# 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]. OpenID Connect es una capa de identidad construida sobre el protocolo OAuth 2.0 que añade autenticación al proceso de autorización existente.

**Características y beneficios de OpenID Connect:**

* **Autenticación estandarizada:** Proporciona un marco simple y seguro para verificar la identidad del usuario.
* **Interoperabilidad:** Al ser un estándar ampliamente adoptado, facilita la integración entre diferentes proveedores de identidad y servicios.
* **Uso de JSON Web Tokens (JWT):** Utiliza JWT para transmitir de forma segura la información de identidad del usuario en un token de identificación (ID Token).

**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 APIs RESTful, surgió la necesidad de un mecanismo eficiente para transmitir información de autenticación y autorización de forma segura y ligera. En 2015, se estandarizaron los JSON Web Tokens (JWT) mediante la RFC 7519 [15].

**Contribución de JWT:**

* **Autenticación sin estado:** Permite que los servidores validen la identidad del usuario sin mantener sesiones en el servidor.
* **Eficiencia y portabilidad:** Al ser compacto y autónomo, es adecuado para aplicaciones web, móviles y APIs.
* **Integridad y seguridad:** Los tokens están firmados, lo que permite verificar su autenticidad y proteger contra manipulaciones.

**Integración con OAuth 2.0 y OpenID Connect:**

JWT complementó y mejoró las capacidades de OAuth 2.0 y OpenID Connect, proporcionando una forma estandarizada de transmitir información de forma segura. En OpenID Connect, el ID Token es un JWT que contiene información sobre la autenticación realizada.

**Referencias:**

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

**2.3.4 Identity as a Service (IDaaS)**

La creciente adopción de servicios en la nube y aplicaciones SaaS en la década de 2010 llevó a las organizaciones a buscar soluciones que pudieran gestionar identidades de manera eficiente en entornos híbridos y multicloud. Identity as a Service (IDaaS) emergió como una solución para estas necesidades [16].

**Evolución y beneficios de IDaaS:**

* **Modelo basado en la nube:** Proporciona servicios de gestión de identidades sin necesidad de infraestructura on-premise.
* **Integración de tecnologías modernas:** Soporta protocolos como OAuth 2.0, OpenID Connect y SAML, facilitando la interoperabilidad.
* **Escalabilidad y elasticidad:** Capacidad para adaptarse rápidamente a cambios en la demanda.

**Desafíos y consideraciones:**

* **Dependencia del proveedor:** La disponibilidad y seguridad dependen del proveedor de servicios.
* **Cumplimiento y regulación:** Es esencial garantizar que el almacenamiento y procesamiento de datos cumplan con regulaciones como GDPR.
* **Integración con sistemas heredados:** Puede requerir esfuerzos significativos para integrar con aplicaciones y sistemas existentes.

IDaaS representa la evolución natural de la gestión de identidades en un entorno donde la movilidad, la nube y la flexibilidad son esenciales.

**Referencias:**

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

**2.3.5 Convergencia de Tecnologías y Futuro de la Autenticación**

La progresión desde sistemas basados en contraseñas hasta las tecnologías modernas refleja la constante evolución para abordar necesidades emergentes y desafíos de seguridad. Cada avance tecnológico surgió en respuesta a las limitaciones de sus predecesores:

* **De contraseñas a LDAP:** La necesidad de gestión centralizada llevó al desarrollo de protocolos como LDAP.
* **De LDAP a MFA:** Las vulnerabilidades de las contraseñas impulsaron la adopción de MFA para añadir capas de seguridad.
* **De MFA a SAML y SSO:** La complejidad de múltiples autenticaciones y la necesidad de interoperabilidad condujeron al desarrollo de SAML y la implementación de SSO.
* **De SAML/SSO a OAuth 1.0:** El surgimiento de aplicaciones web y la necesidad de delegación de acceso seguro impulsaron la creación de OAuth 1.0.
* **De OAuth 1.0 a OAuth 2.0:** La necesidad de mayor flexibilidad y simplicidad llevó a la evolución hacia OAuth 2.0.
* **De OAuth 2.0 a OpenID Connect y JWT:** La necesidad de autenticación estandarizada y eficiente impulsó el desarrollo de OpenID Connect y la adopción de JWT.
* **IDaaS:** La adopción masiva de la nube y aplicaciones SaaS requería soluciones de gestión de identidades escalables y flexibles, dando lugar a IDaaS.

