UC | Facultad de Ciencias

DESARROLLO DE UNA APLICACIÓN PARA LA GENERACIÓN Y EJECUCIÓN AUTOMATIZADA DE NOTEBOOKS EN MÚLTIPLES ENTORNOS

(Development of an Application for Automated Generation and Execution of Notebooks Across Multiple Environments)

Trabajo de Fin de Máster

para acceder al

MÁSTER EN INGENIERÍA INFORMÁTICA

Autor: Juan Luis Padilla Salomé

Director: Pablo Abad Fidalgo

Septiembre – 2024

RESUMEN

En la era actual del Big Data, las empresas generan y procesan grandes volúmenes de datos en un corto período de tiempo. Este fenómeno le ha dado mayor visibilidad a la Ciencia de Datos, donde juega un aspecto vital para las empresas, permitiéndoles analizar los datos disponibles para optimizar la toma de decisiones y mejorar la eficiencia en las operaciones.

Uno de los retos de esta nueva tecnología se debe a lo costoso y desafiante que puede resultar la ejecución de estos procesos. Impulsando así a empresas y organizaciones a explorar nuevas maneras de procesar los datos, centrándose en distintos aspectos. La elección del entorno de ejecución es uno de ellos, y es crucial, dado que las empresas pueden optar por la computación en la nube, por sistemas locales (*on-premise*) o entornos híbridos, cada uno con sus respectivas ventajas y desventajas. Esta decisión puede influir significativamente en factores críticos como el rendimiento, los costos y los riesgos operativos.

Este proyecto tiene como objetivo crear una aplicación para una empresa dedicada al mundo financiero, la cual procesa, cada día, ingentes cantidades de datos. Estas ejecuciones consumen mucho tiempo y recursos, por lo que es vital el identificar cuál es la mejor manera de realizarlas con el menor impacto en los fondos de la empresa. Se pretende analizar los costos y la complejidad inherente tanto a la implementación como a la gestión de este programa.

Palabras clave: Ejecución paralela de tareas, ejecución por lotes, computación en la nube, sistemas locales (on-premise)

ABSTRACT

In the current era of Big Data, companies generate and process large volumes of data in a short period of time. This phenomenon has brought greater visibility to Data Science, which plays a vital role for companies by enabling them to analyze available data to optimize decision-making and improve operational efficiency.

One of the challenges of this new technology is the cost and complexity of executing these processes, which drives companies and organizations to explore new ways of processing data, focusing on different aspects. The choice of execution environment is one of these crucial aspects, as companies can opt for cloud computing, on-premise systems, or hybrid environments, each with its respective advantages and disadvantages. This decision can significantly influence critical factors such as performance, costs, and operational risks.

The objective of this project is to create an application for a company in the financial sector, which processes vast amounts of data daily. These executions consume a lot of time and resources, making it vital to identify the best way to perform them with minimal impact on the company's funds. The project aims to analyze the costs and inherent complexity of both the implementation and management of this program.

Key words: Parallel task execution, batch processing, cloud computing, on-premise systems

Índice de contenidos

Índice de Figuras

Índice de Tablas

1 Introducción

En la actualidad, el avance tecnológico se está produciendo a un ritmo acelerado, particularmente en áreas como la Inteligencia Artificial (IA) y el Aprendizaje Automático (*Machine Learning* o ML). Estos campos, que han cobrado relevancia en la actual era del Big Data, permiten la generación y el procesamiento de grandes volúmenes de datos en períodos de tiempo relativamente cortos. Este logro ha sido impulsado por el crecimiento exponencial de la tecnología y la digitalización, facilitando la generación, el almacenamiento y procesamiento de estos datos. Y es gracias a estos avances, que la Ciencia de Datos (*Data Science*) comenzó a ganar popularidad en la década de los 90 y a consolidarse a principios de los 2000 [1].

La Ciencia de Datos emerge como un aspecto crucial para las organizaciones en esta era digital. Aquellas entidades que han logrado integrarla eficazmente en su modelo de negocio han experimentado mejoras significativas en sus procesos de predicción, optimización de la toma de decisiones, y han logrado una comprensión más profunda de sus operaciones y entorno [2][3]. Esta disciplina, que combina conceptos de matemáticas, estadística y programación, se ha convertido en una herramienta indispensable para aprovechar el potencial oculto en la avalancha de datos que las empresas manejan en la actualidad.

Este trabajo de Fin de Máster se centra, en particular, en *Kaiju Capital Management Limited*, un fondo de cobertura financiera (*hedge fund*) que realiza inversiones activas en los principales índices bursátiles de Estados Unidos [4], como el S&P500 y Nasdaq, así como en índices secundarios como NYSE Composite y AMEX Composite, entre otros. La entidad cuenta con una base de datos extensa, que alberga información exhaustiva acerca del mercado bursátil durante las últimas dos décadas. Dicha información es utilizada por la compañía para examinar y analizar tendencias históricas, proporcionando una base sólida sobre la cual realizar estimaciones informadas con respecto a los futuros movimientos del mercado. En este sentido, la Ciencia de Datos se convierte en un recurso estratégico para este fondo de cobertura financiera, ya que su uso adecuado puede favorecer la rentabilidad de las inversiones realizadas y agilizar la toma de decisiones.

En el núcleo de las operaciones de la compañía están los analistas cuantitativos o *quants*, especialistas en ciencia de datos que desarrollan algoritmos y modelos de datos con fines predictivos. Emplean herramientas de programación interactiva, como los *Jupyter notebooks* o scripts de *R*, y frecuentemente necesitan aplicar sus algoritmos a una amplia variedad de empresas y a lo largo de diversos periodos temporales. De esta forma, se pone de manifiesto la necesidad de crear y ejecutar tantos *notebooks* como diferentes parámetros se quieran analizar, lo que introduce un nivel de complejidad adicional en su labor.

Un caso concreto que se pretende abordar es cuando un *quant* crea un *notebook* para aplicar un algoritmo específico sobre los datos de una compañía que cotiza en la bolsa durante un periodo determinado. Si el analista aspira a replicar este análisis en un con-

junto más amplio de empresas, aproximadamente 10.000, esto supondría la creación de 10.000 *notebooks* independientes. Esta cifra corresponde al número de empresas incluidas en la base de datos de la compañía, las cuales han cotizado en la Bolsa de Nueva York en los últimos 20 años.

La creación de los notebooks no es ni mucho menos trivial, debido a que cada análisis requiere ajustes particulares, como el símbolo bursátil de cada empresa (por ejemplo, *Apple*: AAPL, *Microsoft*: MSFT) así como distintas fechas y valores que son propias de cada una de ellas (*IPO*[1], *splits*[2], *dividends*[3], etc.). Además, cada uno de estos *notebooks* requiere ser ejecutado y procesado, lo que puede ser extraordinariamente exigente en términos de tiempo y recursos, especialmente si se lleva a cabo en un equipo personal. Este procedimiento puede prolongarse durante días o incluso semanas, representando un problema en términos de eficiencia. Ante esta situación, la organización se ve en la necesidad de explorar y evaluar soluciones tecnológicas que permitan un manejo más eficiente y rentable de este proceso, optimizando el equilibrio entre rendimiento y costos asociados.

Actualmente, la empresa gestiona todos sus servicios a través de soluciones en la nube. Sin embargo, con la inminente incorporación de este nuevo proyecto, se ha detectado que los costos asociados han alcanzado un nivel que exige la consideración de alternativas tales como el *colocation*. Esta opción consiste en que las organizaciones instalan y operan sus propios servidores y hardware en un espacio alquilado en un centro de datos físico, propiedad de un tercero. Esta estrategia puede ofrecer un mayor control sobre los recursos y una reducción de los costos operativos a largo plazo, especialmente en escenarios donde el uso intensivo de la infraestructura eleva significativamente los gastos en la nube. Un ejemplo notable es el de Dropbox, que, al migrar la mayor parte de su infraestructura de AWS a centros de datos de *colocation*, logró un ahorro en sus gastos operativos de 74,6 millones de dólares en los siguientes dos años [4].

El objetivo de este trabajo se estructura en varias fases. En primer lugar, se propone desarrollar el software previsto, asegurando su capacidad de ejecución tanto en la nube, dentro del ecosistema del proveedor seleccionado, como en entornos locales o servidores bajo demanda. Una vez configurados ambos entornos, se realizará una evaluación exhaustiva de los costos asociados a las ejecuciones en la nube, con el fin de determinar si, en algún momento, es económicamente viable considerar alternativas como el *colocation*. Esta evaluación incluye un análisis detallado de factores como la capacidad de cómputo, la eficiencia, la seguridad de los datos, el costo total de propiedad y la escalabilidad, entre otros.

1.1.Motivación

La empresa financiera, que cuenta con más de siete años en el mercado, se alinea con la tendencia predominante en las empresas nuevas de confiar en los servicios en la nube para el desarrollo y despliegue de sus productos. Esta elección se fundamenta en la efi-

[1] Una Oferta Pública Inicial es cuando una empresa privada vende acciones al público por primera vez.
[2] Cuando una empresa divide sus acciones existentes en múltiples acciones.
[3] Pagos distribuidos a los accionistas a partir de las ganancias de la empresa.

ciencia de costos, ya que los servicios en la nube eliminan la necesidad de grandes inversiones iniciales en infraestructura física y permiten un escalado de recursos que se adapta a las necesidades operativas. Además, liberan a la empresa de las responsabilidades asociadas con el mantenimiento de infraestructura de TI, permitiéndoles centrarse en su negocio principal y mantenerse competitivos al proporcionar acceso a tecnología avanzada a una fracción del costo.

La empresa obtiene sus datos principalmente de instituciones financieras importantes de Estados Unidos, que registran todas las transacciones ocurridas en la bolsa de valores. La adquisición y manejo de estos datos implica un gasto considerable para la empresa, tanto por el costo de adquisición como por los costos asociados al almacenamiento y preprocesamiento de la información en la nube. Con el volumen de datos en constante crecimiento, y ya superando la escala de *petabytes*[4], la empresa se enfrenta a desafíos en términos de costos de almacenamiento y procesamiento.

A pesar de las ventajas que ofrece la nube, también existen limitaciones, como la dependencia del proveedor y las limitaciones en la personalización de los servicios. Por lo tanto, surge la necesidad de explorar alternativas viables que puedan reducir los costos operativos sin afectar la capacidad de procesamiento y la flexibilidad que actualmente ofrece la nube. En este contexto, se consideran varias estrategias: continuar con el uso exclusivo de servicios en la nube o migrar hacia una infraestructura completamente local.

