UNIVERSIDAD TECNICA FEDERICO SANTA MARIA | DEPARTAMENTO DE INFORMÁTICA

MAGÍSTER EN TECNOLOGÍAS DE LA INFORMACIÓN

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

"MODELO ASISTIDO POR AGENTES PARA LA CREACIÓN AUTOMATIZADA DE APIS EN AZURE"

LEONARDO FUENTES VEGA

TESINA PARA OPTAR AL GRADO
DE MAGÍSTER EN TECNOLOGÍAS DE
LA INFORMACIÓN

Profesor Guía: GASTÓN MÁRQUEZ ORTEGA

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☐ Memoria o trabajo de título ☒ Tesis de Postgrado

Título del trabajo: Modelo asistido por agentes para la creación automatizada de APIs en Azure

Nombre del candidato(a): Leonardo Alfredo Fuentes Vega

Carrera / Grado: Magíster en Tecnologías de la Información.

Campus: San Joaquín **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, **Gastón Márquez**, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 30-10-2026 **Firma:** _____



Estudiante o Candidato(a):

Fecha: 30-10-2026 **Firma:** _____



Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Modelo asistido por agentes para la creación automatizada de APIs en Azure

Leonardo Alfredo Fuentes Vega

Las Malvas 228, Las Condes, Región Metropolitana

Leonardo.fuentes@gmail.com

Resumen: En la industria bancaria, la centralización de la publicación de APIs en células de especialistas genera cuellos de botella operativos que degradan el *time-to-market* y dificultan la escalabilidad. Para resolver esto, este trabajo propone un modelo asistido por agentes de software orquestados en n8n e integrados con Azure API Management y GPT-4o, cuyo objetivo es automatizar la estandarización, generación de políticas de seguridad, despliegue y documentación de APIs, habilitando un esquema de autoservicio gobernado. La propuesta se validó mediante un diseño experimental cuantitativo comparando el desempeño de la automatización sobre una muestra heterogénea de APIs frente a una línea base histórica extraída de Jira (56 publicaciones entre 2024 y 2026). Los resultados demostraron una reducción del *time-to-market* de entre 88,16% y 99,75% en servicios de complejidad baja a media, disminuyendo los tiempos de publicación desde una mediana histórica de 69,28 horas a ciclos de 13 a 18 minutos totales. Se identificó como límite operativo la saturación por densidad sintáctica en contratos complejos, sentando bases para arquitecturas incrementales. El trabajo demuestra la viabilidad de descentralizar la exposición de servicios, liberando capacidad estratégica en los especialistas sin comprometer la gobernanza corporativa.

Palabras Clave: APIs, Cloud Computing, Automatización, Agentes de software, Gestión de APIs.

1 Introducción

El presente capítulo ofrece una visión global de la investigación, contextualizando los desafíos operativos y tecnológicos que motivan la búsqueda de un nuevo paradigma en la exposición de servicios financieros. A lo largo de esta sección, se articula la problemática originada por la centralización en la publicación de APIs, la propuesta de solución basada en automatización agéntica y la metodología experimental diseñada para su validación en un entorno corporativo real. Con ello, se sientan las bases conceptuales y metodológicas que guían el desarrollo de este documento.

1.1 Contexto y motivación

En la era digital actual, la industria bancaria enfrenta una necesidad imperativa de transformar sus ecosistemas tecnológicos para responder con agilidad a las demandas del mercado. La adopción de arquitecturas orientadas a servicios han posicionado a las APIs como los bloques de construcción fundamentales para habilitar la interoperabilidad, la integración de servicios de terceros y la creación de nuevos canales de negocio.

En este escenario, la capacidad de desplegar servicios de forma rápida, estandarizada y segura se ha convertido en un factor determinante para la competitividad institucional. Sin embargo, la creciente complejidad en la infraestructura *cloud* y las estrictas regulaciones de seguridad del sector financiero imponen desafíos significativos en el proceso de desarrollo. La motivación de este trabajo surge de la oportunidad de integrar técnicas emergentes de automatización asistida por agentes de software para optimizar el ciclo de vida de creación de APIs, reduciendo el esfuerzo manual y permitiendo que la organización adopte un modelo de escalabilidad eficiente y sostenible en el tiempo.

1.2 Definición del problema.

En una institución bancaria chilena de gran escala, se conformó un equipo especializado para habilitar iniciativas de *Open Banking* mediante la exposición de APIs en entornos *cloud*. Con el tiempo, la estrategia digital evolucionó hacia un consumo intensivo de información externa para optimizar procesos como el cálculo de rentas, análisis de riesgo y servicios para el segmento PyME. Sin embargo, persiste la visión organizacional del equipo de *Open Banking* como único responsable de la construcción de APIs, centralizando requerimientos de diversas áreas de negocio.

Esta concentración del conocimiento técnico genera efectos adversos. Primero, un elevado costo de oportunidad, pues los especialistas postergan actividades de alto valor estratégico para atender tareas operativas. Segundo, la canalización de solicitudes hacia los backlogs del equipo degrada el *time-to-market* de las soluciones solicitantes, generando fricción operativa y deteriorando la percepción de agilidad institucional.

Como resultado, el equipo actúa como un cuello de botella debido a la ausencia de un modelo que permita descentralizar y estandarizar la creación de APIs bajo los estándares de seguridad y gobierno exigidos. El problema radica en la dependencia de un conocimiento experto concentrado, lo que dificulta la escalabilidad. Surge así la necesidad de mecanismos que asistan y sistematicen la creación de APIs, habilitando a otros equipos a participar de este proceso de forma autónoma sin comprometer la integridad institucional.

1.3 Propuesta de solución

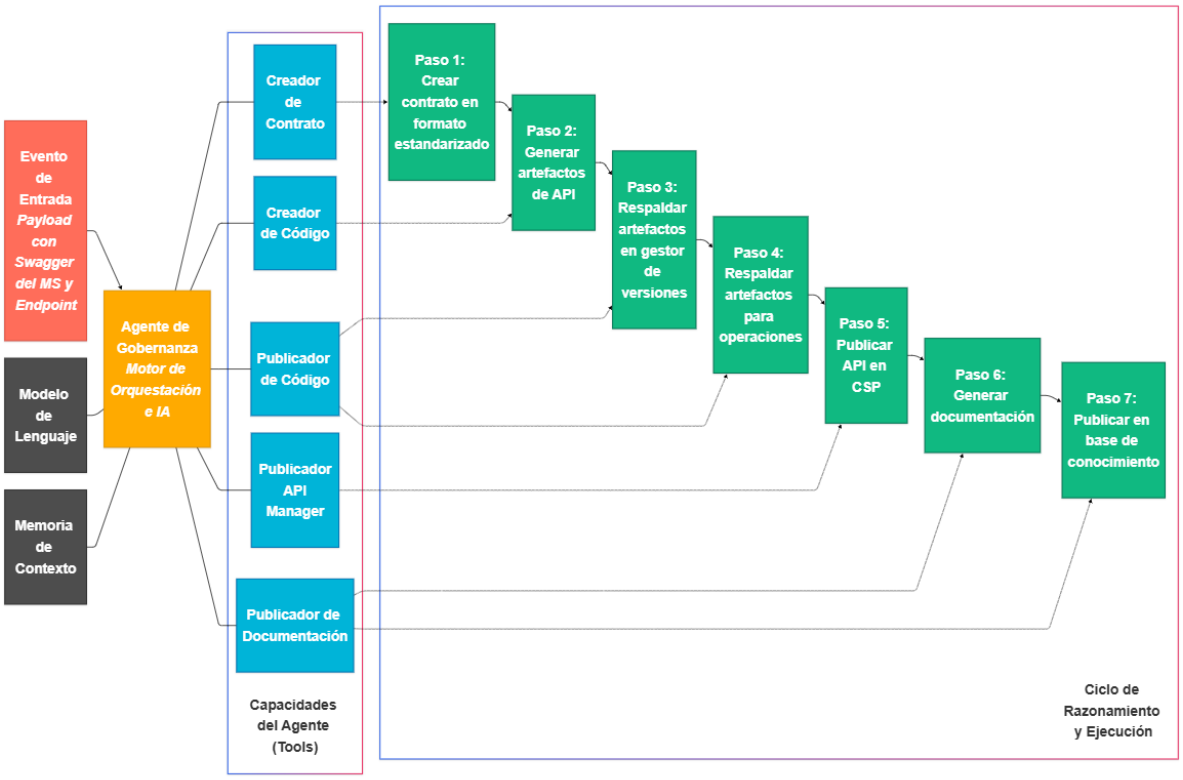


Figura 1. Propuesta de creación de APIs con agentes

Para abordar la problemática descrita, se propone un modelo asistido por agentes para la creación de APIs en entornos *cloud*, orquestado mediante una plataforma de automatización de flujos que estandariza la construcción de servicios. El proceso parte de microservicios previamente desplegados en contenedores, desde los cuales se extrae la especificación OpenAPI/Swagger. Este contrato inicial es procesado por un flujo que

aplica criterios de transformación técnica, incluyendo la normalización de nombres y la traducción de definiciones de español a inglés para alinearlas con las convenciones institucionales.

Una vez obtenida la especificación adaptada, el modelo automatiza la creación de la API en la plataforma de gestión institucional. Para efectos de esta prueba de concepto, y con el objetivo de acotar la complejidad operativa centrando el análisis en la generación inteligente del contrato, políticas XML y su aprovisionamiento en Azure APIM, los artefactos resultantes son resguardados en un almacenamiento *cloud* estructurado (Google Drive) como repositorio de artefactos temporal, sentando las bases para su posterior integración con un repositorio de código fuente en una fase de producción. Paralelamente, el sistema integra dinámicamente políticas de seguridad y gobierno desde un repositorio centralizado, asociándolas a los *endpoints* según sus requerimientos específicos.

El flujo concluye con la generación asistida del código necesario para consumir los microservicios subyacentes, incorporando validaciones y manejo de errores bajo lineamientos establecidos. Toda esta coordinación es ejecutada por agentes de software que operan bajo reglas explícitas, permitiendo que el conocimiento experto se traslade a un modelo reusable. Esto reduce la carga operativa del equipo especialista y habilita a otras células de desarrollo para avanzar con autonomía, manteniendo la integridad técnica y el gobierno del banco.

1.4 Hipótesis y metodología de validación

1.4.1 Hipótesis de investigación

En el marco de la transformación digital de la industria bancaria, la eficiencia operativa en la exposición de servicios resulta crítica para sostener la agilidad del negocio. No obstante, los procesos tradicionales de gobierno, estandarización y provisión de APIs suelen estar altamente centralizados en células de especialistas, generando fricción operativa y postergando el despliegue de soluciones.

Para abordar esta problemática, se formula la siguiente hipótesis principal de investigación:
La implementación de un modelo asistido por agentes de software y flujos orquestados para la creación, gobierno y publicación de APIs en entornos *cloud* reduce de manera sustancial el tiempo total de ciclo (*time-to-market*) requerido para poner un servicio a disposición de la organización, en comparación con el procedimiento tradicional ejecutado de forma manual por un desarrollador especializado.