**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]. Proporciona autenticación y autorización para aplicaciones y servicios modernos.

Características principales:

* Soporte para SSO: Permite a los usuarios autenticarse una vez y acceder a múltiples aplicaciones sin volver a autenticarse.
* Protocolos estándar: Implementa protocolos como OAuth 2.0, OpenID Connect y SAML 2.0.
* Integración con LDAP y Active Directory: Sincroniza usuarios y grupos, facilitando la gestión centralizada.
* Flujos de autenticación personalizados: Permite definir pasos personalizados en el proceso de autenticación (por ejemplo, MFA, preguntas de seguridad).
* Internacionalización: Soporta múltiples idiomas y localizaciones.
* Administración basada en roles (RBAC): Gestión detallada de permisos y roles de usuario.

Análisis de características:

* Extensibilidad: Keycloak permite la extensión de funcionalidades mediante SPI (Service Provider Interfaces), permitiendo personalizar autenticaciones, proveedores de usuarios y más[18].
* Adaptabilidad: Se puede desplegar en diferentes entornos, ya sea on-premise o en la nube.
* Open source: Al ser de código abierto, cuenta con una comunidad que contribuye con mejoras, parches y soporte.

Casos de uso:

* Aplicaciones web y móviles: Proporciona autenticación para aplicaciones que requieren inicio de sesión de usuarios.
* API y microservicios: Protege endpoints de API mediante tokens de acceso y autorización basada en roles.
* Federación de identidades: Permite integrar múltiples fuentes de usuarios y proveedores de identidad externos.

**2.4.2 Vault**

Vault es una herramienta de código abierto para el manejo de secretos y la protección de datos sensibles desarrollada por HashiCorp[19]. Se enfoca en el control de acceso y el almacenamiento seguro de secretos como claves API, contraseñas y certificados.

Características principales:

* Almacenamiento seguro: Utiliza técnicas de cifrado avanzado para almacenar secretos.
* Control de acceso detallado: Políticas finas que determinan quién puede acceder a qué secretos y bajo qué condiciones.
* Credenciales dinámicas: Generación de credenciales de acceso bajo demanda para bases de datos y servicios en la nube, con tiempo de vida limitado.
* Auditoría: Registro detallado de todas las operaciones, lo que facilita el cumplimiento y la detección de actividades sospechosas.
* Integración con herramientas y servicios: Compatible con Kubernetes, Docker, sistemas de CI/CD y más.

Ejemplos de uso en proyectos reales:

* Protección de microservicios: En arquitecturas de microservicios, Vault puede proporcionar un almacén centralizado para secretos, evitando que estos se almacenen en código o configuraciones de servicio[20].
* Rotación de claves y contraseñas: Automatiza la rotación periódica de credenciales, reduciendo el riesgo en caso de compromiso.
* Encriptación como servicio: Proporciona APIs para cifrar y descifrar datos sin exponer las claves de cifrado a las aplicaciones.

Beneficios de utilizar Vault:

* Seguridad centralizada: Unifica la gestión de secretos, reduciendo puntos de fallo y exposición.
* Cumplimiento normativo: Facilita el cumplimiento de estándares como PCI DSS, HIPAA y GDPR al proporcionar controles y auditorías robustas.
* Flexibilidad: Se adapta a diferentes entornos y se integra con múltiples tecnologías.