Esta última alternativa podría ofrecer ahorros significativos a la empresa, especialmente en escenarios de uso intensivo. Sin embargo, esta opción también conlleva mayores responsabilidades en términos de gestión y mantenimiento de la infraestructura, lo que requiere un análisis cuidadoso para asegurar que los beneficios superen los desafíos. Con el objetivo de evaluar la viabilidad de una infraestructura local, este Trabajo de Fin de Máster (TFM) se propone replicar en servidores locales, proporcionados por la Universidad de Cantabria (UC), la infraestructura necesaria para ejecutar el software diseñado inicialmente para la nube.

Ante el rápido crecimiento en el volumen de datos y la necesidad de procesarlos de manera eficiente, la empresa enfrenta el desafío de optimizar sus costos operativos mientras mantiene la capacidad de procesamiento y la flexibilidad que la nube ofrece. Este TFM tiene como objetivo proporcionar un análisis exhaustivo de las opciones disponibles, desarrollando una herramienta que permita a la empresa evaluar y optimizar sus operaciones de procesamiento de datos, minimizando los costos y maximizando la eficiencia.

1.2. Objetivos

El principal objetivo de este TFM es desarrollar una aplicación capaz de generar y ejecutar notebooks de manera automatizada en múltiples entornos, específicamente en la nube (AWS) y en servidores locales bajo demanda. Esta aplicación tiene como finalidad optimizar el procesamiento de grandes volúmenes de datos, asegurando tanto la eficien-

[4] *1,000 TBs* o *1,000,000 GBs*.

cia operativa como la reducción de costos para la empresa. El resto del documento se organiza de acuerdo con los siguientes capítulos:

- Capítulo 2. En este capítulo se describe detalladamente las herramientas empleadas a lo largo de todo el proyecto, abarcando tanto las tecnologías como las metodologías adoptadas. Además, se proporciona una descripción del punto de partida del proyecto y se define la meta que se plantea alcanzar.
- Capítulo 3. Aquí se presenta una descripción exhaustiva de la librería desarrollada, incluyendo las tecnologías y arquitecturas usadas para configurar la solución en la nube y en un servidor *on-demand*. También se analiza el comportamiento de la librería en ambos escenarios, destacando las diferencias clave y los desafíos técnicos abordados.
- Capítulo 4. En este capítulo se realiza un análisis exhaustivo y una evaluación detallada de los costos asociados con dos opciones de infraestructura: la ejecución en AWS y el uso de una infraestructura propia en un centro de *colocation*. Se describe la metodología empleada para la estimación de costos en AWS y se examinan los distintos costos involucrados en la implementación y mantenimiento de una infraestructura local. Además, se exploran los diferentes tipos de gastos, y se presenta una comparativa objetiva de los costos totales de ambas opciones, proporcionando una base sólida para la toma de decisiones estratégicas.
- Capítulo 5. En el capítulo final, se presentan las conclusiones derivadas de la investigación y valoración de las opciones consideradas. Se proponen recomendaciones basadas en los hallazgos obtenidos y se ofrecen pautas para continuar esta investigación en el futuro.

2 Concepto y herramientas utilizadas

Partiendo de los objetivos marcados anteriormente, este capítulo proporciona una breve descripción de los diferentes componentes del framework completo con el que se pretende trabajar (desde el interfaz con el que trabajarán los usuarios hasta la orquestación de los distintos servicios de AWS que entran en juego y que permanecen ocultos al usuario final), así como un análisis de los requisitos especificados por la organización.

2.1. Interfaz de Usuario (Notebooks)

En el contexto de la ciencia de datos y la programación, un notebook se refiere a un entorno de desarrollo interactivo donde los usuarios pueden combinar código ejecutable, texto enriquecido, gráficos y otros elementos multimedia en un solo documento. Este formato facilita la exploración de datos y el análisis iterativo, permitiendo tanto la visualización de resultados como la documentación del proceso en un formato que se puede compartir y reproducir fácilmente. Existen diversas herramientas que permiten la combinación de código, texto y visualizaciones, como *Jupyter Notebooks*, *R Markdown* y *Apache Zeppelin*, todas ellas soportan múltiples lenguajes de programación como *Python*, *R*, *Scala* y *Julia*.

En este Trabajo de Fin de Máster, nos centraremos en los *Jupyter Notebooks*, herramienta que es comúnmente usada por los *quants* de esta organización. Estos documentos de código abierto, utilizados ampliamente en la computación científica, análisis de datos y educación interactiva, permiten crear y compartir material que incluye desde código en vivo hasta visualizaciones dinámicas y narrativas detalladas, facilitando así un entorno robusto para el análisis técnico y la colaboración.

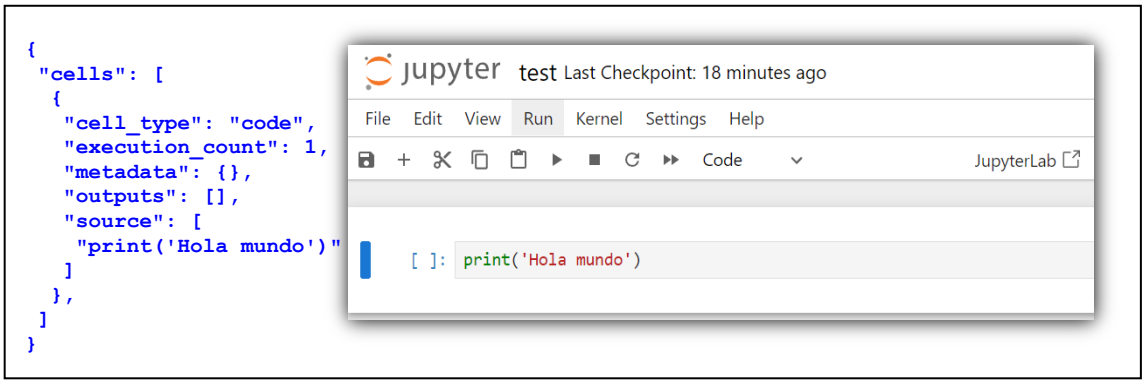

Figura 2-1: Estructura interna de un Jupyter notebook junto a su representación gráfica.

Un *Jupyter Notebook* se guarda como un archivo con extensión *.ipynb*, que en su esencia es un documento en formato *JSON*. Este formato estructura la información en pares de clave-valor, lo que permite que cada celda del notebook se represente como un conjunto de claves que describen su tipo (código o *markdown*), contenido, y metadatos asociados.

Por ejemplo, en la Figura 2-1 se muestra un *Jupyter Notebook* en su formato *JSON* y, a su lado, en su representación visual. Esta estructura permite que los notebooks sean leí-

dos como archivos de texto normales y se manipulen programáticamente para poder ser modificados o generar nuevos notebooks, algo que será útil y se hará uso en la librería desarrollada para replicar los notebooks de entrada.

2.2. Amazon Web Services

Amazon Web Services (AWS) es una plataforma de servicios en la nube ampliamente utilizada que ofrece una infraestructura global y más de 200 servicios completamente integrados desde centros de datos distribuidos en todo el mundo. Estos servicios abarcan una amplia gama de aplicaciones, desde almacenamiento y bases de datos hasta análisis de datos, inteligencia artificial y computación sin servidor. La flexibilidad y escalabilidad de AWS permiten a las organizaciones optimizar sus operaciones, reducir costos y centrarse en su negocio principal, manteniéndose competitivos.

En este proyecto, y siguiendo la directriz impuesta por la empresa financiera, se emplearán diversos servicios de AWS que han sido seleccionados específicamente para satisfacer las necesidades del sistema que se va a desarrollar.

2.2.1. Amazon Simple Storage Service (S3)

El *Amazon Simple Storage Service*, conocido también como Amazon S3, es una solución de almacenamiento de objetos, presentando una escalabilidad eficiente que se adapta a requerimientos de almacenamiento dinámicos. Además, ofrece óptima disponibilidad de datos, lo cual es fundamental para el acceso continuo a la información [6]. En nuestro caso, este servicio se empleará como medio de comunicación entre el usuario y los programas que se deseen ejecutar, siendo también el lugar donde se subirán los resultados de dichos programas.

2.2.2. AWS Lambda

A su vez, *AWS Lambda* es una solución de computación sin servidor que permite la ejecución de código en respuesta a eventos sin la necesidad de aprovisionar o administrar servidores. Por su alta escalabilidad y adaptabilidad a las variaciones en la carga de trabajo, resulta especialmente apto para entornos dinámicos. *AWS Lambda* es ampliamente versátil, aplicable a escenarios como el procesamiento en tiempo real de datos, la automatización de tareas, el desarrollo de aplicaciones sin servidor, y en áreas especializadas como la *Internet de las Cosas* (IoT) y el análisis de datos [7]. Es la respuesta a eventos la que usaremos para poder distinguir cuando se han de ejecutar los *notebooks* o no.

2.2.3. Amazon DynamoDB

Amazon DynamoDB es un servicio de base de datos *NoSQL* que ofrece rendimiento y escalabilidad de manera consistente. Este servicio está optimizado para manejar grandes volúmenes de datos con estructuras diversas, y es conocido por su capacidad de escalar automáticamente para acomodarse a las demandas de tráfico fluctuantes, garantizando tiempos de respuesta rápidos incluso en condiciones de alta carga. Un aspecto distintivo de *DynamoDB* es su modelo de facturación flexible, que permite a los usuarios optar por un modelo de capacidad provisionada o capacidad bajo demanda, lo cual facilita la optimización de costos según los patrones de uso [8]. En este proyecto, *DynamoDB* se

utilizará para registrar y monitorizar el estado de los trabajos, permitiendo un seguimiento en tiempo real de los procesos ejecutados en AWS.

2.2.4. AWS Batch

Por otro lado, *AWS Batch*, es una solución diseñada para facilitar y optimizar la ejecución de trabajos por lotes y tareas de computación de alta intensidad. Este servicio se caracteriza por la capacidad de planificar, organizar y ejecutar eficientemente cargas de trabajo por lotes, sin la necesidad de que los usuarios administren infraestructuras de servidores y clústeres [9]. Si bien, para el desarrollo del programa en cuestión, se ha decidido definir manualmente las máquinas EC2 donde se han de ejecutar cada trabajo, con la finalidad de llevar un registro claro del tipo de instancias usadas y así poder calcular de manera precisa el costo de cada ejecución.

