**AGRADECIMIENTOS**

A mi familia, por seguir animándome a continuar creciendo en la Universidad de Almería. A mis compañeros, porque desde el Grado hemos seguido apoyándonos continuamente en nuestras vidas. A mis amigos, por estar conmigo tanto en las buenas como en las malas. Al profesorado de esta universidad, donde gracias a ellos he podido adquirir estos nuevos conocimientos en esta ingeniería, y también y sobretodo a Manuel Torres Gil, por darme la oportunidad de realizar este proyecto donde he podido poner a prueba toda mi experiencia profesional previa junto a lo aprendido en este máster para mi desarrollo y desempeño personal y laboral.

Índice de contenidos

[Capítulo 01: Introducción 6](#_Toc190605229)

[1.1. Motivación 6](#_Toc190605230)

[1.2. Objetivo general 6](#_Toc190605231)

[1.3. Objetivos específicos 6](#_Toc190605232)

[1.4. Planificación 7](#_Toc190605233)

[1.5. Estructura del documento 8](#_Toc190605234)

[Capítulo 02: Estado del arte 9](#_Toc190605235)

[2.1 Introducción a la seguridad en aplicaciones 9](#_Toc190605236)

[2.2 Métodos tradicionales de autenticación y autorización 10](#_Toc190605237)

[2.2.1 PAP (Password Access Protocol) 10](#_Toc190605238)

[2.2.2 Lightweight directory access protocol (LDAP) 11](#_Toc190605239)

[2.2.3 MFA (Multifactor Authentication) 12](#_Toc190605240)

[2.2.4 SAML (Security Assertion Markup Language) y Single Sign-On (SSO) 12](#_Toc190605241)

[2.2.5 Problemas en soluciones tradicionales 13](#_Toc190605242)

[2.3 Tecnologías modernas 14](#_Toc190605243)

[2.3.1 OAuth 14](#_Toc190605244)

[2.3.2 OpenID Connect (OIDC) 15](#_Toc190605245)

[2.3.3 JSON Web Tokens (JWT) 16](#_Toc190605246)

[2.3.4 Identity as a Services (IDaaS) 17](#_Toc190605247)

[2.4 Herramientas principales: Keycloak y Vault 18](#_Toc190605248)

[2.4.1 Keycloak 18](#_Toc190605249)

[2.4.2 Vault 19](#_Toc190605250)

[2.5 Contenedores, orquestación y DevOps 19](#_Toc190605251)

[2.5.1 Contenedores 19](#_Toc190605252)

[2.5.2 Orquestación con Kubernetes 21](#_Toc190605253)

[2.5.3 DevOps 21](#_Toc190605254)

[2.5.4 DevSecOps 22](#_Toc190605255)

[2.7 Modelos de cloud computing 23](#_Toc190605256)

[2.7.1 Infraestructura como Servicio (IaaS) 24](#_Toc190605257)

[2.7.2 Plataforma como Servicio (PaaS) 24](#_Toc190605258)

[2.7.3 Software como Servicio (SaaS) 25](#_Toc190605259)

[2.8 Infraestructura como código (IaC) 25](#_Toc190605260)

[2.8.1 Terraform 25](#_Toc190605261)

[2.8.2 Pulumi 26](#_Toc190605262)

[2.8.3 Ansible 26](#_Toc190605263)

[2.8.4 Chef 27](#_Toc190605264)

[2.8.5 Puppet 28](#_Toc190605265)

[2.9 Resumen del capítulo 28](#_Toc190605266)

[Capítulo 03: Herramientas y tecnologías 30](#_Toc190605267)

[3.1 Virtualización con Docker 30](#_Toc190605268)

[3.2 Keycloak como servicio 32](#_Toc190605269)

[3.3 Vault como servicio 33](#_Toc190605270)

[3.4 Terraform 34](#_Toc190605271)

[3.5 Resumen del capítulo 36](#_Toc190605272)

[Capítulo 04: Diseño e implementación 37](#_Toc190605273)

[4.1 Investigación, propuesta y formación 37](#_Toc190605274)

[4.2 Inicialización del entorno 37](#_Toc190605275)

[4.3 Desarrollo e implementación del servicio 37](#_Toc190605276)

[4.4 Pruebas 37](#_Toc190605277)

[4.5 Resumen del capítulo 37](#_Toc190605278)

[Capítulo 05: Conclusiones y trabajo futuro 38](#_Toc190605279)

[5.1 Ventajas 38](#_Toc190605280)

[5.2 Desventajas 38](#_Toc190605281)

[5.3 Trabajo futuro 38](#_Toc190605282)

[5.4 Resumen del capítulo 38](#_Toc190605283)

[Capítulo 06: Bibliografía 40](#_Toc190605284)

Índice de figuras

[Figura 1. CIA 9](#_Toc190605285)

[Figura 2. PAP 10](#_Toc190605286)

[Figura 3. Esquema LDAP 11](#_Toc190605287)

[Figura 4. Protocolo MFA 12](#_Toc190605288)

[Figura 5. Protocolo SAML 12](#_Toc190605289)

[Figura 6. Single Sign-On (SSO) 13](#_Toc190605290)

[Figura 7. OAuth 14](#_Toc190605291)

[Figura 8. OIDC 15](#_Toc190605292)

[Figura 9. Token JWT 16](#_Toc190605293)

[Figura 10. Esquema IDaaS 17](#_Toc190605294)

[Figura 11. Arquitectura de Keycloak 18](#_Toc190605295)

[Figura 12. Arquitectura de Vault 19](#_Toc190605296)

[Figura 13. Comparativa máquinas virtuales y contenedores 20](#_Toc190605297)

[Figura 14. Modularidad Docker 20](#_Toc190605298)

[Figura 15. Docker y Kubernetes 21](#_Toc190605299)

[Figura 16. Arquitectura DevOps 22](#_Toc190605300)

[Figura 17. Arquitectura DevSecOps 22](#_Toc190605301)

[Figura 18. Comparativa DevOps y DevSecOps 23](#_Toc190605302)

[Figura 19. Modelos cloud computing 24](#_Toc190605303)

[Figura 20. Esquema flujo IaC 25](#_Toc190605304)

[Figura 21. Flujo de trabajo Terraform 26](#_Toc190605305)

[Figura 22. Pulumi 26](#_Toc190605306)

[Figura 23. Ansible 27](#_Toc190605307)

[Figura 24. Arquitectura Chef 27](#_Toc190605308)

[Figura 25. Puppet 28](#_Toc190605309)

[Figura 26. Virtualización con Docker 30](#_Toc190605310)

[Figura 27. Comandos Docker 31](#_Toc190605311)

[Figura 28. Arquitectura de aplicaciones con Keycloak 32](#_Toc190605312)

[Figura 29. Keycloak como servicio 33](#_Toc190605313)

[Figura 30. Vault como servicio 34](#_Toc190605314)

[Figura 31. Comandos Terraform 35](#_Toc190605315)

[Figura 32. Flujo de comandos Terraform 35](#_Toc190605316)

# 

# ****Capítulo 01: Introducción****

## ****1.1. Motivación****

En la actual era digital, la tecnología avanza a un ritmo vertiginoso, lo que conlleva un aumento significativo en la complejidad y funcionalidad de las aplicaciones. Este crecimiento no solo amplía sus capacidades, sino que también incrementa su exposición a potenciales amenazas de seguridad [1]. Las aplicaciones modernas manejan grandes volúmenes de datos sensibles y requieren mecanismos robustos para proteger la integridad, confidencialidad y disponibilidad de la información.

Sin una gestión adecuada de la seguridad, las aplicaciones son vulnerables a ataques como el robo de credenciales, acceso no autorizado y exfiltración de datos. Implementar internamente sistemas de autenticación y gestión de credenciales puede ser un proceso laborioso y propenso a errores, especialmente si no se cuenta con experiencia especializada en seguridad.

Delegar estas funciones críticas a plataformas especializadas para autenticación, autorización [2] y gestión segura de secrets y credenciales [3] permite a los desarrolladores centrarse en la lógica principal del negocio. Esto no solo mejora la eficiencia en el desarrollo, sino que también garantiza que las prácticas de seguridad estén alineadas con los estándares y mejores prácticas de la industria [4], reduciendo significativamente los riesgos asociados a la seguridad de la información.

## ****1.2. Objetivo general****

El objetivo de este Trabajo Fin de Máster es diseñar e implementar una solución integral que proporcione un sistema de autenticación, autorización y gestión segura de credenciales para aplicaciones que carecen de estas funcionalidades de forma nativa. Al ofrecer una plataforma especializada para estas funciones críticas, se busca permitir a los desarrolladores centrarse en la lógica principal del negocio, mejorando la eficiencia en el desarrollo y garantizando que las prácticas de seguridad estén alineadas con los estándares y mejores prácticas de la industria. Incluso para el caso de aplicaciones ya en producción que carecen de este sistema o servicio de autentificación, tener la posibilidad de intetrarlo en ellas sin una modificación o relativamente mínimo del código fuente original. Esto reducirá significativamente los riesgos asociados a la seguridad de la información.

En concreto, esto se realizará haciendo uso de la combinación de las tecnologías Keycloak y Vault para la creación de un sistema de autenticación, autorización y gestión segura de credenciales para su integración aplicaciones que carecen de estas funcionalidades de forma nativa, combinando con la herramienta utilizando Docker como motor para poder incluir en contenedores estos servicios que componen este sistema.

## ****1.3. Objetivos específicos****

**Los objetivos a alcanzar a nivel técnico son los siguientes:**

* **Configurar un servidor de identidad para autenticación y autorización.** Se implementará un servidor que actúe como proveedor de identidad (IdP), donde se definirán y gestionarán usuarios, grupos y roles. Se configurarán clientes y recursos protegidos, estableciendo políticas de acceso basadas en roles y atributos. Además, se integrará este servidor con las aplicaciones objetivo utilizando protocolos estándar como OpenID Connect [5].
* **Utilizar una solución para la protección y gestión de credenciales sensibles.** Se instalará y configurará un servicio dedicado a la gestión de secrets [6]. Se definirán políticas y controles de acceso para usuarios y aplicaciones que necesiten acceder a información confidencial. Se implementarán mecanismos de autenticación [7] como tokens, AppRole o integración con el proveedor de identidad.
* **Implementar ambas herramientas en un entorno aislado mediante contenedores.** Se crearán imágenes personalizadas con las configuraciones específicas necesarias [8] para cada servicio almacenado en contenedores y se orquestarán de forma conjunta. Se garantizará que los contenedores se ejecuten en redes aisladas y seguras, gestionando adecuadamente los puertos y las comunicaciones internas entre ellos.
* **Automatizar el despliegue con una solución basada en IaaC para garantizar escalabilidad.** Se implementarán scripts de automatización de despliegue de toda la infraestructura necesaria para el funcionamiento y arranque de este nuevo entorno [9] que compone todo el nuevo sistema de autentificación, autentificación y gestión de credenciales.

## **1**.4. Planificación****

El proyecto se ha organizado llevando a cabo una serie de iteraciones o hitos, introduciendo desde un primer inicio tareas de investigación sobre el uso de las diferentes tecnologías que se aplican hasta el desarrollo final del proyecto, realizando revisiones del estado de cada uno de los hitos. Se ha llevado a cabo esta estrategia para que el proyecto durante sus fases de desarrollo con las siguientes características:

* Sea susceptible a continuos cambios.
* Se obtenga una mayor calidad del software.
* Permita una mayor productividad. Se consigue entre otras razones, gracias a la eliminación de la burocracia y a la motivación del equipo que proporciona el hecho de que sean autónomos para organizarse.
* Facilidades en las estimaciones de tiempos de las iteraciones.
* Reducción de posibles riesgos durante su avance. El hecho de llevar a cabo las funcionalidades de más valor en primer lugar y de conocer la velocidad de avance en el proyecto, se permiten detectar errores y riesgos de manera anticipada, pudiendo actuar y tomar decicisones de forma proactiva y consensuada.

El desarrollo del TFM se compone de las siguientes fases:

1. **Estudio y aprendizaje de Keycloak** (40h).
2. **Estudio de los diferentes métodos de autenticación de usuarios** (50h).
3. **Estudio y aprendizaje de Vault** (30h).
4. **Creación servicio gestión y autentificación de usuarios** (40h).
5. **Integración servicio gestión y autentificación de usuarios** (30h).
6. **Automatización configuración del nuevo servicio** (60h).
7. **Despliegue automatizado infraestructura** (20h).
8. **Documentación memoria TFM** (30h).

## ****1.5. Estructura del documento****

El documento seguirá una estructura lógica y coherente para facilitar su lectura y comprensión:

* **Capítulo 01: Introducción**.
* **Capítulo 02: Estado del Arte**.
* **Capítulo 03: Descripción de Herramientas**.
* **Capítulo 04: Diseño e Implementación**.
* **Capítulo 05: Resultados y pruebas**.
* **Capítulo 06: Conclusiones y trabajos future.**
* **Capítulo 7: Bibliografía.**

# Capítulo 02: Estado del arte

Este capítulo ha profundizado en la evolución y el estado actual de la seguridad en aplicaciones, destacando las limitaciones de los métodos tradicionales y la necesidad de adoptar tecnologías y prácticas modernas. La integración de herramientas como Keycloak y Vault, junto con enfoques como DevOps, DevSecOps y la orquestación con Kubernetes, proporciona un marco sólido para abordar los desafíos de seguridad en el panorama tecnológico actual. Además, el uso de Infrastructure as Code y herramientas proporcionadas por compañías como HashiCorp facilita la gestión eficiente y segura de infraestructuras en la nube, permitiendo a las organizaciones mejorar significativamente su postura de seguridad, proteger datos sensibles y cumplir con la normativa y estándares de seguridad.

## 2.1 Introducción a la seguridad en aplicaciones

La seguridad en aplicaciones es un pilar fundamental en el ámbito de la tecnología de la información y las comunicaciones. Con la creciente dependencia de sistemas informáticos en todos los sectores, desde la banca y la salud hasta la educación y el comercio electrónico, garantizar la protección de datos y recursos es más crítico que nunca. La evolución de la seguridad en aplicaciones ha pasado de simples validaciones de usuario y contraseñas estáticas a sistemas avanzados que incorporan inteligencia artificial, análisis de comportamiento y criptografía avanzada.

En las primeras etapas de la computación, la seguridad se centraba en el control de acceso físico y en la protección contra errores humanos. Sin embargo, con la expansión de las redes y el acceso remoto, las amenazas han aumentado exponencialmente. Los ataques cibernéticos modernos son altamente sofisticados, dirigidos y pueden causar daños significativos, incluyendo pérdidas financieras, robo de propiedad intelectual y daños a la reputación.

