UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

Maestría en

Software con mención en Desarrollo Web Y Móvil

RPC-SO-34-NO.778-2021

Opción de Titulación:

Proyecto de titulación con componentes de investigación aplicada y/o de desarrollo

Tema:

DISEÑO DE UN API DE PREDICCIÓN DE DÍAS ÓPTIMOS DE COBRANZA EN EL ÁREA FINANCIERA MEDIANTE REGRESIÓN LINEAL Y ARQUITECTURA DE MICROSERVICIOS EN LA NUBE CON AWS

Autor(es)

Juan Carlos Flores Mantilla

Director:

Bryam David Vega Moreno

GUAYAQUIL – Ecuador

2025

Autor(es):

Juan Carlos Flores Mantilla.
Ingeniero en Sistemas Computacionales
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Guayaquil.
juancfm99@gmail.com

Dirigido por:

Bryam David Vega Moreno.
Ingeniero en Ciencias de la Computación
Magíster en Software con Mención en diseño de arquitecturas
vegabryam40@gmail.com

JUAN CARLOS FLORES MANTILLA
DISEÑO DE UN API DE PREDICCIÓN DE DÍASÓPTIMOS DE COBRANZA EN EL ÁREA FINANCIERAMEDIANTE REGRESIÓN LINEAL Y ARQUITECTURA DEMICROSERVICIOS EN LA NUBE CON AWS

DEDICATORIA

A mi papá, Luis Flores, a mi mamá, Patricia Mantilla, por siempre apoyarme y exigirme que prepare para mi futuro, y sobre todo, por todo lo que hacen por mí siempre con amor. A mis hermanos, Luis y Cesar, que siempre han sido un ejemplo de superación y sacrificio, para poder darle muchas alegrías y orgullo a nuestros padres.

AGRADECIMIENTO

Agradezco a Dios por todo lo bueno y por lo malo, porque al final siempre son experiencias de vida que me forjan para ser una mejor persona cada día. Agradezco a mi familia, por su apoyo incondicional. Agradezco a mi tutor, Ing. Bryam Vega, por su paciencia y compromiso para poder sacar adelante este proyecto de tesis.

Tabla de Contenido

Índice de tablas

Índice de figuras.

Diseño de un API de predicción de días óptimos de cobranza en el área financiera mediante regresión lineal y arquitectura de microservicios en la nube con AWS

Autor(es):

JUAN CARLOS FLORES MANTILLA

Resumen

En el contexto financiero, el tiempo en que una empresa recupera su efectivo, ayuda al flujo y mejora la operación. El uso de la inteligencia artificial ha ayudado a mejorar muchos procesos. Al momento de realizar cobranza se puede usar inteligencia artificial para aborda la cobranza, analizando los patrones de pago con algoritmos que permitan la predicción. En este proyecto se llevará acabo el diseño y despliegue de un API de predicción para determinar los días óptimos de cobranza. Enfocándose en mejorar la eficiencia y efectividad del proceso de recaudación, el algoritmo utiliza un modelo de regresión para analizar variables clave como el día de pago, día de facturación, día máximo de pago y el monto, identificando patrones históricos que influyen en la puntualidad de los pagos. Finalmente, Para cumplir con este objetivo se desplegará el API en una arquitectura de microservicios haciendo uso de los servicios que brinda AWS (Amazon Web Services). Se usará contenedores para poder aislar el algoritmo y hacer uso del mismo mediante tecnología de fastAPI.

Palabras clave:

Microservicios, Amazon Web Services, fastAPI, Docker, Finanzas, Cobranza, Inteligencia Articial.

Abstract

In the financial context, the time it takes a company to recover its cash improves cash flow and improves operations. The use of artificial intelligence has helped improve many processes. When collecting payments, artificial intelligence can be used to address collections by analyzing payment patterns with algorithms that enable prediction. This project will design and deploy a prediction API to determine optimal collection days. Focusing on improving the efficiency and effectiveness of the collection process, the algorithm uses a regression model to analyze key variables such as payment day, billing day, maximum payment day, and amount, identifying historical patterns that influence payment timeliness. Finally, to achieve this objective, the API will be deployed in a microservices architecture using the services provided by AWS (Amazon Web Services). Containers will be used to isolate the algorithm and leverage it through fastAPI technology.

Palabras clave:

Microservices, Amazon Web Services, fastAPI, Docker, Finance, Collection, Artificial Intelligence

1. Introducción

La transformación digital en la actualidad está en constante evolución y eso ha beneficiado a los procesos internos de muchas organizaciones mediante el uso de tecnologías y sistemas inteligentes. Uno de los aspectos críticos para la eficiencia operativa es la gestión de cobranzas, ya que incide directamente en la liquidez, el flujo del dinero y la sostenibilidad financiera de las empresas. Identificar de manera precisa los días óptimos para realizar acciones de cobranza puede mejorar significativamente el proceso de recuperación de cartera y reducir los costos asociados al mismo. En respuesta a esta necesidad, el presente trabajo propone el diseño de un API de predicción que, utilizando técnicas de regresión lineal, permita estimar los días más adecuados para efectuar las cobranzas a clientes. Esta solución propone que se implemente una arquitectura moderna basada en microservicios y está desplegada en la nube gracias a la tecnología de Amazon Web Services (AWS), lo que garantiza escalabilidad, disponibilidad y facilidad de integración con otros sistemas empresariales.

Esta tesis se estructura en torno al análisis de datos históricos de pagos, la implementación de un modelo de predicción con regresión lineal, y el diseño de un API para que pueda ser usado en otros entornos. Asimismo, se evalúa el desempeño del sistema utilizando métricas estándar del aprendizaje automático, entre ellas se usará el error cuadrático medio (MSE) y también el coeficiente de determinación (R^2), y se valida su funcionamiento en un entorno controlado en la nube.

2. Determinación del Problema

Las cuentas por cobrar deben ser gestionadas de una buena forma, esto garantizará que la salud financiera de las empresas sea óptima. Sin embargo, muchas organizaciones enfrentan desafíos significativos en este ámbito, especialmente aquellas que aún dependen de métodos tradicionales y manuales. Esta situación ha generado un creciente interés por incorporar herramientas tecnológicas que permitan mejorar la eficiencia del proceso (Moonflow, 2020).

Los principales desafíos identificados tenemos:

Procesos manuales y obsoletos: Muchas empresas continúan utilizando archivos de excel o registros manuales para gestionar sus cobranzas, lo que incrementa el riesgo de errores y retrasa la recuperación de cartera (Moonflow, 2020).

Comunicación ineficiente con los deudores: La falta de canales de contacto efectivos con los clientes que mantienen la deuda, dificulta el seguimiento y la resolución oportuna de las mismas (Inbux, 2023).

Altos índices de morosidad: La morosidad continúa siendo un problema estructural en muchas organizaciones, afectando directamente su flujo de caja y capacidad operativa (McCollect, 2023).

Falta de automatización: La ausencia de plataformas digitales limita la capacidad de respuesta ante impagos y reduce la eficiencia operativa (Moonflow, 2020).

Cumplimiento normativo y protección de datos: Las empresas deben garantizar el cumplimiento de normativas legales en materia de privacidad y transparencia, lo que representa un desafío adicional (Inbux, 2023).

La ineficiencia en la gestión de cobranzas no solo afecta la liquidez inmediata de efectivo, al contrario, también se pone en evidencia la viabilidad a largo plazo de las

compañías o instituciones, incluso en escenarios extremos y tiempos prolongados de no recuperación de cartera, podría llevar a la quiebra (Tribuna, 2024).

Tabla 1. Resumen de desafíos comunes en la gestión de cobranza.

Desafío	**Descripción**
Procesos manuales y obsoletos	Uso de hojas de cálculo y registros manuales que aumentan errores y retrasos.
Comunicación ineficiente	Dificultad para contactar a deudores y gestionar pagos pendientes.
Altos índices de morosidad	Incremento en cuenta incobrables que afectan la liquidez
Falta de automatización	Ausencia de sistemas que permitan una gestión proactiva y eficiente.

La implementación de soluciones tecnológicas, como sistemas automatizados de cobranza y herramientas de análisis predictivo, se presentan como una estrategia efectiva y que se puede mantener en el tiempo para superar dichos desafíos y poder lograr una mejora en la eficiencia de la recuperación de las cuentas por cobrar. Con ayuda de la implementación de tecnologías en donde se podría considerar sistemas automáticos de gestión de cobranzas, inteligencia artificial y análisis predictivo se ha demostrado que será efectiva para mejorar, optimizando el flujo de efectivo y fortaleciendo la salud financiera de las organizaciones. La transición hacia soluciones tecnológicas en la gestión de cartera no es solo una opción estratégica, sino una necesidad para enfrentar los desafíos financieros y operativos que anteriormente limitaban la eficiencia de las organizaciones.

3. Objetivos

3.1 Objetivo General

Desarrollar un API con inteligencia artificial para predecir el mejor día para realizar cobranza mediante regresión lineal, a través del análisis de patrones de pago de clientes y su comportamiento histórico.

3.2 Objetivos específicos

- Revisar estado del arte de modelos predictivos usados en machine learning para predicciones.
- Recolectar y preparar los datos históricos de pago de clientes e identificar variables relevantes.
- Desarrollar un sistema de predicción basado en los datos históricos.
- Analizar indicadores para validar desempeño del algoritmo y asegurar su efectividad.
- Implementar el API en arquitectura de microservicios en la nube.

4. Justificación del Problema

Actualmente, en el mundo de empresas existe mucha competencia por lo cual la tecnología resulta ser muy relevante para que se cumplan las metas y de esa manera los clientes o consumidores se sientan satisfechos y se mantengan una buena relación laboral y tener una retentiva de clientes prudentes y que ayuden financieramente (Vásquez,2023).

La gestión de los procesos de cobranzas en la actualidad atraviesa desafíos que son clave para las empresas, especialmente en aquellas que la liquidez de dinero es muy fluctuante y depende de las estrategias de cobranza que se lleven acabo por el departamento encargado para poder reducir las cuentas por cobrar. Tradicionalmente, los procesos de cobranza han dependido de estrategias basadas

en reglas fijas o en la experiencia de los gestores, lo que puede generar ineficiencias debido a la variabilidad en el comportamiento de pago de los clientes.

Las estrategias tradicionales de cobranza suelen girar entorno a enfoques generales, como fechas límites de corte y de mora en cartera, lo cual puede resultar en una operación ineficiente y posterior a esto el aumento de los costos de operación para intentar recuperar esos ingresos en mora, también se debe tener en cuenta la participación reactiva del personal, que se debe tener en cuenta para resolver inconvenientes que afecten el estado financiero. (Gómez,2021).

La innovación y continuo avance de la tecnología, ha hecho posible que se usen técnicas en donde la inteligencia artificial juega un papel importante y ayudan a la toma de decisiones con el propósito que se pueda llevar a cabo una optimización de los procesos y, por lo tanto, reducir la brecha de antigüedad de las cuentas por cobrar. Con el uso de la IA, también ayudará a que las personas involucradas vayan adquiriendo una cultura de capacitación permanente para ir mejorando los procesos con ayuda de la tecnología.