Esta hipótesis sustenta que la automatización híbrida, que combina el razonamiento semántico mediante modelos de lenguaje con la ejecución determinista de reglas corporativas, permite eliminar las etapas manuales repetitivas y propenso a errores, mitigando el cuello de botella estructural que afecta a los equipos encargados de la exposición de servicios en API Management.

1.4.2 Variables de estudio

Para validar empíricamente la hipótesis formulada y evaluar de forma rigurosa el desempeño del modelo propuesto en un entorno corporativo real, la investigación se estructura a través de la operacionalización de las siguientes variables de estudio.

- Variable independiente:
    - Estrategia de creación y despliegue de APIs**:** Corresponde al paradigma o método utilizado para llevar a cabo el ciclo de vida de publicación del servicio. Esta variable se evalúa en dos escenarios discretos:
        1. Enfoque tradicional/manual*:* Proceso histórico en el que un desarrollador especializado realiza manualmente la traducción de contratos OpenAPI/Swagger, la redacción de políticas XML de Azure API Management, el aprovisionamiento en el gateway y la redacción de la documentación.
        2. Enfoque asistido por agentes: Proceso automatizado en el cual un pipeline orquestado en n8n coordina agentes basados en LLMs e instrucciones deterministas

para ejecutar de forma secuencial y paralela la estandarización, inyección de políticas, despliegue en Azure APIM y documentación.

- Variable dependiente:
  - Tiempo total del ciclo de publicación $(time - to - market)$: Cuantifica el tiempo transcurrido desde la recepción de la especificación OpenAPI/Swagger base generada por un microservicio backend hasta la disponibilidad efectiva del servicio, totalmente configurado, gobernado y documentado en la plataforma institucional. En el modelo automatizado propuesto, esta variable se descompone cuantitativamente en tres componentes temporales:

$$\text{Tiempo Total} = T_{pa} + T_c$$

Donde T_{pa} representa el tiempo de procesamiento automatizado e ininterrumpido del pipeline agéntico, y T_c cuantifica el tiempo de corrección o ajuste manual que pudiese requerir un operador humano tras la ejecución del flujo.

1.4.3 Metodología de validación

La validación de la hipótesis se llevó a cabo mediante un diseño experimental de carácter cuantitativo, empírico y comparativo, aplicado sobre un caso de estudio real en una institución del sector financiero chileno.

El diseño metodológico se articula en las siguientes fases interconectadas:

1. Construcción de la línea base histórica: Se realizó un análisis cuantitativo descriptivo a partir de los datos históricos registrados en la herramienta de gestión de flujo de trabajo Jira entre los años 2024 y 2026. Se extraerán las marcas de tiempo reales de las solicitudes de publicación de APIs procesadas mediante el enfoque manual tradicional (T_m), estableciendo métricas de tendencia central (mediana y promedio) y dispersión (percentil 80) para definir el comportamiento operativo del proceso previo.
2. Selección del *dataset* de prueba: Se seleccionó una muestra representativa de APIs institucionales que abarque un abanico heterogéneo de complejidad técnica. Dicha heterogeneidad se categorizó según la densidad sintáctica del contrato OpenAPI/Swagger, evaluando parámetros como el número de *endpoints* expuestos, la cantidad de atributos de entrada/salida y la profundidad de las estructuras de datos.
3. Ejecución experimental del pipeline agéntico: Se sometieron los contratos OpenAPI/Swagger de la muestra seleccionada al flujo de automatización orquestado en n8n. Durante la ejecución, se registraron de forma automatizada los tiempos de procesamiento del pipeline (T_{pa}), abarcando la inferencia de los agentes, la generación de políticas XML, el aprovisionamiento en Azure APIM, el resguardo de artefactos y la publicación de la documentación técnica en Confluence.
4. Evaluación de la intervención humana (T_c): Finalizada la ejecución automatizada, un especialista auditó el estado de los recursos desplegados en la infraestructura *cloud* para identificar eventuales discrepancias sintácticas, de nombrado o de configuración de infraestructura. Se cronometró de forma rigurosa el tiempo dedicado a realizar cualquier ajuste manual necesario (T_c) para dejar el servicio en un estado 100% apto para producción.
5. Análisis comparativo e inferencia: Se comparó estadísticamente los tiempos totales obtenidos mediante la propuesta ($T_{pa} + T_c$) frente a la línea base manual histórica (T_m). A partir de este contraste, se calculó los porcentajes de reducción del *time-to-market* y se evaluó los límites operativos del modelo en función de la complejidad de los contratos, permitiendo confirmar o refutar la hipótesis planteada.

1.5 Estructura del documento.

El presente informe de tesina se encuentra estructurado en 5 capítulos principales, los cuales se describen a continuación.

- Capítulo 1 – Introducción: En este capítulo se presenta el contexto y la motivación del trabajo en la industria bancaria, la definición del problema asociado a los cuellos de botella en la publicación de

APIs, la propuesta de solución basada en agentes de software, la hipótesis de investigación junto a las variables de estudio y la metodología para su validación, finalizando con la organización del informe.

- Capítulo 2 – Marco conceptual: Expone los fundamentos teóricos y tecnológicos que sustentan el diseño de la solución. Se abordan conceptos de arquitecturas orientadas a servicios, gestión del ciclo de vida de APIs, computación en la nube y sistemas agénticos, junto a una revisión del estado del arte y la descripción de las herramientas tecnológicas utilizadas.
- Capítulo 3 – Propuesta de solución: Describe en detalle la arquitectura general del sistema impulsado por agentes de software, justificando el patrón orquestador-trabajador, la configuración agéntica con GPT-4o en n8n y los subflujos especializados para la traducción de contratos, generación de políticas XML, persistencia de artefactos, despliegue en Azure APIM y publicación automatizada de documentación.
- Capítulo 4 – Experimentación y análisis de resultados: Detalla el diseño experimental cuantitativo, la definición de la línea base histórica obtenida desde Jira y la ejecución de pruebas sobre un *dataset* heterogéneo de APIs. Asimismo, se presentan los resultados comparativos de *time-to-market* y se realiza la discusión de los hallazgos, incluyendo la causa raíz de las intervenciones humanas y el análisis del mecanismo de falla en escenarios de alta complejidad.
- Capítulo 5 – Conclusiones y trabajos futuros: Sintetiza las conclusiones generales del trabajo, validando la hipótesis planteada y evaluando el impacto del modelo de autoservicio en la organización. Finalmente, se proponen las líneas de trabajo futuro para la evolución de la arquitectura.

2 Marco conceptual

El presente capítulo expone los fundamentos conceptuales, científicos y tecnológicos que sustentan el diseño e implementación del modelo propuesto. Se aborda en primer lugar el marco teórico, revisando los paradigmas de arquitectura de software, gestión de APIs y sistemas agénticos. Posteriormente, se examina el estado del arte en automatización asistida por agentes. Finalmente, se detalla el marco tecnológico, describiendo las herramientas sobre las cuales se construyó el prototipo funcional.

2.1 Marco teórico

El sustento teórico de este trabajo integra conceptos de diversas áreas de las tecnologías de la información para justificar la solución propuesta. Los ejes fundamentales comprenden arquitecturas orientadas a servicios y APIs, computación en la nube, automatización de procesos de desarrollo y sistemas basados en agentes.

2.1.1 Arquitecturas orientadas a servicios y APIs

Las APIs constituyen el conjunto de especificaciones que permiten la interacción controlada entre componentes de software, actuando como el mecanismo central de integración en sistemas heterogéneos para favorecer el desacoplamiento [1]. Bajo el paradigma de microservicios, estas interfaces operan como contratos explícitos entre componentes independientes, donde la definición formal de los recursos y operaciones es fundamental para asegurar la mantenibilidad y evolución del ecosistema tecnológico [2]. En este ámbito, la especificación OpenAPI se ha consolidado como el estándar para describir APIs de forma independiente de su implementación, permitiendo la validación de contratos y facilitando la integración técnica entre equipos [3].

2.1.2 Computación en la nube y servicios PaaS

La computación en la nube se define como un modelo que permite el acceso bajo demanda a un conjunto compartido de recursos computacionales configurables, los cuales pueden ser aprovisionados con un esfuerzo mínimo de gestión [4]. Bajo este paradigma, los proveedores ofrecen servicios gestionados que abstraen la complejidad operativa de la infraestructura, permitiendo que los equipos se enfoquen en la lógica de negocio

mediante el uso de *gateways* de APIs, mecanismos de autenticación y monitoreo [5]. No obstante, este enfoque introduce desafíos críticos en términos de configuración, seguridad y gobierno, especialmente en organizaciones con altos requerimientos regulatorios donde la estandarización del despliegue resulta indispensable para mitigar riesgos operativos.

2.1.3 Gestión del ciclo de vida de APIs

La gestión del ciclo de vida de las APIs comprende actividades críticas desde el diseño y la implementación hasta el monitoreo y retiro de los servicios[6]. Una gestión deficiente puede generar inconsistencias técnicas, duplicación de funciones y rupturas de compatibilidad con los consumidores [7]. Si bien las plataformas *de API management* centralizan el control de acceso, la aplicación de políticas y la documentación, estas suelen enfocarse en la administración de servicios ya existentes, omitiendo la estandarización del proceso previo de diseño y construcción técnica [8]. Esta brecha justifica la necesidad de modelos que automaticen la fase de creación para asegurar que el servicio nazca alineado a los estándares de gobierno.

2.1.4 Automatización de procesos de desarrollo

La automatización de procesos de desarrollo busca mitigar la intervención manual en tareas repetitivas y propensas a errores para aumentar la velocidad y consistencia en la entrega de soluciones. En la ingeniería de software, estas prácticas se materializan en enfoques de integración y entrega continua (CI/CD), así como en la automatización de infraestructura [9]. Aplicada a la creación de APIs, permite estructurar flujos reproducibles para la generación de contratos y provisión de recursos. No obstante, los mecanismos tradicionales suelen limitarse a la ejecución secuencial de pasos predefinidos, omitiendo la asistencia en la toma de decisiones técnicas recurrentes, brecha que este trabajo propone cubrir mediante el uso de agentes.

2.1.5 Sistemas basados en agentes

Los sistemas basados en agentes utilizan entidades capaces de percibir su entorno y actuar sobre él para cumplir objetivos específicos. Un agente se define por su autonomía, capacidad de reacción y comportamiento orientado a metas [10]. En la ingeniería de software, se emplean para apoyar tareas complejas mediante la delegación de responsabilidades acotadas, asistiendo procesos humanos sin reemplazarlos por completo [11]. En este trabajo, el concepto se adopta desde una perspectiva práctica y controlada, donde los agentes actúan como asistentes en la creación de APIs, con un ámbito de acción explícitamente restringido para asegurar la consistencia y el cumplimiento de las reglas institucionales

2.2 Estado del arte

La literatura y las soluciones industriales actuales presentan dos líneas de trabajo fundamentales para esta investigación. En el ámbito académico, se propone un modelo basado en sistemas multiagente que transforma especificaciones OpenAPI en servicios funcionales mediante un lazo iterativo de generación y refinamiento automático de código [12]. Este enfoque demuestra la viabilidad de utilizar modelos de lenguaje para sintetizar artefactos ejecutables de forma reproducible, aunque su aplicación se limita a escenarios controlados sin abordar la gobernanza corporativa compleja.

