Maestría en

CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTORES:

CHANCAY COELLO FELIX MANUEL

MUQUINCHE LEON JOSELYN MICHELLE

SANCHEZ ACOSTA OSCAR VINICIO

SANCHEZ LASCANO EDGAR JOEL

TUTORES:

Paulina Vizcaíno

Iván Reyes Chacón

TEMA

Diseño y Evaluación de un Esquema Integral de Autenticación y Autorización Segura para Apis basado en Principios de Zero Trust y defensa en profundidad, mediante control de acceso por tokens, políticas en Api gateway y administración federada al portal de AWS

Quito - Ecuador

Junio – 20206

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, **Félix Manuel Chancay Coello, Joselyn Michelle Muquinche León, Edgar Joel Sánchez Lazcano, Oscar Vinicio Sánchez Acosta**, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

FELIX MANUEL CHANCAY COELLO
Firmado digitalmente por FELIX MANUEL CHANCAY COELLO
Fecha: 2026.07.24 23:01:04 -05'00'

Firma

Félix Manuel Chancay Coello

JOSELYN MICHELLE MUQUINCHE LEON
Firmado digitalmente por JOSELYN MICHELLE MUQUINCHE LEON
Fecha: 2026.07.24 19:54:35 -05'00'

Firma

Joselyn Michelle Muquinche León

EDGAR JOEL SANCHEZ LASCANO
Firmado digitalmente por EDGAR JOEL SANCHEZ LASCANO
Fecha: 2026.07.24 20:12:43 -05'00'

Firma

Edgar Joel Sánchez Lazcano

OSCAR VINICIO SANCHEZ ACOSTA
Firmado digitalmente por OSCAR VINICIO SANCHEZ ACOSTA
Fecha: 2026.07.24 21:11:19 -05'00'

Firma

Oscar Vinicio Sánchez Acosta

Autorización de Derechos de Propiedad Intelectual

Nosotros**, Félix Manuel Chancay Coello, Joselyn Michelle Muquinche León, Edgar Joel Sánchez Lazcano, Oscar Vinicio Sánchez Acosta,** en calidad de autores del trabajo de investigación titulado ***Diseño y evaluación de un esquema integral de autenticación y autorización segura para APIs basado en principios de Zero Trust y Defensa en Profundidad, mediante control de acceso por tokens, políticas en API Gateway y administración federado al portal de AWS***, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, (junio 2026)

FELIX MANUEL CHANCAY COELLO
Firmado digitalmente por FELIX MANUEL CHANCAY COELLO
Fecha: 2026.07.24 22:59:25 -05'00'

Firma

Félix Manuel Chancay Coello

JOSELYN MICHELLE MUQUINCHE LEON
Firmado digitalmente por JOSELYN MICHELLE MUQUINCHE LEON
Fecha: 2026.07.24 19:55:14 -05'00'

Firma

Joselyn Michelle Muquinche León

EDGAR JOEL SANCHEZ LASCANO
Firmado digitalmente por EDGAR JOEL SANCHEZ LASCANO
Fecha: 2026.07.24 20:13:16 -05'00'

Firma

Edgar Joel Sánchez Lazcano

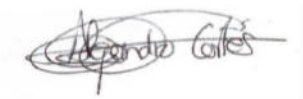
OSCAR VINICIO SANCHEZ ACOSTA
Firmado digitalmente por OSCAR VINICIO SANCHEZ ACOSTA
Fecha: 2026.07.24 21:12:00 -05'00'

Firma

Oscar Vinicio Sánchez Acosta

APROBACIÓN DE DIRECCIÓN Y COORDINACIÓN DEL PROGRAMA

Nosotros, Alejandro Cortés López e Iván Galo Reyes Chacón, declaramos que: Félix Manuel Chancay Coello, Joselyn Michelle Muquinche León, Edgar Joel Sánchez Lazcano, Oscar Vinicio Sánchez Acosta, son los autores exclusivos de la presente investigación y que esta es original, auténtica y personal de ellos.



Alejandro Cortés
Director/a de la
Maestría en Ciberseguridad

Iván Reyes
Coordinador/a de la
Maestría en Ciberseguridad

DEDICATORIA

Dedicamos el presente proyecto a nuestras Familias quienes son nuestro pilar y siempre nos han apoyado e impulsado a seguir adelante. A ellos quienes nos han apoyado en nuestras decisiones y creen en nosotros, muchas gracias.

A nuestros amigos y compañeros, que han estado presente y quienes forman parte de nuestras vidas y nos dan la mano a seguir adelante.

A nuestros docentes que nos han acompañado en este proceso, nos han brindado sus enseñanzas y conocimientos, y que nos han ayudado en nuestra formación.

Joselyn, Félix, Oscar & Edgar

AGRADECIMIENTOS

Ante todo, queremos agradecer a Dios que está presente en nuestras vidas, por darnos fuerzas para continuar ante las adversidades y por darnos la sabiduría para poder seguir por el buen camino y por su infinito amor.

A nuestras familias, queremos expresar nuestro más profundo y sincero agradecimiento, por ser el pilar fundamental de nuestras vidas, por cada palabra de aliento, por cada muestra de amor y apoyo. A ellos quienes han creído en nosotros aun cuando las dificultades parecían grandes y más aún cuando sentíamos que nuestras fuerzas flaqueaban y las dudas comenzaban a invadirnos, muchas gracias por nunca dejarnos caer y por brindarnos la fortaleza necesaria para no rendirnos.

A nuestros amigos y compañeros, que han estado presente y quienes forman parte de nuestras vidas, por acompañarnos a lo largo de este camino, por su apoyo constante y darnos la mano cuando lo hemos necesitado y que nos han ayudado a seguir adelante.

También queremos extender nuestro más sincero agradecimiento a nuestros docentes que nos han acompañado en este proceso formativo, nos han brindado sus enseñanzas, conocimientos y experiencias que han sido fundamentales para nuestro crecimiento académico y personal.

Joselyn, Félix, Oscar & Edgar

RESUMEN

El crecimiento de las arquitecturas basadas en APIs y microservicios ha incrementado los riesgos asociados con el acceso no autorizado y la exposición de servicios en entornos empresariales desplegados en la nube, esta situación ha impulsado la necesidad de implementar mecanismos de protección que sean capaces de fortalecer la seguridad de estos ecosistemas. En este contexto la presente investigación tuvo como objetivo diseñar y evaluar un esquema integral de autenticación y autorización para APIs basado en los principios de Zero Trust y defensa de profundidad orientado a reforzar el control de accesos mediante la integración de tecnologías para la gestión de identidades, la administración de APIs y el acceso federado a servicios de AWS.

La propuesta integró de Keycloak como proveedor de identidad, WSO2 API Manager como plataforma de gestión y protección de APIs, CyberArk Cloud Security para la administración del acceso privilegiado y AWS IAM Identity Center para el acceso federado al entorno de nube, incorporando autenticación basada en tokens JWT firmados digitalmente, autorización basada en roles y políticas de accesos aplicadas desde el API Gateway.

La validación se desarrolló en un laboratorio desplegado sobre Amazon Web Services utilizando contenedores Docker, un microservicio desarrollado en Spring Boot y una base de datos PostgreSQL, donde se ejecutaron múltiples escenarios de autenticación y autorización relacionados con solicitudes sin credenciales, tokens inválidos, firmas alteradas y privilegios insuficientes, obteniéndose respuestas consistentes con el comportamiento esperado y confirmando la eficacia de los mecanismos de seguridad implementados para prevenir accesos no autorizados y reforzar la protección de los servicios expuestos

Los resultados obtenidos evidencian que la arquitectura propuesta constituye una alternativa técnicamente viable para fortalecer la seguridad de APIs en entornos empresariales al incorporar controles de identidad, autorización y acceso federado que reducen la superficie de exposición frente a amenazas relacionadas con la autenticación y el control de acceso, favoreciendo la adopción de un modelo de seguridad alineado con los principios de Zero Trust y las prácticas actuales de protección para arquitecturas basadas en microservicios.

Palabras claves: *Zero Trust, autenticación, autorización, API Gateway, JWT, microservicios, acceso federado, defensa en profundidad.*

ABSTRACT

The growth of API-based and microservices architectures has increased the risks associated with unauthorized access and the exposure of services in enterprise environments deployed in the cloud. This situation has driven the need to implement protection mechanisms capable of strengthening the security of these ecosystems. In this context, this research aimed to design and evaluate an integrated authentication and authorization scheme for APIs based on the principles of Zero Trust and defense in depth, designed to reinforce access control through the integration of technologies for identity management, API administration, and federated access to AWS services.

The proposal integrated Keycloak as an identity provider, WSO2 API Manager as an API management and protection platform, CyberArk Cloud Security for privileged access management, and AWS IAM Identity Center for federated access to the cloud environment, incorporating authentication based on digitally signed JWT tokens, role-based authorization, and access policies applied from the API Gateway.

The validation was performed in a lab deployed on Amazon Web Services using Docker containers, a microservice developed in Spring Boot, and a PostgreSQL database. Multiple authentication and authorization scenarios were executed, including requests without credentials, invalid tokens, altered signatures, and insufficient privileges. The results were consistent with expected behavior, confirming the effectiveness of the implemented security mechanisms in preventing unauthorized access and strengthening the protection of exposed services.

The results demonstrate that the proposed architecture is a technically viable alternative for strengthening API security in enterprise environments. It incorporates identity, authorization, and

federated access controls that reduce the attack surface against threats related to authentication and access control, thus promoting the adoption of a security model aligned with Zero Trust principles and current best practices for protecting microservices-based architectures.

Keywords: *Zero Trust, authentication, authorization, API Gateway, JWT, microservices, federated access, defense in depth.*

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1:

1. INTRODUCCIÓN

1.1.Definición del proyecto

Las arquitecturas modernas basadas en Application Programming Interfaces (APIs) y en microservicios han aumentado las amenazas relacionadas con accesos no autorizados y a la exposición de servicios en ambientes empresariales distribuidos situación que se agrava debido a que varios servicios mantienen mecanismos tradicionales de autenticación y claves estáticas elevando los riesgos dentro de infraestructura distribuida.

En este contexto el presente proyecto consiste en el diseño, la implementación y la evaluación de un esquema integral de la autenticación y la autorización para APIs apoyado en el principio de zero trust y explicado como “nunca confiar, siempre verificar” bajo el cual ninguna solicitud será considerada confiable por defecto aun cuando provenga de los componentes internos de la arquitectura evitando así asumir confianza por pertenecer a una red interna o por provenir de un componente ya conocido.

Este trabajo surge de la necesidad de proteger APIs expuestas en entornos de producción empresariales distribuidos donde la utilización de arquitecturas basadas en microservicios incrementa la posibilidad de un ataque lo cual exige un modelo de seguridad centralizado y consistente para mitigar estos ataques. En este contexto se propone una arquitectura basada en microservicios junto con un API Gateway cuya incorporación permite separar las funciones del sistema, aplicar políticas homogéneas de autenticación y autorización, centralizar la validación de tokens, controlar el tráfico de entrada, registrar eventos de auditoría y limitar la exposición directa de los servicios internos del sistema. De esta manera, la arquitectura propuesta no solo

responde a una necesidad técnica de integración, sino también a un requerimiento de seguridad orientado a reducir accesos no autorizados y fortalecer la rastreabilidad de las solicitudes.

La solución propuesta integra tres capas principales. La primera corresponde a la gestión de identidades y la emisión de tokens mediante Keycloak como proveedor de Identity and Access Management (IAM) modelo encargado de la administración, autorización y gestión de identidades de los usuarios ((Keycloak, 2026). La segunda capa se enfoca en la protección y publicación controlada de APIs mediante WSO2 API Manager que es una plataforma que se encarga de la administración del ciclo de vida de las APIs encargado de validar solicitudes, aplicar políticas de seguridad y controlar el consumo de los servicios publicados (WSO2, s.f.). La tercera capa incorpora CyberArk Cloud Security (CyberArk, s.f.) como mecanismo de acceso federado al portal de Amazon Web Services (AWS) integrándose con AWS IAM Identity Center (AWS, 2022) para centralizar la autenticación, el aprovisionamiento de usuarios y la asignación de permisos bajo el principio de mínimo privilegio permitiendo evitar la dependencia de credenciales locales o accesos estáticos en AWS y refuerza el control de acceso sobre los recursos cloud durante la implementación del laboratorio.

Como parte de la arquitectura propuesta la autenticación y autorización se basarán en el uso de JSON Web Tokens (JWT) que es un estándar abierto para transmitir información entre dos partes de forma segura mediante un objeto JSON firmado digitalmente (JWT, s.f.) cuyos tokens estarán firmados haciendo uso del algoritmo asimétrico RS256 (Jones, JSON Web Token (JWT), 2015). Este enfoque fue seleccionado porque permite validar la autenticidad e integridad de los datos de forma segura mediante el uso de claves públicas fortaleciendo la seguridad del proceso de validación de identidades.

Cada token funcionará como un 'pasaporte' digital que incluye información obligatoria (claims) para dar contexto a la sesión, como el emisor (iss), el usuario (sub), la audiencia (aud) y los tiempos de vigencia (iat, exp, nbf). Además, se incluirán un identificador único (jti) y los permisos específicos del usuario. El API Gateway actuará como el primer filtro verificando que la firma sea válida y que el token no haya expirado mientras que el microservicio realizará una segunda validación más profunda asegurándose de que el usuario tenga los privilegios mínimos necesarios para la acción que intenta realizar.

El proyecto analizará escenarios de riesgo concretos dentro de un entorno controlado considerando situaciones relacionadas como el uso de tokens expirados, alterados o con audiencias distintas, así como accesos con privilegios insuficientes. Asimismo, se evaluará el consumo directo del microservicio sin pasar por el API Gateway, la reutilización de credenciales y el acceso no autorizado a recursos internos con el propósito de verificar si la arquitectura propuesta es capaz de prevenir, bloquear o mitigar ataques frecuentes en ecosistemas de APIs.

1.2.Justificación e importancia del trabajo de investigación

La proliferación de los servicios distribuidos y las aplicaciones móviles ha ido consolidado a las APIs como el eje principal de integración tecnológica, aunque este avance no ha estado acompañando por esquemas de protección equivalentes. De acuerdo con Open Worldwide Application Security Project (OWASP) que es una organización internacional que promueve buenas prácticas para la seguridad de las aplicaciones (OWASP, 2023) las APIs se encuentran entre los componentes más expuestos a vulnerabilidades. Estas vulnerabilidades facilitan el acceso no autorizado, la exposición de información sensible y el escalamiento de privilegios convirtiéndose en un objetivo frecuente para los ciberataques. (OWASP, 2023)

En las arquitecturas modernas de microservicios, la protección individual de cada componente resulta compleja y difícil de administrar (Sam Newman, 2021). En este contexto, el uso de un API Gateway es una práctica recomendada para centralizar mecanismos de seguridad, gestión de tráfico y monitoreo. Asimismo, proporciona un punto único para la aplicación de políticas de seguridad y administración de acceso a los servicios incorporando capacidades de registro y seguimiento de las solicitudes realizadas a los servicios. (Sam Newman, 2021)

Desde la perspectiva de la gestión de identidades, el uso de Keycloak permite desacoplar la lógica de seguridad del código de negocio, mientras que la incorporación de CyberArk Cloud Security habilita el acceso federado al portal de AWS, eliminando el uso de credenciales estáticas y centralizando la gestión de accesos privilegiados a recursos cloud. Así, la propuesta plantea una integración de mecanismo de gestión de identidades, control de acceso y acceso federado a recursos Cloud bajo un enfoque unificado para entornos de microservicios.

La importancia de esta investigación radica en su viabilidad para entornos corporativos reales. No se propone un rediseño disruptivo, sino un modelo escalable que plantea la integración de microservicios, tokens de seguridad y acceso federado a través de CyberArk Cloud Security con el propósito de reducir la superficie de exposición y mejorar la postura de seguridad en ecosistemas de nube y entornos distribuidos.

1.3.Alcance de estudio

El alcance del presente proyecto comprende el diseño, la implementación y la validación de un modelo integral de protección de APIs en un entorno empresarial desplegado sobre AWS, plataforma seleccionada por sus capacidades para soportar arquitecturas basadas en microservicios y servicios en la nube. La propuesta contempla la integración de Keycloak como

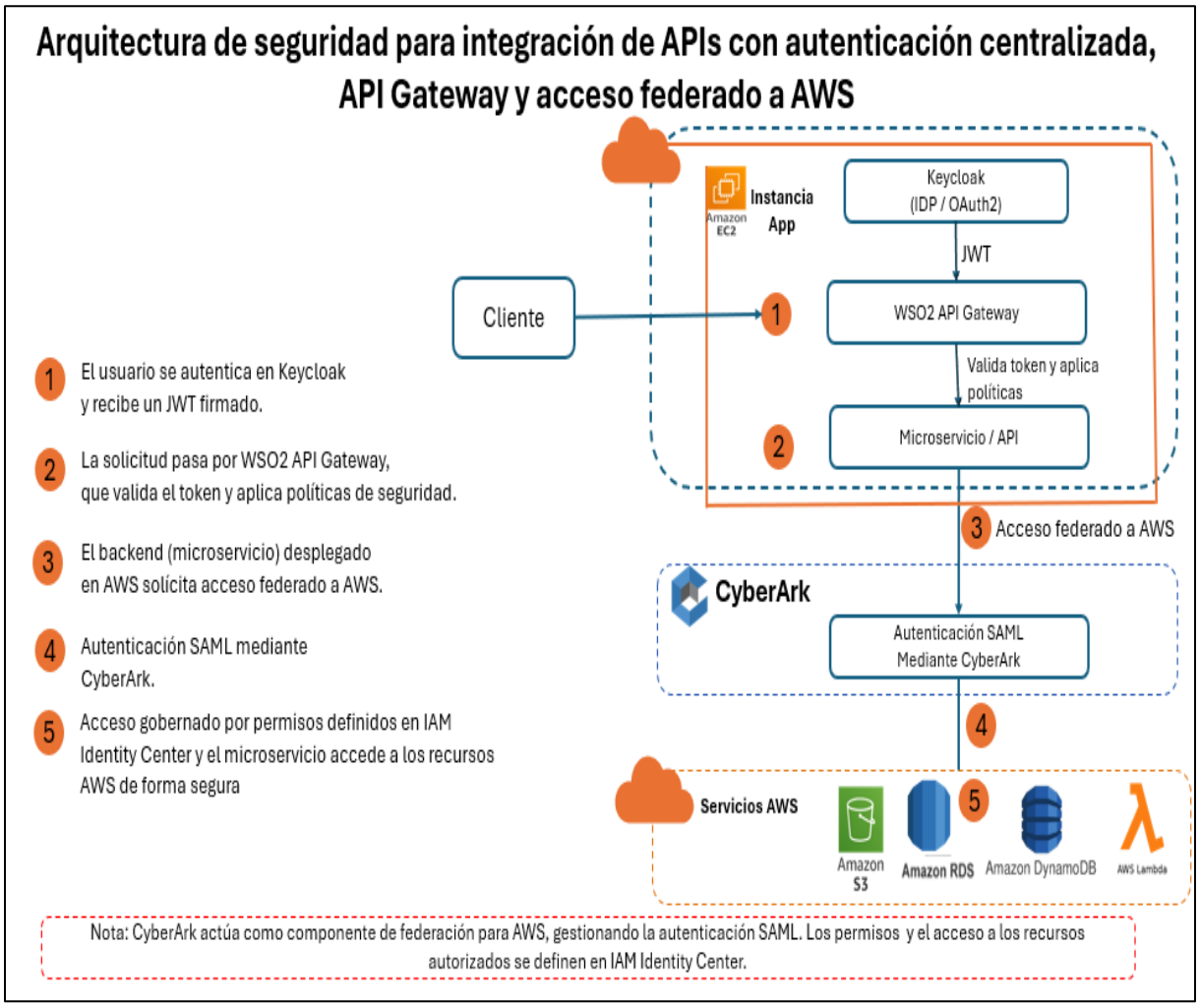
proveedor de identidad, WSO2 API Manager como plataforma de gestión y exposición de APIs, un microservicio de prueba desarrollado en Spring Boot (Broadcom, s.f.), una base de datos PostgreSQL (PostgreSQL Global Development Group, s.f.) y CyberArk Cloud Security como componente de acceso federado al portal de AWS.

Como parte de este alcance se incluirá la configuración de flujos de autenticación y autorización basados en estándares como OAuth 2.0 (D. Hardt, 2012) y OpenID Connect (OpenID Foundation, 2023), junto con la emisión y validación de tokens JWT, la aplicación de políticas de seguridad a nivel de API Gateway y el acceso federado al portal de AWS a través de CyberArk Cloud Security para el control centralizado de accesos a recursos en la nube. También se considerará la ejecución de pruebas funcionales y de seguridad en un laboratorio controlado con el propósito de verificar el comportamiento de la arquitectura propuesta frente a solicitudes legítimas y escenarios de ataque definidos previamente.

El laboratorio de validación estará estructurado a partir de un cliente consumidor de APIs, un proveedor de identidad, un API Gateway, un microservicio de prueba, una base de datos PostgreSQL y un componente de acceso federado a AWS mediante CyberArk Cloud Security. De esta forma, la interacción seguirá un flujo en el cual el cliente obtiene un token desde Keycloak, consume la API a través de WSO2 API Manager, el gateway valida el token y aplica políticas de seguridad, y el microservicio procesa la solicitud únicamente cuando se cumplen los controles de autenticación y autorización definidos. De forma complementaria, CyberArk Cloud Security se integrará con AWS IAM Identity Center para administrar el acceso federado de usuarios al portal de AWS, evitando el uso de credenciales estáticas y permitiendo aplicar permisos diferenciados según el perfil asignado. (CyberArk, 2024; AWS, s.f.)

Figura 1.

Arquitectura de seguridad para integración de APIs con autenticación centralizada, API Gateway y acceso federado a AWS



Como parte de este alcance, se ejecutarán pruebas controladas que simulen amenazas comunes en entornos de APIs. Se contemplará el uso de JWT expirados, mal formados, con firmas manipuladas o destinados a audiencias no válidas, así como intentos de acceso directo a los microservicios sin pasar por el API Gateway. De esta manera se determinará si un control bloqueó o mitigó correctamente el ataque. Cada prueba será evaluada mediante criterios técnicos

previamente definidos, tales como el código de respuesta del Hypertext Transfer Protocol (HTTP) que corresponde al código numérico que devuelve el servidor para indicar el resultado de una solicitud (Fielding, Nottingham, & Reschke, 2022), el mensaje de error generado, el registro del evento en los logs del API Gateway, la ausencia de acceso al recurso protegido y la no ejecución de operaciones no autorizadas en el microservicio o en la base de datos. De esta manera, un control se considerará efectivo cuando la solicitud no autorizada sea rechazada. Además, el evento deberá quedar registrado como evento de seguridad y no deberá producirse cambios ni exposición de información en los recursos internos. El propósito de estas pruebas será verificar la efectividad de los controles implementados frente al movimiento lateral, escalamiento de privilegios (The MITRE Corporation, 2025), uso indebido de credenciales y manipulación de tokens. (OWASP, 2023)

De forma complementaria, se podrán ejecutar pruebas de abuso de endpoints orientadas a validar el comportamiento de los mecanismos de limitación de tasa configurados en el API Gateway. Estas pruebas no constituirán el eje principal de la investigación, sino que servirán como apoyo para observar la respuesta del modelo frente a solicitudes repetitivas o anómalas dentro del laboratorio controlado. Cada escenario será definido previamente, identificando el componente involucrado, el control que debe actuar y el resultado técnico esperado, considerando como respaldo los códigos de respuesta HTTP, los registros generados en el gateway y la ausencia de afectación al microservicio protegido. Con ello, se busca complementar la evaluación de la arquitectura sin duplicar las pruebas principales de autenticación, autorización, protección de secretos y control de acceso.

La evaluación del esquema se apoyará en indicadores técnicos concretos. Para ello, se considerará la efectividad del bloqueo, medida a partir de las solicitudes no autorizadas y ataques

mitigados frente al total de intentos; el rendimiento, medido por el tiempo requerido por WSO2 API Manager para validar los tokens JWT y procesar las solicitudes; la precisión de las reglas, expresada en el porcentaje de validaciones correctas frente al total de escenarios ejecutados; y la visibilidad, entendida como el nivel de registro de los eventos de autenticación, autorización, consumo de APIs y acceso federado a AWS. Estos datos permitirán determinar si la arquitectura reduce la superficie de exposición y si las políticas se aplican de forma consistente en las capas de identidad, gateway, microservicio y acceso cloud.

Cabe señalar que en el marco de este proyecto, la solución de CyberArk ha sido renombrada como Idira, esto debido a que la marca fue adquirida por la empresa Palo Alto. El desarrollo documentado en el presente trabajo refleja el estado del proyecto hasta el 31 de mayo de 2026, fecha previa a dicho cambio de nombre y cambio de los portales de administración de CyberArk. Por tanto, todas las referencias, configuraciones e implementaciones descritas corresponden a la versión del proyecto anterior al cambio de nombre a Idira, en nuestro proyecto su nombre continuará siendo CyberArk.

El proyecto se enfocará en la protección de APIs previamente definidas dentro del backend, sin incluir el rediseño completo de las aplicaciones existentes, el desarrollo de frontend ni la implementación detallada de toda la infraestructura de red en AWS. Para el laboratorio de validación se implementarán únicamente los elementos mínimos necesarios de AWS, correspondientes a Amazon Elastic Compute Cloud (EC2) para alojar los componentes del entorno, Amazon Elastic Container Registry (ECR) para almacenar las imágenes Docker (Docker, s.f.), Amazon Virtual Private Cloud (VPC) para la configuración básica de red e IAM para la gestión de permisos requeridos. Quedarán fuera del alcance otros servicios de AWS, procesos de automatización, pipelines de Integración Continua y Entrega Continua (CI/CD),

orquestación administrada, escalamiento automático y configuraciones avanzadas de alta disponibilidad, debido a que el despliegue se realizará mediante contenedores Docker en un entorno controlado. Asimismo, no formará parte del alcance la incorporación de herramientas avanzadas de detección y respuesta extendida, análisis forense posterior a incidentes o certificaciones formales de cumplimiento normativo.

1.4.Objetivos

1.4.1. Objetivo general

Diseñar y evaluar un esquema integral de autenticación y autorización segura para APIs basado en principios de Zero Trust y defensa en profundidad mediante el uso de tokens, políticas de seguridad en un API Gateway y acceso federado a recursos en la nube a través de CyberArk Cloud Security para fortalecer los mecanismos de control de acceso en entornos empresariales en la nube.

1.4.2. Objetivos específicos

- Diseñar una arquitectura de seguridad para APIs que articule mecanismos de identidad, control perimetral y acceso federado a recursos cloud bajo principios de Zero Trust y mínimo privilegio.

- Implementar un esquema de autenticación y autorización basado en tokens JWT firmados digitalmente, definiendo criterios de emisión, validación, expiración y claims obligatorios.

- Configurar políticas de seguridad en el API Gateway para validar solicitudes, controlar el acceso a los servicios y reforzar la trazabilidad del consumo de APIs.

- Implementar el acceso federado al portal de AWS mediante CyberArk Cloud Security, configurando la integración con IAM Identity Center para gestionar de forma centralizada el acceso de usuarios a recursos cloud bajo políticas de mínimo privilegio.

- Construir un laboratorio de validación que reproduzca el flujo de autenticación, autorización y consumo de APIs en una arquitectura controlada.

- Evaluar el desempeño del esquema propuesto mediante pruebas de autenticación, autorización y escenarios de ataque controlados, con base en indicadores técnicos previamente definidos.

CAPÍTULO 2:

2. REVISIÓN DE LITERATURA

2.1.Estado del Arte

En los últimos años, la seguridad en las APIs ha adquirido una gran importancia esto debido al crecimiento de las arquitecturas basadas en microservicios y de los servicios expuestos en la nube. Diversas investigaciones y organismos especializados han identificado vulnerabilidades relacionados con los mecanismos de autenticación, autorización y protección de datos, dado a que esto representan riesgos importantes para las organizaciones que dependen de estos servicios digitales como parte de su infraestructura tecnológica. (Akamai, 2026)

Diversos estudios y reportes, como el OWASP API Security Top 10 en su versión más reciente publicado en el año 2023, constituyen una referencia muy importante para la identificación de vulnerabilidades en APIs, este estándar incluye riesgos críticos como las fallas en la autenticación, autorización y exposición de datos sensibles. (OWASP, 2023). Estas vulnerabilidades pueden ser explotadas por atacantes maliciosos para acceder a información sensible o realizar acciones no autorizadas dentro de los sistemas.

El OWASP API Security Top 10 (2023) clasifica las principales vulnerabilidades identificadas en las APIs modernas, enfocando su análisis en las debilidades asociadas con los mecanismos de autenticación, autorización y validación de solicitudes y la protección de datos.

Tabla 1.

Principales riesgos de seguridad en APIs según OWASP API Security Top 10 (2023)

CÓDIGO	RIESGO	DESCRIPCIÓN
API1:2023	Autorización a nivel de objeto defectuosa	Permite el acceso a recursos u objetos sin la validación correcta de los permisos del usuario.
API2:2023	Autenticación defectuosa	Fallas en la autenticación que permiten el robo o el uso indebido de tokens y sesiones.
API3:2023	Autorización de nivel de propiedad de objeto defectuosa	Esta categoría combina las normas API3:2019 (Exposición excesiva de datos) y API6:2019 (Asignación masiva), centrándose en la causa raíz: la falta de validaciones adecuadas por la exposición o modificación indebida de datos sensibles.
API4:2023	Consumo de recursos sin restricciones	El uso excesivo de recursos que puede provocar degradación o denegación del servicio.
API5:2023	Autorización de nivel de función defectuosa	Acceso no autorizado a funciones o acciones administrativas dentro de la API.
API6:2023	Acceso sin restricciones a flujos de negocio sensibles	Automatización abusiva de procesos críticos del negocio mediante APIs.
API7:2023	Falsificación de solicitudes del lado del servidor	Manipulación de solicitudes para acceder a recursos internos o externos no autorizados.
API8:2023	Configuración de seguridad incorrecta	los errores de configuración que exponen la API a diferentes vulnerabilidades.
API9:2023	Gestión inadecuada de inventarios	Falta de control y documentación de versiones o endpoints expuestos.
API10:2023	Consumo inseguro de API	Riesgos derivados del uso inseguro de APIs o servicios de terceros.

Nota. Adaptado de OWASP API Security Top 10. (OWASP, 2023)

En la

podemos observar los riesgos identificados por OWASP API Security Top 10 en donde se mencionan las principales vulnerabilidades en las APIs y en cómo están relacionadas con los controles deficientes de autenticación, autorización y validación de solicitudes. Estos aspectos representan elementos críticos dentro de arquitecturas modernas basadas en APIs y microservicios.

Bajo este contexto surge la necesidad de la implementación de herramientas o mecanismos de seguridad más robustos que nos permitan fortalecer los controles de acceso y la protección en las APIs. En especial en entornos empresariales en donde la autenticación, autorización y gestión de credenciales representan elementos críticos dentro de la arquitectura de seguridad.

En los últimos años, muchas organizaciones han comenzado a adoptar modelos de seguridad basados en Zero Trust, cuyo principio se basa en “nunca confiar, siempre verificar”, bajo este enfoque se busca reforzar los mecanismos de seguridad relacionados con la autenticación y validación continua dentro de arquitecturas modernas basadas en APIs y microservicios. (Scott Rose, Oliver Borchert, Stu Mitchell, Sean Connelly, 2020)

La adopción de este modelo nace como respuesta al crecimiento de entornos distribuidos y servicios expuestos en la nube, donde las APIs representan puntos críticos de acceso y comunicación entre sistemas. En este contexto el modelo Zero Trust se centra en la protección de los recursos en donde Zero podría mejorar la postura general de la seguridad de la tecnología de la información de una empresa.

De manera complementaria, los mecanismos de autenticación basados en token JWT son ampliamente usados en APIs y arquitecturas basadas en microservicios, esto permite validar la identidad y controlar el acceso de los usuarios de manera más segura. Además, nos facilitan el intercambio de información entre diferentes servicios, ayudando a reforzar la seguridad y protección de los datos en entornos distribuido. (M. Jones, J. Bradley, N. Sakimura, 2015).

Asimismo, los API Gateway juegan un papel importante dentro de los mecanismos de seguridad para la administración y protección de los servicios expuestos. Estas herramientas facilitan la gestión de solicitudes realizadas por los usuarios y aplicaciones externas hacia los servicios internos. De acuerdo con (Neil Madden, 2020), las arquitecturas orientadas a APIs requieren un enfoque de seguridad multinivel, especialmente en entornos distribuidos y basados en microservicios, para proteger adecuadamente los servicios expuestos y reducir riesgos asociados a posibles vulnerabilidades.

En los últimos años se han realizado diversas investigaciones en los cuales se han propuesto mecanismos de seguridad orientados a fortalecer la protección de los APIs y arquitecturas basadas en microservicios mediante el uso de modelos Zero trust, autenticación basada en JWT y herramientas de control de acceso. Estas investigaciones sirven como punto de partida para el análisis de mecanismos de seguridad aplicados a entornos basados en APIs y microservicios.

Bajo este contexto (Durán, 2025) propuso la integración de GraphQL y Seguridad Zero Trust en el diseño de APIS, este proyecto tuvo como objetivo fortalecer la seguridad en arquitectura basadas en microservicios. La investigación se enfocó en la implementación de mecanismos de autenticación y autorización para el control de acceso, reduciendo los riesgos

relacionados con vulnerabilidades en APIs modernas. Esto con el fin de evitar que información sensible y posibles ataques relacionados con accesos no autorizados.

Para alcanzar estos objetivos el autor incorporó tecnologías orientadas a la validación de usuarios y protección de recursos mediante el uso de herramientas como JWT, Keycloak y Vault, permitiendo controlar mecanismos de autenticación segura y gestión de credenciales dentro de entornos distribuidos. Los resultados de la presente demostraron mejoras relacionadas con la protección de datos, control de acceso y reducción de riesgos en la seguridad.

Dentro de este enfoque, (Christian Portoviejo & Marcos Pacheco, 2024) en su proyecto Análisis, diseño y desarrollo de un sistema modular basado en microservicios y (Amanda Cabascango & Marco Clavijo, 2025) a través de su propuesta orientada a arquitectura distribuidas empresariales, desarrollaron soluciones enfocadas al mejoramiento de la administración, escalabilidad y seguridad dentro de entornos tecnológicos basados en servicios distribuidos y APIs.

Ambos proyectos nacen bajo la necesidad al tener limitaciones presentes en arquitecturas tradicionales. Estas limitaciones se relacionan principalmente con la administración centralizada, control de acceso, disponibilidad de servicios y el intercambio de información entre arquitecturas actuales. Los estudios analizados aportan una referencia para la presente propuesta, al destacar la importancia de centralizar la gestión de accesos y la protección de servicios en entornos distribuidos.

Dentro de sus propuestas, los investigadores desarrollaron arquitecturas orientadas en microservicios, estrategias de comunicación a través de API Gateway, facilitando la centralización y control de la gestión de solicitudes, con esto se busca mejorar el control de

acceso dentro de entornos distribuidos. Además, incorporaron proceso de autenticación y control de acceso mediante JWT, facilitando la identificación de usuarios, garantizando el acceso seguro a los recursos de las aplicaciones

Los resultados de las investigaciones destacan que la implementación de los microservicios, API Gateway y los mecanismos de control de acceso fortalecen la seguridad, escalabilidad y administración de los servicios corporativos, disminuyendo riesgos asociados a accesos no autorizados y vulnerabilidades en APIs modernas.

Asimismo, las arquitecturas modernas basadas en APIs y microservicios no solo requieren mecanismos de autenticación y control de acceso a las aplicaciones. Además, necesitan mecanismos orientados para controlar los accesos a los recursos de nube. Dentro de este contexto, el acceso federado a través de CyberArk Cloud Security surge como alternativa para minimizar riesgos de seguridad y accesos privilegiados no autorizados. (CyberArk, 2024)

De acuerdo con el análisis realizado, las arquitecturas actuales basadas en APIs y microservicios requieren controles de seguridad robustos orientados a mejorar y a reforzar los procesos de autenticación, autorización, protección de claves y control de acceso, ayudando a reducir riesgos de seguridad y reforzando la protección de infraestructura empresariales. En este contexto, la integración de Keycloack, WSO2 API Manager y CyberArk permite abordar estos requerimientos mediante la gestión centralizada de identidades, la protección de APIs y el acceso federado a recursos de nube dentro de una misma arquitectura.

2.2.Marco Teórico

2.2.1. Ciberseguridad en APIs

La ciberseguridad comprende el conjunto de medidas, herramientas, mecanismos y tecnologías orientadas a proteger sistemas, redes, aplicaciones y datos contra accesos no autorizados, ataques y vulnerabilidades informáticas. Hoy en día, las APIs se han convertido en componentes esenciales en las arquitecturas modernas, dado a que posibilitan e intercambio de información entre diferentes servicios y aplicaciones. Debido a esto, asegurar las APIs se ha vuelto esencial porque es un elemento crítico para la seguridad informática y arquitecturas en la nube. (Ramaswamy Chandramouli , Zack Butcher, 2025)

Los elementos en la seguridad de APIs que abarcamos directamente en el proyecto son: la autenticación basada en tokens JWT, autorización mediante políticas de acceso y roles RBAC, control centralizado a través del API Gateway, acceso federado al portal de AWS a través de CyberArk Cloud Security, y finalmente la trazabilidad y auditoría de los accesos.

2.2.2. Application Programming Interface

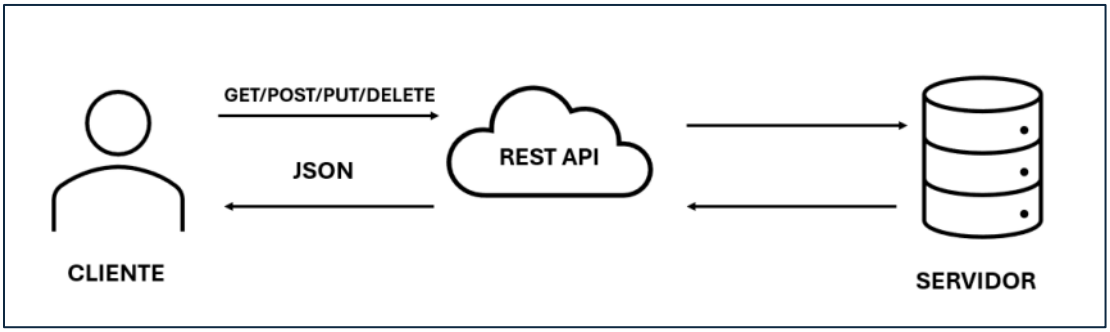
Las Application Programming Interface (API) son herramientas que facilitan la comunicación y el intercambio de información entre diferentes aplicaciones, sistemas y servicios a través de protocolos y reglas definidas. En la actualidad, las APIs desempeñan un papel esencial dentro de las arquitecturas actuales, dado a que facilitan la comunicación e integración de entre aplicaciones, plataformas, servicios web y entornos distribuidos. Estas interfaces posibilitan la comunicación y el intercambio de información entre diferentes aplicaciones, de forma eficiente, a su vez favorece la integración dentro de los sistemas tecnológicos actuales. (Red Hat, 2020)

En la actualidad las Interfaces de Programación de Aplicaciones de Transferencia de Estado Representacional (APIs REST) destacan su uso dado a que juegan un papel muy

importante permitiendo la comunicación entre distintos servicios y aplicaciones. Estas APIs trabajan mediante solicitudes y respuestas utilizando protocolos como HTTP y métodos como GET, POST, PUT y DELETE, los cuales permiten consultar, registrar, actualizar o eliminar información dentro de una aplicación. Asimismo, utilizan formatos ligeros como JSON para la transmisión o comunicación de información entre clientes y servidores.

Figura 2.

Modelo API REST



En la **¡Error! No se encuentra el origen de la referencia.** se puede observar cómo es la comunicación entre un cliente y un servidor mediante una API REST. El cliente realiza solicitudes HTTP usando métodos como GET, POST, PUT y DELETE y el servidor envía su respuesta en formato JSON mediante el API.

Las APIs usan endpoints que son puntos de acceso específicos para acceder a recursos y ejecutar funciones dentro del sistema. Debido a esto, las APIs nos ayudan a la integración de aplicaciones internas y externas, especialmente en arquitecturas modernas y servicios cloud. Sin embargo, a pesar de sus beneficios, la exposición de servicios mediante APIs pueden ser perjudicial y estar asociados a vulnerabilidades asociadas con acceso no autorizados, fuga de

información y errores de autenticación. Por tal motivo es necesario implementar herramientas que permitan fortalecer la seguridad y el control de acceso.

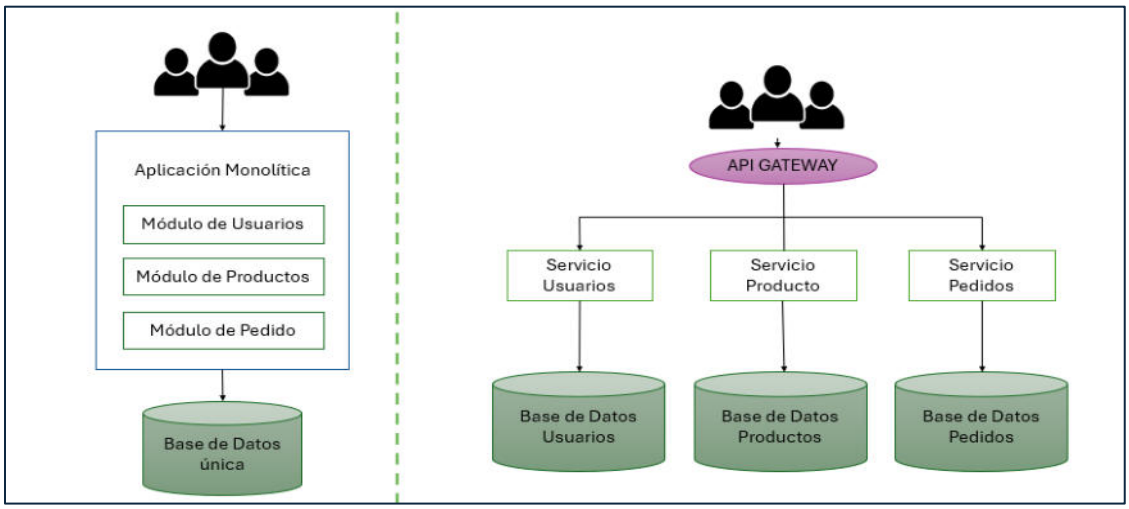
2.2.3. Arquitectura basada en microservicios

Los microservicios son un modelo de arquitectura que consiste en separar una aplicación en múltiples servicios pequeños e independientes capaces de operar y actualizarse de forma autónoma. Cada uno de estos microservicios pueden intercambiar información entre sí, permitiendo una mayor flexibilidad, facilidad de mantenimiento, mejorando la integración en entornos distribuidos y servicios en la nube. (AWS, s.f.)

Los microservicios se diferencian de las arquitecturas monolíticas, porque dividen las funcionalidades en servicios independientes, mientras que las monolíticas integran en una sola estructura. Esto favorece el mantenimiento y escalabilidad de componentes individuales sin comprometer el funcionamiento del sistema. Además, la arquitectura permitirá la integración de múltiples servicios y tecnologías, mejorando la adaptabilidad y disponibilidad de las aplicaciones actuales. (Redhat, 2023)

Figura 3.

Comparación entre arquitectura monolítica y arquitectura basada en microservicios



A pesar de sus ventajas, el uso de microservicios puede incrementar riesgos de seguridad, por la gran cantidad de múltiples servicios expuestos y conexiones dentro de la infraestructura tecnológica. Según la publicación de Chandramouli y Butcher en NIST llamado: “Guidelines for API Protection for Cloud-Native Systems”, la fragmentación de los servicios en microservicios amplió la superficie de ataque, ya que cada servicio expuesto representa un punto de entrada de nuevos ataques. (Ramaswamy Chandramouli y Zack Butcher, 2025)

Las vulnerabilidades relacionadas con los accesos no autorizados y fallas de autenticación y exposición de servicios pueden afectar la seguridad y poner en riesgo la seguridad y la protección de los recursos empresariales. Por tal motivo las arquitecturas modernas requieren de mecanismos o soluciones de seguridad orientadas al fortalecimiento de la autenticación y autorización dentro de los servicios expuestos por las APIs.

2.2.4. Zero trust

El modelo Zero Trust es un mecanismo de seguridad orientado a garantizar la protección de sistemas, usuarios y recursos dentro de la organización mediante controles estrictos de acceso y verificación. Este modelo se basa en el principio “Nunca confiar, siempre verificar”, este modelo indica que ningún elemento, ya sea usuario, dispositivo o servicio dentro o fuera de la red debe ser considerado confiable de manera predeterminada. Lo que quiere decir que todas las solicitudes de acceso deben pasar por proceso de verificación y validación de forma constante antes de autorizar el acceso a los recursos del sistema. (Scott Rose, Oliver Borchert, Stu Mitchell, Sean Connelly, 2020)

Este modelo se diferencia de los modelos tradicionales porque prioriza la validación continua de autenticación y autorización constante de usuarios y servicios sobre los recursos, mientras que otros confían en la red interna. Zero Trust reconoce que ninguna zona de la red es completamente segura, es decir que los riesgos pueden originarse tanto dentro como fuera de la red. Por ello, este modelo exige validaciones y controles continuos de seguridad en todos los acceso y comunicaciones. Por lo que muchas organizaciones han optado por este modelo con el objetivo de fortalecer la seguridad y la protección de recursos en entornos e infraestructuras modernas basadas en APIs y microservicios.

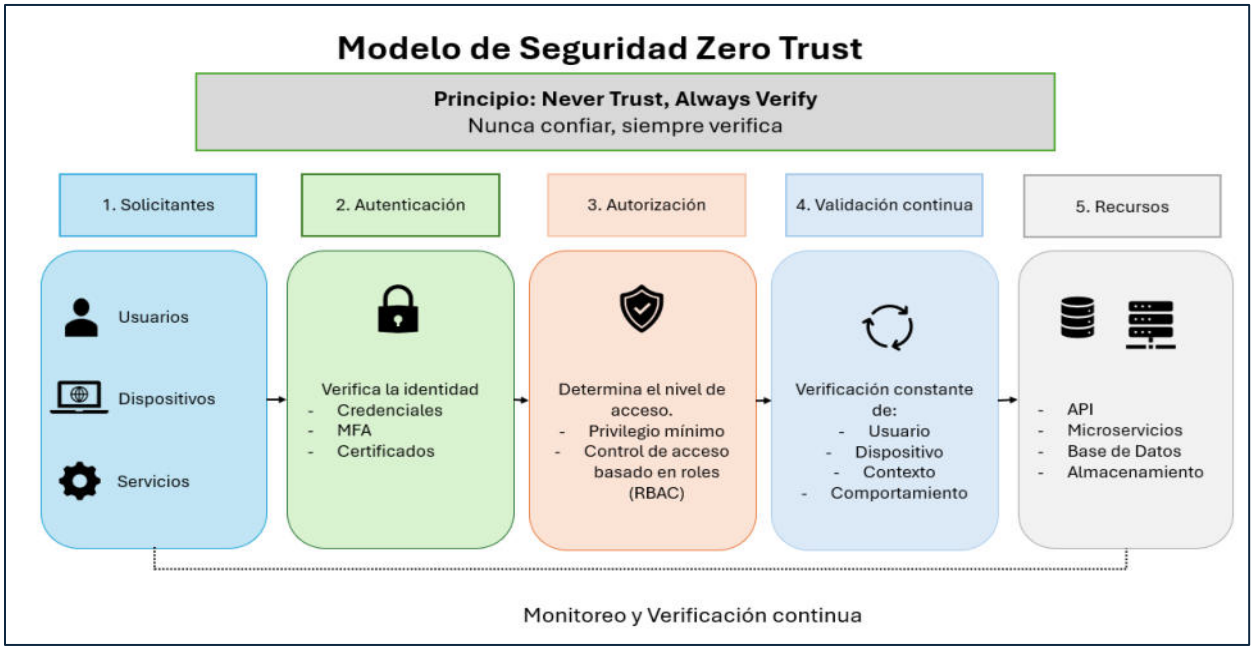
Este enfoque incorpora el principio de privilegio mínimo, es decir limita el acceso de usuarios y servicios únicamente a los recursos estrictamente necesarios para el cumplimiento de sus funciones. Además, este modelo promueve y aplica controles de segmentación y monitoreo continuo de las actividades que se realizan dentro de la infraestructura, minimizando riesgos asociados con accesos no autorizados y movimiento lateral dentro de la red y filtración de datos, permitiendo reducir amenazas y protegiendo información sensible.

Esquema de seguridad para APIs.

Debido al aumento de microservicios y servicios en la nube, Zero Trust se ha convertido en uno de los enfoques de seguridad más usado para proteger arquitecturas modernas basadas en APIs. Esto se debe a que las APIs facilitan la comunicación entre servicios, por lo que es necesario mecanismos de seguridad que permitan fortalecer la autenticación, control de acceso y la protección de datos. (Gregg Lindemulder, Matthew Kosinski, 2024)

Figura 4.

Modelo Zero Trust: flujo de validación continúa



Nota. En la presente figura se hace una representación sobre el flujo de validación continúa aplicado dentro del enfoque Zero Trust.

2.2.5. Autenticación y autorización.

La autenticación y autorización son mecanismos esenciales para la seguridad informática, permitiendo la seguridad y administración de accesos en sistemas actuales. (IBM, 2024) La autenticación es el proceso permitirá validar y confirmar la identidad de quienes intentan acceder

a un sistema o recurso, este proceso es realizado mediante credenciales de acceso como nombre de usuario, contraseña, tokens o certificados digitales.

Por otro lado, la autorización se encarga de establecer los privilegios otorgados a usuarios o servicios, es decir que acciones pueden realizar dentro de una aplicación. Esto permite prevenir accesos indebidos a información sensible del sistema, limitando el acceso a recursos conforme a nivel de privilegio asignado. A diferencia de la autenticación, que, valida la identidad del usuario, la autorización se encarga de controlar que acciones puede realizar dentro del sistema. La combinación de ambos procesos constituye la base para la implementación del control de acceso mediante Keycloak y tokens JWT en le arquitectura de propuesta.

De la misma forma, la gestión de identidades y acceso (IAM) representa un enfoque de seguridad, permitiendo controlar la autenticación, autorización y control de acceso mediante políticas y tecnologías aplicadas dentro aplicaciones y servicios. Los sistemas IAM ayudan a controlar identidades y niveles de acceso de forma centralizada, facilitando la administración de usuarios, roles y permisos en entornos compuestos por múltiples APIs y microservicios. (Matthew Kosinski y Amber Forrest, 2024)

Dentro de las arquitecturas modernas orientas a APIs y servicios distribuidos, los mecanismos de autenticación y autorización desempeñan un papel importante en la protección de recursos empresariales y validación de usuarios y servicios internos. En este contexto, la implementación de una solución de gestión de identidades y acceso contribuye al control de usuarios, roles y permisos entre los diversos microservicios y APIs que forman parte de la solución propuesta (Auth0, 2024).

Tabla 2.

Diferencias entre autenticación y autorización.

Característica	Autenticación	Autorización
Objetivo	Verificar la identidad del usuario, dispositivo o servicio	Determinar los privilegios y recursos disponibles para el usuario.
Información General	¿Quién eres?	¿Qué puedes hacer?
Función	Validar credenciales y confirmar identidad	Controlar el acceso mediante roles, permisos y políticas.
Elementos usados	Usuario, contraseña, token, certificado o MFA	Roles, privilegios, políticas y reglas de acceso.
Momento de aplicación	Se realiza antes del acceso del sistema.	Se lleva a cabo después de validar exitosamente la identidad del usuario.
Resultado	Permite validar la identidad del usuario.	Define los privilegios de acceso al usuario a recursos permitidos.
Ejemplo	Un usuario inicia sesión mediante credenciales.	El sistema controla los recursos y/o módulos a los que puede acceder el usuario.

Nota. Adaptado de (Auth0, 2024) e (IBM, 2024).

2.2.6. OAuth 2.0 y OpenID connect.

OAuth 2.0 es un protocolo de autorización que permite a una aplicación obtener acceso limitado a recursos protegidos en nombre de un usuario, sin necesidad de compartir directamente

sus credenciales. Este estándar se ha convertido en uno de los mecanismos más utilizados en arquitecturas modernas basadas en APIs y microservicios debido a su flexibilidad, escalabilidad y compatibilidad con servicios distribuidos. (Auth0, 2024)

El funcionamiento de OAuth 2.0 se basa en la emisión de tokens de acceso por parte de un servidor de autorización o proveedor de identidad (Identity Provider). Estos tokens son utilizados posteriormente por las aplicaciones cliente para consumir recursos protegidos dentro de una API. De esta forma, se evita el uso directo de usuarios y contraseñas en cada solicitud, reduciendo riesgos relacionados con la exposición de credenciales.

Dentro de OAuth 2.0 intervienen principalmente los siguientes componentes:

- Resource Owner: usuario propietario de la información o recurso protegido.
- Client: aplicación que solicita acceso al recurso.
- Authorization Server: servidor encargado de autenticar al usuario y emitir los tokens.
- Resource Server: API o servicio que valida el token y entrega el recurso solicitado.

OAuth 2.0 define varios flujos de autorización, conocidos como Grant Types, los cuales se utilizan dependiendo del tipo de aplicación y nivel de seguridad requerido. Entre los más utilizados se encuentran:

- Authorization Code Flow: utilizado en aplicaciones web y móviles. Permite obtener un código temporal que posteriormente se intercambia por un token de acceso.
- Client Credentials Flow: utilizado para comunicación entre servicios o microservicios sin intervención directa del usuario.

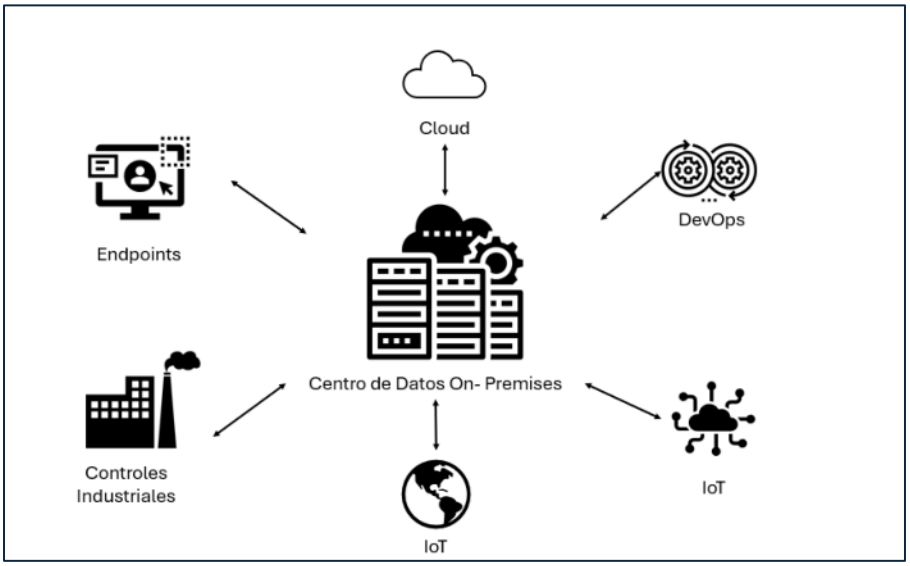
- Refresh Token Flow: permite renovar tokens expirados sin volver a solicitar autenticación al usuario.
- Authorization Code con PKCE: extensión orientada a aplicaciones móviles y clientes públicos para evitar robo de códigos de autorización.

En arquitecturas modernas orientadas a microservicios, OAuth 2.0 se complementa con OpenID Connect (OIDC), el cual añade capacidades de autenticación sobre OAuth 2.0. Mientras OAuth 2.0 se enfoca principalmente en autorización, OpenID Connect permite verificar la identidad del usuario autenticado mediante el uso de un ID Token en formato JWT. (OpenID Foundation, 2023)

OpenID Connect incorpora endpoints estandarizados para autenticación, descubrimiento de configuración, administración de sesiones y obtención de información del usuario. Esto facilita la integración de aplicaciones y APIs con proveedores de identidad modernos.

Figura 5.

Flujo de autenticación y autorización mediante OAuth 2.0 y OpenID Connect



Dentro del presente proyecto se utilizará Keycloak como proveedor de identidad (Identity Provider), debido a que esta herramienta soporta de manera nativa OAuth 2.0 y OpenID Connect. Keycloak permitirá administrar usuarios, roles, permisos y procesos de autenticación centralizados, además de emitir tokens JWT firmados digitalmente para el consumo seguro de APIs.

