# 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**  
 1.1. Motivación  
 1.2. Objetivo General  
 1.3. Objetivos Específicos  
 1.4. Planificación  
 1.5. Estructura del Documento  
  
**Capítulo 02: Estado del Arte**  
 2.1. Introducción a la Seguridad en Aplicaciones  
 2.2. Métodos Tradicionales de Autenticación y Autorización  
 2.3. Tecnologías Modernas: SSO y Gestión de Identidades  
 2.4. Herramientas Principales: Keycloak y Vault  
 2.5. Contenedores, Orquestación y DevSecOps  
  
**Capítulo 03: Herramientas y Tecnologías**  
 3.1. Keycloak  
 3.2. Vault  
 3.3. Contenedores y Virtualización con Docker  
 3.4. Despliegue con Terraform  
  
**Capítulo 04: Diseño e Implementación**  
 4.1. Requerimientos Iniciales  
 4.2. Diseño de la Arquitectura  
 4.3. Implementación  
 4.4. Pruebas  
 4.5. Optimización  
  
**Capítulo 05: Resultados y Validación**  
 5.1. Análisis de la Solución  
 5.2. Comparación con Soluciones Tradicionales  
 5.3. Beneficios Observados  
  
**Capítulo 06: Conclusiones y Futuras Líneas de Trabajo**  
 6.1. Conclusiones Generales  
 6.2. Mejoras Propuestas  
 6.3. Extensiones del Trabajo  
  
**Capítulo 07: Bibliografía**

**Índice de figuras**

**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 cambios. Alta capacidad de reacción ante los cambios de los requisitos iniciales.

* 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 1: Introducción**:
  + Presentación del contexto y motivación del proyecto.
  + Definición de los objetivos generales y específicos.
  + Descripción de la planificación y metodología de trabajo.
  + Resumen de la estructura del documento.
* **Capítulo 2: Estado del Arte**:
  + Revisión de las tecnologías actuales en autenticación, autorización y gestión de secretos.
  + Análisis de soluciones existentes y su comparación con la propuesta del proyecto.
  + Justificación de la selección de **Keycloak** y **Vault**.
* **Capítulo 3: Descripción de Herramientas**:
  + Detalle de las características y funcionalidades de **Keycloak**.
  + Descripción de **Vault** y sus capacidades en la gestión de secrets.
  + Introducción a **Docker** y **Terraform** como herramientas de despliegue y automatización.
* **Capítulo 4: Diseño e Implementación**:
  + Explicación detallada de la arquitectura diseñada.
  + Pasos y procedimientos seguidos para la implementación de **Keycloak** y **Vault**.
  + Configuraciones específicas y consideraciones de seguridad.
  + Integración de las herramientas mediante Docker y Terraform.
* **Capítulo 5: Resultados y Pruebas**:
  + Presentación de los resultados obtenidos tras la implementación.
  + Detalle de las pruebas realizadas y su análisis.
  + Evaluación del desempeño y escalabilidad de la solución.
* **Capítulo 6: Conclusiones y Trabajos Futuros**:
  + Reflexiones sobre el cumplimiento de los objetivos.
  + Discusión de las ventajas y limitaciones encontradas.
  + Recomendaciones para mejoras y futuras investigaciones.
* **Capítulo 7: Bibliografía**
  + Listado de fuentes consultadas, incluyendo libros, artículos y documentación técnica.

**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[1]. 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[2]:

* 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.

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[3]. 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 Sistemas basados en contraseñas**

Los sistemas basados en contraseñas 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]. 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.

**Referencias:**

[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.

**2.2.2 LDAP (Lightweight Directory Access Protocol)**

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 Ligero de Acceso a Directorios (LDAP) fue creado como una versión simplificada del Protocolo de Acceso a Directorios (DAP) utilizado en el modelo X.500 de directorios de la ISO[5]. LDAP fue diseñado para ser más ligero y adecuado para su implementación en sistemas que no podían soportar la complejidad de X.500. 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.

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.

**Referencias:**

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

**2.2.3 Autenticación multifactor (MFA)**

Las crecientes amenazas y vulnerabilidades asociadas con las contraseñas llevaron a explorar 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 (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.

**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.

**Referencias:**

[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.

**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.**

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.

SAML y SSO abordaron la necesidad de interoperabilidad y mejoraron la experiencia del usuario al reducir la cantidad de veces que necesitaban autenticarse. 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.

**Referencias:**

[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.

**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.

**Referencias:**

[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.

**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.

Para superar las limitaciones de OAuth 1.0, se introdujo 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. De aquí surgió la necesidad de un estándar que proporcionara autenticación de manera segura sobre OAuth 2.0 llevó al desarrollo de **OpenID Connect**, donde se unificó y simplificó la autenticación en aplicaciones y servicios más recientes o modernos, permitiendo el inicio de sesión único y mejorando la seguridad al evitar el intercambio de credenciales sensibles entre los diferentes servicios.