Desde la perspectiva industrial, la plataforma IBM API Connect integra capacidades de inteligencia artificial a través de *API Agent,* un marco agéntico diseñado para automatizar el ciclo de vida de las APIs [13]. Esta herramienta permite generar especificaciones, validar políticas de gobierno y crear código de backend a partir de lenguaje natural o contratos formales, facilitando el despliegue automatizado en entornos de ejecución [14]. A diferencia del modelo académico, esta solución ofrece una interfaz conversacional que guía al desarrollador en tareas secuenciales, desde el diseño hasta las pruebas.

En conjunto, estos antecedentes confirman que la automatización asistida por agentes es una tendencia consolidada. Mientras la vertiente académica valida la generación de código a partir de especificaciones, la

solución industrial provee un ecosistema integrado con mecanismos de control operacional, situando estas tecnologías en un espectro que abarca desde pruebas de concepto hasta aplicaciones empresariales de gran escala.

2.3 Marco tecnológico

Para concretar la arquitectura conceptual y operativa del modelo propuesto, es necesario seleccionar herramientas tecnológicas que permitan orquestar flujos de trabajo automatizados y gestionar la exposición de servicios en un entorno corporativo. En esta sección se describen los componentes tecnológicos fundamentales que soportan la solución implementada, abordando tanto la plataforma utilizada para la orquestación agéntica de procesos como el servicio administrado en la nube encargado de la gestión, seguridad y publicación de las APIs.

2.3.1 Plataforma de orquestación n8n

N8n es una plataforma de automatización de flujos de trabajo basada en nodos y de código abierto, diseñada para conectar aplicaciones, servicios y sistemas a través de APIs REST o *webhooks* [15]. A diferencia de las herramientas tradicionales de automatización de procesos de negocio, n8n permite la ejecución de código personalizado en JavaScript o Python directamente en los nodos del flujo, lo que facilita la manipulación fina de datos y la orquestación de tareas complejas [16].

En el contexto de la integración de sistemas agénticos, n8n ofrece capacidades para estructurar flujos interactivos mediante la orquestación de modelos de lenguaje (LLMs), memorias de conversación y herramientas de ejecución [17]. Esta flexibilidad permite construir tuberías de automatización deterministas, garantizando la trazabilidad de los datos, la gestión de errores intermedios y el cumplimiento de reglas institucionales sin perder la extensibilidad que entrega el código nativo [15].

2.3.2 Azure Api *Management*

Azure API *Management* (APIM) es una plataforma como servicio (PaaS) que permite abstraer, publicar, asegurar y gestionar el ciclo de vida completo de las APIs expuestas en la nube de Microsoft Azure. La plataforma actúa como un punto de entrada centralizado, o API *gateway*, que desacopla los clientes de las aplicaciones *backend*, interceptando las peticiones HTTP y aplicando políticas de transformación, enrutamiento y seguridad de forma consistente [18].

Entre sus capacidades clave destaca el uso de políticas XML/CSHTML declarativas que se ejecutan en las distintas etapas del procesamiento de una petición: entrada, *backend*, salida y error. Estas políticas permiten implementar validaciones de tokens JWT, *rate limiting*, cuotas de consumo, almacenamiento en caché y transformaciones de *payload*s sin necesidad de modificar el código subyacente del microservicio [19]. En el contexto del modelo propuesto, APIM actúa como la capa de ejecución donde los agentes aplican dinámicamente las reglas de gobierno y seguridad institucionales.

3 Propuesta de solución

3.1 Arquitectura general de la solución

La propuesta de solución se concreta mediante una arquitectura orientada a eventos e impulsada por agentes de software, diseñada para automatizar de extremo a extremo el ciclo de preparación, gobierno, despliegue y documentación de APIs en Azure API *Management* (APIM). El sistema utiliza como motor de orquestación la plataforma n8n, desde la cual se coordinan flujos de trabajo principales y subflujos especializados que combinan la capacidad de razonamiento e inferencia de modelos de lenguaje con la precisión de nodos de procesamiento deterministas.

Como se ilustra en los diagramas posteriores, la solución está estructurada bajo un patrón orquestador-especialista. Un flujo central (ApiPublisherOrchestrator) actúa como punto de entrada capturando el contrato OpenAPI/Swagger original expuesto por los microservicios *backend*. A partir de esta entrada, el orquestador delega tareas específicas a tres herramientas agénticas (*tools*):

- Traducción y estandarización del contrato (*Swagger Translator Agent*): Adapta la especificación base a la nomenclatura y requisitos del estándar corporativo.
- Generación de políticas transversales (*AllOperations Policy Agent*): Aplica de forma determinista las reglas globales de seguridad, control de tráfico y encabezados a nivel de toda la API.
- Generación de políticas granulares (*Operation Policy Agent*): Diseña mediante inferencia y posprocesamiento en código las reglas específicas para cada *endpoint* u operación HTTP.

Una vez consolidados los artefactos en memoria, la arquitectura habilita dos ramas de ejecución paralela sincronizadas por un nodo coordinador. La primera garantiza el resguardo y trazabilidad digital, generando los paquetes binarios necesarios tanto para la integración con repositorios de código fuente como para la gestión formal del paso entre ambientes operacionales. La segunda efectúa el aprovisionamiento automatizado en la infraestructura *cloud*, invocando la API de gestión de Azure para instanciar la API, registrar sus *namedValues* y aplicar el conjunto de políticas XML.

Finalmente, el pipeline concluye con la actualización automática de la base de conocimiento institucional, donde un agente sintetiza la información técnica y la publica en la plataforma colaborativa de la organización. Esta arquitectura modular no solo elimina las tareas manuales y la fricción operativa entre los equipos de desarrollo y la célula de API Management, sino que garantiza un cumplimiento estricto y auditable de los estándares de gobierno desde el momento del despliegue.

3.1.1 Patrón orquestador-trabajador

La arquitectura del sistema desacopla el proceso de generación e integración de APIs mediante la implementación del patrón orquestador-trabajador. En este esquema, un componente central, el orquestador, asume la responsabilidad exclusiva de gestionar el estado del flujo, coordinar el orden de ejecución y transferir el contexto entre las distintas fases, mientras que un conjunto de agentes y módulos especializados, los trabajadores, ejecutan tareas delimitadas dentro de su dominio especifico.

La adopción de este patrón obedece a tres razones fundamentales de ingeniería de software y diseño de sistemas agénticos. Su estructura de funcionamiento se esquematiza en la Figura 2.

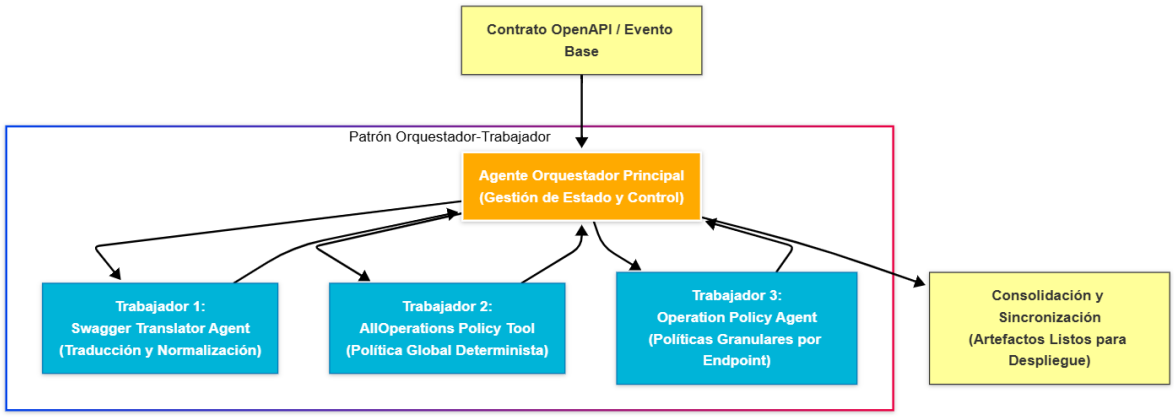


Figura 2 Esquema conceptual del flujo de control bajo el patrón orquestador-trabajador

1. Reducción de la carga cognitiva y limitación de contexto de los LLMs: Intentar que un único modelo de lenguaje procese el contrato OpenAPI, traduzca la semántica, infiera configuraciones de seguridad, estructure las políticas XML y coordine las llamadas a la API de Azure generaría una sobrecarga en la ventana de contexto y en la capacidad de razonamiento del modelo. Al delegar

cada tarea a agentes especializados, trabajadores con *prompts* de sistema altamente acotados, se maximiza la precisión en la generación de código y se reduce drásticamente la tasa de alucinaciones o desviaciones del esquema JSON esperado.

2. Modelo hibrido entre razonamiento probabilístico y ejecución determinista: Las arquitecturas empresariales modernas no deben depender en su totalidad de inferencias estocásticas. El patrón orquestador-trabajador permite combinar estratégicamente agentes basados en inteligencia artificial para tareas de traducción semántica, comprensión contextual y adaptación de esquemas, donde se requiere razonamiento inductivo, con nodos deterministas basados en código ejecutable para operaciones críticas como el cálculo de rutas, formateo sintáctico de archivos, estructuración de peticiones HTTP y la inyección incondicional de políticas transversales de seguridad.
3. Mantenibilidad, modularidad y escalabilidad: El desacoplamiento del flujo facilita la evolución independiente de la solución. Si una regla de negocio en las políticas XML de Azure APIM cambia, o si se requiere actualizar la versión del modelo LLM utilizado por el agente de traducción, la modificación se limita exclusivamente al trabajador correspondiente sin alterar la lógica de gobierno ni los subflujos de integración del orquestador principal.

3.1.2 Ingesta de datos e interfaz *webhook*

En la concepción original de la arquitectura, se proyectó que el flujo orquestador consultara e inspeccionara de manera directa las especificaciones OpenAPI/Swagger expuestas directamente por los microservicios en sus entornos de ejecución. Sin embargo, dada la topología de red institucional y las estrictas políticas de ciberseguridad que rigen en el sector bancario, el entorno de despliegue de la plataforma de automatización n8n se encuentra lógicamente segregado de las zonas de red privadas donde residen los microservicios.

Con el propósito de superar esta restricción de conectividad sin comprometer el perímetro de seguridad ni requerir aperturas de red no homologadas, se adoptó un patrón desacoplado impulsado por eventos mediante un punto de entrada tipo *webhook*. Bajo este enfoque, la especificación OpenAPI base es enviada como *payload* en formato JSON hacia el pipeline agéntico. Esta decisión no solo resolvió la factibilidad técnica de la prueba de concepto, sino que estableció una interfaz agnóstica de integración que permite a posibles pipelines externos o a los propios desarrolladores activar el proceso de publicación de forma segura.