El uso conjunto de OAuth 2.0. (Hardt, 2012), OpenID Connect y Keycloak permite implementar un modelo de autenticación centralizado, desacoplando la lógica de seguridad del código de negocio y fortaleciendo los mecanismos de control de acceso dentro de arquitecturas modernas basadas en APIs y microservicios.

2.2.7. JSON Web Token (JWT)

JSON Web Token (JWT) es un estándar abierto utilizado para transmitir información segura entre diferentes partes mediante objetos JSON firmados digitalmente. Los JWT son ampliamente utilizados en procesos de autenticación y autorización dentro de APIs y arquitecturas basadas en microservicios, debido a que permiten validar identidad, permisos y contexto de sesión de manera segura y eficiente. (Jones, JSON Web Token (JWT), 2015)

Un JWT (Auth0, 2024) está compuesto por tres partes principales separadas por puntos:

Tabla 3.

Partes principales de un JWT.

Partes de un JWT

Header	El Header contiene información relacionada con el tipo de token y el algoritmo criptográfico utilizado para la firma digital. Generalmente incluye valores como typ y alg.
Payload	contiene los claims o atributos asociados al usuario y a la sesión. Estos claims permiten transportar información necesaria para validar autenticación, autorización y contexto de seguridad. (Jones, JSON Web Token (JWT), 2015)
Signature	Corresponde a la firma digital generada a partir del Header, Payload y una clave criptográfica. Esta firma permite garantizar la integridad y autenticidad del token, evitando modificaciones no autorizadas.

Nota. Adaptado de (Auth0, 2024)

Los JWT utilizan claims estandarizados definidos por el RFC 7519 (Jones, JSON Web Token (JWT), 2015). Entre los principales claims utilizados se encuentran:

- iss (Issuer): identifica al emisor del token.
- sub (Subject): identifica al usuario o entidad autenticada.
- aud (Audience): define el destinatario válido del token.
- exp (Expiration Time): indica la fecha y hora de expiración.
- iat (Issued At): especifica cuándo fue emitido el token.
- nbf (Not Before): indica desde cuándo el token es válido.
- jti (JWT ID): identificador único del token utilizado para trazabilidad y prevención de reutilización.

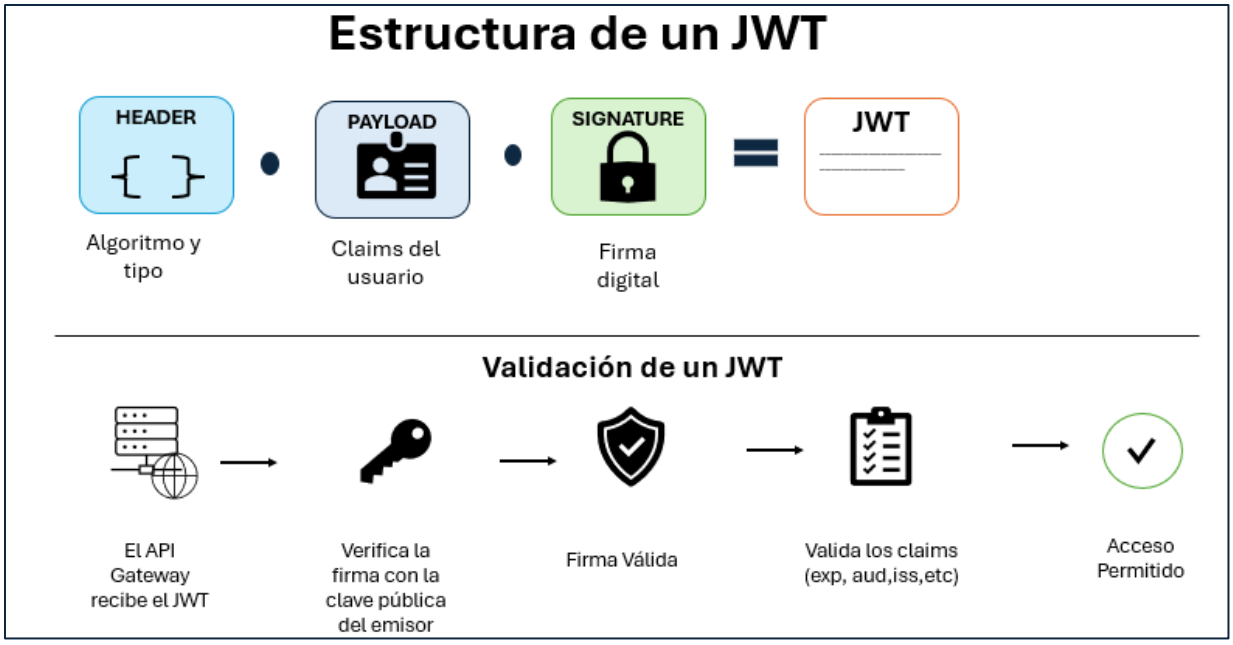
En el presente proyecto se utilizará el algoritmo RS256 para la firma de tokens JWT. RS256 es un algoritmo de firma asimétrica utilizando una clave privada para firmar el token y una clave pública para validar la firma. Este enfoque mejora significativamente la seguridad debido a que el componente que valida el token no necesita conocer la clave privada utilizada para generarlo. (Jones, JSON Web Token (JWT), 2015). El uso del RS256 se justifica ya que permite una mejor separación de responsabilidades y una administración más seguras de claves; en una arquitectura distribuida con múltiples microservicios RS256 elimina la necesidad de compartir la clave secreta entre todos los servicios. (Will Johson, 2022)

El uso de RS256 ofrece varias ventajas dentro de arquitecturas distribuidas:

- Mayor seguridad en la validación de tokens.
- Separación entre generación y validación de credenciales.
- Facilidad para integrar múltiples servicios y microservicios.
- Reducción del riesgo asociado a la exposición de claves secretas compartidas.

Figura 6.

Estructura de un JSON Web Token (JWT)



Nota. Adaptado de (Keycloak, 2024) y (Jones, JSON Web Token (JWT), 2015)

En el modelo propuesto, Keycloak será el encargado de emitir los tokens JWT firmados digitalmente, mientras que el API Gateway y los microservicios validarán la autenticidad de dichos tokens utilizando la clave pública correspondiente. Esto permitirá aplicar controles de autenticación, autorización y validación continua bajo principios de Zero Trust.

2.2.8. API Gateway

Un API Gateway es un componente de infraestructura que actúa como punto central de entrada para todas las solicitudes dirigidas hacia los servicios backend o microservicios. Su principal función consiste en interceptar, validar, controlar y enrutar las solicitudes realizadas por clientes externos hacia los servicios internos de la organización. (Madden, 2020)

Dentro de arquitecturas modernas basadas en microservicios, el API Gateway cumple un rol fundamental debido a que centraliza los mecanismos de seguridad y administración del tráfico, evitando que los microservicios estén expuestos directamente hacia Internet.

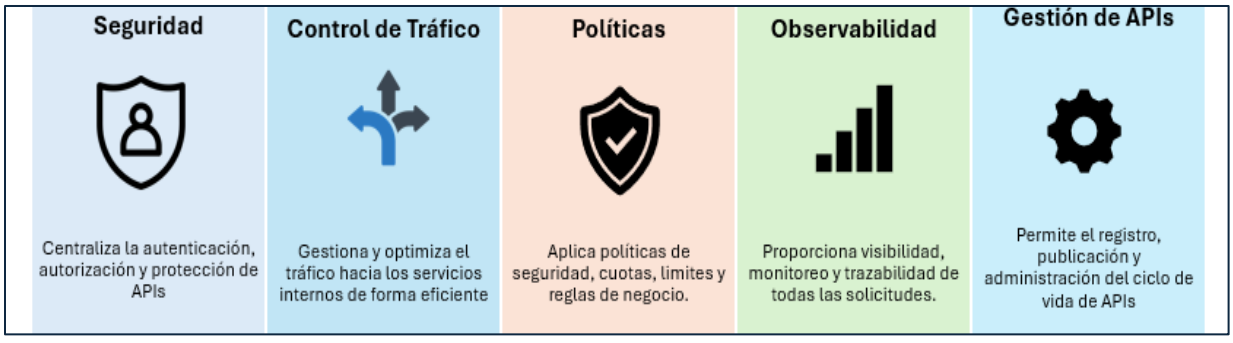
El funcionamiento general de un API Gateway consiste en recibir las solicitudes provenientes de clientes o aplicaciones externas, validar los mecanismos de autenticación y autorización, aplicar políticas de seguridad y posteriormente reenviar únicamente las solicitudes válidas hacia el servicio correspondiente.

Entre las principales funcionalidades de un API Gateway destacan:

- Validación de tokens JWT.
- Aplicación de políticas de autenticación y autorización.
- Control de tráfico y rate limiting.
- Balanceo de carga.
- Registro y auditoría de eventos.
- Protección perimetral de APIs.
- Transformación y enrutamiento de solicitudes.

Figura 7.

Funciones del API Gateway



Uno de los controles más importantes implementados por un API Gateway es la validación de tokens JWT. Este proceso permite verificar que el token haya sido emitido por un proveedor confiable, que la firma digital sea válida y que el token no haya expirado o sido manipulado.

Asimismo, el API Gateway permite aplicar mecanismos de rate limiting, los cuales limitan la cantidad de solicitudes que un cliente puede realizar dentro de un intervalo de tiempo determinado. Este mecanismo ayuda a prevenir abusos, ataques de denegación de servicio y consumo excesivo de recursos.

Otra ventaja importante consiste en la centralización de políticas de seguridad. En lugar de implementar controles individuales dentro de cada microservicio, el API Gateway permite administrar autenticación, autorización y validaciones desde un único punto de control, mejorando la consistencia y reduciendo errores de configuración. (Madden, 2020)

En el presente proyecto se utilizará WSO2 API Manager como API Gateway principal (WSO2, 2024), los módulos específicos de WSO2 son los siguientes: API Manager Gateway que realiza la validación de los tokens JWT, ek Throttling Engine que aplica las políticas de rate limiting, el Publisher y Developer Portal que gestiona las políticas de acceso por API y finalmente el sistema de Analytics que proporciona el registro y auditoría de todos los eventos de acceso. (WSO2, 2024)

La incorporación del API Gateway dentro de la arquitectura propuesta fortalece significativamente la seguridad perimetral y permite implementar controles alineados con principios de Zero Trust y defensa en profundidad.

2.2.9. Keycloak

Keycloak es una plataforma de gestión de identidades y accesos (Identity and Access Management – IAM) de código abierto orientada a la administración centralizada de autenticación, autorización y control de acceso dentro de aplicaciones modernas y arquitecturas distribuidas. (Keycloak, 2024)

Esta herramienta permite administrar usuarios, roles, grupos,cliente, permisos y políticas de seguridad desde una plataforma centralizada, siguiendo los siguientes conceptos clave: Realm que es el espacio que aísla usuarios, roles, clientes y políticas de seguridad; un cliente representa una aplicación o un servicio que interactúa con Keycloak para la autenticación; los Roles son los permisos de acceso que se puedan asignar a los usuarios o grupos y los Tokens son los elementos JWK que Keycloak emite para la autenticación y autorización, facilitando la integración segura entre aplicaciones, APIs y microservicios. Además, soporta protocolos estándares ampliamente utilizados como OAuth 2.0, OpenID Connect y SAML. (Auth0, 2024)

Uno de los principales beneficios de Keycloak es la autenticación federada, la cual permite integrar múltiples proveedores de identidad externos, como Google, Microsoft Active Directory, LDAP u otros sistemas corporativos. Esto facilita que los usuarios puedan autenticarse utilizando identidades previamente existentes dentro de la organización. (IBM, 2024); (Keycloak, s.f.)

Keycloak también permite implementar mecanismos avanzados de seguridad, tales como:

- Single Sign-On (SSO).
- Autenticación multifactor (MFA).
- Gestión centralizada de sesiones.
- Administración de roles y permisos.

- Federación de identidades.
- Emisión y validación de tokens JWT.

Tabla 4.

Características Keycloack

Característica	Descripción
Single Sign-On (SSO)	Permite a los usuarios autenticarse una sola vez para acceder a múltiples aplicaciones integradas.
Autenticación Multifactor (MFA)	Soporta mecanismos adicionales de verificación como OTP, TOTP, WebAuthn y SMS para fortalecer la seguridad del acceso.
Gestión Centralizada	Facilita la administración de usuarios, roles, grupos, permisos y políticas desde una consola centralizada.
Estándares Abiertos	Compatible con protocolos y estándares de identidad como OAuth 2.0, OpenID Connect (OIDC), SAML 2.0 y UMA.
Tokens JWT	Emite y valida tokens JWT firmados digitalmente para autenticación y autorización segura dentro de APIs y servicios.
Federación de Identidades	Permite integrar proveedores de identidad externos y directorios corporativos como Lightweight Directory Access Protocol (LDAP) protocolo para acceder y administrar servicios del directorio, Active Directory (Microsoft, 2025) y Microsoft Entra ID (anteriormente Azure Active Directory o Azure AD) servicios de gestión de identidades y acceso basado en la nube. (Microsoft, 2025)

Dentro del flujo de autenticación Keycloak actúa como Identity Provider (IdP). Cuando un usuario intenta acceder a una API protegida, Keycloak valida las credenciales y genera un token JWT firmado digitalmente. Posteriormente, este token será utilizado para consumir recursos protegidos a través del API Gateway.

La integración de Keycloak con OAuth 2.0 y OpenID Connect permite desacoplar completamente la lógica de autenticación y autorización del backend de negocio, facilitando la administración centralizada de seguridad dentro de arquitecturas basadas en microservicios.

En el presente proyecto, Keycloak será utilizado como componente principal para:

- Administrar usuarios y roles.
- Emitir tokens JWT firmados con RS256.
- Controlar autenticación y autorización.
- Integrarse con el API Gateway WSO2.
- Aplicar principios de mínimo privilegio y Zero Trust.

La utilización de Keycloak dentro de la arquitectura propuesta permite fortalecer la protección de APIs, mejorar la trazabilidad de accesos y reducir riesgos asociados con autenticación insegura o gestión descentralizada de credenciales.

2.2.10. Gestión de identidades privilegiadas y credenciales en entornos cloud.

Los principales desafíos para la seguridad de la identidad digital incluyen la fragmentación de los equipos de infraestructura y seguridad, el crecimiento de las identidades humanas y no humanas, permisos que se extienden a través de los diferentes entornos de nube, por cada persona física se estima que tiene al menos 10 identidades digitales (CyberArk, 2024).

Para la protección de secretos y credenciales se debe tener un enfoque centralizado para la seguridad de la identidad y hacer realidad el concepto de Zero Trust, con el monitoreo continúo y la auditoría de los accesos, siempre aplicando el concepto de mínimo privilegio en las organizaciones (CyberArk, 2024). La siguiente figura muestra un resumen de donde podemos encontrar las diferentes identidades, las podemos encontrar desde dispositivos de red hasta desarrollo de DevOps, Cloud, servicios SaaS y más:

Figura 8.

Fuentes de identidades en un esquema Zero Trust

Nota. Adaptado de (Oracle, 2024)

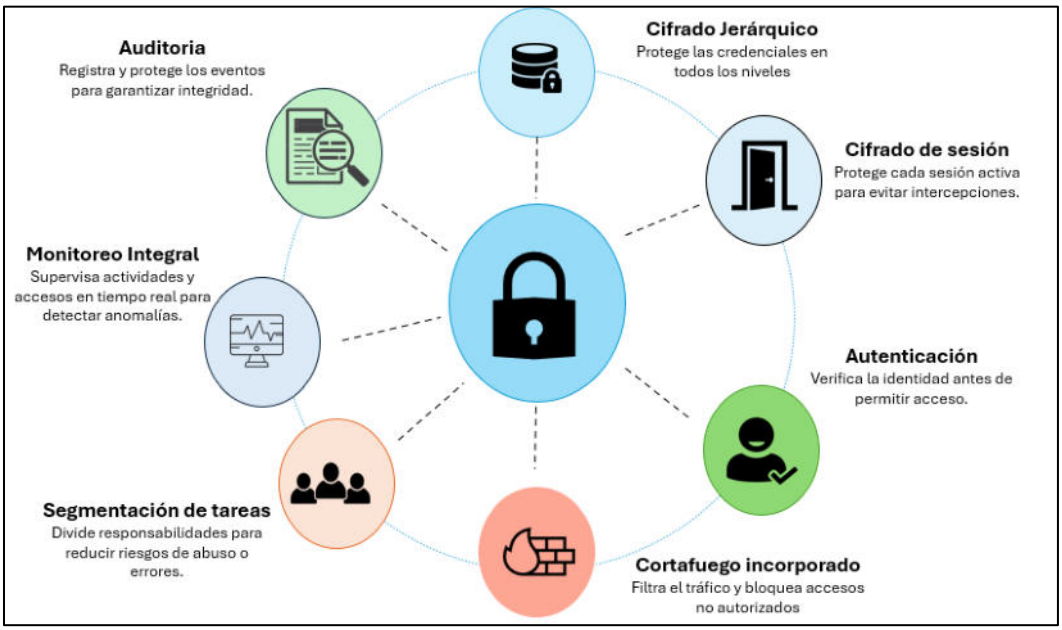
Las identidades privilegiadas como son la cuenta root en servidores Linux, administrator en servidores Windows, o el usuario SA en base de datos SQL, son las que poseen los más altos privilegios en un sistema. El acceso no autorizado a este tipo de cuentas permite comprometer a

una organización y a sus sistemas críticos como: aplicaciones web, base de datos, firewalls, dispositivos de red y la nube. Los sistemas que fueron creados para proteger y auditar estos usuarios se llaman Privileged Access Management (PAM) que es la solución de seguridad que permite controlar y supervisar estás cuentas con privilegios (CyberArk, 2024).

Adicionalmente, las cuentas de servicio o de integración que son utilizadas por ejemplo para que la aplicación web de la organización tengo acceso a la información de la base de datos, estas integraciones muchas veces usaban credenciales embebidas en el código fuente, que no es otra cosa que colocar el usuario y contraseña de esta integración como texto plano en el código fuente de la aplicación. Esto obviamente representa un grave riesgo de seguridad y ocasiona que las aplicaciones sean inseguras. La forma de protección de estas credenciales de integración o secretos es utilizar herramientas que permitan utilizar APIs de forma segura y guardar las credenciales en un entorno seguro, así como los accesos privilegiados a las cuentas de administración de AWS por ejemplo (CyberArk, s.f.).

Figura 9.

Estructura de una bóveda de almacenamiento de credenciales



Nota. Adaptado de CyberArk (CyberArk, s.f.).

Las herramientas de Privileged Access Management (PAM) contienen bóvedas donde se almacenan los datos de las cuentas privilegiadas, y de los secretos de las cuentas de servicio, estás bóvedas cuentan con diferentes capas de protección como son cifrados de las sesiones, cifrado jerárquico para las cuentas, cortafuegos, segregación de tareas, monitoreo integral, la figura anterior muestra en resumen las diferentes capas de protección que se tienen en estas bóvedas.

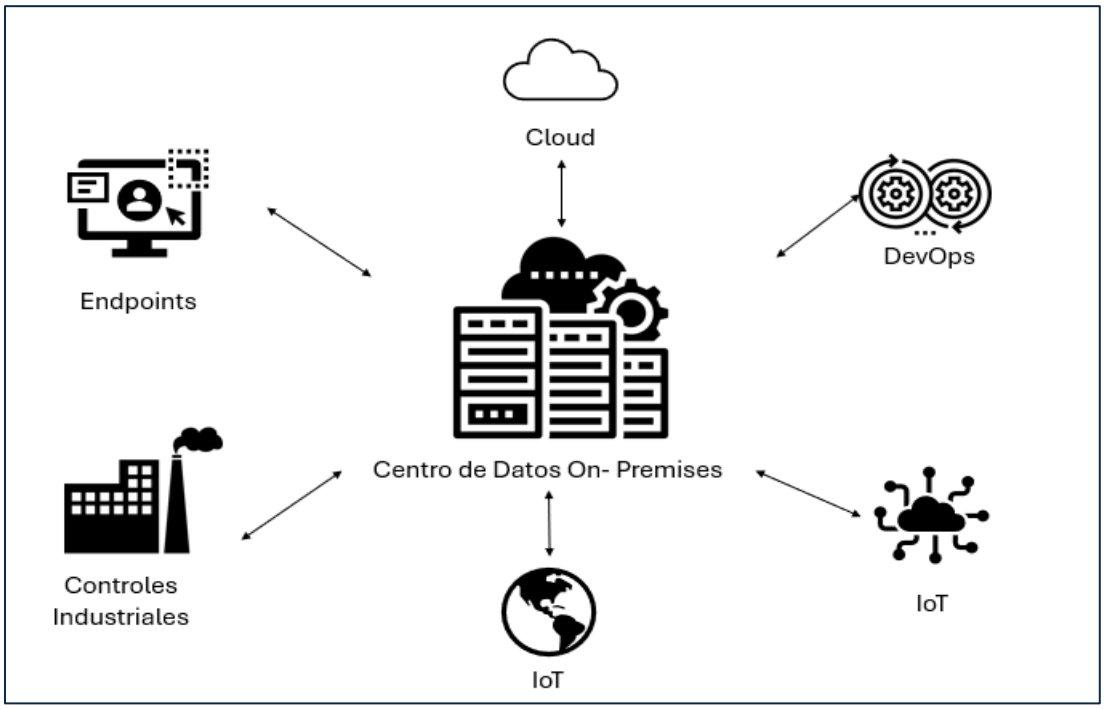
2.2.11. CyberArk

En sus inicios CyberArk nació como una empresa de protección de identidades privilegiadas como PAM, para proteger las cuentas privilegiadas como: administrator en Windows y root de los sistemas Linux para entornos en premisa (CyberArk, 2024). Pero como se

muestra en la siguiente figura la superficie de identidades se han expandido y por ende la superficie de los ataques han aumentado de forma exponencial:

Figura 10.

Nuevas fuentes de identidades privilegiadas y no privilegiadas.



Nota. Adaptado de (CyberArk, 2024).

Tenemos más cuentas privilegiadas y de servicios en entornos de Cloud, DevOps, dispositivos finales, controles industriales, internet de las cosas y otros.

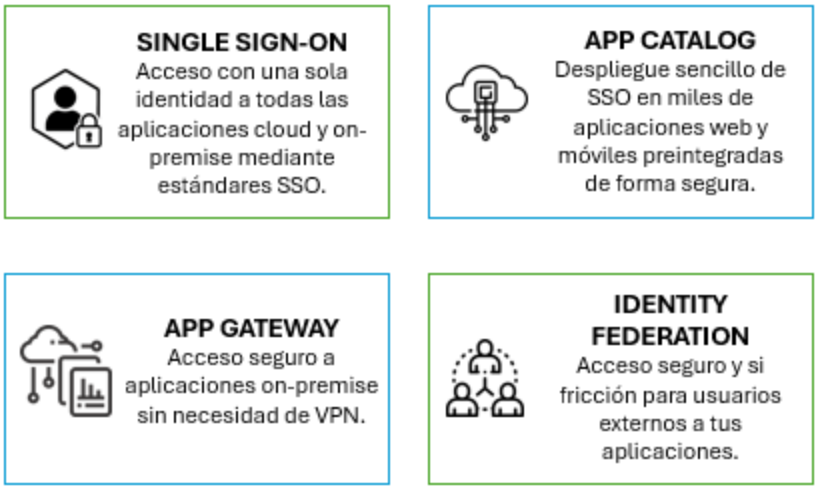
CyberArk es una herramienta que ha evolucionado para proteger las Identidades en cualquier lugar que estas existan, para lo cual ha desarrollado un producto llamado CyberArk Identity que permite la gestión de los accesos privilegiados o no dentro de una sola herramienta,

estos pueden estar en nube, en premisa o en aplicaciones web. Sus principales características incluyen autenticación multifactor adaptable, detección de riesgos basada en contextos, gestión de cunetas privilegiadas y protección de identidades en cloud como AWS, Azure y Google Cloud Platform (GCP) (CyberArk, s.f.).

La siguiente figura muestra un resumen de sus principales características:

Figura 11.

Portal de CyberArk Identity principales funciones



Nota. Adaptado de (CyberArk, 2024).

Una de sus principales características es que permite tener un esquema de multifactor adaptable, esto nos permite establecer niveles de riesgo y alerta en base a criterios como el tipo de sistema operativo del cual se realiza la conexión final, la hora del día, el día de la semana que se realiza la conexión, la geolocalización, por ejemplo, si un usuario se conecta al portal de AWS

de lunes a viernes en horario laboral desde Ecuador, y observa una conexión desde China a las 3 AM, es un factor de riesgo y podemos alertar y bloquear si es necesario este acceso.

Otro de los productos es el CyberArk Cloud Security, que permite la federación de los accesos privilegiados a plataformas de nube como AWS, Azure y Google Cloud, que permite el acceso a los diferentes recursos de la nube mediante una autenticación multifactor y con los mínimos privilegios, sus principales características son:

- **Single Sign On (SSO) y acceso centralizado:**

Accesos centralizados mediante un IdP externo que garantiza la existencia de cuentas privilegiadas únicas de gestión.

- **Mínimos privilegios:**

La configuración de políticas de acceso en CyberArk permite configurar roles con los mínimos privilegios que necesita la cuenta para la administración de los recursos de nube.

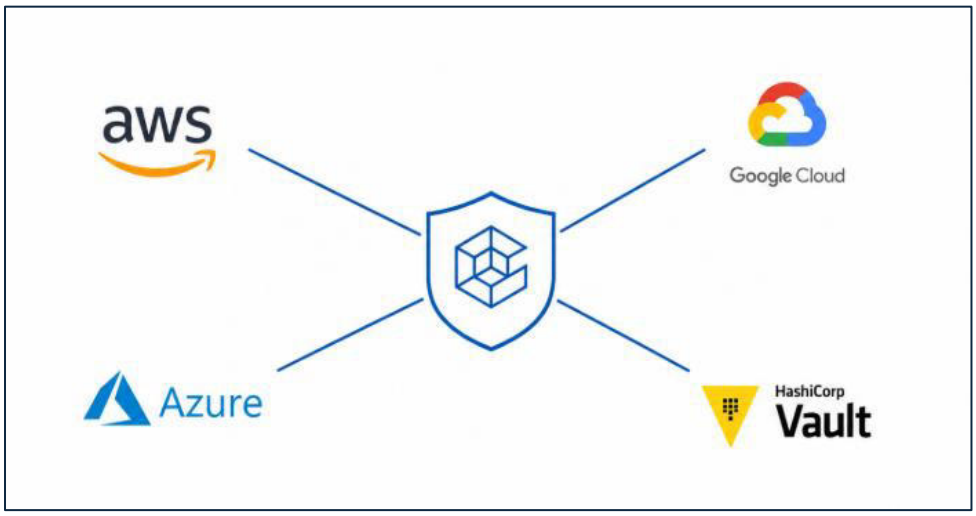
- **Monitoreo completo:**

Podemos conocer los accesos de los usuarios, los recursos de nube y los permisos que necesita para cumplir con sus funciones.

La siguiente figura muestra el resumen de la integración nativa entre CyberArk y los principales proveedores de nube:

Figura 12.

Proveedor de nube que se integran nativamente con CyberArk.



Nota. Adaptado de (CyberArk, s.f.).

2.2.12. Defensa en profundidad

La defensa en profundidad es un diseño de seguridad orientado a proteger sistemas y datos críticos de una organización, este abarca a las personas, la tecnología y las operaciones como un entorno que debe resistir a las amenazas tanto internas como externas, para esto necesitamos implementar el concepto de mínimos privilegios, la separación de responsabilidades entre diferentes sistemas de control, monitoreo y detección continua de los sistemas, y redundancia en los controles implementados (NIST SP, 2020).

En el modelo de protección de APIs propuesto en este proyecto lo que buscamos es la gestión de las identidades, autenticación, políticas de acceso y autorización, así como la auditoria del sistema. Las principales amenazas que buscamos mitigar están basadas en el OWASP Top 10 y son:

Tabla 5.

Amenazas de seguridad de APIs consideradas para la validación de la propuesta.

Amenazas de seguridad de APIs consideradas para la validación de la propuesta	
API1: Broken Object Level Authorization	Acceso no autorizado a objetos mediante manipulación de identificadores en peticiones.
API2: Broken Authentication,	Mecanismos de autenticación débiles o mal implementados que permiten la suplantación de identidad.
API3: Broken Object Property Level Authorization,	Exposición o modificación de propiedades de objetos sensibles sin controles adecuados.
API4: Unrestricted Resource Consumption	Ausencia de límites en peticiones que permite ataques de denegación de servicio o abuso de recursos.
API7: Server Side Request Forgery (SSRF)	El atacante induce al servidor a realizar peticiones a recursos internos no expuestos.
API8: Security Misconfiguration	Configuraciones por defecto inseguras, permisos excesivos y headers de seguridad ausentes.

En este contexto CyberArk actúa como la capa de control de acceso a los recursos de nube de AWS. La federación de autenticación de AWS no solo garantiza usuarios únicos de administración y SSO, sino también que las cuentas de administración tengan los mínimos privilegios, sin acceso a todos los recursos de nube.

Keycloud representa la categoría de soluciones de Key Management Service (KMS) y gestión de secretos orientadas a entornos cloud-native. En este marco teórico, Keycloud encapsula plataformas como AWS Secrets Manager, Azure Key Vault, o soluciones propietarias

con capacidades equivalentes, cuyo propósito central es la gestión del ciclo de vida de claves criptográficas, certificados, tokens y secretos de aplicación en infraestructuras distribuidas. Keycloud opera como la capa criptográfica del modelo de defensa, asegurando que toda comunicación sensible entre APIs esté protegida mediante cifrado de extremo a extremo. Su integración con API Gateways permite la validación automática de tokens JWT firmados, la gestión de certificados TLS para mutual TLS (mTLS) y la rotación transparente de claves sin interrupciones en el servicio.

Un API Gateway es un componente de infraestructura que actúa como punto de entrada único (Single Entry Point) para todas las solicitudes dirigidas a los servicios backend. Funcionalmente, se ubica entre los clientes consumidores y los microservicios, interceptando, inspeccionando y transformando el tráfico según políticas configuradas.

La sinergia de estos componentes no solo eleva significativamente el nivel de seguridad de las APIs, sino que también facilita el cumplimiento normativo, mejora la auditoría del sistema y reduce el tiempo de detección y respuesta ante incidentes. El modelo propuesto constituye una base sólida para el diseño de arquitecturas seguras en entornos empresariales modernos, adaptable tanto a infraestructuras de nube.

Tabla 6.

Análisis comparativo de un esquema de seguridad perimetral y una defensa en profundidad

Característica	Seguridad perimetral	Defensa en profundidad

Modelo de confianza	Confianza implícita en la red interna	Verificación explícita en cada capa bajo el enfoque Zero Trust
Gestión de identidades y accesos	Credenciales estáticas y administración local de usuarios	Acceso federado, permisos centralizados y aplicación del mínimo privilegio
Control de APIs	Control limitado, dependiente principalmente del firewall de red	Validación JWT, autorización por roles, rate limiting y políticas de acceso en el API Gateway
Respuesta ante brechas	Reactiva, con detección tardía de incidentes	Proactiva, con monitoreo, trazabilidad y registro de eventos de seguridad
Adaptabilidad	Limitada, principalmente orientada a entornos locales	Compatible con arquitecturas cloud, microservicios y entornos distribuidos

Como se observa en la Tabla 6 los esquemas anteriores de seguridad perimetral como implementación de firewalls, IDS o IPS dentro de una infraestructura de red no son suficientes para proteger las brechas actuales en cuanto a la seguridad de la información, ya que la superficie de ataque se ha incrementado. Los servicios de nube, DevOps, Internet of Things (IoT), Software as a Service (SaaS), accesos de usuarios fuera de la infraestructura y otros han incrementado el vector de ataque por lo que se debe implementar un esquema basado en la defensa en profundidad.

La efectividad del esquema de defensa en profundidad implementado en este proyecto se evaluará a través de los siguientes criterios: Tasa de bloqueo: porcentaje de ataques simulados correctamente bloqueados por cada capa (Keycloak, WSO2 y CyberArk); la Trazabilidad: porcentaje de eventos de acceso registrados en logs de auditoría; Aislamiento de capas: verificación de que una falla en una capa no compromete automáticamente las demás; y Tiempo de respuesta: latencia adicional introducida por los controles de seguridad en comparación con el acceso directo sin protección.

Es necesario implementar un esquema de defensa en profundidad ya que la protección por capas permite tener mayor visualización sobre lo que estamos protegiendo, y en cada capa tenemos diferentes modos de protección, esta segregación de funciones es lo que garantiza que no se comprometa todo el sistema de protección en caso de una falla o una infiltración dentro del sistema.

CAPÍTULO 3:

3. DESARROLLO

3.1.Desarrollo del Trabajo

La arquitectura general establece un flujo de seguridad que está orientado a garantizar los procesos de autenticación, autorización y protección de credenciales. El proceso inicia cuando el usuario o sistema solicita autenticarse mediante Keycloak, mediante el cual se encarga de validar las credenciales y emitir un token JWT firmado digitalmente. Este token ya contiene información necesaria para poder identificar al usuario y permite controlar los permisos asignados durante la sesión.

Las solicitudes dirigidas a los servicios serán llamados a través de WSO2 API Manager que es responsable de realizar la validación de autenticidad, su vigencia, la audiencia definida y permisos que fueron asignados al usuario antes de reenviar la solicitud al microservicio correspondiente y una vez realizada estas verificaciones, la petición es reenviada al microservicio correspondiente.

El acceso al portal de administración de AWS se establece con CyberArk Cloud Security que gestiona el acceso federado al entorno de nube, permitiendo centralizar el control de accesos a recursos cloud, con los privilegios mínimos necesarios.

3.1.1. Flujo de seguridad

El flujo inicia cuando el usuario se autentica en Keycloak y obtiene un token JWT que contiene la identidad del solicitante junto con los roles y permisos asignados para la sesión, luego la solicitud llega al WSO2 Gateway donde se valida el token antes de permitir el acceso a las APIs. Si la validación es correcta, la petición pasa al servicio desarrollado en Spring Boot, el cual revisa nuevamente la autorización y ejecuta la operación correspondiente. El servicio consulta o guarda la información en PostgreSQL, mientras que CyberArk se utiliza para el acceso federado a AWS.

3.1.2. Esquema de procedimientos de las tecnologías seleccionadas

En el presente capitulo se describe el desarrollo de la propuesta que está orientada al fortalecimiento de la seguridad en arquitecturas basadas en APIs y microservicios, mediante la implementación de una solución que está integrada por Keycloak, WSO2 Gateway, Spring Boot, PostgreSQL y acceso federado al portal de AWS a través de CyberArk Cloud Security.

Se describen los procedimientos que fueron realizados en la configuración e integración de cada componente dentro de la arquitectura propuesta, así como el flujo de seguridad implementado y las pruebas efectuadas para validar su funcionamiento.

- **Keycloak**

Crea y administra los usuarios, define sus roles, establece los alcances (scopes) de lo que pueden hacer y emite un token JWT que acredita su identidad.

- **WSO2 Gateway**

Actúa como puerta de entrada. Valida que el JWT sea auténtico y aplica las políticas de acceso configuradas; si se evidencia inconsistencia, la petición ni siquiera llega a los servicios internos.

- **Spring Boot**

Toma el control cuando la solicitud supera el gateway. Vuelve a comprobar el token, revisa los roles y scopes, y registra todo en los logs de auditoría para mantener trazabilidad completa de quién hizo qué y cuándo.

- **CyberArk**

Gestiona el acceso federado al portal de AWS mediante integración con IAM Identity Center, permitiendo centralizar el control de accesos a recursos cloud bajo políticas de mínimo privilegio, sin el uso de credenciales estáticas.

- **PostgreSQL**

Almacena los recursos de la aplicación, los eventos que se van generando y los resultados de las pruebas, manteniendo los datos seguros, organizados y disponibles para su consulta.

3.1.3. Creación de las APIs con Spring Boot

En el presente proyecto se demostrará estos cinco fundamentos:

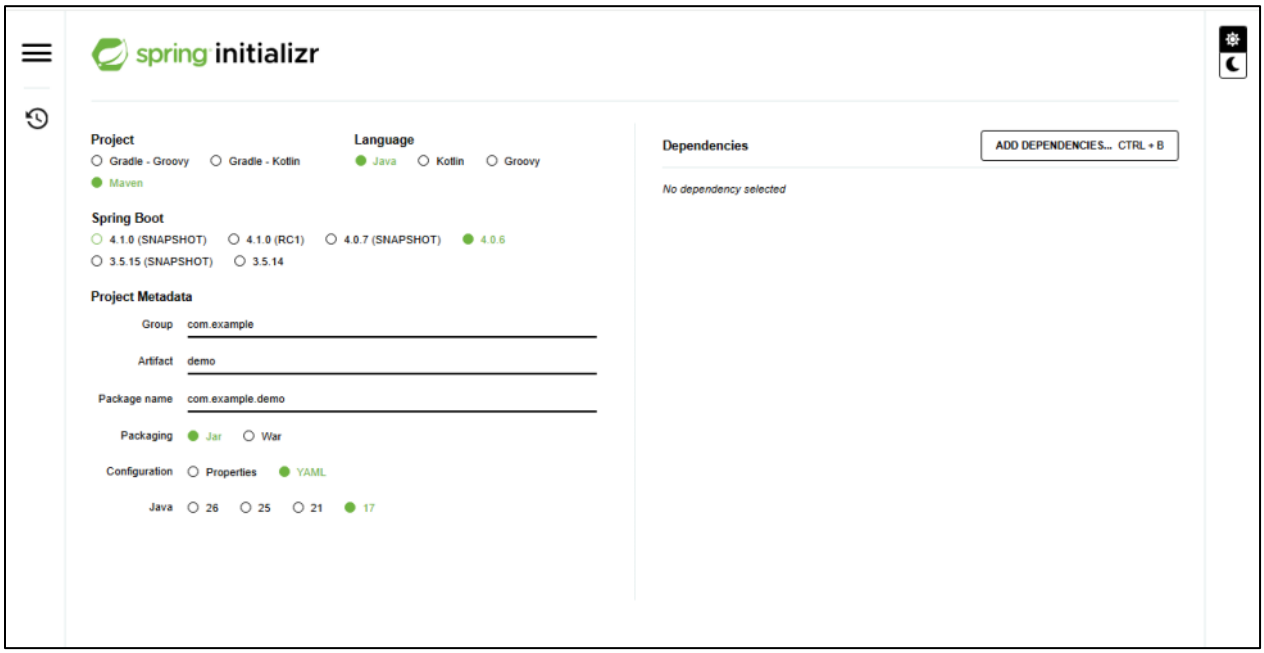
1. Autenticación con JWT
2. Autorización por roles y scopes
3. Validación de paso por API Gateway
4. Acceso federado al portal de AWS a través de CyberArk Cloud Security
5. Evidencia y métricas de ataques controlados

Con base en estos fundamentos, el desarrollo del proyecto en Spring Boot se realizará de la siguiente manera:

- Creación del proyecto con Spring Initializr

Figura 13.

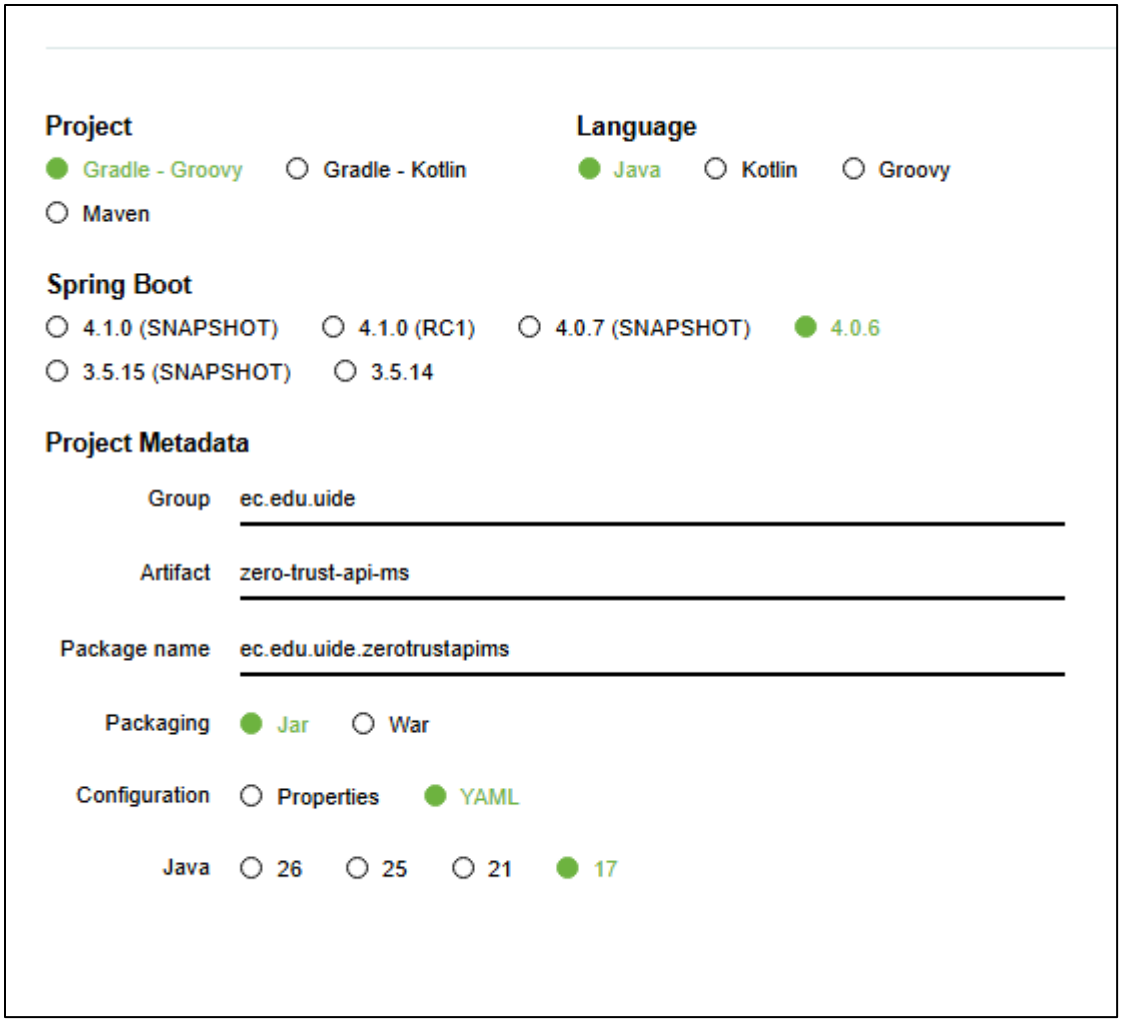
Interfaz de configuración de un proyecto Spring Boot mediante Spring Initializr



Para la implementación del proyecto zero-trust-api-ms se seleccionó Java 11 debido a su compatibilidad con el entorno de laboratorio y con las dependencias necesarias para integrar los componentes de seguridad. Se utilizó Gradle como herramienta de construcción porque permite administrar de forma ordenada las librerías del proyecto y facilita la configuración de versiones. Además, se eligió Spring Boot 2.7.18 porque mantiene compatibilidad con Java 11 y permite trabajar con dependencias relacionadas con seguridad, autenticación mediante JWT e integración con servicios externos. Debido a que Spring Initializr ofrece versiones más recientes por defecto, la configuración del proyecto será ajustada manualmente para trabajar con las versiones seleccionadas.

Figura 14.

Configuración de metadatos y parámetros del proyecto en Spring Initializr



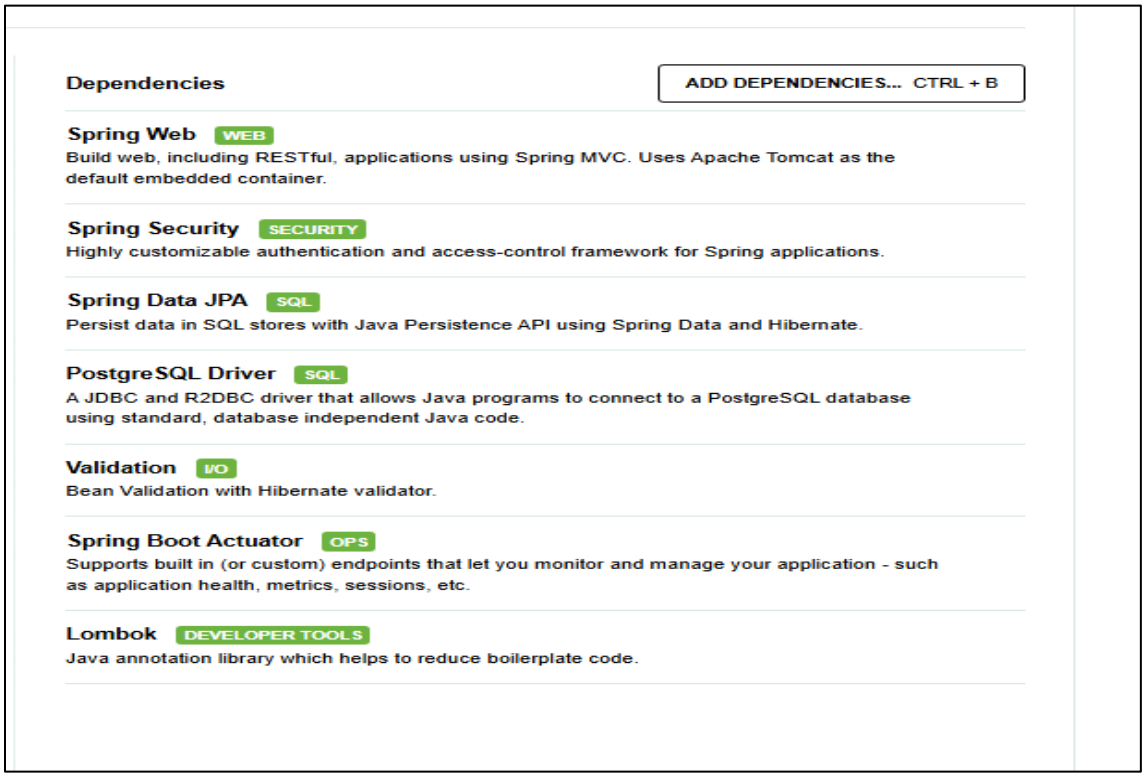
Las dependencias que principalmente se van a utilizar son:

- Spring Web
- Spring Security
- OAuth2 Resource Server
- Spring Data JPA

- PostgreSQL Driver
- Validation
- Actuator
- Lombok

Figura 15.

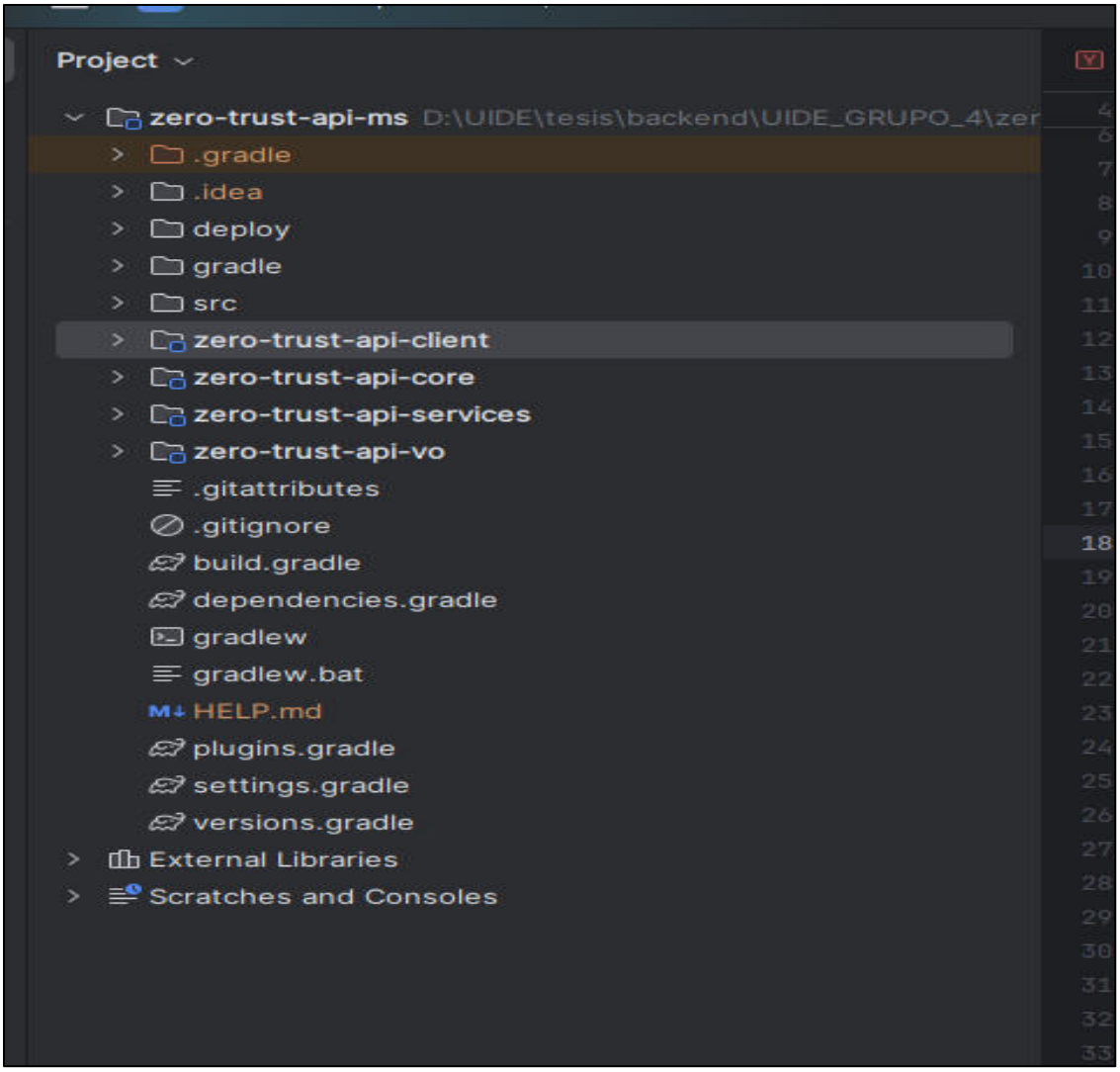
Selección de dependencias del proyecto en Spring Initializr



El proyecto creado se importó en el IDE IntelliJ IDEA, el cual cuenta con varios plugins integrados que facilitarán la creación y gestión de APIs.

Figura 16.

Estructura del proyecto Spring Boot en entorno de desarrollo (IntelliJ IDEA)



Se utilizó una estructura basada en microservicios en donde cada módulo cuenta con sus propios componentes y responsabilidades. Esta organización permite mantener el proyecto más ordenado, facilitar el mantenimiento del código y separar mejor la lógica de cada servicio.

De manera global, también se creó una carpeta llamada deploy, destinada a contener los archivos necesarios para el despliegue del proyecto en AWS. Esto ayuda a centralizar la configuración relacionada con la publicación y ejecución del sistema en la nube.

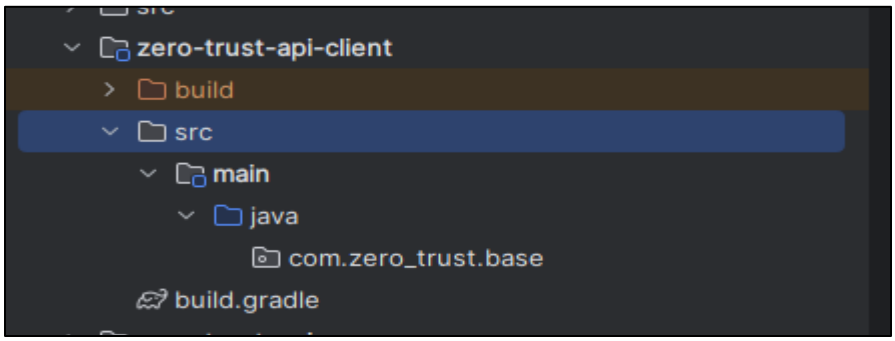
Figura 17.

Estructura de despliegue del proyecto con Docker

Como se puede observar en la estructura del proyecto, se manejan varios módulos separados, cada uno con una función específica dentro del microservicio zero-trust-api-ms. Esta organización permite dividir mejor las responsabilidades del sistema y facilita el mantenimiento del código.

Figura 18.

Estructura del módulo cliente del proyecto Spring Boot

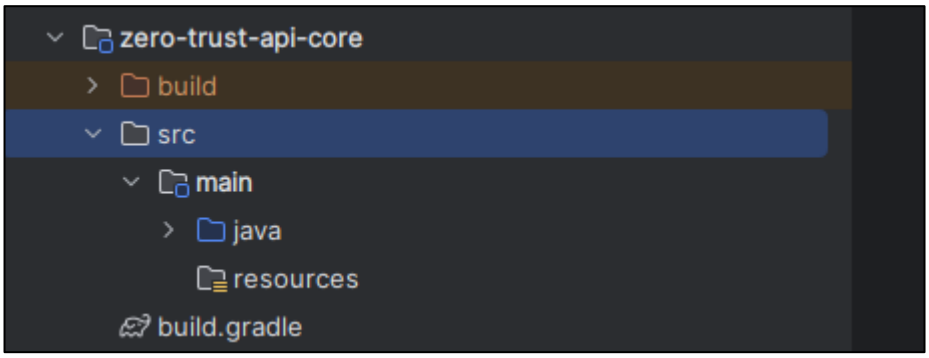


El primer módulo es zero-trust-api-client. En este módulo se crean las entidades relacionadas con la base de datos, las cuales posteriormente permitirán realizar las respectivas

consultas. Además, se definen los repositorios mediante interfaces, donde se declaran los métodos que luego serán utilizados por los demás módulos para la creación de las APIs.

Figura 19.

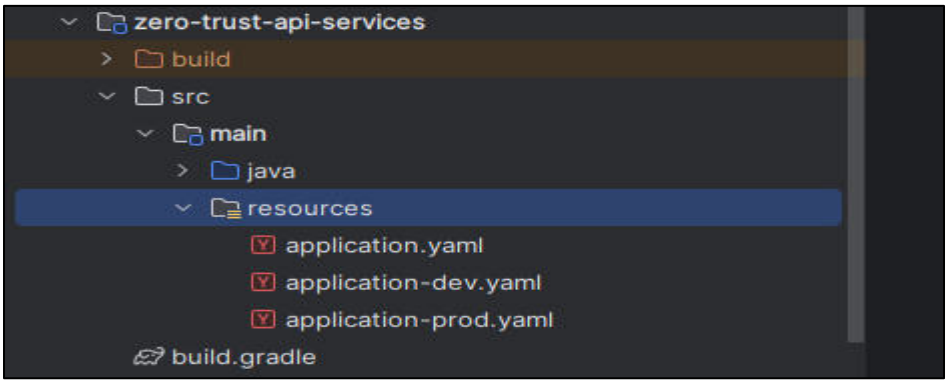
Estructura del módulo núcleo (core) del proyecto Spring Boot



El segundo módulo es zero-trust-api-core. En este se implementa la lógica principal del sistema, tomando como base los métodos definidos anteriormente en el módulo client. Aquí se manejan tanto los repositorios como los servicios principales en Java, permitiendo centralizar la lógica de negocio del proyecto.

Figura 20.

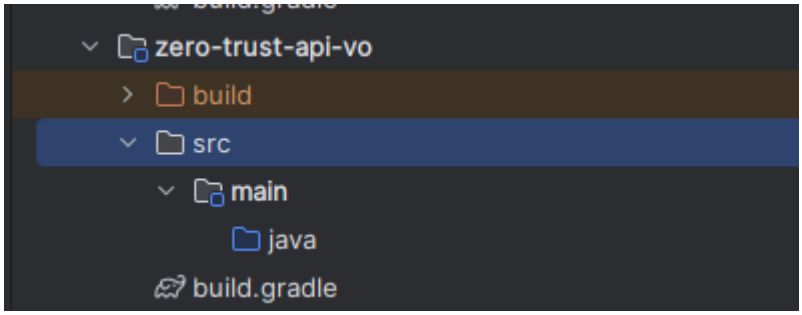
Estructura del módulo de servicios y archivos de configuración del proyecto Spring Boot



El tercer módulo es zero-trust-api-services. Este módulo se encarga de exponer la funcionalidad del sistema mediante APIs, utilizando los métodos creados y consumidos desde el módulo core. Además, este módulo es el encargado de la ejecución principal de la aplicación.

Figura 21.