**2.5 Contenedores, Orquestación y DevOps**

**2.5.1 Contenedores**

Los contenedores son una forma de virtualización a nivel de sistema operativo que permite ejecutar aplicaciones en entornos aislados[21]. Docker es una de las plataformas de contenedorización más populares.

Ventajas de Docker frente a la virtualización de sistemas operativos:

* Eficiencia de recursos: Los contenedores comparten el kernel del sistema operativo, lo que reduce el uso de recursos en comparación con las máquinas virtuales (VM) que requieren un sistema operativo completo por instancia[22].
* Portabilidad: Los contenedores pueden ejecutarse en cualquier sistema que soporte Docker, asegurando consistencia entre entornos de desarrollo, prueba y producción.
* Velocidad: El arranque de contenedores es significativamente más rápido que el de VMs, facilitando el escalado dinámico.
* Modularidad: Las aplicaciones pueden dividirse en componentes más pequeños (microservicios), que pueden desarrollarse, desplegarse y escalarse de forma independiente.

Desafíos de seguridad en contenedores:

* Aislamiento: Aunque los contenedores ofrecen cierto nivel de aislamiento, no es tan fuerte como el de las VMs. Las vulnerabilidades en el kernel pueden afectar a todos los contenedores.
* Imágenes inseguras: Uso de imágenes de contenedores que contienen vulnerabilidades o configuraciones inseguras[23].
* Gestión de secrets: Almacenamiento de claves y contraseñas dentro de imágenes o variables de entorno.

**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].

Funciones clave de Kubernetes:

* Gestión de clústeres: Administra nodos y recursos del sistema.
* Despliegue y escalado: Implementa y escala aplicaciones según la demanda.
* Autoreparación: Reemplaza y reinicia contenedores que fallan o no responden.
* Descubrimiento de servicios y balanceo de carga: Facilita la comunicación entre servicios y distribuye el tráfico de manera eficiente.
* Configuración y almacenamiento: Gestiona configuraciones y almacena datos de forma persistente.

Automatización para despliegues de sistemas seguros:

* Políticas de seguridad: Kubernetes permite definir políticas de seguridad a nivel de pod y contenedor, restringiendo capacidades y accesos[25].
* Integración con herramientas de seguridad: Se puede integrar con soluciones como Vault para gestionar secretos de forma segura.
* Actualizaciones y parches automatizados: Facilita la implementación de actualizaciones sin tiempo de inactividad significativo.

Buenas prácticas de seguridad en Kubernetes:

* Control de acceso: Implementar RBAC (Control de Acceso Basado en Roles) para restringir acciones de usuarios y servicios.
* Espacios de nombres: Utilizar namespaces para aislar recursos y aplicaciones.
* Seguridad de red: Configurar políticas de red para controlar el tráfico entre pods y servicios.

**2.5.3 DevOps**

DevOps es una metodología que combina prácticas de desarrollo de software (Dev) y operaciones de TI (Ops) para acortar el ciclo de vida de desarrollo y proporcionar entregas continuas con alta calidad[26]. Se basa en la colaboración, la comunicación y la integración entre equipos que tradicionalmente han trabajado de forma aislada.

Principios fundamentales de DevOps:

* Colaboración y comunicación: Fomentar una cultura en la que desarrolladores y operaciones trabajen juntos desde el inicio.
* Integración continua (CI): Automatizar la integración de cambios en el código, permitiendo detectar errores de forma temprana.
* Entrega continua (CD): Automatizar el despliegue de aplicaciones a entornos de producción de manera confiable y rápida.
* Infraestructura como código (IaC): Gestionar y aprovisionar la infraestructura mediante definiciones de código en lugar de configuraciones manuales.

Beneficios de DevOps:

* Mayor velocidad de entrega: Permite responder rápidamente a las necesidades del mercado y de los clientes.
* Mejora de la calidad del software: La automatización y las pruebas continuas reducen errores y mejoran la fiabilidad.
* Escalabilidad y eficiencia: Automatiza tareas repetitivas, liberando tiempo para actividades de mayor valor.