Desde una perspectiva de desarrollo de software, esta tesis contribuye al campo de la ingeniería de software aplicada a la inteligencia de negocios, enfatizando el enfoque directo en los datos y su relación en la toma de decisiones. El uso de un sistema inteligente de cobranza no solo optimizará los procesos internos de las empresas, sino que también permitirá la integración con sistemas de gestión financiera y CRM, mejorando la automatización y personalización de las estrategias de contacto con los clientes.

Este estudio es relevante porque combina conocimientos de aprendizaje automático, procesamiento de datos y desarrollo de software, aplicados a un problema real del ámbito financiero. Los resultados de la investigación pueden ser utilizados tanto por empresas que buscan mejorar su proceso de cobranza como por desarrolladores de software interesados en la aplicación de IA en entornos empresariales.

5. Marco teórico referencial

En el capítulo anterior se pudo detallar como la tecnología con el pasar de los años va aumentando su capacidad para mejorar varios procesos dentro de una organización. En este proyecto nos estamos enfocando en el proceso de cobranza dentro de una organización. Con el fin de mejorar tiempos de recolección de cartera, y poder reducir el tiempo del gestor de cobranza para poder realizar actividades o enfocarse en grupos más importante, un algoritmo de predicción puede ser de mucha ayuda.

En este contexto, en capítulo 2 de "Marco teórico referencial" se profundizará en los temas que son fundamentales y metodológicos para sustentar la importancia de la inteligencia artificial y como mejorar el proceso antes mencionado. Se realizará un análisis entorno a la evolución de la inteligencia artificial y su actuar en varios campos de la industria, especialmente su impacto en procesos de cobranza o financieros, o en la vida cotidiana.

Adicionalmente, se profundizará en la computación en la nube, nuestro servicio desarrollado está desplegado en Amazon web services, por lo cual es importante detallar ventajas o desventajas que otorga la tecnología en la nube. Se podrá detallar otros proveedores de este tipo de servicios, hacienda énfasis en los servicios que se usarán para llevar a cabo en este proyecto para formar una arquitectura de microservicios.

Asimismo, los algoritmos de predicción usando regresión lineal se analizarán para poder comprender su funcionalidad y el porqué de su uso en este tipo de requerimiento. Con ello, también se debe tener en cuenta que indicadores de calidad se deben usar para poder definir cuando el algoritmo es válido y su rendimiento es óptimo y aceptable.

5.1 Inteligencia artificial

La inteligencia artificial es una rama de la informática que tiene como objetivo el diseño de programas y sistemas, los mismos que se caracterizan por buscar la capacidad de razonar y aprender, y en ciertos ambientes también deben adaptarse y ser capaces de realizar una toma decisiones de manera autónoma. Este enfoque no se limita exclusivamente al aprendizaje a partir de datos, sino que incluye también metodologías basadas en reglas explícitas y lógica formal. Ejemplos representativos de IA tradicional incluyen la inteligencia artificial simbólica, los sistemas expertos y las bases de conocimiento (Russell & Norvig, 2020)

La inteligencia artificial ha mejorado con el avance de la tecnología, lo que ha provocado que muchas empresas se sientan obligadas a usar las tendencias tecnológicas actuales y optar por implementar estas tendencias en sus funciones administrativas. "La inteligencia artificial es la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos."(Rouse, 2023).

El ser humano siempre ha intentado que las computadoras tengan un razonamiento óptimo y similar a la de una persona, esto también incluye la manera en la que ese razonamiento es influenciado por hechos históricos o situaciones que se aprenden con el pasar del tiempo.

La ingeniería de software ha evolucionado de tal manera que hoy en día hay la toma de decisiones, en una organización o en la vida cotidiana, sea asistida por sistemas inteligentes, todo gracias al uso de la inteligencia artificial, como por ejemplo el uso de asistentes virtuales, como podrían ser Alexa (desarrollada por Amazon), Siri (desarrollada por Apple) o el asistente de Google. Estas herramientas vienen incorporadas en nuestros dispositivos móviles y nos ayudan a resolver de manera más rápida consultas, ya sea de ámbito académico, o alguna incógnita del momento. También, siguiendo unos ciertos estándares, pueden realizar tareas más específicas como poner una alarma, agendar alguna actividad en el calendario, reproducir música, etc. (Tyler et. al., 2023)

También hay que considerar que la IA maneja muchos datos que suelen ser usados para que la toma de decisiones de puestos gerenciales sea más eficiente, esto gracias a que el uso de la tecnología con IA ayuda a procesar variedad masiva de datos en menor tiempo y darle una perspectiva más detallada a las personas que los usan para poder decidir en varios campos que sea necesario. La empresa OpenIA ha diseñado un sistema denominado GPT-4 que, apoyado con IA, trabaja con el lenguaje humano como principal recurso para el procesamiento de solicitudes de parte del usuario, esto ayuda a que sea una herramienta de innovación, que incluso podría generar nuevas visiones de modelos de negocios en varios sectores productivos (Abeliuk & Gutierrez,2021).

La inteligencia artificial desde la perspectiva donde los sistemas son capaces de aprender de acuerdo con patrones, adaptarse para tomar decisiones de manera autónoma, se puede diferenciar 3 campos principales que son el aprendizaje automático (machine learning, en inglés) y el aprendizaje profundo (deep learning, en inglés)

Figura 1. Diagrama de Venn que muestra relación entre distintas subáreas de la inteligencia artificial. (Abeliuk & Gutierres,2021)

5.2 Aprendizaje Automático

Dentro de la IA se encuentra el aprendizaje automático, la misma que se considera como una subdisciplina que estudia el desarrollo de algoritmos que tengan la habilidad de mejorar su desempeño en tareas específicas mediante el análisis de datos, es decir, no necesita de programación explícita para cada tarea. Según Aggarwal (2020), el aprendizaje automático es uno de los principales motores de la innovación actual, gracias a que es capaz de manejar información de manera masiva y descubrir relaciones ocultas en los datos.

El aprendizaje automático se puede clasificar en aprendizaje supervisado, no supervisado y por refuerzo. Esta clasificación demuestra que cada uno posee diferentes características de acuerdo con su aplicación y trabajo específico para la resolución de problemas.

5.3 Aprendizaje Profundo

El aprendizaje profundo representa una extensión del aprendizaje automático, basada en redes neuronales artificiales con múltiples capas (Deep neural networks, en inglés). Estas estructuras han cobrado un poco más de relevancia en los últimos años debido a que permiten modelar características complejas y abstractas de los datos, lo que ha facilitado que el desarrollo de sistemas de uso de reconocimiento de voz sea mejor, así como también la visión por computadora y sobre todo el procesamiento del lenguaje natural.

Como señalan Goodfellow (2021), aprendizaje profundo ha tenido un desarrollo constante gracias a que en la actualidad en muy fácil tener acceso a cantidad de información muy grande, gracias al incremento en la capacidad computacional y al desarrollo de arquitecturas como las redes convolucionales (CNN), recurrentes (RNN) y generativas (GANs).

El aprendizaje profundo requiere, en general, grandes volúmenes de datos y una considerable capacidad computacional para entrenar modelos eficaces, lo que ha

sido facilitado por los avances recientes en infraestructura de hardware y la disponibilidad de bibliotecas de software especializadas.

5.4 Computación en la nube

Otro factor para estudiar es la computación en la nube (Cloud Computing). Esta tecnología ha cambiado de manera radical en que las empresas acceden, procesan y almacenan información, permitiendo la provisión de recursos computacionales bajo demanda a través de internet. Este modelo se caracteriza por ofrecer servicios escalables, elásticos y medibles, eliminando la necesidad de mantener infraestructuras físicas propias y permitiendo a las empresas centrarse en sus objetivos estratégicos.

Según National Institute of Standards and Technology (2020), la computación en la nube es un modelo que se caracteriza por tener un acceso conveniente y bajo demandada, mismo que tiene la opción de compartir recursos configurables para que la interacción sea mínima pero efectiva mientras el proveedor ofrezca los servicios.

Sus principales características incluyen:

Autoservicio bajo demanda: los usuarios pueden aprovisionar recursos automáticamente sin intervención humana.
Acceso amplio a la red: los servicios están disponibles a través de redes estándar como internet.
Agrupación de recursos: el proveedor atiende a múltiples clientes mediante recursos físicos y virtuales compartidos.
Elasticidad rápida: los recursos pueden escalarse de forma automática según la necesidad.
servicio medido: se monitorea y reporta el uso de los servicios para brindar transparencia.
Los modelos de servicio en computación en la nube se clasifican en tres categorías principales. La Tabla 2 resume cada uno:

Tabla 2. Modelos de servicio en computación en la nube.

Modelo	Descripción	Ejemplos
Infraestructura como Servicio (IaaS)	Ofrece recursos básicos como máquinas virtuales, almacenamiento y redes.	Amazon EC2, Microsoft Azure VM
Plataforma como Servicio (PaaS)	Proporciona plataformas para desarrollo y despliegue de aplicaciones sin gestionar infraestructura.	Google App Engine, Heroku
Software como Servicio (SaaS)	Permite usar aplicaciones completas basadas en la web.	Google Workspace, Microsoft 365

Los modelos de despliegue definen cómo se accede y administra la infraestructura en la nube. En la Tabla 3 se detallan los principales:

Tabla 3. Modelos de despliegue en computación en la nube.

Modelo	Descripción
Nube publica	Infraestructura gestionada por terceros, disponible para cualquier usuario.
Nube privada	Recursos exclusivos para una organización. Puede estar alojada localmente o en un proveedor externo.
Nube híbrida	Es la unión de nubes públicas y por otro lado de nubes privadas.
Nube comunitaria	Organizaciones comparten recursos por intereses comunes.

Una ventaja que se puede notar de la computación en la nube es la reducción de costos, la escalabilidad, la alta disponibilidad y la agilidad en el desarrollo de productos y servicios digitales (Hashem,2021). No obstante, este modelo también enfrenta desafíos como la seguridad de los datos, la dependencia de los proveedores y la necesidad de cumplir regulaciones específicas en distintos contextos.

5.5 Arquitectura de microservicios

La arquitectura de microservicios representa una evolución frente a los enfoques monolíticos tradicionales, promoviendo el diseño de sistemas compuestos por múltiples servicios pequeños, independientes y desplegables de forma autónoma. En esta arquitectura los microservicios se encargan de encapsular una funcionalidad específica del negocio y posterior a eso se realiza un enlace con otros servicios haciendo uso de interfaces bien definidas, usualmente mediante APIs REST o mensajería asíncrona (Taibi & Lenarduzzi,2020).

Este enfoque busca resolver problemas comunes de escalabilidad, mantenibilidad y despliegue ágil en aplicaciones complejas. Según Richardson (2021), los microservicios permiten escalar, actualizar y desplegar partes del sistema sin necesidad de interrumpir todo el entorno, mejorando la resiliencia y la eficiencia del desarrollo. A continuación, en la Tabla 4 visualizaremos una comparación entre los enfoques monolíticos y de microservicios (Daza-Caicedo.2020):

Tabla 4. Comparación entre arquitectura monolítica y microservicios

Aspecto	**Monolítica**	**Microservicios**
Estructura	Sistema unificado y centralizado	Servicios independientes y autónomos
Despliegue	Una sola unidad	Despliegue individual por servicio
Escalabilidad	Escalado completo del sistema	Escalado granular por componente

Mantenimiento	Más difícil por el acoplamiento	Simplificado por modularidad
Tecnología	Generalmente homogénea	Posibilidad de heterogeneidad

Entre los principales beneficios se encuentran la flexibilidad tecnológica, la escalabilidad independiente y la mejora en la velocidad de desarrollo (Richardson,2021). Sin embargo, también existen desafíos como la complejidad en la gestión de múltiples servicios, la necesidad de una infraestructura más avanzada (como orquestadores tipo Kubernetes), y una mayor carga operativa (Daza-Caicedo, 2020).