Estructura del módulo de objetos de valor (VO) del proyecto Spring Boot



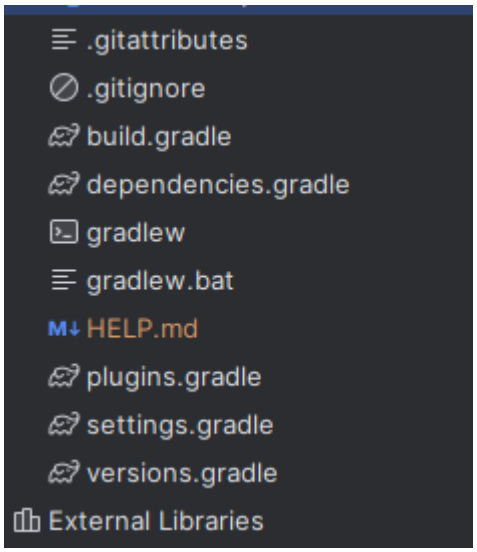
Se encuentra el módulo zero-trust-api-vo, el cual contiene diferentes clases y carpetas destinadas a organizar los objetos de transferencia de datos. Esto permite estructurar mejor la información que se envía y se recibe en formato JSON, facilitando su consumo desde otros sistemas o aplicaciones.

Cada módulo cuenta con su propio archivo build.gradle, lo que permite manejar dependencias específicas y establecer la comunicación entre los diferentes módulos del proyecto.

Cabe señalar que el proyecto cuenta con varios archivos de dependencias, configuración y plugins a nivel global, los cuales ayudan al correcto funcionamiento del sistema. Estos archivos permiten centralizar versiones, definir configuraciones generales y mantener una mejor organización entre los diferentes módulos del proyecto.

Figura 22.

Archivos de configuración y construcción del proyecto con Gradle



Como primer paso, se configuraron los archivos. yaml, los cuales contienen las configuraciones necesarias para los ambientes local, desarrollo y producción del microservicio. En estos archivos se definieron aspectos importantes como los puertos, el nombre del servicio y la respectiva conexión a la base de datos.

Figura 23.

Archivos de configuración de entorno en Spring Boot (application.yaml)

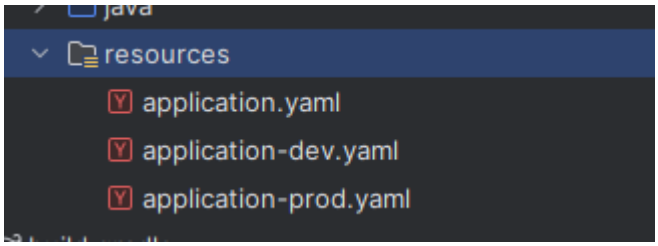
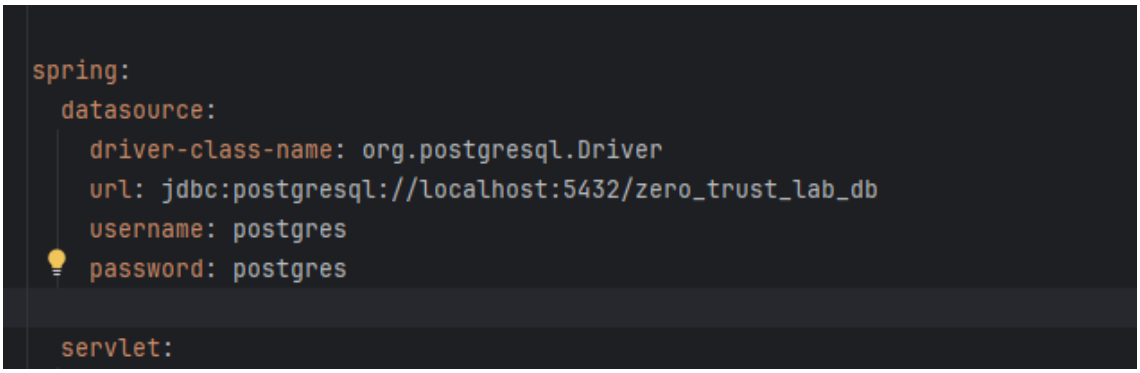


Figura 24.

Configuración de la fuente de datos (DataSource) en Spring Boot



```
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5432/zero_trust_lab_db
    username: postgres
    password: postgres

  servlet:
```

Figura 25.

Configuración de niveles de logging y perfiles en Spring Boot

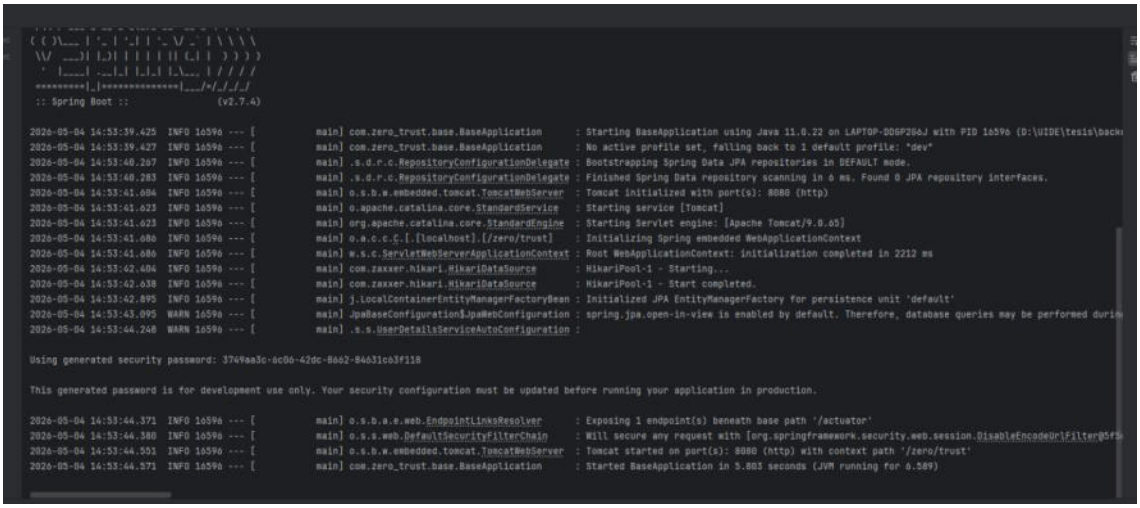
```
logging:
  level:
    org:
      springframework:
        web: INFO
      hibernate: ERROR

spring:
  profiles:
    default: dev
```

Ejecución del aplicativo utilizando la seguridad básica de Spring Security, sin aplicar configuraciones avanzadas de seguridad en esta etapa. Esto permite validar primero el funcionamiento general del microservicio y comprobar que la aplicación levante correctamente antes de implementar reglas de autenticación y autorización más específicas.

Figura 26.

Ejecución e inicialización de la aplicación Spring Boot en consola



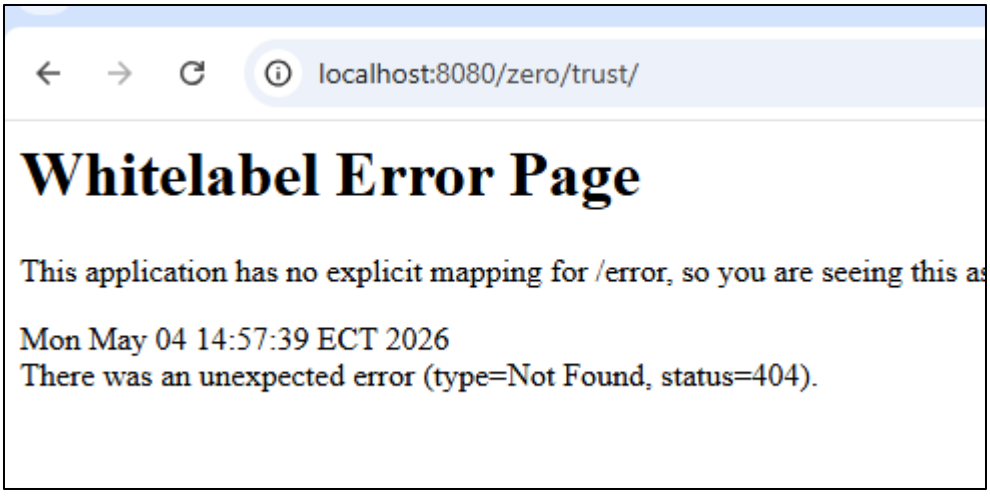
El microservicio cuenta inicialmente con la configuración básica de Spring Security, la clave de acceso y el usuario son generados de forma predeterminada por el framework. Esta configuración fue utilizada únicamente como una validación inicial para comprobar que el microservicio se levantaba correctamente y que las dependencias de seguridad estaban funcionando. Sin embargo, no forma parte del modelo final de seguridad propuesto, ya que posteriormente la autenticación y autorización serán gestionadas mediante los componentes definidos en la arquitectura.

Figura 27.

Interfaz de autenticación de usuario generada por Spring Security

Figura 28.

Página de error Whitelabel (404) en aplicación Spring Boot



3.1.4. Explicación general de la creación y configuración base de datos

- **Creación de la base de datos en PostgreSQL.**

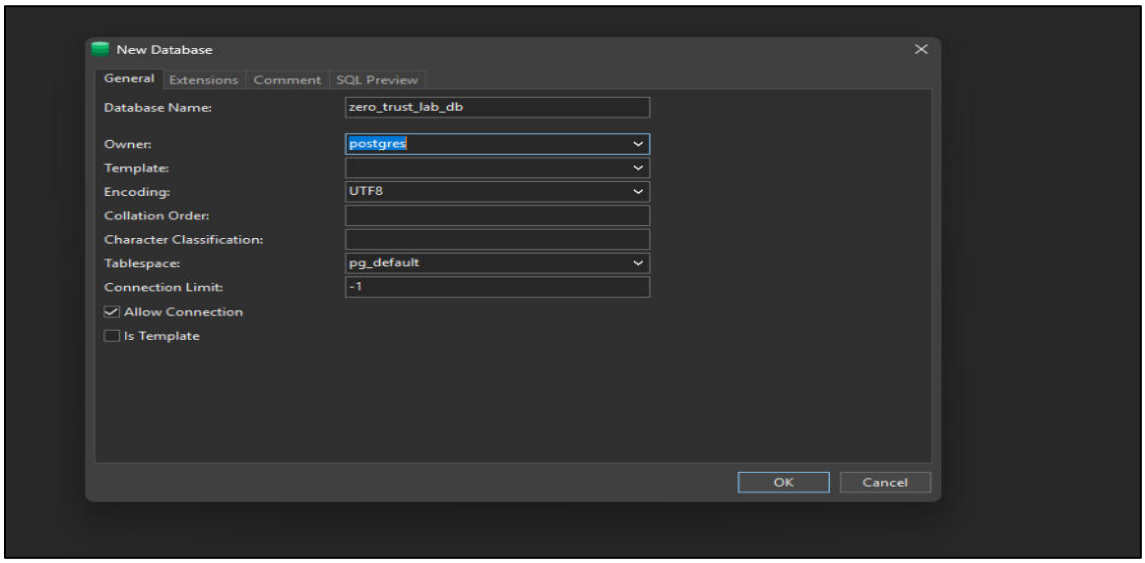
La base de datos será creada inicialmente en un entorno local de PostgreSQL para fines de configuración y validación dentro del laboratorio, bajo el nombre zero_trust_lab_db. Esta

base contendrá los esquemas necesarios para organizar la información relacionada con las APIs del proyecto.

La base de datos será desplegada en AWS, desde donde será consumida por los microservicios definidos en la arquitectura. Para este entorno se utilizarán usuarios específicos con permisos controlados, credenciales protegidas y restricciones de acceso a nivel de red, con el fin de mantener una configuración más segura.

Figura 29.

Creación de base de datos PostgreSQL para la aplicación



La base de datos zero_trust_lab_db fue diseñada como soporte del laboratorio de validación de APIs bajo principios de Zero Trust y defensa en profundidad. Su estructura se divide en tres esquemas: core, audit y lab. El esquema core almacena el recurso protegido expuesto por el microservicio Spring Boot; el esquema audit registra eventos de consumo de APIs, decisiones de acceso, validación de gateway; y el esquema lab almacena los escenarios de

prueba y sus resultados. Esta organización permite validar autenticación, autorización, trazabilidad, validación del gateway, registro de escenarios de prueba y efectividad de controles.

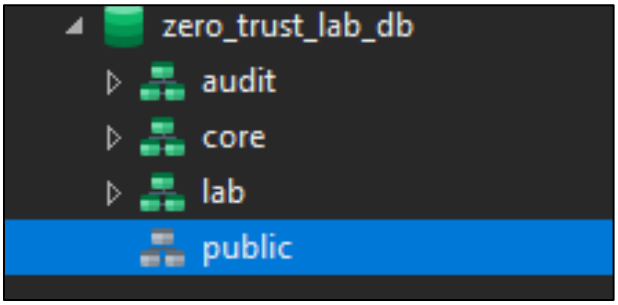
Tabla 7.

Descripción de los esquemas de base de datos

ESQUEMA	FUNCIÓN
CORE	Guarda el recurso protegido que será consumido por las APIs
AUDIT	Guarda la trazabilidad de accesos, decisiones de autorización, validaciones del gateway y eventos de consumo de APIs.
LAB	Guarda los escenarios y resultados de las pruebas controladas

Figura 30.

Esquemas de base de datos en PostgreSQL



Relación de tablas con los indicadores técnicos del proyecto

Para complementar la organización de la base de datos, se establece la relación entre cada indicador técnico del proyecto y las tablas que permiten obtener la información necesaria para su

validación. De esta manera, no solo se describe la función de cada esquema, sino también su aporte dentro de las pruebas de seguridad aplicadas al microservicio.

Tabla 8.

Relación entre indicadores técnicos y tablas de base de datos

Indicador técnico	Tabla utilizada	Información que permite obtener
Consumo de API protegida	core.core_protected_resource	Permite validar la consulta, creación, actualización e inactivación del recurso protegido.
Autenticación y autorización	audit.audit_event	Permite registrar si una solicitud fue permitida o denegada según el token, roles y permisos.
Validación del Gateway	audit.audit_event	Permite identificar si la solicitud pasó por el gateway o si existió acceso directo al microservicio.
Trazabilidad de accesos	audit.audit_event	Permite registrar usuario, endpoint, método HTTP, IP, código de respuesta y recurso afectado.
Escenarios de prueba	lab.lab_test_scenario	Permite definir las pruebas controladas, la amenaza simulada, el endpoint objetivo y el resultado esperado.
Resultados de pruebas	lab.lab_test_execution	Permite registrar el resultado obtenido, código HTTP,

		bloqueo del intento, tiempo de respuesta y evidencia técnica.
Efectividad de controles	lab.lab_test_scenario y lab.lab_test_execution	Permite comparar el resultado esperado con el resultado obtenido en cada prueba ejecutada.

3.1.5. Descripción de la estructura de la base de datos

1. Esquema core

Tabla core.core_protected_resource

La tabla core_protected_resource almacena los recursos protegidos utilizados por el microservicio para simular información sensible. Esta tabla permite ejecutar operaciones CRUD controladas y aplicar sobre ellas mecanismos de autenticación, autorización, validación de gateway, auditoría y pruebas de seguridad.

Figura 31.

Tabla core.core_protected_resource

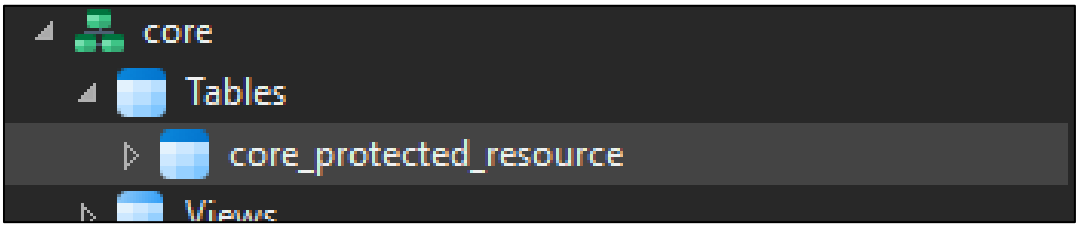
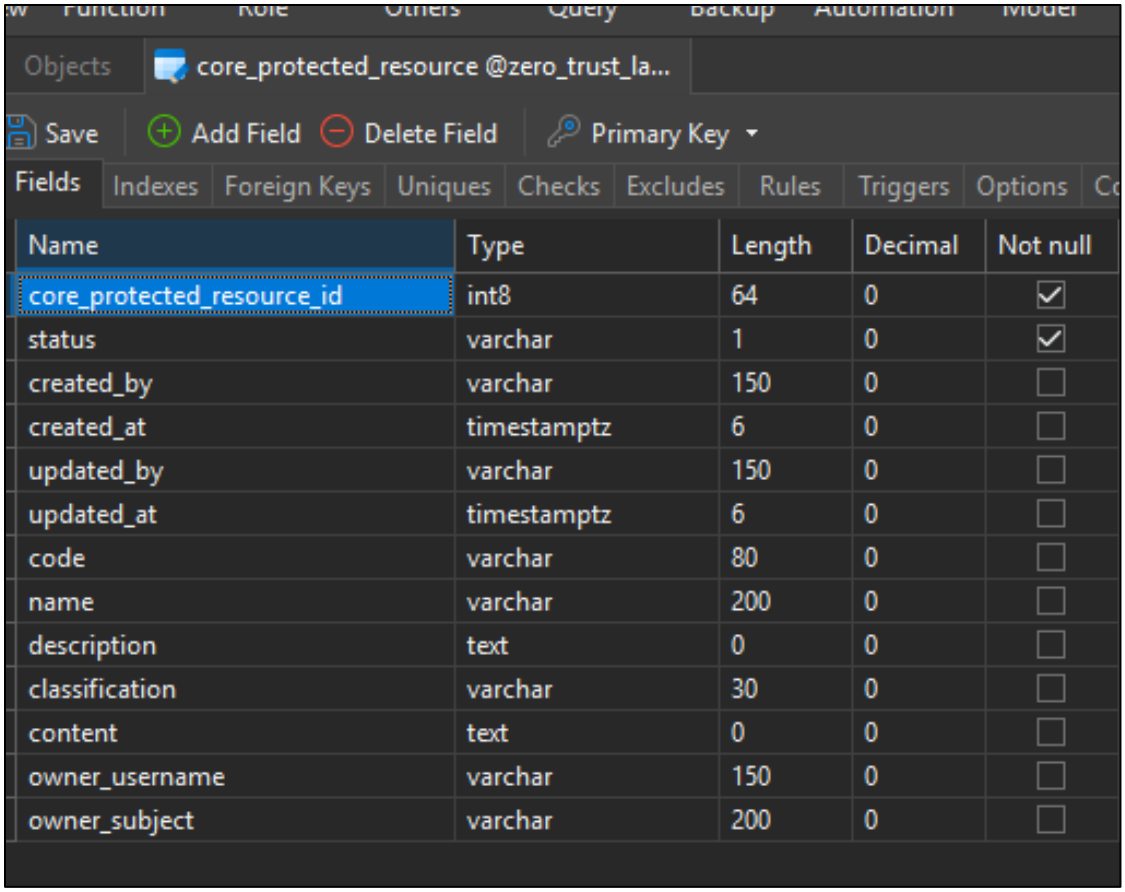


Figura 32.

Columnas y atributos de la tabla core.core_protected_resource



Name	Type	Length	Decimal	Not null
core_protected_resource_id	int8	64	0	☑
status	varchar	1	0	☑
created_by	varchar	150	0	☐
created_at	timestamptz	6	0	☐
updated_by	varchar	150	0	☐
updated_at	timestamptz	6	0	☐
code	varchar	80	0	☐
name	varchar	200	0	☐
description	text	0	0	☐
classification	varchar	30	0	☐
content	text	0	0	☐
owner_username	varchar	150	0	☐
owner_subject	varchar	200	0	☐

Funcionalidad

Permite probar:

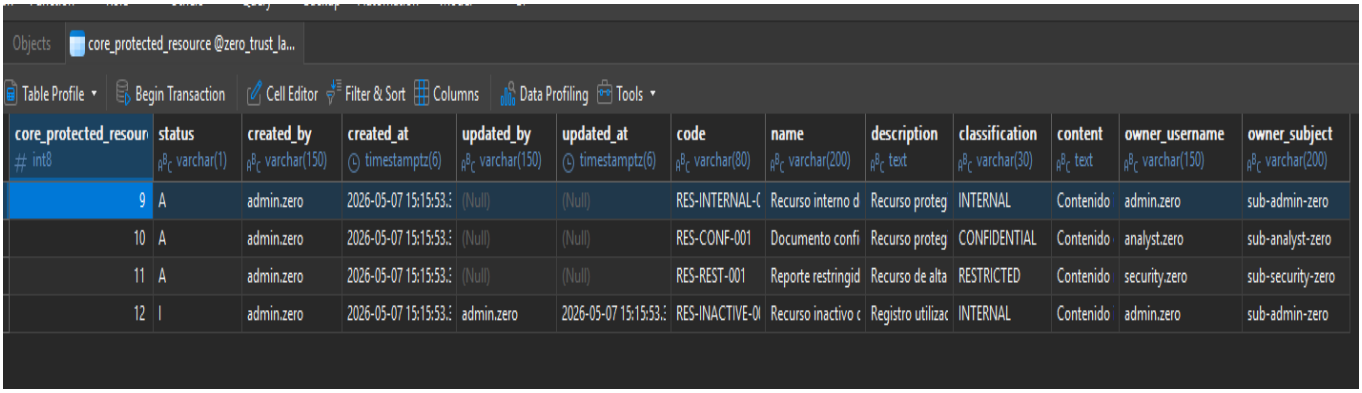
- Lectura de datos protegidos
- Consulta por ID
- Creación de recursos
- Actualización de recursos

- Inactivación lógica
- Validación de acceso
- Auditoría
- Búsqueda con filtros

Data de prueba:

Figura 33.

Datos de prueba insertados manualmente



core_protected_resoun # int8	status varchar(1)	created_by varchar(150)	created_at timestamptz(6)	updated_by varchar(150)	updated_at timestamptz(6)	code varchar(80)	name varchar(200)	description text	classification varchar(30)	content text	owner_username varchar(150)	owner_subject varchar(200)
9	A	admin.zero	2026-05-07 15:15:53.:	(Null)	(Null)	RES-INTERNAL-(	Recurso interno d	Recurso proteg	INTERNAL	Contenido	admin.zero	sub-admin-zero
10	A	admin.zero	2026-05-07 15:15:53.:	(Null)	(Null)	RES-CONF-001	Documento confi	Recurso proteg	CONFIDENTIAL	Contenido	analyst.zero	sub-analyst-zero
11	A	admin.zero	2026-05-07 15:15:53.:	(Null)	(Null)	RES-REST-001	Reporte restringid	Recurso de alta	RESTRICTED	Contenido	security.zero	sub-security-zero
12	I	admin.zero	2026-05-07 15:15:53.:	admin.zero	2026-05-07 15:15:53.:	RES-INACTIVE-0(	Recurso inactivo c	Registro utilizac	INTERNAL	Contenido	admin.zero	sub-admin-zero

2. Esquema audit

Tabla audit.audit_event

Esta tabla es una de las más importantes del proyecto. Guarda la trazabilidad general de lo que ocurre en las APIs.

La tabla audit_event permite mantener trazabilidad completa del consumo de APIs. Esta nos permite registrar datos técnicos de cada solicitud, como usuario, endpoint, método HTTP, código de respuesta, decisión de acceso, modo de autenticación, detección del gateway y recurso

afectado. Esta información es esencial para demostrar visibilidad, rastreabilidad y control sobre las solicitudes procesadas por el microservicio.

Figura 34.

Tabla audit.audit_event

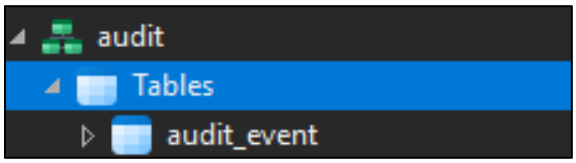
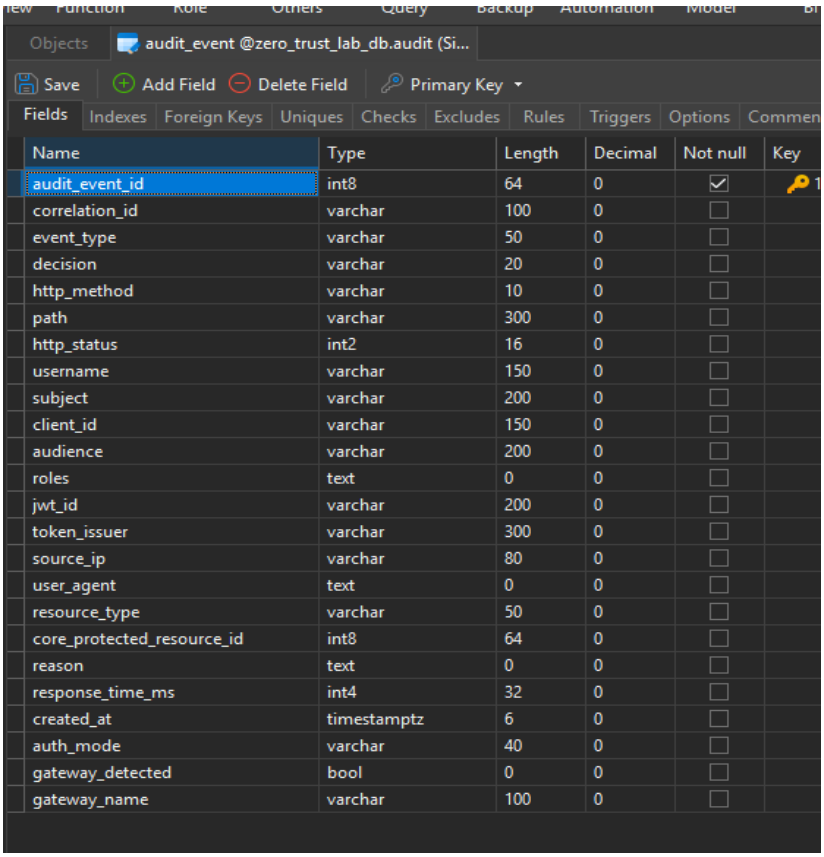


Figura 35.

Columnas y atributos de la tabla audit.audit_event



Name	Type	Length	Decimal	Not null	Key
audit_event_id	int8	64	0	☑	🔑 1
correlation_id	varchar	100	0	☐	
event_type	varchar	50	0	☐	
decision	varchar	20	0	☐	
http_method	varchar	10	0	☐	
path	varchar	300	0	☐	
http_status	int2	16	0	☐	
username	varchar	150	0	☐	
subject	varchar	200	0	☐	
client_id	varchar	150	0	☐	
audience	varchar	200	0	☐	
roles	text	0	0	☐	
jwt_id	varchar	200	0	☐	
token_issuer	varchar	300	0	☐	
source_ip	varchar	80	0	☐	
user_agent	text	0	0	☐	
resource_type	varchar	50	0	☐	
core_protected_resource_id	int8	64	0	☐	
reason	text	0	0	☐	
response_time_ms	int4	32	0	☐	
created_at	timestamptz	6	0	☐	
auth_mode	varchar	40	0	☐	
gateway_detected	bool	0	0	☐	
gateway_name	varchar	100	0	☐	

Funcionalidad

Registra:

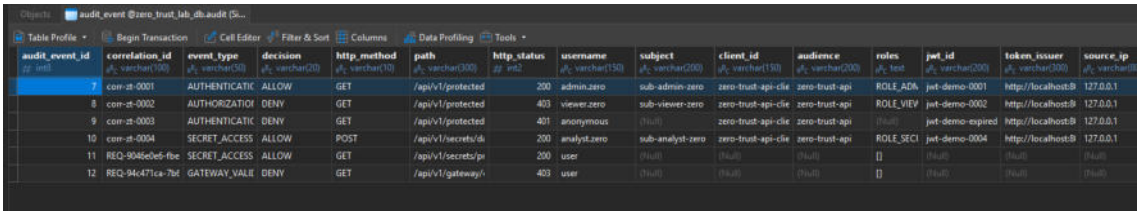
- Quién consumió la API
- Qué endpoint consumió
- Qué método HTTP usó
- Qué código HTTP recibió
- Desde qué IP entró
- Si fue permitido o denegado
- Si pasó por el gateway
- Qué recurso afectó
- Cuánto demoró la respuesta
- Qué modo de autenticación se usó

Estas funcionalidades están ligadas a que el API Gateway ayuda a centralizar validación de tokens, controlar tráfico, registrar auditoría y limitar exposición directa de servicios internos. Esta tabla permite evidenciar, dentro del entorno de nuestro laboratorio, las capacidades de monitoreo.

Data de prueba:

Figura 36.

Datos de prueba insertados manualmente



audit_event_id	correlation_id	event_type	decision	http_method	path	http_status	username	subject	client_id	audience	roles	jwt_id	token_issuer	source_ip
7	corr-zt-0001	AUTHENTICATIC	ALLOW	GET	/api/v1/protected	200	admin.zero	sub-admin-zero	zero-trust-api-clie	zero-trust-api	ROLE_ADN	jwt-demo-0001	http://localhost:8	127.0.0.1
8	corr-zt-0002	AUTHORIZATIOI	DENY	GET	/api/v1/protected	403	viewer.zero	sub-viewer-zero	zero-trust-api-clie	zero-trust-api	ROLE_VIEV	jwt-demo-0002	http://localhost:8	127.0.0.1
9	corr-zt-0003	AUTHENTICATIC	DENY	GET	/api/v1/protected	401	anonymous	(Null)	zero-trust-api-clie	zero-trust-api	(Null)	jwt-demo-expired	http://localhost:8	127.0.0.1
10	corr-zt-0004	SECRET_ACCESS	ALLOW	POST	/api/v1/secrets/da	200	analyst.zero	sub-analyst-zero	zero-trust-api-clie	zero-trust-api	ROLE_SECI	jwt-demo-0004	http://localhost:8	127.0.0.1
11	REQ-9046e0e6-fbe	SECRET_ACCESS	ALLOW	GET	/api/v1/secrets/pi	200	user	(Null)	(Null)	(Null)	[]	(Null)	(Null)	(Null)
12	REQ-94c471ca-7bf	GATEWAY_VALIE	DENY	GET	/api/v1/gateway/-	403	user	(Null)	(Null)	(Null)	[]	(Null)	(Null)	(Null)

3. Esquema lab

Tabla lab.lab_test_scenario

La tabla lab_test_scenario almacena el catálogo de escenarios de prueba definidos para el laboratorio. Cada escenario identifica la amenaza simulada, el endpoint objetivo, el control de seguridad esperado y el código HTTP esperado. Esta tabla permite organizar formalmente las pruebas de autenticación, autorización, validación del gateway, rate limiting, control de acceso e IDOR.

Figura 37.

Tabla lab.lab_test_scenario

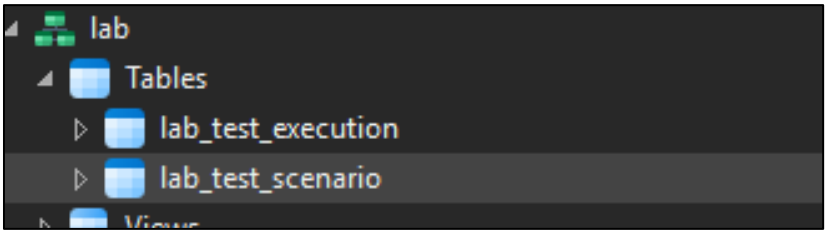
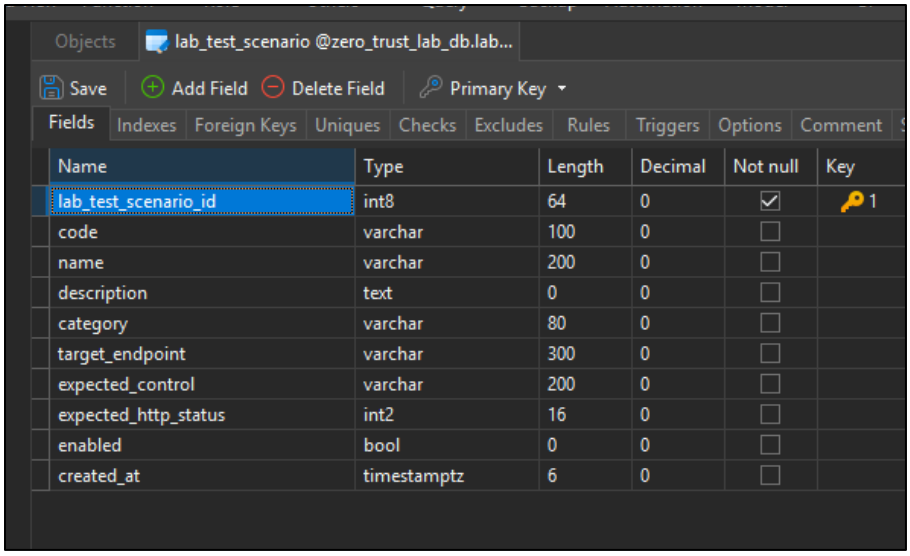


Figura 38.

Columnas y atributos de la tabla lab.lab_test_scenario



Funcionalidad

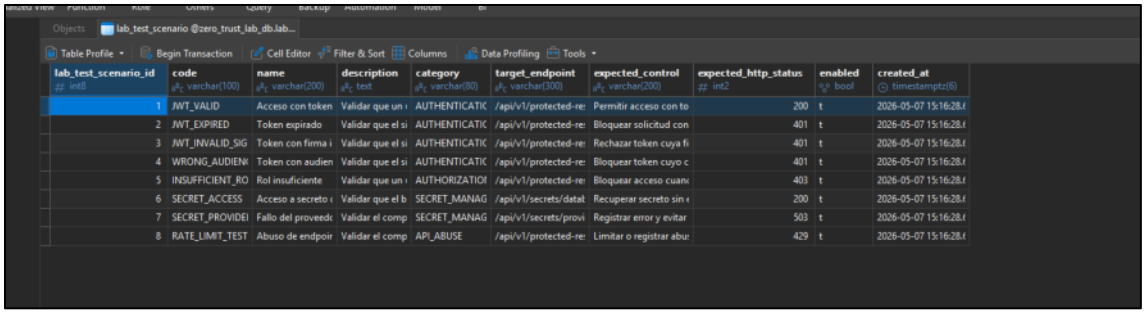
Define los escenarios de seguridad que se van a probar:

- JWT expirado
- JWT con firma inválida
- JWT con audiencia incorrecta
- Acceso con privilegios insuficientes
- Acceso directo sin gateway
- Abuso de endpoint/rate limiting
- Acceso federado a AWS mediante CyberArk Cloud Security

Data de prueba:

Figura 39.

Datos de prueba insertados manualmente



lab_test_scenario_id	code	name	description	category	target_endpoint	expected_control	expected_http_status	enabled	created_at
1	JWT_VALID	Acceso con token	Validar que un	AUTHENTICATIC	/api/v1/protected-re:	Permitir acceso con to	200	t	2026-05-07 15:16:28.(
2	JWT_EXPIRED	Token expirado	Validar que el si	AUTHENTICATIC	/api/v1/protected-re:	Bloquear solicitud con	401	t	2026-05-07 15:16:28.(
3	JWT_INVALID_SIG	Token con firma i	Validar que el si	AUTHENTICATIC	/api/v1/protected-re:	Rechazar token cuya fi	401	t	2026-05-07 15:16:28.(
4	WRONG_AUDIENC	Token con audien	Validar que el si	AUTHENTICATIC	/api/v1/protected-re:	Bloquear token cuyo c	401	t	2026-05-07 15:16:28.(
5	INSUFFICIENT_RO	Rol insuficiente	Validar que un	AUTHORIZATIOI	/api/v1/protected-re:	Bloquear acceso cuanc	403	t	2026-05-07 15:16:28.(
6	SECRET_ACCESS	Acceso a secreto	Validar que el b	SECRET_MANAG	/api/v1/secrets/datat	Recuperar secreto sin	200	t	2026-05-07 15:16:28.(
7	SECRET_PROVIDEI	Fallo del proveedc	Validar el comp	SECRET_MANAG	/api/v1/secrets/provi	Registrar error y evitar	503	t	2026-05-07 15:16:28.(
8	RATE_LIMIT_TEST	Abuso de endpoir	Validar el comp	API_ABUSE	/api/v1/protected-re:	Limitar o registrar abu:	429	t	2026-05-07 15:16:28.(

1. Tabla lab.lab_test_execution

La tabla lab_test_execution registra la ejecución real de las pruebas controladas. Permite almacenar el resultado observado, el código HTTP obtenido, si el intento fue bloqueado, el tiempo de respuesta y la evidencia técnica. Esta información se utiliza para calcular métricas de efectividad, precisión, rendimiento y visibilidad del esquema de seguridad.

Figura 40.

Tabla lab.lab_test_execution

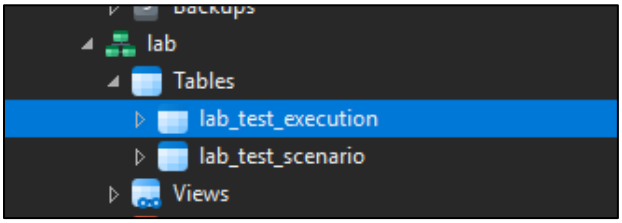
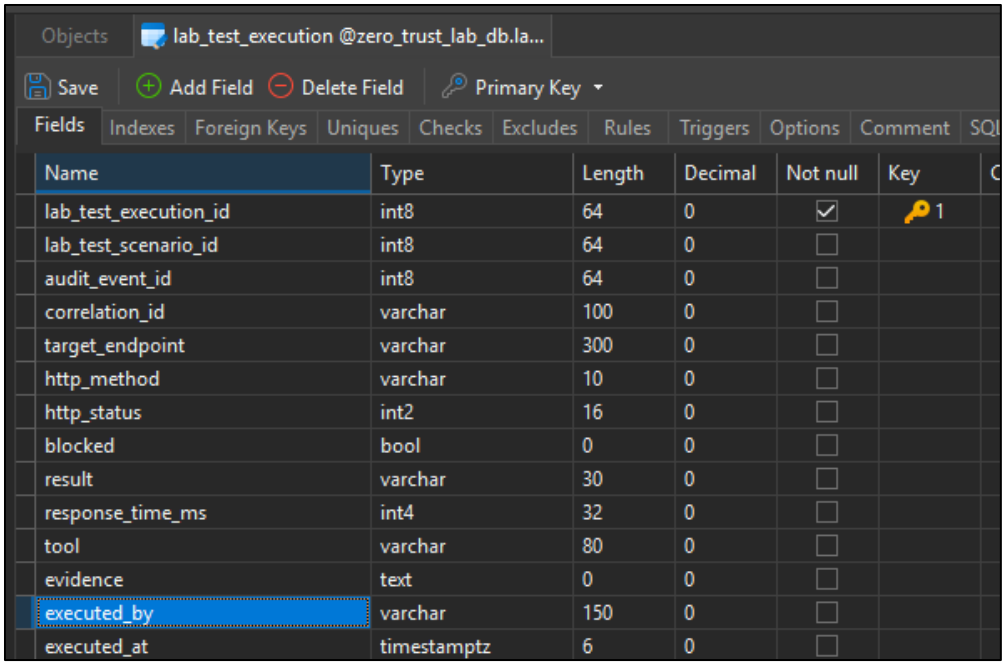


Figura 41.

Columnas y atributos de la tabla lab.lab_test_execution



2. Funcionalidad

Registra:

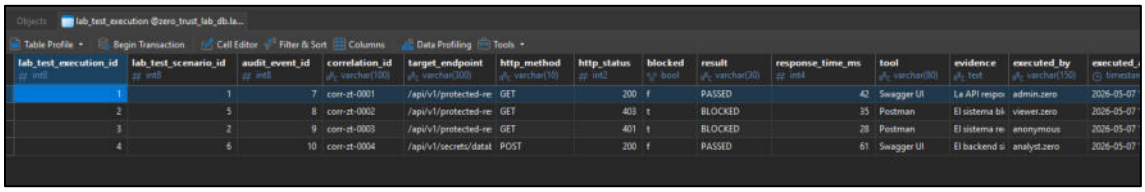
- Qué escenario se probó
- Qué endpoint se atacó o consumió
- Qué código HTTP respondió
- Si el intento fue bloqueado
- Si el resultado fue esperado o inesperado
- Con qué herramienta se probó

- Qué evidencia técnica se obtuvo

3. **Data de prueba:**

Figura 42.

Datos de prueba insertados manualmente



Con la base de datos ya estructurada y con el proyecto Backend ya construido se procedió a realizar las siguientes APIs.

1. GET /api/v1/system/health

Para qué sirve

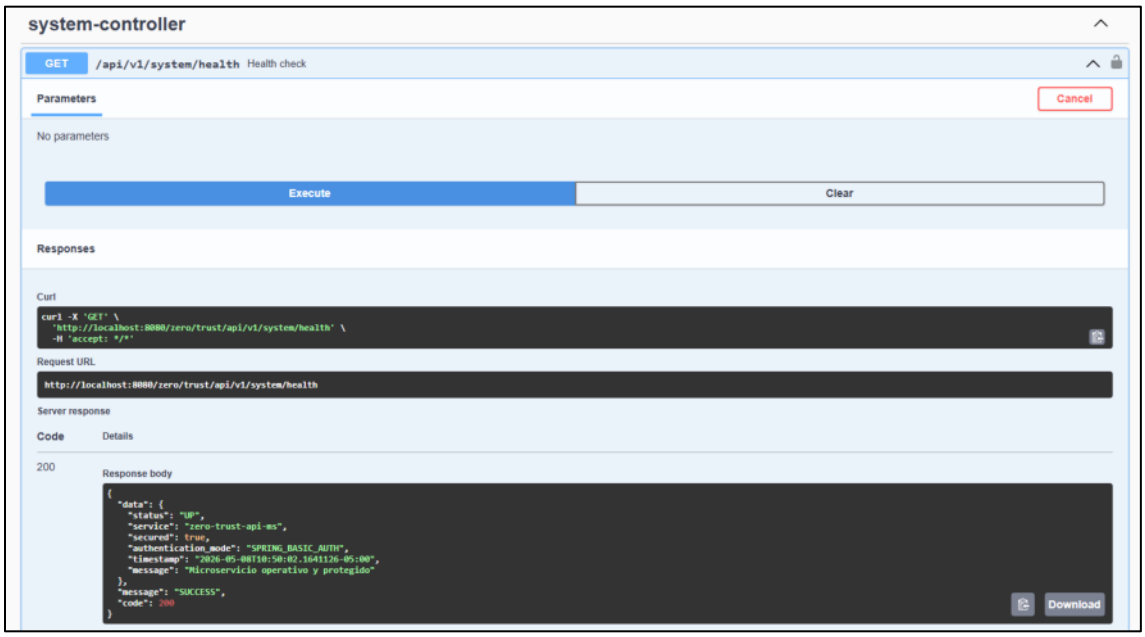
Verifica que el microservicio esté levantado y protegido. Esta API permite validar el estado operativo del microservicio y confirmar que las APIs se encuentran protegidas desde la fase inicial mediante Spring Security.

Control de seguridad validado

Protección inicial del endpoint y validación de disponibilidad del microservicio.

Figura 43.

Funcinamiendo de la API GET /api/v1/system/health



2. GET /api/v1/system/architecture

Para qué sirve

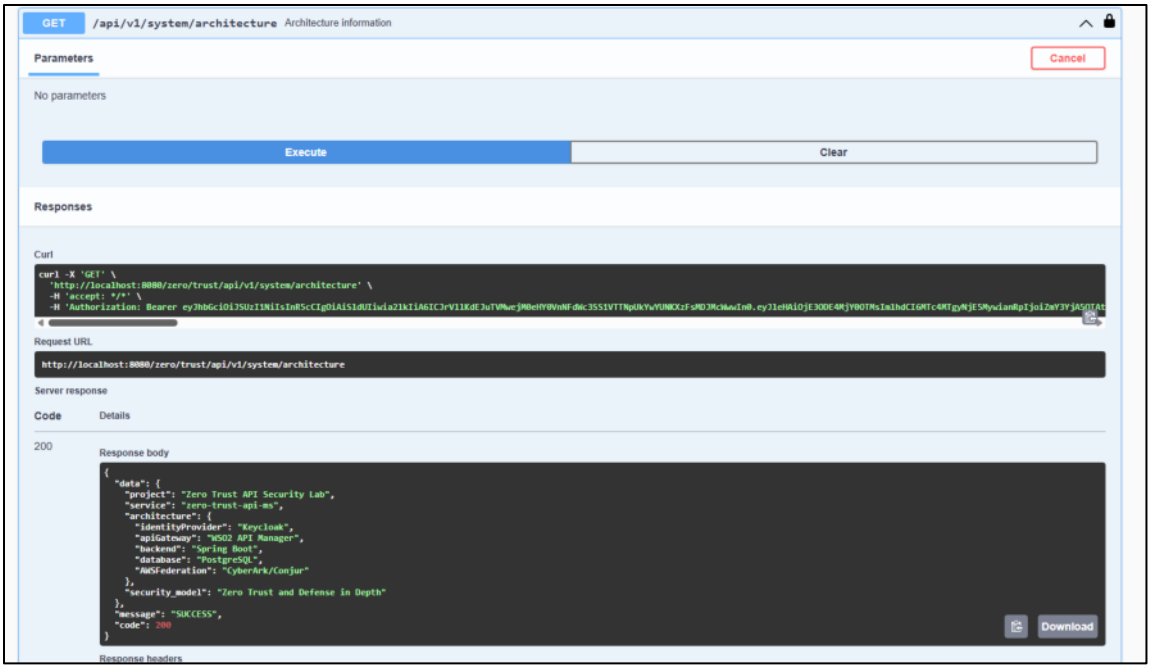
Muestra la arquitectura objetivo del laboratorio. Esta API documenta los componentes técnicos de la arquitectura: Keycloak, WSO2 API Manager, Spring Boot, PostgreSQL y CyberArk Cloud Security. Sirve como punto de referencia para explicar el flujo de seguridad que será implementado progresivamente.

Control de seguridad validado

Acceso controlado a la información técnica de la arquitectura.

Figura 44.

Funcionamiento de la API GET /api/v1/system/architecture



3. GET /api/v1/identity/me

Para qué sirve

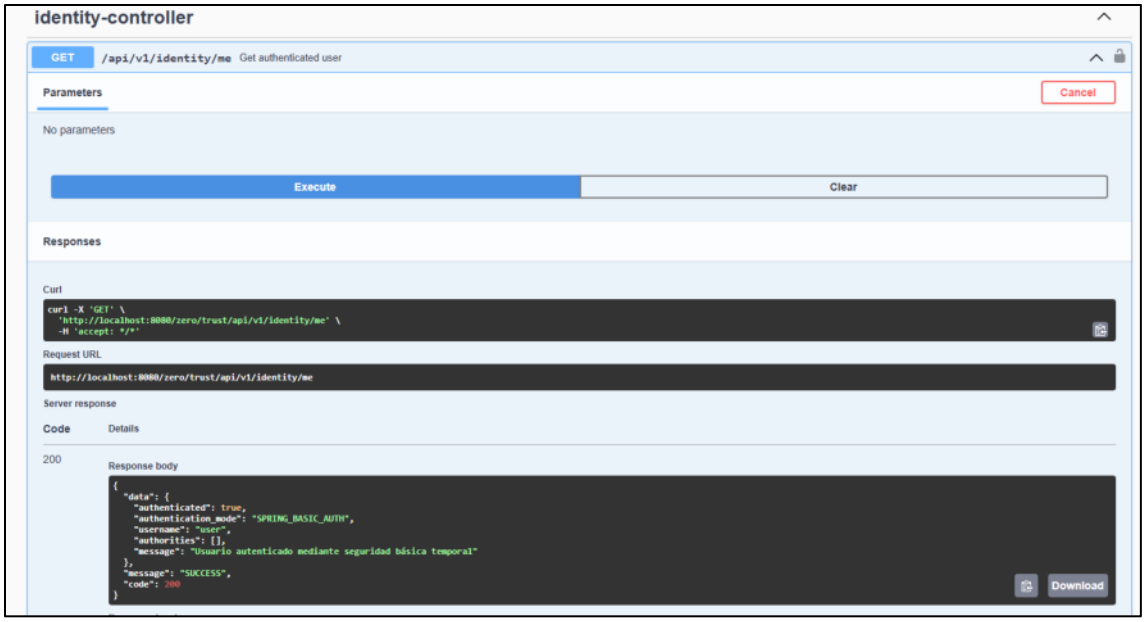
Muestra el usuario autenticado. Esta API permite comprobar la identidad del consumidor de la API. En la fase inicial muestra el usuario autenticado mediante Spring Security básico; posteriormente mostrará los claims del token JWT emitido por Keycloak.

Control de seguridad validado

Identidad del usuario autenticado.

Figura 45.

Funcionamiento de la API GET /api/v1/identity/me



4. GET /api/v1/protected-resources

Para qué sirve

Lista recursos protegidos activos. Esta API permite consultar recursos protegidos aplicando filtros de búsqueda y clasificación. Es útil para validar autenticación, auditoría, paginación y rate limiting.

Control de seguridad validado

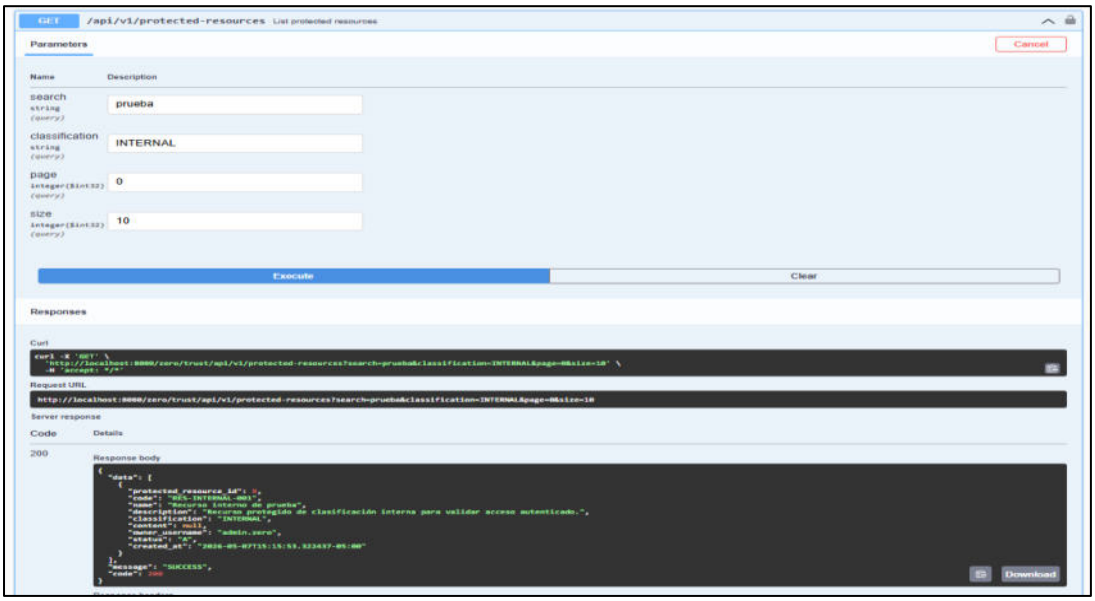
Autenticación, auditoría, paginación, rate limiting y pruebas de entrada sobre recursos protegidos.

Query de prueba

http://localhost:8080/zero/trust/api/v1/protected-resources?search=prueba&classification=INTERNAL&page=0&size=10

Figura 46.

Funcionamiento de la API GET /api/v1/protected-resources



5. GET /api/v1/protected-resources/{id}

Para qué sirve

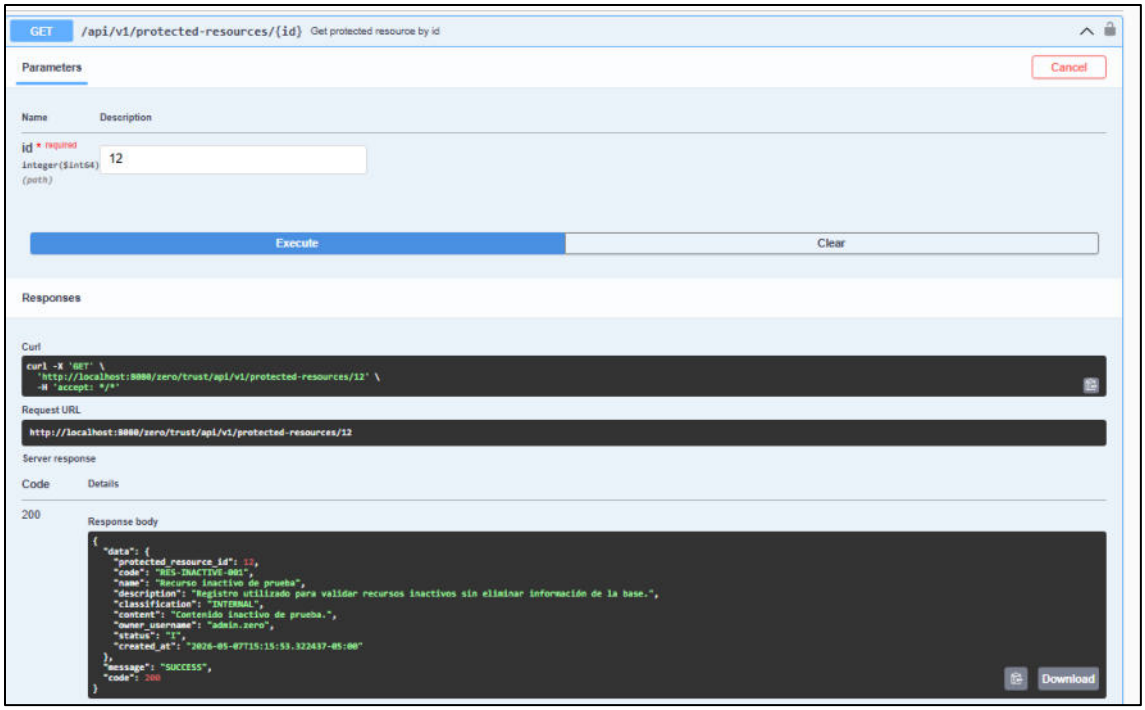
Consulta el detalle de un recurso protegido. Esta API permite evaluar el acceso directo a un recurso específico. Es útil para pruebas de IDOR.

Control de seguridad validado

Acceso directo a un recurso específico y pruebas de IDOR.

Figura 47.

Funcionamiento de la API GET /api/v1/protected-resources/{id}



6. POST /api/v1/protected-resources

Para qué sirve

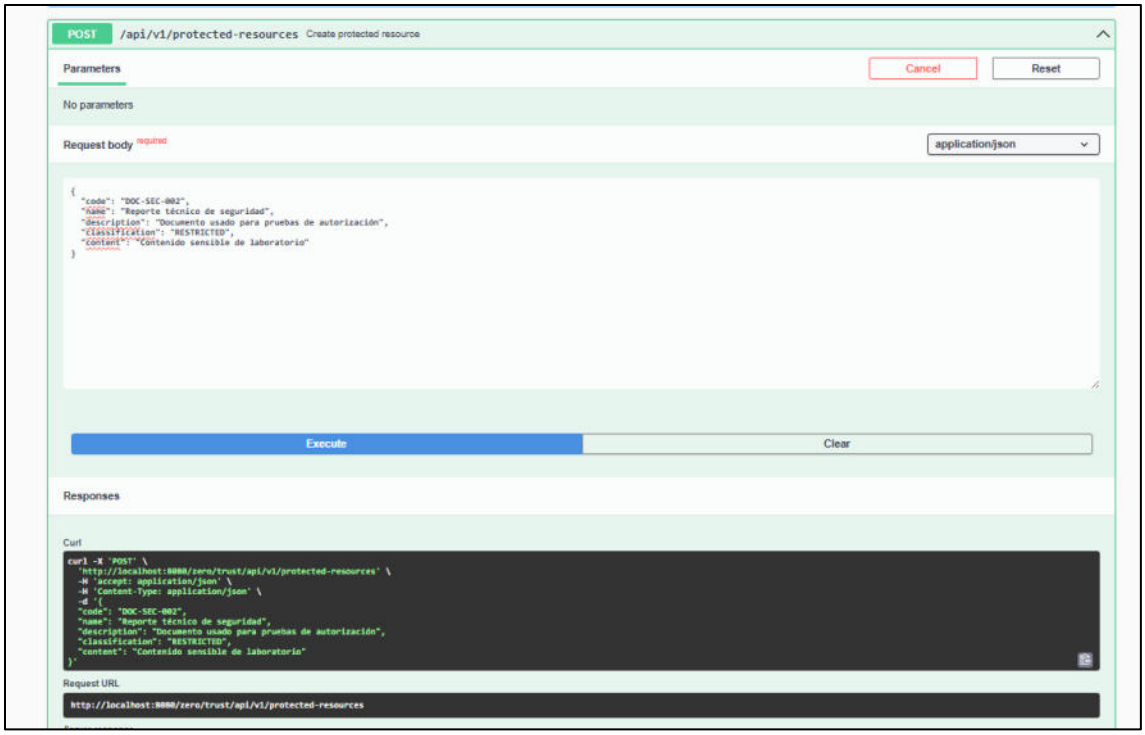
Crea un recurso protegido. Esta API permite crear información protegida de laboratorio. Se usa para validar controles de entrada, creación de datos sensibles y generación de eventos de auditoría.