Herramientas comunes en DevOps:

* Sistemas de control de versiones: Git, SVN.
* Herramientas de CI/CD: Jenkins, GitLab CI/CD, CircleCI.
* Herramientas de gestión de configuración: Ansible, Puppet, Chef.
* Monitoreo y logging: Prometheus, Grafana, ELK Stack.

Desafíos en DevOps:

* Cambio cultural: Requiere un cambio en la mentalidad y prácticas establecidas, lo que puede encontrar resistencia.
* Seguridad: La rapidez en los despliegues puede llevar a omitir prácticas de seguridad si no se integran adecuadamente.
* Herramientas y automatización: Seleccionar y configurar las herramientas adecuadas puede ser complejo.

**2.6 DevSecOps**

DevSecOps es la evolución de DevOps que integra prácticas de seguridad en todo el ciclo de vida de desarrollo de software[27]. Reconoce que la seguridad es responsabilidad de todos y no solo del equipo de seguridad.

Principios clave de DevSecOps:

* Integración temprana de seguridad: Incorporar prácticas y pruebas de seguridad desde las etapas iniciales del desarrollo.
* Automatización: Utilizar herramientas que automaticen análisis de seguridad, pruebas y despliegues.
* Colaboración interfuncional: Fomentar la comunicación y colaboración entre desarrolladores, operaciones y seguridad.
* Respuesta rápida: Capacidad para responder y remediar vulnerabilidades y incidentes de manera ágil.

Beneficios de DevSecOps:

* Reducción de riesgos: Identificación temprana de vulnerabilidades, reduciendo el costo y el impacto de las correcciones.
* Cumplimiento continuo: Automatización de auditorías y reportes, facilitando el cumplimiento de regulaciones.
* Mejora de la calidad del software: Aplicación de prácticas de codificación segura y revisiones de código.

Herramientas y prácticas comunes:

* Análisis de código estático (SAST): Detecta vulnerabilidades en el código fuente.
* Análisis de código dinámico (DAST): Prueba aplicaciones en ejecución para identificar vulnerabilidades.
* Gestión de dependencias: Escanea bibliotecas y componentes de terceros en busca de vulnerabilidades conocidas[28].
* Infraestructura como código (IaaC): Define y gestiona la infraestructura mediante código, permitiendo revisiones y controles de seguridad.

**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 computacional virtualizados a través de internet[29]. Los proveedores de IaaS ofrecen servicios como máquinas virtuales, almacenamiento, redes y sistemas operativos que facilitan la gestión y automatización de infraestructuras en la nube y on-premise[30].

Características de IaaS:

* Recursos escalables: Permite ajustar recursos según las necesidades, evitando inversiones en infraestructura física.
* Pago por uso: Los clientes pagan solo por los recursos que utilizan, reduciendo costos operativos.
* Acceso global: Recursos disponibles desde cualquier lugar con conexión a internet.

Principales proveedores de IaaS:

* Amazon Web Services (AWS)
* Microsoft Azure
* Google Cloud Platform (GCP)
* IBM Cloud
* HashiCorp

Herramientas clave de HashiCorp:

* Terraform: Una herramienta de Infrastructure as Code (IaC) que permite definir y aprovisionar infraestructuras en múltiples proveedores de IaaS utilizando un lenguaje declarativo.
* Consul: Solución para el descubrimiento y configuración de servicios en entornos distribuidos.
* Vault: Como se mencionó anteriormente, para la gestión de secretos y protección de datos sensibles.
* Nomad: Un orquestador de aplicaciones para despliegues en cualquier infraestructura.

Importancia de IaaC y HashiCorp en la gestión de IaaS:

* Consistencia y reproducibilidad: IaC permite que las infraestructuras se definan en código, facilitando despliegues consistentes y evitando errores manuales.
* Automatización: Herramientas como Terraform automatizan el aprovisionamiento y configuración de recursos, mejorando la eficiencia.
* Colaboración: Al tratar la infraestructura como código, los equipos pueden utilizar prácticas de desarrollo como control de versiones, revisiones de código y CI/CD.
* Integración con DevOps y DevSecOps: IaC es fundamental para integrar prácticas de seguridad y operaciones en el ciclo de desarrollo.