3.1.3 Modelo de activación a demanda

En el ciclo de vida de desarrollo institucional, no todos los microservicios construidos por las células requieren ser expuestos hacia clientes externos o aplicaciones de terceros a través del *gateway* de APIs. La decisión de publicar un *endpoint* en Azure API Management responde a un análisis de arquitectura, dominio de negocio y controles de seguridad que recae de forma soberana en la célula de desarrollo responsable del servicio.

Por este motivo, el diseño de la solución contempla un esquema de activación a demanda impulsado por eventos. En la etapa actual de implementación, el *webhook* expuesto por el orquestador en n8n es gatillado directamente por el desarrollador o referente técnico de la célula una vez que el microservicio ha sido validado y se ha determinado formalmente la necesidad de su exposición. Esta interacción se realiza enviando el contrato OpenAPI/Swagger como *payload* mediante una petición HTTP POST segura.

Este enfoque desacoplado otorga la flexibilidad necesaria para que la automatización respete la autonomía de las células, evitando sobrecargar la infraestructura de API *management* con servicios internos o temporales que no forman parte del catálogo institucional de APIs.

3.2 Configuración del motor de inferencia

Antes de detallar el flujo de ejecución del pipeline, es necesario definir la capa de inteligencia que sustenta la capacidad de razonamiento, traducción y generación de artefactos dentro del sistema. En la arquitectura propuesta, los agentes no operan como componentes de código estático, sino como nodos de orquestación en n8n impulsados por un modelo de lenguaje de gran escala.

3.2.1 Selección del modelo de lenguaje

La elección del modelo de lenguaje de gran escala o LLM para la ejecución de las tareas de inferencia, traducción de contratos OpenAPI y estructuración de políticas XML estuvo directamente determinada por las políticas de arquitectura de TI, gobernanza de datos y ciberseguridad vigentes en la institución bancaria.

La adopción de GPT-4o responde a los siguientes pilares de arquitectura *enterprise* y capacidades técnicas.

- Disponibilidad *enterprise* y cumplimiento normativo: El modelo GPT-4o se encontraba previamente aprovisionado, homologado y habilitado dentro de la suscripción institucional de Azure OpenAI Services del banco. Esta condición garantizó que el procesamiento de la información se llevara a cabo de manera confinada dentro del perímetro de seguridad del *tenant* corporativo, asegurando el cumplimiento normativo, el cifrado de datos en tránsito y reposo, y la garantía explícita de que los *payloads* empleados no son utilizadas para el entrenamiento de modelos públicos
- Soporte nativo para salidas estructuradas: GPT-4o ofrece soporte avanzado para el cumplimiento estricto de esquemas JSON. Esta característica técnica es crítica para la arquitectura propuesta, pues garantiza que las respuestas generadas por los agentes se ajusten con precisión matemática a las interfaces esperadas por el orquestador en n8n, reduciendo drásticamente los errores sintácticos de *parsing.*

3.2.2 Salida estructurada y límites sintácticos

La interacción entre el orquestador n8n y el modelo GPT-4o se rige bajo dos condicionantes de diseño fundamentales para garantizar la estabilidad de la infraestructura *cloud.*

- Inferencia estructurada mandatoria (*Structured Output Parser*): Para evitar que el LLM genere respuestas conversacionales o texto introductorio que corrompa la ejecución del pipeline, se acopló a cada nodo agéntico el componente *Structured Output Parser* de n8n. Este mecanismo fuerza a GPT-4o a responder ciñéndose estrictamente a un esquema JSON predefinido. De este modo, las descripciones traducidas y los bloques XML de políticas para Azure API *management* son entregados en un formato directamente interpretable por los nodos subsiguientes del flujo.
- Ventana de contexto vs. límite muestral de salida: Si bien GPT-4o dispone de una amplia ventana de contexto de 128,000 tokens para recibir especificaciones OpenAPI extensas y *prompts* de gobierno corporativo, la emisión de respuestas en una sola llamada está sujeta al límite estándar del proveedor de 4,096 tokens de salida. Esta especificación define el umbral máximo de capacidad sintáctica que el agente orquestador puede estructurar en una única respuesta, constituyendo una restricción técnica relevante para el procesamiento de contratos con elevada densidad de *endpoints* o metadatos.

3.2.3 Seguridad y prevención de inyección de *prompts*

La adopción de un modelo agéntico alimentado por especificaciones OpenAPI/Swagger externas mediante *webhooks* introduce vectores de riesgo de ciberseguridad que deben ser gestionados bajo un enfoque de seguridad por diseño y SecOps.

Al utilizar modelos de lenguaje de gran escala para procesar metadatos e instrucciones provenientes de descripciones textuales en los contratos OpenAPI, tales como los campos de descripción, resumen o ejemplo redactados por desarrolladores, existe el riesgo latente de ataques por inyección de *prompts* indirecta. En este tipo de vulnerabilidad, un actor malicioso podría manipular deliberadamente el texto dentro del contrato OpenAPI para alterarla lógica de razonamiento del agente, intentando omitir políticas de seguridad obligatorias en Azure APIM, por ejemplo, forzando la remoción de validaciones JWT.

Para mitigar estos riesgos en la arquitectura propuesta y garantizar que la automatización no introduzca brechas en el *gateway* corporativo, se implementaron las siguientes capas de contención:

- Encapsulamiento y salida estructurada (*Structured output parser*): el uso de *parsers* con esquemas JSON estrictos impide que el LLM devuelva comandos ejecutables fuera del esquema predefinido, acotando el espacio de respuesta de los agentes.

- Patrón híbrido determinista: la inyección de las políticas transversales de seguridad a nivel global (*Alloperations policy*) se delegó a módulos de código deterministas no expuestos a la inferencia del LLM. Con ello se asegura que las reglas normativas de ciberseguridad del banco se apliquen incondicionalmente, sin importar la entrada procesada.

3.3 Diagrama general del pipeline (Orquestador principal)

El flujo principal, denominado *ApiPublisher Orchestrator*, constituye el plano de control y coordinación del sistema de automatización agéntica. Su diseño responde a una arquitectura orientada a eventos e impulsada por el patrón orquestador-trabajador, asumiendo la responsabilidad exclusiva de gestionar el estado del proceso, mantener el contexto de la transacción, orquestar las llamadas a los agentes y herramientas especializadas y coordinar la ejecución concurrente de los procesos de despliegue y documentación.

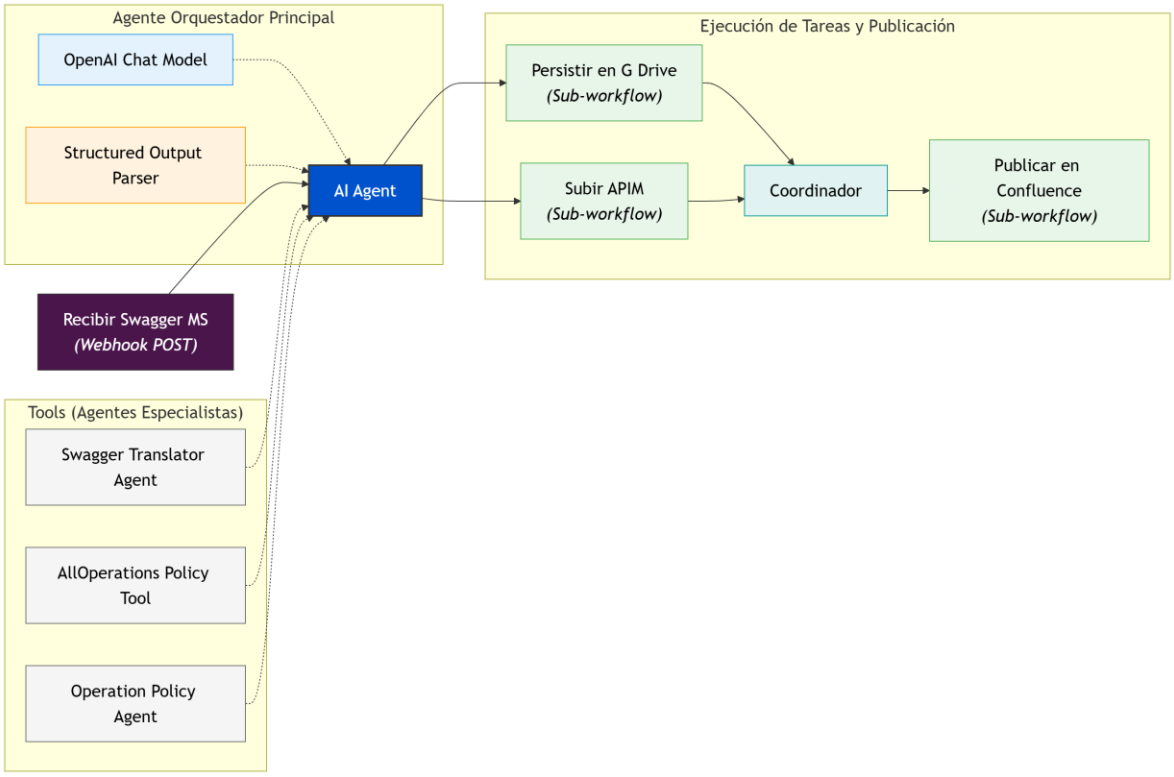


Figura 3 Flujo agente orquestador

El ciclo de vida completo de la publicación dentro del orquestador se descompone en tres fases de ejecución secuenciales e interconectadas.

- Fase de preparación y generación de artefactos: El pipeline se gatilla mediante un nodo de tipo *webhook* HTTP POST que recibe el documento OpenAPI/Swagger original expuesto por el microservicio. El orquestador toma este *payload* y ejecuta de manera secuencial e iterativa las tres herramientas especializadas.
    - *Swagger Translator Agent*: Normaliza el contrato original, traduce las definiciones semánticas al idioma inglés y adapta la nomenclatura a los estándares de nombrado corporativo en *PascallCase*.
    - *AllOperations Policy Tool*: Aplica de forma determinista, mediante código ejecutable, la plantilla XML de políticas transversales a nivel global (*All operations* de APIM).

- *Operation Policy Agent*: Analiza recursivamente cada *endpoint* del contrato y sintetiza las políticas XML de ámbito granular (*Operation* a nivel de APIM).

- Fase de persistencia y aprovisionamiento: Una vez que los artefactos (contrato traducido, políticas globales y políticas por operación) quedan validados y consolidados en la memoria local del orquestador, la ejecución se bifurca en dos ramas paralelas de ejecución.
  - Persistencia de Artefactos *(*subflujo Persistir en Google Drive*)*: Estructura la jerarquía de directorios y transforma las entidades en memoria a paquetes binarios (.json y .xml) como mecanismo de resguardo y trazabilidad.
  - Aprovisionamiento en Infraestructura (subflujo Subir APIM*)*: Consume la API REST de gestión de Azure API Management para aprovisionar formalmente la API, instanciar las variables de entorno, o *named values*, y aplicar la jerarquía de políticas XML.
- Fase de sincronización, gobierno y cierre: Ambas ramas paralelas convergen en un nodo de sincronización denominado "Coordinador", que actúa bajo el patrón de barrera. Una vez confirmada la respuesta exitosa de los procesos de almacenamiento y despliegue, el orquestador gatilla el subflujo "Publicar en Confluence". Un agente sintetizador traduce los metadatos técnicos acumulados a formato *HTML Storage Format* y los inyecta en la plataforma wiki corporativa, dando por finalizado el ciclo de publicación.