Control de seguridad validado

Controles de entrada, creación de datos sensibles y generación de eventos de auditoría.

Figura 48.

Funcionamiento de la API POST /api/v1/protected-resources



7. PUT /api/v1/protected-resources/{id}

Para qué sirve

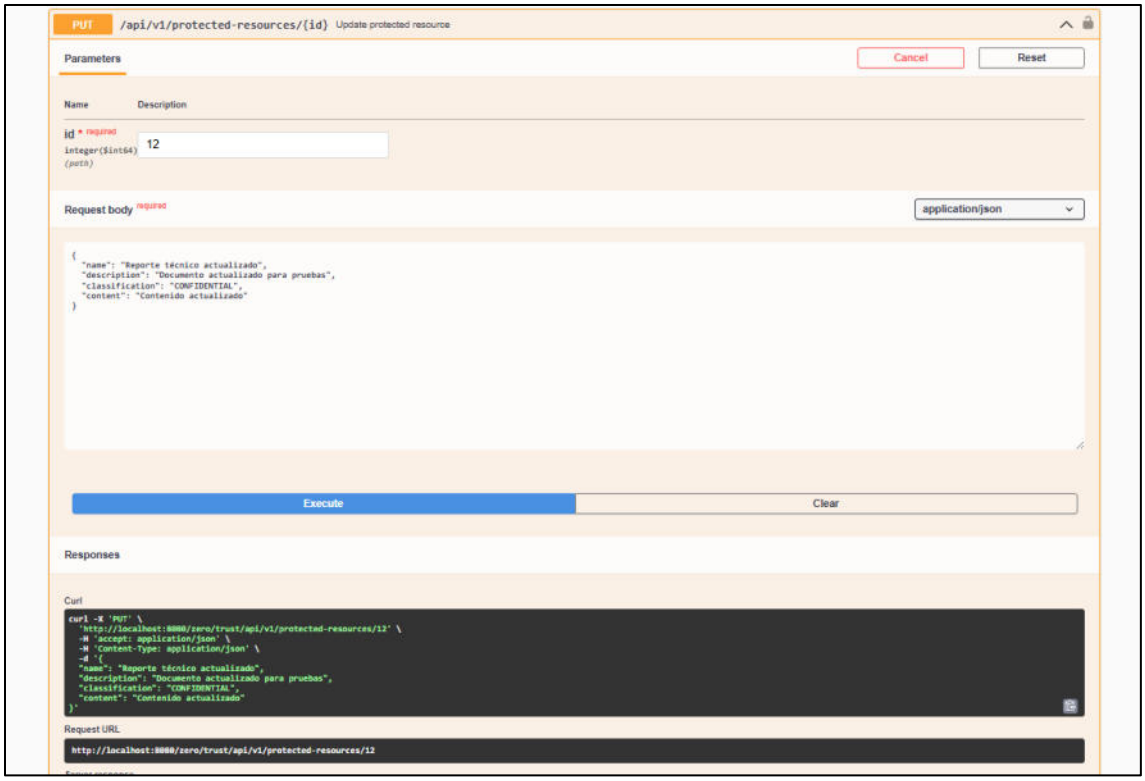
Actualiza un recurso protegido. Esta API permite modificar un recurso protegido. Será útil para demostrar autorización diferenciada cuando se asignen roles de escritura mediante Keycloak.

Control de seguridad validado

Autorización diferenciada para modificación de recursos protegidos.

Figura 49.

Funcionamiento de la API PUT /api/v1/protected-resources/{id}



8. DELETE /api/v1/protected-resources/{id}

Para qué sirve

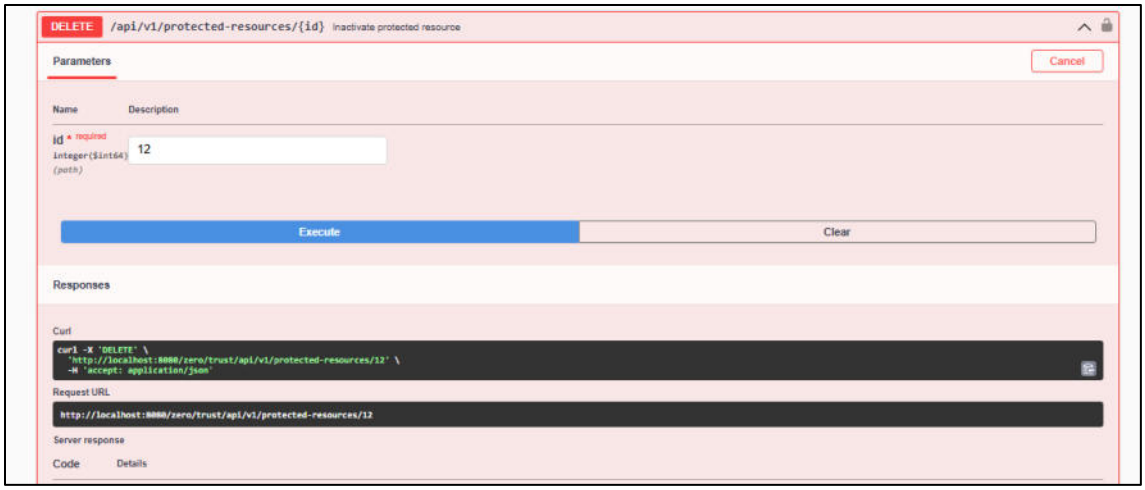
Inactiva un recurso protegido. No borra físicamente, solo cambia el estado del registro. Esta API representa una operación crítica. Permite demostrar eliminación lógica y, posteriormente, control de acceso restringido para usuarios administradores.

Control de seguridad validado

Control de acceso restringido para una operación crítica e inactivación lógica del recurso.

Figura 50.

Funcionamiento de la API DELETE /api/v1/protected-resources/{id}



9. GET /api/v1/gateway/context

Para qué sirve

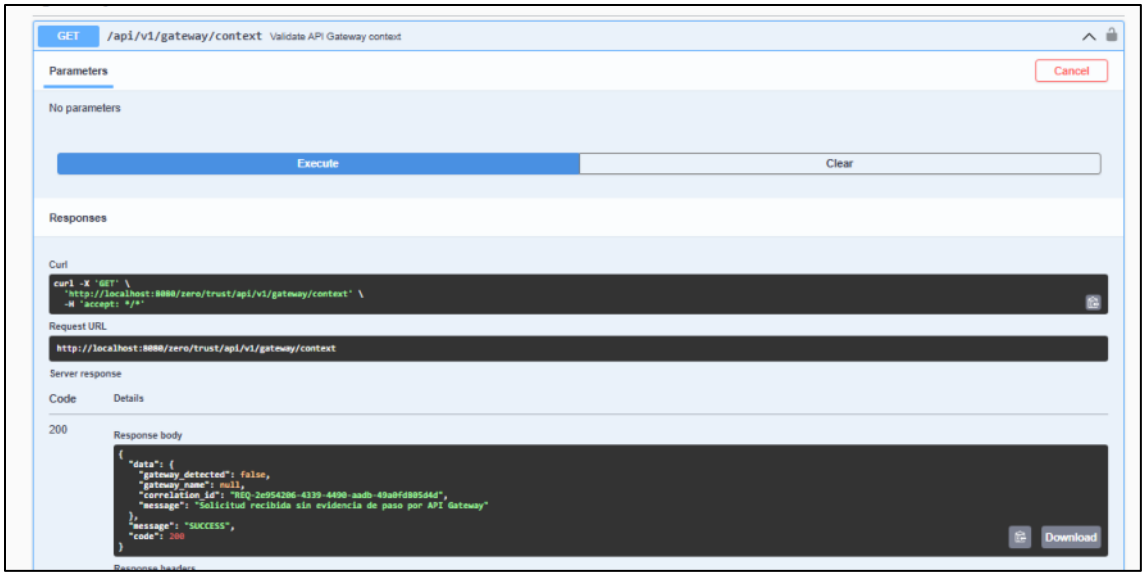
Verifica si la solicitud llegó desde WSO2 API Manager mediante headers. Esta API permite evaluar si el microservicio recibe solicitudes canalizadas por el API Gateway o intentos de acceso directo. Es relevante para validar el control contra bypass del gateway.

Control de seguridad validado

Validación del paso por el API Gateway y control contra bypass del gateway.

Figura 51.

Funcionamiento de la API GET /api/v1/gateway/context



10. GET /api/v1/audit/events

Para qué sirve

Lista eventos de auditoría. Esta API permite consultar la trazabilidad generada por el microservicio. Se utiliza para verificar la visibilidad de eventos de autenticación, acceso a recursos, validación del gateway y pruebas ejecutadas.

Control de seguridad validado

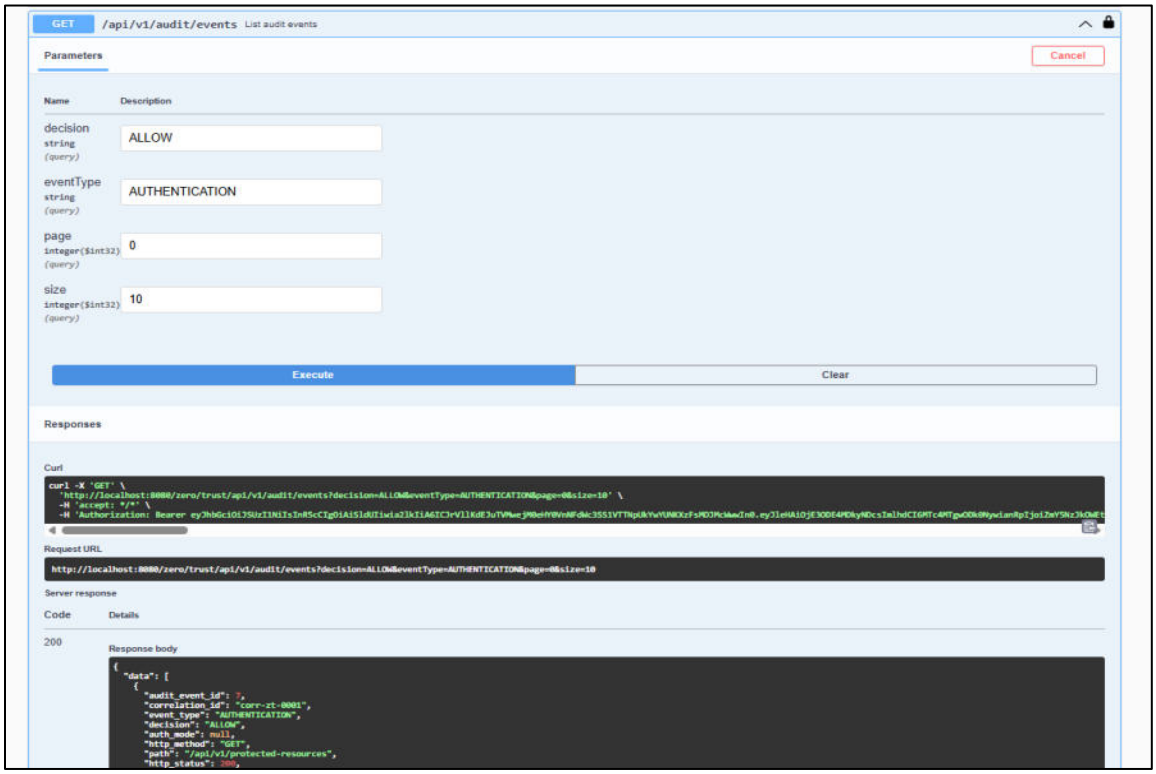
Trazabilidad de eventos de autenticación, acceso a recursos, validación de gateway, secretos y pruebas ejecutadas.

Query de prueba

http://localhost:8080/zero/trust/api/v1/audit/events?decision=ALLOW&eventType=AUTHENTICATION&page=0&size=10

Figura 52.

Funcionamiento de la API GET /api/v1/audit/events



11. GET /api/v1/lab/test-scenarios

Para qué sirve

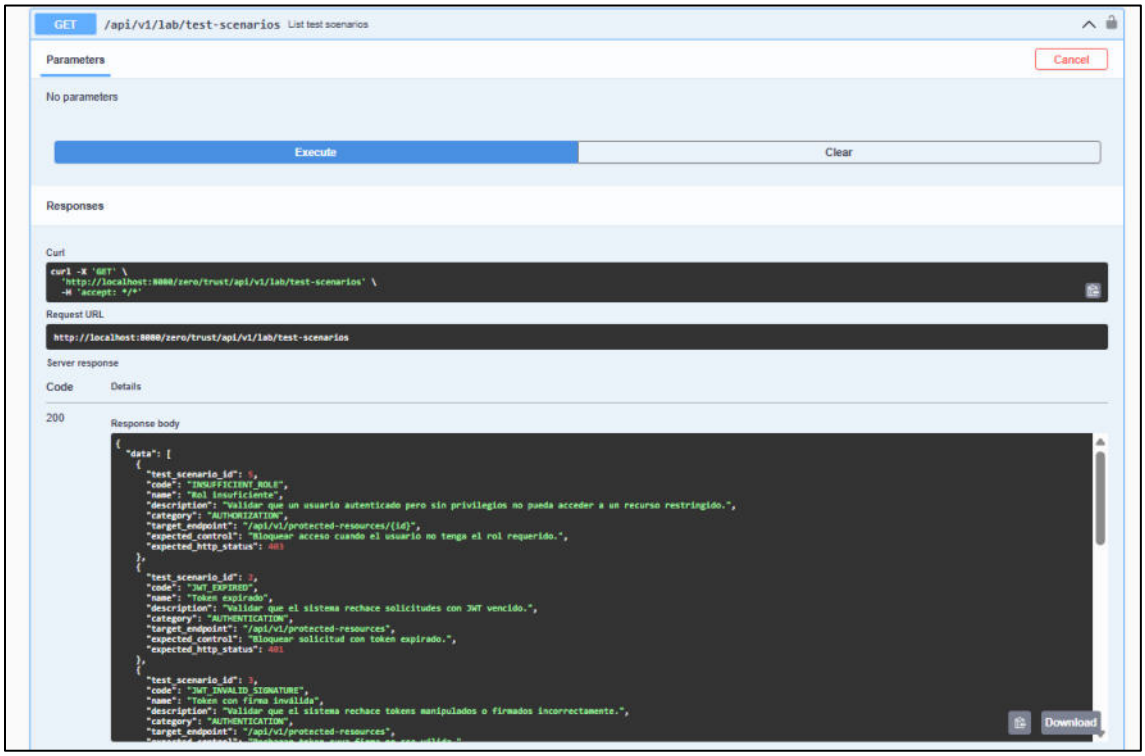
Lista los escenarios de pruebas definidos. Esta API permite visualizar el catálogo de pruebas controladas. Cada escenario define la amenaza, el endpoint objetivo, el control y el resultado esperados.

Control de seguridad validado

Definición de escenarios de pruebas controladas, amenaza, endpoint objetivo, control y resultado esperado.

Figura 53.

Funcionamiento de la API GET /api/v1/lab/test-scenarios



12. POST /api/v1/lab/test-executions

Para qué sirve

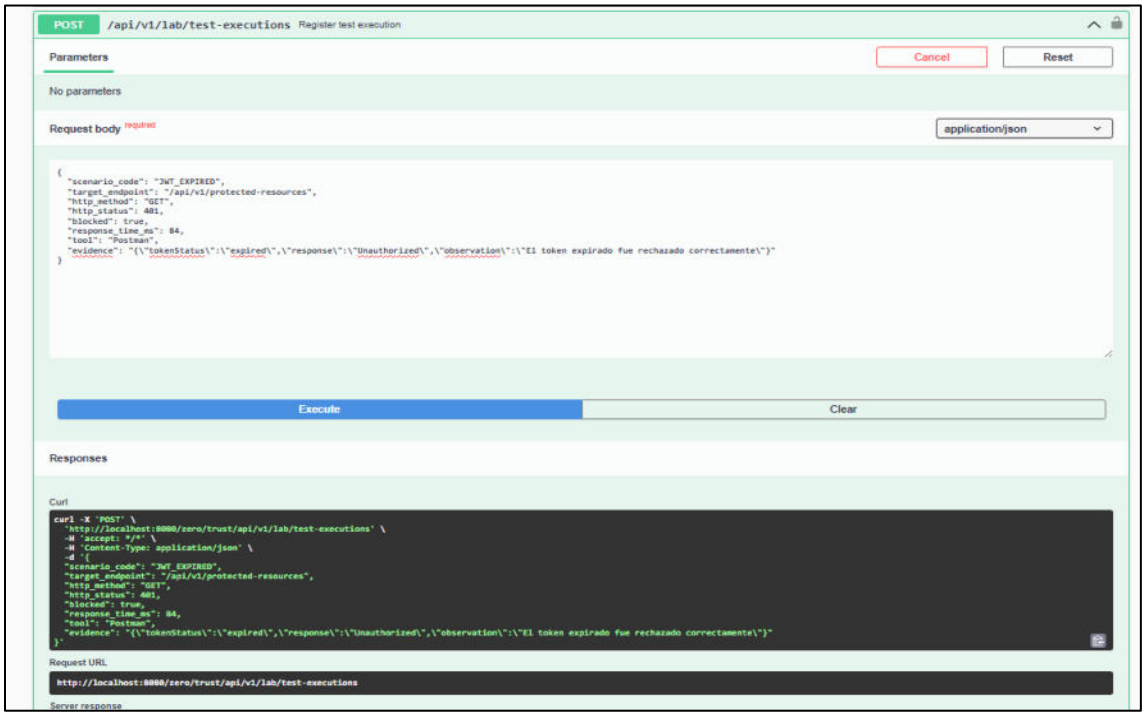
Registra el resultado de una prueba ejecutada. Esta API registra la evidencia técnica obtenida durante una prueba controlada. Permite guardar el resultado observado, el código HTTP recibido, si el intento fue bloqueado y la herramienta utilizada.

Control de seguridad validado

Registro de evidencia técnica, resultado observado, código HTTP recibido, bloqueo del intento y herramienta utilizada.

Figura 54.

Funcionamiento de la API POST /api/v1/lab/test-executions



13. GET /api/v1/lab/security-metrics

Para qué sirve

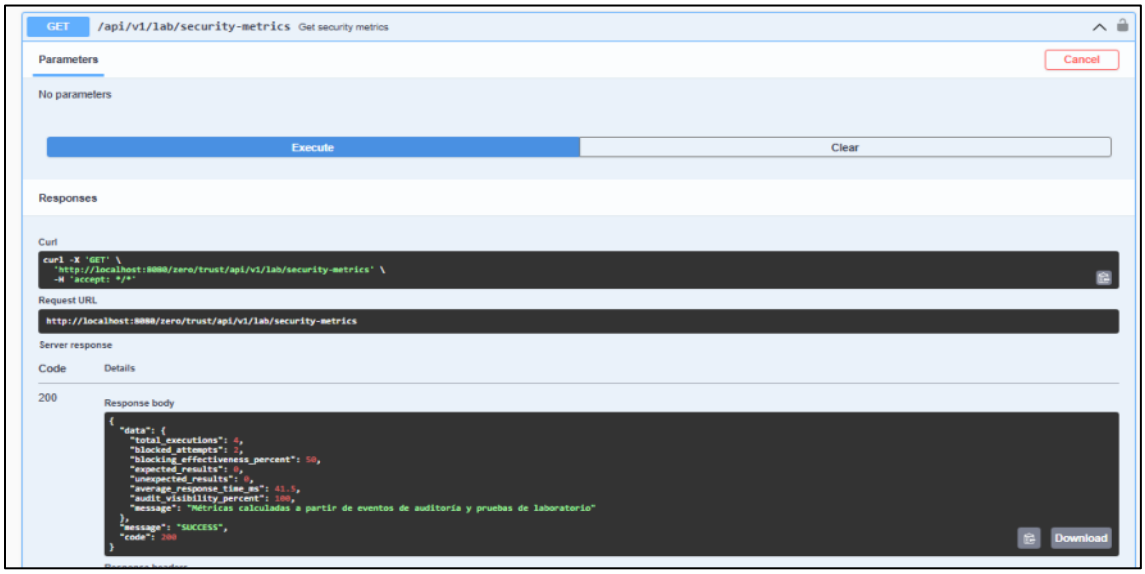
Calcula métricas de seguridad a partir de las pruebas registradas. Esta API resume los resultados del laboratorio mediante indicadores técnicos. Permite calcular efectividad de bloqueo, resultados esperados, resultados inesperados, tiempo promedio de respuesta y visibilidad de auditoría.

Control de seguridad validado

Cálculo de efectividad de bloqueo, resultados esperados, resultados inesperados, tiempo promedio de respuesta y visibilidad de auditoría.

Figura 55.

Funcionamiento de la API GET /api/v1/lab/security-metrics



3.1.6. Observación de las APIs

Los endpoints descritos fueron inicialmente desplegados con la configuración de seguridad por defecto de Spring Security, con el propósito de verificar su funcionamiento antes de integrar los mecanismos definitivos de autenticación y autorización. Esta validación inicial permitió comprobar la disponibilidad del microservicio, la correcta exposición de los recursos y la estructura funcional de las APIs del laboratorio.

Una vez validado el comportamiento funcional de cada endpoint, se procedió a la configuración de Keycloak como proveedor de identidad centralizado. Con esta integración, el

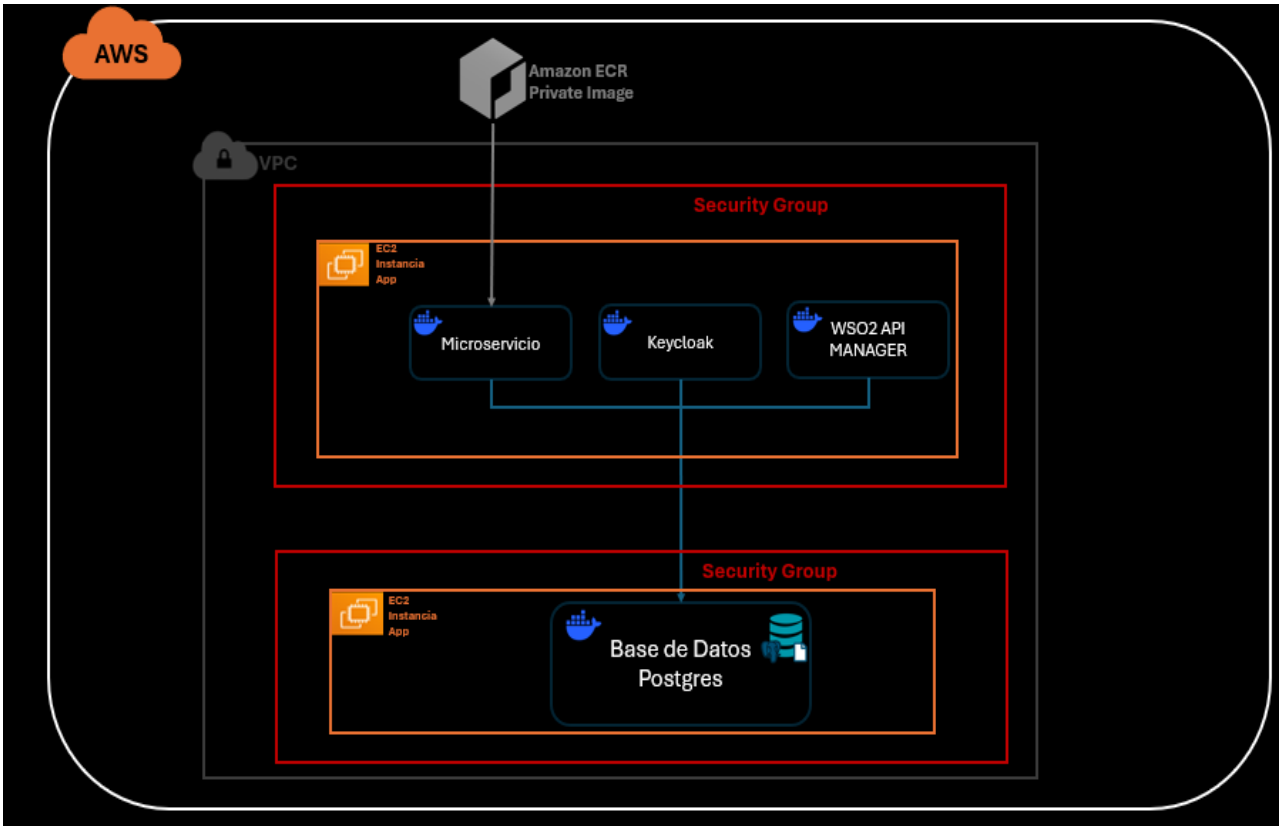
consumo de las APIs quedó condicionado a la presentación de un token JWT válido y a la verificación de roles específicos para cada operación protegida.

3.1.7. Despliegue de infraestructura en AWS

Para el desarrollo del proyecto se planteó la siguiente arquitectura de infraestructura en una nube pública AWS:

Figura 56.

Arquitectura de infraestructura AWS

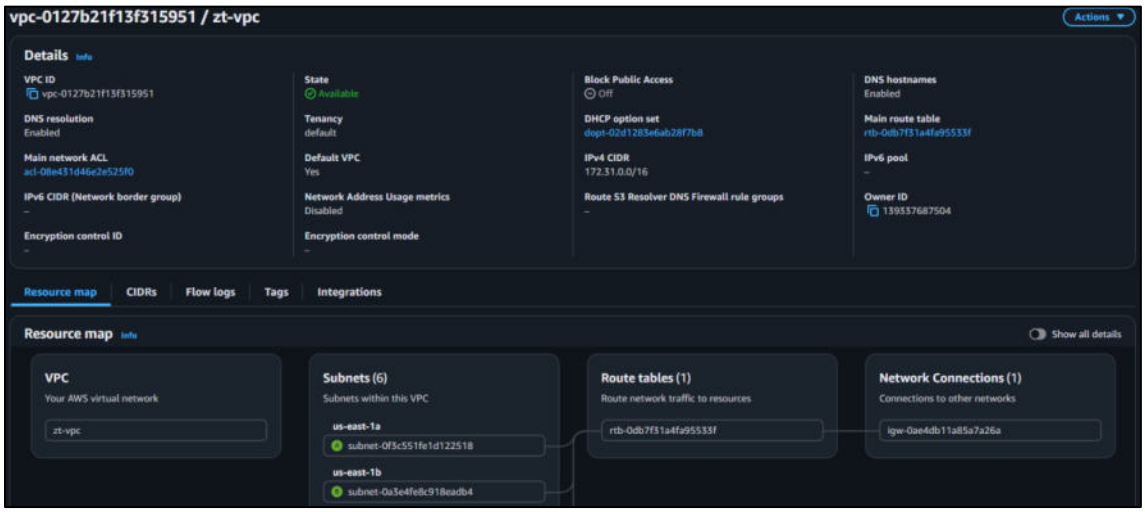


Dicha arquitectura está compuesta por los siguientes componentes:

- VPC: Configuraciones de red tanto subredes públicas como subredes privadas.

Figura 57.

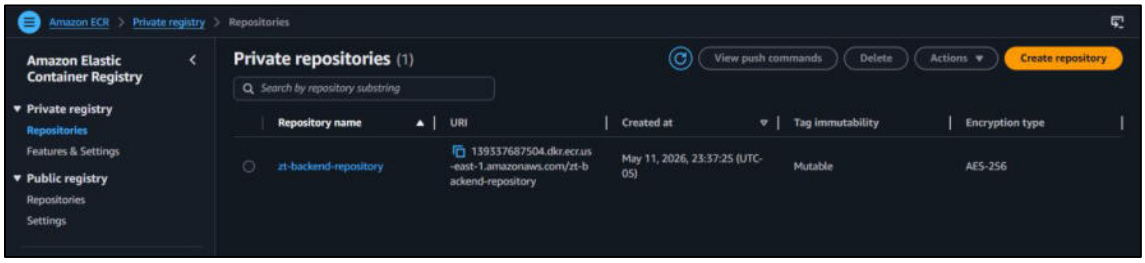
VPC de la solución



- ECR: Repositorio privado para publicar o pushear las imágenes Docker generas para el proyecto en este caso el microservicio.

Figura 58.

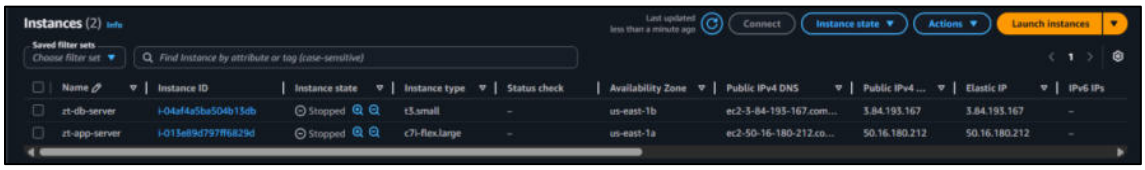
Repositorio de ECR



- EC2: Conocidas como instancias o máquinas virtuales donde se montarán los contenedores pertinentes para la solución en función de la arquitectura propuesta.

Figura 59.

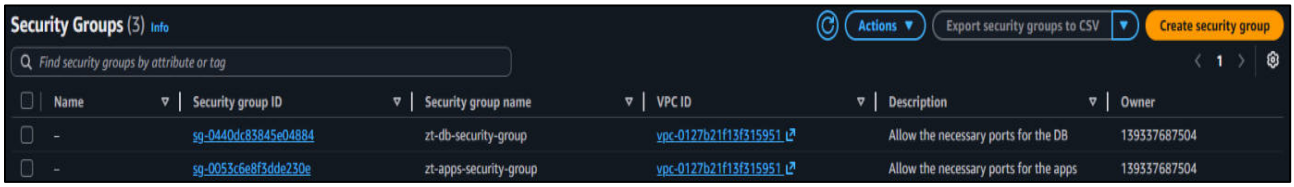
Instancias proyecto



- Security Group: Reglas de entrada o salida de cada uno de los recursos de red en la VPC.

Figura 60.

Grupos de seguridad



Proceso montar Infraestructura

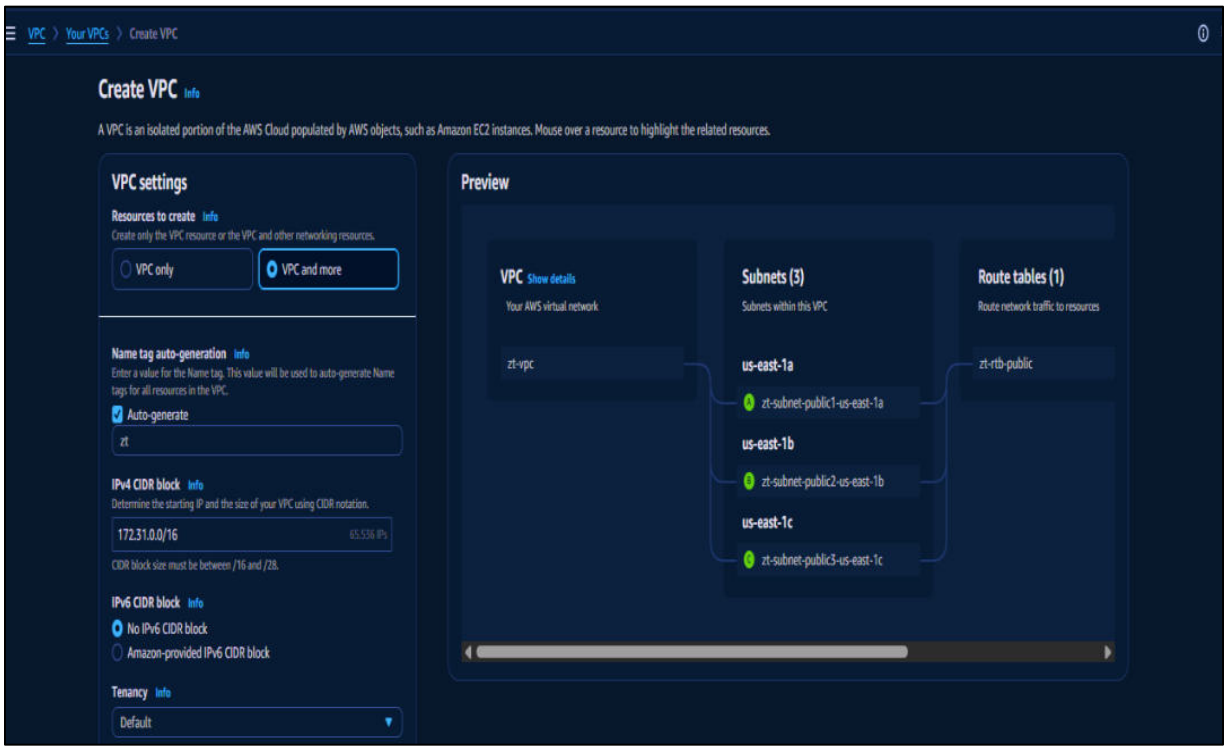
Para este proyecto se usó el cliente de AWS para montar cada uno de los componentes previamente mencionados. Dicho proceso es el siguiente

VPC

Se hicieron las configuraciones básicas ya que el Security Group filtrará tanto cuales van a ser los canales de salida como de entrada de cada uno de los componentes de la red

Figura 61.

Creación VPC



Security Group

Como se mencionó previamente el Security Group serán los encargados de saber que escucha y que recibe cada uno de los servidores que se montarán en la nube. Para ello se creó un SG para cada uno de los servidores que se montarán:

Figura 62.

Security Group Apps

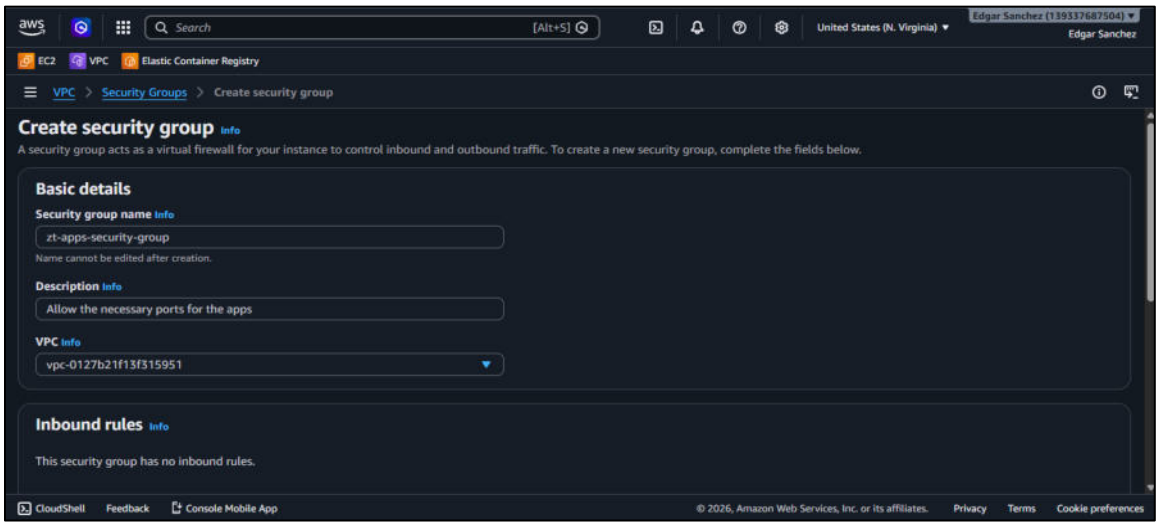
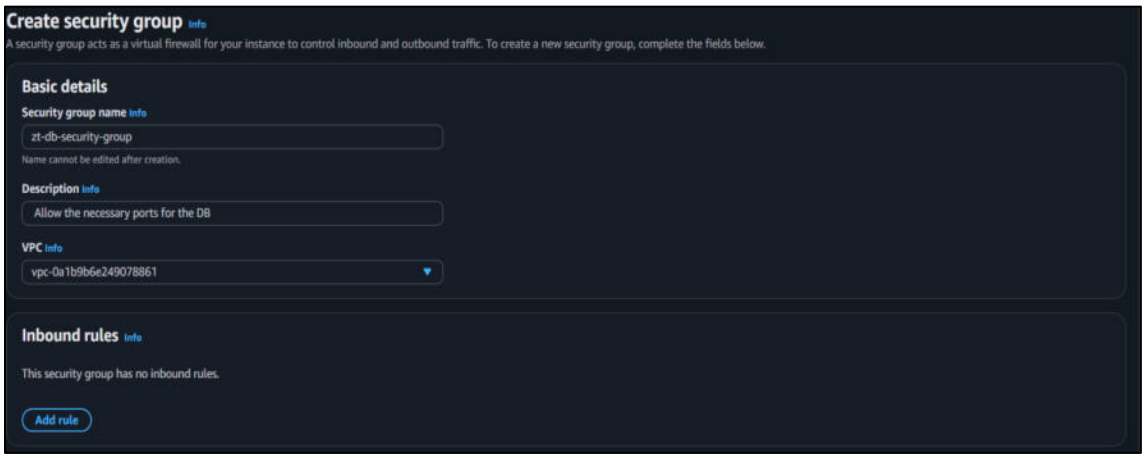


Figura 63.

Security Group Base de Datos



Instancias

Las instancias se montaron con el mismo sistema operativo el cual es Ubuntu server 26.04, pese a eso cada una de las instancias difiere en varias características como se mostrará en la siguiente tabla:

Tabla 9.

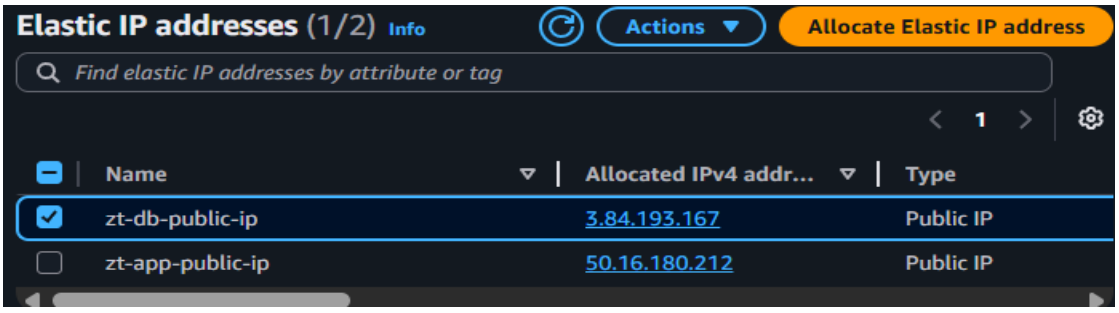
Recursos Instancias.

Recurso	zt-db-server	zt-app-server
Vcpu	2	2
Memoria GiB	2	4
Tipo de Instancia	t3.small	c7i-flex.large
Security Group	zt-db-security-group	zt-apps-security-group
Almacenamiento GiB	20 gp3	20 gp3
Elastic IP	zt-db-public-ip	zt-app-public-ip
Par de llaves	db-aws-kp.pem	app-aws-kp.pem

Como se pudo observar en la tabla 9 los recursos difieren en cosas muy puntuales, pero al final ayudan a generar una infraestructura siguiendo un lineamiento zero trust, un punto importante a recalcar es que se crearon IPs elásticas para cada una de las instancias:

Figura 64.

IPs Elasticas de Instancias



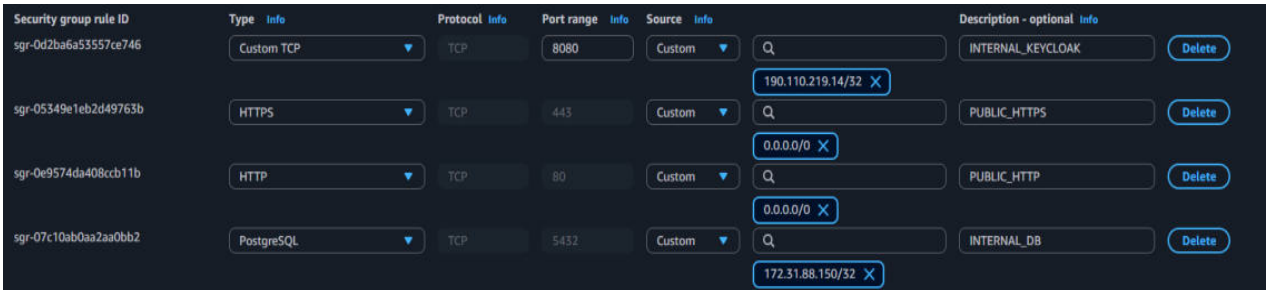
Esto quiere decir que se puede desasociar en función de su necesidad ya que en consideración la BD podría manejarse pura y netamente por desde el servidor de aplicaciones. Adicional cada una de las instancias los canales de entrada como de salida se configuran para

cada uno de ellos usando los puertos específicos.

En el caso del servidor de App tendrá habilitado los puertos para keycloak, base de datos, http, https y puerto 22 en caso de que sea necesario conectarse por ssh pero apuntando a la IP específica de la persona que competa.

Figura 65.

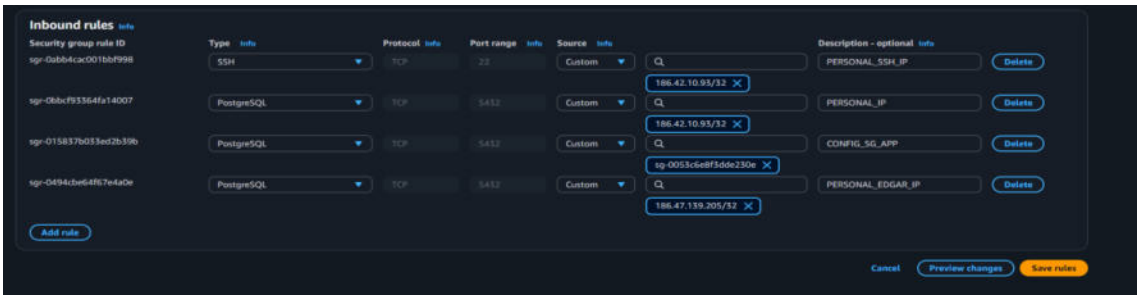
Reglas de Entrada SG app



Para el servidor de base de datos se escuchará el puerto 5432 para todo el grupo de seguridad del aplicativo ya que es el único que podrá tener acceso a la base, en caso de necesitar conexión a la base de la misma manera se podrá habilitar los puertos pertinentes.

Figura 66.

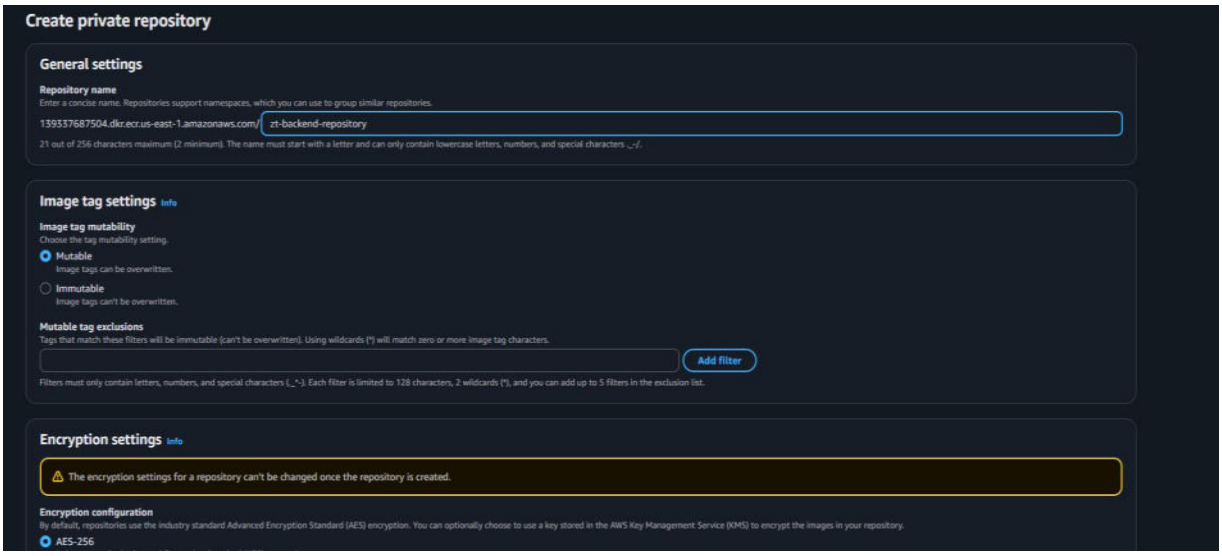
Reglas de Entrada SG BD



ECR: Se creó un repositorio privado para poder hacer el push y el pull para la imagen del servicio que se subirá al servidor de aplicativos.

Figura 67.

Creación Repositorio ECR



Montar Infraestructura

Servidor Base de datos

Usando un cliente ssh con el par de llaves app-aws-kp.pem y la IP pública del servidor usando el usuario creado por defecto nos conectamos a la instancia.

Dentro de la instancia se instaló en primer lugar Docker siguiendo el instructivo del proveedor oficial de Docker.

Figura 68.

Instalación Docker Server BD

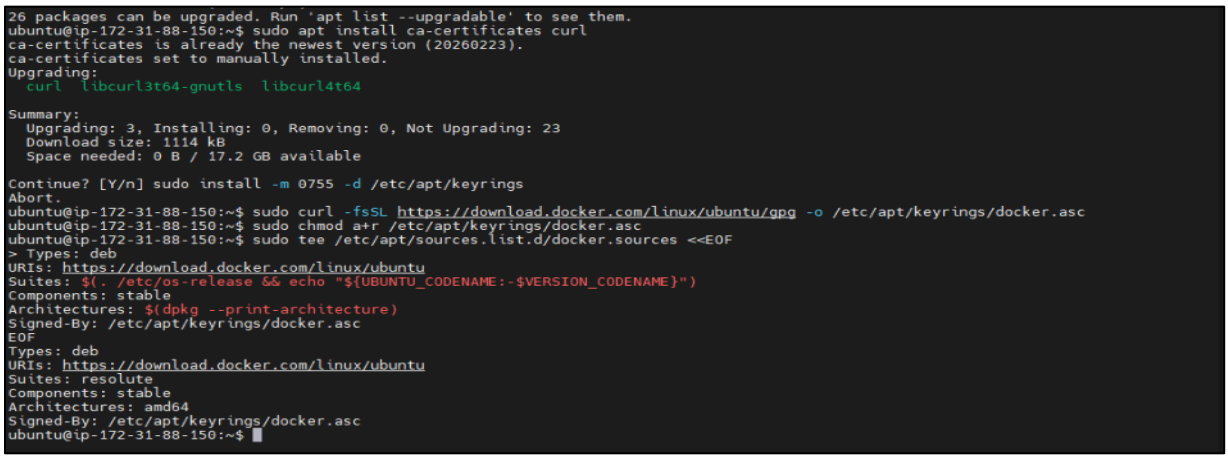
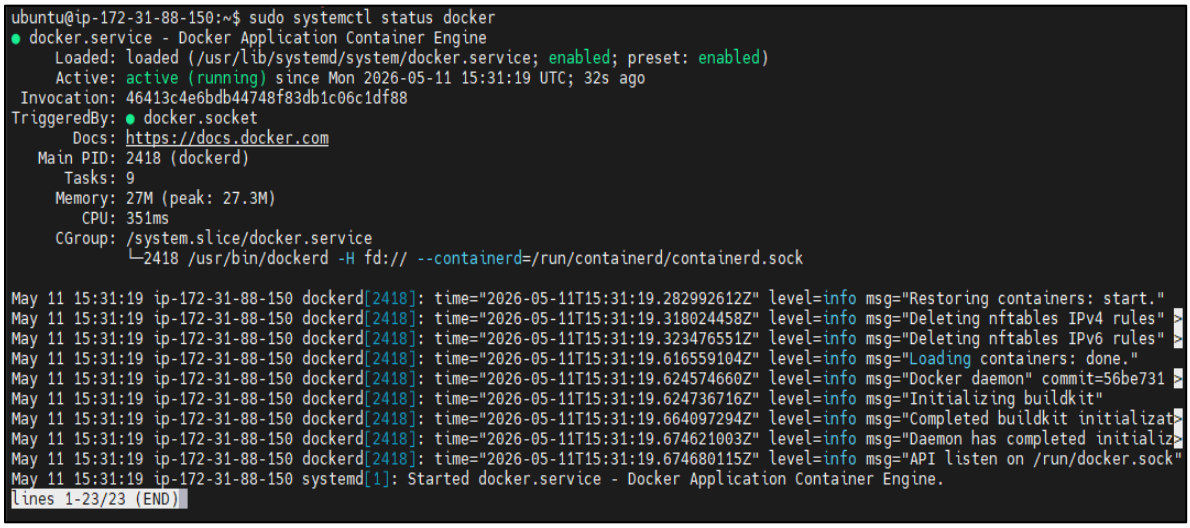


Figura 69.

Validación servicio Docker BD



Con el servidor levantado se generó un archivo de tipo docker-compose.yaml el cual tiene las configuraciones del aplicativo a levantarse el cual será postgres.

Archivo despliegue Postgres

```
      services:
        postgres:
          image: postgres:${POSTGRES_VERSION:-17.9-bookworm}
          container_name: postgres-zt
          restart: unless-stopped
          environment:
            POSTGRES_USER: ${POSTGRES_USER:-admin}
            POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-admin123}
            POSTGRES_DB: ${POSTGRES_DB:-appdb}
          ports:
            - "${POSTGRES_PORT:-5432}:5432"
          volumes:
            - postgres_data:/var/lib/postgresql/data
            - ./initdb:/docker-entrypoint-initdb.d:ro
          healthcheck:
            test: ["CMD-SHELL", "pg_isready -U $${POSTGRES_USER} -d
$${POSTGRES_DB}"]
            interval: 10s
            timeout: 5s
            retries: 10

      volumes:
        postgres_data:
```

Para levantar el servicio adicional este tiene una rchivo .env el cual tiene las variables del Docker copose este lleva claves, usuarios y n archivo 01-create-databases.sql el cual tiene scrips creando esquemas para que los aplicativos como keycloak o wso2 tengan donde montar sus bases respectivas.

Archivos de configuración Postgress

.env

```
      POSTGRES_VERSION=17.9-bookworm
      POSTGRES_USER=postgres
      POSTGRES_PASSWORD=pwd
      POSTGRES_DB=zero_trust_lab_db
POSTGRES_PORT=5432
```

01-create-databases.sql

```
      CREATE USER keycloak WITH PASSWORD 'pwd';
      CREATE DATABASE keycloakdb OWNER keycloak;

      CREATE USER wso2 WITH PASSWORD 'pwd';
      CREATE DATABASE wso2db OWNER wso2;
```

```
      CREATE DATABASE wso2_shared_db OWNER wso2;
CREATE DATABASE wso2am_db OWNER wso2;
```

Hay que tener en cuenta que estos archivos deben estar en la misma carpeta de donde está el docker-compose.yaml. Con estos archivos podemos ejecutar docker compose up -d el cuál es el comando para levantar el contenedor de postgres.

Servidor Apps

Para el servidor de app se realizaron los mismos pasos que en el servidor de BD como se puede observar en las ilustraciones 13 y 14. Con este punto de partida se ejecutó lo siguiente para levantar cada uno de los aplicativos del servidor de app:

Keycloak

Previo a que se levante el aplicativo, se genera un DNS par en duck-dns.org para poder hacer la publicación del aplicativo usando un proxi inverso usando nginx.

Tabla 10.

Comandos proxy inverso

Commandos Proxy Inverso
sudo apt install nginx -y sudo systemctl start nginx sudo systemctl enable nginx

Una vez instalado se apunta al puerto que se va a levantar el aplicativo en los archivos de configuración de nginx con esto usando certbot podemos generar los archivos para usar cadenas de confianza

sudo certbot --nginx -d keycloak-zt.duckdns.org

Para este aplicativo, así como en postgres se generaron su respectivo archivo docker-compose.yaml y .env, con las configuraciones previas ya se puede levantar este aplicativo.