2.2.5. Amazon Elastic Compute Cloud (EC2)

El *Amazon EC2*, proporciona a los usuarios la posibilidad de alquilar instancias virtuales de servidores en los que pueden ejecutar aplicaciones y procesar datos. Constituye una solución de computación en la nube altamente elástica, confiable y segura, que se adapta a una diversidad de requisitos y aplicaciones gracias a su capacidad de personalización y su integración con el ecosistema AWS [10]. Y, como se ha comentado, será donde se ejecutarán los trabajos que se lancen.

2.2.6. Amazon Elastic Container Registry (ECR)

Para la ejecución de los trabajos en este proyecto, se va a hacer uso de *Docker*. En concreto, se utilizará una imagen base para crear tantos contenedores como notebooks sea necesario ejecutar, asignando a cada contenedor la ejecución de un notebook específico. Esta elección no es trivial ni está cerrada, pudiendo cambiarse en un futuro, pues al final depende del *notebook* que se va a ejecutar y las operaciones que se van a desarrollar en ella.

Para la definición de estos contenedores se va a usar *Amazon ECR*, un servicio que facilita el almacenamiento, la gestión y la implementación de imágenes de contenedores. Este servicio es esencial para las arquitecturas modernas basadas en contenedores, como las que utilizan *Docker*, y proporciona un repositorio seguro y escalable [11].

2.2.7. Boto3 y AWS CLI

Finalmente, *Boto3* es una biblioteca de SDK para *Python* que facilita la interacción programática con los servicios de AWS, permitiendo la gestión de recursos, automatización de tareas y despliegue de soluciones en la nube de manera eficiente. A través de *Boto3*, se puede acceder a una amplia gama de servicios de AWS, como los anteriores listados, desde un entorno Python.

Por otro lado, AWS *Command Line Interface* (CLI) es una herramienta que permite interactuar con los servicios de AWS directamente desde la línea de comandos, ideal para tareas de administración y monitoreo. Ambas herramientas, Boto3 y AWS CLI, son esenciales para proyectos en la nube que utilicen AWS. En este proyecto, se em-

plean debido a su capacidad para ofrecer un control flexible y mejorar significativamente la eficiencia en el desarrollo y gestión de soluciones.

2.3.Sistema *en propiedad*

Como propuesta alternativa y haciendo uso de los servidores puestos a disposición por el grupo Computer Engineering (https://www.ce.unican.es) perteneciente la Universidad de Cantabria, se procede a implementar una estructura similar a la utilizada por la empresa para su despliegue en AWS. Aunque no es posible, ni se busca, replicar exactamente la misma estructura, se intentará mantener una similitud notable entre ambas para asegurar que las comparaciones sean justas en la mayoría de las situaciones.

A diferencia del enfoque adoptado con el *cloud* de *AWS*, en el cual se utilizaban servicios preexistentes que simplemente requerían una configuración adecuada para que se ejecuten los *notebooks*, en este caso es necesario instalar y configurar todo con un nivel de detalle aún mayor. Este escenario presenta una gama más amplia de opciones, permitiendo la elección entre varias herramientas que puedan cumplir la misma función.

El primer aspecto que se ha priorizado es la identificación de un método eficaz para ejecutar los *notebooks* en los nodos proporcionados por la universidad. Tras considerar diversas alternativas, se ha optado por usar *Slurm* [17], un sistema de gestión de recursos y programación de tareas ampliamente adoptado en entornos de computación de alto rendimiento (HPC). *Slurm* permite organizar y distribuir eficientemente las cargas de trabajo en clústeres grandes, asegurando que los recursos disponibles se utilicen de manera óptima. Su flexibilidad y capacidad de escalado lo convierten en una herramienta ideal para maximizar el rendimiento en entornos con altas demandas computacionales.

Al igual que se utilizaba *boto3* para acceder a los servicios de AWS, en este contexto se emplea *Paramiko* [18]. Esta librería utiliza protocolos de red como SSHv2 y facilita la creación y gestión de conexiones seguras a máquinas remotas para la ejecución de comandos y transferencia de archivos. En este caso, la transferencia de archivos se realizará a una carpeta específica localizada en el Sistema de Archivos en Red (NFS), un protocolo que permite trabajar con archivos en un servidor remoto de manera similar a los archivos locales.

La ejecución de los notebooks se puede realizar de múltiples formas, siendo la más directa el uso de la línea de comandos, previa instalación de un entorno virtual de *Python* con todas las dependencias necesarias. Desafortunadamente, este método ha presentado problemas, ya que el módulo *Jupyter* que se usa para su ejecución crea puertos para cada ejecución, originando conflictos cuando varios trabajos intentan usar el mismo puerto. Una solución podría ser cambiar de *Jupyter notebook* a un script de *Python*, pero esto conlleva la pérdida de contenido y estructura del notebook. Ante este problema, se ha optado por volver a utilizar contenedores *Docker* para ejecutar cada notebook de manera independiente, evitando así posibles errores. Esta decisión plantea la cuestión de dónde almacenar las imágenes que se van a utilizar. Por simplicidad, se va a usar *Docker Registry*, un repositorio de aplicaciones empaquetadas como imágenes de Docker, que sirven de base para ejecutar contenedores. Por razones de seguridad y confidencia-

lidad, se decidió configurarlo como privado. Esta utilidad se utilizará de manera similar a cómo se utilizará el ECR de AWS.

2.4. Análisis de requisitos

En este apartado se va a identificar y describir tanto los requisitos funcionales como los no funcionales que guiarán la implementación y el comportamiento de la librería. El objetivo es asegurar que la herramienta pueda ser utilizada de manera eficiente, integrándose adecuadamente en los entornos previstos y cumpliendo con las expectativas de rendimiento y usabilidad.

2.4.1. Requisitos funcionales

Los requisitos funcionales especifican los servicios y características que la librería ofrecerá, así como su comportamiento esperado en diferentes situaciones. Definen cómo el sistema debe procesar la información, interactuar con otros sistemas y responder a las acciones del usuario. En la Tabla 2-1 se enumeran los servicios que la librería proporcionará y las respuestas esperadas ante distintos eventos y condiciones.

Tabla 2-1: Requisitos funcionales para la librería Parallel-Scripts.

ID	Descripción	Prioridad	Comentarios
RF001	La librería debe permitir al usuario cargar un notebook base y un archivo de parámetros.	Alta	
RF002	La librería debe aceptar archivos de parámetros en formato *.csv*, *.yaml* o *.json*.	Media	Proporciona flexibilidad en la entrada de datos según las preferencias del usuario.
RF003	La librería debe generar automáticamente un notebook por cada conjunto de parámetros en el archivo.	Alta	Facilita la creación masiva de notebooks personalizados.
RF004	La librería debe sustituir los parámetros en una celda específica del notebook plantilla utilizando el tag "*parameters*".	Alta	El tag debe ser definido por el usuario.
RF005	La librería debe guardar el contenido de la variable ubicada en la celda que contiene el tag "*results*".	Alta	Permite capturar los resultados clave de cada ejecución.
RF006	La librería debe agrupar o concatenar los resultados de las celdas que, además del tag "*results"*, contengan el tag "*concatenate*".	Media	Facilita la compilación de resultados en un único output para análisis posterior.
RF007	La librería debe permitir la ejecución de los notebooks en la propia máquina que se ejecuta (local).	Alta	
RF008	La librería debe permitir la ejecución de los notebooks en la nube (AWS).	Alta	Optimiza el uso de recursos escalables en entornos de producción en la nube.
RF009	La librería debe permitir la ejecución de los notebooks en un sistema en propiedad (UC).	Alta	Proporciona flexibilidad para utilizar recursos alternativos a AWS.

Es crucial considerar la forma en que el usuario puede especificar cuáles serán los parámetros y los resultados del notebook (RF004-RF006). Este proceso debe ser sencillo y no intrusivo en el desarrollo de este. Es decir, no se debería requerir al usuario que lo planifique de antemano, ni que sea necesario modificar el código para que funcione.

La herramienta debe tener la capacidad de almacenar una variable del notebook, que podría ser un array de *NumPy* o un *DataFrame* de *Pandas*, ambas librerías ampliamente utilizadas por los analistas de datos (RF005 y RF006). Además, debe ofrecer la posibilidad de concatenar estos elementos. De esta forma, tras la generación y ejecución de cientos de notebooks, será esencial consolidar las listas de cada notebook en una única estructura, una característica muy apreciada especialmente para esta empresa y por los analistas cuantitativos.

Uno de los principales requisitos de la aplicación es que pueda ser ejecutada en la nube (RF007-RF009). Por lo tanto, se debe permitir al usuario lanzar ejecuciones tanto en local como en la nube o un servidor dedicado, como el proporcionado por la *Universidad de Cantabria*.

2.4.2. Requisitos no funcionales

Los requisitos no funcionales describen las características del sistema que, aunque no estén directamente vinculadas a funcionalidades específicas, son esenciales para su rendimiento, usabilidad, seguridad y escalabilidad. En la Tabla 2-2 se detallan los aspectos clave que aseguran el óptimo funcionamiento de la librería en diversos entornos y condiciones.

Tabla 2-2: Requisitos no funcionales para la librería Parallel-Scripts.

ID	Descripción	Prioridad	Comentarios
RNF001	La librería debe ofrecer una función única y clara para el usuario, que automatice el proceso de generación y ejecución de notebooks.	Alta	Mejora la usabilidad, asegurando que el usuario no necesite conocer los detalles internos de la librería.
RNF002	La librería debe estar completamente desarrollada en inglés, ya que está destinada a una empresa estadounidense.	Alta	Asegura la coherencia lingüística con los usuarios finales y la documentación internacional.
RNF003	La librería debe estar optimizada para minimizar el tiempo de generación de notebooks, especialmente para grandes volúmenes de parámetros.	Alta	Rendimiento es clave en escenarios con muchos notebooks.
RNF004	La librería debe ser fácil de instalar mediante *pip*, siguiendo los estándares de distribución de *wheels* en *Python*.	Alta	Facilitar la instalación para usuarios con conocimientos básicos de Python.
RNF005	La librería debe garantizar la seguridad y confidencialidad de los datos de los notebooks y los parámetros utilizados.	Alta	Especialmente relevante al trabajar con datos sensibles en entornos financieros.