Beneficios de utilizar IaC y herramientas de HashiCorp:

* Agilidad: Respuesta rápida a cambios en requisitos y demanda.
* Escalabilidad: Facilita la ampliación o reducción de infraestructuras sin intervención manual.
* Seguridad: Permite incorporar controles de seguridad directamente en las definiciones de infraestructura.
* Eficiencia: Optimiza el uso de recursos y reduce el tiempo dedicado a tareas manuales.

# Capítulo 03: Herramientas y Tecnologías

En este capítulo se detallan las herramientas y tecnologías utilizadas en el proyecto, proporcionando una comprensión profunda de cada componente y su integración en el sistema global. Las tecnologías clave incluyen Docker para la contenedorización, Keycloak para la gestión de identidad y acceso, Vault para la gestión de secretos y Terraform para 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, facilitando su despliegue y escalabilidad. Al aislar aplicaciones en contenedores, se garantiza un entorno consistente desde el desarrollo hasta la producción.

**Principales características de Docker:**

* **Portabilidad:** Los contenedores Docker pueden ejecutarse en cualquier sistema que tenga instalado Docker, independientemente de las diferencias en el sistema operativo o las configuraciones subyacentes.
* **Eficiencia de recursos:** Los contenedores son más ligeros que las máquinas virtuales tradicionales, ya que comparten el kernel del sistema operativo anfitrión.
* **Aislamiento:** Cada contenedor tiene su propio sistema de archivos, procesos y red, lo que asegura que las aplicaciones no interfieran entre sí.
* **Rapidez de despliegue:** Los contenedores pueden iniciarse en cuestión de segundos, acelerando el proceso de desarrollo y despliegue.

**Creación de imágenes Docker:**

* **Imágenes base:** Se utilizan imágenes base oficiales o de confianza como punto de partida para construir nuevas imágenes que contengan la aplicación y sus dependencias.
* **Dockerfile:** Es un archivo de texto que contiene una serie de instrucciones para construir una imagen Docker personalizada. Define cómo se debe configurar el entorno dentro del contenedor.
* **Construcción de imágenes:** Mediante el comando docker build, se construyen imágenes a partir de un Dockerfile, generando una plantilla que puede desplegarse en cualquier entorno Docker.

**Redes virtuales en Docker:**

* **Tipos de redes:**
  + **Bridge (por defecto):** Cada contenedor se conecta a una red bridge que permite la comunicación con otros contenedores en la misma red.
  + **Host:** El contenedor comparte la pila de red del anfitrión, útil para aplicaciones que necesitan alto rendimiento de red.
  + **Overlay:** Permite la comunicación entre contenedores en diferentes hosts Docker, útil en entornos de orquestación como Docker Swarm o Kubernetes.
* **Gestión de redes:** Docker proporciona comandos para crear y gestionar redes personalizadas, permitiendo definir cómo se comunican los contenedores y controlar el acceso.

**Volúmenes para persistencia de datos:**

* **Necesidad de persistencia:** Los datos almacenados dentro de un contenedor se pierden cuando este se elimina. Los volúmenes permiten que los datos persistan más allá del ciclo de vida de un contenedor.
* **Tipos de volúmenes:**
  + **Volúmenes administrados por Docker:** Son gestionados por Docker y almacenados en ubicaciones específicas del sistema, proporcionando aislamiento del sistema de archivos del anfitrión.
  + **Bind mounts:** Permiten montar directorios o archivos del sistema anfitrión dentro del contenedor, facilitando el intercambio de datos y la configuración.
* **Beneficios:**
  + **Persistencia de datos críticos:** Es esencial para bases de datos y servicios que requieren mantener información entre reinicios o actualizaciones.
  + **Compartición de datos entre contenedores:** Varios contenedores pueden acceder al mismo volumen, facilitando la colaboración entre servicios.
  + **Facilidad de backup y migración:** Los volúmenes pueden respaldarse y restaurarse independientemente de los contenedores.