La CIA (Confidenciality, Integrity and Avaliability) sigue siendo la base de la seguridad de la información:

* **Confidencialidad**. Garantiza que la información sea accesible solo para las personas autorizadas.
* **Integridad**. Asegura que la información sea precisa y completa, y que no haya sido alterada de manera no autorizada.
* **Disponibilidad**. Asegura que los sistemas y datos estén disponibles para los usuarios autorizados cuando se necesiten.



Figura 1. CIA

La creciente adopción de tecnologías emergentes como la nube, IoT (Internet de las Cosas) y la inteligencia artificial ha introducido nuevas formas de ataque. Además, la pandemia de COVID-19 aceleró la digitalización de muchos procesos de negocio de las distintas empresas y servicios online que ofrecen, aumentando aún más la exposición a amenazas cibernéticas. Por lo tanto, es imperativo que las organizaciones adopten enfoques proactivos y tecnologías avanzadas para proteger sus aplicaciones y datos.

## ****2.2 Métodos tradicionales de autenticación y autorización****

### ****2.2.1 PAP (Password Access Protocol)****

Los sistemas basados en contraseñas, cuyo protocolo se conoce como **PAP**, son uno de los métodos de autenticación más antiguos y ampliamente utilizados en la historia de la informática. Su origen se remonta a la década de 1960 con el desarrollo del Compatible Time-Sharing System (CTSS) en el Instituto Tecnológico de Massachusetts (MIT), donde se implementaron por primera vez contraseñas para separar y proteger los archivos de diferentes usuarios [1].

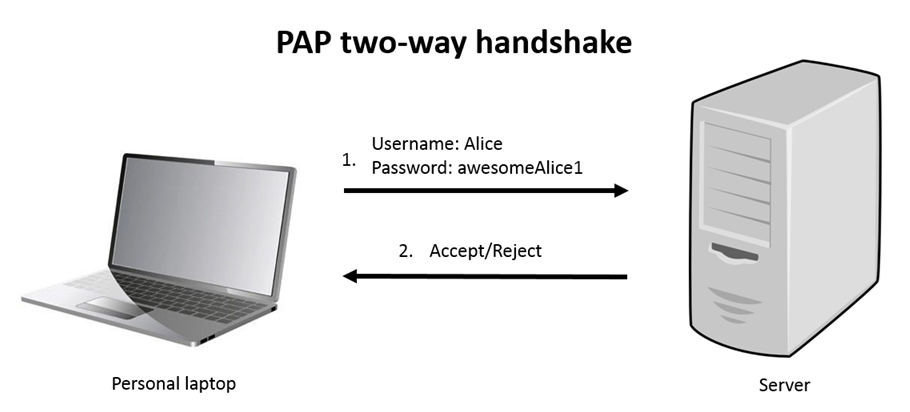


Figura 2. PAP

Antes de la introducción de las contraseñas, el acceso a los sistemas informáticos era limitado y controlado físicamente, lo que significaba que la seguridad dependía en gran medida del control de acceso al hardware, cuyas limitaciones eran las siguientes:

* **Reutilización de contraseñas:** Estudios indican que aproximadamente el 60% de las personas reutilizan las mismas contraseñas en múltiples sitios web [2]. Esto significa que si un atacante obtiene una contraseña en un sitio, es muy probable que pueda acceder a otras cuentas del usuario.
* **Debilidad de contraseñas**: Los usuarios suelen elegir contraseñas fáciles de recordar, como "123456" o "contraseña", que son fácilmente vulnerables a ataques de fuerza bruta y de diccionario [3].
* **Phishing e ingeniería social:** Los atacantes pueden engañar a los usuarios para que revelen sus contraseñas mediante correos electrónicos fraudulentos o sitios web falsos [4].
* **Gestión y almacenamiento:** Las organizaciones deben almacenar las contraseñas de forma segura, generalmente mediante técnicas de hash y salting, pero las malas prácticas pueden conducir a filtraciones.

Estas limitaciones evidenciaron la necesidad de sistemas más seguros y centralizados para la gestión de identidades. A medida que las organizaciones crecían y los sistemas se volvían más complejos, se hizo evidente que las contraseñas por sí solas no eran suficientes para garantizar la seguridad.

### ****2.2.2 Lightweight directory access protocol (LDAP)****

Para afrontar las limitaciones de los sistemas basados en contraseñas y mejorar la gestión de identidades, se desarrollaron protocolos que permitían una administración centralizada. En la década de 1990, el protocolo **LDAP** fue creado como una versión simplificada del protocolo DAP (Directory Access Protocol) utilizado en el modelo X.500 de directorios de la ISO [5]. Algunas de sus características a destacar son**:**

* **Almacenamiento centralizado de identidades:** Permite gestionar usuarios, grupos y permisos en un repositorio único.
* **Integración con sistemas existentes:** Compatible con diversas aplicaciones y sistemas operativos.
* **Escalabilidad:** Adecuado para organizaciones de diferentes tamaños.

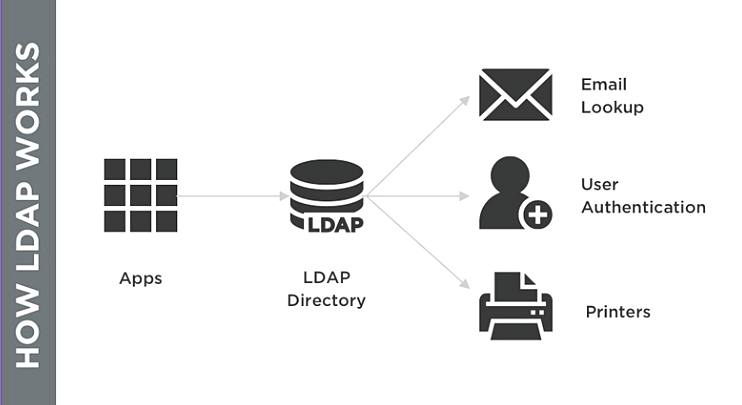


Figura 3. Esquema LDAP

LDAP mejoró significativamente la gestión de identidades al proporcionar un método estandarizado para acceder y mantener directorios de información. Sin embargo, aún dependía principalmente de contraseñas para la autenticación, lo que no resolvía completamente los problemas de seguridad asociados. Además, su implementación y mantenimiento requerían conocimientos especializados, lo que podía ser un obstáculo para algunas organizaciones.

### ****2.2.3 MFA (Multifactor Authentication)****

Las crecientes amenazas y vulnerabilidades asociadas con las contraseñas llevaron a desarrollar métodos que añadieran capas adicionales de seguridad. A finales de los años 90 y principios de los 2000, las organizaciones comenzaron a implementar la autenticación multifactor conocida como **MFA [6]**. MFA surgió como respuesta directa a las limitaciones de la autenticación basada únicamente en contraseñas, proporcionando una forma de verificar la identidad del usuario utilizando múltiples factores independientes.

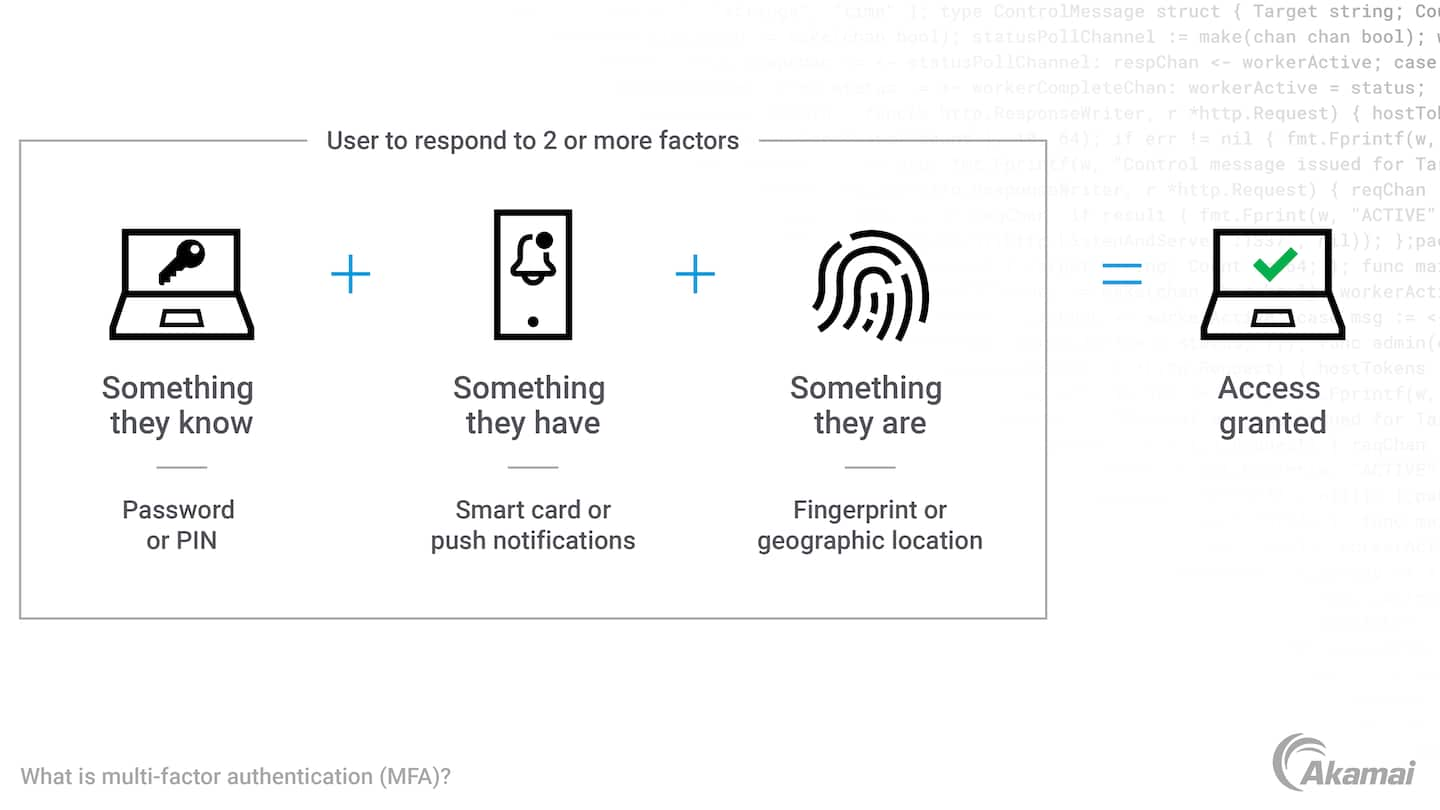


Figura 4. Protocolo MFA

**La autenticación multifactor requiere que los usuarios proporcionen dos o más factores de autenticación independientes para verificar su identidad [7]. Los factores de autenticación se dividen en:**

* **Conocimiento:** Algo que el usuario sabe (contraseña, PIN).
* **Posesión:** Algo que el usuario tiene (token, tarjeta inteligente, dispositivo móvil).
* **Inherencia:** Algo que el usuario es (huella dactilar, reconocimiento facial, patrón de iris).

MFA mejoró significativamente la seguridad al hacer más difícil para los atacantes comprometer una cuenta. Sin embargo, también introdujo desafíos en términos de costo, complejidad y experiencia del usuario. La necesidad de dispositivos adicionales y la posible resistencia de los usuarios a procesos más complicados limitaron su adopción generalizada en algunos contextos.

### ****2.2.4 SAML (Security Assertion Markup Language) y Single Sign-On (SSO)****

Con el aumento de servicios y aplicaciones web a principios de los 2000, surgió la necesidad de una forma segura y estandarizada de intercambiar información de autenticación y autorización entre diferentes dominios. Las soluciones propietarias dificultaban la interoperabilidad, lo que llevó al desarrollo de **SAML** por parte de OASIS en 2002 [8].

**SAML es un estándar abierto basado en XML para el intercambio de datos de autenticación y autorización entre dominios de seguridad [9], en donde se requiere de un proveedor de identidad (IdP) para autenticar al usuario y un proveedor de servicio (SP) para permitir su acceso.**

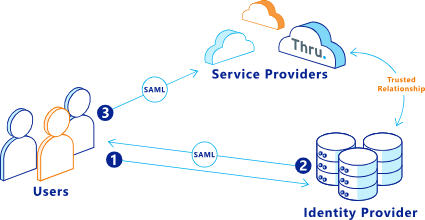


Figura 5. Protocolo SAML

Por otro lado, **Single Sign-On (SSO)** **permite el acceso a múltiples aplicaciones con una sola autenticación [10]**, mejorando consigo también la seguridad y la experiencia del usuario, siguiendo un sistema de validación de usuario y acceso similar para la autentificación a la que se emplea con SAML.

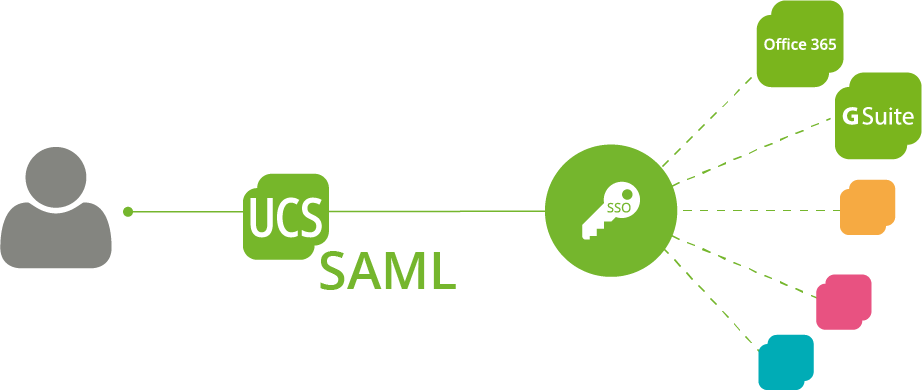


Figura 6. Single Sign-On (SSO)

**SAML y SSO abordaron la necesidad de interoperabilidad y mejoraron la experiencia del usuario al reducir la cantidad de veces que necesitaban autenticarse en diferentes aplicaciones y servicios**. Sin embargo, la complejidad en su implementación y la necesidad de una comprensión profunda del protocolo limitaron su adopción en algunos casos. Además, con la evolución de las aplicaciones web y móviles, surgieron nuevas necesidades que SAML no podía satisfacer completamente.