Aunque los expertos son competentes y pueden manejar la herramienta suministrada con destreza, el objetivo es minimizar cualquier complicación adicional a sus responsabilidades existentes (RNF001). Esto implica que la herramienta debe ser fácilmente integrable en el flujo de trabajo existente y no requiere un conocimiento profundo para su utilización.

Por motivos de seguridad y privacidad, cada tarea se ejecutará dentro de un contenedor (RNF005). Esto dificulta que los usuarios que puedan acceder al clúster puedan visualizar el contenido que se está procesando. Estos contenedores se crean utilizando imágenes que han sido previamente registradas en Amazon ECR.

2.5.Punto de partida

Independientemente del entorno en el que se ejecuten los notebooks, todas las ejecuciones siguen una serie de pasos iniciales y finales que se realizan de manera local. Estos pasos son cruciales para garantizar la integridad y consistencia de los procesos, asegurando que todos los recursos necesarios estén disponibles antes de la ejecución y que los resultados sean almacenados y verificados al finalizar.

El primer paso fundamental en este proyecto ha sido la creación de una base de datos que registra y monitorea el estado de las ejecuciones de notebooks: desde su creación, pasando por su finalización, hasta aquellas que permanecen pendientes. Aunque en un entorno local esta medida podría parecer innecesaria, en un entorno de nube resulta imprescindible. Las ejecuciones en la nube están sujetas a riesgos como la pérdida de conexión o el cierre abrupto del programa, lo que podría llevar a la pérdida total del trabajo realizado, y con ello, el dinero invertido, generando un impacto negativo tanto en tiempo como en recursos.

Por ello, la base de datos no solo ofrece un registro detallado y actualizado del estado de cada notebook, sino que también permite retomar ejecuciones previas en caso de interrupciones. El diseño actual de la base de datos se ilustra en la Figura 2-2.

Otro beneficio significativo del uso de esta base de datos es la reducción en la cantidad de peticiones realizadas a AWS para verificar el estado de las tareas. Una vez que una tarea ha sido completada en la nube y marcada como tal en la base de datos local, no es necesario seguir solicitando información adicional, lo que optimiza el uso de recursos y mejora la eficiencia general del sistema.

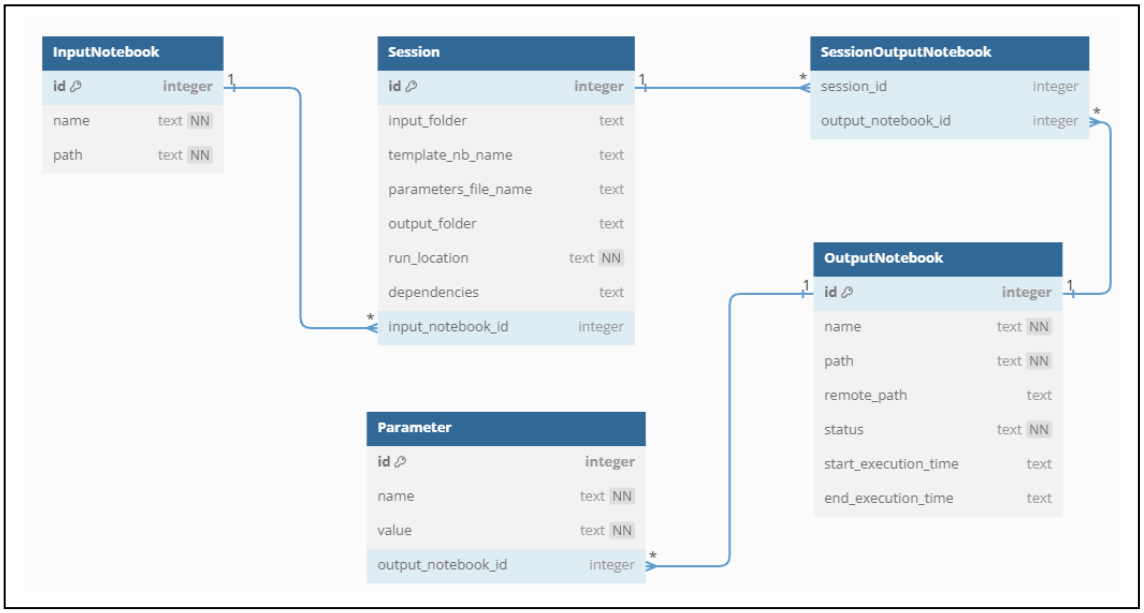

Figura 2-2: Diagrama de la base de datos que muestra la relación entre las distintas tablas usadas.

El siguiente paso en el proceso es la generación de los notebooks. Para ello, se lee el notebook de entrada junto con el archivo de parámetros, según las rutas especificadas por el usuario. Como se explicó en la Sección 2.1, los notebooks son archivos de texto

en formato JSON. Esto permite leerlos como texto plano, identificar las celdas de interés (aquellas que contienen un *tag* específico) y reemplazar dichas celdas con los nuevos parámetros (en el caso del tag "*params*") o con líneas de código que permitan guardar el contenido de la variable como binario (en el caso del tag "*results*").

En cuanto al archivo de entrada, la herramienta admite formatos como *Comma Separated Value* (CSV), JSON y YAML, seleccionando el que sea más conveniente y familiar por el usuario. A partir de este archivo, se extraen los valores necesarios para crear un notebook por cada conjunto de parámetros leído.

Esta tarea se lleva a cabo de manera local, ya que la generación de notebooks es un proceso que, por lo general, no requiere mucho tiempo ni recursos. Para optimizar el rendimiento, se utiliza un pool de threads que distribuye la carga de trabajo entre múltiples hilos de ejecución. Dependiendo de las capacidades de la máquina en la que se ejecuta, este enfoque permite generar un gran número de notebooks en un tiempo muy reducido. De hecho, en la mayoría de los casos, es posible generar aproximadamente 500 notebooks en menos de un minuto.

Tras esta fase de generación, se procede a establecer conexión con los servidores correspondientes para iniciar las ejecuciones. Al finalizar, se descargan los notebooks ejecutados y los archivos de resultados generados, en caso de haberlos.

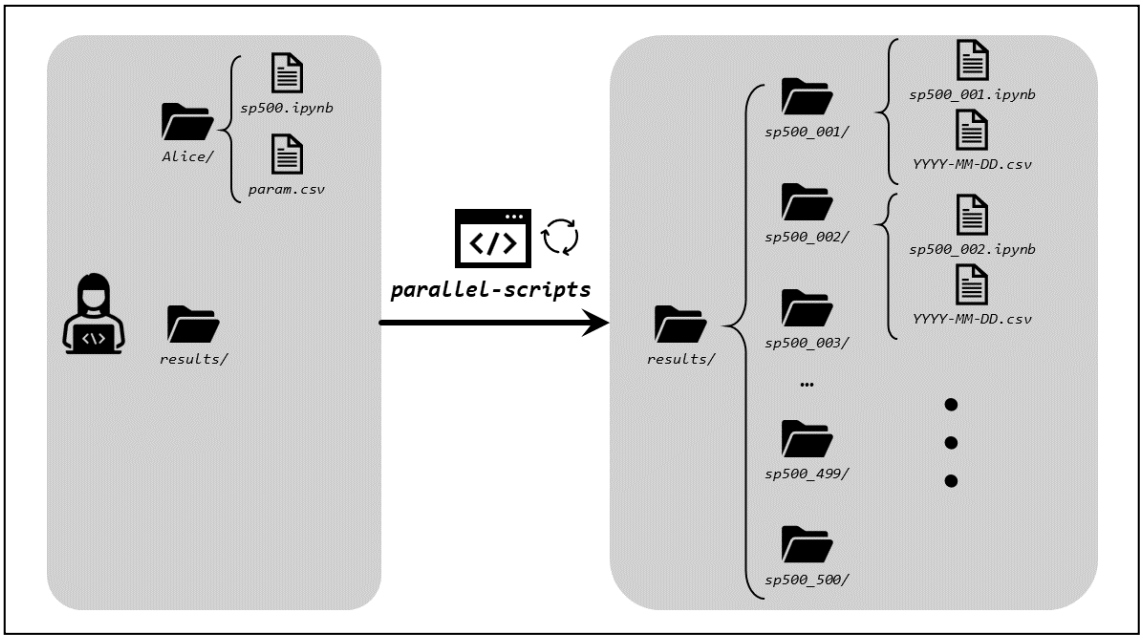

Figura 2-3: Situación inicial y final tras la ejecución de la librería.

La Figura 2-3 ilustra este proceso, mostrando la situación inicial a la izquierda, con un notebook plantilla (*sp500.ipynb*) y un archivo de parámetros (*param.csv*). A la derecha, se presenta la situación final, donde se han generado 500 carpetas, una para cada conjunto de parámetros. Cada carpeta contiene el notebook generado junto con sus resultados correspondientes.

En caso de que la ejecución se defina como “local”, todos los notebooks se ejecutarán utilizando el mismo pool de threads en el ordenador del usuario, garantizando una distribución eficiente de los recursos disponibles.

La última etapa es la generación de los archivos concatenados. Esta fase depende de si el usuario desea o no generar estos archivos. Si el usuario opta por no generarlos, la ejecución finaliza inmediatamente. De lo contrario, los archivos descargados se combinan en un único *Dataframe* de *Pandas*, que luego se guarda como un archivo CSV en el directorio de salida especificado.

Para conectarse a los servicios de AWS, es imprescindible contar con credenciales específicas, como un identificador y una clave secreta. Aunque actualmente estas credenciales se almacenan en archivos de texto en el directorio HOME del usuario, facilitando las pruebas y el desarrollo de la librería, se sugiere el uso de variables de entorno como una opción más segura para proteger esta información sensible. Esta práctica reduce los riesgos asociados al almacenamiento de credenciales en archivos de texto plano.

2.6. Otras soluciones actuales

Aunque la problemática descrita no es novedosa y es algo con lo que las empresas del sector ya están familiarizadas, las restricciones y requisitos específicos impuestos por la empresa financiera, añaden un nivel de complejidad adicional que no tienen solución con algunos de los programas existentes en el mercado.