3.4 Componentes y subflujos especializados

A continuación, se profundiza en la arquitectura interna, diseño de prompts y transformaciones de datos de cada uno de los subflujos que componen la solución

3.4.1 Herramienta de traducción de OpenAPI/Swagger (*Swagger Translator Agent*)

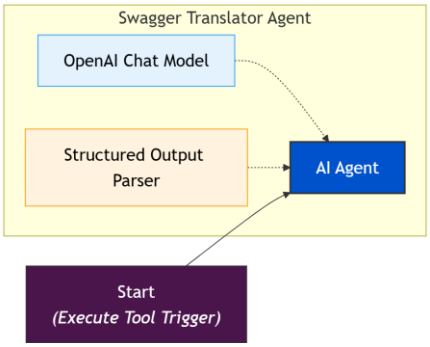


Figura 4 Subflujo *Swagger Translator Agent*

El subflujo *Swagger Translator Agent* actúa como una herramienta especializada invocada por el orquestador principal durante la fase inicial del procesamiento. Su objetivo fundamental es recibir el contrato OpenAPI/Swagger original expuesto por el microservicio y normalizarlo según las especificaciones y estándares definidos por la organización.

- Punto de entrada e ingesta (*Start / Execute Tool Trigger*): Recibe como entrada una estructura JSON que contiene el contrato OpenAPI/Swagger original sin procesar.
- Lógica del agente e instrucción de sistema (*AI Agent & system prompt*): El nodo agéntico está respaldado por el modelo GPT-4o. El *system prompt* del agente está estrictamente parametrizado para actuar como un auditor de contratos OpenAPI corporativo. Las directrices del *prompt* imponen:
  - Traducción semántica: Convertir descripciones, resúmenes, títulos, atributos y rutas de acceso del español al inglés técnico formal.
  - Normalización de nomenclatura*:* Transformar las claves de parámetros, atributos de peticiones y respuestas a la convención *PascalCase.*

- Preservación sintáctica*:* Prohibir explícitamente la alteración o eliminación de tipos de datos (*string*, *integer*, *boolean*, etc), estructuras de arreglos u objetos anidados, garantizando la integridad funcional del microservicio.

- Mecanismo de salida estructurada (*Structured Output Parser*): Para evitar inconsistencias de formato o respuestas conversacionales que interrumpan el pipeline, se acopla un analizador sintáctico de esquema estricto (JSON *Schema*). La respuesta debe retornar obligatoriamente un único objeto JSON que contiene el contrato OpenAPI traducido y sintácticamente válido.

3.4.2 Herramienta de generación de política global (*AllOperations Policy Tool*)

En arquitecturas empresariales y bancarias, las reglas de gobierno, control de acceso y resiliencia a nivel transversal, tales como la gestión de límites de tasa, el manejo estandarizado de errores, la conexión a servicios centralizados de autorización y la validación de peticiones, no deben quedar sujetas a modelos probabilísticos que puedan introducir alucinaciones o descalces de seguridad. Por este motivo, la herramienta *AllOperations Policy Tool* adopta un paradigma de ejecución determinista basado en código y plantillas estructuradas.

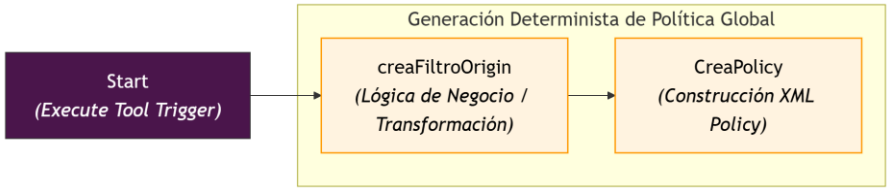


Figura 5 Subflujo AllOperations Policy Tool

- Ingesta y activación (Start/*Execute Tool Trigger*): Captura el contexto de la API desde el orquestador principal, incluyendo la definición de rutas y verbos HTTP expuestos por el microservicio *backend*.
- Nodo evaluador (creaFiltroOrigin): Este nodo analiza la topología de *endpoints* del microservicio y genera dinámicamente el código de control de acceso según las reglas de negocio de la institución. Su responsabilidad principal es mapear y diferenciar qué rutas y métodos HTTP específicos requieren filtrado por dirección IP origen versus filtrado por dominio. Por ejemplo, en un mismo recurso, el nodo evaluador puede establecer que las peticiones de lectura GET sean validadas mediante la cabecera *Origin*, permitiendo el acceso desde aplicaciones web/*frontends* autorizados, mientras que las operaciones de escritura o de alta criticidad: POST, PUT, DELETE, sean restringidas estrictamente por listas de control de acceso basadas en IP, garantizando que solo provengan de segmentos de red o *gateways* internos seguros.
- Nodo de construcción XML (CreaPolicy): Este nodo alberga la plantilla canónica inmutable de la política global *AllOperations* de la API. En ella se definen las directrices arquitectónicas transversales de la organización, tales como:
  - Mapeo y gestión estandarizada de errores*:* Definición del bloque <*on-error*> para transformar fallos del *backend* o de la pasarela en esquemas de respuesta homogeneizados.
  - Límites de tasa y consumo o *rate limiting:* Configuración de cuotas máximas de peticiones por segundo/minuto por clave de suscripción para mitigar ataques de denegación de servicio (DoS).
  - Conexión a servicios de seguridad centralizados*:* Invocación a los microservicios internos que proveen dinámicamente las tablas de direcciones IP y dominios permitidos.

  El nodo CreaPolicy toma el fragmento de código contextual generado por creaFiltroOrigin y lo inyecta dentro del bloque <*inbound*> de la plantilla global, empaquetando el artefacto XML final listo para ser desplegado en el plano de control de Azure APIM.

3.4.3 Herramienta de políticas por operación (*Operation Policy Agent*)

A diferencia de las políticas globales transversales, las políticas de ámbito operacional requieren adaptar y transformar la estructura del tráfico en tiempo de ejecución para garantizar la interoperabilidad entre las interfaces expuestas hacia el exterior en idioma inglés y las implementaciones internas de los microservicios de *backend* en idioma español.

El subflujo *Operation Policy Agent* opera de manera iterativa, se invoca individualmente por cada una de las operaciones definidas en la especificación Swagger/OpenAPI. Su diseño combina el razonamiento agéntico para la traducción de esquemas de datos con una plantilla determinista que encapsula la seguridad y autenticación.

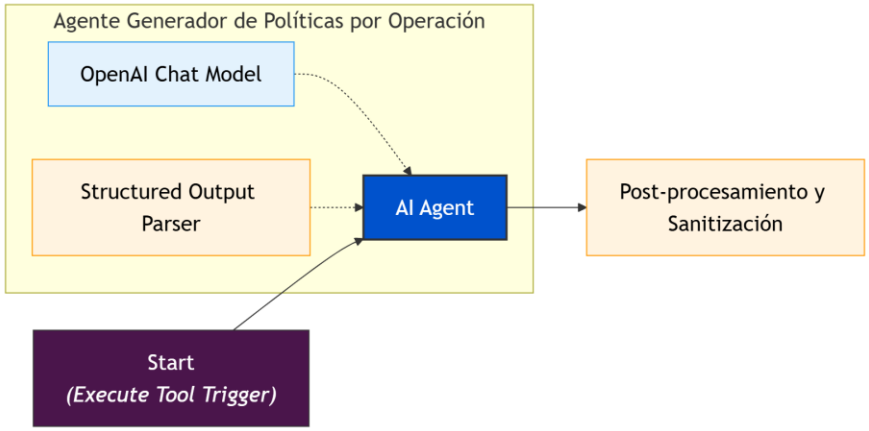


Figura 6 Subflujo *Operation Policy Agent*

Dinámica de funcionamiento del subflujo

- Ingesta y parámetros de entrada (*Start/Execute Tool Trigger*): Por cada *endpoint* extraído del contrato OpenAPI, el subflujo recibe como carga útil los esquemas JSON de petición y respuesta en ambos dominios lingüísticos: la definición pública estandarizada en inglés y la estructura interna del microservicio en español.
- Transformación semántica e inferencia de esquemas (*AI Agent*)**:** El nodo agéntico analiza las equivalencias de los campos y asume la responsabilidad de sintetizar dos transformaciones clave:
  - Mapeo de entrada (inglés a español): Genera código de expresiones de Azure APIM en C# para transformar el cuerpo y parámetros de la petición recibida en inglés hacia el formato en español esperado por el microservicio *backend*.
  - Mapeo de salida (español a inglés): Diseña una plantilla de transformación en lenguaje *Liquid* para interceptar la respuesta devuelta por el microservicio en español y traducirla a la estructura canónica en inglés expuesta al cliente.
  - Estandarización de tipos de datos: En ambas transformaciones, el agente aplica explícitamente las reglas de formato institucional para la representación de fechas en ISO 8601 y la precisión de valores numéricos y monetarios.
- Ensamblaje y posprocesamiento determinista: El nodo de posprocesamiento toma el bloque de código C# y la plantilla *Liquid* sintetizados por el modelo de IA y los inyecta dentro de una plantilla de política XML predefinida. Esta plantilla estructural ya resuelve y abstrae de forma inmutable los aspectos de infraestructura y seguridad, tales como autenticación y autorización hacia el microservicio *backend* mediante la inyección de tokens *Bearer* y cabeceras específicas.
- Entregable del subflujo: Como resultado de este procesamiento, el subflujo retorna una política XML completa, sintácticamente válida y lista para ser aplicada a la operación específica en Azure APIM.

3.4.4 Subflujo de persistencia y resguardo de artefactos (Persistir en Google Drive)

Para garantizar la auditabilidad, la trazabilidad operativa y dar soporte a futuros modelos de despliegue basados en principios *GitOps*, el subflujo de persistencia actúa como la capa de resguardo de todos los artefactos técnicos generados durante el ciclo de orquestación. Su función principal es empaquetar, convertir y almacenar la totalidad de las especificaciones y políticas XML/JSON en un repositorio persistente estructurado.