**Referencias:**

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

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

**2.3.2 OpenID Connect**

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. 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.

**Referencias:**

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

**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. 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.

**Referencias:**

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

* + 1. **Identity as a Service (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. También 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.

**Referencias:**

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

**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.

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.

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]. 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.

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.

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.

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.

**2.6 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.

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]. Asimismo, la **infraestructura como código (IaaC)** permite describir y administrar la infraestructura por medio de definiciones en código, facilitando así la realización de revisiones y controles de seguridad integrados en el proceso de desarrollo.

**2.7 Introducción a Infrastructure as a Service (IaaS)**

**Infrastructure as a Service (IaaS)** es un modelo de servicio en la nube que proporciona recursos de infraestructura virtualizados a través de internet [29], como máquinas virtuales, almacenamiento, redes y sistemas operativos, lo cual facilita la gestión y automatización de infraestructuras en la nube y en entornos on-premise [30]. Entre sus características principales destaca la posibilidad de escalar recursos según las necesidades, evitando costosas inversiones en infraestructura física, junto con el modelo de pago por uso que reduce gastos operativos y el acceso global desde cualquier lugar con conexión a internet. Algunos de los principales proveedores de IaaS son Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud y HashiCorp.

HashiCorp ofrece herramientas clave como **Terraform**, que permite definir y aprovisionar infraestructuras mediante Infrastructure as Code (IaC) en múltiples proveedores utilizando un lenguaje declarativo, **Consul**, enfocada en el descubrimiento y configuración de servicios en entornos distribuidos, **Vault**, orientada a la gestión de secretos y la protección de datos sensibles, y **Nomad**, un orquestador de aplicaciones para despliegues en cualquier infraestructura.

El enfoque de IaC y las soluciones de HashiCorp resultan esenciales para la gestión de IaaS, pues garantizan consistencia al describir la infraestructura en código y evitan errores manuales, automatizan el aprovisionamiento y configuración de recursos para agilizar procesos y fomentan la colaboración mediante prácticas como el control de versiones y la integración continua. Además, se integran con metodologías como DevOps y DevSecOps, incorporando seguridad y operaciones en fases tempranas del ciclo de desarrollo. Esto ofrece ventajas como una mayor agilidad para responder a cambios, escalabilidad para aumentar o disminuir capacidades sin intervención manual, seguridad al incluir controles directamente en las definiciones de infraestructura y eficiencia al optimizar el uso de recursos y reducir el trabajo manual.

# 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 Contenedores y 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:

* **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.

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. 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.

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 requiere integrar Keycloak en un flujo de orquestación más complejo, puede convivir 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 sistemas de archivos locales. 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.

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 el 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.

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.

La principal ventaja de utilizar Terraform radica en 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.

**Capítulo 04: Diseño e Implementación**

**4.1 Requerimientos Iniciales**

Especificación funcional y no funcional.

**4.2 Diseño de la Arquitectura**

Diagrama de la arquitectura: componentes, flujos de datos y dependencias.

Justificación de elecciones tecnológicas.

**4.3 Implementación**

Configuración de Keycloak para SSO.

Integración de Vault con aplicaciones mediante API REST.

Ejemplo: implementación de roles y secretos dinámicos.

**4.4 Pruebas**

Estrategias de pruebas: unitarias, integrales y de carga.

Ejemplo: simulación de ataques para validar seguridad.

**4.5 Optimización**

Escalabilidad del sistema.

Mejora en tiempos de respuesta.

Propuestas para minimizar fallos.

**Capítulo 05: Resultados y Validación**

**5.1 Análisis de la Solución**

Evaluación del cumplimiento de objetivos iniciales.

Métricas clave: tiempo de autenticación, reducción de vulnerabilidades.

**5.2 Comparación con Soluciones Tradicionales**

Comparación de eficiencia y facilidad de uso.

Análisis de coste-beneficio.

**5.3 Beneficios Observados**

Mayor seguridad en la gestión de identidades.

Reducción en la complejidad de mantenimiento.

Flexibilidad en integración con aplicaciones heterogéneas.

**Capítulo 06: Conclusiones y Futuras Líneas de Trabajo**

**6.1 Conclusiones Generales**

Resumen del impacto del proyecto.

Reflexión sobre el aprendizaje adquirido y su relevancia en la industria.

**6.2 Mejoras Propuestas**

Inclusión de autenticación basada en biometría.

Implementación de IA para detección proactiva de anomalías.

**6.3 Extensiones del Trabajo**

Integración con otras herramientas como Kubernetes y Jenkins.

Adaptación para empresas de diferentes sectores.

**Capítulo 07: Bibliografía**

[1] OWASP Foundation, "OWASP Top Ten Security Risks," 2021. Disponible en: <https://owasp.org/www-project-top-ten/>.

[2] "Keycloak Documentation," Red Hat, 2023. Disponible en: <https://www.keycloak.org/documentation>.

[3] "Vault by HashiCorp," HashiCorp, 2023. Disponible en: <https://www.vaultproject.io/docs>.

