. Por último, documentación y auditabilidad: Ansible proporciona un registro detallado de todas las acciones realizadas, lo que facilita la documentación y la auditoría de los procesos de seguridad. Esto es esencial para cumplir con los requisitos reglamentarios y facilitar la resolución de problemas. Los

registros de Ansible ayudan a rastrear y revisar las acciones realizadas, lo que proporciona un mayor nivel de transparencia y responsabilidad en la gestión de la seguridad informática.

Estas conclusiones evidencian los beneficios que puede aportar Ansible en la mejora de la seguridad informática y la gestión de sistemas en entornos tanto pequeños como empresariales.

# Glosario

**Infraestructura**: se refiere al conjunto de componentes *hardware* y *software* que sustentan las operaciones de una organización. Incluye servidores, redes, sistemas de almacenamiento, aplicaciones y los servicios que estos proporcionan. La infraestructura es el entorno en el cual se ejecutan los sistemas y aplicaciones y, por lo tanto, es un objetivo clave para los ataques cibernéticos.

***Open Source*** (código abierto): es un modelo de desarrollo de software en el que el código fuente de un programa está disponible para cualquier persona para su uso, estudio, modificación y distribución. Este modelo fomenta la colaboración, la transparencia y la innovación, ya que permite a una comunidad de desarrolladores contribuir al proyecto.

**IaC** (Infraestructura como código): es una práctica de gestión de infraestructura que utiliza lenguajes de programación para definir y provisionar recursos informáticos. En lugar de configurar manualmente servidores, redes y otros componentes, se emplean archivos de configuración que describen la infraestructura deseada. Estos archivos se almacenan en un repositorio de código fuente y se utilizan para automatizar la creación, modificación y destrucción de la infraestructura.

**Aprovisionamiento**: se refiere al proceso de configurar y hacer disponibles los recursos necesarios para un sistema o servicio.

**CI/CD**: son las siglas de Integración Continua y Entrega Continua (o Despliegue Continuo). Es una metodología de desarrollo de software que busca automatizar el proceso de construcción, prueba y despliegue de aplicaciones.

- Integración Continua (CI): Los cambios de código se integran y se verifican automáticamente en un repositorio central con frecuencia.
- Entrega Continua (CD): Los cambios de código se despliegan automáticamente en un entorno de producción o preproducción.

***Hardening*** (endurecimiento): es el proceso de fortalecer la seguridad de un sistema operativo, aplicación o dispositivo mediante la aplicación de una serie de medidas de seguridad. El objetivo es minimizar la superficie de ataque y reducir la probabilidad de que un sistema sea comprometido.

# Bibliografía

A. V. Yunga Toaquiza (2018). Implementación del aprovisionamiento automático de configuraciones en infraestructuras multivendor con Ansible. Disponible en: http://dspace.espoch.edu.ec/bitstream/123456789/10935/1/98T00227.pdf. Acceso en: 02/05/2024

J. C. Seco (2023). DevOps: evolución a DevSecOps y principales herramientas - Izertis. Disponible en: https://www.izertis.com/es/-/blog/devops-evolucion-a-devsecops-y-principales-herramientas. Acceso en: 15/12/2023.

J. Castro Sánchez (2020). DevSecOps: Implementación de seguridad en DevOps a través de herramientas open source (Trabajo final de máster). Universidad Oberta de Catalunya. Recuperado de https://openaccess.uoc.edu/bitstream/10609/126766/11/jcastrosancheTFM1220memoria.pdf. Acceso en: 12/12/2023

J. M. Ali (2023). DevOps and continuous integration/continuous deployment (CI/CD) automation, *Advances in Engineering Innovation*, vol. 4, no. 1, pp. 38–42. Disponible en: https://www.jetbrains.com/es-es/teamcity/ci-cd-guide/continuous-deployment/ . Acceso en: 12/122023.

RedHat (2023). ¿Qué es DevSecOps? Seguridad integrada dentro de DevOps. Disponible en: https://www.redhat.com/es/topics/devops/what-is-devsecops. Acceso en: 13/12/2023.

RedHat (2021). Cinco formas de utilizar la automatización de la TI para implementar DevSecOps de forma exitosa. Disponible en: https://www.redhat.com/es/resources/5-ways-to-devsecops-with-automation-checklist. Acceso en: 15/12/2023.

RedHat (2023). Red Hat Ansible Automation Platform para la automatización de la seguridad. Disponible en: https://www.redhat.com/es/technologies/management/ansible/security-automation. Acceso en: 15/12/2023.

IBM Cloud Docs (2024). Instalar roles desde Ansible Galaxy. Disponible en: https://cloud.ibm.com/docs/schematics?topic=schematics-ansible-roles-galaxy&locale=es. Acceso en: 01/10/2024.

Ansible Community (2024). Guía de usuario de Ansible. Disponible en: https://docs.ansible.com/ansible/latest/galaxy/user_guide.html. Acceso en: 01/10/2024.

Ansible Community (2024). Guía de instalación de Ansible. Disponible en: https://docs.ansible.com/ansible/latest/installation_guide/index.html. Acceso en: 03/10/2024.

Pulumi (2024). Documentación oficial de Pulumi. Disponible en: https://www.pulumi.com/docs/concepts/. Acceso en: 03/10/2024.

HashiCorp (2024). Documentación oficial de Terraform. Disponible en: https://developer.hashicorp.com/terraform/intro. Acceso en: 03/10/2024.

Puppet (2024). Documentación oficial de Puppet. Disponible en: https://www.puppet.com/docs/puppet/8/puppet_overview. Acceso en: 03/10/2024.

Progress Software Corporation (2024). Documentación oficial de Chef. Disponible en: https://docs.chef.io/automate/. Acceso en: 05/10/2024.

VMWare, Inc. (2024). Documentación oficial de Salt. Disponible en: https://docs.saltproject.io/en/latest/topics/about_salt_project.html. Acceso en: 05/10/2024.