### ****2.2.5 Problemas en soluciones tradicionales****

A pesar de los avances, las soluciones tradicionales enfrentaron varios problemas en el contexto de las tecnologías emergentes:

* **Escalabilidad y adaptabilidad:** Con la aparición de aplicaciones móviles y servicios en la nube, los sistemas basados en contraseñas, LDAP y SAML mostraron limitaciones para escalar eficientemente y adaptarse a nuevos modelos arquitectónicos como los microservicios [11].
* **Experiencia del usuario:** Los procesos de autenticación podían ser engorrosos, afectando la usabilidad y la productividad.
* **Seguridad insuficiente:** Las amenazas avanzadas como el phishing dirigido y los ataques de intermediario (MITM) requerían soluciones más robustas.
* **Cumplimiento normativo:** Regulaciones como el GDPR impusieron requisitos estrictos sobre la protección de datos y la privacidad, que las soluciones tradicionales podían no cumplir adecuadamente.

Estas limitaciones impulsaron la búsqueda de nuevas tecnologías que pudieran satisfacer las necesidades de seguridad, usabilidad y cumplimiento en un entorno tecnológico en rápida evolución.

## ****2.3 Tecnologías modernas****

En respuesta a los desafíos presentados por las soluciones tradicionales, surgieron tecnologías modernas diseñadas para ofrecer mayor seguridad, flexibilidad y mejor experiencia de usuario. Estas tecnologías adoptan enfoques innovadores para la autenticación y autorización, adaptándose a las demandas actuales del mercado y las regulaciones vigentes.

### ****2.3.1 OAuth****

Con el auge de las redes sociales y las APIs web a finales de la década de 2000, surgió la necesidad de permitir que aplicaciones de terceros accedieran a los datos del usuario sin exponer sus credenciales. En 2007, **OAuth 1.0** fue desarrollado para abordar esta necesidad [12]. Este protocolo permitió la delegación de acceso, permitiendo a los usuarios autorizar a aplicaciones a acceder a sus datos en otro servicio de forma segura, aunque el proceso de autentificación requería de unas firmas criptográficas bastante complejas de implementar y mantener, además de que no era fácilmente adaptable a las aplicaciones.

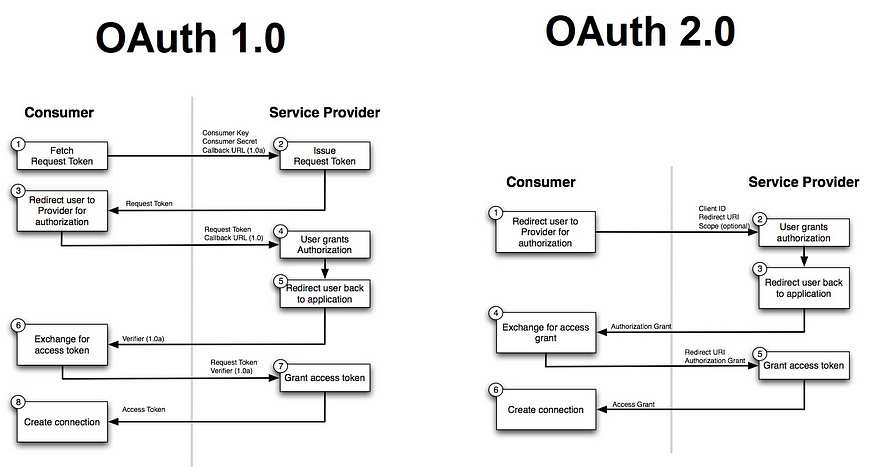


Figura 7. OAuth

Para superar las limitaciones de OAuth 1.0, apareció una evolución de este protocolo conocido como **OAuth 2.0** en 2012, proporcionando un marco más flexible y simplificado [13]. OAuth 2.0 se centró en mejorar la facilidad de uso y la adaptabilidad a diferentes tipos de aplicaciones, desde aplicaciones web hasta móviles y de escritorio.

Entre los avances hacia la versión actual de OAuth 2.0 y los beneficios que aporta son:

* **Flexibilidad mejorada:** Soporta diversos flujos de autorización (*grant types*) para adaptarse a diferentes escenarios y tipos de aplicaciones.
* **Simplicidad:** Eliminó la necesidad de firmar cada solicitud, reduciendo la complejidad y facilitando su adopción.
* **Amplia adopción:** Se convirtió en el estándar de facto para autorización en aplicaciones modernas y servicios en línea.

Sin embargo,no aborda directamente la autenticación del usuario, lo que llevó a interpretaciones erróneas y posibles vulnerabilidades si se implementaba incorrectamente, con lo que surgió la necesidad de un estándar que proporcionase un método de autenticación más seguro y simple de implementar.

### ****2.3.2 OpenID Connect (OIDC)****

Reconociendo que OAuth 2.0 no proporcionaba un mecanismo estándar para la autenticación de usuarios, la Fundación OpenID desarrolló **OpenID Connect** en 2014 [14]. Esta tecnología actúa como una capa de identidad sobre el protocolo OAuth 2.0, añadiendo el paso de autenticación al proceso de autorización ya existente. OAuth 2.0 se centra principalmente en la autorización (otorgar acceso limitado a recursos), pero no resuelve la autenticación (verificar la identidad del usuario). OpenID Connect llena este vacío al añadir una capa de autenticación que, al finalizar el proceso de OAuth 2.0, permite a la aplicación no solo saber que el usuario concedió acceso, sino también quién es.

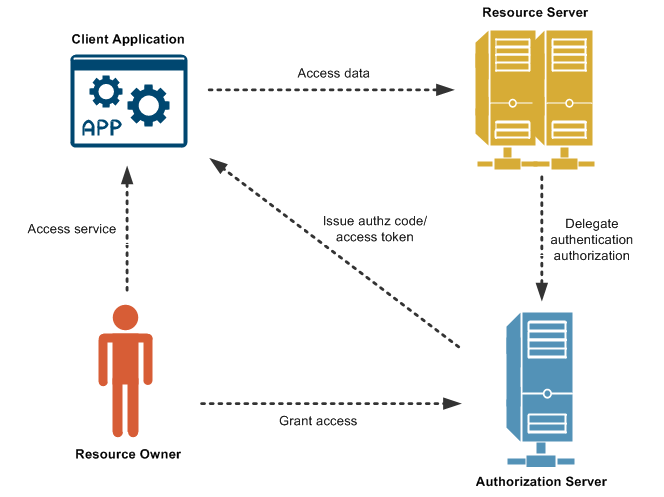


Figura 8. OIDC

Al emplear estándares como JSON Web Tokens (JWT), OpenID Connect agiliza la implementación de la autenticación en diferentes plataformas y lenguajes, permitiendo compartir de forma segura la información de identidad entre varios sistemas.

En cuanto a las características y beneficios de OpenID Connect, ofrece un marco sencillo y seguro para confirmar la identidad del usuario de manera estandarizada, lo que garantiza que el sistema reconozca con fiabilidad quién está iniciando sesión. Además, facilita la interoperabilidad al ser un estándar ampliamente adoptado, con lo cual se simplifica la integración entre diversos proveedores de identidad y aplicaciones o servicios sin necesidad de soluciones personalizadas. Por último, utiliza JWT para transmitir de forma segura la información de identidad del usuario mediante el ID Token, lo que simplifica tanto la validación como la integración con otros sistemas.

### ****2.3.3 JSON Web Tokens (JWT)****

Con la creciente adopción de arquitecturas sin estado y de APIs RESTful, surgió la necesidad de contar con un mecanismo eficiente para transmitir información de autenticación y autorización de forma segura y ligera. En 2015, se estandarizaron los **JSON Web Tokens (JWT)** a través de la **RFC 7519** [15], convirtiéndose en una solución ampliamente utilizada.

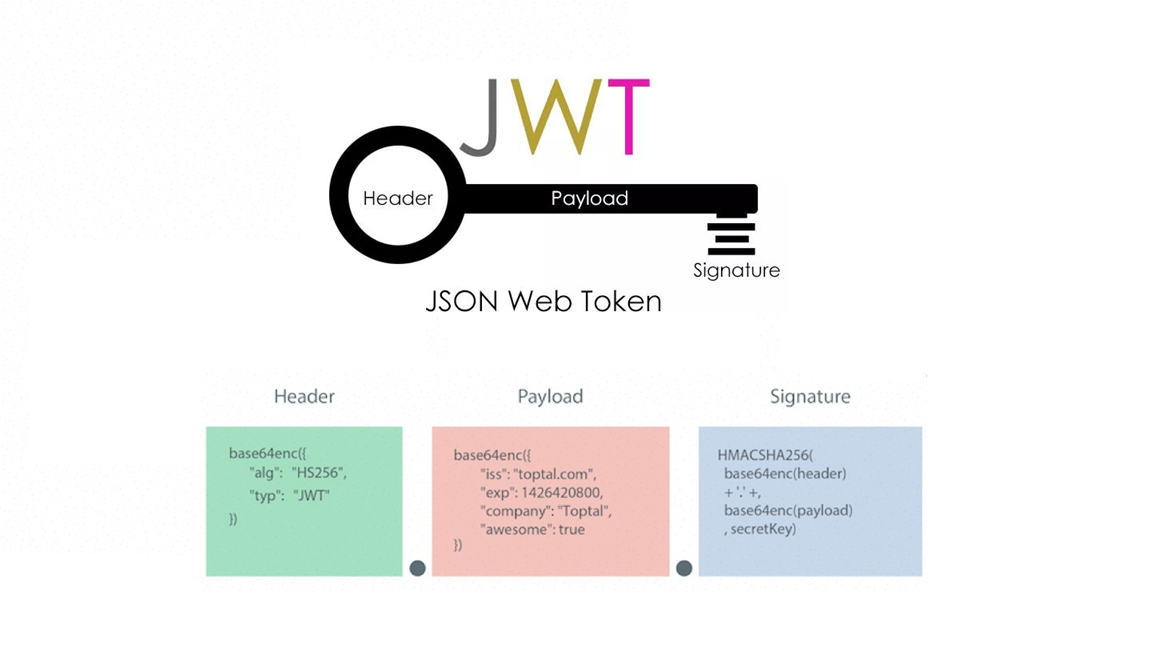


Figura 9. Token JWT

El uso de JWT hizo que se mejorara sobre los siguientes aspectos:

* **Autenticación sin estado**: Permite a los servidores validar la identidad del usuario sin mantener sesiones en el servidor, reduciendo la complejidad y los recursos necesarios para la gestión de estados.
* **Eficiencia y portabilidad**: Al ser compacto y autónomo, es ideal para aplicaciones web, móviles y APIs, facilitando la integración y reduciendo el consumo de ancho de banda.
* **Integridad y seguridad**: Los tokens están firmados criptográficamente, lo que permite comprobar su autenticidad y evitar manipulaciones, garantizando así la confiabilidad de los datos transmitidos.

La incorporación de JWT mejoró además las capacidades de **OAuth 2.0** y **OpenID Connect** al proporcionar un formato estandarizado y seguro para transmitir información. En OpenID Connect, el **Token ID** se implementa como un JWT que incluye datos relativos a la autenticación y la identidad del usuario, unificando así el proceso de autorización y autenticación dentro de un mismo flujo.

### ****2.3.4 Identity as a Services (IDaaS)****

Con el auge de los servicios en la nube y las aplicaciones SaaS a partir de la década de 2010, las organizaciones empezaron a buscar formas más eficientes de gestionar identidades en entornos híbridos y multicloud. Como respuesta a estas necesidades surgió **Identity as a Service (IDaaS) [16]**, un modelo que centraliza la gestión de identidades y accesos a través de servicios totalmente basados en la nube, que ofrece servicios de administración de identidades sin requerir infraestructura local (on-premise), reduciendo así costos y complejidad en el despliegue. Además, es compatible con protocolos de autenticación y autorización como OAuth 2.0, OpenID Connect y SAML, facilitando la interoperabilidad con múltiples sistemas y proveedores actuales. También tiene una gran escalabilidad y se adapta de forma rápida a los cambios en la demanda, permitiendo aumentar o disminuir recursos según sea necesario.

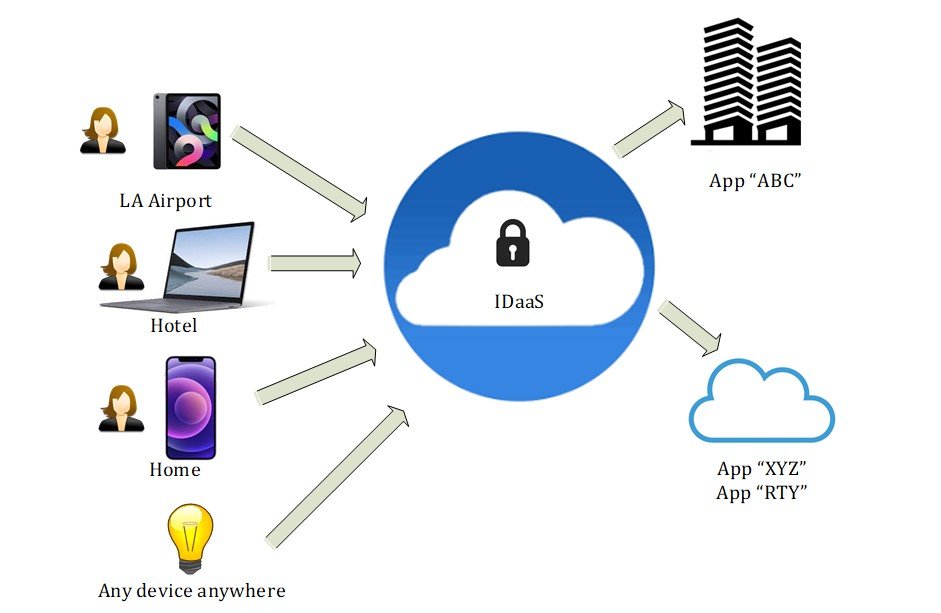


Figura 10. Esquema IDaaS

Este modelo presenta una serie de anomalías:

* **Dependencia del proveedor**: La disponibilidad y la seguridad del servicio dependen en gran medida del proveedor, lo que implica ceder parte del control directo.
* **Cumplimiento y regulación**: Resulta fundamental garantizar que el almacenamiento y procesamiento de datos cumplan con normativas como el GDPR, lo cual puede exigir medidas adicionales de protección de datos.
* **Integración con sistemas heredados**: La vinculación de IDaaS con aplicaciones y sistemas existentes puede requerir un esfuerzo considerable de adaptación y migración.

En definitiva, **IDaaS** representa la evolución natural de la gestión de identidades en un entorno cada vez más marcado por la movilidad, la adopción de la nube y la necesidad de flexibilidad.