Archivos configuración Keycloak

.env

```
KEYCLOAK_VERSION=26.0
KEYCLOAK_PORT=8080
KEYCLOAK_ADMIN=admin
KEYCLOAK_ADMIN_PASSWORD=pwd

KC_DB_URL_HOST=172.31.88.150
KC_DB_URL_PORT=5432
KC_DB_DATABASE=keycloakdb
KC_DB_USERNAME=keycloak
KC_DB_PASSWORD=pwd
```

docker-compose.yaml

```
services:
  keycloak:
    image: quay.io/keycloak/keycloak:${KEYCLOAK_VERSION:-26.0}
    container_name: keycloak-dev
    restart: unless-stopped
    command: start
    environment:
      KC_DB: postgres
      KC_DB_URL_HOST: ${KC_DB_URL_HOST:-172.31.88.150}
      KC_DB_URL_PORT: ${KC_DB_URL_PORT:-5432}
      KC_DB_URL_DATABASE: ${KC_DB_DATABASE:-keycloakdb}
      KC_DB_USERNAME: ${KC_DB_USERNAME:-keycloak}
      KC_DB_PASSWORD: ${KC_DB_PASSWORD:-keycloak123}

      KC_HTTP_ENABLED: "true"
      KC_HOSTNAME: "https://keycloak-zt.duckdns.org"
      KC_HOSTNAME_STRICT: "true"
      KC_PROXY_HEADERS: "xforwarded"

      KC_BOOTSTRAP_ADMIN_USERNAME: ${KEYCLOAK_ADMIN:-admin}
      KC_BOOTSTRAP_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD:-admin123}

    ports:
      - "127.0.0.1:${KEYCLOAK_PORT:-8080}:8080"
```

WSO2

Para WSO debido a que ya se tiene levantado el proxy inverso lo único que se hace es generar la cadena de confianza y apuntar a los puertos de los productos de WSO2 hay que tener en cuenta que en wso2 se generan dos dominios puesto que el Gateway va a ser el canal de salida de los servicios.

sudo certbot --nginx -d wso2-zt.duckdns.org

sudo certbot --nginx -d wso2gw-zt.duckdns.org

Configuración proxy WSO2

wso2

```
server {
    listen 80;
    server_name wso2-zt.duckdns.org;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name wso2-zt.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/wso2-zt.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/wso2-zt.duckdns.org/privkey.pem;

    client_max_body_size 100m;
    location / {
        proxy_pass https://127.0.0.1:9443;
        proxy_ssl_verify off;
        proxy_ssl_server_name on;
        proxy_http_version 1.1;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port 443;
```

```
            proxy_connect_timeout 120s;
            proxy_send_timeout 120s;
            proxy_read_timeout 120s;

            proxy_buffer_size 128k;
            proxy_buffers 4 256k;
            proxy_busy_buffers_size 256k;
        }
}
```

wso2-gateway

```
    server {
        listen 80;
        server_name wso2gw-zt.duckdns.org;

        return 301 https://$host$request_uri;
    }

    server {
        listen 443 ssl;
        server_name wso2gw-zt.duckdns.org;

        ssl_certificate /etc/letsencrypt/live/wso2gw-zt.duckdns.org/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/wso2gw-
zt.duckdns.org/privkey.pem;

        client_max_body_size 100m;

        location / {
            proxy_pass https://127.0.0.1:8243;

            proxy_ssl_verify off;
            proxy_ssl_server_name on;
            proxy_http_version 1.1;

            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;

            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto https;
            proxy_set_header X-Forwarded-Host $host;
            proxy_set_header X-Forwarded-Port 443;
            proxy_connect_timeout 120s;
            proxy_send_timeout 120s;
            proxy_read_timeout 120s;
            proxy_buffer_size 128k;
            proxy_buffers 4 256k;
            proxy_busy_buffers_size 256k;
        }
}
```

Con los dominios configurados el siguiente paso es aplicar los artefactos necesarios con las configuraciones pertinentes para montar el contenedor.

docker-compose.yaml

```
services:
  wso2am:
    build:
      context: .
      dockerfile: Dockerfile
    image: wso2am-postgres:4.4.0
    container_name: wso2am-single-node
    restart: unless-stopped

    environment:
      JAVA_OPTS: ${JAVA_OPTS:--Xms1024m -Xmx2048m}
      WSO2_SHARED_DB_URL: ${WSO2_SHARED_DB_URL}
      WSO2_APIM_DB_URL: ${WSO2_APIM_DB_URL}
      WSO2_DB_USERNAME: ${WSO2_DB_USERNAME:-wso2}
      WSO2_DB_PASSWORD: ${WSO2_DB_PASSWORD}

    ports:
      - "127.0.0.1:${WSO2_HTTPS_PORT:-9443}:9443"
      - "127.0.0.1:${WSO2_GATEWAY_HTTPS_PORT:-8243}:8243"
      - "127.0.0.1:${WSO2_GATEWAY_HTTP_PORT:-8280}:8280"
      - "127.0.0.1:${WSO2_PASS_THROUGH_HTTP_PORT:-9763}:9763"

    volumes:
      - ./conf/deployment.toml:${WSO2_HOME:-/home/wso2carbon/wso2am-
4.4.0}/repository/conf/deployment.toml:ro
      - wso2am_repository:${WSO2_HOME:-/home/wso2carbon/wso2am-
4.4.0}/repository/deployment/server

volumes:
  wso2am_repository:
```

.env

```
WSO2AM_VERSION=4.4.0
WSO2_HOME=/home/wso2carbon/wso2am-4.4.0
WSO2_HTTPS_PORT=9443
WSO2_GATEWAY_HTTPS_PORT=8243
WSO2_GATEWAY_HTTP_PORT=8280
WSO2_PASS_THROUGH_HTTP_PORT=9763

JAVA_OPTS=-Xms1024m -Xmx2048m
WSO2_SHARED_DB_URL=jdbc:postgresql://172.31.88.150:5432/wso2_shared_db
WSO2_APIM_DB_URL=jdbc:postgresql://172.31.88.150:5432/wso2am_db
WSO2_DB_USERNAME=wso2
```

```
WSO2_DB_PASSWORD=pwd
```

deploy.toml

```
[server]
hostname = "wso2-zt.duckdns.org"
node_ip = "127.0.0.1"
base_path = "https://wso2-zt.duckdns.org"

[transport.https.properties]
proxyPort = 443

[transport.http.properties]
proxyPort = 80

[super_admin]
username = "admin"
password = "pwd"
create_admin_account = true

[user_store]
type = "database_unique_id"

[database.shared_db]
type = "postgre"
url = "$env{WSO2_SHARED_DB_URL}"
username = "$env{WSO2_DB_USERNAME}"
password = "$env{WSO2_DB_PASSWORD}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"

[database.apim_db]
type = "postgre"
url = "$env{WSO2_APIM_DB_URL}"
username = "$env{WSO2_DB_USERNAME}"
password = "$env{WSO2_DB_PASSWORD}"
driver = "org.postgresql.Driver"
validationQuery = "SELECT 1"

[keystore.tls]
file_name = "wso2carbon.jks"
type = "JKS"
password = "wso2carbon"
alias = "wso2carbon"
key_password = "wso2carbon"

[keystore.primary]
```

```
file_name = "wso2carbon.jks"
type = "JKS"
password = "wso2carbon"
alias = "wso2carbon"
key_password = "wso2carbon"
[keystore.internal]
file_name = "wso2carbon.jks"
type = "JKS"
password = "wso2carbon"
alias = "wso2carbon"
key_password = "wso2carbon"

[truststore]
file_name = "client-truststore.jks"
type = "JKS"
password = "wso2carbon"
[[apim.gateway.environment]]
name = "Default"
type = "hybrid"
display_in_api_console = true
description = "Default Gateway"
service_url = "https://wso2-zt.duckdns.org/services/"
username = "${admin.username}"
password = "${admin.password}"

ws_endpoint = "ws://wso2gw-zt.duckdns.org"
wss_endpoint = "wss://wso2gw-zt.duckdns.org"
http_endpoint = "http://wso2gw-zt.duckdns.org"
https_endpoint = "https://wso2gw-zt.duckdns.org"
[apim.devportal]
url = "https://wso2-zt.duckdns.org/devportal"
enable_key_provisioning = true
[apim.key_manager]
enable_provisioned_app_validation = false
[apim.oauth_config]
enable_outbound_auth_header = true
[apim.publisher]
url = "https://wso2-zt.duckdns.org/publisher"
[apim.analytics]
enable = false
[apim.cors]
allow_origins = [
    "https://wso2-zt.duckdns.org"
]
allow_methods = [
    "GET",
    "PUT",
    "POST",
    "DELETE",
    "PATCH",
    "OPTIONS"
]
allow_headers = [
    "authorization",
```

```
    "Access-Control-Allow-Origin",
    "Content-Type",
    "SOAPAction",
    "apikey",
    "Internal-Key"
]
allow_credentials = false
```

App

Para desplegar el aplicativo se genera el certificado usando el comando

sudo certbot --nginx -d zt-api-secure.duckdns.org

Y se aplica los archivos de despliegue de Docker respectivos

docker-compose.yaml

```
      services:
        zero-trust-api-services:
          image: 139337687504.dkr.ecr.us-east-1.amazonaws.com/zt-backend-
repository:1.0.0
          container_name: zero-trust-api-services
          restart: unless-stopped
          pull_policy: always

          # La aplicación escucha en 8080 dentro del contenedor.
          # En el servidor queda publicada en 8081 para no cruzarse con Keycloak.
          ports:
            - "127.0.0.1:8081:8080"
          env_file:
            - .env
          environment:
            TZ: ${TZ}
            SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE}
            SERVER_PORT: 8080
            DB_HOST: ${DB_HOST}
            DB_PORT: ${DB_PORT}
            DB_NAME: ${DB_NAME}
            DB_USERNAME: ${DB_USERNAME}
            DB_PASSWORD: ${DB_PASSWORD}

            OIDC_ISSUER_URI: ${OIDC_ISSUER_URI}
```

```
        OIDC_CLIENT_ID: ${OIDC_CLIENT_ID}
      mem_limit: 1g
      cpus: "0.25"
      networks:
        - zero-trust-network
  networks:
    zero-trust-network:
driver: bridge
```

3.1.8. Configuración de Keycloak como proveedor de identidad

Para la capa de autenticación y autorización se configuró Keycloak como proveedor de identidad centralizado del laboratorio. Esta configuración responde al enfoque definido en el proyecto, donde la primera capa de seguridad corresponde a la gestión de identidad y emisión de tokens mediante Keycloak, mientras que el backend valida dichos tokens antes de permitir el acceso a los recursos protegidos. Esta decisión se alinea con el modelo propuesto, en el cual el usuario se autentica en Keycloak, recibe un token JWT firmado y posteriormente consume las APIs protegidas.

El objetivo principal de esta configuración fue desacoplar la gestión de usuarios y permisos del código del backend. De esta forma, el microservicio no administra usuarios ni contraseñas directamente, sino que delega esa responsabilidad a Keycloak. El backend únicamente valida el token JWT recibido y aplica reglas de autorización según los roles incluidos en dicho token.

Creación del Realm

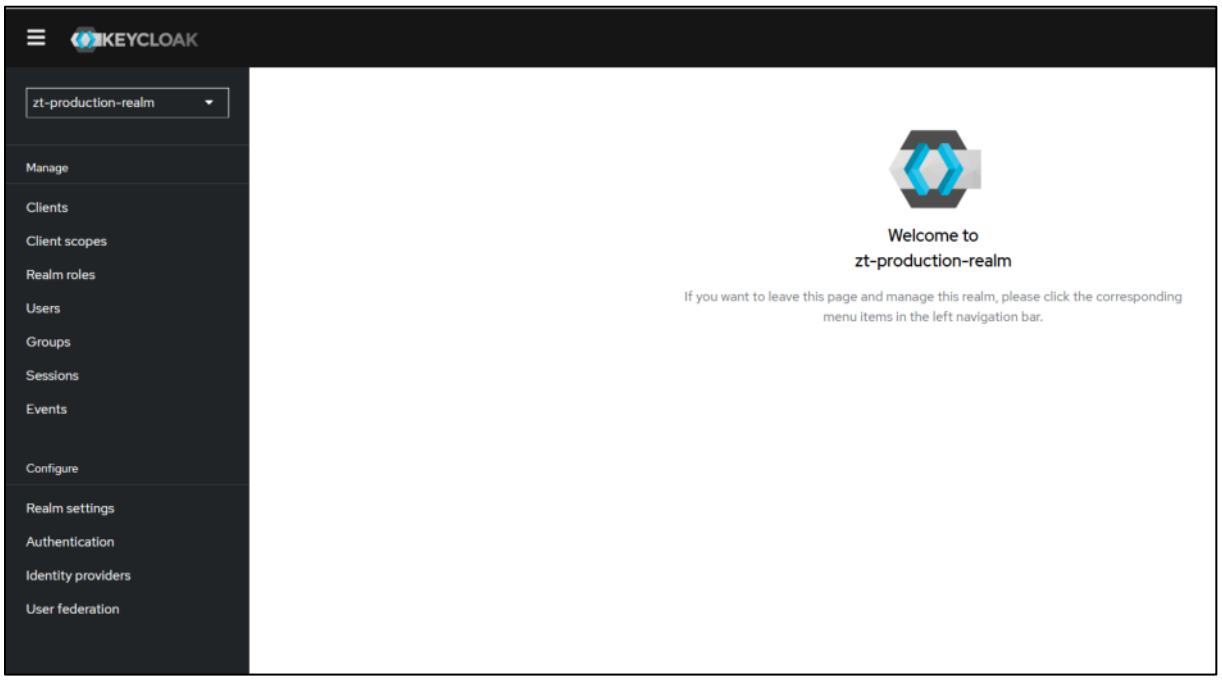
En Keycloak se creó un realm independiente denominado:

zt-production-realm

Este realm agrupa todos los elementos de seguridad del laboratorio, incluyendo usuarios, roles, clientes y configuración de autenticación. La separación mediante un realm propio que permite aislar la configuración del proyecto respecto al realm administrativo principal de Keycloak.

Figura 70.

Realm zt-production-realm configurado en Keycloak



Se creó el realm zt-production-realm con el propósito de aislar la configuración de seguridad del laboratorio. Dentro de este realm se gestionaron los usuarios de prueba, los roles de autorización y el cliente OIDC utilizado por el backend. Esta separación permite mantener una administración ordenada de las identidades y evita mezclar configuraciones del proyecto con el realm administrativo de Keycloak.

Creación del cliente OIDC para el backend

Dentro del realm zt-production-realm se creó el cliente:

zt-backend-client

Este cliente representa al microservicio backend dentro de Keycloak. Su función es permitir que los tokens emitidos por Keycloak incluyan los roles asociados al backend y puedan ser validados posteriormente por Spring Security.

El cliente fue configurado como un cliente de tipo OpenID Connect, con autenticación de cliente habilitada. Durante la etapa de laboratorio se mantuvo activo el flujo de acceso directo para generar tokens desde Postman y Swagger.

Configuración aplicada al cliente:

Tabla 11.

Parámetros de configuración del cliente backend en Keycloak

Parámetro	**Valor configurado**
Client ID	zt-backend-client
Name	Zero Trust Backend Client
Client authentication	ON
Authorization	OFF
Standard Flow	ON
Direct access grants	ON
Implicit Flow	OFF
Service accounts roles	OFF

Figura 71.

Configuración general del cliente OIDC zt-backend-client

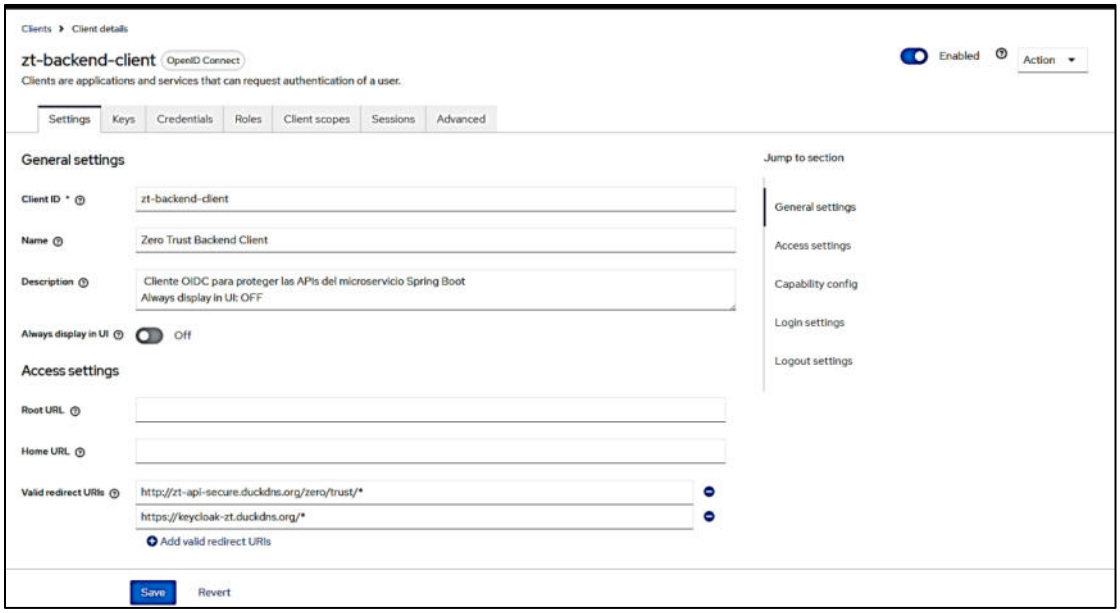
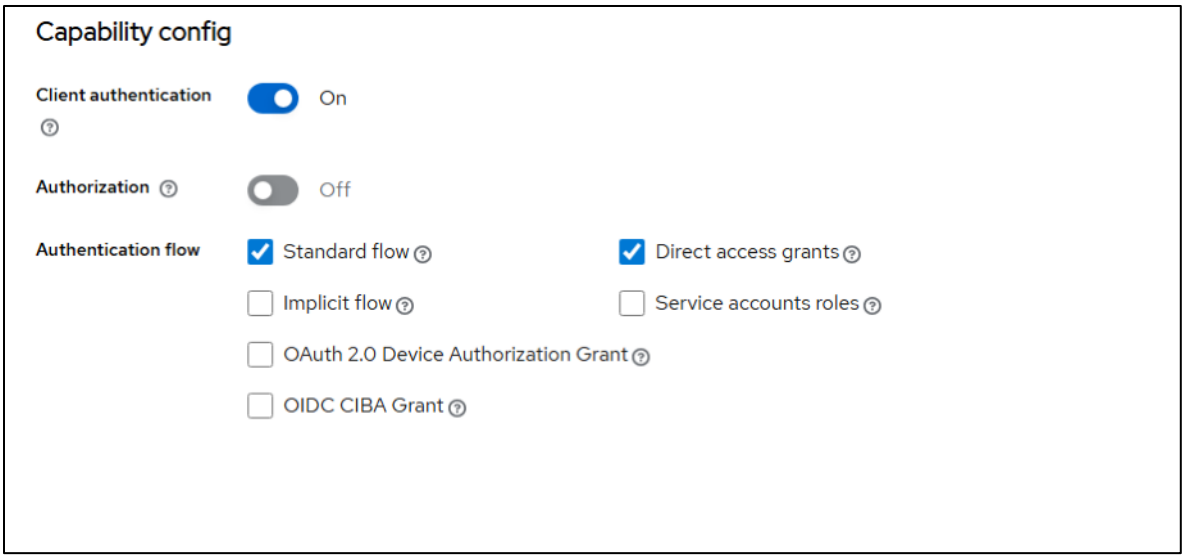


Figura 72.

Configuración de capacidades del cliente zt-backend-client



Se configuró el cliente OIDC zt-backend-client para representar al backend dentro del realm zt-production-realm. Este cliente fue utilizado para asociar roles específicos a las APIs protegidas y permitir que los tokens emitidos por Keycloak incluyan los permisos correspondientes. La autenticación del cliente se mantuvo habilitada debido a que se trata de un cliente confidencial, mientras que el flujo Direct Access Grants fue utilizado únicamente para pruebas controladas desde Postman.

3.1.9. Configuración de tokens JWT

La autenticación del laboratorio se basó en tokens JWT emitidos por Keycloak. El proyecto plantea el uso de JWT firmados con RS256, incluyendo claims obligatorios como iss, sub, aud, iat, exp, nbf, jti y permisos específicos del usuario.

En la implementación, el token generado por Keycloak incluyó información del usuario autenticado, el cliente autorizado y los roles asignados. Los roles se ubicaron dentro del claim:

Figura 73.

Generación de token JWT desde Postman mediante Keycloak

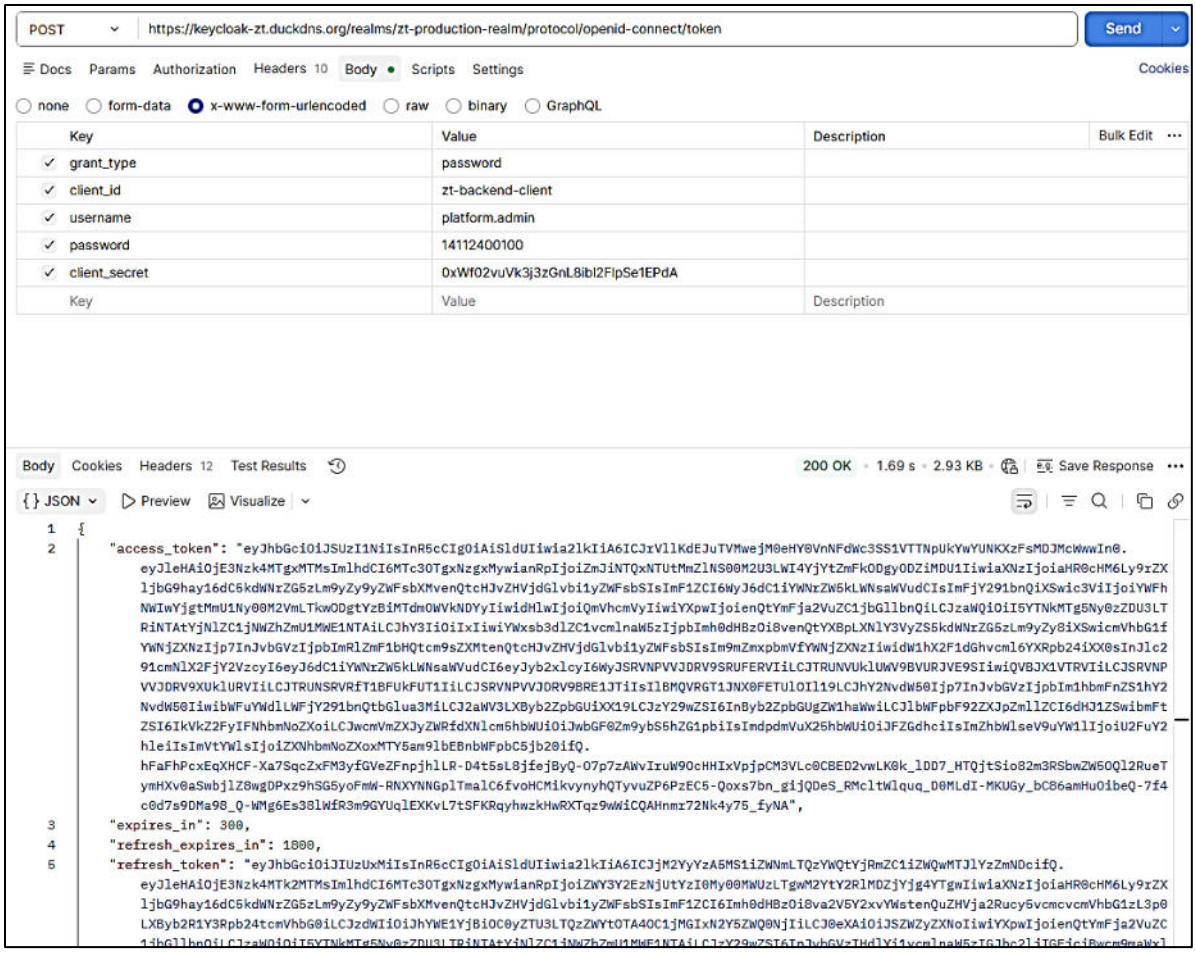


Figura 74.

Claims de firma del token JWT emitido por Keycloak

JSON Claims Breakdown


```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "kVYJtBnMS0z34xv4VsEug7I-UM3iRF0aCJ_1l02Lql0"
}
```

Figura 75.

Claims principales del token JWT emitido por Keycloak

JSON Claims Breakdown


```
{
  "exp": 1779818113,
  "iat": 1779817813,
  "jti": "fbb54155-2fe5-43e7-b8b6-fad88286b055",
  "iss": "https://keycloak-zt.duckdns.org/realms/zt-production-realm",
  "aud": [
    "zt-backend-client",
    "account"
  ],
  "sub": "aaa5b0b8-2e57-43ef-9088-c0b17f9ed462",
  "typ": "Bearer",
  "azp": "zt-backend-client",
  "sid": "9a3d1897-3d57-4b50-b3ed-c5fafe51a550",
  "acr": "1",
  "allowed-origins": [
    "https://zt-api-secure.duckdns.org/"
  ],
  "realm_access": {
    "roles": [
      "default-roles-zt-production-realm",
      "offline_access",
      "uma_authorization"
    ]
  },
  "resource_access": {
    "zt-backend-client": {
      "roles": [
        "RESOURCE_READER",
        "SECURITY_AUDITOR",
        "API_USER",
        "RESOURCE_WRITER",
        "SECRET_OPERATOR",
        "RESOURCE_ADMIN",
        "PLATFORM_ADMIN"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "profile email",
  "email_verified": true,
  "name": "Edgar Sanchez",
  "preferred_username": "platform.admin",
  "given_name": "Edgar",
  "family_name": "Sanchez",
  "email": "esanchez1169joel@gmail.com"
}
```


El token JWT emitido por Keycloak contiene los claims necesarios para identificar al usuario, validar la procedencia del token y determinar los permisos asociados al cliente zt-backend-client. Entre los claims verificados se encuentran el emisor, la audiencia, el

identificador del usuario, el nombre de usuario y los roles definidos para el backend. Esta información es utilizada por Spring Security para permitir o denegar el acceso a las APIs protegidas.

Integración del backend con Keycloak

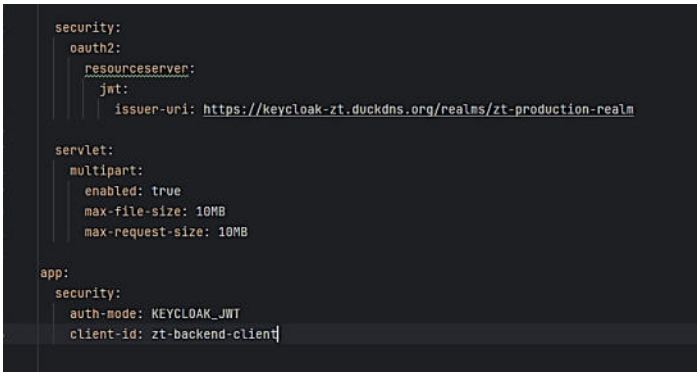
Para integrar el backend con Keycloak se configuró Spring Boot como **Resource Server**. Esto permitió que el microservicio valide tokens JWT emitidos por el realm zt-production-realm.

La configuración final ya no utiliza el adaptador antiguo de Keycloak, sino la integración nativa de Spring Security para OAuth2 Resource Server. Esto permite validar el token mediante el issuer-uri del realm.

3.1.10. Configuración aplicada en el backend:

Figura 76.

Configuración del backend como Resource Server para validar tokens de Keycloak



El backend fue configurado como Resource Server mediante Spring Security. Esta configuración permite validar automáticamente los tokens JWT emitidos por Keycloak a partir del issuer-uri del realm zt-production-realm. Además, se configuró el client-id zt-backend-client

para extraer los roles definidos en el claim resource_access y aplicar autorización granular sobre cada API.

Durante la implementación se descartó el uso del adaptador antiguo de Keycloak y se adoptó la validación nativa de JWT mediante Spring Security OAuth2 Resource Server. Esta decisión permitió alinear la implementación con el uso de tokens estándar y evitar dependencias innecesarias en el backend.

3.1.11. Roles definidos en Keycloak

Los roles de autorización fueron definidos como roles de cliente dentro de zt-backend-client. Esta decisión permite aplicar el principio de mínimo privilegio sobre las APIs del microservicio, ya que cada permiso está asociado directamente a una operación del backend. De esta manera, los usuarios reciben únicamente los roles necesarios para ejecutar sus funciones dentro del laboratorio.

Tabla 12.

Tabla de roles

Rol	Descripción
API_USER	Rol base para usuarios autenticados que pueden consumir APIs generales del sistema.
RESOURCE_READER	Permite listar y consultar recursos protegidos.

RESOURCE_WRITER	Permite crear y actualizar recursos protegidos.
RESOURCE_ADMIN	Permite inactivar recursos protegidos y ejecutar operaciones administrativas sobre recursos.
SECURITY_AUDITOR	Permite consultar eventos de auditoría, escenarios de prueba, ejecuciones y métricas de seguridad.
PLATFORM_ADMIN	Rol administrador con acceso completo a las APIs del laboratorio.

Figura 77.

Roles de cliente definidos para zt-backend-client

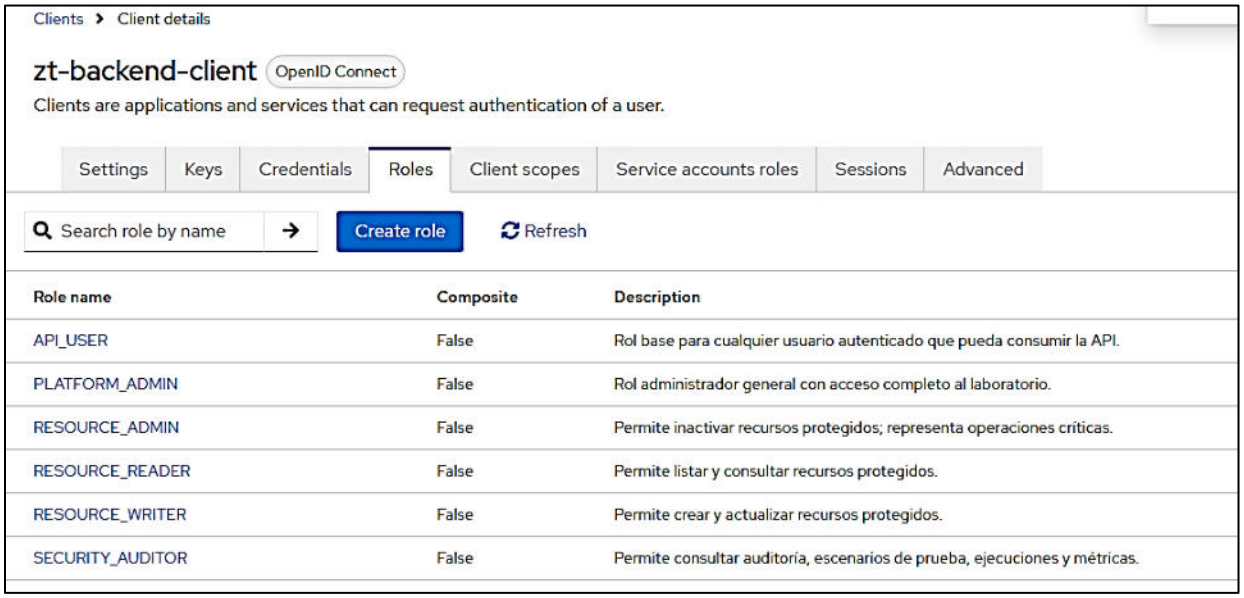


Tabla 13.

Matriz de autorización por API

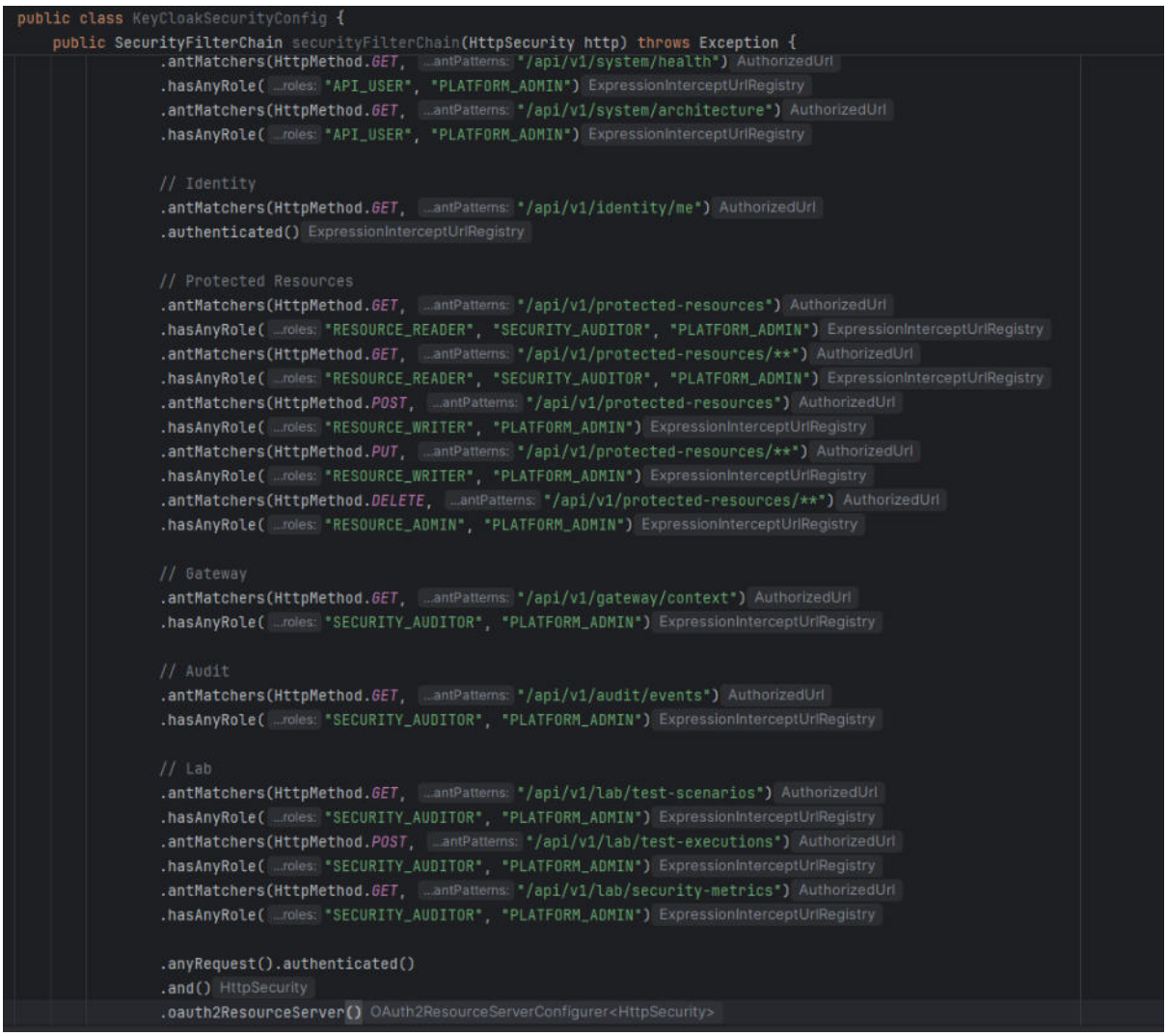
N.º	Módulo	Método	Endpoint	Roles autorizados
1	System	GET	/api/v1/system/health	API_USER, PLATFORM_ADMIN
2	System	GET	/api/v1/system/architecture	API_USER, PLATFORM_ADMIN
3	Identity	GET	/api/v1/identity/me	Cualquier usuario autenticado
4	Protected Resources	GET	/api/v1/protected-resources	RESOURCE_READER, SECURITY_AUDITOR, PLATFORM_ADMIN
5	Protected Resources	GET	/api/v1/protected-resources/**	RESOURCE_READER, SECURITY_AUDITOR, PLATFORM_ADMIN
6	Protected Resources	POST	/api/v1/protected-resources	RESOURCE_WRITER, PLATFORM_ADMIN
7	Protected Resources	PUT	/api/v1/protected-resources/**	RESOURCE_WRITER, PLATFORM_ADMIN
8	Protected Resources	DELETE	/api/v1/protected-resources/**	RESOURCE_ADMIN, PLATFORM_ADMIN

9	Gateway	GET	/api/v1/gateway/context	SECURITY_AUDITOR, PLATFORM_ADMIN
10	Audit	GET	/api/v1/audit/events	SECURITY_AUDITOR, PLATFORM_ADMIN
11	Lab	GET	/api/v1/lab/test-scenarios	SECURITY_AUDITOR, PLATFORM_ADMIN
12	Lab	POST	/api/v1/lab/test-executions	SECURITY_AUDITOR, PLATFORM_ADMIN
13	Lab	GET	/api/v1/lab/security-metrics	SECURITY_AUDITOR, PLATFORM_ADMIN

La matriz de autorización permitió establecer qué roles pueden ejecutar cada operación del backend. Las operaciones de lectura fueron separadas de las operaciones de escritura, administración, auditoría y validación de secretos. Esta separación permite aplicar mínimo privilegio y reducir el riesgo de escalamiento de privilegios dentro del laboratorio.

Figura 78.

Configuración de reglas de autorización por rol en el backend



3.1.12. Pruebas

Usuarios de prueba configurados en Keycloak

Para validar la autorización se crearon usuarios con distintos niveles de privilegio. Cada usuario representa un perfil específico dentro del laboratorio.

Tabla 14.

Usuarios de prueba configurados en Keycloak

Usuario	Roles asignados	Propósito de prueba
reader.user	API_USER, RESOURCE_READER	Validar acceso de solo lectura.
writer.user	API_USER, RESOURCE_READER, RESOURCE_WRITER	Validar creación y actualización de recursos.
resource.admin	API_USER, RESOURCE_READER, RESOURCE_WRITER, RESOURCE_ADMIN	Validar operaciones administrativas sobre recursos.
security.auditor	API_USER, SECURITY_AUDITOR	Validar acceso a auditoría, escenarios y métricas.
platform.admin	Todos los roles	Validar acceso total al laboratorio.

Figura 79.

Usuarios de prueba creados en el realm zt-production-realm

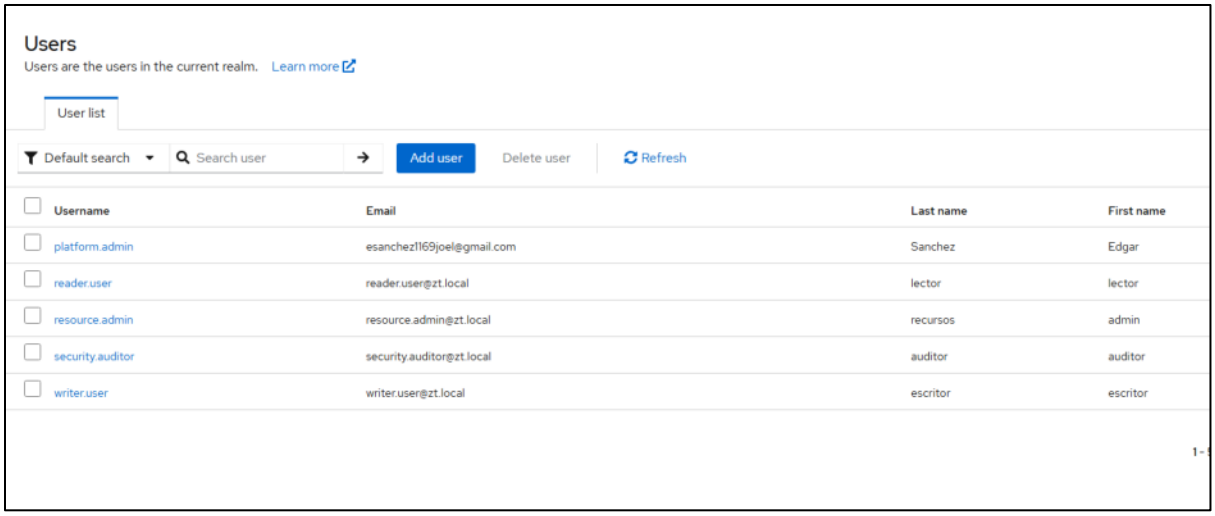
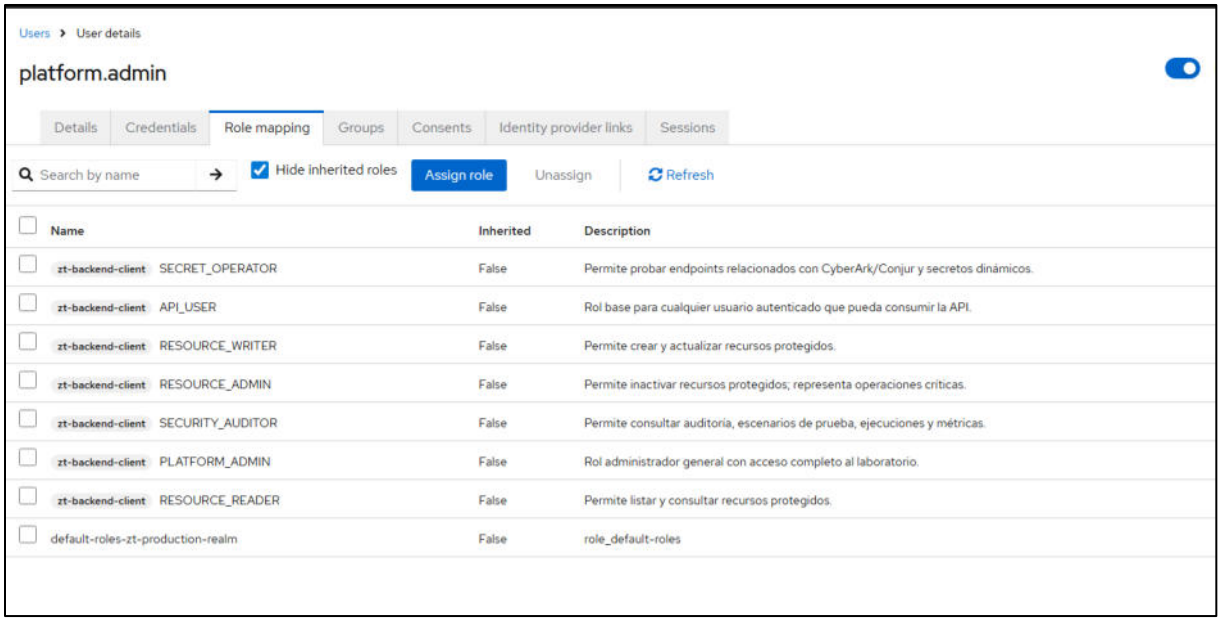


Figura 80.

Asignación de roles de cliente al usuario platform.admin



Se crearon usuarios de prueba con roles diferenciados para validar el comportamiento del sistema ante distintos niveles de privilegio. Esta estrategia permitió comprobar que un usuario autenticado no necesariamente puede ejecutar todas las operaciones, sino únicamente aquellas autorizadas por los roles asignados en Keycloak.

Prueba de generación del token

La generación del token se realizó desde Postman mediante el endpoint de token de Keycloak:

https://keycloak-zt.duckdns.org/realms/zt-production-realm/protocol/openid-connect/token

Se utilizó el método POST con cuerpo x-www-form-urlencoded.

Parámetros usados:

Tabla 15.

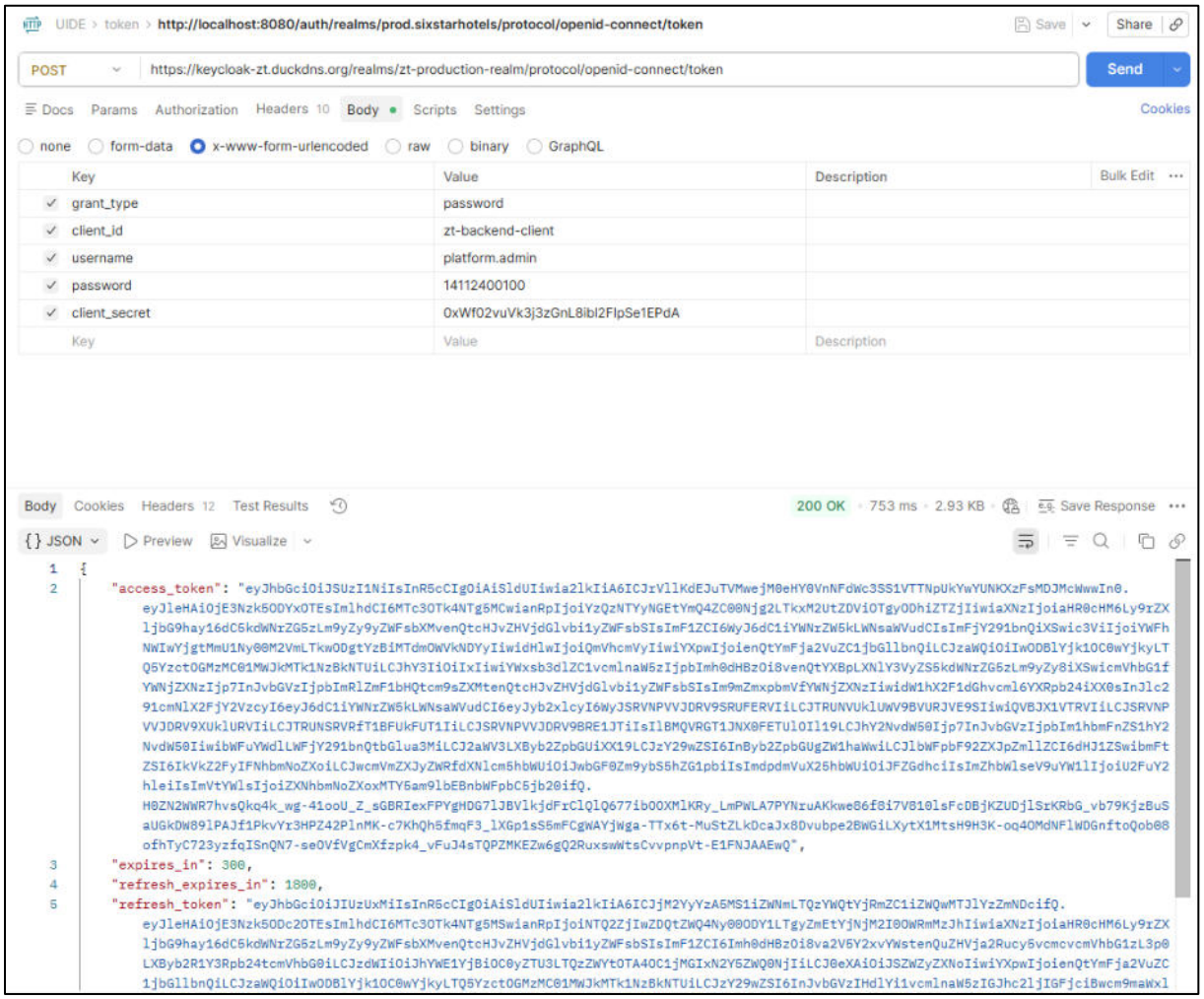
Prueba de generación de tokens de acceso

Parámetro	Descripción
grant_type	Flujo utilizado para solicitar el token. En pruebas se usó password.
client_id	Cliente OIDC configurado en Keycloak: zt-backend-client.
client_secret	Secreto del cliente confidencial. No debe exponerse en el informe.
Username	Usuario de prueba creado en Keycloak.
Password	Contraseña del usuario de prueba. No debe exponerse en el informe.

La opción Direct Access Grants y el flujo password grant fueron utilizados únicamente para pruebas controladas desde Postman. Su finalidad fue validar la generación del token de acceso y el consumo inicial de las APIs protegidas dentro del laboratorio, por lo que no representan el mecanismo final de autenticación para producción.

Figura 81.

Solicitud de generación de token JWT desde Postman



Para validar la emisión de tokens, se realizó una solicitud POST al endpoint de token de Keycloak utilizando Postman. La respuesta incluyó un access_token firmado, el tiempo de expiración del token, el tipo de token y un refresh_token. Para efectos de documentación, los valores sensibles fueron ocultados, manteniendo únicamente la evidencia de que el token fue emitido correctamente.

Prueba de autenticación exitosa

Se probó el consumo de la API:

GET /api/v1/system/health

Utilizando un token válido emitido por Keycloak. El backend respondió correctamente con código HTTP 200, lo que demuestra que el token fue aceptado por Spring Security.

Figura 82.

Ingreso del token valido generado en Postman

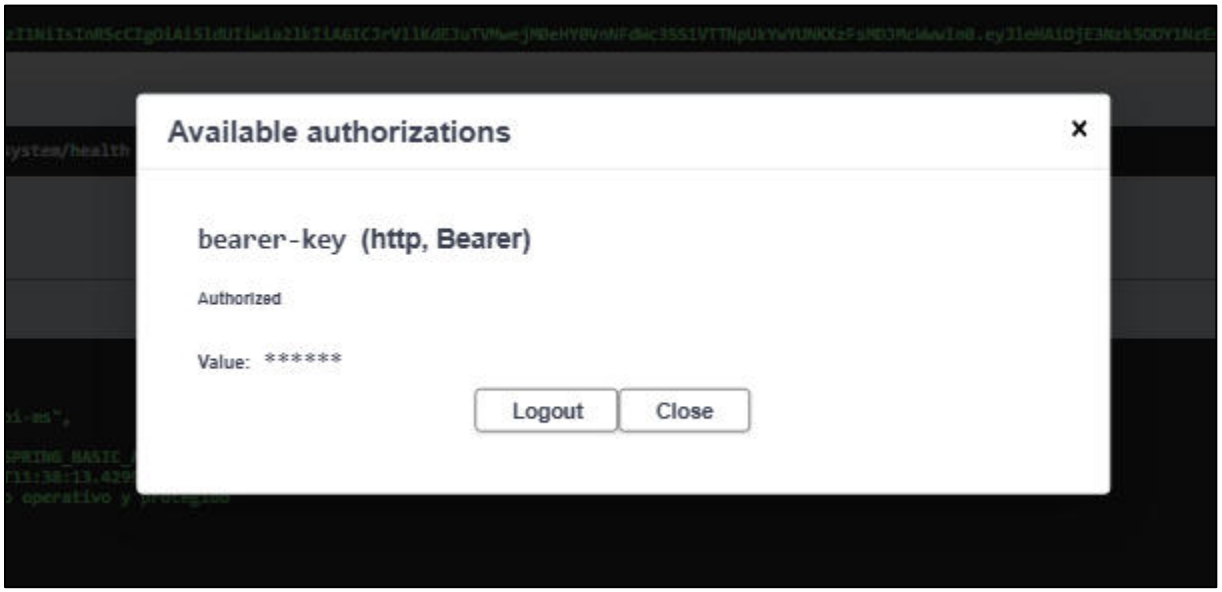
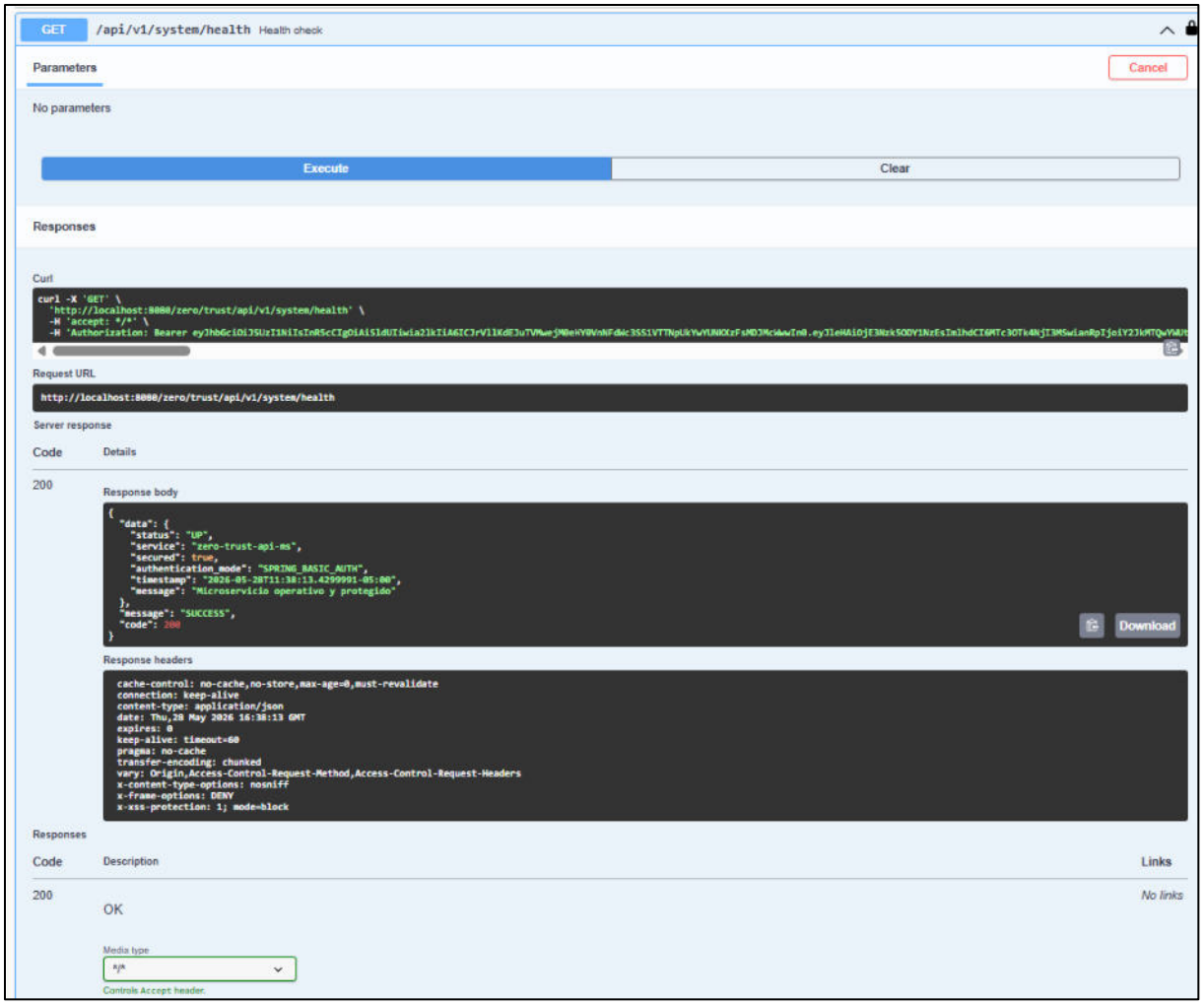


Figura 83.

Consumo exitoso de API protegida con token JWT válido



La primera prueba consistió en consumir una API general del sistema utilizando un token JWT válido. El resultado obtenido fue HTTP 200, lo que evidencia que el backend validó correctamente la firma, vigencia y procedencia del token emitido por Keycloak.

Prueba de autorización insuficiente

Se usó el usuario:

reader.user

con roles:

API_USER

RESOURCE_READER

Luego se intentó consumir la API:

POST /api/v1/protected-resources

Esta API requiere:

RESOURCE_WRITER

PLATFORM_ADMIN

Como reader.user no tiene ninguno de esos roles, el backend respondió:

403 Forbidden

Figura 84.

Generación del token con el usuario reader.user mediante Postman

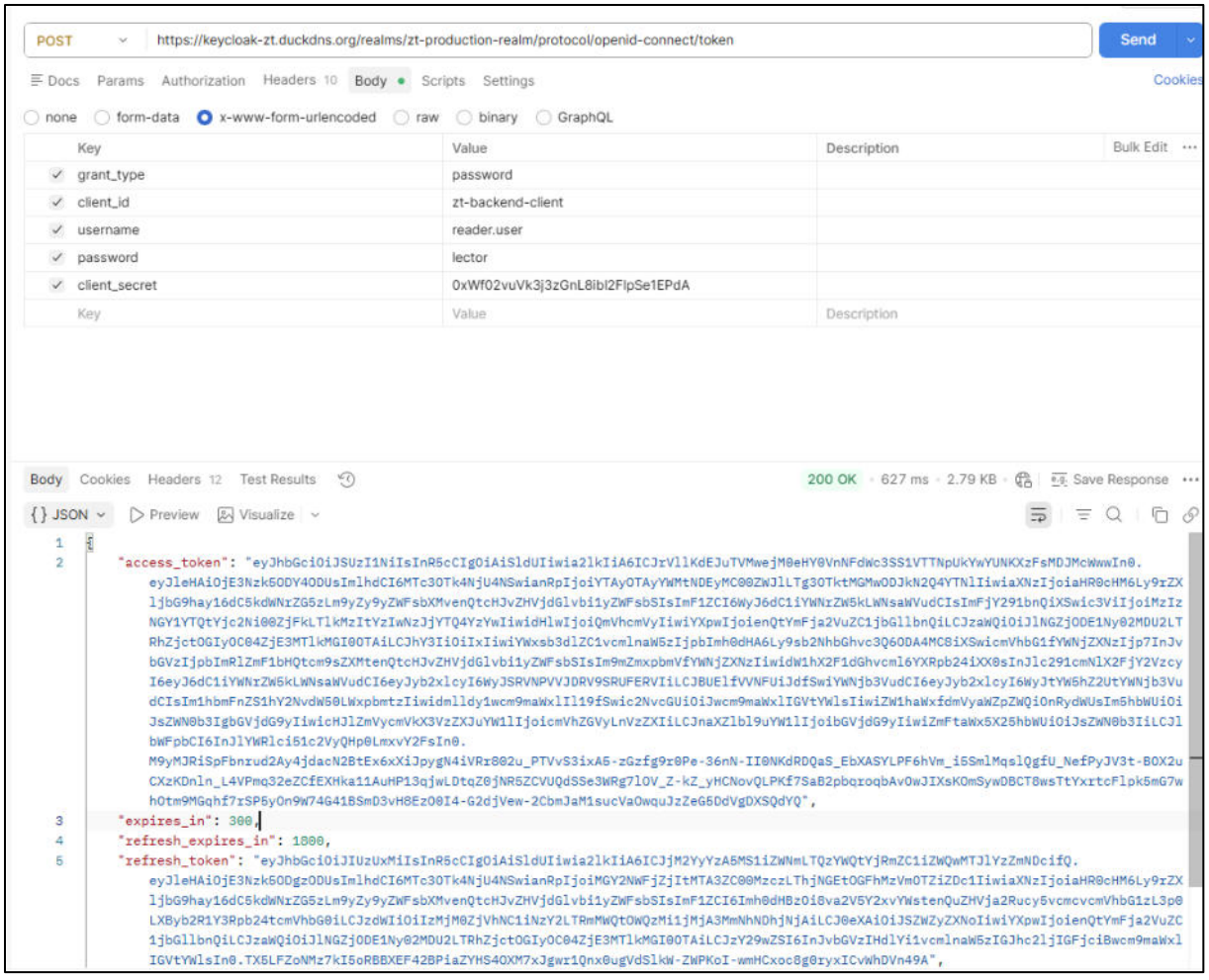
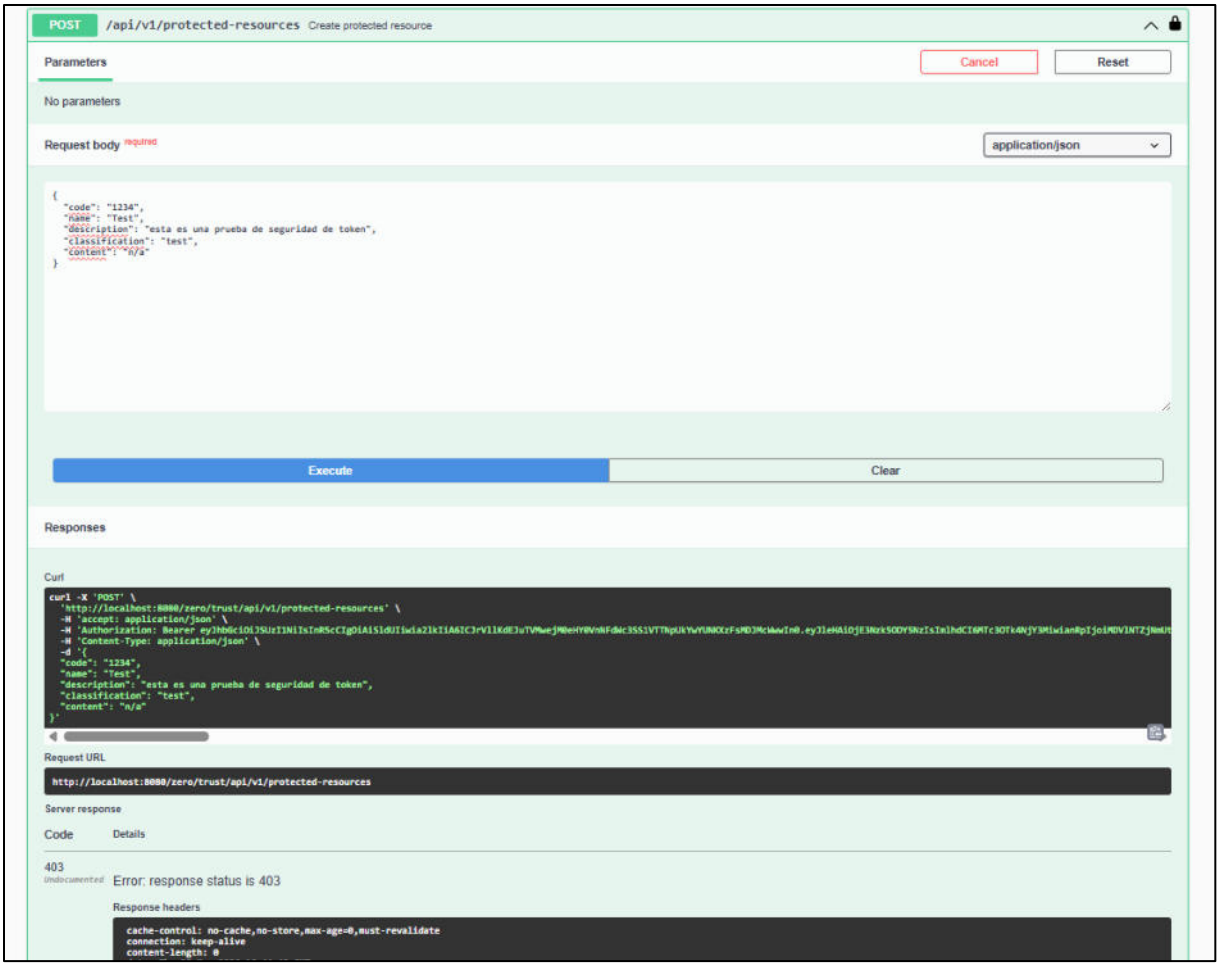


Figura 85.