Una de las alternativas consideradas fue *Papermill*, una herramienta que facilita la parametrización y ejecución de notebooks de manera similar a nuestro enfoque. Sin embargo, *Papermill* se limita a ejecuciones locales y no ofrece la capacidad de modificar celdas con diferentes tags, una funcionalidad crucial para que nuestra librería pueda guardar variables específicas de la ejecución. Por este motivo, se optó por desarrollar la parametrización desde cero, sin depender de esta librería.

Otra herramienta evaluada fue *Ploomber*, que permite la automatización de flujos de trabajo para *Jupyter Notebooks*. Aunque *Ploomber* puede alcanzar resultados similares a los de *Papermill*, también se limita a ejecuciones locales y añade dependencias adicionales que incrementan la complejidad del sistema.

Por lo tanto, la solución que se considera más adecuada para cumplir con los requisitos de la organización es una librería que ha sido desarrollada a medida y probada durante este Trabajo de Fin de Máster, denominada *Parallel-Scripts*.

2.7. *Git* público

Antes de profundizar con la descripción de la herramienta, es necesario señalar que todo el código que se describe en este documento se encuentra alojado en un repositorio público (tanto la herramienta desarrollada como las pruebas y experimentos que se han llevado a cabo) por si fuera de utilidad en el desarrollo de investigaciones en el área y para su consulta por parte del tribunal. Las menciones que se hagan al código harán referencia a dicho repositorio, para que la longitud de este documento no sea extensa y

para que pueda ser verificado por el lector y toda persona que busque un instrumento de estas características.

El enlace para acceder a él es el siguiente:

https://github.com/jlpadillas/Parallel-Script

3 Herramienta *Parallel-Script*

Para cumplir con los requisitos de la empresa financiera, y asegurar que la librería sea fácil de usar, incluso para aquellos sin conocimientos informáticos, se ha priorizado la simplicidad en su diseño. Teniendo en cuenta que los usuarios utilizan *Jupyter Notebooks*, se ha decidido desarrollar la herramienta en *Python* y generar un *Wheel*, lo que permite que la instalación sea simple y accesible, basta con ejecutar un comando de *pip install*. Para garantizar que el usuario no necesite entender la funcionalidad interna de la herramienta, se propone exponer al usuario una única función que deban conocer y será la responsable de llevar a cabo todas las operaciones.

En relación con los resultados, se han explorado diversas metodologías para almacenar *arrays* de *NumPy*, *DataFrames* de *pandas* y otros tipos de datos utilizando los formatos *CSV*, *pickle* y *parquet*. Entre estos, los formatos binarios *pickle* y *parquet* destacan por su eficiencia. Particularmente, *pickle* se ha identificado como la opción óptima debido a su rapidez en los procesos de lectura y escritura (Figura 3-1), así como su eficiencia en la utilización del espacio en disco. Por consiguiente, se ha decidido emplear *pickle* como el formato estándar para almacenar variables, independientemente de si los datos requieren posteriormente algún tipo de agrupación adicional.

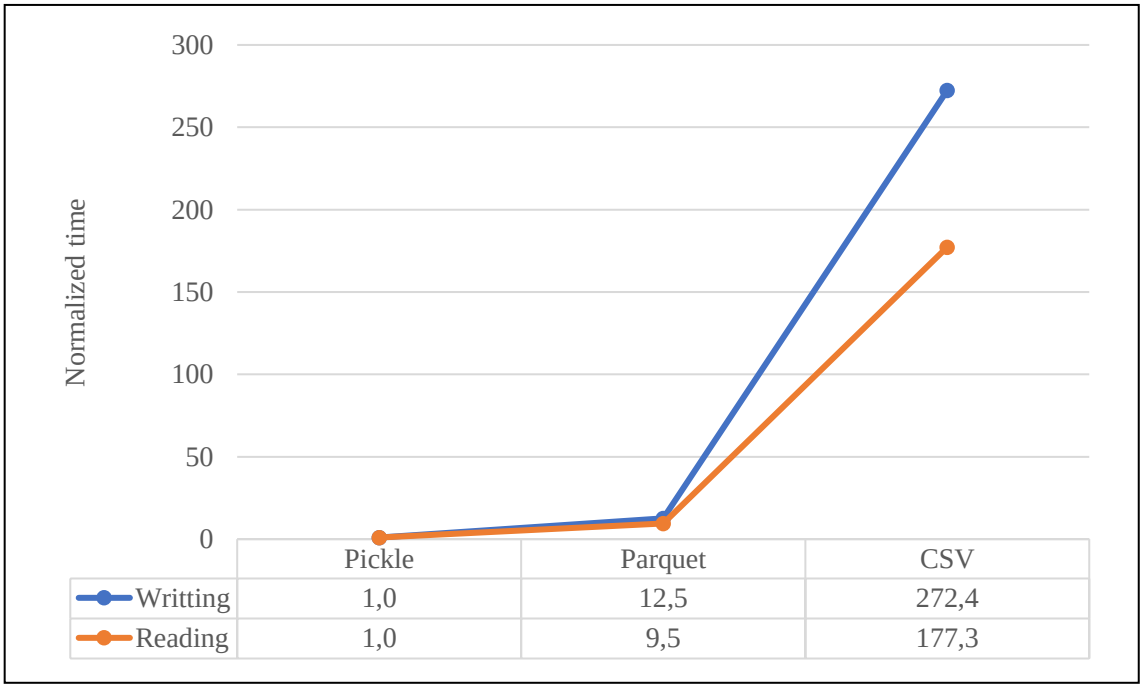

Figura 3-1: Comparación de los tiempos normalizados de lectura y escritura para 5 millones de elementos utilizando los formatos Pickle, Parquet y CSV.

En cuanto a la definición de parámetros y resultados, se opta por utilizar los metadatos de las celdas del notebook, y en concreto los tags que se pueden asignar a estas. De esta manera, los parámetros que se planean modificar deben estar en una celda del *Jupyter Notebook* y deben ser etiquetados con el tag "*parameters*", como aparece en la Figura 3-2. El programa reemplazará esta celda con los nuevos parámetros leídos del archivo de parámetros. En cuanto a los resultados, es necesario definir la variable a almacenar, o

de interés, en una única celda y etiquetarla con "*results*". La biblioteca modificará el código de la celda para guardar la variable como *pickle*, y en una ejecución posterior, agregará todas ellas o no, dependiendo de si la celda de resultado tiene, además, un tag "*concatenate*" o no.

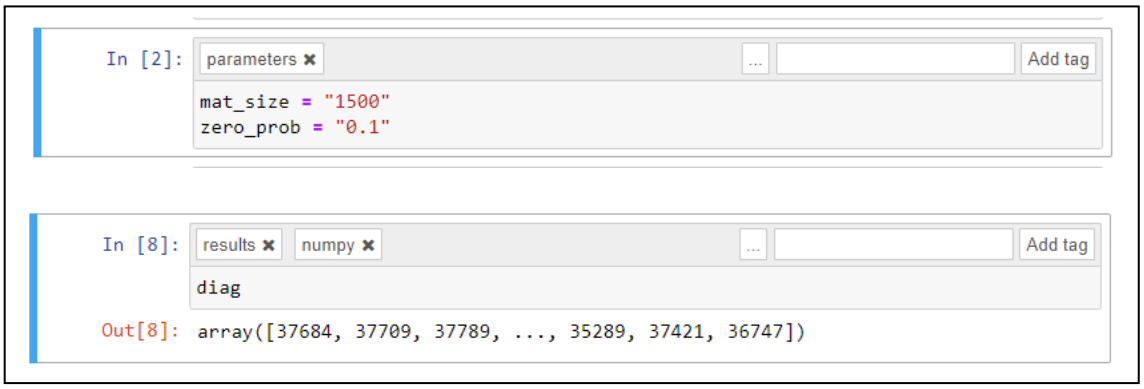

Figura 3-2: Etiquetas necesarias para definir las celdas de parámetros y de resultados.

Por último, el destino de ejecución se tiene que proporcionar como una variable por parte del usuario en la única función que conoce. Esta opción le permite elegir entre una ejecución "local" para ejecutar los notebooks en el ordenador del usuario, "AWS" si se desea enviar los notebooks a Amazon para que se ejecuten en la nube, o "UC" que es el nombre que se le ha asignado al servidor de la *Universidad de Cantabria* y que se usará como un servidor *on-demand* para hacer las pruebas oportunas.

Con estos enfoques, se dan solución a los requisitos que plantea la empresa. La herramienta desarrollada ha sido escrita en *Python* y realiza la función de *front-end* que permite al usuario poder comunicarse con AWS y enviar y recibir los archivos que necesitan ser ejecutados. Todo ello, a espaldas del usuario que sólo necesita conocer el añadir los tags a los notebooks y cómo generar el archivo de parámetros que desea usar.

En la Figura 3-3 se ilustra la simplicidad de uso de la librería, mostrando cómo configurar los parámetros básicos. A continuación, se detallan dichos elementos de la configuración:

1. Carpeta de entrada: Ubicación donde se buscarán los archivos de entrada y dependencias necesarias, en caso de haberlas.
2. Nombre del *Jupyter notebook* de plantilla: Identifica el archivo base utilizado para las ejecuciones.
3. Nombre del archivo de parámetros: Especifica los parámetros que regirán la generación y ejecución del notebook.
4. Carpeta de salida: Directorio donde se almacenarán los notebooks ejecutados y los resultados generados.
5. Lugar de ejecución: Permite seleccionar entre opciones como AWS, local y UC.
6. Dependencias: Lista de nombres de archivos adicionales necesarios para la ejecución en la nube.

Los parámetros se presentan en el orden en que aparecen en la Figura 3-3, desde la línea 8 hasta la línea 13, facilitando su comprensión y aplicación práctica.

```python
import time

import ParallelScripts as ps

start_time = time.time()
# ------------------------------------------------------------------------------ #
ps.run_notebooks(
    input_folder="/home/padilla/TFM/Dummy_matmul/input",
    template_notebook="matmul.ipynb",
    parameters_file="parameters_AWS.csv",
    output_folder="/home/padilla/PS_Execution/AWS",
    run_location="AWS",
    dependencies=None,
)
# ------------------------------------------------------------------------------ #
print(f"Total time: {time.time() - start_time} seconds.")

```

Figura 3-3: Ejemplo de cómo realizar una ejecución con Parallel-Scripts.