## 2.4 Herramientas principales: Keycloak y Vault

### 2.4.1 Keycloak

**Keycloak** es una solución de gestión de identidades y accesos (IAM) de código abierto desarrollada por Red Hat [17]. Ofrece autenticación y autorización para aplicaciones y servicios modernos, incluyendo inicio de sesión único (SSO) que posibilita autenticar al usuario una sola vez para luego acceder a múltiples aplicaciones sin repetir el proceso de autenticación. Además, es compatible con protocolos estándares como OAuth 2.0, OpenID Connect y SAML 2.0, puede integrarse con sistemas LDAP o Active Directory para sincronizar usuarios y grupos, y admite flujos de autenticación personalizados que añaden pasos como MFA o preguntas de seguridad. También brinda soporte para varios idiomas gracias a su función de internacionalización y permite la administración basada en roles (RBAC), con lo cual se definen permisos y roles de manera detallada.

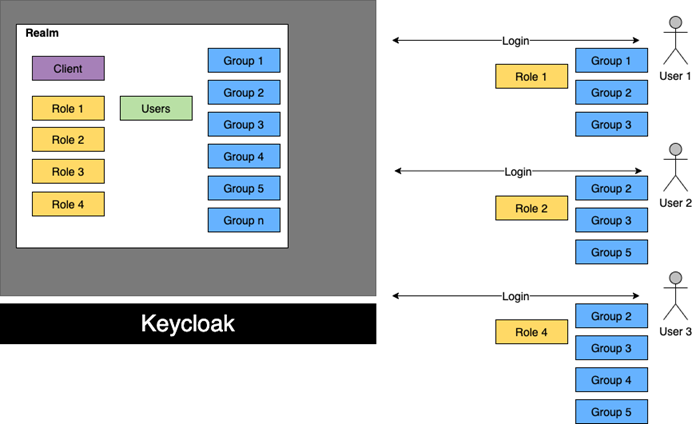


Figura 11. Arquitectura de Keycloak

Keycloak destaca por su extensibilidad a través de SPI (Service Provider Interfaces) [18], que posibilita adaptar la solución con autenticaciones o proveedores de usuarios a medida, y por su flexibilidad de despliegue tanto en entornos locales (on-premise) como en la nube. Al ser de código abierto, cuenta con el respaldo de una comunidad que contribuye con mejoras y parches, y se encarga de ofrecer soporte. Entre sus principales casos de uso se incluyen la autenticación en aplicaciones web y móviles, la protección de microservicios y endpoints de API con tokens de acceso y roles de autorización, así como la federación de identidades mediante la integración de diferentes fuentes de usuarios y proveedores externos de identidad.

### 2.4.2 Vault

**Vault** es una herramienta de código abierto creada por HashiCorp [19] para administrar secretos y proteger datos sensibles. Se centra en el control de acceso y el almacenamiento seguro de información como claves API, contraseñas y certificados, utilizando cifrado avanzado para mantenerlos a salvo. Además de definir políticas detalladas que restringen quién puede acceder a cada secreto y bajo qué condiciones, también permite generar credenciales dinámicas para bases de datos y servicios en la nube con tiempos de vida limitados, lo que fortalece la seguridad y reduce riesgos. Su sistema de auditoría registra todas las operaciones, facilitando la identificación de actividades sospechosas y el cumplimiento de normativas como PCI DSS, HIPAA y GDPR. Vault es compatible con Kubernetes, Docker y sistemas de CI/CD, ofreciendo una integración sencilla en entornos modernos.

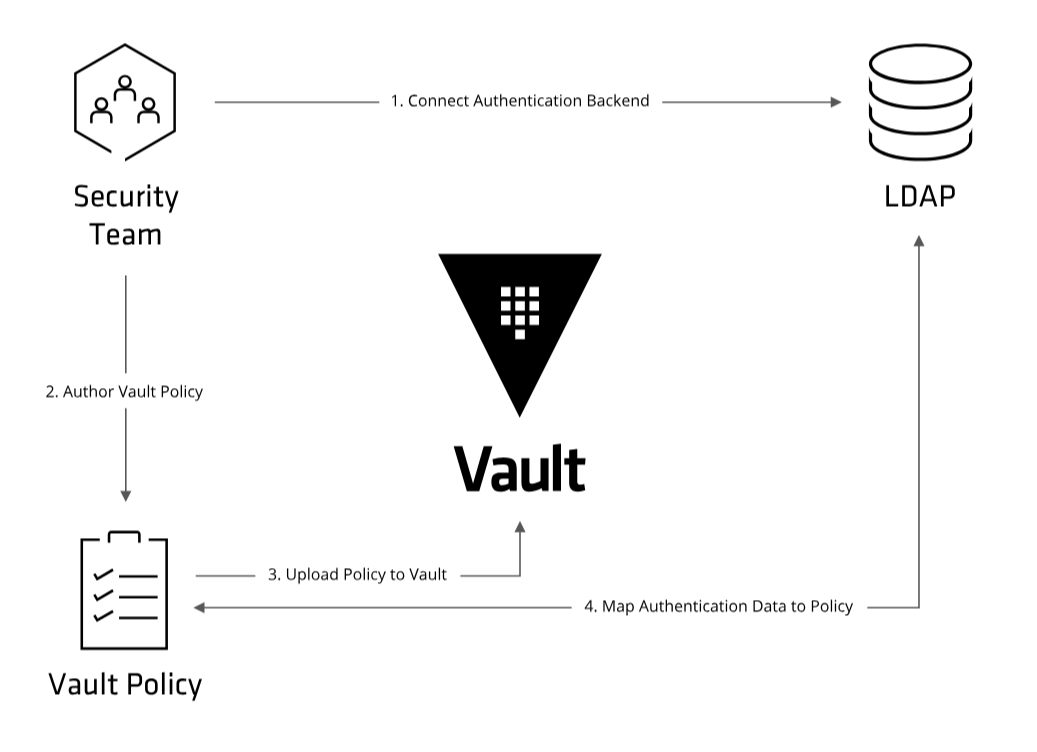


Figura 12. Arquitectura de Vault

En proyectos reales, puede actuar como un almacén centralizado de secretos para arquitecturas de microservicios [20], evitando que estos datos se incluyan en el código o en archivos de configuración. También automatiza la rotación de claves y contraseñas para minimizar el riesgo ante posibles filtraciones, y proporciona APIs de “encriptación como servicio” que permiten cifrar y descifrar datos sin exponer las claves directamente a las aplicaciones. En definitiva, ofrece seguridad centralizada, facilidad de cumplimiento normativo y la flexibilidad necesaria para adaptarse a diversos entornos y tecnologías, tanto on-premise como en la nube.

## 2.5 Contenedores, orquestación y DevOps

### 2.5.1 Contenedores

Los **contenedores** son una forma de virtualización a nivel de sistema operativo que aíslan las aplicaciones en entornos independientes [21]. **Docker**, como una de las plataformas de contenedorización más usadas, permite que los contenedores compartan el kernel del sistema anfitrión, lo cual mejora la eficiencia de recursos en comparación con las máquinas virtuales tradicionales que requieren un sistema operativo completo por instancia [22].

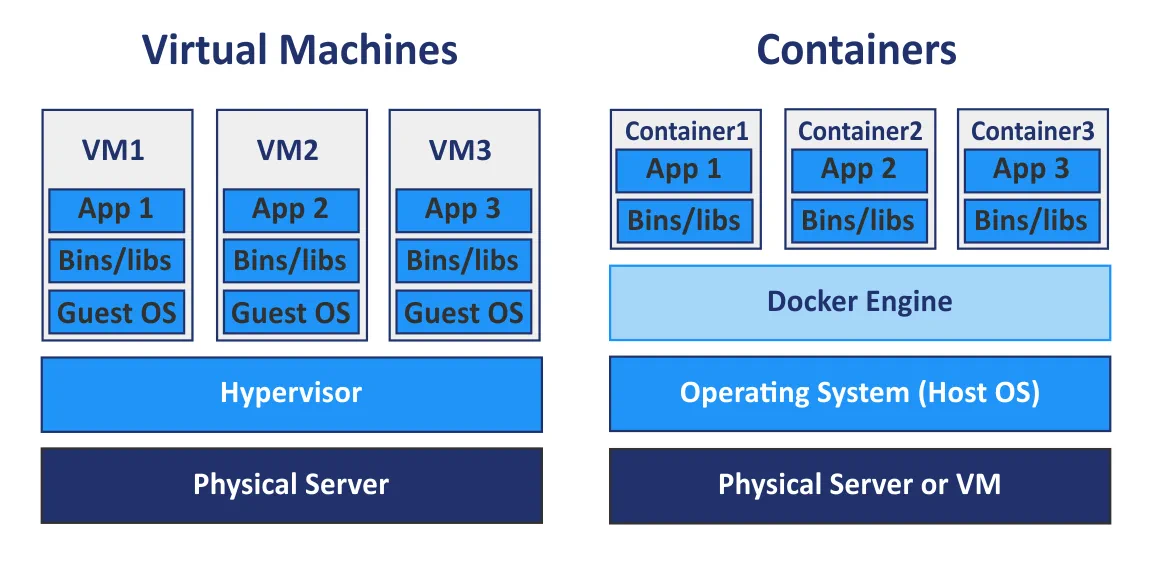


Figura 13. Comparativa máquinas virtuales y contenedores

Esta tecnología también proporciona portabilidad, garantizando que los contenedores se ejecuten de forma consistente en diferentes entornos de desarrollo, pruebas y producción, y agiliza el arranque, lo que facilita el escalado dinámico. Además, fomenta la **modularidad**, ya que las aplicaciones pueden dividirse en componentes más pequeños (microservicios) que pueden desarrollarse y desplegarse de forma independiente.

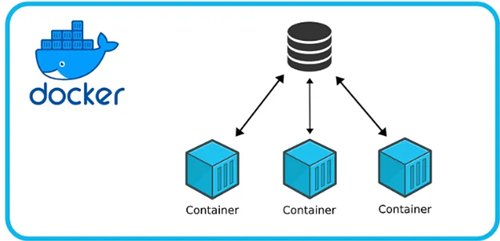


Figura 14. Modularidad Docker

No obstante, los contenedores no ofrecen el mismo nivel de aislamiento que las máquinas virtuales. Si se produce una vulnerabilidad en el kernel, todos los contenedores pueden verse afectados. También existe el riesgo de usar **imágenes inseguras**, las cuales pueden contener fallos o configuraciones deficientes [23]. Asimismo, la gestión de secretos (por ejemplo, claves o contraseñas) dentro de las imágenes o de variables de entorno plantea riesgos de exposición de datos sensibles.

### 2.5.2 Orquestación con Kubernetes

**Kubernetes** es una plataforma de orquestación de contenedores que automatiza el despliegue, escalado y gestión de aplicaciones en contenedores [24]. Se encarga de administrar los recursos del clúster, implementa y escala las aplicaciones según la demanda y reemplaza o reinicia los contenedores que no responden, proporcionando un mecanismo de autorreparación. Además, ofrece descubrimiento de servicios y balanceo de carga para distribuir el tráfico de manera eficiente, y maneja configuraciones y almacenamiento de datos de forma persistente.

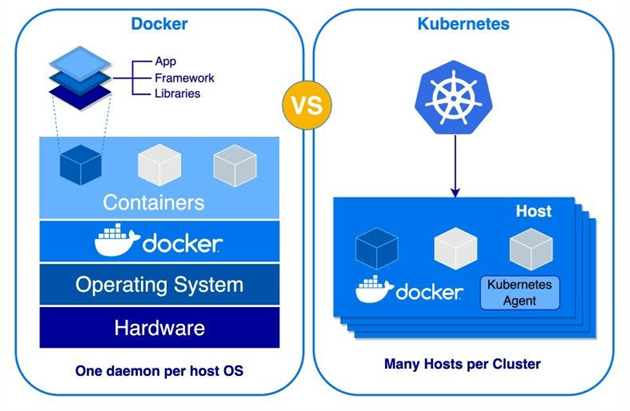


Figura 15. Docker y Kubernetes

Para automatizar despliegues de sistemas seguros, Kubernetes permite definir políticas de seguridad a nivel de contenedor y pod, restringiendo capacidades y accesos [25]. Puede integrarse con herramientas como Vault para gestionar secretos de forma segura, y facilita la aplicación de actualizaciones y parches con interrupciones mínimas. Entre las buenas prácticas de seguridad se encuentran el uso de Control de Acceso Basado en Roles (RBAC) para restringir las acciones de usuarios y servicios, el uso de namespaces para aislar recursos y aplicaciones, y la configuración de políticas de red para regular el tráfico entre pods y servicios

### 2.5.3 DevOps

**DevOps** es una metodología que integra las prácticas de desarrollo de software (Dev) y operaciones de TI (Ops) para reducir el ciclo de vida de desarrollo y ofrecer entregas continuas de alta calidad [26]. Su base está en la colaboración y la comunicación, haciendo que desarrolladores y operaciones trabajen juntos desde el inicio y que se automatice e integre gran parte del proceso. Dentro de sus pilares se encuentra la **integración continua (CI)**, que consiste en automatizar la inserción de cambios en el código para detectar errores con rapidez, y la **entrega continua (CD)**, que agiliza y hace confiable el despliegue de aplicaciones en entornos de producción. También promueve la **infraestructura como código (IaC)**, un enfoque que gestiona y aprovisiona la infraestructura mediante definiciones en código en lugar de configuraciones manuales.