Bloqueo de creación de recurso protegido por privilegios insuficientes



Se ejecutó una prueba de autorización utilizando el usuario reader.user. Este usuario cuenta con permisos de lectura, pero no posee el rol RESOURCE_WRITER requerido para crear recursos protegidos. Al intentar consumir la API POST /api/v1/protected-resources, el backend respondió con HTTP 403 Forbidden. Este resultado demuestra que el sistema no solo valida la autenticación del usuario, sino también los privilegios específicos necesarios para ejecutar cada operación.

Prueba de autorización exitosa para creación

Usuario:

writer.user

Roles:

API_USER

RESOURCE_READER

RESOURCE_WRITER

API:

POST /api/v1/protected-resources

Resultado esperado:

201 Created

Figura 86.

Generación del token con el usuario writer.user mediante Postman

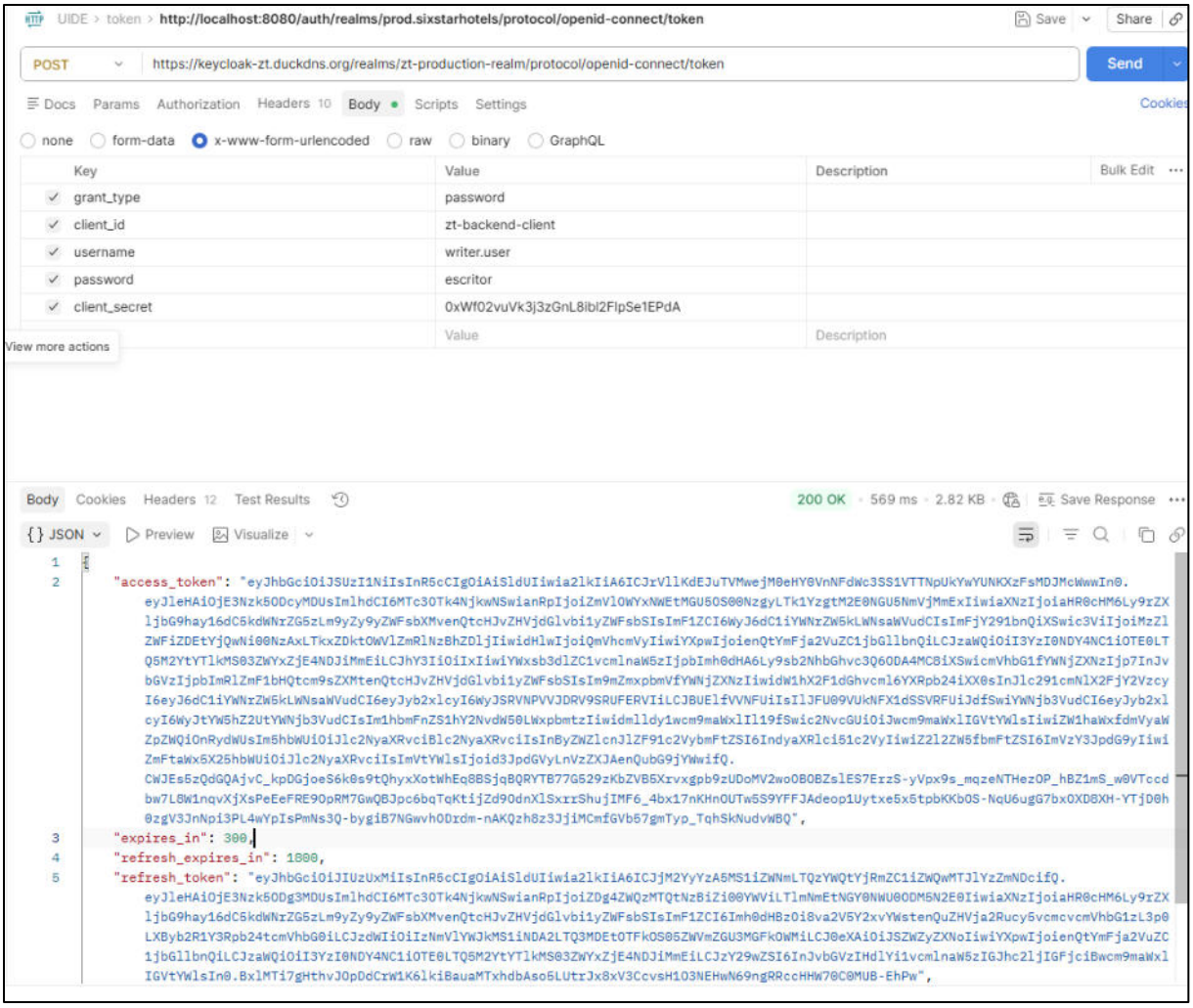
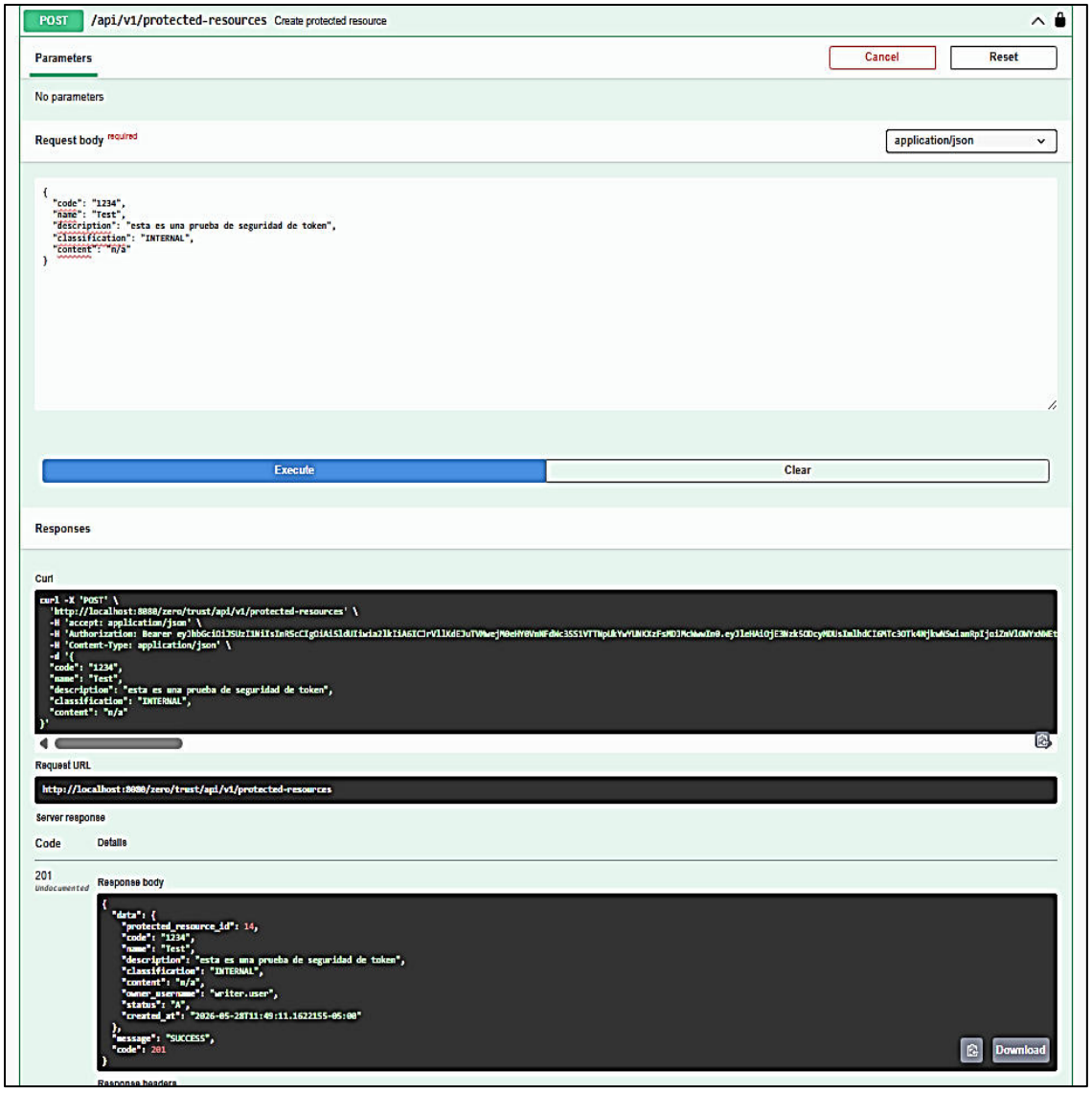


Figura 87.

Creación exitosa de recurso protegido con usuario autorizado



Posteriormente, se realizó la misma prueba con el usuario writer.user, el cual posee el rol RESOURCE_WRITER. En este caso, la solicitud fue aceptada por el backend y se permitió la creación del recurso protegido. Esto confirma que la autorización se aplica de forma diferenciada según los roles incluidos en el token JWT.

Prueba sin token

Prueba sin enviar el header:

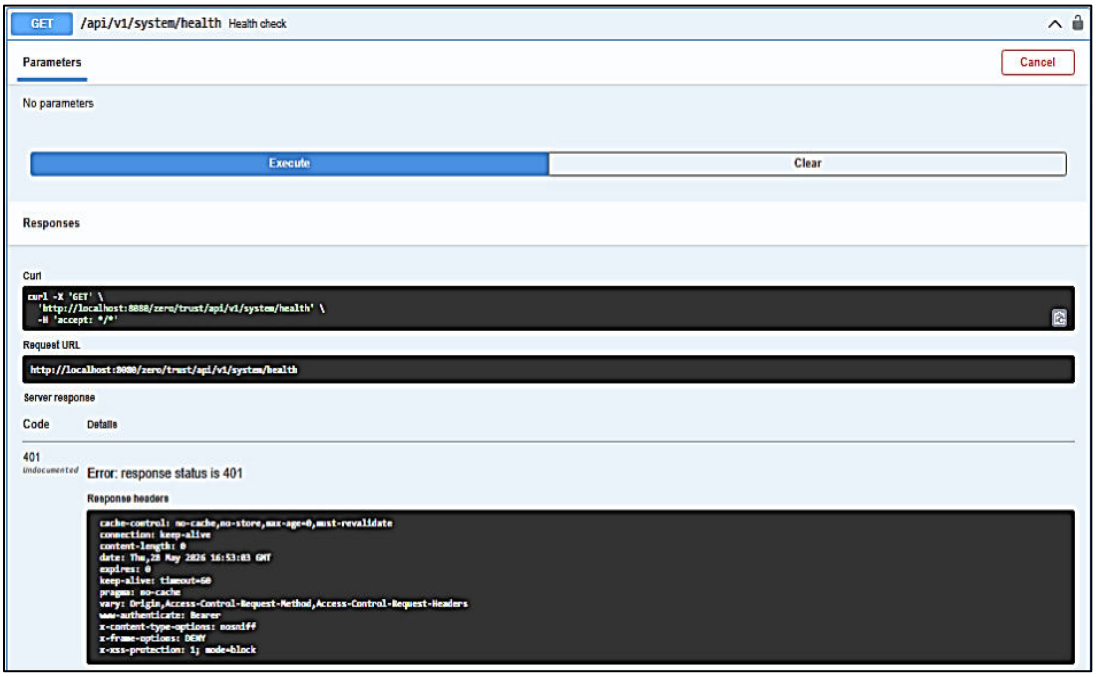
Authorization: Bearer <token>

Resultado esperado:

401 Unauthorized

Figura 88.

Bloqueo de solicitud sin token de acceso



Se ejecutó una solicitud sin token de acceso hacia una API protegida. El backend respondió con HTTP 401 Unauthorized, indicando que la autenticación es obligatoria para consumir los endpoints protegidos. Este resultado confirma que las APIs no permiten acceso anónimo.

Pruebas complementarias de validación de tokens JWT

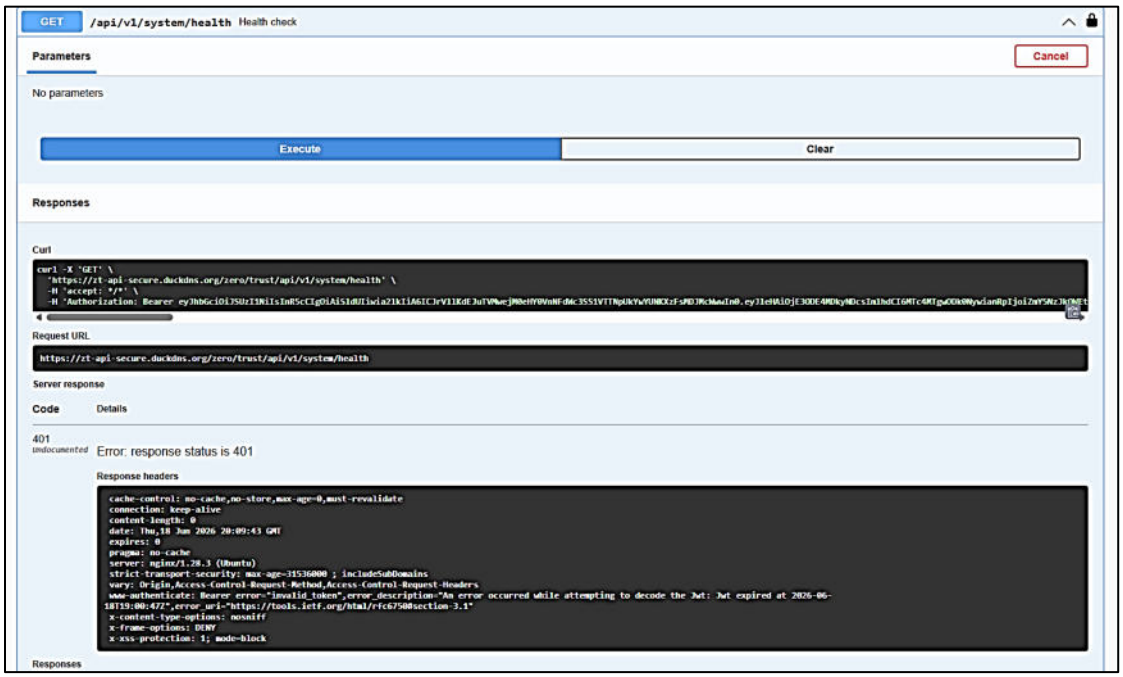
Además de las pruebas realizadas con usuarios y roles, se validó el comportamiento del backend frente a tokens JWT inválidos. Para ello, se ejecutaron solicitudes al endpoint protegido GET /api/v1/system/health utilizando tokens expirados, mal formados y manipulados. Estas pruebas permiten comprobar que el sistema no solo exige un token de acceso, sino que también valida su vigencia, estructura e integridad.

Token JWT expirado

Para esta prueba se utilizó un token emitido por Keycloak cuyo tiempo de expiración ya había finalizado. Luego, se envió una solicitud al endpoint GET /api/v1/system/health incluyendo el token vencido en el encabezado Authorization. Como resultado, el backend respondió con HTTP 401 Unauthorized. En los encabezados de respuesta se observa el mensaje asociado a la expiración del JWT, lo que evidencia que el sistema valida el claim exp antes de permitir el acceso al recurso protegido.

Figura 89.

Rechazo de solicitud con token JWT expirado

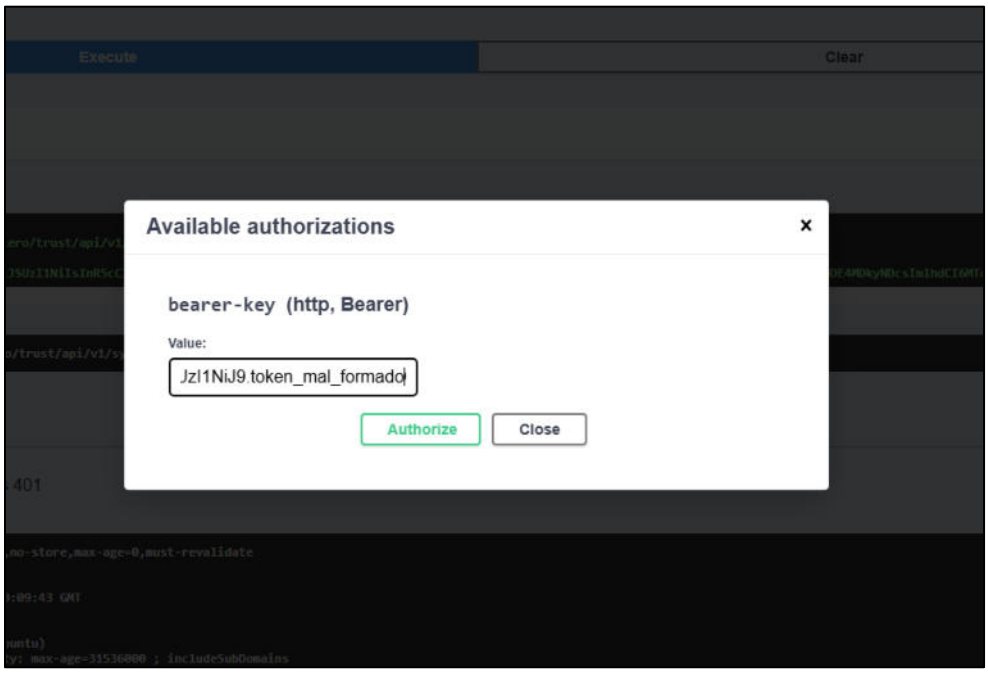


- **Token JWT mal formado**

También se realizó una prueba con un token que no cumplía con la estructura estándar de un JWT. Para ello, se ingresó en Swagger un valor incompleto como Bearer Token, sin mantener correctamente las tres partes requeridas: header, payload y signature.

Figura 90.

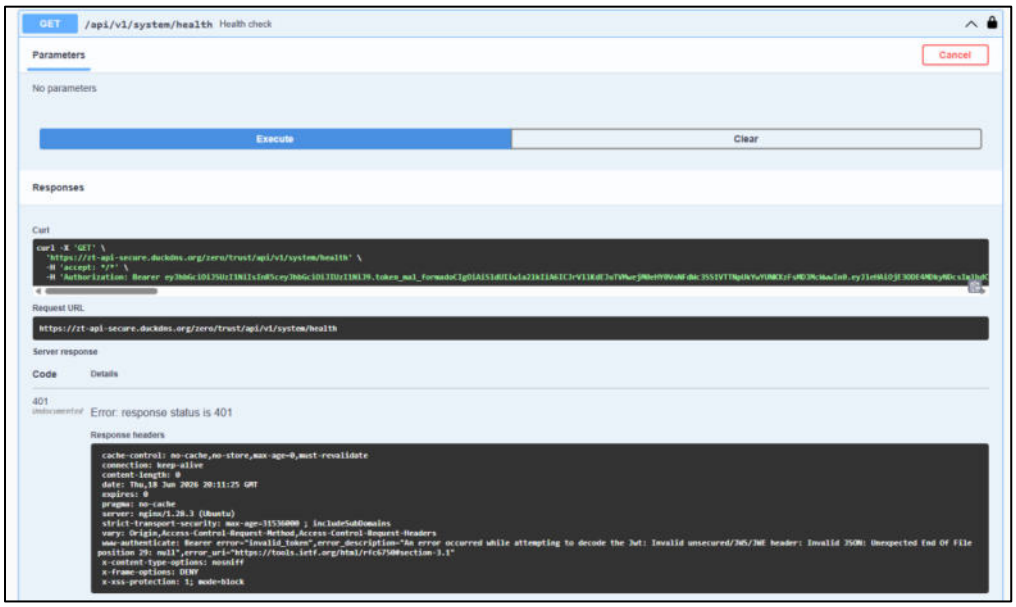
Ingreso de token JWT mal formado en Swagger



Al ejecutar la solicitud hacia el endpoint GET /api/v1/system/health, el backend respondió con HTTP 401 Unauthorized. La respuesta indica que el token no pudo ser decodificado correctamente, debido a que su estructura no corresponde a un JWT válido. Con esto se comprueba que el sistema rechaza tokens incompletos o construidos de forma incorrecta antes de procesar la solicitud.

Figura 91.

Rechazo de solicitud con token JWT mal formado

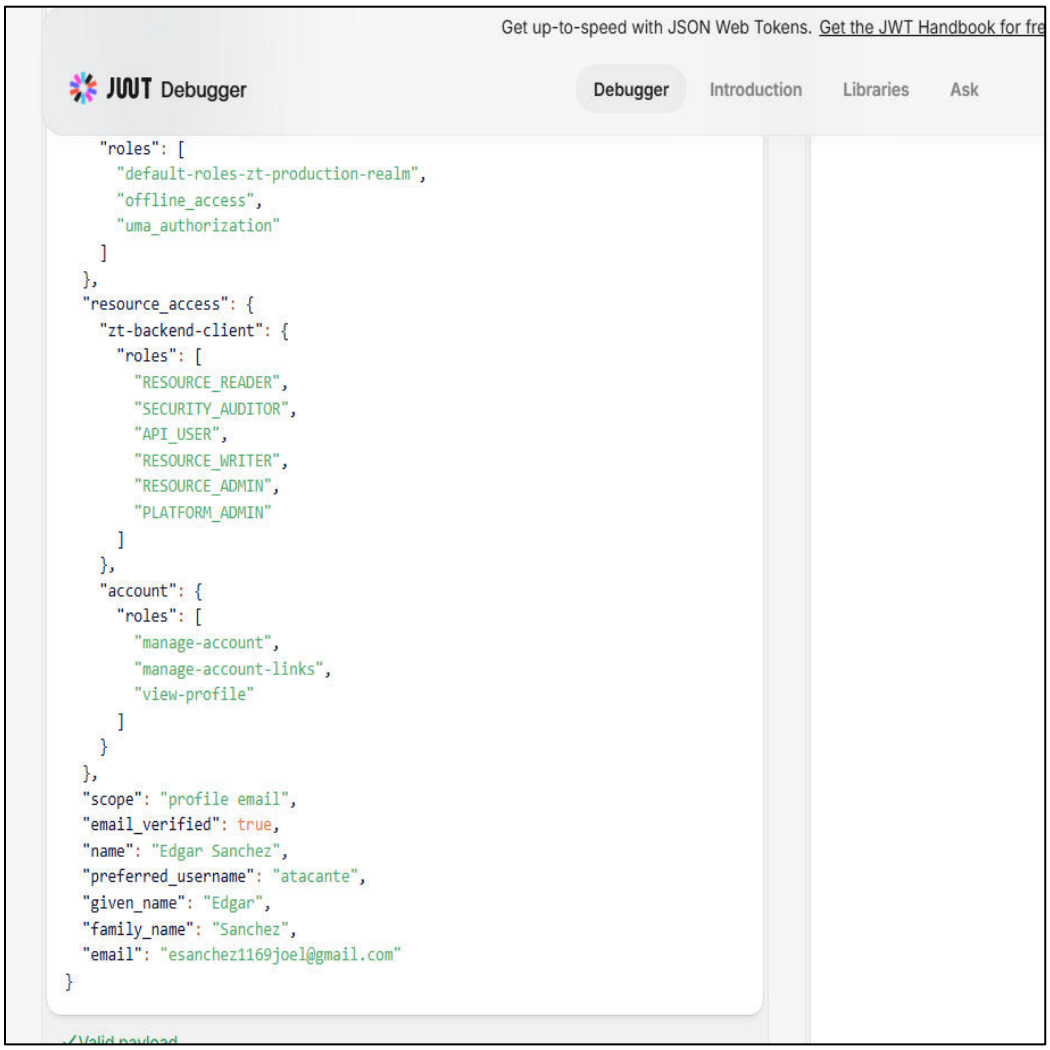


- **Token JWT con firma manipulada**

Para esta prueba se tomó un JWT válido y se modificó manualmente el valor del claim preferred_username, cambiándolo por un usuario diferente. Esta modificación alteró el contenido del token, pero no generó una nueva firma válida desde Keycloak.

Figura 92.

Modificación del claim preferred_username en el token JWT



Posteriormente, el token modificado fue enviado en una solicitud al endpoint GET /api/v1/system/health. El backend respondió con HTTP 401 Unauthorized y en los encabezados de respuesta se muestra un mensaje relacionado con el rechazo de la firma del JWT. Este resultado demuestra que cualquier cambio realizado sobre el contenido del token después de su emisión invalida la firma digital y provoca el bloqueo de la solicitud.

Figura 93.

Rechazo de solicitud con firma JWT inválida

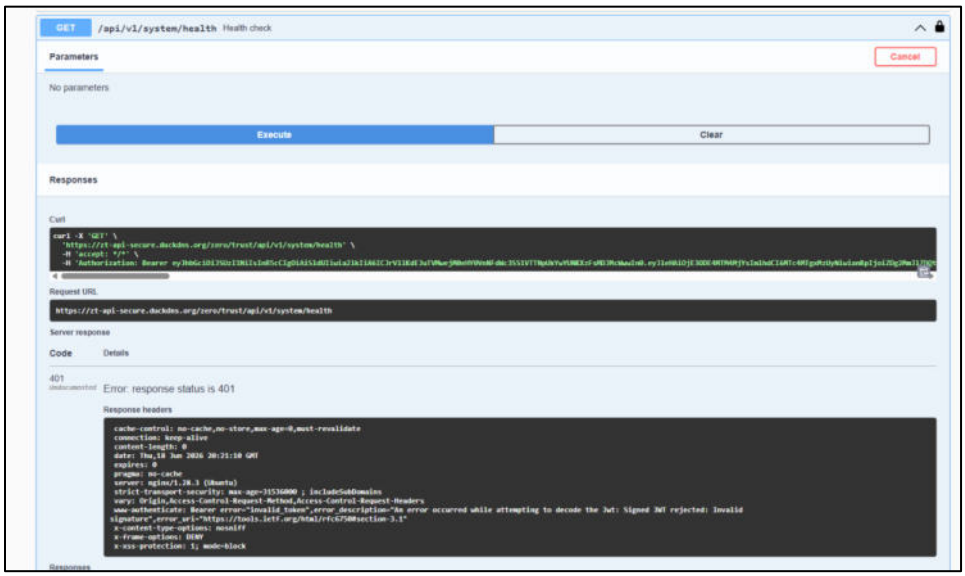


Tabla 16.

Tabla resumen de pruebas de keycloak

N.º	Escenario	Usuario	API probado	Resultado esperado	Resultado obtenido
1	Solicitud sin token	Ninguno	GET /api/v1/system/health	401	401
2	Token válido y rol autorizado	platform.admin	GET /api/v1/system/health	200	200
3	Lectura de recurso protegido	reader.user	GET /api/v1/protected-resources	200	200
4	Creación sin privilegio suficiente	reader.user	POST /api/v1/protected-resources	403	403

5	Creación con rol autorizado	writer.user	POST /api/v1/protected-resources	201	201
6	Consulta de auditoría sin rol auditor	reader.user	GET /api/v1/audit/events	403	403
7	Consulta de auditoría con rol auditor	security.auditor	GET /api/v1/audit/events	200	200
8	Acceso total de administrador	platform.admin	APIs principales	200/201	200/201
9	Token JWT expirado	platform.admin	GET /api/v1/system/health	401	401
10	Token JWT mal formado	No aplica	GET /api/v1/system/health	401	401
11	Token JWT con firma manipulada	platform.admin	GET /api/v1/system/health	401	401

Los resultados obtenidos evidencian que el backend valida correctamente la autenticación y autorización de las solicitudes protegidas. Las pruebas con usuarios y roles permitieron comprobar que cada perfil solo puede ejecutar las operaciones permitidas. Además, las pruebas complementarias con tokens JWT expirados, mal formados y manipulados fueron rechazadas con HTTP 401 Unauthorized, confirmando que el sistema verifica la vigencia, estructura e integridad del token antes de permitir el acceso a los endpoints.

3.1.13. Observación de la implantación de keycloak

La configuración realizada en Keycloak permitió implementar una capa centralizada de identidad y autorización para el laboratorio. Mediante el uso de un realm independiente, un cliente OIDC, roles de cliente y usuarios de prueba, se logró validar el acceso seguro a las APIs del backend. Las pruebas realizadas demostraron que el sistema bloquea solicitudes sin autenticación, rechaza usuarios con privilegios insuficientes y permite operaciones únicamente a usuarios con roles autorizados. Estos resultados evidencian la aplicación práctica de los principios de Zero Trust y mínimo privilegio en la protección de APIs.

Cabe recalcar que, posterior a la implementación de Keycloak, se continuará con la implementación del API Gateway mediante WSO2, conforme al esquema de seguridad planteado para el proyecto. Actualmente, durante la fase inicial de configuración e integración del entorno, se manejan credenciales de prueba para los usuarios, las cuales aún no cuentan con un nivel de seguridad robusto. Una vez implementado el ecosistema completo, estas claves serán reemplazadas por contraseñas más seguras y alineadas con los estándares definidos, con el fin de ejecutar las pruebas de seguridad contempladas en el proyecto, tales como validación de autenticación, autorización, control de acceso, uso de tokens y escenarios de ataque controlados.

3.1.14. Configuración del API Gateway WSO2 integrado con Keycloak

Configuración inicial del API Gateway WSO2

Como primer paso, se realizó la configuración base del API Gateway WSO2 API Manager, definiendo los parámetros necesarios para habilitar el Developer Portal y permitir la integración con aplicaciones externas. En esta etapa se configuró la URL pública del portal de desarrolladores, la cual permite a los consumidores visualizar, suscribirse y probar las APIs publicadas.

Figura 94.

Configuración inicial del API Gateway WSO2

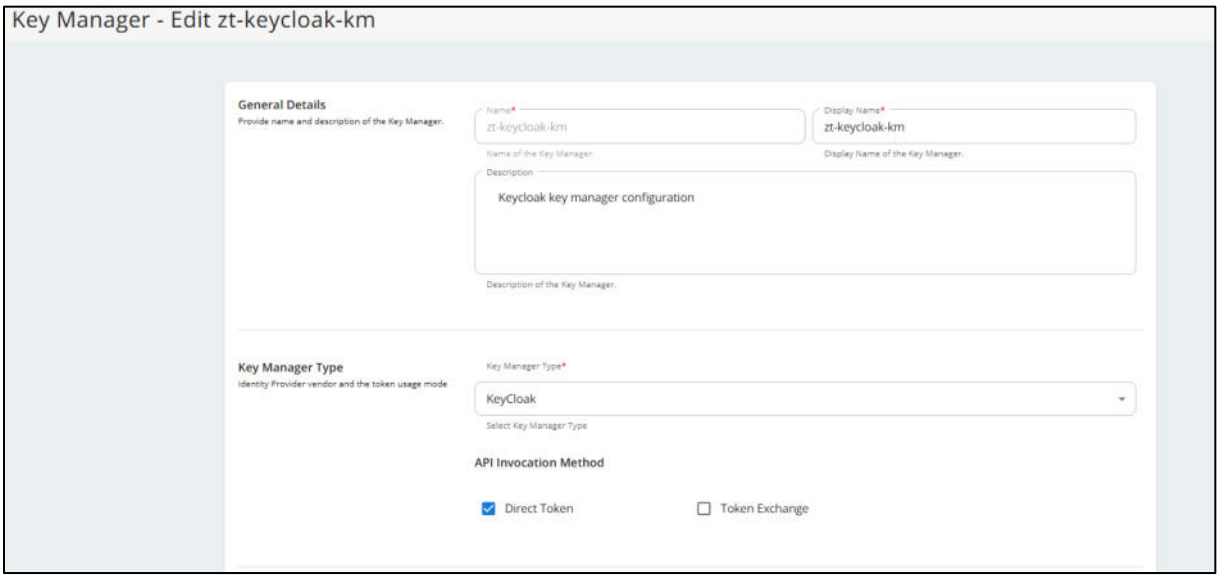
Además, se ajustó la validación de aplicaciones provisionadas, con el objetivo de facilitar la integración entre WSO2 y el proveedor de identidad externo Keycloak. Esta configuración permite que las credenciales gestionadas desde el Key Manager puedan ser utilizadas posteriormente por las aplicaciones consumidoras dentro del Developer Portal.

Configuración del Key Manager externo basado en Keycloak

Posteriormente, se configuró Keycloak como Key Manager externo dentro de WSO2 API Manager. Esta configuración permite que WSO2 delegue la gestión de identidad, autenticación y emisión de tokens al servidor Keycloak, manteniendo al API Gateway como punto central de validación y control de acceso.

Figura 95.

Configuración del Key Manager externo basado en Keycloak



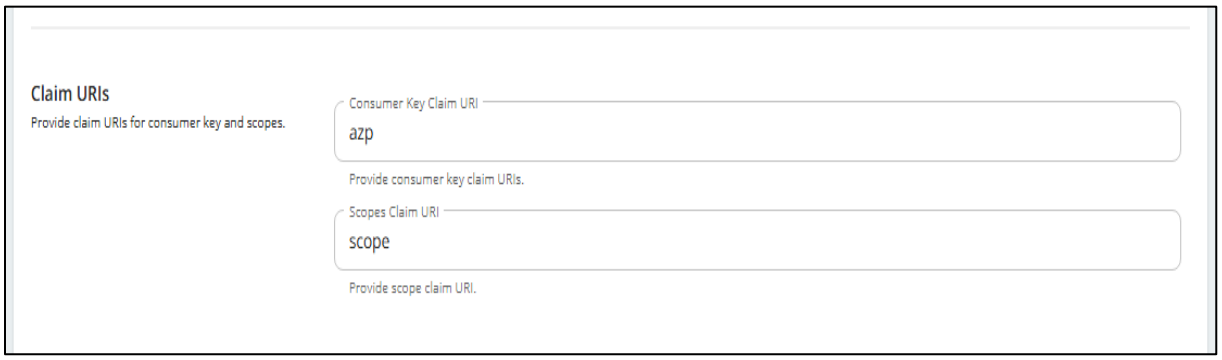
Mediante esta integración, WSO2 puede validar los tokens OAuth2/JWT emitidos por Keycloak antes de permitir el consumo de las APIs protegidas. De esta forma, el acceso a los servicios no depende únicamente del microservicio, sino que pasa primero por una capa de seguridad centralizada en el gateway.

- **Definición de claims y scopes utilizados para la validación del token**

Dentro de la configuración del Key Manager se definieron los claims utilizados para interpretar la información contenida en los tokens JWT. El claim azp permite identificar el cliente autorizado que solicita el acceso, mientras que el claim scope permite reconocer los permisos asociados al token emitido por Keycloak.

Figura 96.

Definición de claims y scopes para la validación del token



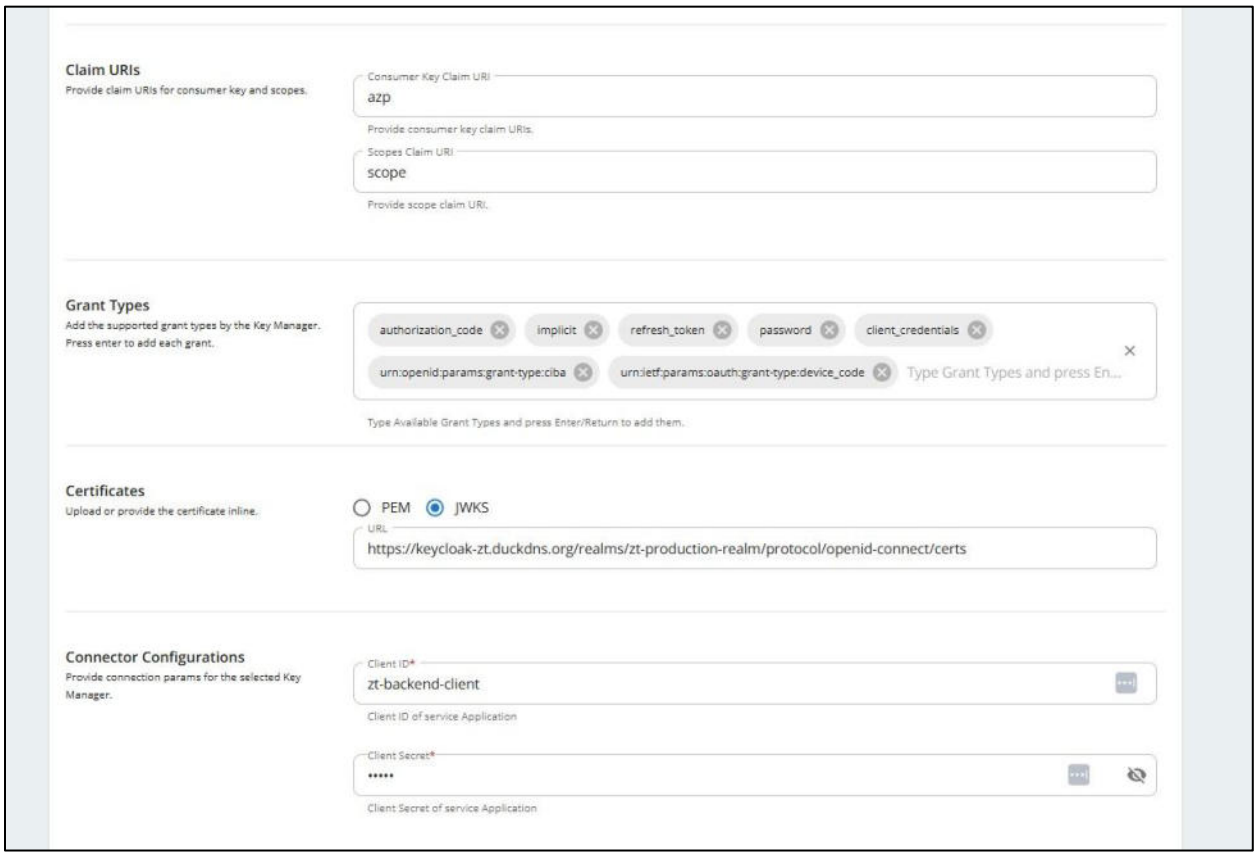
Esta configuración es importante porque permite que WSO2 API Manager analice el contenido del token y determine si la solicitud cuenta con los permisos necesarios para acceder a los recursos publicados. Con ello se refuerza el principio de mínimo privilegio, ya que cada solicitud debe presentar un token válido y con los alcances correspondientes.

3.1.15. Configuración de certificados JWKS para validar tokens JWT firmados

Para validar la firma digital de los tokens JWT emitidos por Keycloak, se configuró en WSO2 la URL de certificados JWKS del realm correspondiente. Esta URL permite que el API Gateway obtenga las claves públicas necesarias para verificar que los tokens no hayan sido alterados y que realmente provengan del proveedor de identidad autorizado.

Figura 97.

Configuración de certificados JWKS para validación de tokens JWT



El uso de JWKS es fundamental dentro del esquema de seguridad, ya que permite validar tokens firmados de forma asimétrica sin exponer claves privadas. De esta manera, WSO2 puede comprobar la autenticidad e integridad del token antes de enrutar la solicitud hacia el microservicio protegido.

En esta etapa se configuraron las credenciales del cliente OAuth2 que permiten la comunicación entre WSO2 API Manager y Keycloak. Para ello se registró el identificador del cliente y su secreto correspondiente, los cuales permiten que WSO2 reconozca el cliente autorizado dentro del realm configurado.

3.1.16. Configuración de credenciales del cliente OAuth2 en WSO2

Figura 98.

Configuración del cliente OAuth2 en WSO2

Estas credenciales son utilizadas para establecer la relación de confianza entre el API Gateway y el proveedor de identidad. Por motivos de seguridad, el valor del secreto del cliente debe mantenerse oculto en la documentación final, ya que representa una credencial sensible del entorno.

3.1.17. Exposición del contrato OpenAPI del microservicio

Una vez configurado el entorno base, se verificó que el microservicio Spring Boot exponga correctamente su contrato OpenAPI mediante una ruta pública. Este contrato contiene la definición de los endpoints, métodos HTTP, parámetros, respuestas y esquemas utilizados por la API.

Figura 99.

Exposición del contrato OpenAPI del microservicio

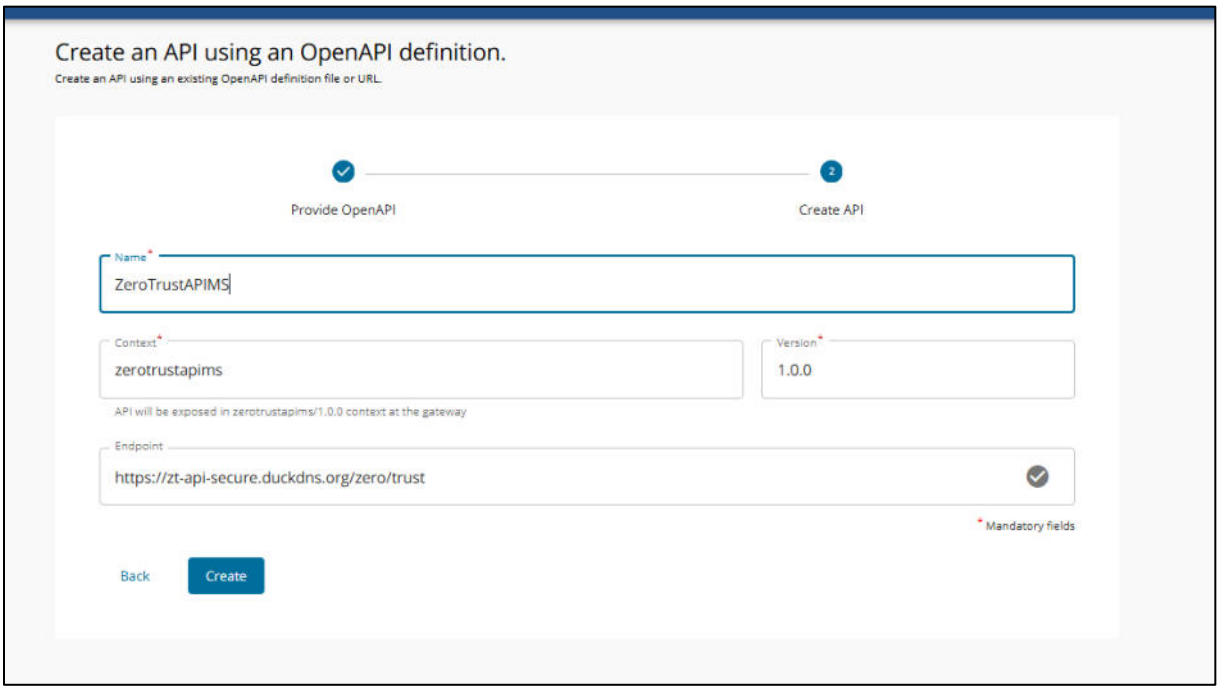
La disponibilidad del archivo OpenAPI permite que WSO2 API Manager importe automáticamente la estructura del servicio, evitando la creación manual de cada recurso. Esta definición sirve como base para publicar la API dentro del gateway y aplicar posteriormente las políticas de seguridad correspondientes.

3.1.18. Creación de la API en WSO2 mediante definición OpenAPI

A partir del contrato OpenAPI expuesto por el microservicio, se procedió con la creación de la API dentro de WSO2 API Manager. En esta configuración se definió el nombre de la API, el contexto de publicación, la versión y el endpoint base hacia el cual el gateway redirigirá las solicitudes válidas.

Figura 100.

Creación de la API en WSO2 mediante definición OpenAPI



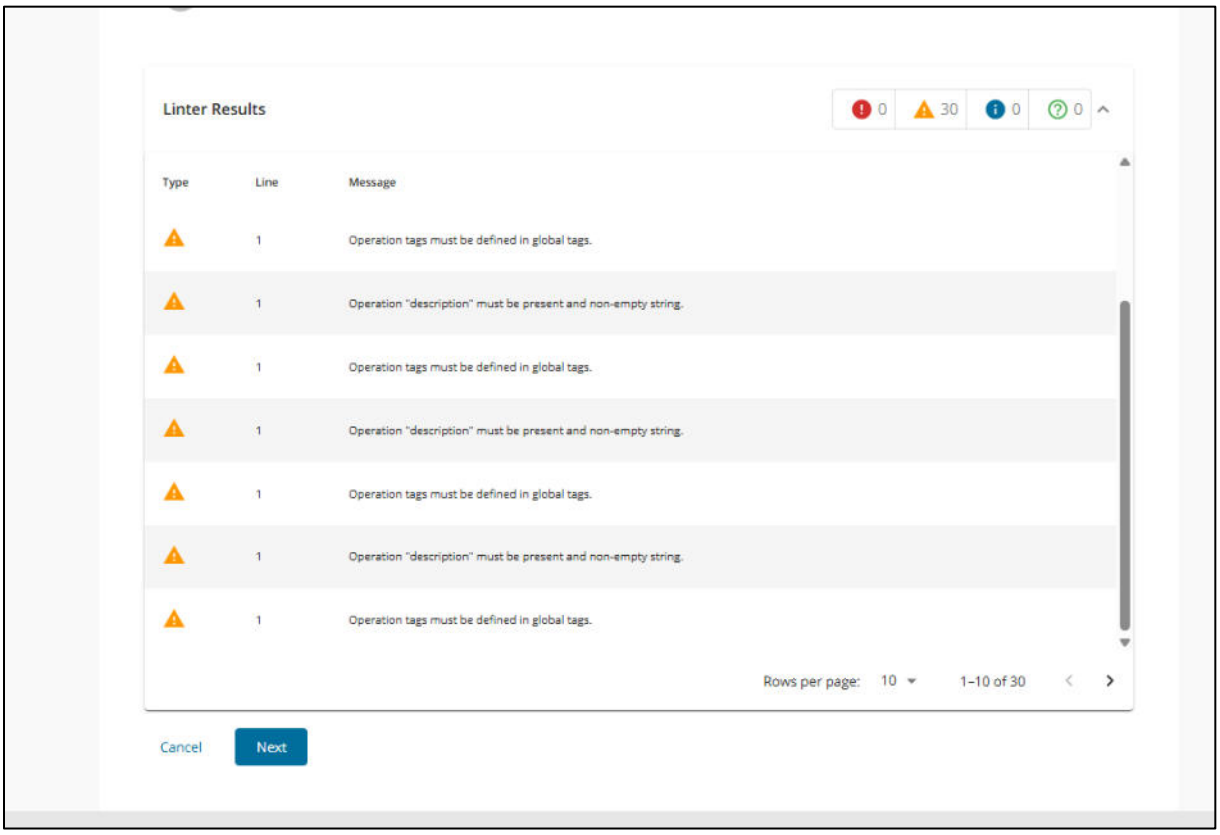
Esta etapa permite que el microservicio quede representado dentro de WSO2 como una API administrada. A partir de este punto, las solicitudes ya no deberían consumirse directamente desde el backend, sino a través del API Gateway, donde se aplican los controles de autenticación, autorización y gestión de tráfico.

3.1.19. Validación de la definición OpenAPI antes de la publicación

Antes de finalizar la creación de la API, WSO2 ejecutó una validación automática de la definición OpenAPI importada. Esta revisión permitió identificar advertencias relacionadas con la documentación del contrato, como la ausencia de descripciones en algunas operaciones o la falta de definición global de ciertas etiquetas.

Figura 101.

Validación de la definición OpenAPI antes de la publicación



Estas observaciones no impidieron la creación de la API, pero permiten evidenciar que el contrato fue revisado antes de su publicación. En un entorno productivo, estas advertencias deben corregirse para mejorar la calidad documental de la API y facilitar su mantenimiento por parte de desarrolladores y consumidores.

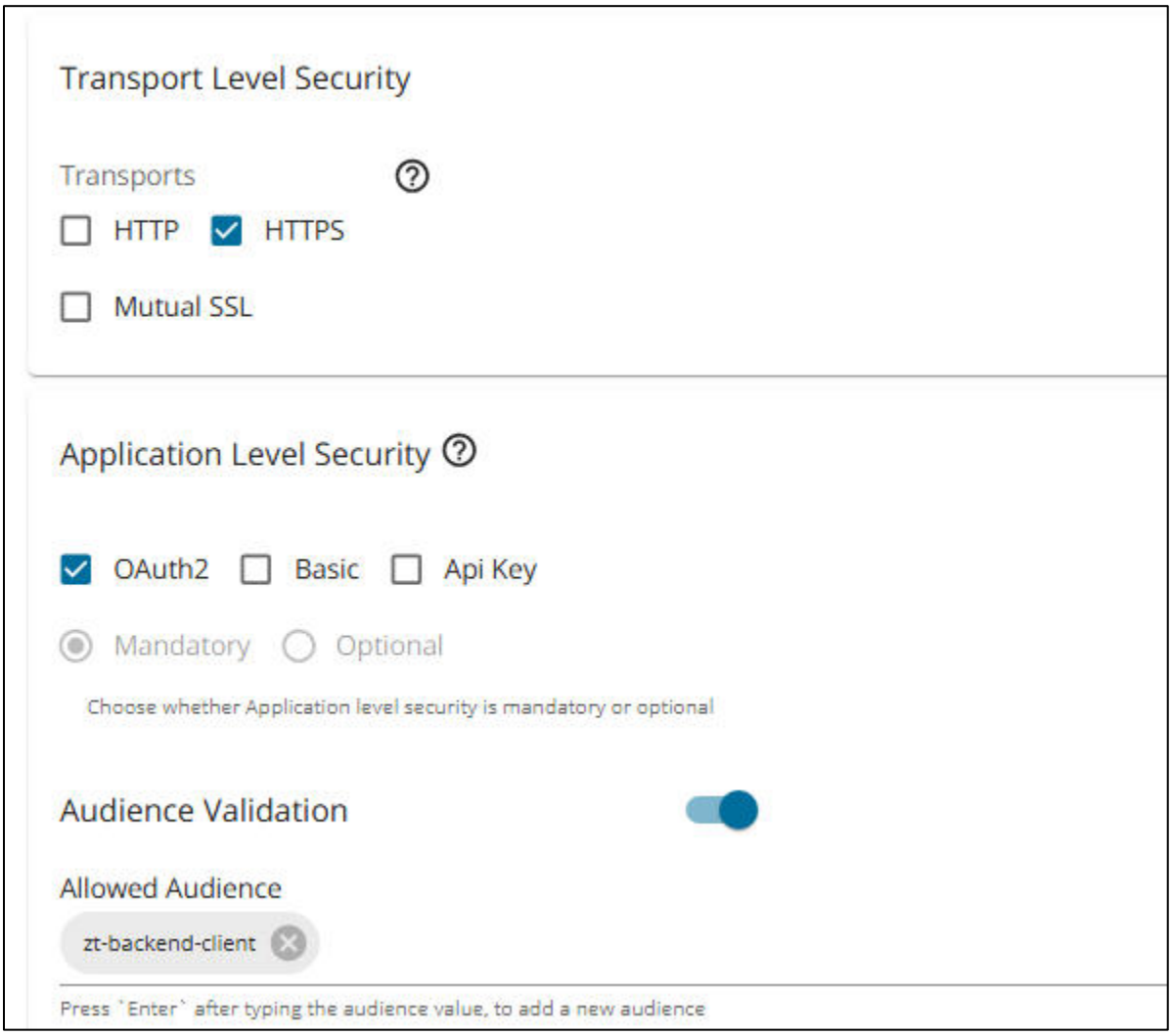
3.1.20. Activación de seguridad OAuth2 en la API publicada

Una vez creada la API, se habilitó la seguridad a nivel de aplicación mediante OAuth2, dejando deshabilitados los mecanismos Basic y API Key. Esta configuración obliga a que las

solicitudes enviadas hacia la API incluyan un token de acceso válido en la cabecera Authorization.

Figura 102.

Activación de seguridad OAuth2 en la API publicada



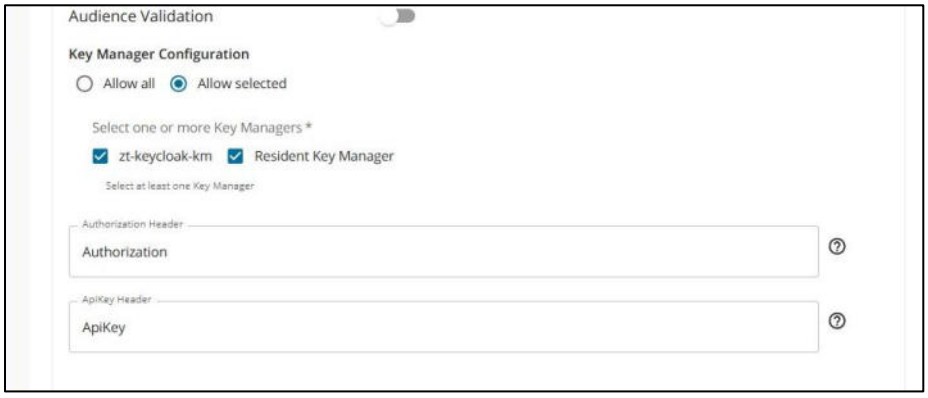
Adicionalmente, se configuró el transporte seguro mediante HTTPS y se activó la validación de audiencia del token, estableciendo como audiencia permitida el cliente zt-backend-client. Con esta configuración, WSO2 API Manager verifica que el token presentado corresponda al consumidor autorizado antes de permitir el acceso al recurso publicado.

3.1.21. Selección del Key Manager Keycloak como validador de tokens

En la configuración de seguridad de la API se seleccionó el Key Manager externo zt-keycloak-km, basado en Keycloak, como responsable de la validación de los tokens OAuth2/JWT. Al encontrarse deshabilitado el Resident Key Manager, WSO2 API Manager centraliza la validación únicamente sobre el proveedor de identidad externo configurado.

Figura 103.

Selección del Key Manager Keycloak como validador de tokens



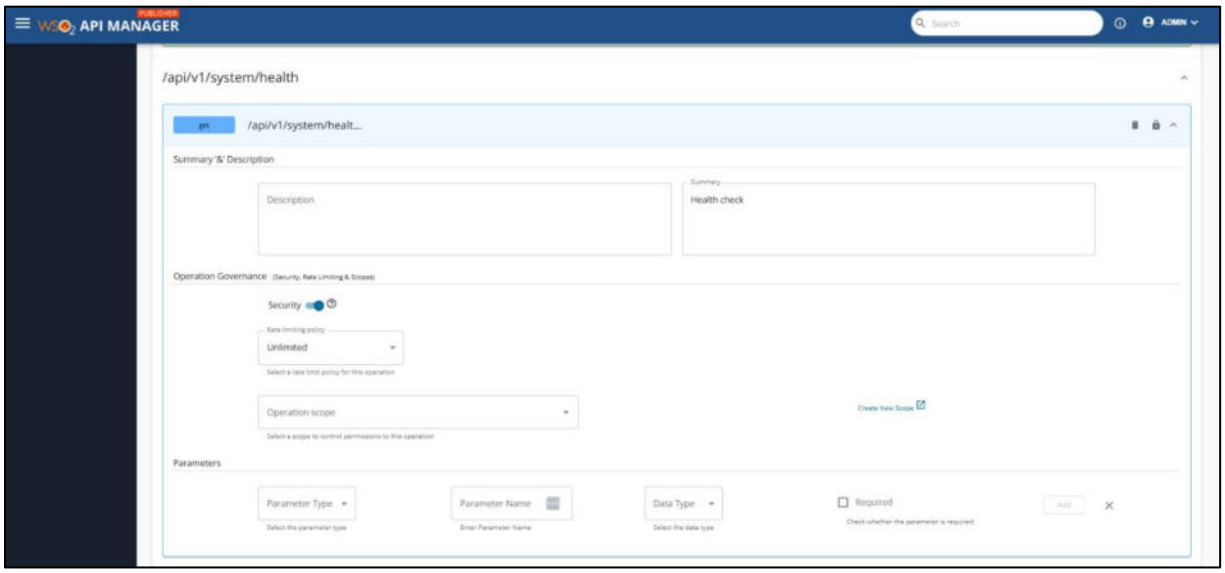
De esta forma, las solicitudes que no presenten un token válido, que no hayan sido emitidas por Keycloak, que se encuentren expiradas o que no cumplan con la audiencia permitida, serán rechazadas antes de llegar al microservicio protegido.

3.1.22. Configuración de seguridad y límites de consumo por recurso

Después de activar la seguridad general de la API, se revisó la configuración individual de los recursos publicados. En esta sección se puede habilitar la seguridad por operación, asignar políticas de limitación de consumo y definir scopes específicos según el nivel de protección requerido por cada endpoint.