## 3.2 Keycloak

Keycloak es una solución de código abierto para la gestión de identidad y acceso (IAM) que proporciona autenticación y autorización a aplicaciones y servicios modernos. Permite centralizar la gestión de usuarios, roles y permisos, facilitando la integración con múltiples aplicaciones y servicios.

**Instalación y configuración básica:**

* **Descarga e instalación:** Keycloak puede instalarse manualmente en un servidor de aplicaciones Java o, preferiblemente, desplegarse como un contenedor Docker para simplificar el proceso y asegurar la portabilidad.
* **Creación del usuario administrador:** Al iniciar Keycloak por primera vez, se crea un usuario administrador con privilegios para configurar el sistema.
* **Inicio del servidor:** Una vez instalado, se inicia el servidor de Keycloak, que estará listo para configurarse y gestionar autenticaciones.

**Uso de realms, roles y mapeos de usuarios:**

* **Realms:** Son espacios de gestión de seguridad que permiten aislar conjuntos de usuarios, credenciales y configuraciones. Cada realm es independiente, lo que facilita la gestión de múltiples aplicaciones o entornos.
* **Roles:** Representan permisos o privilegios asignables a usuarios o grupos, y pueden ser globales (a nivel de realm) o específicos de una aplicación (a nivel de cliente).
* **Mapeos de usuarios:**
  + **Asignación de roles a usuarios y grupos:** Facilita la gestión de permisos y acceso a recursos.
  + **Atributos y federación:** Permite integrar usuarios de fuentes externas como LDAP o bases de datos, centralizando la autenticación.

**Integración con aplicaciones mediante protocolos estándar como OAuth2:**

* **Configuración de clientes:** Cada aplicación que requiere autenticación se registra como un cliente en Keycloak, configurando parámetros como el tipo de cliente y las URLs de redirección.
* **Flujos de autenticación:** Soporta diversos flujos de OAuth2 y OpenID Connect, adaptándose a las necesidades de diferentes tipos de aplicaciones (web, móviles, SPA).
* **Integración en la aplicación:** A través de adaptadores y bibliotecas proporcionadas por Keycloak, las aplicaciones pueden integrarse fácilmente para manejar autenticación y autorización.

**Keycloak como servicio Docker:**

Desplegar Keycloak como un contenedor Docker ofrece varias ventajas:

* **Portabilidad:** El servicio puede ejecutarse en cualquier entorno que soporte Docker.
* **Facilidad de despliegue:** Se reduce la complejidad de instalación y configuración inicial.
* **Escalabilidad:** Es más sencillo escalar el servicio horizontalmente, añadiendo más instancias según la demanda.
* **Aislamiento:** El contenedor funciona de forma independiente, evitando conflictos con otras aplicaciones o servicios en el sistema anfitrión.

## 3.3 Vault

Vault es una herramienta de código abierto para la gestión de secretos, diseñada para controlar el acceso a tokens, contraseñas, certificados y claves de cifrado de manera centralizada y segura.

**Conceptos clave: secrets, policies y autenticación:**

* **Secrets:**
  + **Almacenamiento seguro de secretos estáticos:** Permite almacenar credenciales y datos sensibles de forma cifrada, accesible solo para entidades autorizadas.
  + **Generación de secretos dinámicos:** Puede crear credenciales temporales para servicios, aumentando la seguridad al limitar el tiempo de exposición.
* **Policies:**
  + **Definición de políticas de acceso:** Controlan qué operaciones pueden realizar usuarios o aplicaciones, permitiendo un control granular sobre los permisos.
* **Autenticación:**
  + **Métodos de autenticación variados:** Incluye tokens, AppRole, LDAP y otros, facilitando la integración con sistemas de autenticación existentes.
  + **Asignación de políticas durante la autenticación:** Al autenticar, se asignan las políticas correspondientes, controlando los permisos y accesos.