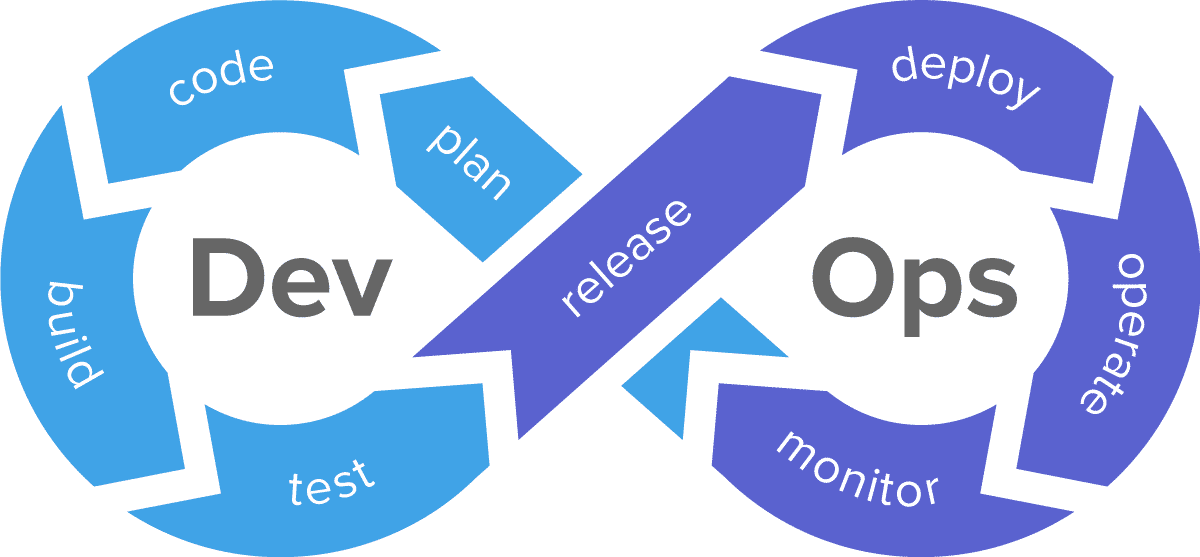


Figura 16. Arquitectura DevOps

Los beneficios de DevOps incluyen una mayor velocidad de entrega de soluciones, una mejora continua en la calidad del software gracias a la automatización de pruebas y la posibilidad de escalar las aplicaciones de forma eficiente al automatizar tareas repetitivas, lo que libera tiempo para actividades de mayor valor. Para implementar DevOps, suelen usarse **sistemas de control de versiones** (Git, SVN), **herramientas de CI/CD** (Jenkins, GitLab CI/CD, CircleCI), **herramientas de gestión de configuración** (Ansible, Puppet, Chef) y **soluciones de monitoreo y registro** (Prometheus, Grafana, ELK Stack). Sin embargo, su adopción plantea desafíos como la necesidad de un cambio cultural, la integración de la seguridad en despliegues rápidos y la complejidad a la hora de elegir y configurar las herramientas más adecuadas.

### 2.5.4 DevSecOps

**DevSecOps** es la evolución de DevOps que integra las prácticas de seguridad a lo largo de todo el ciclo de vida de desarrollo de software [27]. Reconoce que la seguridad es responsabilidad de todos y no solo del equipo especializado en esta área, por lo que promueve la integración temprana de controles y pruebas de seguridad desde las fases iniciales de desarrollo, así como la automatización de análisis y despliegues. También impulsa la colaboración entre desarrolladores, operaciones y equipos de seguridad, y la capacidad de responder de manera ágil ante vulnerabilidades e incidentes.

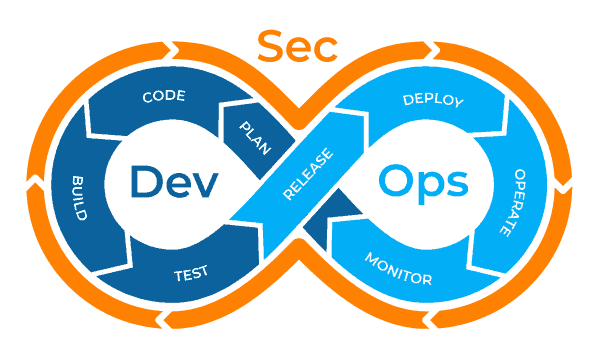


Figura 17. Arquitectura DevSecOps

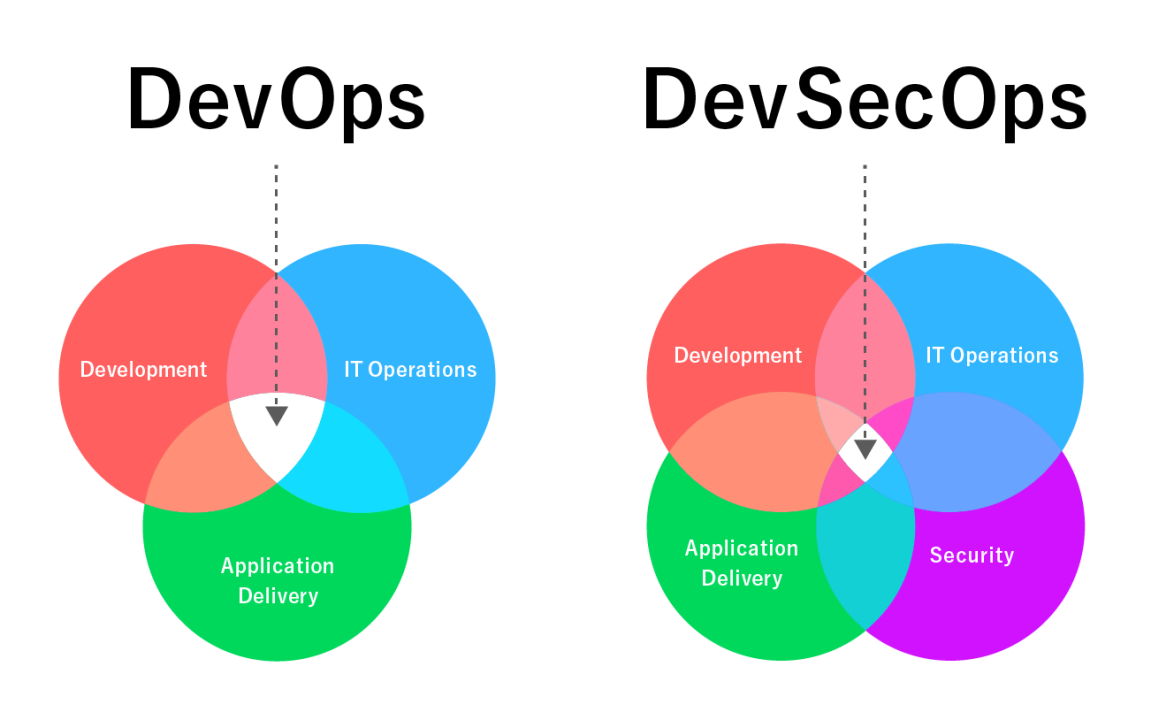


Figura 18. Comparativa DevOps y DevSecOps

Sus principales beneficios incluyen la identificación temprana de riesgos, lo que minimiza el costo y el impacto de las correcciones, y el cumplimiento continuo de normativas gracias a la automatización de auditorías y reportes. Además, fortalece la calidad del software al aplicar buenas prácticas de codificación segura y revisiones constantes del código. Entre las herramientas y prácticas más comunes se encuentran el análisis de código estático (SAST) para detectar vulnerabilidades en el código fuente, el análisis de código dinámico (DAST) que evalúa las aplicaciones en tiempo de ejecución y la gestión de dependencias que examina bibliotecas y componentes de terceros en busca de fallos conocidos [28].

## 2.7 Modelos de cloud computing

Los modelos de cloud computing, ofrecen diferentes niveles de abstracción y servicios que permiten a las organizaciones adoptar soluciones flexibles y escalables, ajustándose a sus exigencias tecnológicas y de negocio. Además de brindar la posibilidad de desplegar recursos de forma más rápida y con un menor coste inicial, estos modelos facilitan la administración y actualización continua de la infraestructura. De acuerdo con la definición del Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST, por sus siglas en inglés) [29], los principales modelos de servicio en cloud computing son Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS). Cada uno de ellos implica un grado diferente de control y responsabilidad sobre la infraestructura, así como la delegación de tareas de administración y mantenimiento en el proveedor de servicios, lo que ofrece a las empresas la flexibilidad necesaria para centrar sus esfuerzos en la innovación y el desarrollo de soluciones de valor.

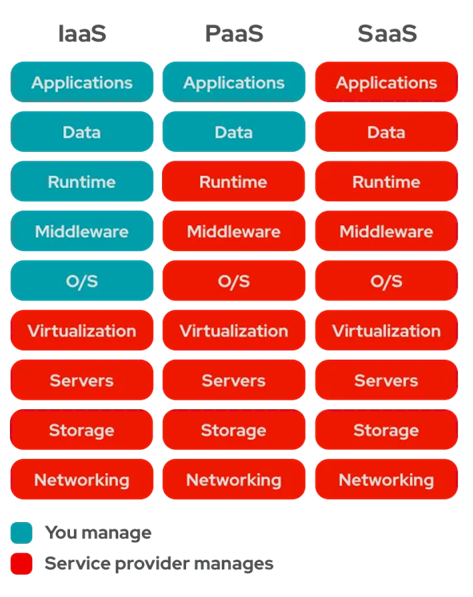


Figura 19. Modelos cloud computing

Estos tres modelos pueden coexistir dentro de una estrategia de nube híbrida o múltiple, donde cada servicio se emplea para tareas específicas de acuerdo con las demandas del negocio. Sin embargo, la adopción de la nube implica contemplar aspectos de seguridad, gobernanza y disponibilidad, así como evaluar las necesidades particulares de cada organización. Una planeación adecuada y la supervisión constante de la infraestructura son esenciales para maximizar los beneficios de los modelos de cloud computing [30].

### 2.7.1 Infraestructura como Servicio (IaaS)

En este modelo, el proveedor de servicios proporciona recursos de computación virtualizados, tales como máquinas virtuales, capacidad de procesamiento, redes y almacenamiento. El cliente puede instalar sistemas operativos, software y herramientas según sus necesidades, administrando todo lo relacionado con la configuración de la infraestructura y el despliegue de aplicaciones. Algunos ejemplos de herramientas conocidas son Amazon Elastic Compute Cloud (EC2), Microsoft Azure Virtual Machines y Google Compute Engine [31]. IaaS aporta un alto grado de flexibilidad y control, pero requiere mayor responsabilidad en la gestión y seguridad por parte del usuario.

### 2.7.2 Plataforma como Servicio (PaaS)

Ofrece un entorno completo de desarrollo y despliegue de aplicaciones, proporcionando no solo la infraestructura subyacente, sino también el sistema operativo, el middleware y servicios de base de datos, entre otros componentes. Esto permite a los desarrolladores concentrarse en la creación de aplicaciones sin tener que preocuparse por la gestión de servidores o la configuración de sistemas. Algunos ejemplos pueden ser Google App Engine, Heroku y Microsoft Azure App Service [32]. Dado que el proveedor gestiona la mayor parte de la infraestructura y las actualizaciones, las empresas pueden agilizar los ciclos de desarrollo y reducir costos de mantenimiento.

### 2.7.3 Software como Servicio (SaaS)

Se trata de la entrega de aplicaciones listas para usar a través de Internet, donde el proveedor se encarga de la infraestructura, la plataforma y la propia aplicación. El usuario simplemente accede al software mediante un navegador o interfaz cliente, pagando por su uso o mediante suscripciones. Soluciones como Salesforce, Microsoft Office 365 y Google Workspace permiten la implementación de este modelo [33]. SaaS elimina la necesidad de instalar, mantener o actualizar el software de forma local, lo que reduce significativamente la complejidad técnica y los costos de soporte en las organizaciones.

## 2.8 Infraestructura como código (IaC)

La Infraestructura como Código (IaC) consiste en describir y gestionar los recursos de TI (servidores, redes y almacenamiento, entre otros) mediante archivos de configuración legibles por los usuarios, los cuales se almacenan en sistemas de control de versiones como Git, posibilitando un control y seguimiento similar al del desarrollo de software [34]. Gracias a esta práctica, se facilita la revisión por pares, la colaboración entre los distintos equipos de desarrollo y operaciones, y la trazabilidad de los cambios, contribuyendo así a una cultura ágil dentro de las organizaciones.

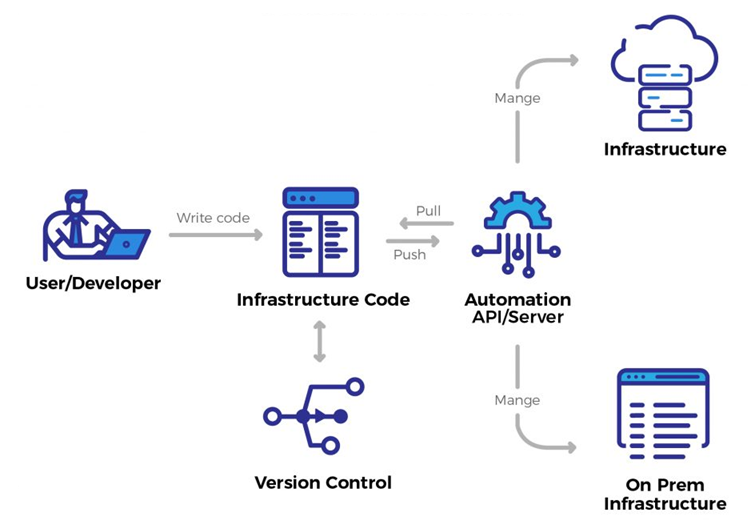


Figura 20. Esquema flujo IaC

Una característica a destacar de la IaC es la inmutabilidad de los entornos. Si se precisa modificar la infraestructura, en vez de realizar cambios directos se recrean los servidores y recursos con la nueva configuración, lo que minimiza el riesgo de inconsistencias o de “deriva de configuración”. Asimismo, se promueve la idempotencia, es decir, la capacidad de obtener siempre el mismo resultado al aplicar varias veces la misma configuración, lo que reduce la complejidad y la probabilidad de error [35].

### 2.8.1 **Terraform**

Terraform [36] es una herramienta que facilita la definición y administración de recursos en múltiples proveedores de nube (AWS, Azure, Google Cloud, etc.) mediante un enfoque declarativo, donde el usuario especifica la infraestructura y configuración específicas para que el sistema se encargue de crearlo o actualizarlo.