Figura 104.

Configuración de seguridad por recurso en la API publicada.



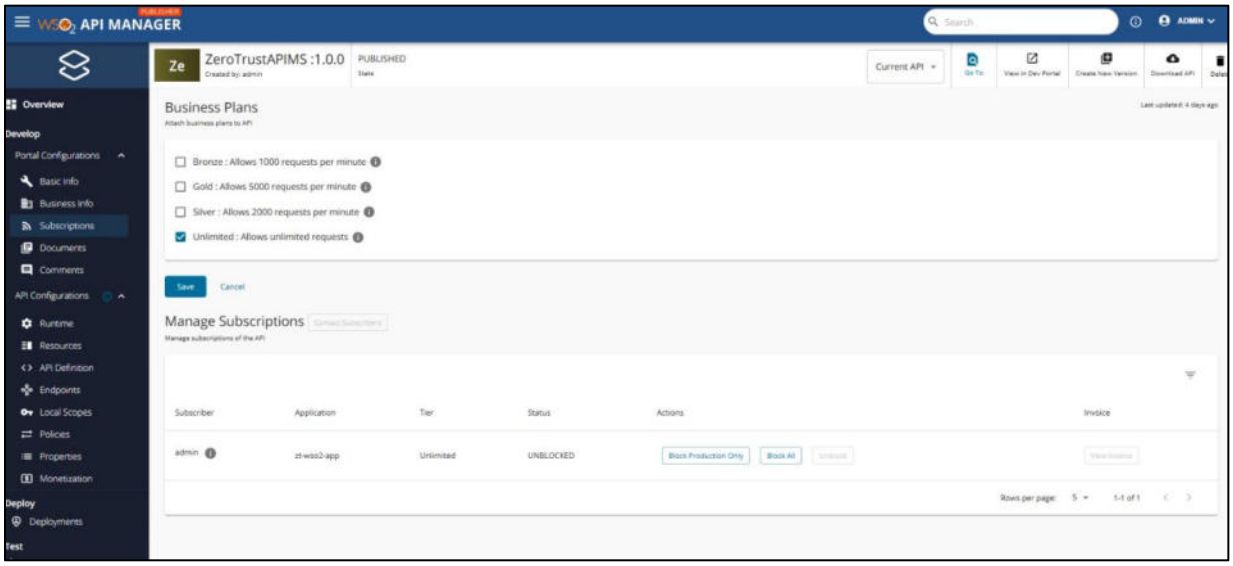
En el caso del endpoint de verificación de estado, se mantuvo una política de consumo amplia debido a su finalidad de monitoreo; sin embargo, WSO2 permite aplicar límites más restrictivos en operaciones críticas como creación, actualización o eliminación de recursos.

Asignación de planes de consumo para controlar solicitudes

Para controlar el consumo de la API, se configuraron planes de suscripción dentro de WSO2 API Manager. Estos planes permiten establecer límites de llamadas por minuto según el tipo de consumidor o nivel de servicio asignado.

Figura 105.

Asignación de planes de consumo para controlar solicitudes



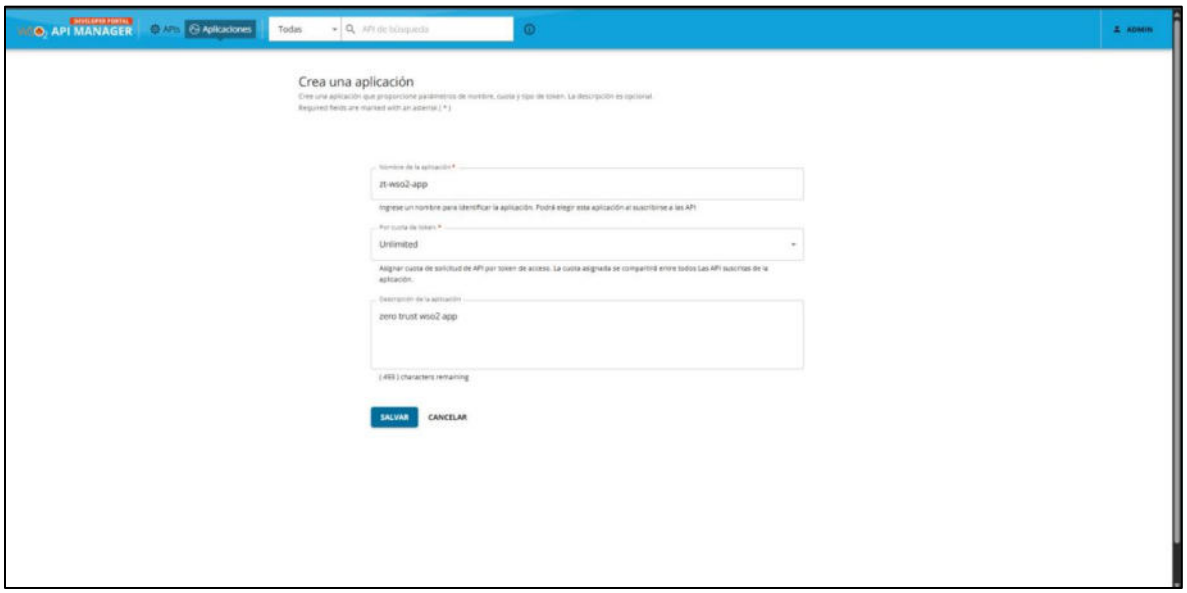
La existencia de planes como Bronze, Silver, Gold y Unlimited permite administrar el tráfico hacia la API y prevenir escenarios de abuso o consumo excesivo. Este mecanismo también sirve como base para realizar pruebas de rate limiting, verificando si el gateway responde adecuadamente ante múltiples solicitudes en un periodo corto de tiempo.

Creación de la aplicación consumidora en el Developer Portal

En el Developer Portal se creó una aplicación consumidora denominada zt-wso2-app. Esta aplicación representa al cliente autorizado que consumirá la API protegida a través de WSO2 API Manager.

Figura 106.

Creación de la aplicación consumidora en el Developer Portal



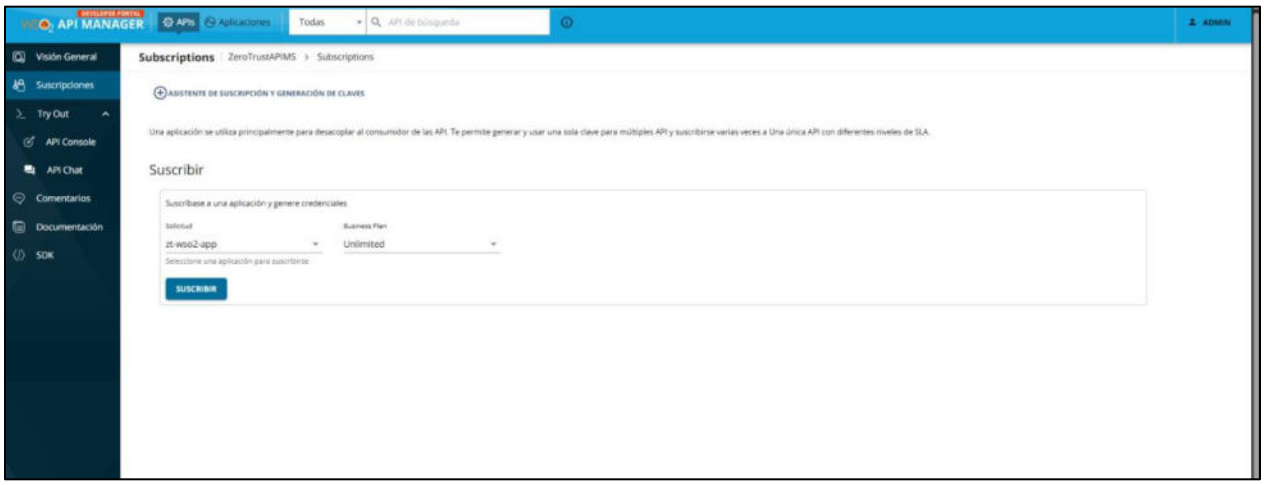
La creación de una aplicación permite desacoplar la API del consumidor final, ya que las suscripciones, credenciales y planes de consumo se gestionan a nivel de aplicación. Esto facilita el control de acceso, la trazabilidad de solicitudes y la administración de credenciales asociadas al consumo de servicios.

Suscripción de la aplicación consumidora a la API protegida

Luego de crear la aplicación consumidora, se realizó la suscripción de dicha aplicación a la API publicada. En esta etapa se seleccionó la aplicación zt-wso2-app y se asignó el plan de consumo correspondiente.

Figura 107.

Suscripción de la aplicación consumidora a la API protegida



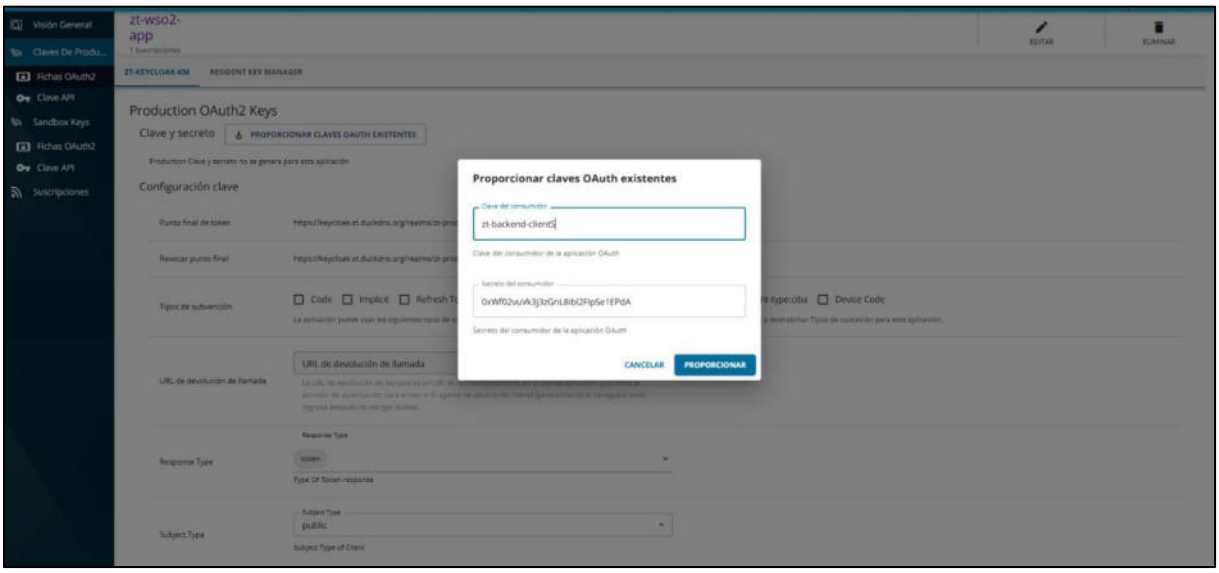
La suscripción permite que la aplicación quede autorizada para consumir la API a través del gateway. Sin esta relación, el consumidor no podría utilizar las credenciales asociadas para invocar los recursos protegidos. Esta configuración también permite monitorear y administrar el consumo de la API por cada aplicación registrada.

3.1.23. Vinculación de credenciales OAuth2 de Keycloak con WSO2

Después de suscribir la aplicación consumidora a la API protegida, se realizó la vinculación de las credenciales OAuth2 existentes de Keycloak con la aplicación registrada en WSO2 API Manager. Para ello, se proporcionó el identificador del cliente zt-backend-client y su secreto correspondiente, permitiendo que WSO2 relacione la aplicación consumidora con el cliente previamente configurado en el proveedor de identidad.

Figura 108.

Vinculación de credenciales OAuth2 de Keycloak con WSO2



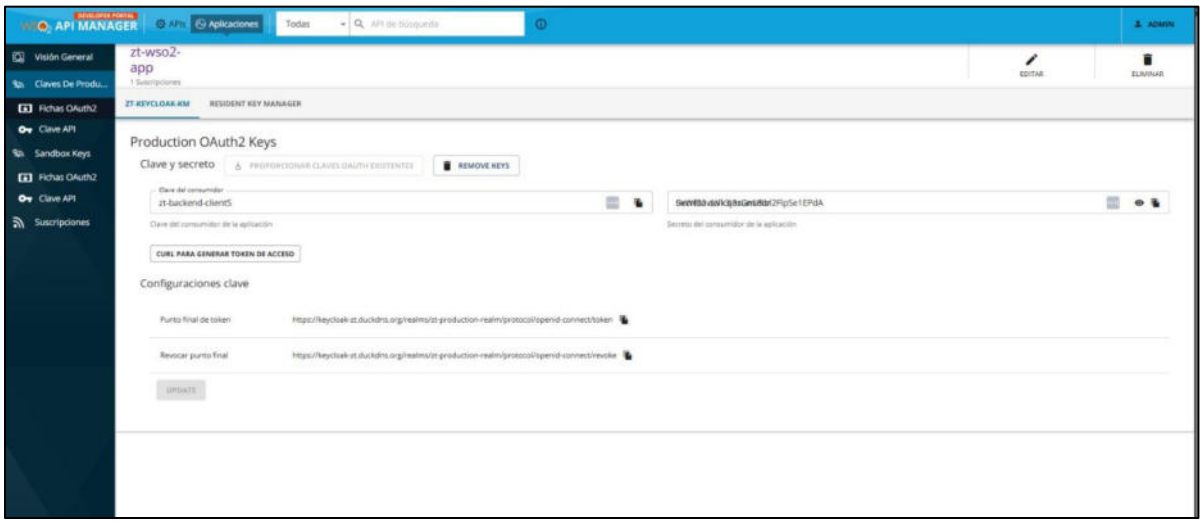
Esta vinculación permite que la aplicación zt-wso2-app trabaje con el Key Manager externo basado en Keycloak y utilice tokens generados por este proveedor de identidad para consumir la API protegida. Con ello, WSO2 puede asociar el token presentado por el consumidor con una aplicación suscrita y autorizada dentro del API Gateway.

3.1.24. Configuración de claves de producción para el consumo de la API

Una vez proporcionadas las credenciales OAuth2 existentes, WSO2 API Manager registró las claves de producción asociadas al Key Manager externo. Estas claves permiten que la aplicación consumidora utilice credenciales válidas para obtener o presentar tokens de acceso emitidos por Keycloak durante el consumo de la API.

Figura 109.

Configuración de claves de producción para el consumo de la API



En esta sección también se visualizan los endpoints de token y revocación correspondientes al realm configurado en Keycloak. Estos endpoints forman parte del flujo OAuth2 y permiten gestionar la emisión y revocación de tokens utilizados para consumir la API protegida desde WSO2. De esta manera, se completa la integración entre la aplicación consumidora, el API Gateway y el proveedor de identidad, manteniendo un flujo centralizado de autenticación y autorización.

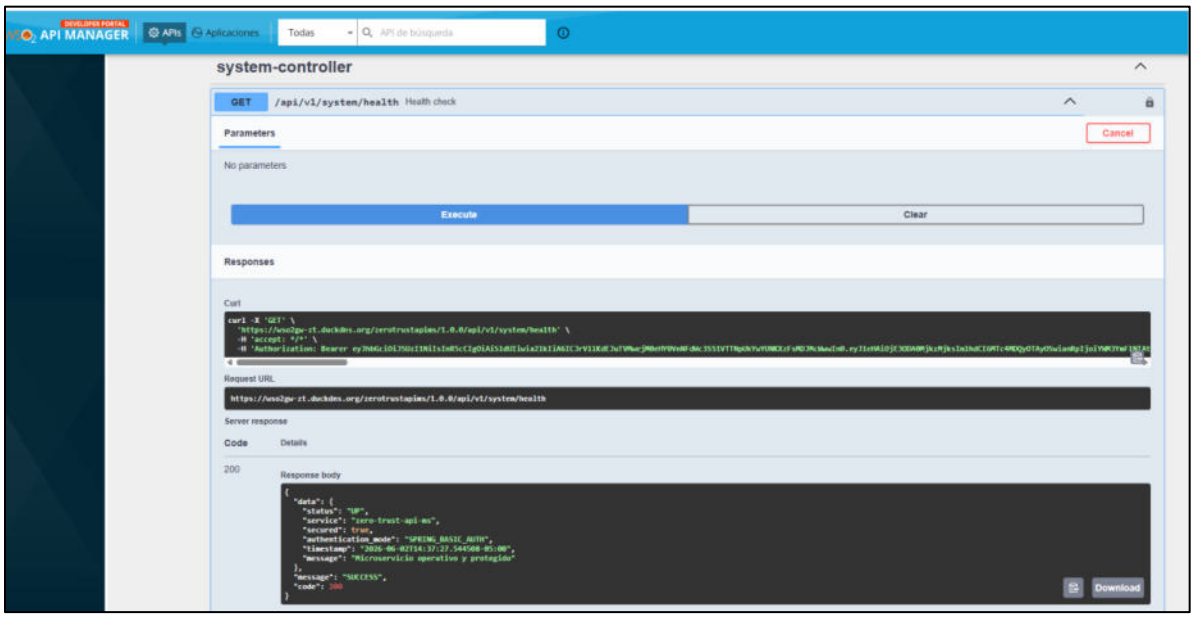
3.1.25. Pruebas controladas del API Gateway WSO2

Prueba de consumo exitoso de la API protegida mediante token Bearer

Una vez configurado el API Gateway, se procedió a verificar su funcionamiento ejecutando una solicitud de consumo sobre el endpoint /api/v1/system/health desde la consola integrada del Developer Portal. Para esta prueba se utilizó un token Bearer emitido por Keycloak, asociado al cliente zt-backend-client y a la aplicación consumidora zt-wso2-app.

Figura 110.

Prueba de consumo exitoso de la API protegida mediante token Bearer



La solicitud se envió a través de la URL publicada por el API Gateway, bajo el esquema de seguridad OAuth2 configurado previamente. WSO2 API Manager validó el token, confirmó que la aplicación consumidora contaba con suscripción activa y verificó que la audiencia del token coincidiera con la esperada, antes de reenviar la solicitud al microservicio protegido.

Que la solicitud haya pasado por WSO2 API Manager se confirma con la propia URL utilizada: la petición no se dirige al puerto del microservicio sino a la ruta publicada por el API Gateway. Adicionalmente, la solicitud incluye el encabezado Authorization: Bearer, validado por WSO2 antes de reenviar la petición al backend. Este comportamiento puede verificarse también consultando el endpoint /api/v1/gateway/context y los registros de auditoría, donde queda identificado si la solicitud fue canalizada por el gateway antes de llegar al microservicio.

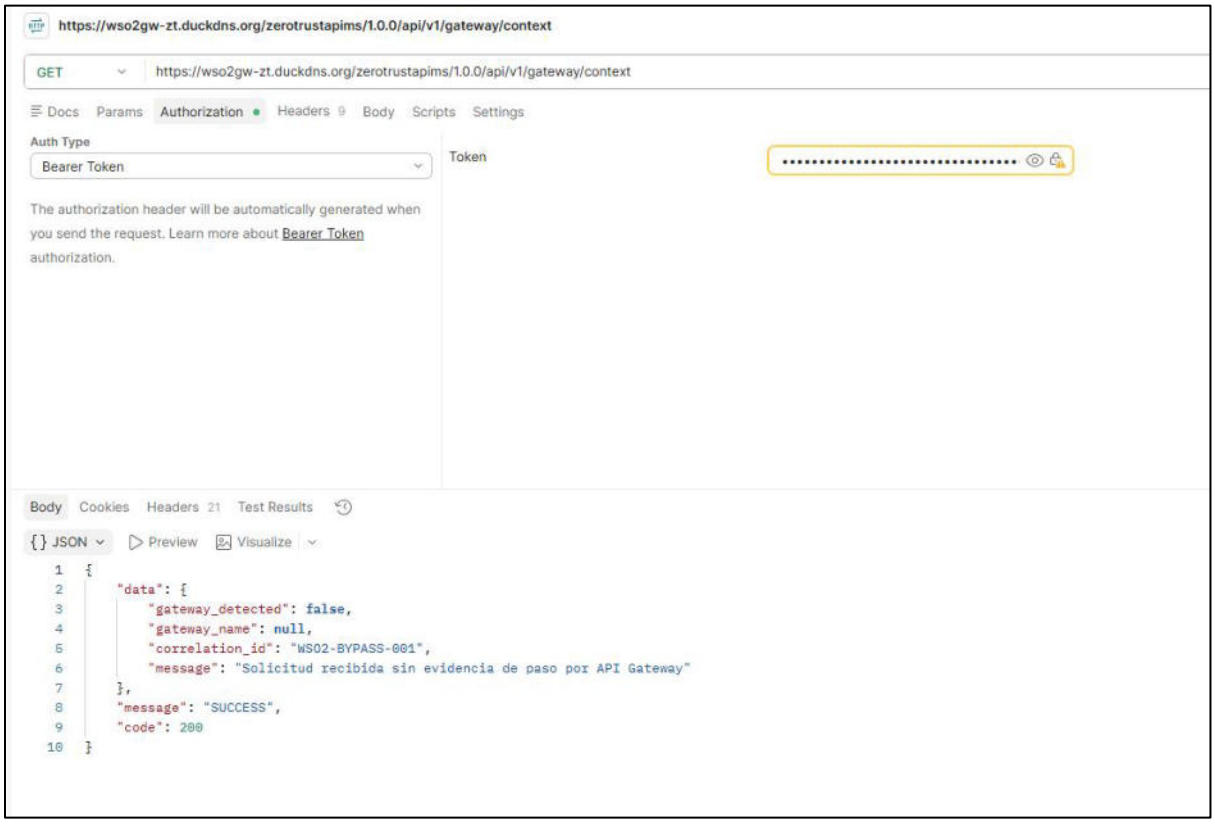
La respuesta obtenida fue HTTP 200 OK, lo que indica que el flujo de autenticación, autorización y validación del token operó correctamente en esta primera prueba.

Acceso directo sin pasar por el API Gateway

Con el fin de comprobar si el microservicio podía consumirse directamente sin pasar por el API Gateway, se realizó una prueba comparativa sobre el endpoint /api/v1/gateway/context.

Figura 111.

Consumo del endpoint de contexto a través del API Gateway WSO2

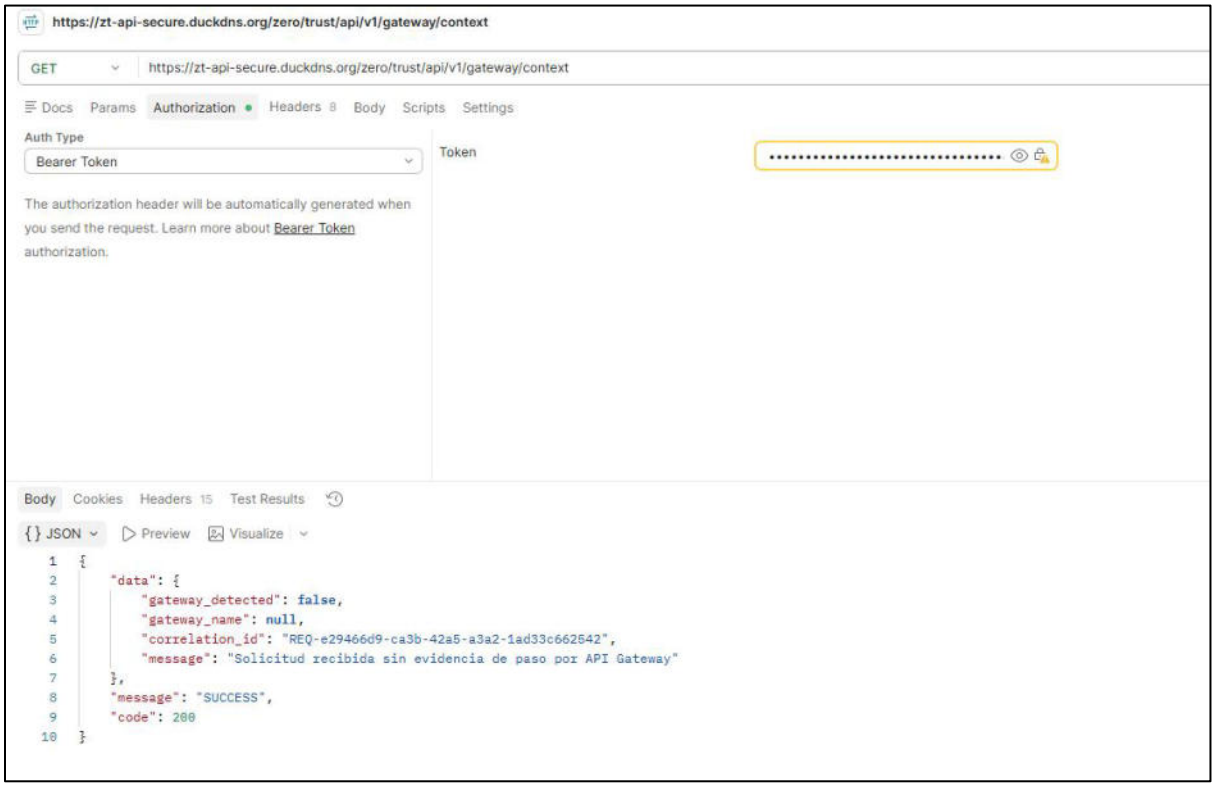


Se ejecutó la misma solicitud de forma directa contra la URL pública del microservicio. En este segundo caso, la respuesta también fue HTTP 200 OK y mantuvo el valor "gateway_detected": false, confirmando que el backend pudo ser consumido directamente con un

token válido. Este resultado constituye un hallazgo relevante del laboratorio: aunque el microservicio mantiene su propia validación de autenticación, el acceso directo al backend no está completamente restringido a nivel de red o proxy. Como mejora, se recomienda limitar el acceso público al microservicio y permitir únicamente tráfico proveniente del API Gateway WSO2, mediante reglas de Security Group, configuración de Nginx o validación obligatoria de encabezados internos generados por el gateway.

Figura 112.

Consumo directo del microservicio sin pasar por el API Gateway



Limitación de consumo por exceso de solicitudes

Para validar el mecanismo de limitación de consumo del API Gateway, se creó en WSO2 una política de suscripción denominada ZT_5_REQ_MIN. Esta política fue configurada con un

límite de cinco solicitudes por minuto, utilizando el criterio de conteo de solicitudes. Posteriormente, la política fue agregada como plan de consumo de la API ZeroTrustAPIMS:1.0.0 y la aplicación consumidora zt-wso2-app fue suscrita a dicho plan.

Figura 113.

Creación de política de limitación de consumo en WSO2

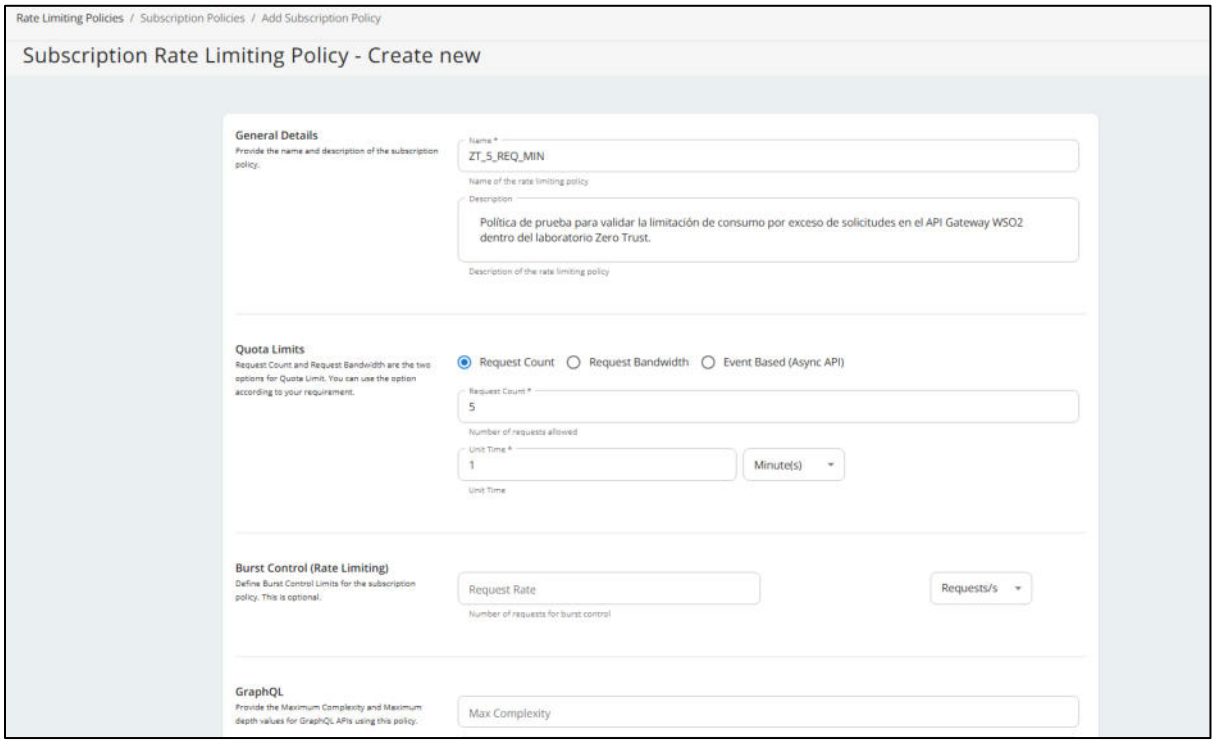


Figura 114.

Asignación de la política ZT_5_REQ_MIN como plan de consumo de la API

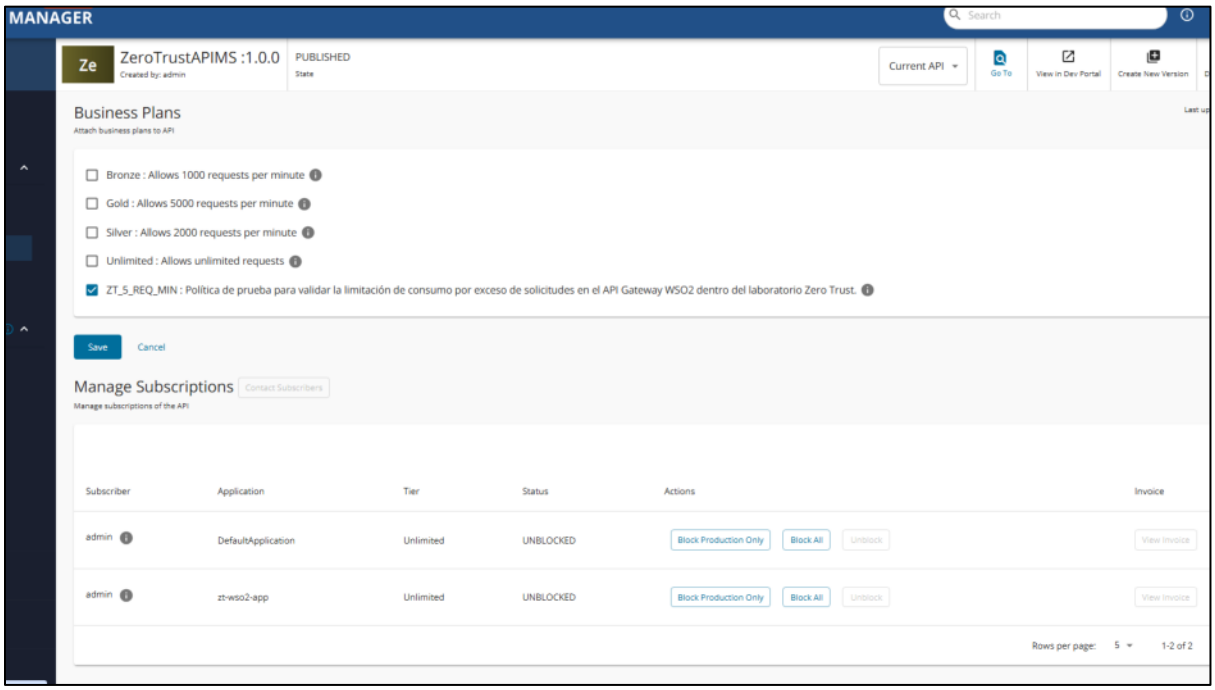


Figura 115.

Suscripción de la aplicación consumidora al plan limitado ZT_5_REQ_MIN

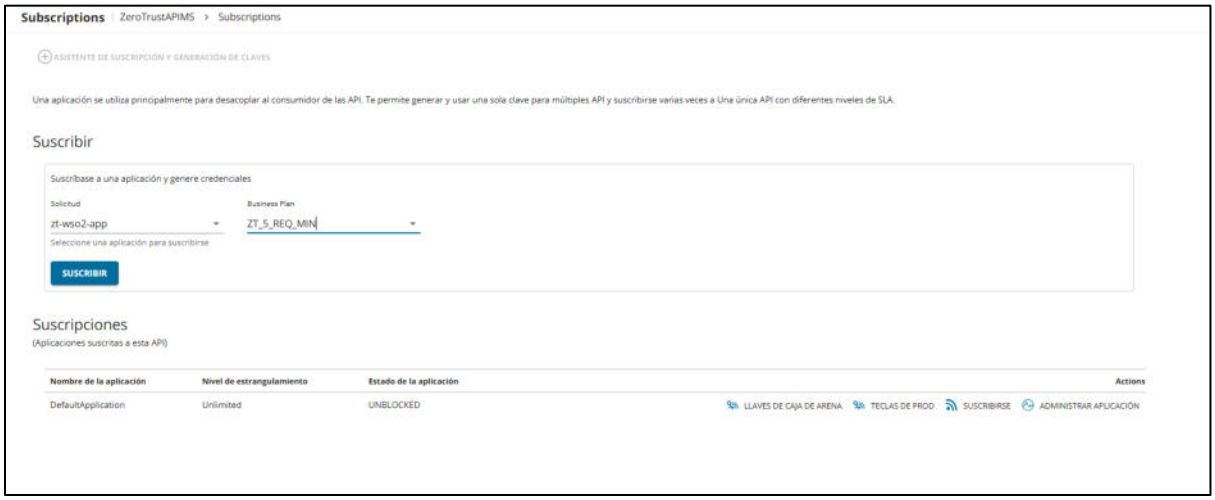


Figura 116.

Aplicación zt-wso2-app suscrita a la API con política ZT_5_REQ_MIN

Como validación inicial, se ejecutó el endpoint GET /api/v1/system/health desde la consola Try Out, obteniendo una respuesta HTTP 200 OK. Esto confirmó que la API podía ser consumida correctamente antes de superar la cuota configurada. Luego, mediante Postman Runner, se ejecutaron veinte solicitudes consecutivas con un retardo de 0 milisegundos. Durante la ejecución se observaron respuestas HTTP 429 en varias iteraciones, evidenciando que WSO2 bloqueó las solicitudes una vez superado el límite permitido por la política de consumo.

Figura 117.

Consumo exitoso inicial de la API antes de superar el límite configurado

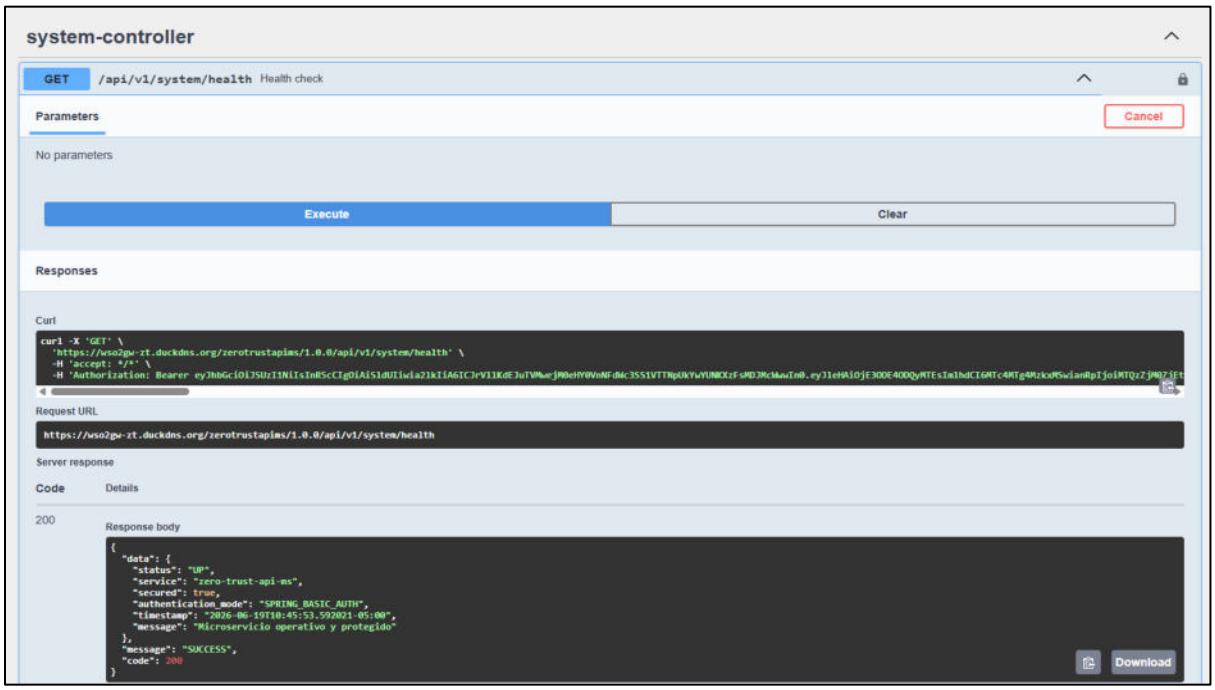
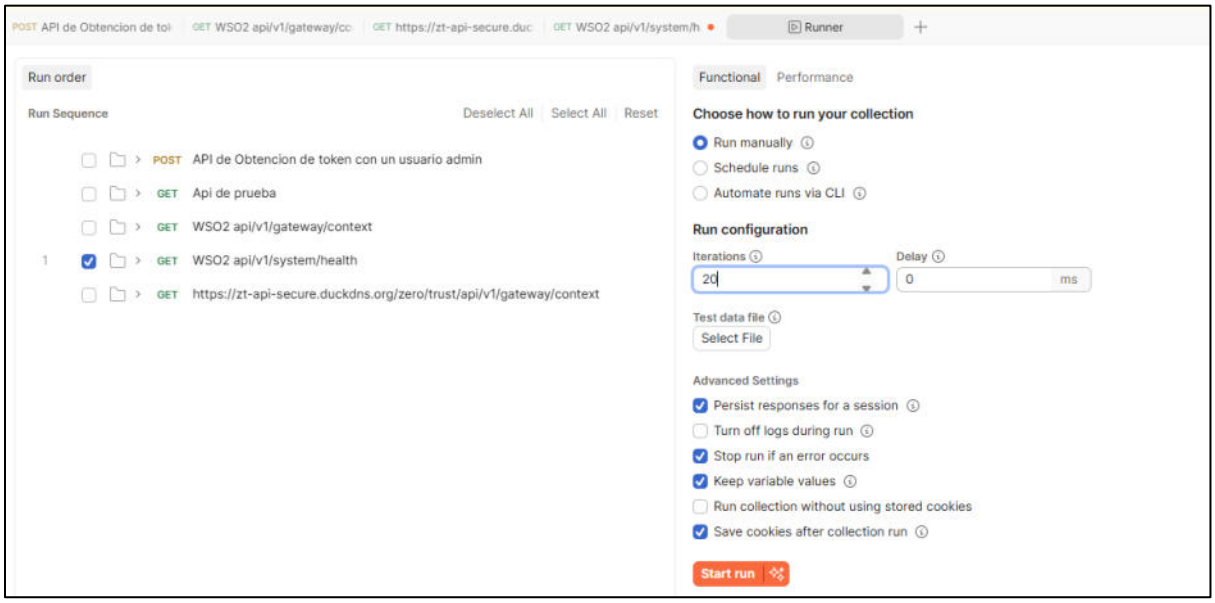


Figura 118.

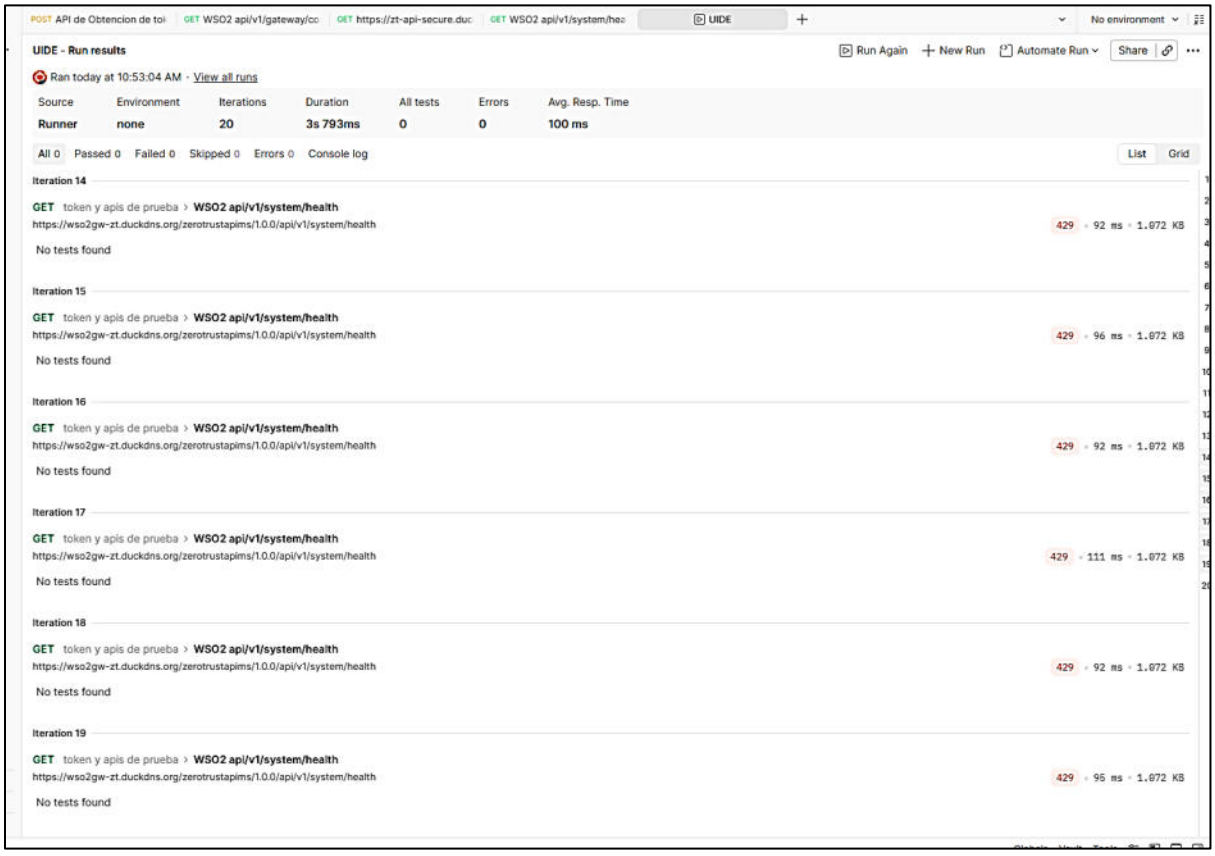
Configuración del Postman Runner para ejecutar solicitudes repetitivas



El resultado confirma que la política de rate limiting configurada en WSO2 funciona como se esperaba: permite el consumo dentro del límite establecido y rechaza las solicitudes que lo exceden mediante el código HTTP 429, lo cual reduce el riesgo de abuso del endpoint ante solicitudes repetitivas.

Figura 119.

Bloqueo de solicitudes por exceder el límite de consumo configurado en WSO2



Validación de suscripción de aplicación consumidora

Para comprobar que el consumo de la API depende de una suscripción válida en WSO2, se ejecutó una prueba sobre la aplicación consumidora zt-wso2-app. Inicialmente, la aplicación se encontraba suscrita a la API ZeroTrustAPIMS con estado UNBLOCKED, lo cual permitió

consumir correctamente el endpoint GET /api/v1/system/health y obtener una respuesta HTTP 200 OK.

Figura 120.

Aplicación zt-wso2-app suscrita a la API con plan Unlimited

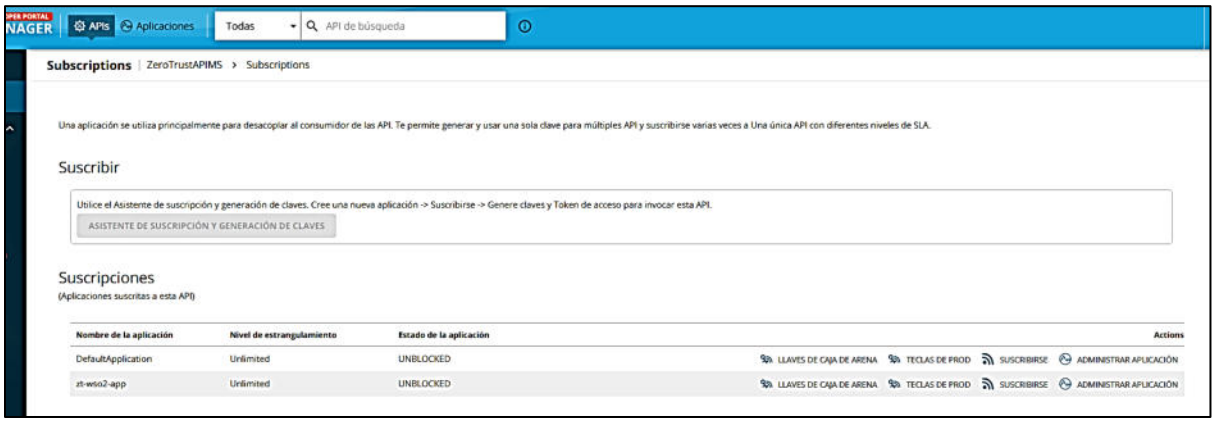


Figura 121.

Consumo exitoso de la API con aplicación consumidora suscrita

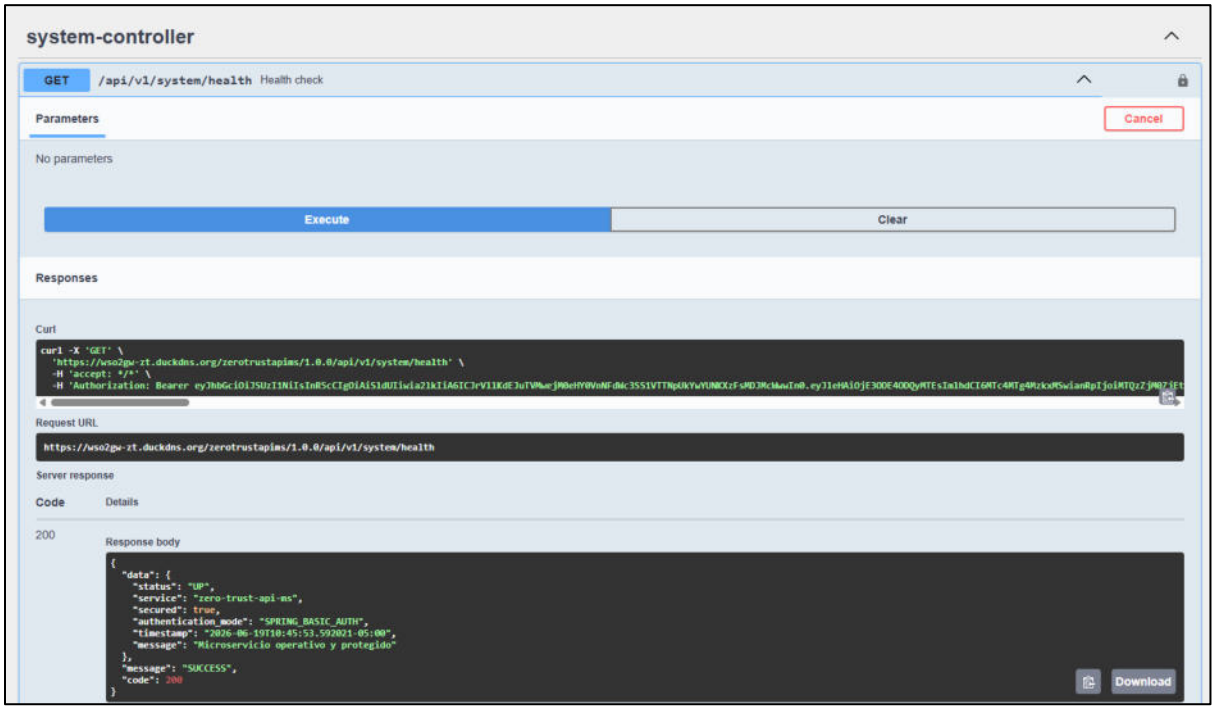
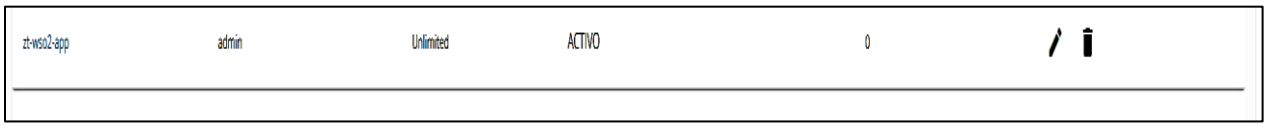


Figura 122.

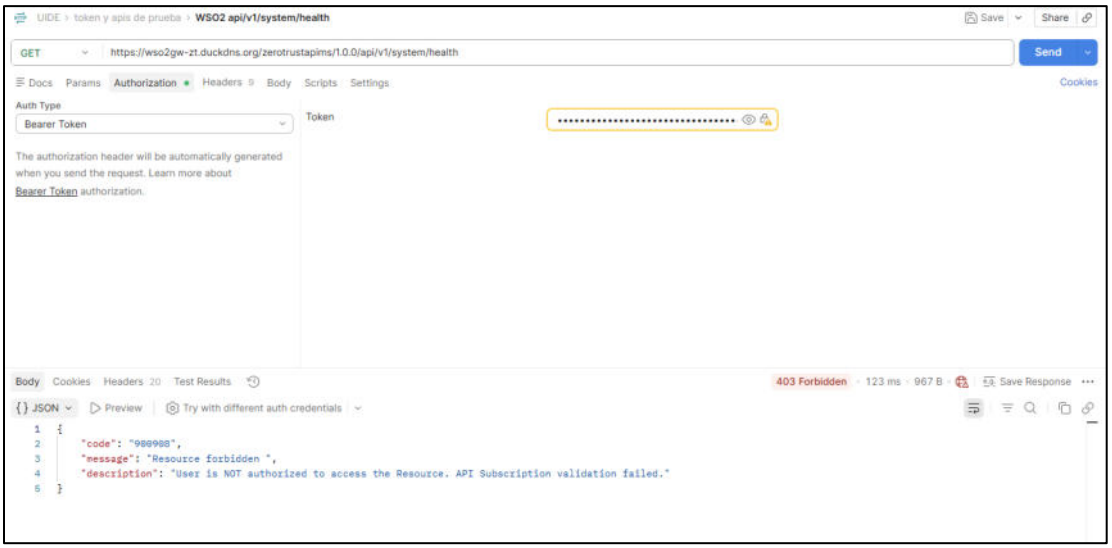
Aplicación consumidora zt-wso2-app registrada en WSO2



Posteriormente, se ejecutó una solicitud en un escenario donde la validación de suscripción no fue aceptada por WSO2. En este caso, el API Gateway respondió con HTTP 403 Forbidden y el código 900908, acompañado del mensaje "Resource forbidden" y la descripción "User is NOT authorized to access the Resource. API Subscription validation failed.". Este resultado demuestra que WSO2 no permite el consumo de la API únicamente por presentar un token Bearer, sino que también exige que la aplicación consumidora cuente con una suscripción válida y autorizada.

Figura 123.

Bloqueo de consumo por validación fallida de suscripción en WSO2



Finalmente, al contar nuevamente con la suscripción activa de la aplicación zt-wso2-app, se repitió el consumo del endpoint protegido y se obtuvo una respuesta HTTP 200 OK. Con ello se verificó que la suscripción de la aplicación consumidora forma parte del control de acceso aplicado por WSO2 antes de permitir el paso de la solicitud hacia el microservicio.

Figura 124.

Restauración de la suscripción de la aplicación consumidora en WSO2

Figura 125.

Consumo exitoso posterior a la validación de suscripción

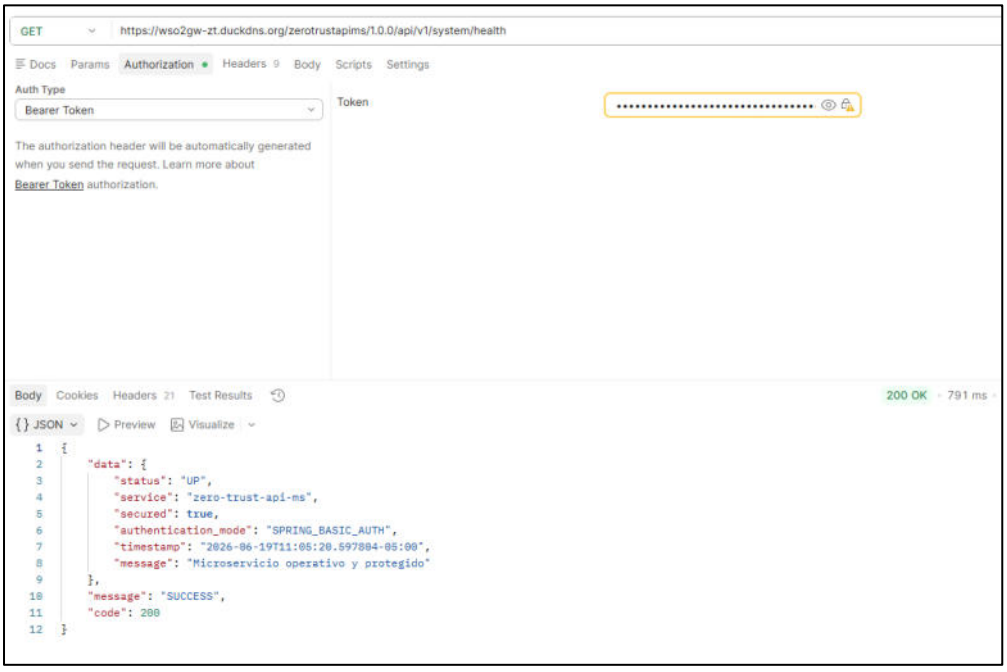


Tabla 17.

Tabla de resultados

N.º	Prueba ejecutada	Componente evaluado	Resultado esperado	Resultado obtenido	Estado
1	Consumo de la API protegida mediante token Bearer desde el Developer Portal	WSO2 API Gateway / Keycloak	Validar el token, confirmar la suscripción de la aplicación y reenviar la solicitud al microservicio	La solicitud fue procesada correctamente, obteniendo HTTP 200 OK con el token validado y la suscripción confirmada	Cumplido
2	Acceso mediante API Gateway y acceso directo al microservicio	WSO2 API Gateway / Microservicio Spring Boot	Verificar si el backend identifica el paso por WSO2 y si el acceso directo está restringido	La solicitud por WSO2 respondió HTTP 200, pero el endpoint indicó gateway_detected: false. La solicitud directa al microservicio también respondió HTTP 200 con gateway_detected: false	Parcial / Hallazgo técnico
3	Limitación de consumo por exceso de solicitudes	WSO2 API Manager	Bloquear solicitudes repetitivas al superar el límite del plan de consumo	Se configuró la política ZT_5_REQ_MIN con 5 solicitudes por minuto. En Postman Runner se ejecutaron 20 solicitudes y se observaron respuestas	Cumplido

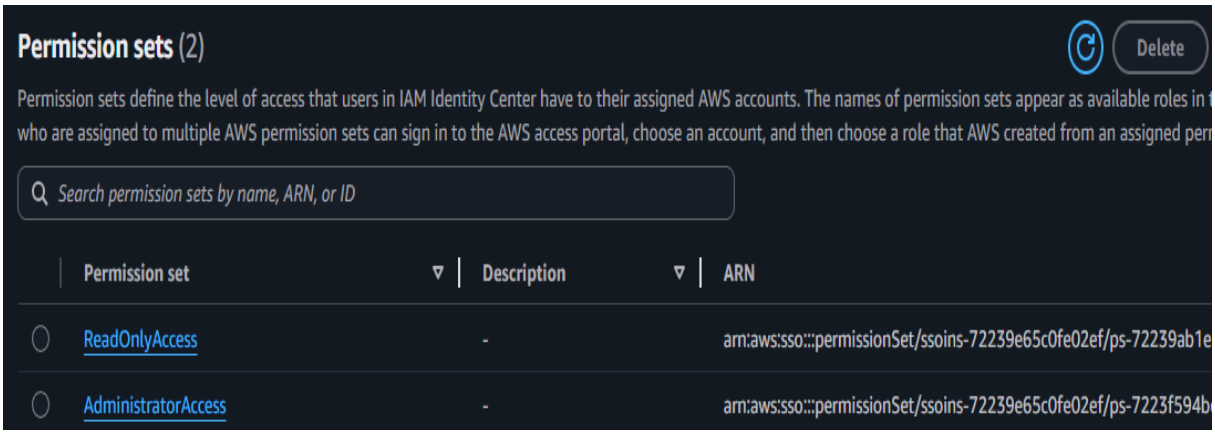
				HTTP 429 en varias iteraciones	
4	Validación de suscripción de aplicación consumidora	WSO2 Developer Portal / API Gateway	Permitir el consumo solo cuando la aplicación tenga suscripción válida	Con suscripción válida se obtuvo HTTP 200 OK. Cuando falló la validación de suscripción, WSO2 respondió HTTP 403 Forbidden con código 900908 y mensaje API Subscription validation failed	Cumplido

3.1.26. **Configuración de CyberArk para protección de accesos a AWS**

En la configuración en AWS en la sección de IAM Identity Center se configurará 2 perfiles de acceso que son: AdministratorAccess y ReadOnlyAccess, el primero permite realizar cambios en el Tenant de AWS y el otro solo de auditoría.