Figura 7 Persistir en Google Drive

Secuencia y lógica de procesamiento

- Jerarquía y convención de directorios: El subflujo evalúa el contexto de la transacción extraído del orquestador principal. Utilizando la API del servicio de almacenamiento, crea de forma automatizada una estructura jerárquica de carpetas siguiendo la convención de nombres: {/APIs_Publicadas/{Nombre_API\}/{Fecha_de_generación}
  Donde Fecha_de_generación corresponde al timbre de tiempo en formato estandarizado (YYYY-MM-DD-HH-mm) registrado en el momento de la ejecución.
- Generación e identificación de artefactos binarios: En memoria, el subflujo transforma las estructuras de texto y objetos codificados a los formatos binarios UTF-8 application/json y application/xml, generando de manera individualizada la siguiente suite de archivos:
  - Contrato OpenAPI traducido ({Nombre_API_EN}.json): El documento Swagger/OpenAPI normalizado y traducido al idioma inglés por el *Swagger Translator Agent*.
  - Política global (AllOperationsPolicy.xml): El artefacto XML transversal generado por la herramienta *AllOperations Policy Tool*, que integra las reglas globales de *rate limit*, manejo de errores y lista de accesos por IP/Origen.
  - Políticas de operación atomizadas ({verbohttp}{path_sin_barras}.xml): Un archivo XML individual por cada *endpoint* o ruta expuesta en el Swagger, por ejemplo: getAccounts.xml, postTransfers.xml. Cada archivo contiene las transformaciones específicas de e integración de autenticación hacia el microservicio *backend*.
  - Configuración de variables (namedValues.json): Un archivo JSON consolidado que registra el catálogo de claves y referencias de las variables de entorno e integración requeridos por las políticas de Azure APIM para su correcto aprovisionamiento en el *gateway*.
- Carga y resguardo en el repositorio**:** A través del conector de Google Drive en n8n, se ejecutan peticiones de carga masiva y paralela asignando los metadatos correspondientes.

Si bien el paradigma DevOps exige la trazabilidad mediante GitOps sobre repositorios de código fuente como GitHub o similares, para la implementación y validación de esta prueba de concepto se optó por un esquema de persistencia intermedia en Google Drive. Esta decisión permitió acotar la complejidad operacional y de autenticación de la infraestructura de pruebas, aislando el experimento para evaluar con precisión la capacidad del pipeline agéntico en la estructuración, generación y resguardo organizado de los paquetes binarios

3.4.5 Subflujo de despliegue en APIM (Subir APIM)

El subflujo Subir APIM constituye el núcleo de ejecución de la solución. Su función es interactuar directamente con los *endpoints* de gestión de Azure API Management para instanciar la API, asociar sus reglas de gobierno y dejar el servicio completamente operativo en el *gateway* de la institución.

La ejecución se realiza de manera secuencial y estructurada para garantizar la integridad referencial de los recursos dentro de Azure.

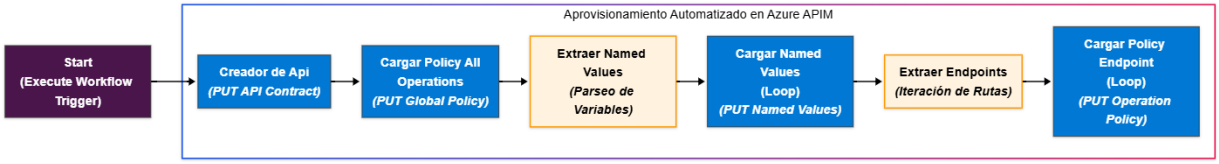


Figura 8 Subflujo Subir APIM

Secuencia de Aprovisionamiento:

- Punto de entrada (*Start/Executed Workflow Trigger*): Recibe desde el orquestador los esquemas OpenAPI validados, las variables de entorno y las definiciones XML de las políticas globales y por operación.
- Creación del recurso API (Creador de Api): Ejecuta una petición HTTP PUT contra el *endpoint* de Azure APIM para importar la definición OpenAPI. Este paso crea el recurso principal de la API en el *gateway*, registrando sus rutas y esquemas base.
- Inyección de política global (Cargar *Policy All Operations*): Aplica el documento XML de políticas transversales generado previamente. Con esto se aseguran comportamientos globales de la API, como esquemas de autenticación o *rate limit* general.
- Gestión e inyección de variables (Extraer *Named Values* / Cargar *Named Values*): Mediante un bucle iterativo registra o actualiza los pares definidos de clave-valor dentro de la sección de variables de ambiente en la instancia de Azure APIM.
- Configuración de políticas granulares: Iterativamente, el flujo recorre la lista de operaciones registradas en el contrato Swagger. Para cada *endpoint*, ejecuta una llamada PUT inyectando su correspondiente archivo XML de política de operación.

3.4.6 Subflujo de generación y publicación de documentación (Publicar en Confluence)

La etapa final del pipeline automatizado asegura la actualización del repositorio central de conocimiento de la institución. El subflujo Publicar en Confluence recibe los insumos resultantes de los procesos previos para redactar una documentación técnica clara, estructurada y en formato HTML/*Storage Format* de Confluence, eliminando la carga de trabajo manual que habitualmente recae sobre los equipos de desarrollo.

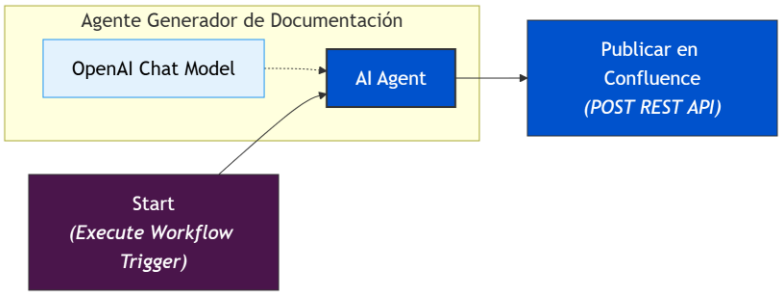


Figura 9 Subflujo Publicar en Confluence

Estructura y secuencia de procesamiento:

- Ingesta de contexto: (*Start/Execute Workflow Trigger*): Recibe como entrada el conjunto de datos enriquecidos durante la ejecución previa del pipeline: contrato Swagger traducido, *endpoints*, políticas aplicadas y metadatos de despliegue.
- Agente redactor de documentación (*AI Agent* + *OpenAI Chat Model*): Utiliza el modelo de lenguaje para sintetizar la información de la API y convertirla en una página de documentación técnica

formal. El agente estructura la información incluyendo descripciones de negocio, parámetros de entrada y salida, códigos de respuesta HTTP y detalles de las políticas de seguridad aplicadas.

- Publicación automatizada (Publicar en Confluence): Ejecuta una llamada vía POST a la API REST de Confluence, para crear o actualizar dinámicamente la página correspondiente a la API en el espacio de trabajo de la organización, completando así el ciclo de gobierno y visibilidad del nuevo servicio.

4 Experimentación y análisis de resultados

El propósito fundamental de este capítulo es evaluar de manera cuantitativa el desempeño, la efectividad y el impacto operacional del modelo propuesto para la automatización de publicaciones de APIs en Azure API Management. Frente a los enfoques tradicionales de despliegue que dependen fuertemente de tareas manuales, la arquitectura desarrollada en n8n combina el razonamiento agentico asistido por modelos de lenguaje con la precisión de flujos deterministas en código.

Para llevar a cabo una validación rigurosa, se planteó un diseño experimental de carácter empírico y comparativo. Este diseño busca contrastar el esfuerzo requerido bajo el paradigma manual histórico frente a la ejecución del pipeline automatizado asistido por agentes. La evaluación no solo contempla la medición de los tiempos de ejecución del desarrollo, sino que analiza aspectos críticos de la operación real como la tolerancia a la complejidad sintáctica, la necesidad de intervención humana posterior y el comportamiento del sistema ante condiciones de falla.

4.1 Caracterización de la línea base histórica

Con el fin de contextualizar los resultados experimentales sobre una métrica objetiva y representativa de la organización, se realizó una extracción y análisis de datos históricos desde el sistema de gestión de flujo de trabajo Jira. Este análisis abarca la totalidad de solicitudes de publicación de APIs registradas entre los años 2024 y 2026.

El universo de datos consolidado comprende un total de 83 publicaciones analizadas. De este conjunto, 27 registros fueron descartados por presentar incongruencias en las marcas de tiempo, solicitudes cerradas con tiempos cercanos a 0 minutos declarados, obteniéndose una muestra efectiva de 56 publicaciones con tiempos reales de ejecución.

El análisis estadístico descriptivo de la muestra histórica refleja la alta variabilidad y el significativo cuello de botella que representa el proceso manual tradicional.

- Mediana operativa: 69,28 horas. Indica que el 50% de los despliegues históricos requirió este tiempo o más para completar su ciclo de desarrollo y publicación en ambiente de integración.
- Promedio real: 109,04 horas. Refleja el impacto de la dispersión de tiempos generada por iteraciones de corrección entre los equipos de desarrollo y la célula de API Management.
- Percentil 80 (P_{80}): 192,94 horas. Muestra que el 80% de los servicios analizados tardó hasta más de 24 días hábiles en su desarrollo.

Esta caracterización estadística establece la línea base sobre la cual se midió la eficiencia del sistema propuesto.

4.2 Diseño experimental y escenarios de prueba

Para las pruebas de ejecución del pipeline asistido por IA, se seleccionó un conjunto de 4 APIs representativas de la organización, las cuales presentan distintos niveles de densidad de datos y complejidad arquitectónica. Esta selección garantiza representatividad técnica dentro de la variabilidad del ecosistema corporativo.

1. api-hub-settlements-reports (baja complejidad): Servicio liviano compuesto por 2 *endpoints*. Representa las integraciones estándar con baja densidad de atributos.
2. api-hub-registrations (complejidad media): Servicio compuesto por 3 *endpoints*, con una carga promedio de 10 atributos de entrada y 10 de salida

3. api-pensionquotes (complejidad media-alta**):** Componente con 3 *endpoints*, contrato con estructuras de datos complejas.
4. api-consents (alta complejidad): Componente con 7 *endpoints*. Presenta una alta densidad sintáctica con un promedio de 15 atributos de entrada y hasta 40 atributos de respuesta/salida.

4.2.1 Variables y métricas de rendimiento

Para evaluar empíricamente la hipótesis formulada, la fase experimental cuantifica de manera precisa el desempeño del proceso mediante la recolección de tres componentes temporales clave. Estas variables permiten contrastar la línea base histórica frente a la efectividad del modelo asistido por agentes orquestados.

- Tiempo manual histórico (T_m): Expresado en horas o minutos convertidos para efectos comparativos, corresponde al tiempo total registrado en la plataforma Jira que le tomó a un desarrollador especialista llevar a cabo el proceso tradicional. Esta métrica abarca la revisión sintáctica del contrato base, la traducción manual de definiciones al idioma inglés bajo la convención *PascalCase*, la construcción de los documentos de políticas XML de Azure API Management, tanto globales como por *endpoint*, el aprovisionamiento de recursos en el *gateway* y la corrección de errores de configuración. Cabe destacar que esta variable histórica es conservadora, ya que frecuentemente no contabilizaba el tiempo posterior dedicado a la redacción manual de la documentación técnica institucional.
- Tiempo pipeline agentico (T_{pa}): Expresado en minutos, comprende la latencia de procesamiento automatizado e ininterrumpido que toma la cadena de ejecución orquestada en n8n. Esta variable contempla la captura del contrato vía *webhook*, las llamadas de inferencia a los modelos de lenguaje, la estructuración y validación sintáctica mediante *parsers*, la generación en código de las políticas XML, el aprovisionamiento de recursos y variables de ambiente en *Azure API Management*, el resguardo de artefactos binarios en el almacenamiento *cloud* y la generación e inyección automatizada de la documentación en la plataforma Confluence.
- Tiempo de corrección u ajuste manual (T_c): Expresado en minutos, cuantifica el esfuerzo humano indispensable requerido de forma posterior a la ejecución del pipeline para auditar, ajustar o subsanar discrepancias de nombrado o parámetros de infraestructura antes de autorizar la promoción del servicio a ambientes superiores. Esta variable es fundamental para determinar el impacto real del concepto *human-in-the-loop* dentro de la propuesta de solución.