5.6 Amazon web services

La plataforma de computación en la nube, impulsada por Amazon, es AWS (Amazon Web Services). AWS proporciona una amplia variedad de servicios tecnológicos bajo demanda que un desarrollador puede hacer uso. AWS fue lanzado en el 2006, y a partir de ahí ha provocado una evolución constante en este tipo de plataformas hasta convertirse en el proveedor líder mundial de servicios de infraestructura en la nube, como principal competencia de Azure, creada por Microsoft.

La disponibilidad, la seguridad, y el modelo de pago por uso. Estas características permiten a las organizaciones implementar soluciones eficientes sin necesidad de invertir en infraestructura física, facilitando la innovación y la optimización de costos (Amazon Web Services, 2024a).

El uso de AWS en el ámbito académico y empresarial ha crecido de forma sostenida, gracias a su robusta infraestructura, amplia documentación y ecosistema de herramientas integradas. AWS ofrece distintos tipos de servicios agrupados en categorías como se detalla en la Tabla 5

Tabla 5. Principales servicios de AWS por categoría

Categoría	***Servicios en AWS***
Servidores	Amazon EC2, AWS Lambda, AWS Elastic Beanstalk, Amazon Lightsail
Almacenamiento	Amazon S3, Amazon EBS, Amazon EFS, AWS Backup, Amazon Glacier
Base de datos	Amazon RDS, Amazon DynamoDB, Amazon Aurora, Amazon Redshift
Redes	Amazon VPC, AWS CloudFront, AWS Route 53, AWS Global Accelerator
Inteligencia artificial y Aprendizaje automático	Amazon SageMaker, Amazon Rekognition, Amazon Comprehend, Amazon Lex
DevOps	AWS CodeBuild, AWS CodePipeline, AWS CodeDeploy, AWS Cloud9
Contenedores y orquestación	Amazon ECS, Amazon EKS, AWS Fargate
Seguridad y autenticación	AWS Identity and Access Management (IAM), AWS Key Management Service (KMS), AWS Shield, AWS WAF
Monitoreo	Amazon CloudWatch, AWS CloudTrail, AWS Config, AWS Trusted Advisor

5.7 Tecnología Serverless en AWS

En el desarrollo de este proyecto se ha optado por un desarrollo sin servidor para poder desplegar contenedores y manipular la API y los recursos de mejor manera. Para comprender esta tecnología (serverless computing, en inglés) se debe tener en cuenta que es un modelo de despliegue de manera no local, sino en la nube, donde el proveedor de servicios, en este caso AWS, ofrece infraestructura necesaria que

el usuario configurará para ejecutar el código, lo que permite que en los equipos de desarrollo se puedan enfocar exclusivamente en la lógica del aplicativo o del negocio, esta ventaja de usar este tipo de tecnología se la define como escalabilidad automática. Otra ventaja, que está ligada a la escalabilidad es el momento de pagar por el servicio, ya que solo se paga de acuerdo con los recursos que se usan y en el momento que se usan, es decir, no se mantiene un servicio que no consume muchos recursos de la nube, el pago podría llegar a ser nulo ya que se cuenta con una capa de gratuidad. La facilidad de despliegue también es una ventaja, ya que no será necesario gestionar o configurar de manera detallada una infraestructura. (Amazon Web Services, 2024ª).

El despliegue de este proyecto se ha realizado con el servicio de AWS Fargate. Este servicio ayuda a que se ejecuten contenedores sin necesidad de servidores, y se ha usado junto con ECS, que es un gestor de contenedores.

AWS Fargate

Es un servicio de modelo serverless para contenedores que funciona con Amazon Elastic Container Service (ECS) y Amazon Elastic Kubernetes Service (EKS). Su principal característica es que permite ejecutar contenedores sin tener que aprovisionar ni administrar servidores o clústeres de máquinas virtuales. El desarrollador solo necesita definir los requisitos del contenedor (CPU, memoria, puertos, etc.), y el proveedor es el encargado de ejecutar y escalar los contenedores según la demanda (Amazon Web Services, 2024a).

Para utilizar Fargate, el usuario define una tarea (task definition) que especifica los contenedores, sus imágenes, puertos, recursos asignados, políticas de red y almacenamiento. Luego, ECS o EKS lanzan estas tareas en un entorno completamente administrado por AWS.

AWS Fargate ha sido adoptado globalmente por organizaciones que buscan eficiencia operativa, reducción de costos y escalabilidad automática en el despliegue de contenedores. A continuación, en la Tabla 6 se presentan 2 casos de

empresas muy conocidas de cómo han usado servicios serveless para mejorar sus procesos.

Tabla 6. Casos de uso de AWS Fargate

Casos de uso	**Ejemplo real**
Automatización de flujos CI/CD	Samsung Electronics automatiza su ciclo de vida de aplicaciones móviles con pipelines que ejecutan pruebas, builds y despliegues en contenedores gestionados por AWS Fargate, facilitando el escalado sin afectar el rendimiento (Amazon Web Services, 2024b).
APIs y backend sin servidor	Pinterest usa Fargate para alojar su backend sin servidor de forma más segura, eliminando puntos únicos de fallo y permitiendo respuesta automática a aumentos de tráfico sin intervención (Amazon Web Services, 2023).

Amazon Elastic Container Service (ECS)

Es un servicio que permite la administración de contenedores con la capacidad se poder ser escalable y de alto rendimiento. Estas características permiten realizar ejecuciones de contenedores docker en un clúster de servidores administrado por AWS (Amazon Web Services, 2024a). ECS permite a los desarrolladores desplegar microservicios, tareas en segundo plano, APIs o cualquier aplicación empaquetada en contenedores de manera sencilla, sin tener que operar su propia infraestructura de orquestación. Para poder realizar tareas con ECS se usará Fargate, por lo cual

aws entenderá que toda la infraestructura no será configurada por el usuario, sino que será de acuerdo con el contenedor que se vaya a desplegar.

Una ventaja de este servicio que brinda aws es que permite de manera nativa implementar varios servicios adicionales, al momento de configurarlo dará opciones que no son obligatorias donde se puede seleccionar monitoreo o servicios de autenticación. Al ser contenedores, también permite tener más control sobre los costos del servicio y que la escalabilidad se automatice y de esa manera el equipo de desarrollo siga enfocado en la lógica.

Uno de los casos típicos de uso de esta tecnología es el de despliegue de microservicio (seleccionado para el desarrollo de esta tesis), ya que esto permite mantener separadas las funcionalidades en servicios independientes.

Para el uso de ECS hay que tener en cuenta algunos componentes que se van a configurar en el despliegue del microservicio. A continuación, tenemos una lista de los componentes principales a usar y configurar (Amazon Web Services, 2024a).

Tarea (Task): Es la unidad mínima de ejecución que contiene uno o varios contenedores definidos en una Task Definition.

Servicio (Service): Permite ejecutar múltiples tareas de forma continua y asegurar la disponibilidad.

Clúster: Grupo lógico de recursos donde se ejecutan las tareas. Tendrá recursos Fargate para la ejecución.

Task Definition: Documento JSON donde se define la configuración de contenedores, incluyendo imagen, puertos, variables de entorno, volumen, etc. Esta definición se genera automáticamente de acuerdo con las elecciones del usuario.

5.8 Algoritmos de predicción

Los algoritmos de predicción son componentes esenciales en sistemas de inteligencia artificial y aprendizaje automático, cuyo objetivo es anticipar valores futuros o categorías a partir de datos históricos. Estos algoritmos extraen patrones estadísticos o estructurales de los datos para realizar inferencias que permiten tomar decisiones informadas en entornos dinámicos. Según Alzate-Valencia (2021), estos algoritmos permiten optimizar procesos empresariales mediante la estimación de comportamientos futuros, tales como la demanda de productos, el riesgo de crédito o el comportamiento. de pago de clientes. Para ello, se emplean distintos enfoques de aprendizaje automático, tanto supervisado como no supervisado. Entre los algoritmos más utilizados se encuentran:

Regresión lineal: Predice valores continuos a partir de una relación lineal entre variables.

Árboles de decisión: Estructura jerárquica para clasificar o predecir valores mediante reglas de decisión.

Bosques aleatorios (Random Forest): Conjunto de árboles de decisión que mejora la precisión y reduce el sobreajuste.

Máquinas de soporte vectorial (SVM): Algoritmo que encuentra un hiperplano óptimo para separar clases.

Redes neuronales: Modelo inspirado en el cerebro humano, capaz de capturar relaciones no lineales complejas.

k-Vecinos más cercanos (k-NN): Clasifica en función de la similitud con instancias cercanas en el espacio de características.

Cada algoritmo tiene ventajas y limitaciones según el dominio, volumen de datos, ruido presente, y necesidad de interpretabilidad. Por ejemplo, los árboles de

decisión son fáciles de interpretar, mientras que las redes neuronales suelen tener un mejor rendimiento, pero son menos explicables (González-Ríos, 2022).

La calidad de los algoritmos de predicción se evalúa mediante métricas específicas, como:

Precisión (Accuracy): Porcentaje de predicciones correctas.

Precisión y recall: Métricas importantes en clasificación desequilibrada.

F1-score: Media armónica entre precisión y recall.

Error cuadrático medio (RMSE): Evaluación de errores en regresión.

Estos algoritmos se aplican en diversos contextos, como: Predicción de ventas y demanda en comercio electrónico, diagnóstico médico automatizado, detección de fraudes financiero, segmentación de clientes y marketing personalizado, estimación de riesgo crediticio o de pago.

Algoritmos de Regresión Lineal

Uno de los algoritmos que son más utilizados en el aprendizaje automático supervisado son los algoritmos de regresión lineal. Estos algoritmos son de mucha utilidad en procesos de predicción de variables continuas. Este modelo se basa en la suposición de que existe una relación lineal entre variables. Su objetivo es ajustar una función lineal que minimice el error entre los valores predichos y los observados (Pineda-Bautista, 2021). Cuando se extiende a múltiples variables independientes, se denomina regresión lineal múltiple.

La regresión lineal se utiliza ampliamente en áreas como: Predicción de precios (inmuebles, acciones, productos), estimación de la demanda en logística y retail, modelos de riesgo financiero y análisis económico, estimaciones ambientales y científicas.

En la actualidad, la regresión lineal sigue siendo usada como modelo de base en pipelines de machine learning más complejos. En escenarios de gran volumen de

datos o no linealidad evidente, se pueden utilizar transformaciones polinómicas o modelos más robustos como árboles de decisión o redes neuronales (Martínez-Rodríguez, 2020).

Para predecir valores continuos, la regresión lineal es un modelo estadístico utilizado ampliamente debido a que se basa en la relación entre una variable dependiente y una o más variables independientes. En términos de predicción del día ideal para realizar una cobranza, esta técnica puede aplicarse para estimar el número de días que transcurren entre la fecha de emisión de una factura y la fecha en que el cliente realiza el pago (Pineda-Bautista, 2021).