Figura 126.

Set de permisos de acceso al portal de administración de AWS



En el portal de Cloud Security de CyberArk se creará una nueva organización de AWS, lo que permite esta integración es tener acceso al Tenant para configurar de forma automática los Stacks de CloudFormation utilizados en la integración.

Figura 127.

Datos de configuración de AWS para la creación de la organización

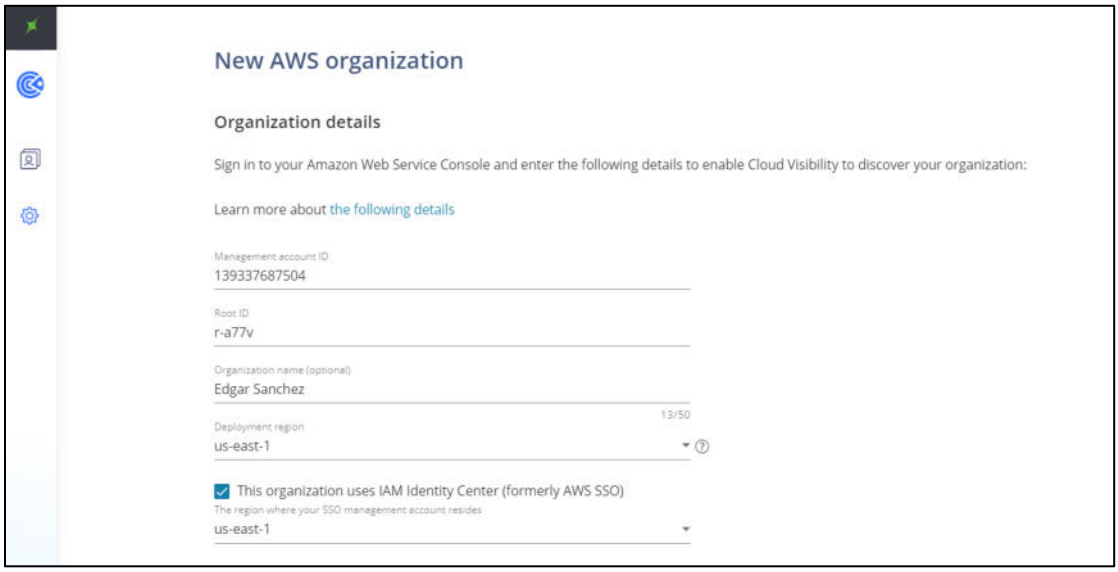
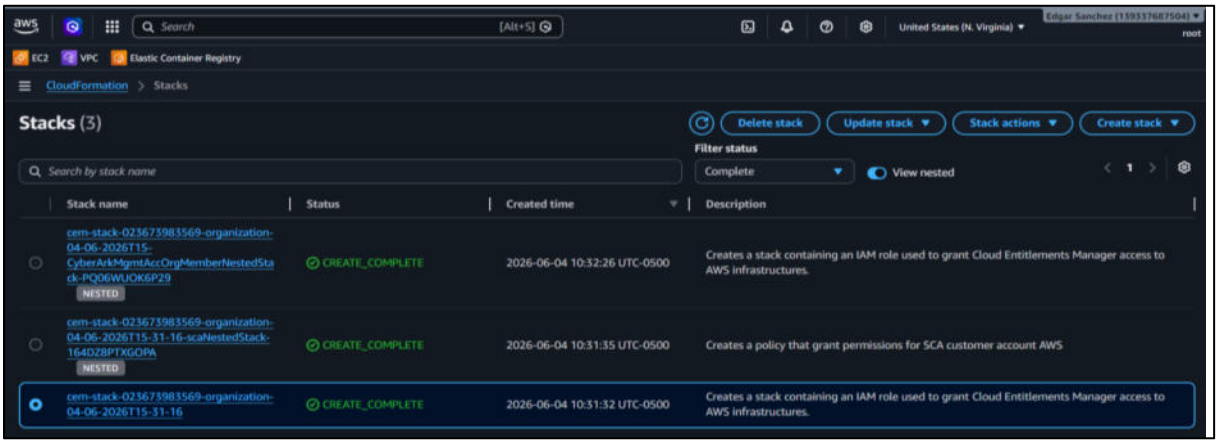


Figura 128.

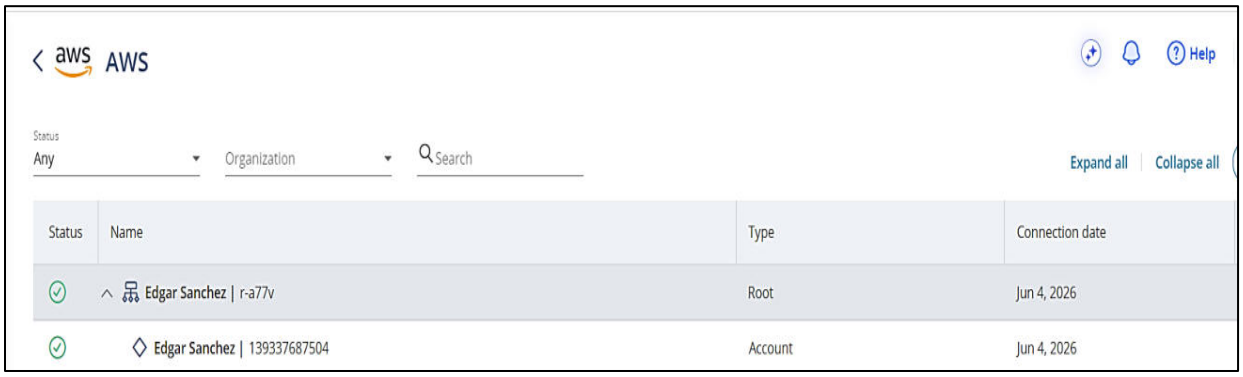
AWS configuración automática de Stacks en CloudFormation



Se comprueba que los Stack se hayan configurado correctamente en AWS CloudFormation.

Figura 129.

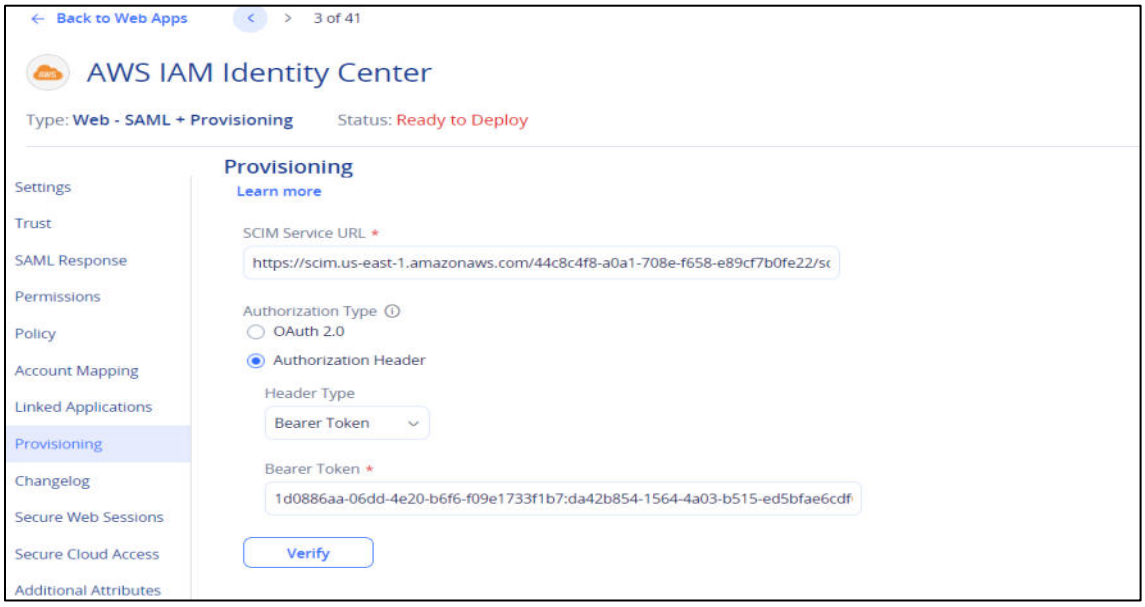
Portal CyberArk verificación de integración con la organización AWS



En CyberArk comprobamos que la organización Edgar Sanchez está registrada en el portal.

Figura 130.

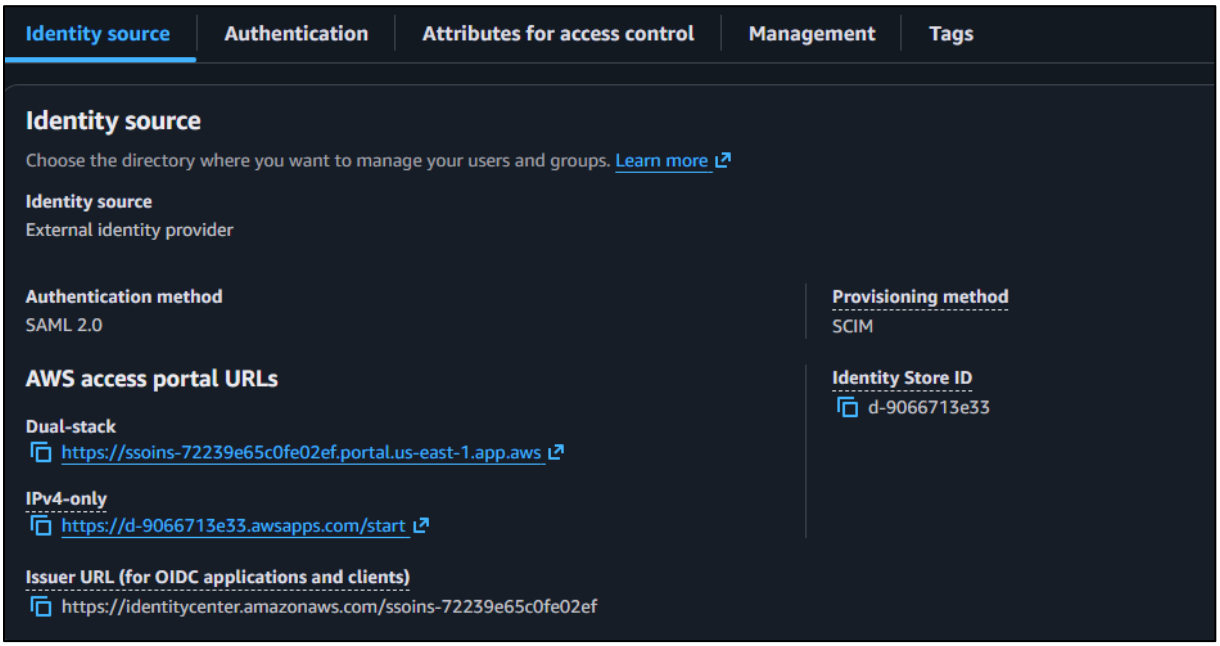
Configuración de aplicación AWS IAM Identity Center en el portal de CyberArk



Con la integración se procederá a configurar la aplicación como IdP o fuente de identidad para la autenticación y aprovisionamiento automático de los usuarios.

Figura 131.

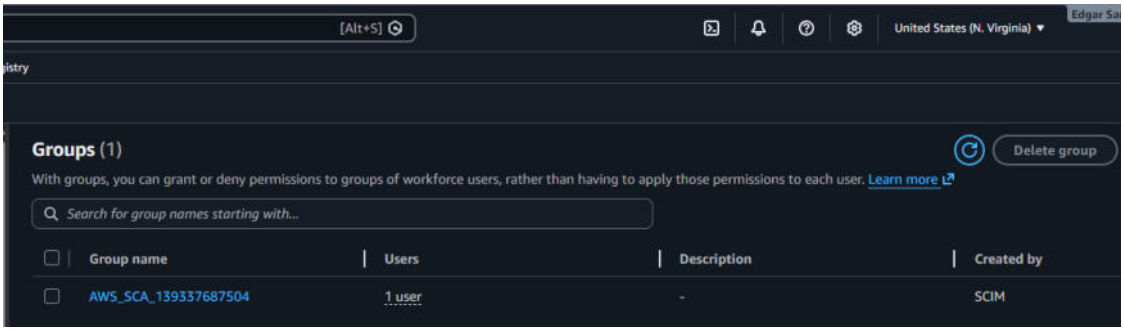
Configuración de External Identity Provider en IAM Identity Center con CyberArk



En el portal de AWS IAM Identity Center procedemos a finalizar la integración mediante el uso de SAML con la metadata de CyberArk.

Figura 132.

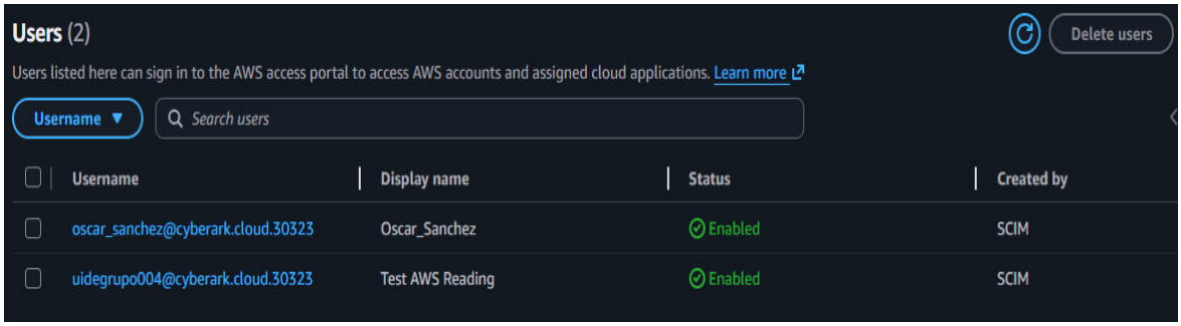
Comprobación de aprovisionamiento del grupo AWS_SCA_139337687504 en AWS



En CyberArk se tiene el grupo AWS_SCA_139337687504 y como se observa el mismo está en el portal de grupos de IAM Identity Center, con lo que se confirma el aprovisionamiento desde CyberArk.

Figura 133.

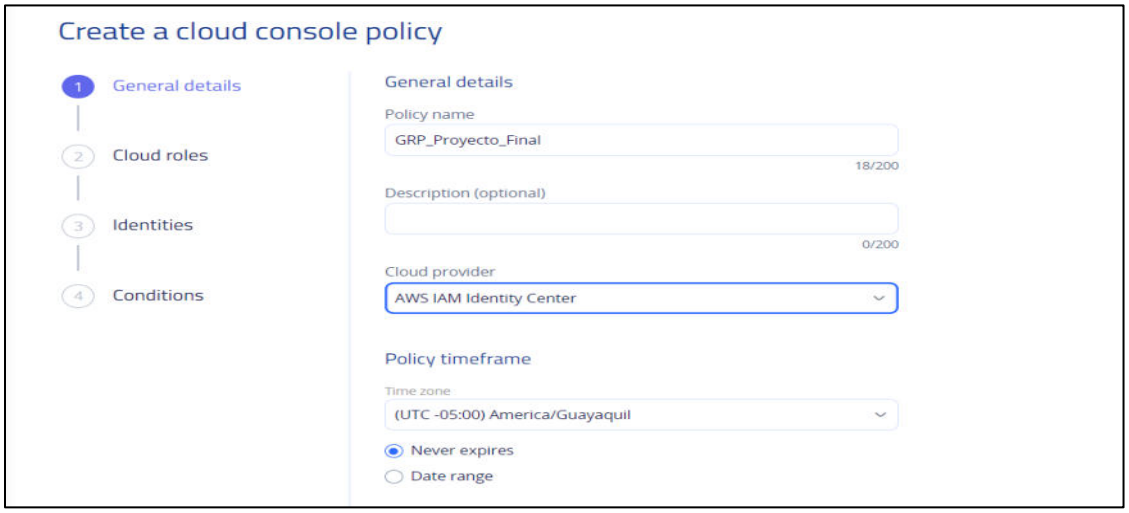
Verificación de aprovisionamiento de usuarios en AWS



En la imagen se observa 2 usuarios que en la forma de creación indica SCIM, que son cuentas que se han creado a través de este protocolo, por lo que se confirma que fueron aprovisionados desde CyberArk.

Figura 134.

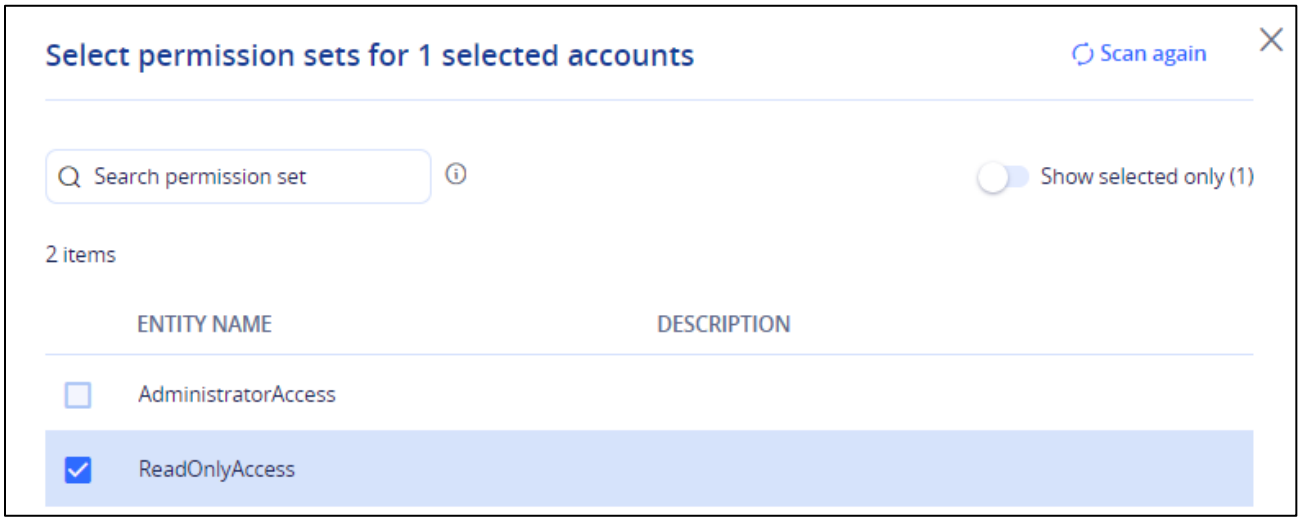
Configuración de política de acceso en el portal de CyberArk para la administración AWS



En la federación se configura la política en la cual se establece los permisos que obtendrá el usuario para el acceso al portal de AWS, en este caso tenemos 2 usuarios, uno con permisos de configuración y otro con permisos de lectura.

Figura 135.

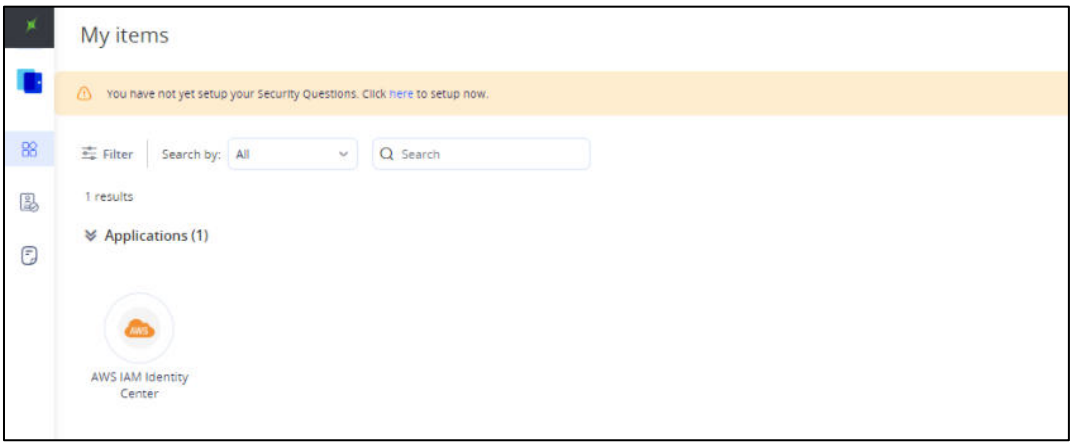
Permisos de acceso al portal de AWS definidos dentro del portal de CyberArk



Los usuarios federados no tienen permisos estáticos configurados, a través de CyberArk definimos los permisos que tendrán en el portal de AWS, así como la duración de la conexión, horarios de conexión y si requieren aprobación adicional de conexión.

Figura 136.

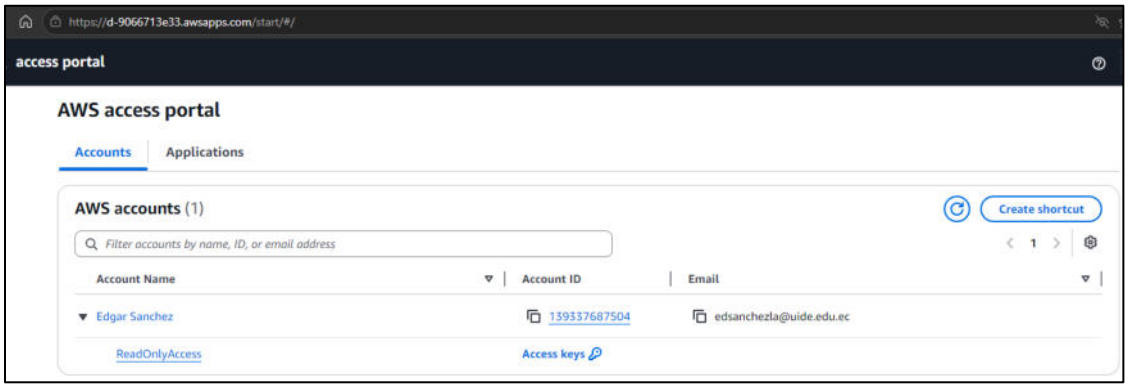
Portal de acceso de usuario final a través de CyberArk



El usuario final creado para el acceso a través de CyberArk es: uidegrupo004@cyberark.cloud.30323, a este usuario el portal de acceso le muestra la aplicación de AWS IAM Identity Center con la cual puede conectarse al portal de AWS.

Figura 137.

Acceso al portal de AWS con los permisos definidos en el portal de CyberArk



Los permisos en este caso son los configurados a través del portal de CyberArk de ReadOnlyAccess, y con el usuario aprovisionado desde CyberArk, en este proceso el acceso al portal es únicamente por los usuarios federados.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

En el presente capítulo se presentan los resultados obtenidos mediante las pruebas realizadas sobre el modelo de seguridad propuesto tras completar la implementación y configuración de los componentes descritos en el Capítulo 3 cuya validación se llevó a cabo tras la ejecución de diferentes pruebas dentro de un entorno de laboratorio controlado con el fin de verificar el correcto funcionamiento de los mecanismos de autenticación, autorización, auditoría, control perimetral y administración federada de identidades.

El análisis de los resultados permite evaluar el cumplimiento de los objetivos definidos para la investigación y determinar la efectividad de los controles de seguridad que fueron implementados bajo los principios de Zero Trust y mínimo privilegio que fueron definidos para la arquitectura propuesta.

4.1.Pruebas de Concepto

Con el propósito de evaluar el esquema integral de autenticación y autorización segura para APIs propuesto en la presente investigación se ejecutaron diferentes pruebas de concepto dentro del entorno de laboratorio implementado cuyas pruebas permitieron validar el funcionamiento de los mecanismos de autenticación, autorización, auditoría, gestión de secretos y administración federada de identidades verificando el cumplimiento de los principios de Zero Trust y mínimo privilegio como base para la protección de las APIs.

4.1.1. Validación de autenticación y autorización mediante Keycloak

La primera prueba de concepto tuvo como finalidad validar el funcionamiento de los mecanismos de autenticación y autorización implementados mediante Keycloak para lo cual se ejecutaron diferentes escenarios de acceso utilizando usuarios con diferente perfiles y niveles de privilegio permitiendo verificar el comportamiento de los controles establecidos de acuerdo con los permisos asignados.

Tabla 18.

Resultados de autenticación y autorización mediante Keycloak

N.º	Escenario	Usuario	API probado	Resultado esperado	Resultado obtenido
1	Solicitud sin token	Ninguno	GET /api/v1/system/health	401	401
2	Token válido y rol autorizado	platform.admin	GET /api/v1/system/health	200	200
3	Lectura de recurso protegido	reader.user	GET /api/v1/protected-resources	200	200
4	Creación sin privilegio suficiente	reader.user	POST /api/v1/protected-resources	403	403
5	Creación con rol autorizado	writer.user	POST /api/v1/protected-resources	201	201
6	Consulta de auditoría sin rol auditor	reader.user	GET /api/v1/audit/events	403	403

7	Consulta de auditoría con rol auditor	security.auditor	GET /api/v1/audit/events	200	200
8	Acceso total de administrador	platform.admin	APIs principales	200/201	200/201
9	Token JWT expirado	platform.admin	GET /api/v1/system/health	401	401
10	Token JWT mal formado	No aplica	GET /api/v1/system/health	401	401
11	Token JWT con firma manipulada	platform.admin	GET /api/v1/system/health	401	401

Los resultados obtenidos evidenciaron una coincidencia total entre los resultados esperados y los resultados observados durante la ejecución de las pruebas confirmando el comportamiento previsto de los mecanismos implementados debido a que las solicitudes realizadas sin autenticación fueron rechazadas correctamente mediante HTTP 401, mientras que las solicitudes realizadas con credenciales válidas fueron procesadas de forma exitosa.

De igual manera, los usuarios con permisos insuficientes recibieron respuestas HTTP 403 Forbidden, mientras que los usuarios con privilegios adecuados ejecutaron correctamente las operaciones autorizadas para su rol confirmando que los controles de autorización aplicaron las restricciones definidas conforme a los niveles de acceso establecidos para la arquitectura propuesta.

4.1.2. Validación de políticas de seguridad mediante WSO2 API Gateway

La segunda prueba de concepto tuvo como finalidad validar el funcionamiento de las políticas de seguridad configuradas en WSO2 API Manager como componente de control perimetral para lo cual se verificó la validación de tokens emitidos por Keycloak, el comportamiento del sistema ante intentos de acceso directo al microservicio, el funcionamiento de las políticas de limitación de consumo además de la validación de suscripción de la aplicación consumidora.

Como parte de la primera verificación se consumió el endpoint GET/api/v1/system/health desde el Developer Portal de WSO2 utilizando un token Bearer emitido por Keycloak para el cliente zt-backend-client y la aplicación zt-wso2-app cuya solicitud fue enviada a través de la URL publicada por el API Gateway bajo el esquema de seguridad OAuth2, obteniendo como resultado una respuesta HTTP 200 OK que confirmó que WSO2 validó correctamente el token, la suscripción de la aplicación y la audiencia esperada antes de reenviar la solicitud al microservicio.

En la siguiente etapa se evaluó la posibilidad si el microservicio podía ser consumido de forma directa sin pasar por el API Gateway utilizando el endpoint /api/v1/gateway/context donde la solicitud realizada a través de WSO2 respondió HTTP 200 OK indicando "gateway_detected": false motivo por el cual se repitió la misma solicitud directamente sobre la URL pública del microservicio obteniendo nuevamente una respuesta HTTP 200 OK confirmando que el backend pudo ser consumido sin pasar por el gateway siempre que se contara con un token válido constituyendo un hallazgo relevante dentro del laboratorio al evidenciar que el acceso directo al backend no se encuentra completamente restringido a nivel de red o proxy.

Posteriormente se validó el mecanismo de limitación de consumo configurando en WSO2 una política de suscripción denominada ZT_5_REQ_MIN con un límite de cinco solicitudes por minuto para lo cual mediante Postman Runner se ejecutaron veinte solicitudes consecutivas sobre el endpoint protegido observándose respuestas HTTP 429 en varias de las iteraciones una vez superado el límite configurado en la política.

Como parte de la validación final se verificó que el consumo de la API dependiera de una suscripción válida de la aplicación consumidora simulando un escenario donde dicha suscripción no era aceptada ante lo cual WSO2 respondió con HTTP 403 Forbidden y el código 900908 indicando que la aplicación no contaba con autorización para acceder al recurso mientras que al restablecer la suscripción de zt-wso2-app el consumo del endpoint volvió a responder correctamente con HTTP 200 OK confirmando el funcionamiento esperado del mecanismos de control implementado.

Tabla 19.

Resultados de políticas de seguridad mediante WSO2 API Gateway

N.º	Escenario	Componente evaluado	Resultado esperado	Resultado obtenido
1	Consumo de la API mediante token Bearer desde el Developer Portal.	WSO2 API Gateway / Keycloak	Validar el token y reenviar la solicitud al microservicio.	200
2	Acceso directo al microservicio sin pasar por el API Gateway.	WSO2 API Gateway / Microservicio Spring Boot.	Verificar si el acceso directo está restringido a nivel de red.	200 (sin restricción de red)

3	Limitación de consumo por exceso de solicitudes.	WSO2 API Manager	Bloquear solicitudes que superen el límite del plan de consumo.	429 a partir de la solicitud que excede la cuota.
4	Validación de suscripción de la aplicación consumidora.	WSO2 Developer Portal / API Gateway	Permitir el consumo solo con suscripción válida.	403 sin suscripción / 200 con suscripción activa.

En los resultados obtenidos se evidenciaron que WSO2 validó correctamente los tokens emitidos por Keycloak además de aplicar de forma efectiva las políticas de suscripción y limitación de consumo configuradas para la API comprobando que las solicitudes que excedieron el límite establecido fueron bloqueadas mediante el código HTTP 429 mientras que las solicitudes provenientes de aplicaciones sin suscripción activa fueron rechazadas con HTTP 403.

La prueba de acceso directo al microservicio permitió identificar que la restricción del API Gateway como único punto de entrada no se garantiza únicamente por la configuración de WSO2 sino que requiere reforzarse a nivel de infraestructura mediante reglas de Security Group configuración de Nginx o validación obligatoria de encabezados internos generados por el Gateway constituyendo un hallazgo que no compromete la validación de identidad del esquema debido que en ambos escenarios el token fue verificado correctamente aunque representa un punto de mejora para fortalecer la segmentación de red del entorno de laboratorio.

4.1.3. Validación de auditoría y trazabilidad

Con la finalidad de garantizar la trazabilidad de los eventos generados dentro de la plataforma se validó el funcionamiento de los mecanismos de auditoría implementados mediante las tablas de registro definidas para el laboratorio.

Tabla 20.

Resultados de auditoría.

Elemento evaluado	Resultado esperado	Resultado obtenido
Registro de usuario	Registrar identidad del usuario	Registrado correctamente
Registro de endpoint	Registrar recurso consumido	Registrado correctamente
Registro de dirección IP	Registrar origen de la solicitud	Registrado correctamente
Registro de método HTTP	Registrar operación ejecutada	Registrado correctamente
Registro de resultado	Registrar código de respuesta	Registrado correctamente
Registro temporal	Registrar fecha y hora del evento	Registrado correctamente

Los resultados demostraron que los eventos generados durante las pruebas fueron almacenados correctamente permitiendo mantener trazabilidad sobre las operaciones realizadas por los usuarios dentro de la plataforma.

La información registrada facilita procesos de monitoreo, análisis forense e investigación de incidentes de seguridad contribuyendo evidencia útil para el seguimiento de los eventos además de contribuir al cumplimiento de los principios de visibilidad continúa establecidos por el modelo Zero Trust.

4.1.4. Validación de acceso federado a AWS mediante CyberArk Cloud Security

La siguiente prueba estuvo orientada a validar el acceso federado al portal de AWS mediante CyberArk Cloud Security e IAM Identity Center verificando la correcta sincronización de usuarios, grupos y asignaciones de permisos conforme al principio de mínimo privilegio permitiendo comprobar que el proceso de autenticación y autorización respondiera de manera consistente a las políticas definidas para la administración de identidades.

Tabla 21.

Resultados de Accesos

Elemento evaluado	Resultado obtenido
Sincronización de usuarios	Correcto
Sincronización de grupos	Correcto
Aprovisionamiento mediante SCIM	Correcto
Aplicación de permisos	Correcto
Acceso federado al portal AWS	Correcto
Administración centralizada de accesos	Correcto
Aplicación del principio de mínimo privilegio	Correcto

Los resultados obtenidos demostraron la correcta sincronización de identidades entre las plataformas además de la adecuada aplicación de los permisos asignados para el acceso a recursos dentro del entorno AWS verificando que las autorizaciones configuradas fueran aplicadas de manera consistente durante la ejecución de pruebas.

La integración permitió administrar usuarios y permisos desde una plataforma centralizada reduciendo la dependencia de credenciales locales y fortaleciendo la gobernanza de identidades dentro del entorno cloud favoreciendo un mayor control sobre los accesos.

4.2.Análisis de Resultados

Los resultados obtenidos durante la ejecución de las pruebas de concepto permiten concluir que el esquema integral de autenticación y autorización implementado cumple satisfactoriamente con los objetivos definidos en la investigación.

Tabla 22.

Resultados en base a objetivos propuestos.

Objetivo específico	**Evidencia obtenida**	**Nivel de cumplimiento**
Diseñar una arquitectura de seguridad para APIs bajo principios Zero Trust y mínimo privilegio	Integración de Keycloak, WSO2 API Manager, Spring Boot, PostgreSQL, CyberArk Cloud Security e IAM Identity Center	Cumplido
Implementar autenticación y autorización mediante JWT firmados digitalmente	Validación de tokens válidos, expirados, mal formados y firma manipulada	Cumplido
Configurar políticas de seguridad en el API Gateway	Validación de solicitudes, suscripciones, rate limiting y trazabilidad mediante WSO2 API Manager	Cumplido
Implementar acceso federado al portal AWS	Integración CyberArk Cloud Security e IAM Identity Center	Cumplido
Construir un laboratorio de validación	Implementación del entorno de laboratorio y escenarios de prueba	Cumplido
Evaluar el desempeño del esquema propuesto	Resultados obtenidos durante las pruebas de autenticación, autorización y acceso federado	Cumplido

Durante las pruebas realizadas mediante Keycloak se verificó que solo los usuarios con credenciales validas pudieron acceder a los recursos protegidos además de comprobar que los usuarios que no disponían de los permisos requeridos recibieron respuestas de denegación de acceso a pesar de haber completado correctamente el proceso de autenticación confirmando que resultados fueron los esperados porque coincide con el comportamiento definido en la

configuración de roles y permisos lo que confirma el correcto funcionamiento de los controles de acceso implementados.

Por otra parte, los mecanismos de auditoría implementados registraron información importante durante la ejecución de pruebas permitiendo identificar los accesos realizados las solicitudes procesadas y el resultado de cada operación cuya información almacenada facilitó el seguimiento de actividades ejecutadas dentro de la plataforma proporcionando evidencia útil para las tareas de monitoreo y revisión de eventos.

En cuanto a la administración de identidades la integración entre CyberArk Cloud Security y AWS IAM Identity Center permitió centralizar la gestión de usuarios, grupos y permisos dentro del entorno cloud facilitando el aprovisionamiento automatizado y además de reducir riesgos asociados al uso de credenciales estáticas.

En conjunto con los resultados obtenidos permitieron comprobar que los mecanismos de autenticación, autorización, auditoria y administración de identidades operan de forma coordinada dentro de la arquitectura propuesta verificando que los controles implementados respondieron a los requerimientos establecidos para la protección de las APIs y microservicios considerados en el entorno de laboratorio.

CAPÍTULO 5

5. CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

5.1.1. Relación con los objetivos específicos del proyecto

La ejecución del proyecto permitió contrastar cada uno de los objetivos específicos definidos para la investigación con los resultados obtenidos durante la implementación y las pruebas de concepto desarrolladas permitiendo determinar el nivel de cumplimiento alcanzado para cada objetivo a partir de las evidencias técnicas recopiladas cuya descripción se presenta en los siguientes apartados.

El primer objetivo consistió en diseñar una arquitectura de seguridad para APIs con mecanismos de identidad control perimetral y gestión dinámica de secretos bajo principios de Zero Trust y mínimo privilegio alcanzando un cumplimiento satisfactorio mediante una arquitectura en capas compuesta por tres planos de control diferenciados conformado por Keycloak como proveedor de identidad centralizado WSO2 API Manager como componente de control perimetral y Spring Boot actuando como Resource Server con validación independiente de tokens complementándose la solución mediante CyberArk Cloud Security para el acceso federado al portal de AWS eliminando el uso de credenciales estáticas centralizando la administración de accesos a recursos cloud y materializando el principio de nunca confiar, siempre verificar mediante controles aplicados en cada plano sin asumir confianza implícita entre sus componentes.

El segundo objetivo fue implementar un esquema de autenticación y autorización basado en tokens JWT firmados digitalmente definiendo criterios de emisión validación expiración y claims obligatorios alcanzándose su cumplimiento mediante la configuración del realm zt-production-realm en Keycloak donde se estableció el algoritmo asimétrico RS256 para la firma de tokens además de definir los claims obligatorios iss, sub, aud, exp, iat y roles de cliente junto con tiempos de expiración acordes con el principio de mínimo privilegio.

El microservicio desarrollado en Spring Boot fue configurado como Resource Server habilitando la validación de firma mediante el endpoint JWKS público de Keycloak garantizando así que ningún token auto-emitido o manipulado pudiera ser aceptado durante el proceso de autenticación mientras las pruebas ejecutadas con escenario con tokens expirados, ausentes y válidos confirmaron el comportamiento esperado del esquema con una tasa de efectividad del 100% en los ocho escenarios de prueba desarrolladas.

El tercer objetivo estuvo orientado a configurar políticas de seguridad en el API Gateway para validar solicitudes controlar el acceso a los servicios y reforzar la trazabilidad del consumo de APIs alcanzado su cumplimento mediante la integración de WSO2 API Manager con Keycloak como Key Manager externo a través de la configuración de claims, scopes y certificados JWKS además de la definición del contrato OpenAPI del microservicio como base para la creación de la API en WSO2 cuya activación de seguridad OAuth2 a nivel de recurso junto con la asignación de planes de consumo para control de tasa establecieron una primera línea de defensa perimetral verificándose su funcionamiento mediante el consumo exitoso de la API protegida mediante token Bearer a través de WSO2.

El cuarto objetivo consistió en implementar el acceso federado al portal de AWS mediante CyberArk Cloud Security configurando la integración con IAM Identity Center para gestionar de forma centralizada el acceso de usuarios a recursos cloud evidenciándose su cumplimiento mediante la integración de CyberArk con AWS mediante la configuración del proveedor de identidad externo el aprovisionamiento automatizado de grupos y usuarios además de la definición de políticas de acceso granulares desde el portal de CyberArk garantizando que las credenciales utilizadas para acceder a recursos cloud sean administradas rotadas y auditadas de forma centralizada eliminando el riesgo asociado al uso de access keys estáticas incorporadas en código o archivos de configuración identificado como una de las vulnerabilidades más frecuentes identificadas en el estado del arte.

El quinto objetivo consistió en construir un laboratorio de validación que reprodujera el flujo de autenticación autorización y consumo de APIs en una arquitectura controlada alcanzándose su cumplimiento mediante la implementación del laboratorio sobre AWS con los componentes Keycloak WSO2 el microservicio Spring Boot zero-trust-api-ms y la base de datos PostgreSQL zero_trust_lab_db cuya estructura incorporó tres esquemas funcionales denominados core para los recursos protegidos audit para eventos de seguridad y lab para escenarios y ejecuciones de prueba permitiendo reproducir de manera controlada y repetible los flujos de autenticación y autorización además como simular escenarios de ataque con distintos perfiles de usuario y estados de token.

El sexto y último objetivo consistió en evaluar el desempeño del esquema propuesto mediante pruebas de autenticación autorización y escenarios de ataque controlados considerando indicadores técnicos previamente definidos evidenciándose su cumplimiento mediante las ocho pruebas ejecutadas sobre el realm zt-production-realm con los usuarios platform.admin

reader.user writer.user y security.auditor cuyos resultados coincidieron en su totalidad con los valores esperados confirmando mediante códigos de respuesta HTTP 401 para solicitudes sin token HTTP 403 para accesos con privilegios insuficientes y HTTP 200/201 para solicitudes autorizadas confirmando el correcto funcionamiento del esquema de control de acceso basado en roles (RBAC) implementado tanto en el backend como en el API Gateway.

5.1.2. Valoración del modelo de seguridad implementado

El modelo de seguridad desarrollado en este proyecto demuestra que es técnicamente factible implementar un esquema integral de autenticación y autorización para APIs en entornos empresariales reales sin requerir el rediseño de las aplicaciones existentes alcanzadose este resultado mediante la selección de tecnologías de código abierto como Keycloak y WSO2 combinadas con una solución empresarial de gestión de acceso privilegiado como CyberArk permitiendo construir una arquitectura que aplica el principio de defensa en profundidad de forma efectiva.

La decisión de utilizar tokens JWT firmados con RS256 en lugar de HS256 fue determinante para garantizar la separación entre el servidor de autorización y los servidores de recursos permitiendo que cualquier servicio que disponga de la clave pública de Keycloak valide localmente un token sin la necesidad de realizar consultas permanentes al servidor de autorización en cada solicitud favoreciendo el rendimiento del sistema además de reducir la superficie de ataque asociada al componente de gestión de identidades.

La implementación de una matriz de autorización para la API integrada por quince endpoints distribuidos en seis módulos funcionales System, Identity Protected Resources Gateway Secrets y Audit y cinco roles diferenciados API_USER RESOURCE_READER

RESOURCE_WRITER RESOURCE_ADMIN y PLATFORM_ADMIN demostró que el principio de mínimo privilegio puede aplicarse de manera granular y mantenible dentro de arquitecturas basadas en microservicios verificándose que ningún rol concentró permisos innecesarios mientras la inexistencia de un rol con acceso global salvo PLATFORM_ADMIN para escenarios administrativos refleja una correcta aplicación del principio de segregación de funciones.

La integración de CyberArk con AWS IAM Identity Center representó el principal elemento diferenciador del proyecto respecto a implementaciones convencionales de seguridad para APIs debido a que hizo posible el aprovisionamiento de identidades federadas la administración de accesos con tiempo de vida limitados y la auditoria de todos los accesos privilegiados desde una consola centralizada representando un avance importante frente al modelo tradicional basado en credenciales estáticas que aún permanece en muchas organizaciones.

Finalmente los resultados obtenidos durante las pruebas de concepto permitieron confirmar que el esquema propuesto es robusto ante los vectores de ataque más comunes identificados por el OWASP API Security Top 10 (2023) especialmente en lo que respecta a Broken Object Level Authorization Broken Authentication y Broken Function Level Authorization verificándose que la combinación de mecanismos de validación implementados en múltiples capas gateway, resource server y la base de datos de auditoría garantiza que una vulnerabilidad presente en una capa no comprometa la seguridad del sistema completo.

5.2. Recomendaciones

A partir de los hallazgos obtenidos durante el desarrollo y evaluación del proyecto fue posible identificar oportunidades de mejora además de líneas de trabajo futuro que permitirían extender el alcance del modelo de seguridad implementado hacia escenarios de mayor complejidad y exigencia operativa cuyas recomendaciones se presentan a continuación con el propósito de servir como referencia para sus entornos productivos.

5.2.1. Implementación de TLS para comunicación entre microservicios

Aunque el esquema implementado en el presente proyecto protege eficazmente el acceso externo a las APIs mediante JWT y el API Gateway la comunicación entre microservicios dentro del clúster no fue abordada en el alcance del trabajo por lo que se recomienda complementar el modelo mediante la implementación de autenticación mutua mediante certificados TLS (mTLS) preferentemente gestionada a través de un service mesh como Istio o Linkerd permitiendo que únicamente los servicios con una identidad criptográficamente verificada puedan comunicarse entre sí reduciendo el riesgo de movimiento lateral en caso de que un componente interno sea comprometido.

5.2.2. Incorporación de rotación automática de tokens y gestión avanzada de sesiones

El esquema implementado actualmente establece tiempos de expiración fijos para los tokens JWT por lo que se recomienda explorar la implementación de Refresh Token Rotation con invalidación automática en caso de reutilización complementando esta funcionalidad con un mecanismo de administración centralizada de sesiones activas en Keycloak además de evaluar para escenarios con mayores exigencias de seguridad la integración con mecanismos de

detección de anomalías que permitan invalidar tokens activos ante comportamientos sospechosos como accesos desde ubicaciones geográficas inusuales o patrones de consumo atípicos.

5.2.3. Automatización de pruebas de seguridad en el pipeline de CI/CD

Las pruebas de concepto ejecutadas durante este proyecto fueron realizadas de forma manual utilizando Postman por lo que para entornos productivos se recomienda integrar las pruebas de seguridad en el pipeline de integración y despliegue continuo mediante herramientas como OWASP ZAP Trivy o Snyk permitiendo detectar de forma temprana regresiones en las políticas de seguridad configuraciones incorrectas de tokens o exposición accidental de endpoints no protegidos ante cada modificación realizada sobre el código o la configuración del sistema.

5.2.4. Implementación de Observabilidad y correlación de eventos de seguridad

Si bien el proyecto incorporó una tabla de auditoría en la base de datos PostgreSQL para el registro de eventos se recomienda ampliar esta capacidad mediante la implementación de una plataforma de observabilidad centralizada integrando una solución SIEM como Elastic SIEM Splunk o AWS Security Hub con el propósito de correlacionar en tiempo real los eventos de autenticación y autorización generados por Keycloak los registros del API Gateway WSO2 y las evidencias de registros de auditoría del microservicio en tiempo real fortaleciendo la detección de patrones de ataque persistente y para el cumplimiento de marcos normativos como ISO 27001, SOC 2 y NIST SP 800-53.

5.2.5. Evaluación de rendimiento y escalabilidad bajo carga

El laboratorio implementado fue diseñado para validar la corrección operación funcional del esquema de seguridad sin considerar su comportamiento bajo condiciones de carga elevada

por lo que se recomienda realizar pruebas de rendimiento utilizando herramientas como Apache JMeter o k6 para evaluar la latencia generada por la validación de tokens JWT en el API Gateway el impacto del proceso de introspección de tokens y el comportamiento del esquema bajo escenarios de alta concurrencia permitiendo dimensionar adecuadamente los recursos de cómputo y memoria necesarios para garantizar los niveles de servicio en entornos productivos.

5.2.6. Adopción del modelo como estándar de referencia organizacional

El esquema de seguridad diseñado en este proyecto presenta un alto nivel de modularidad además de un grado de independencia respecto al dominio de negocio como para ser adoptado como referencia para organizaciones que expongan APIs en entornos distribuidos por lo que se recomienda documentar el modelo como un Architecture Decision Record (ADR) además de establecer un proceso de revisión periódica que incorpore las actualizaciones del OWASP API Security Top 10 los cambios en las políticas de Keycloak y WSO2 junto con las recomendaciones emitidas por el NIST relacionadas con el modelo Zero Trust especialmente las directrices del NIST SP 800-207 y el más reciente NIST SP 800-228.

6. REFERENCIAS BIBLIOGRÁFICAS.

1. Akamai. (2026). ¿Cuáles son los riesgos de seguridad de las APIs? https://www.akamai.com/es/glossary/what-are-api-security-risks
2. Amazon Web Services. (2022, junio 26). What is AWS IAM Identity Center? https://docs.aws.amazon.com/singlesignon/latest/userguide/what-is.html
3. Amazon Web Services. (s. f.). Microservices on AWS. https://aws.amazon.com/es/microservices/
4. Auth0. (2024). Authentication and authorization. https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization
5. Auth0. (2024). JSON Web Tokens (JWT). https://auth0.com/docs/secure/tokens/json-web-tokens
6. Broadcom. (s. f.). Spring Boot documentation. https://docs.spring.io/spring-boot/
7. Cabascango, A., & Clavijo, M. (2025). Diseño de una arquitectura de software basada en microservicios para una empresa dedicada a la venta y distribución de servicios de energía eléctrica [Tesis de maestría, Universidad Politécnica Salesiana]. https://dspace.ups.edu.ec/bitstream/123456789/31988/1/MSQ1176.pdf
8. CyberArk. (2024). CyberArk Privileged Access Manager. https://www.cyberark.com/products/privileged-access-manager/
9. CyberArk. (s. f.). Secure Cloud Access. https://www.cyberark.com/products/secure-cloud-access/
10. Docker. (s. f.). Docker documentation. https://docs.docker.com/

11. Durán, G. G. (2025). Integración de GraphQL y seguridad Zero Trust en el diseño de APIs [Tesis de maestría, Universidad Politécnica Salesiana]. https://dspace.ups.edu.ec/bitstream/123456789/31813/1/MSQ1136.pdf
12. Fielding, R. T., Nottingham, M., & Reschke, J. (2022). RFC 9110: HTTP semantics. RFC Editor. https://www.rfc-editor.org/info/rfc9110
13. Hardt, D. (2012). The OAuth 2.0 authorization framework (RFC 6749). RFC Editor. https://doi.org/10.17487/RFC6749
14. IBM. (2024). Authentication vs. authorization. https://www.ibm.com/think/topics/authentication-vs-authorization
15. IBM. (2024). What is identity and access management (IAM)? https://www.ibm.com/think/topics/identity-access-management
16. Johnson, W. (2022, mayo 4). RS256 vs. HS256: What's the difference? Auth0. https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/
17. Jones, M. B. (2015). JSON Web Token (JWT) (RFC 7519). RFC Editor. https://doi.org/10.17487/RFC7519
18. JWT.io. (s. f.). Introduction to JSON Web Tokens. https://jwt.io/introduction#what-is-json-web-token
19. Keycloak. (2024). Server Administration Guide. https://www.keycloak.org/documentation
20. Kosinski, M., & Forrest, A. (2024). What is identity and access management (IAM)? IBM. https://www.ibm.com/think/topics/identity-access-management
21. Lindemulder, G., & Kosinski, M. (2024). What is Zero Trust? IBM. https://www.ibm.com/think/topics/zero-trust

22. Madden, N. (2020). API security in action. Manning Publications. https://cdn.ttgtmedia.com/rms/pdf/bookshelf_apisecurityinaction_excerpt.pdf
23. Microsoft. (2025, agosto 15). Active Directory Domain Services overview. Microsoft Learn. https://learn.microsoft.com/es-mx/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview
24. National Institute of Standards and Technology. (2020, septiembre). Security and privacy controls for information systems and organizations (NIST Special Publication 800-53, Revision 5). https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final
25. Newman, S. (2021). Building microservices (2nd ed.). O'Reilly Media. https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/
26. OpenID Foundation. (2023). OpenID Connect Core 1.0 incorporating errata set 2. https://openid.net/specs/openid-connect-core-1_0.html
27. Oracle. (2024, octubre 7). What is Zero Trust? https://www.oracle.com/latam/security/what-is-zero-trust/
28. OWASP Foundation. (2023). OWASP API Security Project. https://owasp.org/API-Security/
29. OWASP Foundation. (2023). OWASP API Security Top 10 – 2023. https://owasp.org/API-Security/editions/2023/en/0x11-t10/
30. Portoviejo, C., & Pacheco, M. (2024). Análisis, diseño y desarrollo de un sistema modular de gestión de expedientes salesianos con enfoque full-stack para implementación a nivel nacional [Tesis de grado, Universidad Politécnica Salesiana]. https://dspace.ups.edu.ec/bitstream/123456789/28917/1/UPS-CT011750.pdf

31. PostgreSQL Global Development Group. (s. f.). What is PostgreSQL? https://www.postgresql.org/docs/current/intro-whatis.html
32. Ramaswamy Chandramouli, & Butcher, Z. (2025, marzo 25). NIST Special Publication 800-228 (Initial Public Draft). National Institute of Standards and Technology. https://csrc.nist.gov/pubs/sp/800/228/ipd
33. Red Hat. (2020, mayo 8). What is a REST API? https://www.redhat.com/en/topics/api/what-is-a-rest-api
34. Red Hat. (2023, mayo 1). What are microservices? https://www.redhat.com/en/topics/microservices/what-are-microservices
35. Red Hat. (2024, 19 de septiembre). Securing applications and services guide. https://www.keycloak.org/docs/25.0.6/securing_apps/index.html
36. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero Trust Architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207
37. Secure Software Development. (s. f.). JSON Web Token (JWT). https://ssd.mrw0l05zyn.cl/recomendaciones/json-web-token-jwt
38. The MITRE Corporation. (2025, abril 25). ATT&CK: TA0008 – Lateral Movement. https://attack.mitre.org/tactics/TA0008/
39. WSO2. (2024). WSO2 API Manager documentation. https://apim.docs.wso2.com
40. WSO2. (s. f.). Governance overview. https://apim.docs.wso2.com/en/latest/administer/governance/overview/

7. APÉNDICES

Información de acceso a los laboratorios.

N.º	Componente	Portal / Recurso	URL de acceso	Usuario / Cuenta	Contraseña	Propósito dentro del laboratorio	Observación
1	Keycloak	Consola de administración de Keycloak	https://keycloak-zt.duckdns.org/	kc-admin-permanent	Joel14112400100@	Administración del proveedor de identidad, configuración del realm, cliente OIDC, usuarios, roles y emisión de tokens JWT.	Se utiliza para validar la capa de identidad del modelo Zero Trust.
2	Microservicio protegido	Swagger UI del backend Spring Boot	https://zt-api-secure.duckdns.org/zero/trust/swagger-ui/index.html	No aplica	No aplica	Consulta y prueba directa de los endpoints protegidos del microservicio.	Para consumir las APIs se requiere ingresar un token Bearer emitido por Keycloak.

3	WSO2 API Manager	Portal de administración / configuración del Key Manager	https://wso2-zt.duckdns.org/admin	admin	w502admPwd2o2g	Configuración administrativa de WSO2, políticas de seguridad, planes de consumo y Key Manager externo.	En este portal se administra la integración de WSO2 con Keycloak.
4	WSO2 Developer Portal	API Console / Try Out	https://wso2-zt.duckdns.org/devportal/apis/3bd73da1-f399-4256-b27e-95e14eb894f3/api-console	admin	w502admPwd2o2g	Ejecución de pruebas de consumo de la API publicada desde la opción Try Out.	Aquí se deben realizar las pruebas funcionales de consumo mediante WSO2.
5	WSO2 Publisher Portal	Portal de publicación de APIs	https://wso2-zt.duckdns.org/publisher/apis	admin	w502admPwd2o2g	Publicación, edición y administración de la API ZeroTrustAPIMS.	Se utiliza para configurar recursos, endpoints, seguridad OAuth2 y planes de consumo.
6	AWS / IAM Identity Center	Portal de acceso federado AWS	https://d-9066713e33.awsapps.com/start	uidegrupo004@cyberark.cloud.30323	UIDE2013proyectfinal.	Acceso federado al portal de AWS mediante CyberArk Cloud Security e IAM Identity Center.	Se utiliza para validar el acceso cloud sin credenciales estáticas locales.

7	Gmail para MFA	Cuenta de correo asociada a MFA	Gmail	uidegrupo004@gmail.com	UIDE2013proyectfinal.	Recepción o validación de códigos asociados al acceso del laboratorio.	Cuenta utilizada como apoyo para el proceso de autenticación multifactor. **Para fines de laboratorio el doble autentificador se desactivo.**
8	Postman	Colección de pruebas APIs UIDE	https://interstellar-crater-864591.postman.co/workspace/soa~189f224c-31fe-4877-8b68-03bf05fcd374/collection/18391880-12157313-a418-4168-bca2-ba8ae1bf1215?action=share&creator=18391880	Cuenta Postman autorizada	No aplica	Ejecución de solicitudes para obtener token, consumir APIs protegidas y validar escenarios de prueba.	La colección incluye la solicitud para obtener el token y consumir endpoints protegidos.

9	GitHub	Repositorio del proyecto	https://github.com/EdgarJoel123/UIDE_GRUPO_4.git	Cuenta GitHub autorizada	No aplica	Repositorio del código fuente, archivos de configuración y README del laboratorio.	Sirve como respaldo técnico del desarrollo implementado.