La simplicidad en el uso de la librería oculta una serie de procesos clave que garantizan la eficiencia del pipeline. A continuación, se detalla los ajustes necesarios tanto para entornos AWS como para servidores *on-demand*; y, se explica cómo las tecnologías, discutidas en la Sección 2, se integran para facilitar la ejecución de notebooks en la nube.

3.1. Configuración en la nube (AWS)

En esta sección se presenta de manera detallada la configuración y despliegue de los servicios en la nube utilizando AWS. A continuación, se destacarán las opciones más relevantes y se explicará la razón detrás de la elección de cada configuración. Esto facilitará una mejor comprensión de las decisiones técnicas adoptadas y de cómo estas influyen en el rendimiento y la seguridad del sistema.

Como se mencionó en la Sección 2.2.1, el programa requiere un espacio para almacenar tanto los archivos que se envían para ejecutar, como los resultados generados. Para ello, se ha decidido crear un *bucket* de S3, denominado *tfm-parallel-scripts*. Es importante recordar que tanto la subida como la descarga de archivos desde y hacia S3 implican costos asociados que serán facturados por AWS al final de cada mes. Todos estos costos deben ser considerados en la planificación del proyecto para evitar sorpresas en la facturación.

En la Figura 3-4 se muestra cómo crear este *bucket* usando la interfaz gráfica, con las configuraciones por defecto. Si bien, es también posible hacerlo mediante línea de comandos usando el AWS CLI, tal y como se muestra en la Figura 3-5.

AWS CLI permite automatizar la creación y gestión de recursos en AWS con comandos directos, lo que no solo ahorra tiempo, sino que también facilita el volver a generar la

infraestructura en diferentes entornos. Por esta razón, en este proyecto se usarán scripts para configurar la infraestructura de la nube, disponibles en el repositorio del proyecto.

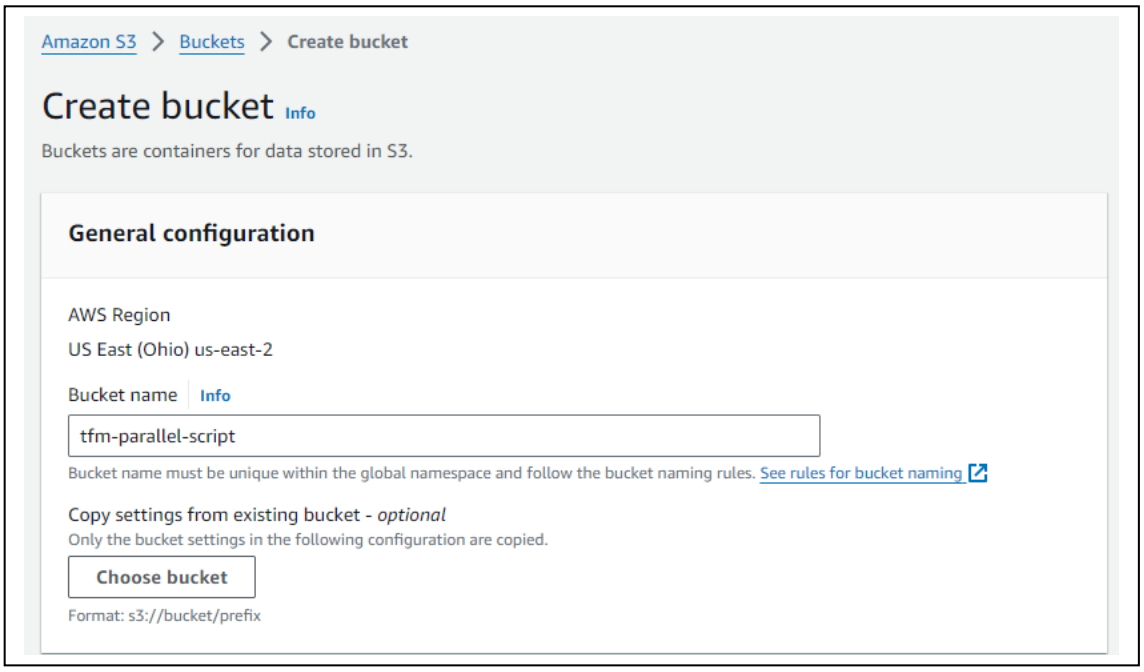

Figura 3-4: Creación de un bucket en S3 mediante interfaz gráfica.

```bash
#/bin/bash

# ------------------------------------------------------------------------------ #
# @Author: Juan Padilla                                                          #
# @Date:   2023-05-19 21:51:25                                                   #
# @Description:                                                                  #
#   This script creates the AWS S3 Bucket needed.                                #
# ------------------------------------------------------------------------------ #

BUCKET_NAME="tfm-parallel-scripts"

REGION="us-east-2"

aws s3 mb s3://$BUCKET_NAME --region $REGION
```

Figura 3-5: Creación de un bucket en S3 mediante script en bash que usa el AWS CLI.

3.1.1. Amazon DynamoDB

Además de la base de datos local mencionada en la Sección 2.5, se ha implementado una base de datos en AWS que permite que todos los servicios de AWS se comuniquen de manera efectiva entre sí. Nuestra librería interactúa con esta base de datos realizando consultas para obtener información actualizada sobre el estado de los notebooks en ejecución.

Es importante recordar que la configuración de esta base de datos debe realizarse con anticipación, optando por un modelo de capacidad bajo demanda. Este modelo resulta ser el más económico en escenarios donde el uso no es constante, permitiendo pagar únicamente por las operaciones de escritura y lectura que se realicen. Por ejemplo, podría darse el caso de que un usuario ejecute unas 500 peticiones en un momento específico y luego no utilice el servicio por el resto del día. En tales circunstancias, el modelo bajo demanda ofrece una solución de costo efectivo en comparación con la alternativa denominada capacidad provisionada, que interesa cuando su uso es continuo y predecible.

Se han habilitado dos funcionalidades adicionales para esta base de datos en AWS. La primera es el *Time To Live* (TTL) de los registros, que permite la eliminación automática de entradas después de un período determinado. Se ha configurado un TTL de 7 días desde el momento de la creación del registro, lo que se considera un plazo razonable para corregir y relanzar ejecuciones en caso de fallo. Además, este período es lo suficientemente corto como para evitar costos innecesarios por el almacenamiento prolongado de datos en *DynamoDB*, contribuyendo así a la reducción de gastos en AWS.

La segunda funcionalidad es la activación de *streams*, que es crucial para el correcto funcionamiento de las funciones Lambda. Cuando se actualiza un registro en la tabla, se activa automáticamente un proceso que permite a las funciones Lambda intervenir. Estas funciones verifican si el registro asociado a un notebook está listo para ejecutarse o si ha cambiado su estado de ejecución, permitiendo una gestión más dinámica y eficiente del flujo de trabajo.

La segunda funcionalidad es la activación de los *DynamoDB Streams*, que es crucial para el correcto funcionamiento de las funciones *Lambda*. Estos *Streams* permiten capturar una secuencia de cambios en los registros de una tabla *DynamoDB*, como inserciones, actualizaciones y eliminaciones. Cuando se actualiza un registro en la tabla, el *Stream* asociado captura el evento y lo envía a una de las funciones *Lambda*.

3.1.2. AWS Lambda

En total, se han implementado dos funciones Lambda, ambas escritas en Python, para crear las tareas en Batch y controlar cuando terminan su ejecución:

a. *tfm-ps-submit-job*: Esta función reacciona a los *Streams* de *DynamoDB*. Cuando un registro asociado a un notebook está listo para ejecutarse, la función Lambda obtiene los detalles necesarios y crea una tarea en AWS Batch. Esto permite una gestión dinámica y eficiente del flujo de trabajo, asegurando que cada notebook se ejecute en el momento adecuado.
b. *tfm-ps-check-job-status*: Esta segunda función tiene una función similar, pero se encarga de monitorear los cambios en los jobs de AWS Batch. Cuando un job finaliza, ya sea con éxito o error, esta función es activada para capturar el job terminado. A continuación, actualiza el estado y la hora de finalización en la base de datos de *DynamoDB*, manteniendo un registro preciso y actualizado de cada ejecución.

Cada función debe ser empaquetada en un archivo .zip y cargada al crear la función Lambda correspondiente. Además, es crucial asignar los permisos y roles adecuados para asegurar que las funciones tengan acceso a los recursos necesarios.

3.1.3. AWS Batch

AWS Batch permite ejecutar tareas por lotes utilizando *Fargate* o EC2 como entornos de ejecución. *Fargate* es un servicio sin servidor que gestiona automáticamente la infraestructura subyacente, eliminando la necesidad de configurar y escalar servidores. Sin embargo, en este caso, se ha optado por configurar AWS Batch para operar en un entorno controlado con instancias EC2, lo que permite prever y gestionar los costos de manera más eficaz. Para optimizar el rendimiento, se ha seleccionado un clúster compuesto por 9 instancias EC2 con las siguientes especificaciones:

- Procesador *Intel Xeon Scalable* de hasta 2,4 GHz (*Broadwell E5-2686 v4* o *Haswell E5-2676 v3*)
- Optimizadas para EBS sin costos adicionales
- Soporte para redes mejoradas
- Equilibrio de recursos informáticos, de memoria y de red

Tabla 3-1: Detalles sobre las instancias usadas, de acuerdo con AWS [13].

Instancia	*CPU virtual*	*Memoria (GiB)*	*Tarifa por hora bajo demanda*	*Banda ancha de EBS dedicado (Mbps)*	*Rendimiento de red*
m4.4xlarge	16	64	0,80 USD	2000	Alto

De acuerdo con la Tabla 3-1, este clúster proporciona un total de 144 *vCPUs* que están disponibles para procesar las tareas conforme llegan. Es importante destacar que Amazon cobra por cada minuto que las máquinas están encendidas; por lo tanto, para maximizar la eficiencia, se ha decidido que el sistema mantenga las máquinas apagadas por defecto, activándolas sólo cuando es necesario. Esta gestión se realiza de manera automática por *AWS Batch*, lo que simplifica el proceso.