Métricas de evaluación para modelos de regresión lineal

Para determinar la eficacia de un modelo de regresión lineal en la predicción del día ideal de cobranza, es fundamental utilizar métricas estadísticas que permitan cuantificar el error entre los valores predichos y los valores reales. A continuación, se describen las métricas más comunes empleadas en este tipo de modelos (González-Rodríguez et a, 2020; Ramírez-Gallego et al., 2021)

Error Absoluto Medio (MAE) es la métrica que refleja el promedio de los errores absolutos.

Coeficiente de determinación (R^2) representa la proporción de la varianza de la variable dependiente que es explicada por el modelo. Un valor de R^2 cercano a 1 indica que el modelo explica bien la variabilidad de los datos.

Estas métricas permiten no solo evaluar la precisión del modelo, sino también compararlo con otros algoritmos más complejos como redes neuronales, bosques aleatorios o modelos híbridos (Ramírez-Gallego et al., 2021).

.

Para el desarrollo de esta tesis, se emplearon varias herramientas y tecnologías de la computación en la nube, en la Tabla 7 se tiene un resumen de las herramientas usadas en la etapa de desarrollo:

Tabla 7. Herramientas usadas para el desarollo

Categoría	**Herramienta**
Framework	FastAPI
Servidor ASGI	Uvicorn
Documentación automática	Swagger ui
Despliegue	Docker y AWS
Gestión de dependencias	Pip + requirements.txt
Entorno de desarrollo	VS Code

6. Desarrollo

Con el fin de cumplir con la arquitectura de microservicio, se ha realizado el despliegue usando los servicios de AWS como se grafica en el Figura 2.6

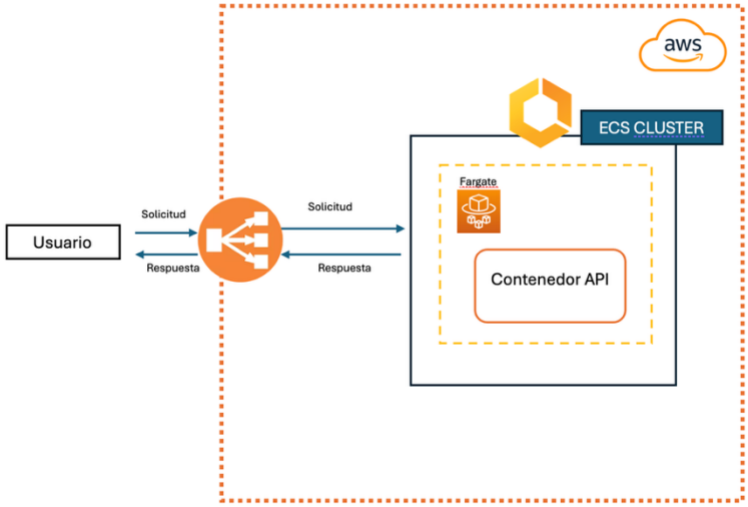


Figura 2. Diagrama de despliegue del microservicio creado

6.1 Diseño del modelo predictivo

Teniendo en cuenta la literatura analizada anteriormente, la regresión lineal es el modelo ideal y clásico para resolver problemas de predicción y estimación continua. En el escenario presentado en esta tesis, las características de la regresión lineal de ser comprensible y eficiente y además su precisión, lo convierten en el modelo ideal para el contexto de la predicción de días óptimos para realizar cobranzas.

Cuando se dispone de datos bien estructurados y sin una alta complejidad no lineal, la regresión lineal puede alcanzar niveles de precisión adecuados, especialmente si se combina con técnicas de preprocesamiento como normalización y selección de variables. Dado que es un algoritmo ampliamente conocido y soportado por bibliotecas estándar como scikit-learn en Python, su implementación e integración en una arquitectura de microservicios, como la propuesta con FastAPI y AWS, es muy mantenible en el tiempo.

Además, en estudios recientes que explican el uso de inteligencia artificial en el contexto financiero, se ha demostrado que los modelos lineales siguen siendo competitivos en tareas de predicción temporal cuando los patrones son regulares y estables (Alvarado, 2023).

Proceso del sistema predictivo

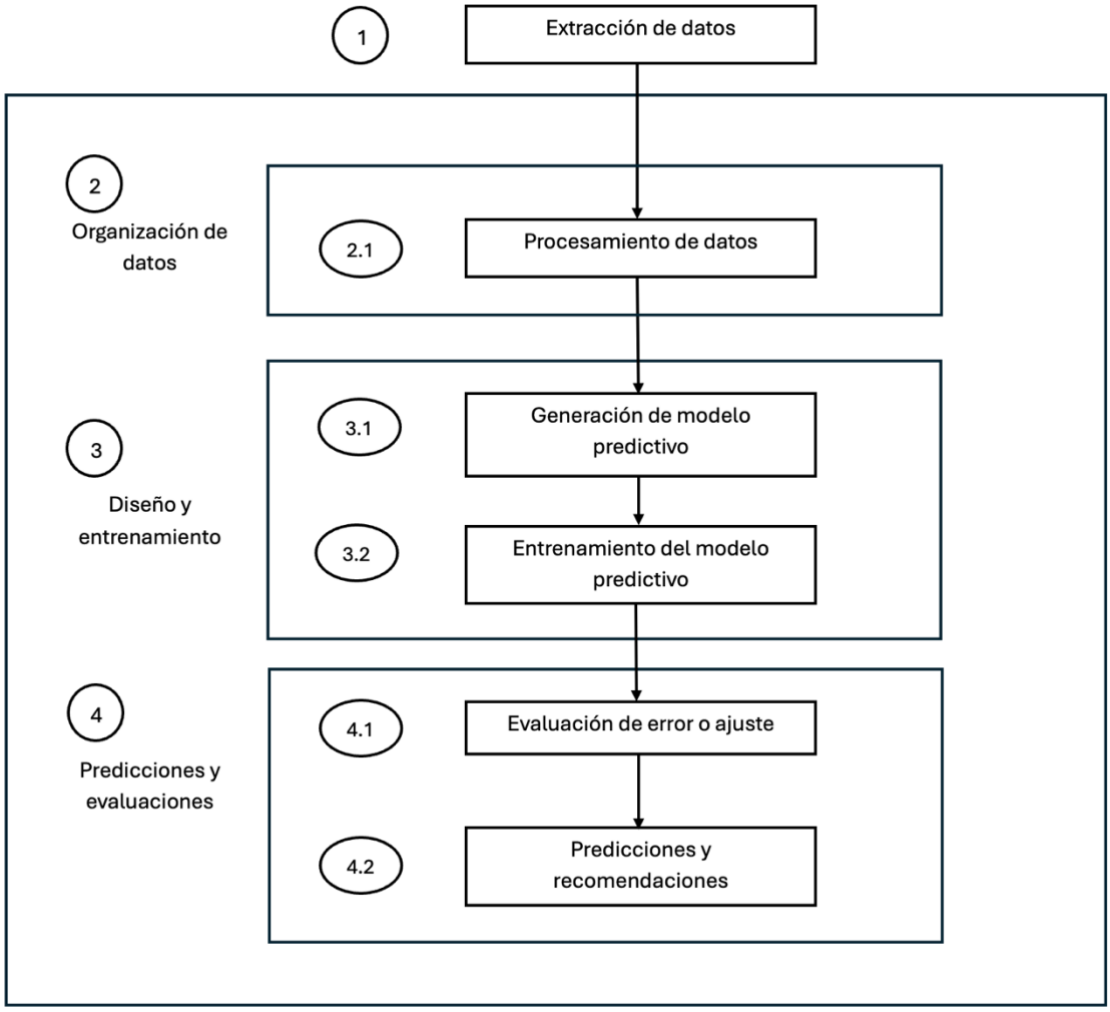


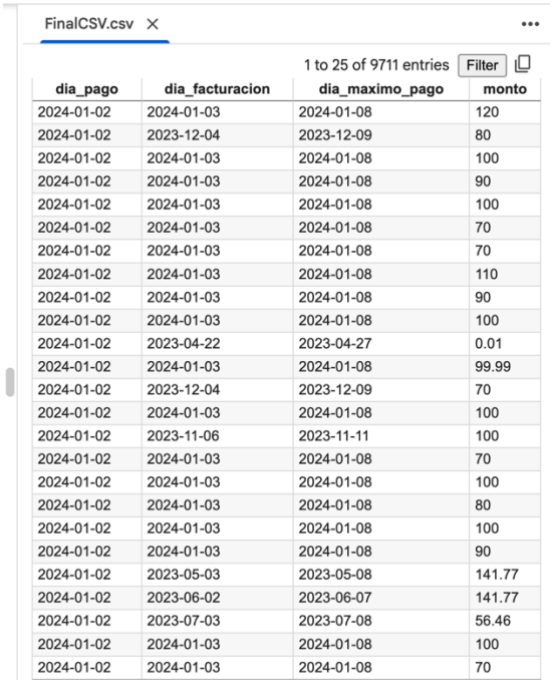
Figura 3. Pasos para el desarrollo del modelo predictivo.

Fase 1: Extracción de datos

En esta fase se realiza la recolección de datos usando una base de datos de pagos históricos correspondientes al 2024 y 2025 de un departamento de cobranza de un colegio particular.

Fase 2: Organización de datos

Teniendo en cuenta esta información, se ha tenido en cuenta la fecha de facturación, y día máximo de pago como variables importantes en la cobranza, ya que lo que se necesita es recuperar cartera en el mes establecido. La información obtenida son 9711 registros. En la figura 4 podemos observar una previsualización de los datos desde Google Colab, herramienta usada para el desarrollo del algoritmo, en este punto la información ya está ordenada lista para ser usada en el entrenamiento del algoritmo.



FinalCSV.csv

1 to 25 of 9711 entries Filter

dia_pago	dia_facturacion	dia_maximo_pago	monto
2024-01-02	2024-01-03	2024-01-08	120
2024-01-02	2023-12-04	2023-12-09	80
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2024-01-03	2024-01-08	90
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2024-01-03	2024-01-08	70
2024-01-02	2024-01-03	2024-01-08	70
2024-01-02	2024-01-03	2024-01-08	110
2024-01-02	2024-01-03	2024-01-08	90
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2023-04-22	2023-04-27	0.01
2024-01-02	2024-01-03	2024-01-08	99.99
2024-01-02	2023-12-04	2023-12-09	70
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2023-11-06	2023-11-11	100
2024-01-02	2024-01-03	2024-01-08	70
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2024-01-03	2024-01-08	80
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2024-01-03	2024-01-08	90
2024-01-02	2023-05-03	2023-05-08	141.77
2024-01-02	2023-06-02	2023-06-07	141.77
2024-01-02	2023-07-03	2023-07-08	56.46
2024-01-02	2024-01-03	2024-01-08	100
2024-01-02	2024-01-03	2024-01-08	70

Figura 4. Previsualización de datos ordenados para el entrenamiento del algoritmo.

Fase 3: Diseño y entrenamiento

En la fase del diseño y entrenamiento tenemos dos pasos que son importantes, la generación del modelo y el entrenamiento de este. Para esta etapa ya el modelo se genera utilizando el método de regresión seleccionado para poder generar la predicción. En el entrenamiento se hace con el dataset organizado con la

información necesaria, en este caso, con la fecha de facturación, fecha máxima de cobranza, fecha de pago.