A partir de estas componentes, se define el "tiempo total de publicación del modelo agéntico" **(T_{total})** como:

$$T_{total} = T_{pa} + T_c$$

El indicador clave de rendimiento (KPI) utilizado para validar la hipótesis es el "porcentaje de reducción del time-to-market" **($\Delta T - T - M$)**, calculado mediante la relación:

$$\Delta T - T - M = \left(\frac{T_m - (T_{pa} + T_c)}{T_m} \right) \times 100$$

4.3 Resultados experimentales

A continuación, se consolidan las mediciones temporales obtenidas tras la ejecución del experimento sobre la muestra seleccionada de APIs representativas. Con el fin de sostener la transparencia metodológica de la investigación, el *dataset* incluye tanto las ejecuciones exitosas como el caso en el que la automatización se interrumpió por desbordamiento sintáctico.

En el marco del diseño experimental, la variable dependiente T_c está definida conceptualmente como el tiempo invertido por un usuario o célula solicitante para auditar y parametrizar un conjunto de artefactos exitosamente desplegados por el pipeline agéntico en *Azure API Management*.

En el caso particular del servicio api-consents, el pipeline automatizado no logró completar la fase de generación de artefactos ni alcanzar la etapa de despliegue en el Azure APIM debido a un error sintáctico previo en la etapa de *parsing* del JSON de salida. Por este motivo, metodológicamente no resulta procedente ni técnicamente válido computar un tiempo de ajuste T_c, ya que no existió un objeto de infraestructura provisto sobre el cual realizar parametrizaciones finales. Asignar un valor arbitrario a T_c en este escenario habría distorsionado la naturaleza de la variable y enmascarado la falla del flujo automatizado.

Tabla 1 Resultados de la experimentación

Nombre de la API	Complejidad del contrato	Tiempo manual (T_m)	Tiempo pipeline agentico (T_{pa})	Tiempo corrección (T_c)	Tiempo total de publicación (T_{total})	Porcentaje de reducción ($\Delta T - T - M$)	Estado de la ejecución
api-hub-settlements-reports	Baja (2 *endpoints*)	1,83 hrs (109,8 min)	10 min	3 min	13 min	88,16%	Exitosa
api-hub-registrations	Media (3 *endpoints*, 10 atributos)	76,26 hrs (4.575 min)	15 min	3 min	18 min	99,61%	Exitosa
api-pensionquotes	Media-Alta (3 *endpoints* complejos)	98,54 hrs (5.912 min)	10 min	5 min	15 min	99,75%	Exitosa
api-consents	Alta (7 *endpoints*, 40 atributos)	162,77 hrs	N/A	N/A	N/A	N/A	Fallida

En términos de tasa de efectividad operativa, la primera iteración de pruebas del modelo de autoatención asistido por agentes registró un 75% de éxito de despliegue directo sin intervención de desarrollo (3 de 4 casos evaluados). En los tres casos exitosos, el pipeline no solo completó el ciclo *end-to-end*, sino que redujo *el time-to-market* en más de un orden de magnitud. Por su parte, la falla en el 25% restante (1 caso) es analizada en profundidad a continuación, no como una invalidez de la hipótesis, sino como la identificación empírica de una barrera de densidad sintáctica en el prompting único.

4.4 Análisis de resultados y hallazgos

El análisis detallado de la evidencia empírica permite comprender el comportamiento operativo y las fronteras técnicas de la solución.

4.4.1 Desempeño en servicios de baja y media complejidad

En los servicios correspondientes a las categorías de complejidad baja y media (api-hub-settlements-reports, api-hub-registrations y api-pensionquotes), la arquitectura híbrida, que combina inferencia semántica por LLM con ejecución determinista, demostró una aceleración drástica en los tiempos de entrega.

Al analizar los datos recabados, se observa que el tiempo de procesamiento automatizado puramente agéntico (T_{pa}) se mantuvo extraordinariamente estable en un rango determinista de 3 a 5 minutos, independientemente de si el servicio poseía 2 o 3 endpoints. Al sumar el tiempo de intervención y supervisión humana posterior (T_c), el cual osciló entre 10 y 15 minutos, el tiempo total de ciclo (T_{total}) se posicionó entre los 13 y 18 minutos.

Al contrastar estos valores con la línea base histórica (T_m), los resultados ratifican empíricamente la hipótesis de investigación:

1. En la API api-hub-settlements-reports, el tiempo de publicación se redujo de 1,83 horas (109,8 minutos) a solo 13 minutos totales ($T_{pa} + T_c$), representando una disminución del 88,16% en el *time-to-market*.

2. En la API api-hub-registrations, el impacto fue aún más contundente, disminuyendo de un tiempo histórico de 76,26 horas (4.575,6 minutos) a un ciclo automatizado y auditado de 18 minutos, lo que equivale a un porcentaje de reducción del 99,61%.
3. En la API api-pensionquotes, el tiempo se redujo de 98,54 horas (5.912,4 minutos) a 15 minutos totales, alcanzando un ahorro temporal del 99,75%.

Estos hallazgos confirman que la automatización asistida por agentes elimina la mayor parte del esfuerzo manual requerido para la configuración y despliegue de artefactos en Azure APIM. En términos cualitativos, esta drástica contracción del *time-to-market* remueve de manera efectiva el cuello de botella en las células especialistas, al sustituir horas de trabajo operativo repetitivo por una ejecución orquestada e instantánea. Esto permite liberar capacidad técnica crítica en los especialistas, quienes quedan habilitados para reorientar su dedicación hacia actividades de arquitectura estratégica y diseño de soluciones complejas.

4.4.2 Análisis de la intervención humana (T_c)

Las correcciones posejecución (10 a 15 minutos) se auditaron en código y se determinó que no fueron provocadas por fallas sintácticas en las políticas XML generadas por los agentes, sino por dos factores externos a estas.

- Discrepancias de nomenclatura de atributos: Descalces menores entre la convención de nombres en de atributos usados previamente, por ejemplo: *"pension quotes*" versus "*pension contribution*".
- Seteo de variables de ambiente (*named values*): Inconsistencia de referencias con variables globales preexistentes en APIM. Por ejemplo, mientras el agente generó un *named value* nuevo llamado "*url-backend-registrations"*, el entorno de Azure APIM ya utilizaba una variable global estandarizada llamada *backend-url*, requiriendo un ajuste manual para reutilizar el artefacto existente.

4.4.3 Análisis de falla en servicio de alta complejidad

El fallo registrado en la ejecución del servicio api-consents constituye uno de los hallazgos técnicos de mayor significancia de la presente investigación, al permitir delimitar las fronteras de consolidación de datos en arquitecturas agénticas jerárquicas aplicadas a DevOps.

4.4.3.1 Impacto de la complejidad estructural

El componente api-consents representa el escenario de mayor densidad estructural dentro del *dataset*, caracterizado por exponer 7 *endpoints* independientes, con esquemas de entrada/salida que alcanzan más de 40 atributos, además de incluir estructuras de objetos anidados y reglas de validación específicas por recurso. En la arquitectura implementada, el subflujo *ApiPublisher Orchestrator* actúa como un agente orquestador que delega la generación de reglas de negocio invocando *N* veces al *Operation Policy Agent*, una iteración por cada endpoint presente en el contrato OpenAPI. Si bien la generación individualizada de las políticas por operación se ejecuta correctamente para cada endpoint de manera aislada, la vulnerabilidad del modelo se manifiesta en el momento en que el agente orquestador debe sintetizar, consolidar y estructurar la respuesta global en un artefacto JSON unificado para transferirlo a las etapas posteriores del pipeline.

4.4.3.2 Desbordamiento en la fase de consolidación

El desencadenamiento de la falla operó bajo la siguiente secuencia técnica.

1. Acumulación de *payload* en el agente orquestador: Tras completar exitosamente las *N* llamadas iterativas al agente de políticas, el agente orquestador principal reunió la totalidad de las políticas XML generadas, las descripciones traducidas y las configuraciones de los 7 *endpoints* para construir el *payload* final de salida.

2. Saturación del límite de tokens de salida: Durante el proceso de síntesis y estructuración de la respuesta consolidada, el volumen total de código XML y JSON superó el límite de tokens de salida permitidos en una sola llamada de respuesta del agente principal.
3. Truncamiento de la respuesta: Al alcanzar la longitud máxima de respuesta soportada por el modelo en la llamada del orquestador, la salida generada fue truncada de forma abrupta antes de lograr cerrar las estructuras de sintaxis del objeto JSON principal.
4. Excepción en el *structured output parser*: El nodo "*structured output parser*" de n8n, diseñado para validar que la salida del agente orquestador cumpla con un esquema JSON estricto antes de llamar a los subflujos de despliegue y documentación, intentó procesar la cadena truncada. Al detectar un documento malformado, el nodo lanzó la excepción *"Model output doesn't fit requiered format*".

4.4.3.3 Lecciones de diseño en arquitecturas agénticas

Este hallazgo demuestra empíricamente que en arquitecturas agénticas jerárquicas, el cuello de botella no reside en la capacidad del agente especialista para resolver una tarea atómica, sino en la capacidad del agente orquestador para consolidar y estructurar un gran volumen de información en un solo paso final. La evidencia indica que concentrar la responsabilidad de ensamblaje en un único objeto JSON consolidado expone al sistema a fallas por saturación de tokens de salida cuando aumenta la densidad sintáctica del servicio. Esta limitación establece un antecedente crítico para el diseño de canalizaciones agénticas en entornos empresariales, identificando que el patrón de entrega monolítico debe evolucionar para evitar puntos únicos de fallo en la capa de orquestación.

5 Conclusiones y trabajos futuros

5.1 Conclusiones

La presente investigación abordó de forma integral la problemática de la fricción operativa y los extensos tiempos de despliegue en la exposición de APIs para una institución del sector bancario. A través del diseño, desarrollo e implementación de un modelo asistido por agentes de software orquestados en la plataforma n8n e integrados con Azure API Management, se logró validar experimentalmente que la automatización del ciclo de gobierno, estandarización y provisión técnica transforma radicalmente la eficiencia del ecosistema tecnológico institucional.

A partir de la evaluación de la propuesta en un entorno de pruebas con casos de uso reales, se extraen las siguientes conclusiones.