Para lanzar ejecuciones en *AWS Batch*, es necesario configurar tres aspectos clave:

a. *Compute environment*: En esta sección se especifica el uso de instancias EC2 para la ejecución de tareas. A diferencia de entornos donde se utilizan diversos tipos de instancias, en nuestro caso, *Batch* utilizará un único tipo de instancia, garantizando así una utilización eficiente y constante, independientemente de la cantidad de trabajos en espera. El entorno está configurado para adaptarse automáticamente a las necesidades de procesamiento, con una capacidad variable que oscila entre un mínimo de 0 *vCPUs* y un máximo de 144 *vCPUs*. Esto significa que, si no hay tareas en la cola, no se mantendrán instancias EC2 activas, pero se activarán progresivamente a medida que se reciban nuevas tareas, optimizando así el uso de recursos y costos.
b. *Job queue*: Esta es la cola en la que se almacenan las tareas hasta que el *compute environment* esté disponible para procesarlas. La *job queue* se vincula directa-

mente al *compute environment* previamente configurado, asegurando una gestión eficiente del flujo de trabajo.

c. *Job definition*: Aquí se define la tarea específica que realizará cada contenedor durante la ejecución. Es necesario registrar la URL de la imagen del contenedor que se ejecutará, así como los requisitos de cada contenedor y los parámetros de entrada que se esperan. En este proyecto, cuando la primera función Lambda cree las tareas, pasará como parámetros el ID y nombre del notebook, junto con las rutas de S3 tanto para los datos de entrada como de salida.

3.2.Ejecución en la nube (AWS)

El proceso inicial, ya detallado en la sección 2.5, es común a todos los escenarios y sirve como base para las etapas subsiguientes. Una vez generados los *Jupyter Notebooks* ya parametrizados en local, la biblioteca procede a la siguiente fase, que implica la gestión de estos archivos en la nube.

En la Figura 3-6 se presenta la función *_execute_notebooks_AWS()*, encargada de gestionar las operaciones en la nube, y cuyos detalles se exploran a continuación. Esta función automatiza la carga de los notebooks al *bucket* de S3 (línea 18). Una vez que los notebooks están en S3, la interacción avanza hacia la base de datos de *DynamoDB*, donde se replican aspectos claves de la base de datos local (línea 21). Este paso es vital para un monitoreo efectivo del proceso e incluye datos esenciales como el estado actual del notebook, la hora de inicio y la hora de finalización de su ejecución, lo que es fundamental para la administración y seguimiento de cada notebook durante su ciclo de vida en la nube.

Una vez que todos los notebooks han sido registrados en *DynamoDB*, su estado cambia a "*ready*" (línea 24), lo que activa la primera función *Lambda*, esencial para iniciar el proceso en *AWS Batch*. Esta función *Lambda* está programada para crear un trabajo en *AWS* Batch por cada entrada en *DynamoDB* cuyo estado haya cambiado a "*ready*". Esto significa que se generará un trabajo individual en *AWS Batch* para cada notebook que esté listo para su ejecución.

Una vez se han creado todos los trabajos, *AWS Batch* toma el control, encargándose de iniciar las ejecuciones en el entorno de cómputo configurado previamente. El proceso dentro de cada contenedor se desarrolla de la siguiente manera:

1. Descarga de S3 el notebook que se le ha asignado, así como las posibles dependencias.
2. Ejecuta su notebook de manera independiente y guarda su estado tras su ejecución y los posibles resultados que haya generado.
3. Al terminar, sube los archivos generados, si los hay, y sobrescribe el notebook ya ejecutado.

```python
def _execute_notebooks_AWS(tracker: Tracker) -> None:
    """
    Execute the notebooks in AWS using AWS Batch. The notebooks are uploaded
    to S3 and the dependencies are uploaded to S3 and registered in the
    tracker.

    Parameters
    ----------
    tracker : Tracker
        Tracker with the notebooks to execute.
    """
    aws_manager = AWSManager()  # Create the s3Manager object.

    tracker.add_remote_paths(aws_manager.remote_path)      # Update the tracker with the S3 paths

    local_and_s3_notebooks_list = tracker.get_local_and_remote_nb_list()  # Get the local and S3 notebooks

    aws_manager.upload_files_to_s3(local_and_s3_notebooks_list)

    logging.info("Uploading the tracker to S3.")
    aws_manager.upload_tracker_to_dynamodb(tracker=tracker)

    logging.info("Setting the Is_Ready flag to True.")
    aws_manager.set_is_ready_flag_to_true(tracker)

    logging.info("Waiting for the processes to finish.")
    aws_manager.wait_for_processes_to_finish(tracker)

    if not tracker.are_notebooks_executed_correctly():  # Check if there are any errors in the execution
        raise RuntimeError("There were errors in the execution.")

    s3_and_local_files_list = tracker.get_remote_and_local_files()  # Get the list of files to download

    logging.info("Downloading the output notebooks from S3.")
    aws_manager.download_files_from_s3(s3_and_local_files_list)

    logging.info("Cleaning the S3 bucket.")
    aws_manager.remove_files_from_s3([s3_file for s3_file, _ in s3_and_local_files_list])

    logging.info("Closing the connection to AWS.")
    aws_manager.close_connection()
```

Figura 3-6: Función _execute_notebooks_AWS() que se encarga de ejecutar los notebooks en AWS.

Después de cada tarea ejecutada por *Batch*, la segunda función *Lambda* se activa para verificar el estado con el que ha concluido y actualizar el registro correspondiente en *DynamoDB*.

Durante todo este proceso, la librería se mantiene en un estado de petición constante (*polling*), realizando consultas cada cierto intervalo de tiempo para verificar si las ejecuciones han concluido (línea 27). Si alguna de ellas ha fallado, *Batch* hace un segundo reintento automáticamente y, en caso de volver a fallar, se notifica al usuario y el programa se detiene lanzando un error (línea 30). En cambio, si todas las ejecuciones finalizan satisfactoriamente, se procede a descargar los archivos (línea 35) y a eliminarlos de S3 inmediatamente después para minimizar el tiempo de almacenamiento y evitar costos adicionales por mantener los datos en los servidores de S3 (línea 38).

Cabe destacar que, en caso de que todas las tareas se ejecuten correctamente, excepto la última, los notebooks y archivos generados se conservarán en S3. Esto permite que puedan ser recuperados y procesados manualmente si es necesario, garantizando así que no se pierda el trabajo realizado incluso si ocurre un fallo al final del proceso.

Por otro lado, aunque el método de *polling* usado no es el más eficiente económicamente, se ha elegido por su sencillez y bajo la premisa de que los notebooks no tardarán más de 15 minutos. Realizar dos peticiones por minuto durante 15 minutos resulta en un total, de 30 peticiones por notebook, un coste que se puede considerar asumible. Por otro lado, se han contemplado alternativas más económicas para futuras mejoras, como el uso de un *ok-flag* para señalar la conclusión de las tareas o el envío de peticiones HTTP a un servidor específico; aunque, recordemos, que todo tráfico saliente de las instancias virtuales de Amazon, incurre en un gasto que es facturado.

Así, todas las tecnologías mencionadas colaboran conjuntamente en el proceso, tal como se ilustra en la Figura 3-7.

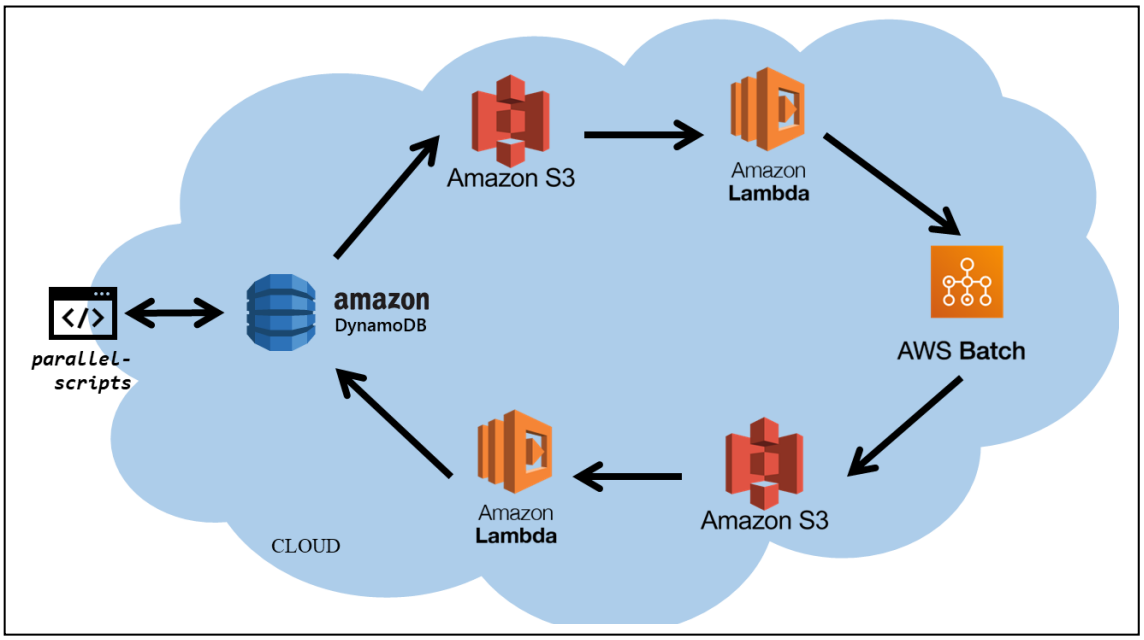

Figura 3-7: Flujo de trabajo donde se integran los distintos servicios de AWS y la librería desarrollada.

3.3. Configuración en un servidor on-demand (UC)

El grupo *Computer Engineering* de la *Universidad de Cantabria* cuenta con equipamiento hardware alojado en el Centro de Procesamiento de Datos (CPD) *3Mares*, situado en la facultad de Ciencias, que ha facilitado el uso de 16 nodos para llevar cabo algunas pruebas de este TFM. La arquitectura de este centro de datos se compone por un *front-end* que permite la conexión mediante SSH y actúa como puente hacia otros servicios y nodos disponibles.

Para acceder a estos recursos es necesario disponer de credenciales específicas, en este caso, un nombre de usuario y una contraseña. Por lo tanto, *Parallel-Scripts* realizará la función de intermediario entre el usuario y este *front-end*. Tal y como se refleja en la Figura 3-8, el *CE-FE* es el *front-end* y los *ce100*, *ce101*, …, *ce115* son los nodos disponibles para realizar pruebas.