**Configuración de backend de almacenamiento:**

* **Elección del backend:** Vault almacena sus datos cifrados en diferentes backends, como Consul, etcd o sistemas de archivos locales, dependiendo de las necesidades de persistencia y disponibilidad.
* **Inicialización y sellado:** Tras la configuración, Vault debe inicializarse, generando claves maestras y un token raíz. Vault permanece "sellado" hasta que se proporcionan las claves maestras, protegiendo los secretos almacenados.

**Manejo de credenciales dinámicas:**

* **Generación de credenciales temporales para servicios:** Mejora la seguridad al reducir el tiempo en que las credenciales son válidas.
* **Proceso automatizado:** Las aplicaciones pueden solicitar credenciales dinámicas a Vault cuando las necesiten, siguiendo las políticas establecidas.

**Vault como servicio Docker:**

Al igual que Keycloak, Vault puede desplegarse como un contenedor Docker, lo que ofrece beneficios como:

* **Simplificación del despliegue:** Evita configuraciones complejas y agiliza la puesta en marcha.
* **Aislamiento y seguridad:** El contenedor está aislado del sistema anfitrión, reduciendo superficies de ataque.
* **Facilidad de actualización:** Es más sencillo actualizar o reemplazar versiones del servicio sin afectar al sistema en general.
* **Escalabilidad:** Permite implementar estrategias de alta disponibilidad y balanceo de carga con mayor facilidad.

## 3.4 Despliegue con Terraform

Terraform es una herramienta de infraestructura como código que permite definir, provisionar y gestionar la infraestructura de manera eficiente y reproducible, utilizando archivos de configuración declarativos.

**Automatización de infraestructura como código:**

* **Enfoque declarativo:** Se define el estado deseado de la infraestructura, y Terraform se encarga de realizar los cambios necesarios para alcanzarlo.
* **Proveedor agnóstico:** Soporta múltiples proveedores y servicios, permitiendo gestionar diferentes tipos de infraestructura con la misma herramienta.
* **Planificación previa:** Permite visualizar los cambios que se realizarán en la infraestructura antes de aplicarlos, reduciendo el riesgo de errores.

**Despliegue de infraestructura local con Terraform:**

* **Definición de recursos locales:** Aunque Terraform es conocido por su uso con proveedores de nube, también puede gestionar recursos locales, como contenedores Docker en un sistema local.
* **Automatización del despliegue de servicios Docker:** Terraform puede utilizarse para orquestar el despliegue de contenedores Docker, redes y volúmenes necesarios para el funcionamiento de Keycloak y Vault.
* **Gestión del ciclo de vida:** Terraform permite crear, actualizar y destruir recursos de manera controlada, facilitando el mantenimiento y la evolución de la infraestructura.

**Ventajas de Terraform frente a otras herramientas de orquestación:**

* **Consistencia y reproducibilidad:** Al definir la infraestructura como código, se asegura que cada despliegue sea idéntico, reduciendo errores y facilitando la colaboración entre equipos.
* **Amplio ecosistema de proveedores:** Soporta una variedad de proveedores y servicios, permitiendo una gestión unificada de infraestructuras heterogéneas.
* **Comunidad activa y módulos reutilizables:** La disponibilidad de módulos compartidos acelera el desarrollo y permite implementar prácticas recomendadas fácilmente.
* **Integración con herramientas CI/CD:** Terraform se integra bien en pipelines de integración y entrega continua, automatizando aún más el proceso de despliegue y permitiendo una infraestructura inmutable.
* **Control de versiones:** Al ser archivos de texto, las configuraciones pueden versionarse usando sistemas de control como Git, permitiendo rastrear cambios y colaborar eficientemente.

**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>.