Fase 4: Predicciones y evaluaciones

En la última fase, predicciones y evaluaciones, el modelo generado y entrenado ya puede otorgar resultados que sean de uso para el usuario. En la evaluación de errores o ajuste, se analizarán los indicadores de error absoluto (MAE), coeficiente de determinación R^2.

Librerías usadas en el desarrollo del modelo predictivo

Python es el lenguaje que se usó para el desarrollo del modelo predictivo, eso conllevo a que se usarán diversas librerías ampliamente reconocidas en el campo del análisis de datos, la visualización y el aprendizaje automático. En primer lugar, se empleó la librería panda, que es fundamental para el análisis de estructuras de datos tabulares mediante la creación de DataFrames. Esta librería permitió cargar el conjunto de datos históricos desde un archivo CSV, así como realizar operaciones de transformación y limpieza sobre columnas de fechas, ¿valores nulos (?, 2022).

Además, se utilizó NumPy para facilitar cálculos numéricos eficientes y operaciones sobre arreglos multidimensionales. Aunque su uso fue más implícito, esta librería sustenta muchas funciones internas de pandas y scikit-learn, ¿y es esencial para el manejo optimizado de datos científicos.

Para la visualización de datos, se emplearon las bibliotecas Matplotlib y Seaborn, las cuales permitieron generar gráficos para el análisis de las variables y comprobar el correcto funcionamiento del modelo, también para demostrar gráficamente las métricas de eficiencias del modelo. La parte predictiva del algoritmo fue desarrollada mediante la librería scikit-learn, una de las más completas para la implementación de modelos de aprendizaje supervisado. Se utilizaron diferentes regresores como LinearRegression (regresión lineal), RandomForestRegressor (bosques aleatorios) y GradientBoostingRegressor (boosting por gradiente).

Adicionalmente, se hizo uso de módulos para seleccionar conjuntos de datos en entrenamiento y prueba (train_test_split), y para evaluar el rendimiento mediante métricas como el error absoluto medio (MAE) y el coeficiente de determinación (R^2)

En conjunto, las librerías permitieron construir un flujo de trabajo confiable e integro que empieza en el análisis de los datos hasta la predicción automatizada de fechas ideales de cobranza, ofreciendo una solución eficiente, reproducible y científicamente validada

6.2 Despliegue en producción en AWS

Despliegue de Contenedor en AWS ECR

En esta sección se analizará el proceso y puntos a tener en cuenta para poder desplegar y alojar el contenedor de nuestra API en los servicios de AWS. El servicio que se usará para alojar el contener es Amazon ECR, que nos permite tener varios repositorios privados y luego usarlos en instancias independiente. Como primer punto, debemos tener listo el archivo Dockerfile en nuestro proyecto, con las indicaciones que va a tener nuestro repositorio para que el ambiente de despliegue sea el correcto.

Figura 5. Archivo Dockerfile configurado

Como se puede observar, en la Figura 4 estamos usando python en su versión 3.10, hay un archivo requirements.txt listo para que se instalen todas las dependencias y

no haya ningún problema de compatibilidad, y como ultimo punto está el comando para que se use como servidor uvicorn para el inicio del API en el punto 8000. Luego de que se crea el contenedor y se puede verificar que funcione de manera local, se procede a realizar configuración en el panel de AWS. Entre los servicios se busca ECR para poder registrar nuestro contenedor y posteriormente usarlo.

Como se puede observar en la Figura 5, una vez dentro del servicio de ECR, se crea el repositorio con el nombre a identificar, y dicho nombre tendrá una URI por defecto, que es la que nos permitirá hacer la conexión y poder subir nuestro contenedor desde nuestro entorno de desarrollo.

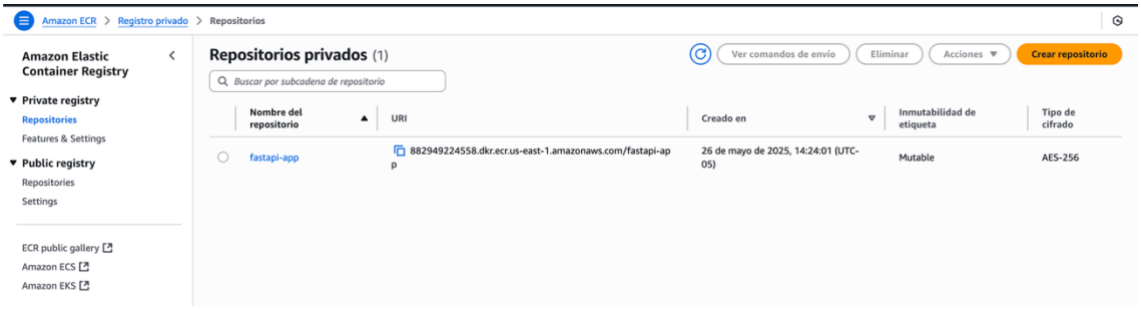


Figura 6. Configuración en ecr

Esa URI, se usará junto con un comando para poder desplegar el contenedor. El primer comando que se debe tener en cuenta es docker tag, mismo comando que primero nos permitirá comprobar la conexión con el contenedor. posterior a eso se podrá realizar un push para que el contenedor que ya hemos creado en nuestro entorno local de desarrollo se despliegue en AWS.

Comando usado:	docker tag <fastapi-app:latest> <882949224558.dkr.ecr.us-east-1.amazonaws.com/fastapi-app:latest>

Figura 8. Comando Dockertag

Comando usado:	docker push <882949224558.dkr.ecr.us-east-1.amazonaws.com/fastapi-cobranza:latest>

Figura 7. Comando dockerpush

El común denominador de estos comandos (Figura 7 y 8) usados es que siempre se debe identificar y tener la URI que AWS nos otorga. Como podemos ver en la Figura 9 Una vez ejecutado estos comandos, podemos visualizar en nuestro contenedor de AWS ya tiene una imagen desplegada con la etiqueta de "lastest".

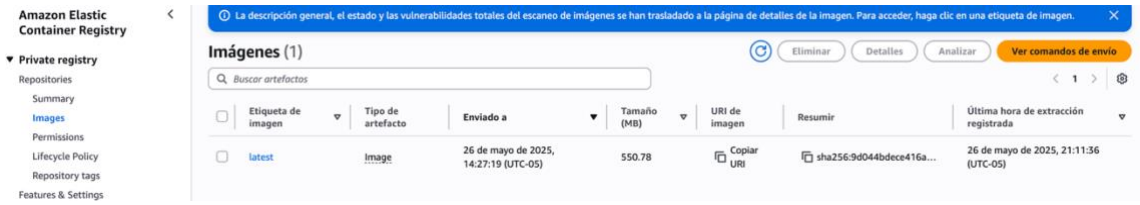


Figura 9. Imagen desplegada en contenedor en aws

Configuración de Balanceador de carga

Para que nuestro servicio esté disponible online y así se pueda usar, es necesario un balanceador de carga, que a la vez ayudará a que el servicio este activo en caso de problemas de seguridad. AWS nos permite crear un balanceador de carga de diferente manera de acuerdo con las necesidades. Para crear un balanceador de carga debemos ir a las opciones de EC2 y en el menú encontraremos "balanceo de carga", como primer punto debemos primero crear un grupo de destino. Dentro de esta opción

El grupo destino es para poder configurar las ip se nos permitirán el balanceo de la aplicación cargando recursos en destino Ipv4. Como se puede ver en la Figura 10 se configura definiendo un nombre a nuestro grupo, eligiendo direcciones ip como tipo de destino. Esa sería la configuración básica para poder usarlo posteriormente en el balanceador de carga.

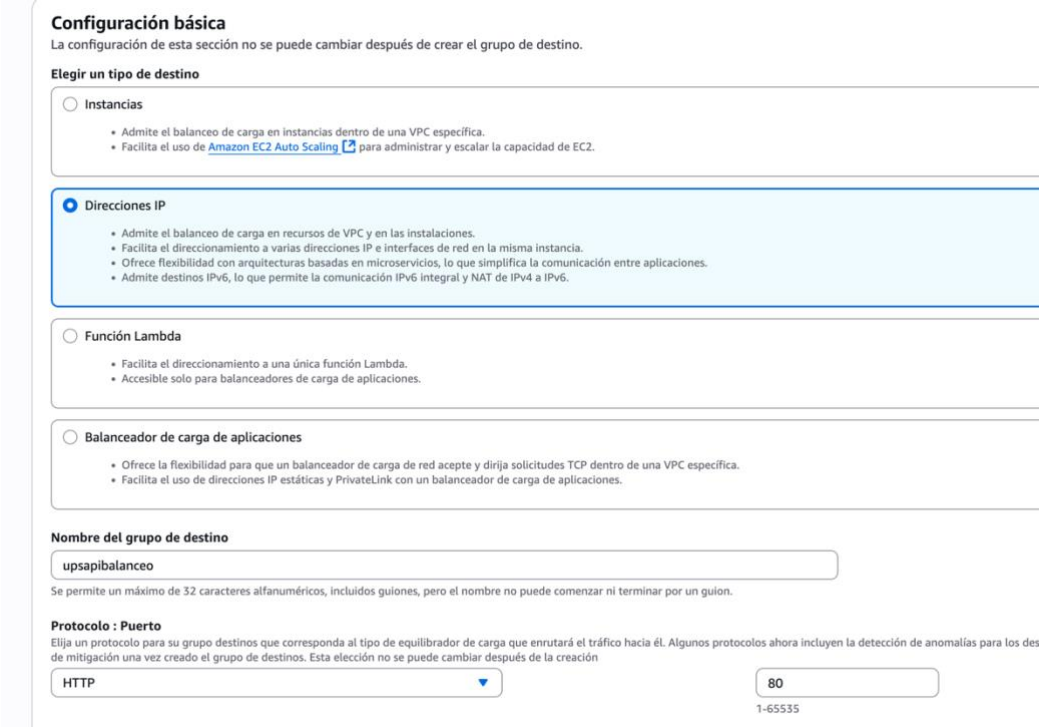


Figura 10. Configuración de grupo destino

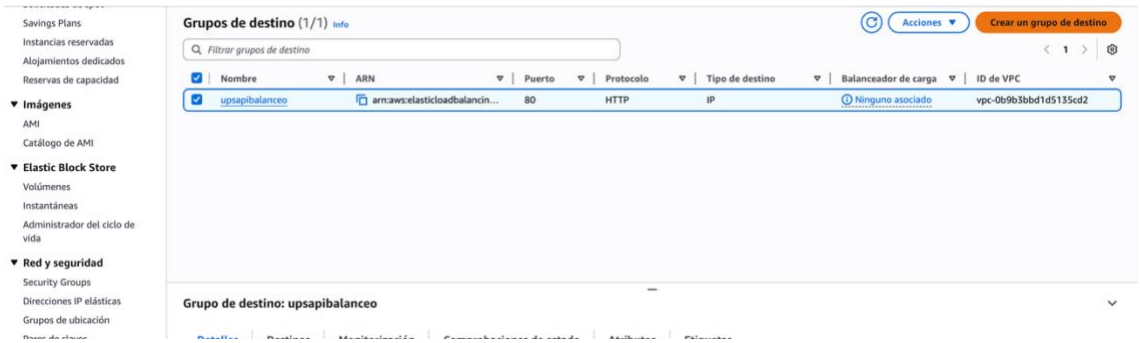


Figura 11. Grupo destino creado y listo para usar

Una vez definido nuestro grupo destino, debemos crear el balanceador como tal. Al momento de ir a la sección de crear balanceador de carga tendremos la opción de elegir entre 3 alternativas (ver Figura 12), balanceador de aplicación, balanceador de carga de red y equilibrador de carga de gateway, la principal diferencia entre estas opciones son la seguridad que te puede brindar cada una, en nuestro caso se elegirá la opción de carga de aplicación, ya que nos brinda características flexibles para las solicitudes http y https.