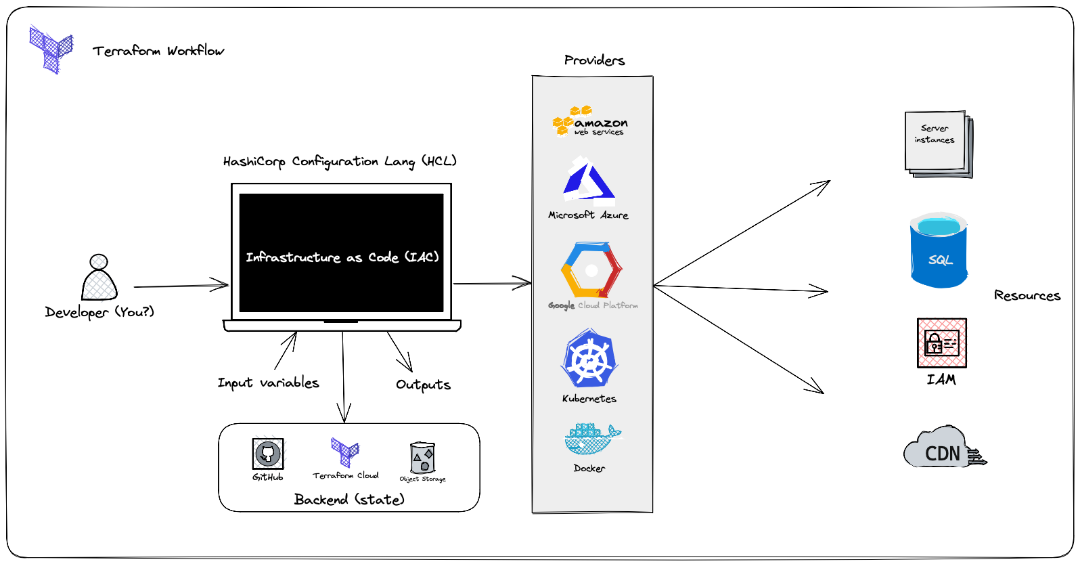


Figura 21. Flujo de trabajo Terraform

### 2.8.2 Pulumi

Pulumi [37] adopta un método distinto, al posibilitar describir la infraestructura con lenguajes de programación de uso general (como TypeScript, Python o Go), lo que facilita la integración con librerías y herramientas de software convencionales.

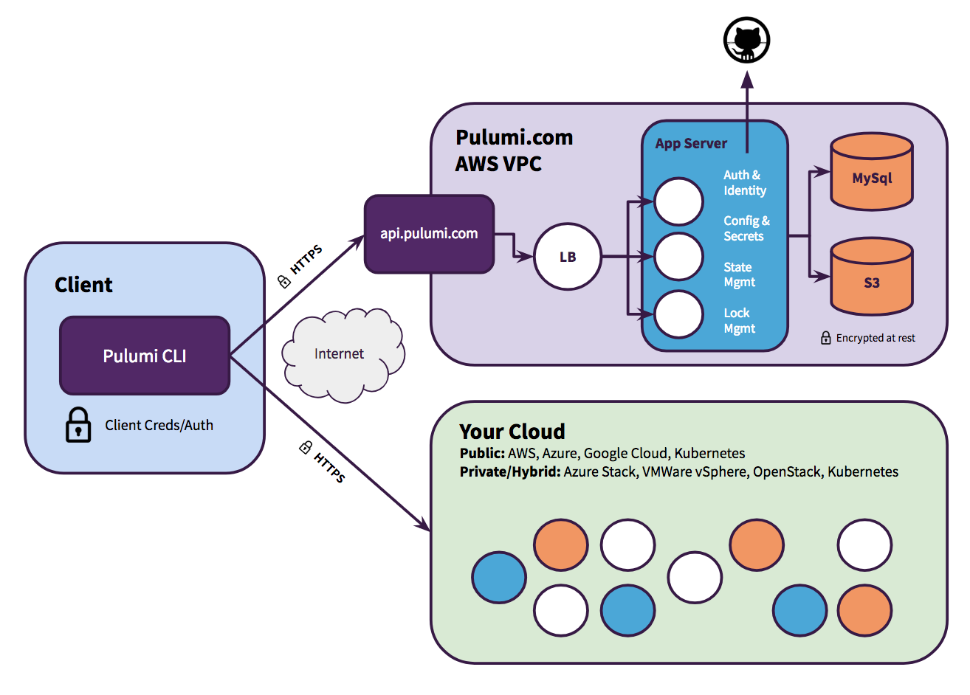


Figura 22. Pulumi

### 2.8.3 Ansible

Ansible [38] se caracteriza por ser una solución *agentless*, es decir, no requiere la instalación de agentes en las máquinas que va a administrar. Emplea *Playbooks* escritos en YAML, donde se describen las tareas necesarias para configurar los sistemas y desplegar aplicaciones. Gracias a este enfoque, es posible manejar servidores de manera remota haciendo uso de protocolos de conexión estándar (como SSH), lo cual simplifica la gestión y reduce la complejidad de instalación.

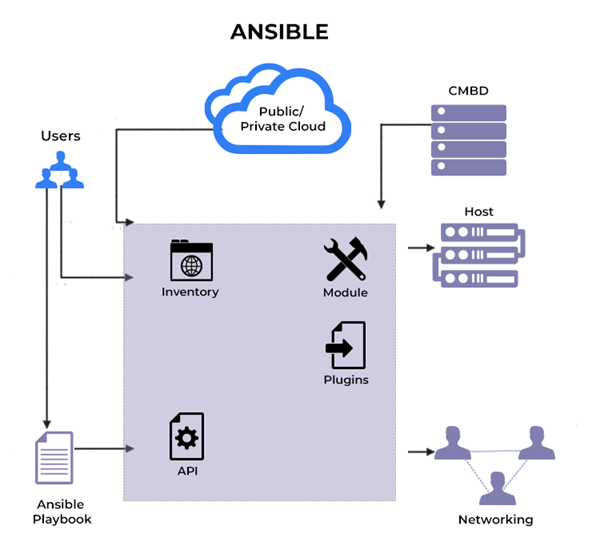


Figura 23. Ansible

Ansible resulta especialmente útil en entornos dinámicos y en infraestructuras de múltiples nubes o entornos híbridos, ya que su curva de aprendizaje es relativamente baja y permite automatizar procesos de forma ágil.

### 2.8.4 Chef

Chef [39] opera habitualmente en un modelo cliente-servidor, donde los nodos (o máquinas gestionadas) se comunican con un servidor Chef para recibir las configuraciones deseadas. Dichas configuraciones se describen en *cookbooks* y *recipes* escritos en un DSL (lenguaje específico de dominio) basado en Ruby. El enfoque de Chef se centra en definir el estado objetivo de cada sistema, permitiendo realizar cambios de manera automática y consistente.

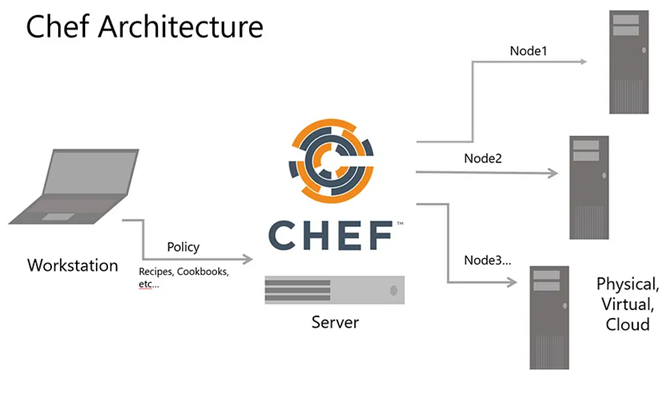


Figura 24. Arquitectura Chef

Aunque requiere un conocimiento más profundo de la infraestructura y del lenguaje Ruby, Chef ofrece gran flexibilidad y un robusto ecosistema de *cookbooks* creados por la comunidad, lo que facilita la integración con múltiples plataformas y servicios.

### 2.8.5 Puppet

Al igual que Chef, Puppet [40] funciona mayormente con una arquitectura cliente-servidor y se vale de *manifiestos* para describir el estado deseado de cada nodo. Este lenguaje declarativo propio de Puppet indica cómo debe configurarse el sistema, y el agente de Puppet se encarga de evaluar y aplicar los cambios necesarios para alcanzar ese estado.

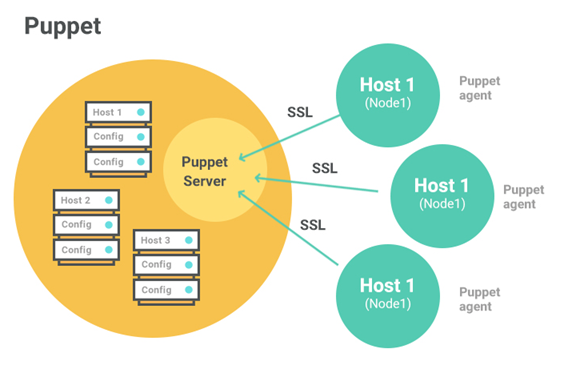


Figura 25. Puppet

Puppet destaca por ofrecer un modelo de configuración idempotente: aplicar el mismo *manifiesto* varias veces no debería alterar el estado final del sistema. Además, cuenta con una amplia comunidad y un extenso catálogo de módulos predefinidos para diferentes servicios y aplicaciones.

## 2.9 Resumen del capítulo

En este capítulo se estudia cómo han evolucionado las diferentes estrategias de seguridad en aplicaciones y qué tecnologías han ido implementándose con el paso del tiempo, para reforzar las vulnerabiliaddes que han ido surgiendo. En primer lugar, se repasan los métodos tradicionales de autenticación y autorización, como los sistemas basados en contraseñas, LDAP y SAML, junto a la autenticación multifactor (MFA). A pesar de su solidez en cuanto a la gestión de identidades, presentan limitaciones importantes en términos de escalabilidad, usabilidad y seguridad.

Posteriormente aparecieron tecnologías como OAuth 2.0, OpenID Connect (OIDC) y JSON Web Tokens (JWT), las cuales permiten una gestión más ágil de cara al usuario y al administrador del sistema, ya que se unifican los procesos de autenticación y autorización. Como resultado de ello las credenciales de los usuarios de las distintas aplicaciones queden menos expuestas y por lo tanto menos vulnerables, reforzándose así su seguridad. Tambén aparecen soluciones que adoptan una nueva arquitectura, una de ellas es conocida como IDaaS (Identity as a Service), donde gestión de identidades queda externalizada del entorno aplicativo, aunque esto deja posibles riesgos en manos de terceros de cara a la normativa vigente relacionada con la seguridad de la información y cumplimiento normativo.

Herramientas actuales y en las que se basa el actual trabajo son Keycloak, diseñada y utilizada para la administración centralizada de identidades y accesos, y Vault, que se centra en proteger y gestionar secrets (contraseñas, claves API, etc.) así como otros datos sensibles. Ambas herramientas ayudan a cualquier aplicación ya sea que disponga una arquitectura monolítica o basadas en microservicios y contenedores, incluyéndose como una capa externa de seguridad.

Finalmente, se introducen metodologías de desarrollo como son DevOps y DevSecOps, que integran la seguridad a lo largo de todo el ciclo de vida del software, además de emplear entornos basados en contenedores orquestados con Kubernetes. En conjunto con la Infraestructura como Código (IaC) haciendo uso de herramientas como Terraform, Ansible, Chef o Puppet permiten desplegar sistemas escalables y seguros, optimizando el rendimiento y garantizando el cumplimiento de estándares y normativas de protección de datos.

# Capítulo 03: Herramientas y tecnologías

En este capítulo se describen las herramientas y tecnologías empleadas en el proyecto, ofreciendo una visión detallada de cada componente y de cómo se integran dentro del sistema global. Entre las tecnologías clave se encuentran **Docker**, utilizado para la contenedorización; **Keycloak**, para la gestión de identidad y acceso; **Vault**, para la gestión de secretos, y **Terraform**, que permite el despliegue automatizado de la infraestructura.

## 3.1 Virtualización con Docker

**Docker** es una plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portátiles, lo que agiliza su despliegue y escalabilidad. Al aislar las aplicaciones en contenedores, se obtiene un entorno homogéneo y coherente que se mantiene tanto en la fase de desarrollo como en la de producción, evitando diferencias debidas a configuraciones del sistema o a variaciones en bibliotecas. Las principales características de Docker son las siguientes:

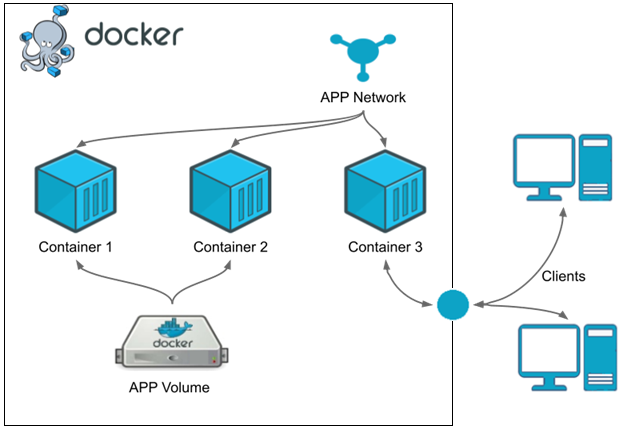


Figura 26. Virtualización con Docker

* **Portabilidad:** Los contenedores pueden ejecutarse en cualquier sistema que tenga Docker instalado, sin importar las diferencias en la configuración o en el sistema operativo subyacente.
* **Eficiencia de recursos:** Dado que comparten el kernel del sistema anfitrión, los contenedores son más ligeros que las máquinas virtuales, reduciendo el consumo de recursos de hardware.
* **Aislamiento:** Cada contenedor posee su propio sistema de archivos, procesos y red, lo que evita interferencias entre aplicaciones.
* **Rapidez de despliegue:** Los contenedores pueden iniciarse en segundos, lo que acelera significativamente los procesos de desarrollo, prueba y puesta en producción.

La creación de imágenes en Docker se fundamenta en la idea de contar con plantillas reutilizables que incluyan tanto la aplicación como todas las dependencias necesarias para su ejecución. El proceso suele comenzar con una imagen base, que puede ser oficial o de confianza, y sobre la cual se añaden las capas que conforman el entorno requerido por la aplicación.

Para llevar a cabo esta construcción se emplea el Dockerfile, un archivo de texto que especifica, mediante una serie de instrucciones, cómo configurar y preparar ese entorno dentro del contenedor. Por ejemplo, se puede establecer la versión del lenguaje a utilizar, instalar dependencias de sistema, copiar el código fuente y definir el comando de inicio.



Figura 27. Comandos Docker

El comando “docker build” es el encargado de traducir esas instrucciones a capas de imagen, generando una plantilla final que se puede distribuir y desplegar en cualquier sistema que tenga Docker instalado. Cada capa se cachea para optimizar las construcciones posteriores y agilizar los procesos de integración continua. De esta forma, se logra una estandarización del entorno de ejecución de la aplicación, lo que facilita la portabilidad y reduce los errores originados por diferencias en la configuración del servidor.

En Docker, las redes virtuales permiten comunicar contenedores entre sí y con el exterior, garantizando un mayor control sobre la topología de red y el tráfico de datos. La configuración de red por defecto se basa en la creación de una red bridge, donde cada contenedor recibe una dirección IP interna y puede comunicarse con otros contenedores que pertenezcan a la misma red. Sin embargo, Docker ofrece otras opciones de configuración, como la red host, donde el contenedor comparte la pila de red del sistema anfitrión, o las redes overlay, que facilitan la comunicación entre contenedores en distintos servidores, lo cual resulta esencial en entornos distribuidos u orquestados, como Docker Swarm o Kubernetes.