5.1.1 Validación de la hipótesis e impacto en el *time-to-market*

La hipótesis central de este trabajo de tesis ha sido confirmada empíricamente. Los resultados experimentales demostraron que la adopción de un pipeline agéntico orquestado permite reducir el tiempo total de ciclo de publicación, *time-to-market*, de un servicio API en porcentajes que van desde un 88,16% hasta un 99,75% para escenarios de complejidad baja y media. El proceso manual histórico, que exhibía una mediana operativa de 69,28 horas y promedios superiores a las 100 horas debido a iteraciones de corrección, fue sustituido exitosamente por un ciclo automatizado que resuelve la preparación, despliegue y documentación técnica en un rango de 13 a 18 minutos totales, incluyendo la supervisión y ajuste humano.

5.1.2 Validación del paradigma híbrido agéntico/determinista

Una de las contribuciones arquitectónicas más relevantes de esta investigación consiste en la demostración de que los modelos de lenguaje deben ser integrados bajo un esquema de responsabilidad acotada y delimitada. La combinación de componentes de inferencia generativa, encargados de la traducción semántica y diseño

dinámico de políticas por operación, con nodos de ejecución determinista en código, encargados de la aplicación de reglas globales de seguridad, estructuración XML y provisión en Azure APIM, demostró ser el enfoque óptimo. Este diseño híbrido elimina el riesgo de alucinaciones en la infraestructura *cloud* y garantiza que el 100% de los servicios desplegados de forma exitosa cumpla rigurosamente con los estándares de gobierno corporativo sin requerir reescrituras de código base.

5.1.3 Descentralización y modelo de autoatención

El hallazgo más significativo del modelo radica en su capacidad para romper la dependencia histórica respecto al equipo de especialistas en API Management. Al codificar las reglas de negocio, gobierno y seguridad dentro de la arquitectura de agentes y flujos orquestados, la solución permite democratizar el proceso de creación de APIs. Esto habilita un modelo de autoatención en el cual células de desarrollo con menor nivel de especialización en Azure APIM pueden generar, estandarizar y desplegar sus propios servicios de manera autónoma, rápida y segura. Las mínimas intervenciones observadas (T_c, entre 10 y 15 minutos) no requieren el criterio de un desarrollo experto, sino simples confirmaciones de contexto local por parte del equipo solicitante, confirmando que la carga operativa y el rol de cuello de botella quedan completamente eliminados del equipo central.

5.1.4 Limites sintácticos en inferencia monolítica

El fallo sistemático registrado durante la prueba del servicio de alta complejidad (api-consents, compuesto por 7 *endpoints* y más de 40 atributos de salida) aportó un hallazgo científico y técnico de elevado valor. La evidencia empírica demostró la existencia de un umbral o límite de densidad sintáctica en arquitecturas de inferencia en un solo paso. Al intentar mapear de forma simultánea un contrato denso, la cantidad de tokens sobrepasó la capacidad de respuesta estructurada del LLM, generando un truncamiento del JSON de salida y desencadenando un error de lectura en el analizador sintáctico de n8n. Este resultado delimita con precisión la frontera operativa del modelo actual y sienta las bases teóricas para la evolución hacia patrones de procesamiento iterativo por subrecursos.

5.1.5 Efecto colateral positivo en la documentación

Si bien el objetivo central del modelo se enfocó en reducir los tiempos de publicación y eliminar cuellos de botella operativos, la integración del subflujo automatizado de documentación en Confluence demostró ser un beneficio colateral significativo. La ejecución del pipeline asegura que cada servicio desplegado en Azure APIM quede documentado de forma automática y estandarizada en la plataforma wiki corporativa al mismo tiempo de su publicación. Esto elimina la necesidad de tareas manuales posteriores de redacción y garantiza una sincronización transparente entre la infraestructura desplegada y el catálogo de conocimiento de la organización.

5.2 Trabajos futuros

A partir de los hallazgos y limitaciones identificadas durante la fase experimental, se proponen las siguientes líneas de desarrollo e investigación futura.

- Diccionario centralizado de nomenclatura y mapeo semántico: Implementar un componente de almacenamiento y consulta que funcione como un diccionario persistente de términos, entidades y reglas de traducción corporativas. Al integrar este repositorio dentro del flujo del *Swagger Translator Agen*t, se garantizará una consistencia estricta en el nombrado de atributos y variables de ambiente a través de todas las APIs de la organización, reduciendo a cero las intervenciones humanas (T_c) por discrepancias de nomenclatura.
- Evolución hacia una arquitectura de despliegue incremental y flujo modular: A partir de la limitación identificada en los ensayos del servicio de alta densidad sintáctica (api-consents), se propone rediseñar

el patrón de integración del orquestador principal. En lugar de exigir que el agente orquestador consolide las N respuestas de los agentes especialistas en un único artefacto masivo en memoria, el pipeline en n8n evolucionará hacia un procesamiento por flujos individuales. Bajo este nuevo esquema, cada política generada por el "*operation policy agent*" para un *endpoint* individual será procesada, validada y provista directamente en Azure API Management y Confluence de manera iterativa e independiente. Este enfoque distribuido eliminará el riesgo de truncamiento por límite de tokens de salida en la fase de síntesis del orquestador, otorgándole al sistema una capacidad de escalabilidad lineal capaz de procesar contratos OpenAPI con un número indeterminado de *endpoints* sin comprometer la estabilidad de la ejecución.

- Bucle de autocorrección agéntica: Incorporar un mecanismo de retroalimentación donde, si la API REST de Azure APIM rechaza la política XML por un error de sintaxis o falta de parámetro, el mensaje de error de Azure sea capturado y enviado nuevamente al agente generador para que corrija el documento XML y reintente el despliegue de forma autónoma.
- Evolución hacia GitOps e integración con repositorios de código: En la presente prueba de concepto se utilizó Google Drive como repositorio intermedio de artefactos para simplificar la infraestructura de pruebas. Como trabajo futuro, se contempla sustituir este módulo por la integración directa mediante APIs hacia repositorios de código fuente corporativos, tales como GitHub. Esto permitirá adoptar un modelo estricto de GitOps, donde cada cambio generado por el pipeline agéntico desencadene automáticamente un *Pull Request* con los contratos OpenAPI y las políticas XML, habilitando revisiones de código pareadas y auditorías de seguridad automatizadas.
- Módulo de *guardrails* avanzados y evaluación automática de ciberseguridad: Para la evolución de la solución hacia un entorno de producción crítico, se aconseja la incorporación de un módulo dedicado de *guardrails* de seguridad en la fase de ingesta del *webhook*. Este componente actuará como un filtro previo de sanitización para auditar el contrato OpenAPI recibido, detectando patrones de inyección de *prompts* indirecta o contenidos anómalos antes de ser procesados por los agentes agénticos. Asimismo, se propone integrar un motor de validación estática de políticas XML posgeneración, el cual evaluará automáticamente que los artefactos generados por el "*operation policy agent*" cumplan con los estándares de *OWASP API Security Top 10* antes de su aprovisionamiento en Azure API Management.
- Integración nativa en pipelines de CI/CD: Si bien en el presente trabajo la activación del *webhook* recae en una decisión consciente e interactiva del desarrollador para respetar el gobierno de exposición de microservicios, la arquitectura desacoplada basada en eventos deja las bases sentadas para su integración en pipelines de CI/CD corporativos. Como trabajo futuro, se propone ejecutar la invocación de este *webhook* como una tarea personalizada en el pipeline de despliegue de microservicios en Azure DevOps. De esta manera, cuando una célula determine que la versión de su microservicio incluye cambios en el contrato de API que deban ser publicados, podrá habilitar un paso condicional en su pipeline de entrega continua, gatillando automáticamente la orquestación agéntica como parte del flujo de despliegue institucional.

5.3 Agradecimientos

Deseo expresar mi más profundo agradecimiento a mis padres, Zoila Vega y Luis Fuentes, por su presencia incondicional en mi vida y por haberme inculcado desde siempre el valor y el respeto por la educación como pilar fundamental de mi desarrollo. Asimismo, extiendo mi gratitud a mi pareja, Gabriela Dubó, por su apoyo constante en cada etapa de este proceso, por su aliento inagotable y por ser mi principal pilar y motivación; su compañía ha sido una fuente invaluable de fortaleza para culminar con éxito este desafío.

Referencias

[1] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2003.
[2] S. Newman, *Building Microservices, 2nd Edition*. Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/

[3] «OpenAPI Specification v3.2.0». Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://spec.openapis.org/oas/v3.2.0.html
[4] P. Mell y T. Grance, «The NIST Definition of Cloud Computing», National Institute of Standards and Technology, NIST Special Publication (SP) 800-145, sep. 2011. doi: 10.6028/NIST.SP.800-145.
[5] T. E. Mahmood Ricardo Puttini, Zaigham, *Cloud Computing: Concepts, Technology & Architecture*. Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://www.oreilly.com/library/view/cloud-computing-concepts/9780133387568/
[6] «What is API Lifecycle Management?» Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://swagger.io/blog/what-is-api-lifecycle-management/
[7] G. Bondel, A. Landgraf, y F. Matthes, «API Management Patterns for Public, Partner, and Group Web API Initiatives with a Focus on Collaboration», en *Proceedings of the 26th European Conference on Pattern Languages of Programs*, en EuroPLoP '21. New York, NY, USA: Association for Computing Machinery, ene. 2022, pp. 1-17. doi: 10.1145/3489449.3490012.
[8] M. M. Amundsen Erik Wilde, Ronnie Mitra, Mike, *Continuous API Management*. Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://www.oreilly.com/library/view/continuous-api-management/9781492043546/
[9] D. F. Humble Jez, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/
[10] Russell/Norvig, *Artifical Intelligence: A Modern Approach | Fourth Edition| By Pearson*.
[11] «An Introduction to MultiAgent Systems, 2nd Edition | Wiley», Wiley.com. Accedido: 15 de agosto de 2026. [En línea]. Disponible en: https://www.wiley.com/en-us/an-introduction-to-multiagent-systems-2nd-edition-p-978EUDTE00553R150
[12] S. Chauhan *et al.*, «From Specification to Service: Accelerating API-First Development Using Multi-Agent Systems», 22 de octubre de 2025, *arXiv*: arXiv:2510.19274. doi: 10.48550/arXiv.2510.19274.
[13] «API Agent - IBM API Connect». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://www.ibm.com/products/api-connect/api-agent
[14] «IBM API Connect Cloud Service». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://www.ibm.com/docs/es/api-connect/cloud/10.0.x_saas?topic=creating-deploying-publishing-apis-using-api-agent
[15] «n8n Docs». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://docs.n8n.io
[16] «Code | Nodes | n8n Docs». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.code
[17] «Sub-nodes | Nodes | n8n Docs». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://docs.n8n.io/integrations/builtin/cluster-nodes/sub-nodes
[18] PatAltimore, «Azure API Management - Overview and Key Concepts». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts
[19] PatAltimore, «Policies in Azure API Management». Accedido: 16 de agosto de 2026. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-policies