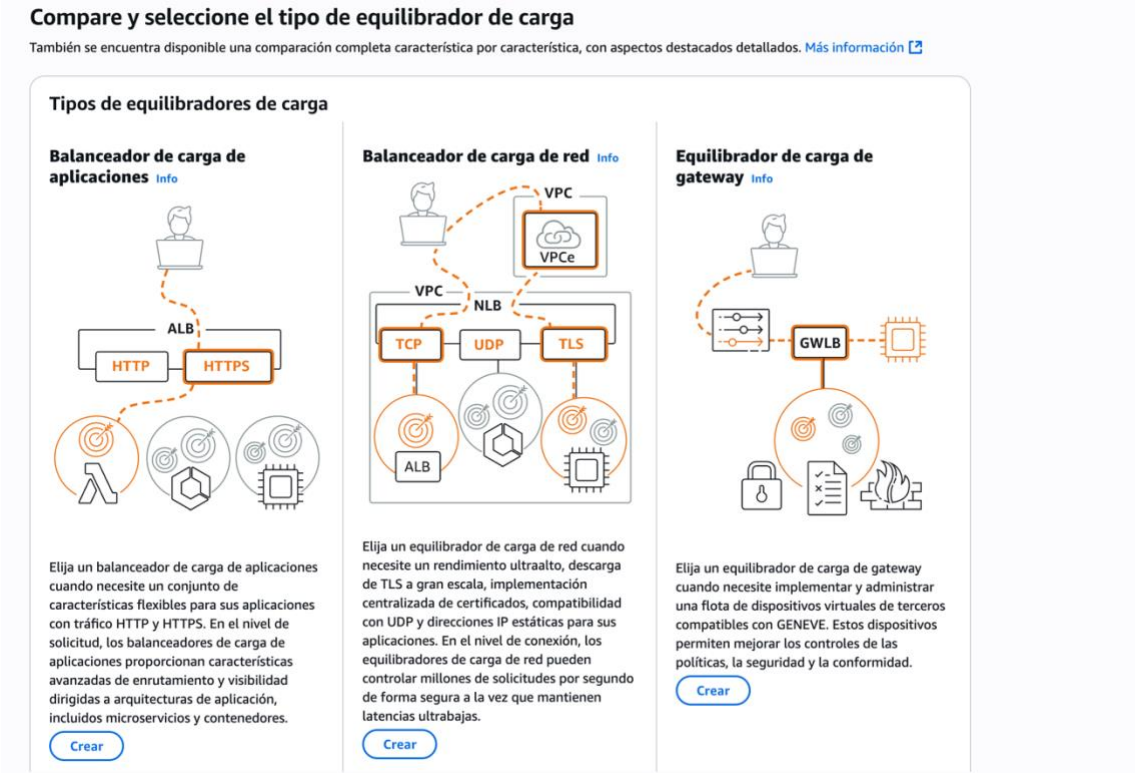


Figura 12. Opciones de balanceo de carga.

En la Figura 13 podemos notar que se debe definir un nombre para nuestro balanceador de carga, en el esquema necesitamos que sea expuesto a internet ya que nuestro objetivo es poder conectarnos a la api y poder hacer consultas.

Figura 13. Configuración dentro del balanceo de aplicaciones

En esta sección, Figura 14, es donde debemos hacer uso del grupo destino que hemos creado, seleccionándolo en la opción de acción predeterminada de nuestros agentes de escucha y direccionamiento.

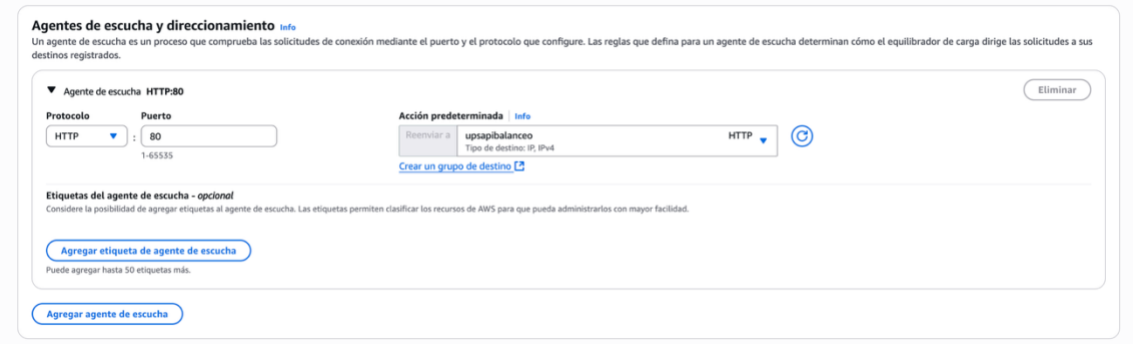


Figura 14. Seleccion de grupo destino creado anteriormente

Una vez se realiza estas configuraciones tendremos listo el balanceador de carga que se usará en nuestro servicio para poder desplegar de manera online nuestro servicio de api. En la Figura 15 podemos visualizar el balanceador creado, y en este punto es importante identificar el "Nombre de DNS" que es la dirección que se usará para acceder al servicio.

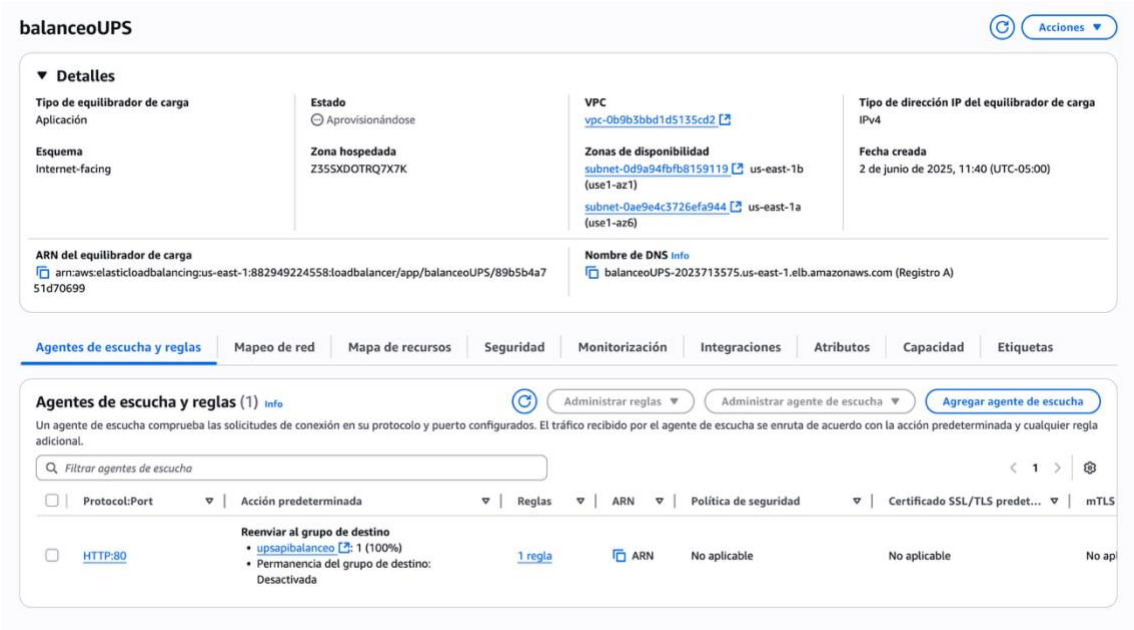


Figura 15. Balanceador de carga creado con éxito

Configuración de ECS

En los servicios de ECS, se debe crear clúster que es el que nos permitirá organizar tareas para poder hacer uso del contenedor de nuestro api, mismo que también nos ayudará a conectar con un balanceador de carga para poder hacer público el api. Al momento de crear el clúster se debe ingresar un nombre y en la infraestructura seleccionar FARGATE, ya que es la arquitectura sin servidor que es la que usaremos. Los clústeres puedes funcionar con servicio de monitoreo CloudWatch que ofrece AWS, pero hay que tener en cuenta que esos servicios tienen un costo que no cubre la capa gratuita de AWS.

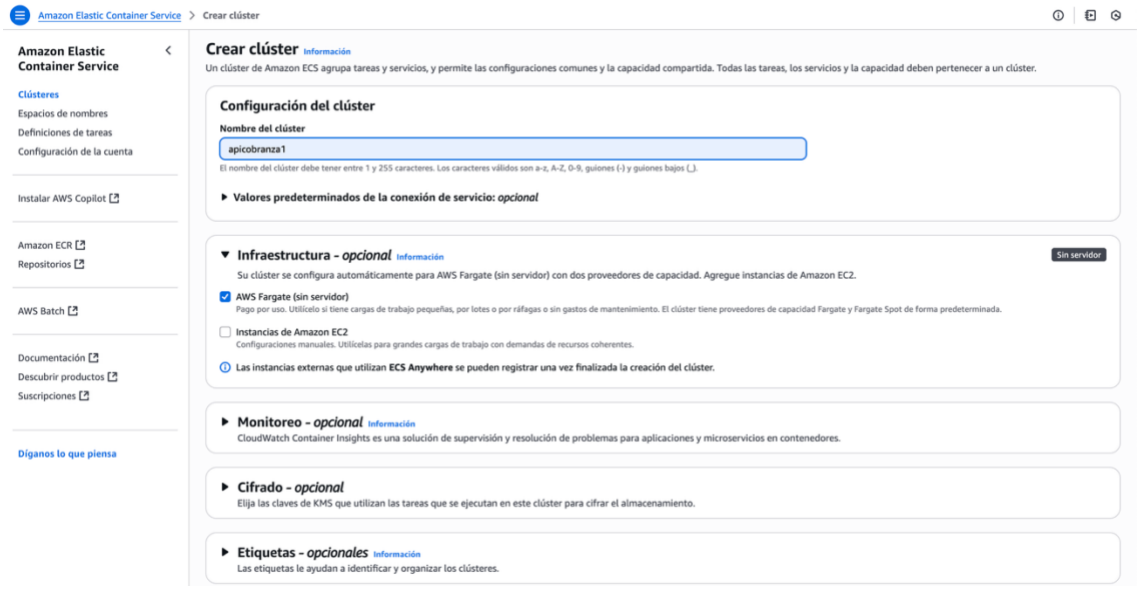


Figura 16. Creación del cluster

En la Figura 17, se puede ver el proceso para crear la tarea que se usará en el servicio del clúster. En este caso se le debe dar un nombre para poder identificarlo, posterior a eso en los requisitos de la infraestructura se debe tener en cuenta en sistema en el que fue desarrollado para evitar inconveniente de compatibilidad. Como nuestro sistema no es muy robusto, con 1 CPU y 5 GB de RAM es suficiente, pero en otros escenarios esas características son editables.