Para la gestión de redes, Docker proporciona comandos que permiten crear y administrar redes personalizadas, asignar contenedores a diferentes redes y definir políticas de acceso. De esta manera, cada servicio puede colocarse en la red que mejor se ajuste a sus requisitos de seguridad y conectividad, controlando la exposición de los puertos al exterior y limitando el alcance de cada contenedor.

El uso de contenedores implica que, por defecto, todo lo que se almacena dentro de un contenedor se pierde cuando este se elimina o se detiene. Para solventar esta limitación, Docker implementa el concepto de volúmenes, que permiten que los datos existan más allá del ciclo de vida del contenedor y sean accesibles para otros contenedores o servicios. Existen dos enfoques principales: por un lado, están los volúmenes administrados por Docker, donde la plataforma gestiona la ubicación y el acceso a los datos. Por otro lado, se encuentran los bind mounts, que vinculan un directorio o archivo del sistema anfitrión con el contenedor, permitiendo compartir y editar ficheros de manera sencilla.

Este mecanismo de persistencia es crucial para bases de datos y servicios que requieren conservar información entre reinicios, actualizaciones o despliegues. Además, facilita el respaldo y la migración de los datos, dado que los volúmenes pueden copiarse o trasladarse de forma independiente al ciclo de vida de los contenedores. Asimismo, diversos contenedores pueden acceder al mismo volumen, posibilitando la colaboración entre servicios, la compartición de archivos y la centralización de ciertos recursos. De esta forma, Docker no solo simplifica la ejecución de aplicaciones, sino también el almacenamiento de la información que dichas aplicaciones necesitan.

## 3.2 Keycloak como servicio

**Keycloak** es una solución de código abierto para la gestión de identidad y acceso (IAM) que ofrece autenticación y autorización centralizadas a aplicaciones y servicios modernos. Su objetivo principal es unificar la gestión de usuarios, roles y permisos, de modo que distintas aplicaciones o entornos puedan integrar sus procesos de autenticación sin necesidad de manejar sus propias credenciales de forma aislada. Para lograrlo, se hace uso de conceptos como **realms**, que funcionan como espacios de seguridad independientes donde se configuran usuarios y credenciales, así como **roles** y mapeos de usuarios, los cuales permiten asignar privilegios o permisos de manera precisa.

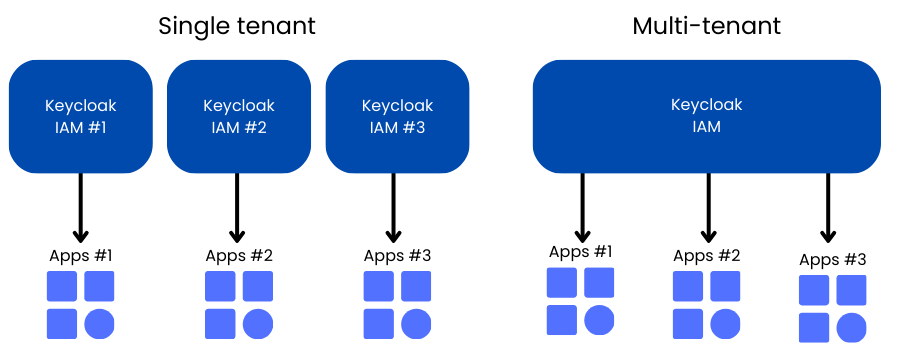


Figura 28. Arquitectura de aplicaciones con Keycloak

Al mismo tiempo, se puede integrar con fuentes externas como LDAP o bases de datos, centralizando así la autenticación y la gestión de atributos de cada usuario.

La instalación y configuración inicial de Keycloak puede llevarse a cabo de distintas formas. Es posible instalarlo manualmente en un servidor de aplicaciones Java o, de manera más sencilla y portable, desplegarlo dentro de un contenedor Docker. Al iniciarse por primera vez, se crea un usuario administrador encargado de configurar el sistema y de definir las aplicaciones (clientes) que necesitarán autenticación, ya sea mediante OAuth2, OpenID Connect u otros protocolos compatibles. Cada cliente se registra con información como el tipo de aplicación o las direcciones de redirección, y puede utilizar distintos flujos de autenticación según las necesidades (aplicaciones web, móviles o de página única).

Para asegurar una integración fluida con las aplicaciones, Keycloak proporciona adaptadores y bibliotecas que simplifican la implementación de autenticación y autorización. Esto facilita que los desarrolladores se centren en la lógica de negocio en lugar de tener que desarrollar soluciones de seguridad desde cero. Asimismo, el servidor de Keycloak, una vez en marcha, puede administrar todos los inicios de sesión y cierres de sesión, mantener la sesión del usuario y asignar o revocar roles y permisos de forma centralizada.

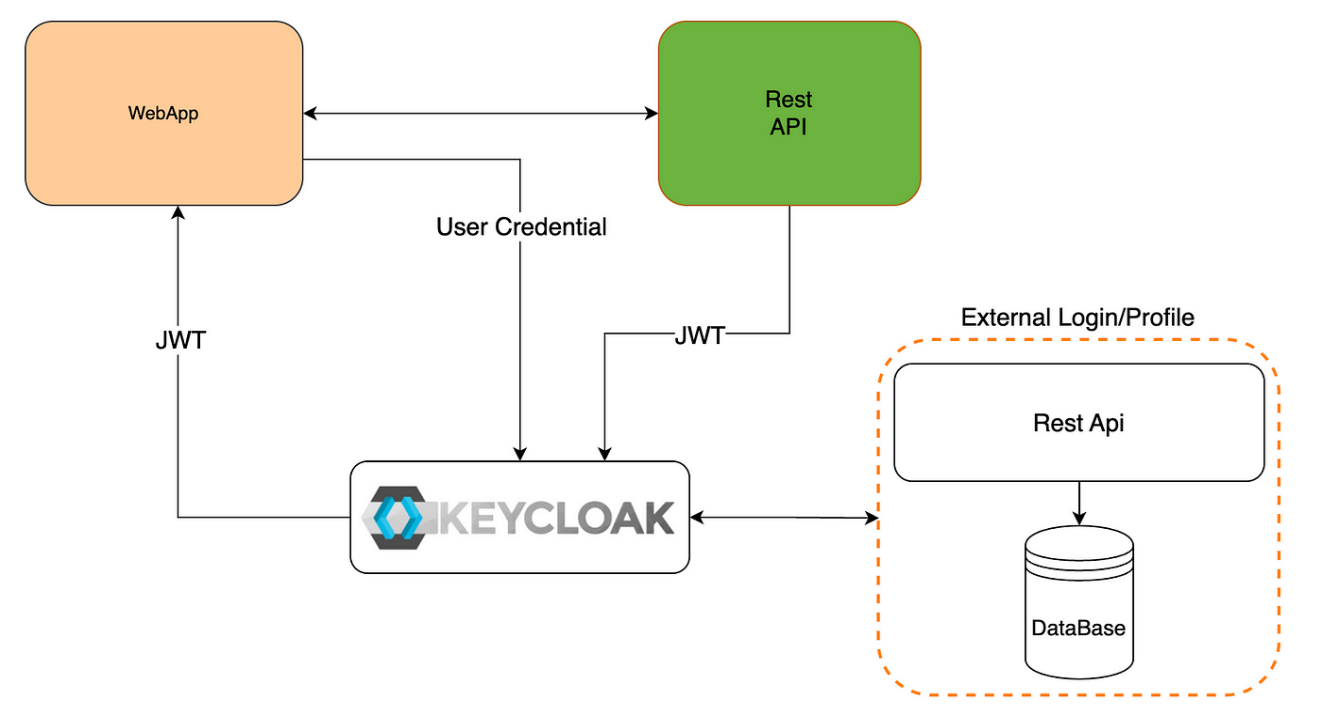


Figura 29. Keycloak como servicio

Al ejecutar **Keycloak como un servicio Docker**, se obtienen beneficios adicionales en términos de portabilidad, facilidad de despliegue y escalabilidad. El proceso de instalación se reduce a arrancar un contenedor con la imagen oficial de Keycloak y configurar ciertas variables de entorno para especificar aspectos como el usuario administrador, la contraseña y los puertos de red. Esto agiliza enormemente la puesta en marcha, ya que no es necesario instalar ni mantener un servidor de aplicaciones independiente. Además, se puede replicar o escalar el servicio con solo lanzar instancias adicionales del contenedor, ajustándose así a las fluctuaciones de carga y garantizando alta disponibilidad. El aislamiento proporcionado por Docker también minimiza conflictos con otras aplicaciones que se ejecuten en el mismo servidor, dado que cada contenedor opera en su propio entorno. Si se necesita integrar Keycloak en un flujo de orquestación más complejo, puede funcionar sin problemas con Kubernetes u otras herramientas de administración de contenedores, conservando la misma experiencia de instalación y uso en cualquier plataforma compatible con Docker.

## 3.3 Vault como servicio

**Vault** es una herramienta de código abierto enfocada en la gestión de secretos, diseñada para controlar de manera segura y centralizada el acceso a tokens, contraseñas, certificados y claves de cifrado. Su funcionamiento gira en torno a varios conceptos clave.

En primer lugar, maneja **“secrets”**, que pueden ser valores estáticos (almacenados de forma cifrada para que solo entidades autorizadas tengan acceso) o secretos dinámicos, generados temporalmente para servicios y aplicaciones con el fin de limitar el tiempo durante el cual las credenciales resultan válidas. Otro pilar fundamental son las **políticas de acceso**, que establecen qué operaciones pueden llevar a cabo usuarios y aplicaciones, permitiendo un control muy granular sobre los permisos. Además, Vault se integra con múltiples **métodos de autenticación** (como tokens, AppRole o LDAP), de modo que, cuando las entidades inician sesión, se les asignan las políticas que determinan qué acciones pueden o no realizar.

Para almacenar y proteger la información de manera persistente, Vault utiliza diferentes backends, como Consul, etcd o archivos locales con su configuración. Una vez configurado el backend, la herramienta debe iniciarse mediante un proceso de inicialización, en el que se generan las claves maestras y un token raíz. Vault permanece “sellado” hasta que dichas claves se proporcionan nuevamente, un mecanismo que blinda el acceso a los datos cifrados y evita la filtración de secretos. Un aspecto especialmente destacado es la posibilidad de **gestionar credenciales dinámicas**, ya que las aplicaciones pueden solicitar credenciales temporales solo cuando las necesitan, reduciendo así la ventana de exposición a brechas de seguridad.

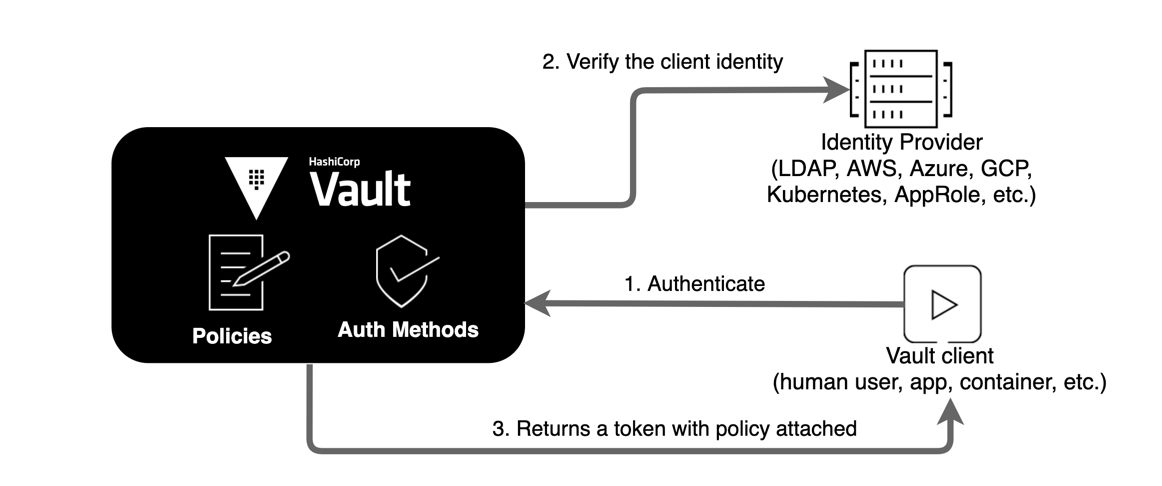


Figura 30. Vault como servicio

La ejecución de **Vault como un servicio Docker** ofrece numerosas ventajas. El despliegue se simplifica, puesto que basta con lanzar el contenedor con la imagen oficial de Vault y configurar las variables necesarias para indicar la configuración del backend de almacenamiento, la ruta de sellado y otros ajustes. Esto acorta drásticamente el tiempo de instalación y reduce los pasos manuales, aprovechando la portabilidad inherente a los contenedores. Además, el aislamiento que proporciona Docker minimiza los posibles conflictos con otras aplicaciones, ya que cada contenedor funciona de manera independiente. De esta forma, se combina la robustez y versatilidad de Vault con la facilidad de despliegue y la escalabilidad que ofrece la tecnología de contenedores, resultando en una solución más ágil y segura para la gestión de secretos en diferentes entornos y escenarios de producción.

## 3.4 Terraform

**Terraform** es una herramienta de infraestructura como código que permite definir, aprovisionar y gestionar la infraestructura de forma eficiente y reproducible mediante archivos de configuración declarativos. Con este enfoque, se especifica el estado deseado de la infraestructura y Terraform se encarga de aplicar los cambios necesarios para alcanzarlo, lo que brinda claridad y reduce errores al automatizar tareas que antes se realizaban de manera manual o con múltiples herramientas. Además, su soporte para diferentes proveedores y servicios lo convierte en una opción versátil para entornos locales y en la nube.