[4] International Organization for Standardization, "ISO/IEC 27001:2013 Information Security Management Systems," ISO, 2013.

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

[6] "Vault Authentication Methods," HashiCorp, 2023. Disponible en: <https://developer.hashicorp.com/vault/docs/auth>.

[7] "Docker Documentation," Docker Inc., 2023. Disponible en: <https://docs.docker.com/>.

[8] "Terraform by HashiCorp," HashiCorp, 2023. Disponible en: <https://developer.hashicorp.com/terraform/docs>.

[9] K. Morris, Infrastructure as Code: Managing Servers in the Cloud, 2ª ed., Sebastopol, CA, USA: O'Reilly Media, 2020.

[10] Ross Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2008.

[11] National Institute of Standards and Technology, Security and Privacy Controls for Federal Information Systems and Organizations. NIST SP 800-53, 2013.

[12] World Economic Forum, The Global Risks Report 2021. Disponible en <https://www.weforum.org/reports/the-global-risks-report-2021>.

[13] Rick Shay, Saranga Komanduri, Adam L. Durity, P. Sawaya Huh, Michelle L. Mazurek y Blase Ur, "Can long passwords be secure and usable," en Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2014, pp. 2927-2936.

[14] Verizon, 2020 Data Breach Investigations Report. Disponible en <https://www.verizon.com/business/resources/reports/dbir/>.

[15] Ryan Heartfield y George Loukas, "A taxonomy of attacks and a survey of defence mechanisms for semantic social engineering attacks," ACM Computing Surveys (CSUR), vol. 48, no. 3, pp. 1-39, 2016.

[16] Fadi Aloul, "The Need for Effective Information Security Awareness," Journal of Advances in Information Technology, vol. 1, no. 3, pp. 130-133, 2010.

[17] Rachna Dhamija y Lisa Dusseault, "The Seven Flaws of Identity Management: Usability and Security Challenges," IEEE Security & Privacy, vol. 6, no. 2, pp. 24-29, 2008.

[18] Timothy Howes, Mark Smith y Gordon Good, Understanding and Deploying LDAP Directory Services. Addison-Wesley, 2003.

[19] Scott Cantor, John Kemp, Rob Philpott y Eve Maler, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, 2005.

[20] Sam Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Inc., 2015.

[21] Michael Jones, John Bradley y Nat Sakimura, JSON Web Token (JWT). IETF, 2015.

[22] Dick Hardt, The OAuth 2.0 Authorization Framework. IETF, 2012.

[23] Nat Sakimura, John Bradley, Michael Jones, Breno de Medeiros y Edvard Jay, OpenID Connect Core 1.0. OpenID Foundation, 2014.

[24] Eve Maler y Drummond Reed, "The Venn of Identity: Options and Issues in Federated Identity Management," IEEE Security & Privacy, vol. 6, no. 2, pp. 16-23, 2008.

[25] Gartner, Magic Quadrant for Access Management, 2019. Disponible en <https://www.gartner.com/>.

[26] Red Hat, Keycloak Documentation. Disponible en <https://www.keycloak.org/documentation>.

[27] Oguzhan Esen y Marko Hölbl, "Open-source identity and access management solutions: A comparative analysis," International Journal of Information Security Science, vol. 5, no. 3, pp. 580-588, 2016.

[28] HashiCorp, Vault Documentation. Disponible en <https://www.vaultproject.io/docs>.

[29] Martin Fowler, "Microservices: a definition of this new architectural term."Disponible en <https://martinfowler.com/articles/microservices.html>.

[30] Claus Pahl, "Containerization and the PaaS cloud," IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, 2015.

[31] Wesley Felter, Alexandre Ferreira, Ram Rajamony y Juan Rubio, "An updated performance comparison of virtual machines and Linux containers," en 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2015, pp. 171-172.

[32] Rui Shu, Yong Chao Chung y Weidong Wu, "A study of security vulnerabilities on Docker Hub," en Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, 2017, pp. 269-280.

[33] Brendan Burns y Joe Beda, Kubernetes: Up and Running. O'Reilly Media, Inc., 2015.

[34] Rania R. Moussa y Andreas Schlenker, "Security in Kubernetes Clusters: A Review of Threats and Solutions," en 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2019, pp. 1-6.

[35] Jez Humble y David Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.

[36] Debashis Sen y Ashutosh K. Ghose, "From DevOps to DevSecOps: A security practitioner's perspective," en Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 2421-2423.

[37] Abdelkader Zerouali, Arie van Deursen, Tom Mens y Ayda Santos, "A formal framework for measuring technical lag in component repositories and its application to npm," Empirical Software Engineering, vol. 24, no. 2, pp. 903-952, 2019.

[38] Peter Mell y Timothy Grance, The NIST Definition of Cloud Computing. NIST Special Publication 800-145, 2011.

[39] HashiCorp, HashiCorp Products. Disponible en <https://www.hashicorp.com/products>.