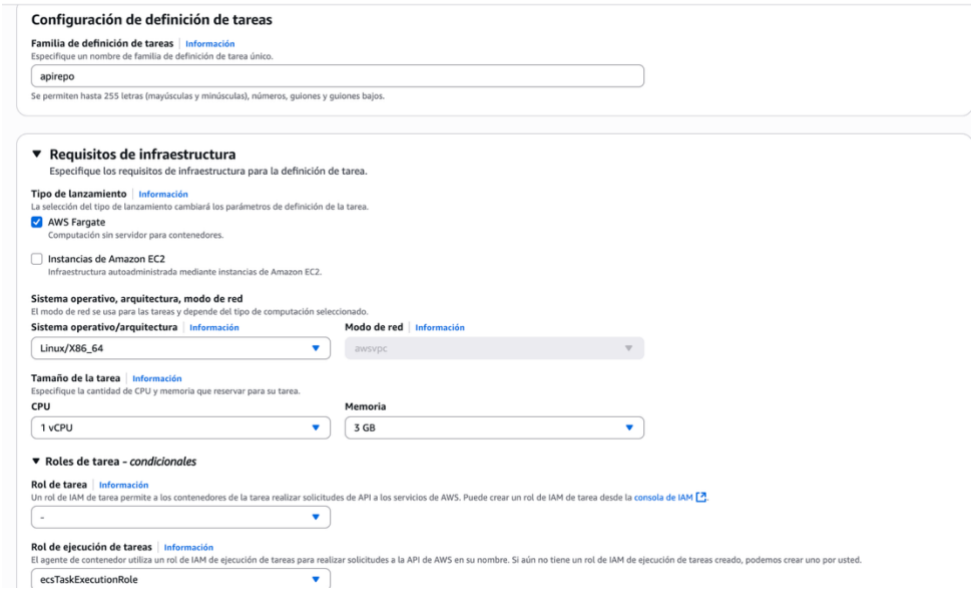


Figura 17. Creación de la tarea

Dentro de la tarea es donde indicaremos que contenedor queremos que se use en el servicio. Una de las opciones que visualizaremos es la de contenedores". En la Figura 18 podemos ver que nos pide información donde pondremos un nombre y la URI que ya previamente hemos obtenido desde nuestro servicio de almacén de contenedores. En nuestro caso no modificaremos más opciones porque estas son las esenciales para que nuestro servicio funcione.

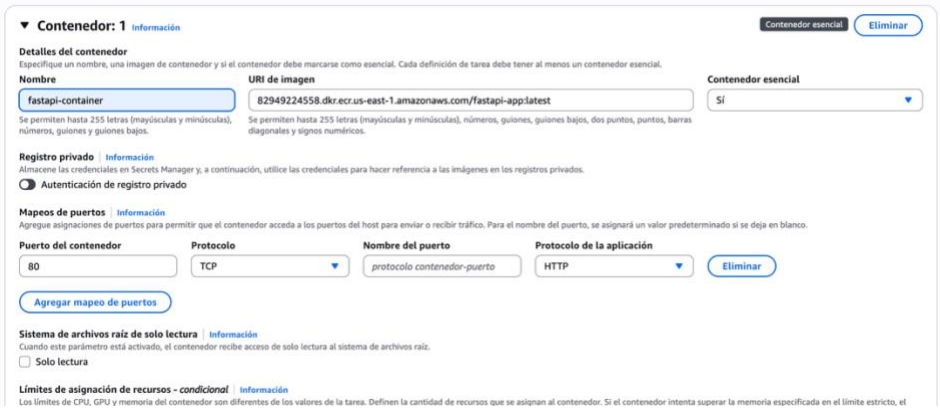


Figura 18. Identificar contenedor en la tarea

Una vez se ha creado la tarea, se procede a regresar al clúster creado anteriormente para poder definir el servicio que desplegará el contenedor. Para poder hacer este pasó ya debemos tener el balanceador de carga creado. En esta sección, Figura 19 se debe usar la tarea previamente creada y en el campo de Nombre de servicio se puede usar el que AWS da por defecto o cambiarlo a uno más personalizado.

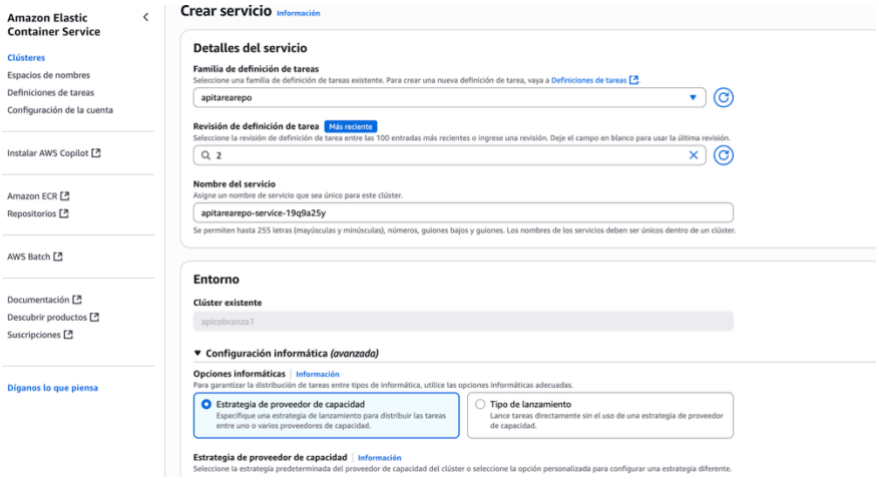


Figura 19. Configuración del servicio a usar en el cluster

Luego se podrá configurar el balanceador de carga, en la Figura 20 debemos seleccionar el balanceador creado, y en los agentes y grupo de destino seleccionar los que ya previamente se han creado, en este caso AWS nos brindará una lista de las instancias que hemos creado antes.

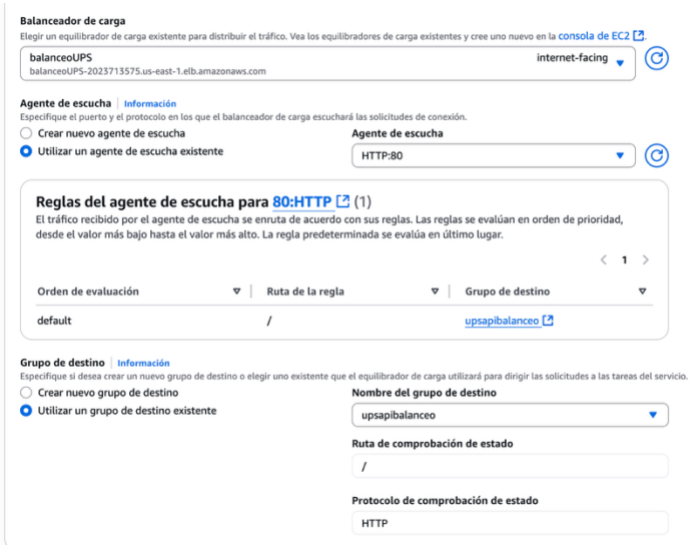


Figura 20. Configurar balanceador de carga en el servicio

Posterior a esto, se guarda la configuración y automáticamente nuestro servicio intentará iniciar la tarea con el contenedor ya instanciado de esa manera podremos acceder gracias al DNS del balanceador. Como podemos visualizar en la Figura 21 estamos accediendo a la documentación swagger de nuestro api desde un navegador usando la dirección DNS.

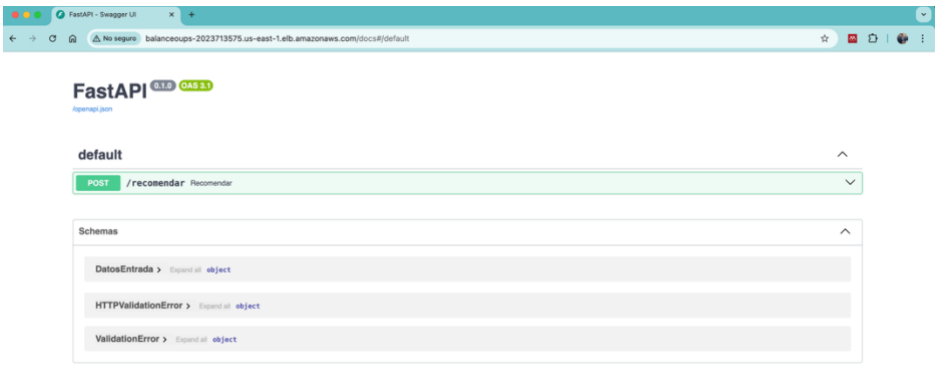


Figura 21. Servicio desplegado en aws

Resultados y discusión

Para realizar las pruebas del modelo predictivo se determinaron variables para que ayuden a que el modelo realice una predicción correcta. Se hace uso de las fechas que se importan desde el archivo de los datos históricos, en este caso se hace uso de la fecha de facturación (día facturación), la fecha máxima de pago (día máximo pago), y de la fecha de pago (día pago). Se ha obtenido un total 9711 registros de pagos, estos pagos son una muestra del 2024 y 2025 de una unidad educativa. Estos registros tienen la información necesaria para que sea procesada en la predicción. En el proceso de depuración de datos se procedió a eliminar los pagos que fueron anticipados y pagados entre los primero 5 días, ya que esos pagos son considerados al día y no habría nada que predecir. En la Tabla 8 se muestran los resultados antes de la depuración. Se puede comprobar que ofrece un margen de error de muchos días de diferencia (30 a 40 días aproximadamente). En cambio, en la Tabla 9, donde ya se realizó la depuración, el margen de error medido en días es de 3 aproximadamente, teniendo en cuenta que son 5 días de espera para el pago, este indicador es considerado tolerable.

Tabla 8. Comparación de MAE entre modelos de regresión antes de depuración de datos

Modelo	**MAE(días)**
Random Forest	30.05
Linear Regression	42.45
Gradient Boosting	34.27
XGboost	30.10

Tabla 9. Comparación de MAE entre modelos de regresión después de depuración de datos

Modelo	**MAE(días)**
Random Forest	3.46
Linear Regression	3.52
Gradient Boosting	3.45
XGboost	3.49

Teniendo en cuenta estos dos escenarios, también se procedió a el análisis de la métrica R^2. Esta métrica mientras más cercana al 1 es mejor. Con esta información podremos ver que antes de la depuración de los datos (Tabla 10), el indicador se mantenía en negativo, mientras que en la Tabla 11, una vez la información fue depurada, ya se encuentra en un número cómodo que denota confianza en el modelo. Si bien es cierto que el indicador se mantiene bajo, es una señal de que la capacidad de los datos debe mejorar.

Tabla 10. Comparación de R^2 entre modelos de regresión antes de depuración de datos

Modelo	**R^2**
Random Forest	-0.10
Linear Regression	-0.23
Gradient Boosting	-0.11
XGboost	-0.13

Tabla 11. Comparación de R^2 entre modelos de regresión después de depuración de datos

Modelo	**R^2**
Random Forest	0,27
Linear Regression	0.01
Gradient Boosting	0.24
XGboost	0.38

Finalizada la depuración de los datos que se han recogido y posteriormente analizados con la ayuda de las métricas de evaluación, se procede a la puesta en producción en la nube para cumplir con los objetivos previamente establecidos.