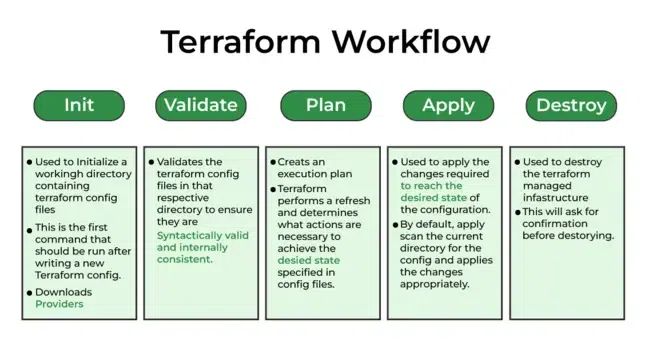


Figura 31. Comandos Terraform

Aunque Terraform es frecuentemente asociado con la gestión de proveedores como AWS, Azure o GCP, también resulta útil para manejar recursos locales. Por ejemplo, puede utilizarse para orquestar contenedores Docker, redes y volúmenes requeridos para servicios como Keycloak y Vault en un entorno local. Esta capacidad de crear, actualizar y destruir recursos de forma controlada facilita el mantenimiento y la evolución de los sistemas, asegurando coherencia y repetibilidad en cada despliegue. Otra funcionalidad destacable es la etapa de planificación previa, donde es posible visualizar los cambios antes de aplicarlos, lo que minimiza el riesgo de errores y otorga mayor confianza a los equipos de desarrollo y operaciones.

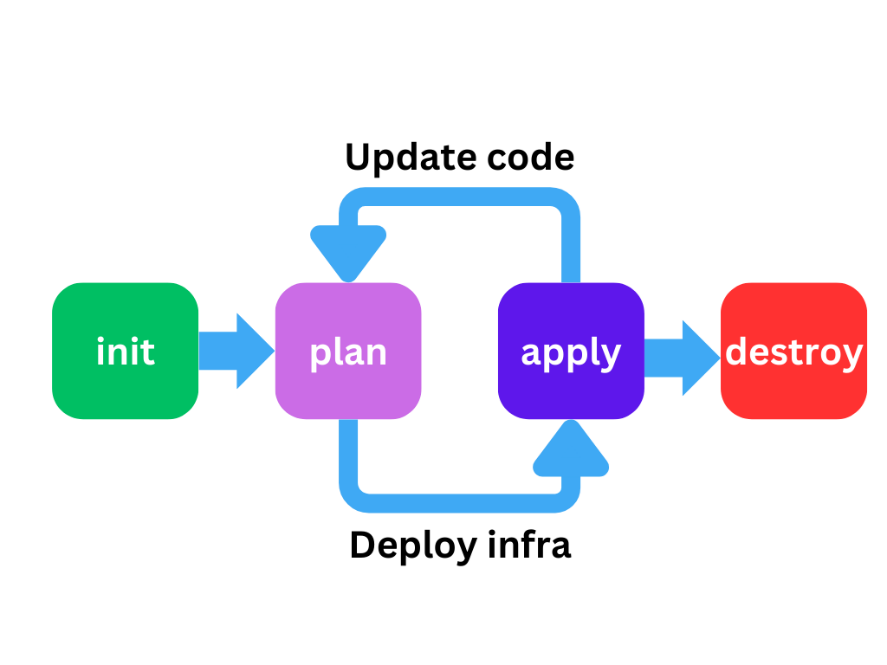


Figura 32. Flujo de comandos Terraform

La principal ventaja de utilizar Terraform es su consistencia y reproducibilidad, ya que al definir la infraestructura como código se garantiza que cada despliegue sea idéntico, independientemente del entorno donde se realice. Sumado a esto, la herramienta dispone de un ecosistema muy amplio de proveedores y de una comunidad activa que comparte módulos reutilizables, acelerando la adopción de buenas prácticas. Esta integración fluida con sistemas de control de versiones como Git y con pipelines de CI/CD permite mantener un historial de cambios y colaborar de manera más eficiente, posibilitando la implementación de una infraestructura inmutable y robusta.

## 3.5 Resumen del capítulo

En este capítulo se presentan las principales herramientas y tecnologías utilizadas en el proyecto, destacando cómo se integran y complementan para lograr un entorno seguro y automatizado. En primer lugar se describe la virtualización con Docker, que permite aislar aplicaciones en contenedores ligeros y portátiles, optimizando el despliegue y la escalabilidad. Seguidamente, se aborda la implementación de Keycloak como servicio para la gestión centralizada de identidades y accesos, y la de Vault para la protección y administración de secrets (credenciales, claves, certificados). Ambas soluciones se integran como servicios haciendo uso de contenedores Docker, aislando esta capa de seguridad y permitiendo una integración más fácil, evitando modificar mínimamente el código fuente de la aplicación.

Finalmente, se explica el uso de Terraform como herramienta de Infraestructura como Código (IaC), la cual automatiza la creación y configuración de estos servicios. Se define de manera declarativa la infraestructura a desplegar. La combinación de estas tecnologías proporciona una base sólida para construir y gestionar entornos de aplicaciones haciéndolas más escalables, al tiempo que simplifica el control de cambios y la colaboración de diferentes trabajos de desarrollo que modificasen cualquier ámbito de la aplicación.

# Capítulo 04: Diseño e implementación

## 4.1 Investigación, propuesta y formación

## 4.2 Inicialización del entorno

## 4.3 Desarrollo e implementación del servicio

## 4.4 Pruebas

## 4.5 Resumen del capítulo

# Capítulo 05: Conclusiones y trabajo futuro

Guardar un token, como en el caso del presente trabajo un token de refresco generado por Keycloak y que está almacenado en el localStorage del navegador conlleva una serie de ventajas y desventajas.

### 5.1 Ventajas

Una de las ventajas principales de tener el token de refresco en el localStorage del navegador es su control y accesibilidad para cualquier script que se ejecute del lado del cliente de una aplicación o página web que necesite hacer uso de él, siendo su acceso más sencillo. Esto facilita mucho la implementación de sistemas que necesitan enviar el token en cada petición a la API para procesos de autenticación o autorización.

Otra gran ventaja de que el token sea persistente se traduce a una mejora del rendimiento de la aplicación y una experiencia de usuario más fluida, en condiciones las que el usuario actualice la página o la cierra y vuelva a abrir el navegador, siempre que no se borren los datos de navegación. Mientras el usuario no borre los datos de navegación, se evita la necesidad de solicitar el inicio de sesión continuamente cada vez que abandone la aplicación. Aplicaciones que renderizan su contenido en una sola página, conocidas como SPA (Single Page Applications) el uso del localStorage es bastante práctico para guardar y recuperar el token y resto de información, traduciéndose en un mejor rendimiento.

### 5.2 Desventajas

Algunos equipos desarrolladores confían en que, si el token está “oculto” en el Local Storage, está protegido. Sin embargo, el Local Storage no ofrece ningún mecanismo de protección frente a scripts maliciosos que se ejecuten en el contexto de la misma página web.

Si un atacante logra inyectar código malicioso en la aplicación mediante XSS (Cross-Site Scripting), estos scripts pueden leer el contenido del localStorage y, por tanto, robar el token. Consecuencia de ello podría ser que el atacante consiga realizar peticiones a la API suplantando la identidad del usuario o incluso renovar el token. Si el atacante obtiene la capacidad de ejecutar JavaScript en la aplicación, no existe un control nativo que impida que se lea el token, llegaría a obtenerlo y leer su contenido.

### 5.3 Trabajo futuro

Como trabajo futuro de mejorar de seguridad en las distintas aplicaciones sin tener que modificar librerías o código fuente nativo, se podría almacenar el token de acceso con cookies en lugar de en el localStorage del navegador. Haciendo uso de los flags “HttpOnly” (que impide el acceso al Local Storage y al token desde JavaScript) y “Secure” (que fuerza la transmisión solo por HTTPS), consiguiendo así reducir significativamente la probabilidad de su acceso no autorizado, ya que el localStorage del navegador está diseñado solo para ser accesible desde el lado cliente, agregándose una capa adicional de seguridad frente a XSS y plagio de tokens.

Como configuración adicional se podrían definir y establecer tiempos de caducidad de los tokens de acceso y de refresco en función del tipo de usuarios y del tipo de información que se almacene.

### 5.4 Resumen del capítulo

En este capítulo se exponen las implicaciones de almacenar un token de refresco en el localStorage del navegador, haciendo hincapié en que dicha práctica ofrece ciertas ventajas, como facilitar el acceso al token por parte de scripts del lado del cliente y mejorar la experiencia de usuario al mantener la sesión iniciada incluso tras cerrar o actualizar la página; no obstante, también se señala la vulnerabilidad que supone frente a ataques de tipo XSS, ya que el localStorage puede ser leído si se inyecta código malicioso, lo que permitiría a un atacante suplantar la identidad del usuario o renovar el token. En cuanto al trabajo futuro, se propone el uso de cookies con flags de seguridad como HttpOnly y Secure para reducir significativamente el riesgo de acceso no autorizado, así como establecer políticas adecuadas de caducidad para los tokens de acceso y refresco según el tipo de usuario y la información manejada.

# Capítulo 06: Bibliografía

[1] F. J. Corbató y V. A. Vyssotsky, "Introduction and overview of the Multics system", Proceedings of the Fall Joint Computer Conference, pp. 185-196, 1965.

[2] T. Hunt, "Passwords are broken, and nobody seems to care", IEEE Security & Privacy, vol. 15, no. 5, pp. 70-74, 2017.

[3] Verizon, "2016 Data Breach Investigations Report", Verizon Enterprise, 2016.

[4] A. O. Udo, "Phishing: A growing challenge for internet users", Journal of Information Security and Applications, vol. 21, pp. 1-9, 2015.

[5] W. Yeong, T. Howes, y S. Kille, "Lightweight Directory Access Protocol", RFC 1777, 1995.

[6] S. Furnell, "Tokens and Beyond: Multi-Factor Authentication for the Masses", Computer Fraud & Security, vol. 2005, no. 8, pp. 12-16, 2005.

[7] National Institute of Standards and Technology, "Electronic Authentication Guideline", NIST Special Publication 800-63-2, 2013.

[8] P. Madsen, "A Guide to Understanding SAML", Sun Microsystems, 2003.

[9] OASIS, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 2005.

[10] D. R. Kuhn, "An overview of single sign-on technology", Proceedings of the 24th NIST-NCSC National Information Systems Security Conference, pp. 27-31, 2001.

[11] A. K. Das, N. Kumar, y J. J. P. C. Rodrigues, "Authentication protocols and schemes for smart mobile devices: security vulnerabilities, attacks, countermeasures, and open issues", Telecommunication Systems, vol. 67, no. 2, pp. 249-274, 2018.

[12] E. Hammer-Lahav, "The OAuth 1.0 Protocol", RFC 5849, 2010.

[13] D. Hardt, "The OAuth 2.0 Authorization Framework", RFC 6749, 2012.

[14] N. Sakimura et al., "OpenID Connect Core 1.0", The OpenID Foundation, 2014.

[15] M. Jones, "JSON Web Token (JWT)", RFC 7519, 2015.

[16] Gartner, "Magic Quadrant for Identity as a Service, Worldwide", Gartner Research, 2019.

[16] Gartner, “Identity as a Service (IDaaS)”, Gartner IT Glossary. Disponible: https://www.gartner.com/en/information-technology/glossary/identity-as-a-service-idaas.

[17] Red Hat, “Keycloak Documentation”. Disponible en https://www.keycloak.org/documentation.

[18] Red Hat, “Server Developer Guide: Service Provider Interfaces (SPI)”, Keycloak Documentation, 2023. Disponible en https://www.keycloak.org/docs/latest/server\_development/#\_providers.

[19] HashiCorp, “Vault: Secrets Management”, Vault by HashiCorp, 2023. Disponible en https://www.vaultproject.io/docs.

[20] K. Jackson, “Managing Secrets in Microservices with Vault”, ACM Queue, vol. 18, no. 5, pp. 34–47, 2020.

[21] M. Merkow y J. Breithaupt, “Virtualization and Containers: A Guide to Software Isolation and Security”, 2.ª ed. Boston, MA, EE. UU.: Pearson, 2021.

[22] Docker Inc., “Docker Overview”, Docker Documentation, 2023. Disponible en https://docs.docker.com/get-started/overview.

[23] Docker Inc., “Best practices for writing Dockerfiles”, Docker Documentation, 2023. Disponible en https://docs.docker.com/develop/develop-images/dockerfile\_best-practices.

[24] The Kubernetes Authors, “Kubernetes Documentation”, Kubernetes.io, 2023. Disponible en https://kubernetes.io/docs/home.

[25] The Kubernetes Authors, “Pod Security Standards”, Kubernetes Documentation, 2023. Disponible en https://kubernetes.io/docs/concepts/security/pod-security-standards.

[26] G. Kim, J. Humble, P. Debois y J. Willis, “The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations”, 2.ª ed. Portland, OR, EE. UU.: IT Revolution Press, 2021.

[27] N. Zubair, “DevSecOps: Integrating Security into DevOps”, IEEE Software, vol. 37, no. 3, pp. 20–27, 2020.

[28] OWASP, “Testing Guide”, OWASP Foundation, 2023. Disponible en https://owasp.org/www-project-web-security-testing-guide.

[29] P. Mell y T. Grance, “The NIST Definition of Cloud Computing”, NIST Special Publication 800-145, Gaithersburg, MD, USA, 2011.

[30] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg y I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”, Future Generation Computer Systems, vol. 25, núm. 6, pp. 599–616, 2009.

[31] T. Erl, R. Puttini y Z. Mahmood, “Cloud Computing: Concepts, Technology & Architecture”. Upper Saddle River, NJ, USA: Prentice Hall, 2013.

[32] M. Armbrust et al., “A View of Cloud Computing”, Communications of the ACM, vol. 53, núm. 4, pp. 50–58, 2010.

[33] M. Cusumano, A. Gawer y D. B. Yoffie, “The Business of Platforms: Strategy in the Age of Digital Competition, Innovation, and Power”. New York, NY, USA: Harper Business, 2019.

[34] K. Morris, “Infrastructure as Code: Managing Servers in the Cloud”. Sebastopol, CA, USA: O’Reilly Media, 2016.

[35] P. M. Fitts, S. L. Atkinson y J. Watson, “Infrastructure as Code: Principles and Practices of Immutable Infrastructure,” en Proceedings of the 2019 IEEE Conference on Software Engineering (ICSE), Montreal, Canada, 2019, pp. 123–130.

[36] M. Duffy, “Infrastructure as Code with Terraform: Provisioning Cloud Infrastructure in a Modern and Reproducible Way”. Sebastopol, CA, USA: O’Reilly Media, 2020.

[37] Pulumi, “Pulumi Documentation”. Disponible en https://www.pulumi.com/docs.

[38] Red Hat, “Ansible Documentation”. Disponible en https://docs.ansible.com.

[39] Progress Software, “Chef Documentation”. Disponible en https://docs.chef.io.

[40] Puppet, “Puppet Documentation”. Disponible en https://puppet.com/docs.