Analizando los resultados por individual, se mostró que modelos más complejos como Random Forest, Gradient Boosting y XGBoost ofrecieron mejores resultados. En particular, XGBoost alcanzó un MAE de 3.49 días con un R^2 de 0.38, superando de forma consistente al modelo lineal. Este comportamiento indica que existen patrones no lineales en los datos que son mejor capturados por estos modelos avanzados, los cuales son capaces de modelar interacciones complejas entre variables sin necesidad de una transformación manual.

En definitiva, aunque la regresión lineal mostró limitaciones claras en este caso, no debe descartarse por completo. Puede ser utilizada como una base sobre la cual aplicar técnicas más sofisticadas o como referencia para evaluar la ganancia obtenida al utilizar modelos más complejos.

Por otro lado, cuando hablamos de computación en la nube es importante hablar del ahorro que representa. En la Tabla 12 está detallado como aws ha realizado el cobro de los servicios durante las pruebas de desarrollo, cabe recalcar que estos valores son muy bajos porque cuando no se estaban usando las instancias fueron eliminadas y creadas dependiendo de las pruebas, esto permitió que durante la realización de esta tesis los valores a pagar fueran bajos.

Tabla 12. Precio de servicios de AWS en pruebas de desarollo

Servicio	**Horas**	**Dinero ($)**
AWS FARGATE MEMORY	20,998	0,07
AWS FARGATE vCPU	6,999	0,23
AWS VIRTUAL PRIVATE CLOUD	13,843	0,07
TOTAL ($)		0,37

Como se puede observar estamos haciendo uso del servicio de fargate previamente configurado, estos valores van a varias de acuerdo con el uso del servicio y de cuan escalable podría volverse. Se ha realizado un cálculo en 24 horas, ver Tabla 13, y un cálculo de 744 que representarían 31 dias de trabajo continuo, ver Tabla 14. Esto demuestra como los precios a pagar por un servicio en aws serverless pueden ser muy bajos, incluso, se podrían paralizar como en la etapa de desarrollo y de esa manera solo se generarían costos cuando se instancie el contenedor del api.

Tabla 13. Precio estimado de servicios de AWS en 24 horas de uso.

Servicio	**Horas**	**Dinero ($)**
AWS FARGATE MEMORY	24	0,08
AWS FARGATE vCPU	24	0,79
AWS VIRTUAL PRIVATE CLOUD	24	0,12
TOTAL ($)		0,99

Tabla 14. Precio estimado de servicios de AWS en 744 horas de uso(31 días).

Servicio	**Horas(31 días)**	**Dinero ($)**
AWS FARGATE MEMORY	744	2,48
AWS FARGATE vCPU	744	24,49
AWS VIRTUAL PRIVATE CLOUD	744	3,72
TOTAL ($)		30,69

Para poder comparar estos valores se ha analizado un estudio hecho por Eduardo Chica (Chica Bermudez, 2020) , donde se han obtenido valores 3 representativos para un presupuesto básico, ver Tabla 15, donde se considera el salario de un administrador, servidor, y acondicionamiento del lugar donde estará eel servidor. Es un presupuesto básico porque no se ha considerado licencias, y demás espacio físico, pero se puede determinar que el valor sigue siendo superior, incluso solo teniendo en cuenta el salario mensual del administrador.

Tabla 15. Precio estimado de servicios físicos

Servicio	**Cantidad**	**Dinero ($)**
Administrador de servidor	1	1,100
Servidor	1	600
Aire acondicionado del cuarto de servidor	1	200
TOTAL ($)		1800

Conclusiones

La utilización de un modelo de regresión lineal para predecir los días óptimos de cobranza ayuda a un enfoque estadístico que sea entendimiento y de gran utilidad para un equipo dedicado a la finanzas o recuperación de cartera. La regresión lineal nos permite establecer relaciones entre las variables como el monto de la factura, el día máximo de pago, el día se que realizó la fractura, y nuestra variable objetivo que es el día en que se realiza el pago. Los resultados de la predicción no solo ayudan a acercar en el día de cobranza, sino también ayuda a fijarse en la relevancia de cada factor en el comportamiento de pago de los clientes.

Gracias a la generación del algoritmo se contribuye a priorizar automatizar la gestión de cobranza, identificando con tiempo de anterioridad qué facturas tienen mayor probabilidad caer en mora y de esa manera el flujo de dinero podrá ser mas predecible en otros departamentos de una organización. Al tener métricas de evaluación como el error absoluto medio (MAE) o el coeficiente de determinación (R^2), el departamento financiero puede ser objetivo con respecto a la calidad de las predicciones y justificar su uso en la toma de decisiones.

Por otro lado, el despliegue del algoritmo a través de una arquitectura basada en microservicios y tecnologías serverless en AWS amplifica los beneficios del modelo al nivel operativo y tecnológico. Este enfoque permite contar con una infraestructura que se adapte de manera automáticamente a la demanda, lo que garantiza mantener un presupuesto razonable para evitar gastos innecesarios

Finalmente, los servicios serverless disminuye la carga administrativa que tiene que ver con el mantenimiento de servidores, y el algoritmo de regresión lineal en este contexto ofrece un equilibrio entre simplicidad, transparencia y capacidad predictiva que resulta altamente valioso para la gestión financiera.

Glosario

MAE: Error Absoluto Medio. Métrica usada en algoritmos de predicción.

Gradient Boosting: Algoritmo de boosting optimizado y altamente eficiente para tareas de predicción.

Coeficiente de Determinación (R^2): Métrica que explica la proporción de varianza de un modelo.

Boosting: Técnica de potenciación o refuerzo que combina varios modelos débiles para formar uno fuerte.

AWS: Siglas de Amazon Web Services, plataforma de servicios en la nube.

API: Permite la comunicación entre diferentes sistemas de software (Interfaz de Programación de Aplicaciones).

Machine Learning: Es una rama de la inteligencia artifial. (Aprendizaje Automático, en español).

CRM: Sistema administrarivo de interacciones con clientes actuales y potenciales (Gestión de Relaciones con el Cliente, en español).

IA: Inteligencia Artificial,estudia la creación de sistemas capaces de realizar tareas que simulan la inteligencia humana.

Framework: Marco de trabajo que proporciona una estructura y herramientas reutilizables para facilitar el desarrollo de software.

ASGI: Es un estandar para aplicaciones web desarrolladas en python.

ECR: Servicio de AWS para administraciones de imágenes de contenedores.

ECS: Elastic Container Service, servicio de AWS para ejecutar y administrar contenedores Docker.

Dockerfile: Archivo que contiene la configuración para que una imagen docker funcione..

Apendicé

En este capítulo se adjunta el link del repositorio de este proyecto de tesis:
https://github.com/Juanfma10/cobranzaBackend.git

Referencias

Chica Bermudez, E. X. (2020). Análisis de costo total de propiedad (TCO) en un proyecto/inversión TI para el modelo tradicional y su comparación con la implementación en nube pública (IaaS) para las empresas ecuatorianas. Revista Tecnológica – ESPOL.

Daza-Caicedo, S. M., Cárdenas-Rodríguez, G., & Gaitán-Angulo, G. J. (2020). Arquitectura de microservicios: fundamentos, características, ventajas y desventajas. Revista Vínculos, 17(2), 101–112. https://revistas.ufps.edu.co/index.php/vinculos/article/view/3127

de Tyler, C., Graell, R. G., & T. R., C. E. (2023). La administración empresarial y la utilización de la inteligencia artificial y GPT-4: aportes y desafíos para la ingeniería del software y los sistemas de información. Revista Científica Guacamaya, 8(1), 128–141. https://doi.org/10.48204/j.guacamaya.v8n1.a4323

González-Rodríguez, M., García-Mateos, G., & Muñoz-Merino, P. J. (2020). Evaluación de algoritmos de regresión para predicción de series temporales en educación. Revista Iberoamericana de Tecnologías del Aprendizaje, 15(3), 187–194. https://doi.org/10.1109/RITA.2020.3016307

González-Ríos, H., Sánchez-Rodríguez, F., & Rincón-Díaz, N. (2022). Análisis comparativo de algoritmos de aprendizaje supervisado para la predicción de datos financieros. Revista Ciencia en Desarrollo, 13(2), 121–134. https://doi.org/10.19053/01217488.v13.n2.2022.13791

Goodfellow, I., Bengio, Y., & Courville, A. (2021). Deep Learning. MIT Press.

Gómez, J. M. (2021). Gestión de cobranza y su impacto en la gerencia financiera de la empresa PP S.A: Periodo 2014-2016. Puriq, 3(1), 151–164. https://doi.org/10.37073/puriq.3.1.121

Hashem, I. A. T., Yaqoob, I., Marjani, M., & Gani, A. (2021). The rise of 'big data' on cloud computing: Review and open research issues. Information Systems, 101, 101924. https://doi.org/10.1016/j.is.2021.101924

Inbux. (2023). 5 desafíos comunes en la cobranza para cooperativas. Recuperado el 27 de mayo de 2025 de https://inbuxweb.com/5-desafios-comunes-en-la-cobranza-para-cooperativas

Martínez-Rodríguez, J., Medina-Cárdenas, E., & Sánchez-Correa, V. (2020). Comparación de modelos de regresión para la predicción de series temporales económicas. Ingeniería y Universidad, 24(2), 199–218. https://revistas.javeriana.edu.co/index.php/iyu/article/view/30369

McCollect. (2023). Los desafíos de la cobranza en el sector financiero. Recuperado el 27 de mayo de 2025 de https://mccollect.com.mx/2023/10/31/los-desafios-de-la-cobranza-en-el-sector-financiero

Moonflow. (2020). Cómo automatizar la cobranza y mejorar el flujo de caja. Recuperado el 27 de mayo de 2025 de https://www.moonflow.ai/es-pe/blog/desafios-gerentes-de-cobranza

National Institute of Standards and Technology. (2020). NIST cloud computing reference architecture (Special Publication 500-292 Revision 1). https://doi.org/10.6028/NIST.SP.500-292r1

Pineda-Bautista, D., Herrera-Cely, I., & Torres-Martínez, J. (2021). Predicción de la demanda de productos mediante regresión lineal y aprendizaje automático. Revista Colombiana de Estadística, 44(1), 105–122. https://doi.org/10.15446/rce.v44n1.90220

Ramírez-Gallego, S., García, S., & Herrera, F. (2021). Revisión crítica sobre métricas de evaluación para modelos de regresión en ciencia de datos. Revista de Métodos Cuantitativos para la Economía y la Empresa, 31, 28–48. https://www.upo.es/revistas/index.php/RevMetCuant/article/view/5660

Richardson, C. (2021). Microservices patterns: With examples in Java. Manning Publications.

Rouse, W. (2023). Bigger Pictures for Innovation: Creating Solutions, Managing Enterprises & Influencing Policies. Routledge.

Russell, S. J., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

Taibi, D., & Lenarduzzi, V. (2020). On the definition of microservice bad smells. Journal of Systems and Software, 167, 110610. https://doi.org/10.1016/j.jss.2020.110610

Tribuna, L. (2024). Identifican la gestión de cobranza como clave para evitar la quiebra de las empresas. Recuperado el 27 de mayo de 2025 de https://www.latribuna.cl/economia/2024/08/20/identifican-la-gestion-de-cobranza-como-clave-para-evitar-la-quiebra-de-las-empresas.html

Vásquez, F. G. Z. (2023). La importancia de la inteligencia artificial en las comunicaciones en los procesos de marketing. Vivat Academia, 155, 105–123. https://doi.org/10.15178/va.2023.155.105-123