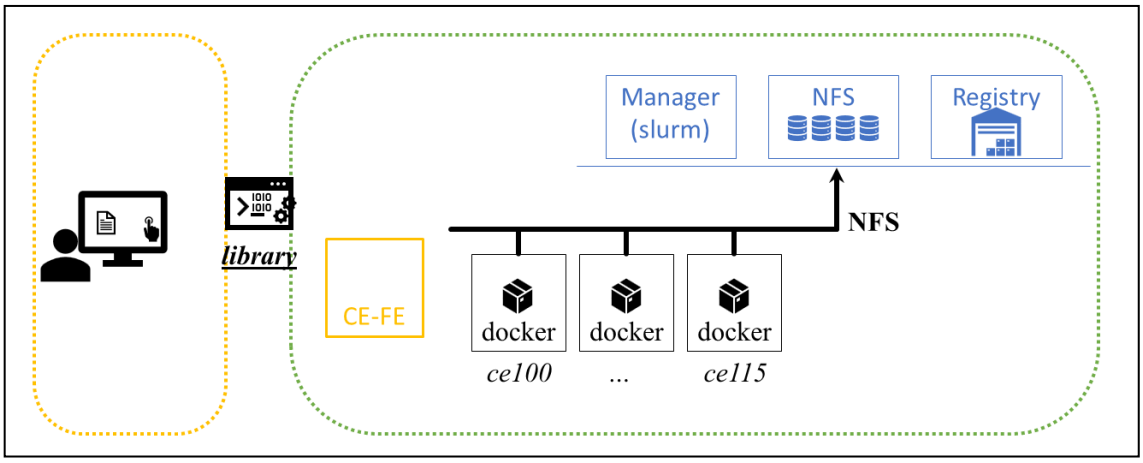

Figura 3-8: Comunicación entre la librería y los distintos servicios del clúster on-demand.

Siguiendo una metodología similar a la utilizada anteriormente con AWS, y con el objetivo de mantener una coherencia en el manejo de las credenciales, se ha optado por almacenar éstas en un archivo de configuración ubicado en el directorio HOME del usuario. Aunque este enfoque simplifica la administración durante el desarrollo, no es aconsejable utilizarlo en entornos de producción. Por ello, se recomienda adoptar prácticas más seguras, como el uso de variables de entorno para la gestión de credenciales sensibles.

El nodo *ce100* actuará como el nodo maestro, en el que se instalarán los distintos servicios descritos en la Sección 2.3. Para ello, se asumirá que todos los nodos cuentan con la misma versión de SO, en este caso, *Ubuntu Server 22.04*, y que son visibles entre sí en la red.

Como se ha comentado anteriormente, *Slurm* es un sistema de gestión de recursos y programación de tareas. Aunque los pasos para instalar y configurar este servicio no son triviales, se ha optado por seguir la documentación oficial de *Slurm* [12] que proporciona instrucciones detalladas para la configuración en distintos sistemas. Dado que los mismos pasos se tienen que seguir en todos los nodos, se ha creado un script que permite instalar y configurar este servicio. A parte de la configuración inicial, no será necesario realizar ajustes adicionales para que se pueda ejecutar el software desarrollado en este TFM. Esto asegura un sistema de cola que emula el comportamiento y configuración de *AWS Batch*.

El siguiente paso consiste en configurar NFS, para lo cual se destinará un disco SSD de 2 TB que será compartido por el resto de los nodos. Se ha configurado el servicio para que se inicie automáticamente después de cada reinicio, y los permisos se han dejado abiertos para facilitar la configuración. Si bien, en un entorno de producción esto habría que ajustarlo para que sea más restrictivo y sólo accedan aquellos nodos permitidos.

En cuanto al *Docker Registry*, se ha configurado el sistema para que funcione mediante comunicación HTTP en lugar de HTTPS. Esto es algo que requeriría generación y distribución de certificados, detalles que no se pretende alcanzar en este punto. Aunque el uso de HTTPS es recomendable para entornos de producción, en este proyecto no se contempla su aplicación. Para la instalación de *Docker Registry*, es necesario instalar

Docker previamente, algo que se hace en el *Bash script* que se encuentra en el repositorio de este proyecto y que sigue la guía oficial.

Finalmente, el sistema requiere la instalación del módulo de Python *Paramiko*, que se puede obtener fácilmente ejecutando el comando *pip install paramiko*.

3.4.Ejecución en un servidor en propiedad (UC)

En la Figura 3-9 se presenta la función *_execute_notebooks_UC()*, encargada de gestionar las operaciones en el clúster de la UC, y cuyos detalles se exploran a continuación.

```python
def _execute_notebooks_UC(tracker: Tracker) -> None:
    uc_manager = SSHManager()
    tracker.add_remote_paths(uc_manager.remote_path)
    local_and_uc_notebooks_list = tracker.get_local_and_remote_nb_list()
    uc_manager.upload_files_to_uc(local_and_uc_notebooks_list)

    remote_notebooks_to_run = [nb_tuple[1] for nb_tuple in local_and_uc_notebooks_list]
    files_created = tracker.generate_batch_script(remote_notebooks_to_run)

    tuple_local_remote_b_script = [
        (local_batch_script, str(Path(tracker.remote_output_folder, local_batch_script.rsplit("/")[-1])))
        for _, local_batch_script in files_created
    ]

    uc_manager.upload_files_to_uc(tuple_local_remote_b_script)

    for nb_row in tracker.get_output_nb_to_run():
        tracker.set_nb_status(nb_row[0], Status.RUNNING)

    for i, (job_name, remote_batch_script_path) in enumerate(zip(files_created, tuple_local_remote_b_script)):
        try:
            uc_manager.run_command(f"chmod +x {remote_batch_script_path[1]}")
            stdout = uc_manager.run_command(f"sbatch {remote_batch_script_path[1]}")
            pid = stdout.split(" ")[-1].strip()
            uc_manager.wait_for_jobs(job_name[0], pid)
        except Exception as e:
            logging.error(f"Error executing the batch script: {e}")
            raise e

        if not uc_manager.check_jobs_finished_correctly(job_name[0], pid, tracker.remote_output_folder, len(remote_notebooks_to_run)):
            raise Exception("Error executing the notebooks.")

    uc_manager.download_files_from_uc(tracker.get_remote_and_local_files())

    for nb_row in tracker.get_output_nb_to_run():
        tracker.set_nb_status(nb_row[0], Status.COMPLETED)

    uc_manager.close_connection()
```

Figura 3-9: Función _execute_notebooks_UC() que se encarga de ejecutar los notebooks en la UC.

Una vez más, se parte desde la creación de los *Jupyter notebook* en el entorno local. Posteriormente, se procede a subir estos archivos al servidor de la UC (línea 5) usando la librería *Paramiko*, la cual facilita las conexiones SSH y la transferencia de archivos. Para acelerar este proceso, se ha decidido usar múltiples threads; específicamente, reutilizando el *pool* de threads que se usó para crear los archivos en local, para subir los mismos al disco configurado en NFS.

Una vez que los archivos, y las dependencias si las hubiera, están en el disco NFS del sistema – accesible por todos los nodos del clúster –, se procede a crear las tareas de *Slurm* (línea 8). Esto se realiza mediante un *Bash script* diseñado para ser interpretado por *Slurm*, especificando detalles relevantes para la ejecución, como el nombre de la tarea, el lugar de ejecución, el número de nodos a utilizar, los logs, entre otros. De tal modo que cada notebook se ejecuta en un contenedor Docker separado, y *Slurm* gestiona la distribución y ejecución de estos contenedores en el clúster.

El siguiente paso es darles a estos scripts permiso de ejecución (línea 22) y ejecutarlos (línea 23); después, queda esperar que todas las tareas finalicen, para lo cual, se hace *polling* cada cierto tiempo (línea 25), al igual que con AWS. *Slurm* ofrece varios comandos para la gestión de tareas y colas de trabajo; se utiliza el comando *squeue* para monitorizar el estado de las tareas lanzadas. Este comando, junto con otros parámetros, permite filtrar las tareas activas y esperar a que todas terminen.

Al igual que en AWS, es necesario tener en cuenta el tiempo de ejecución de las tareas. Siguiendo la misma premisa que con AWS, los notebooks no deberían tardar más de 15 minutos. Por ello, en el caso de este servidor *on-demand*, se ha limitado el tiempo de ejecución a dicho valor. Es decir, si las tareas terminan antes de ese límite, el proceso puede continuar; de lo contrario, si una o varias tareas exceden este tiempo, la librería *Parallel-Scripts* aborta las tareas que no cumplen con la suposición inicial. Este comportamiento puede ajustarse según sea necesario, incrementando o reduciendo el tiempo máximo permitido durante la modificación de una constante en la librería.

El manejo de archivos en este caso es distinto del utilizado en AWS. Gracias al sistema NFS, los archivos son accesibles localmente para todos los nodos, eliminando la necesidad de descargar y subir archivos como se hacía con S3 y *boto3*, incurriendo en gastos facturados luego por AWS. En este caso, simplemente se realiza una copia de los notebooks al contenedor, se ejecutan, y luego otra copia para sobrescribirlos y mover los resultados si los hubiera.

La función del *Registry* es análoga a la de ECR en AWS, ya que *Slurm* consulta este servicio para obtener la imagen del contenedor que debe ejecutar. Por tanto, previo a todas estas ejecuciones es necesario crear una imagen que cumpla con la funcionalidad descrita.

Finalmente, una vez que las tareas han terminado, la librería se encarga de descargar, usando una vez más *Paramiko* con múltiples threads, los notebooks ya ejecutados y los resultados, si los hay (línea 33). Tal como se mencionó anteriormente, la librería puede realizar la concatenación de resultados o devolverlos directamente al usuario, según lo solicitado.

3.5. Pruebas y evaluación

En el desarrollo de la librería, se ha puesto mucho énfasis en las pruebas para garantizar que el software sea estable y fiable. Hasta la fecha, se han desarrollado unas pruebas unitarias básicas que cubren las funcionalidades más críticas de la librería. Estas pruebas están diseñadas para validar que las operaciones importantes se ejecuten como se espera y para confirmar que cualquier modificación en el código no introduzca errores inadvertidos.

3.5.1. Pruebas unitarias

Las pruebas implementadas hasta ahora están contenidas en varios archivos específicos para cada módulo de la librería. Por ejemplo, se han creado pruebas para los módulos *aws_connect* y *nb_tracker*, entre otros, que son esenciales para la operatividad del

sistema en entornos de nube. Estas pruebas verifican la correcta interacción con los servicios de AWS y la gestión adecuada de las bases de datos local, respectivamente.

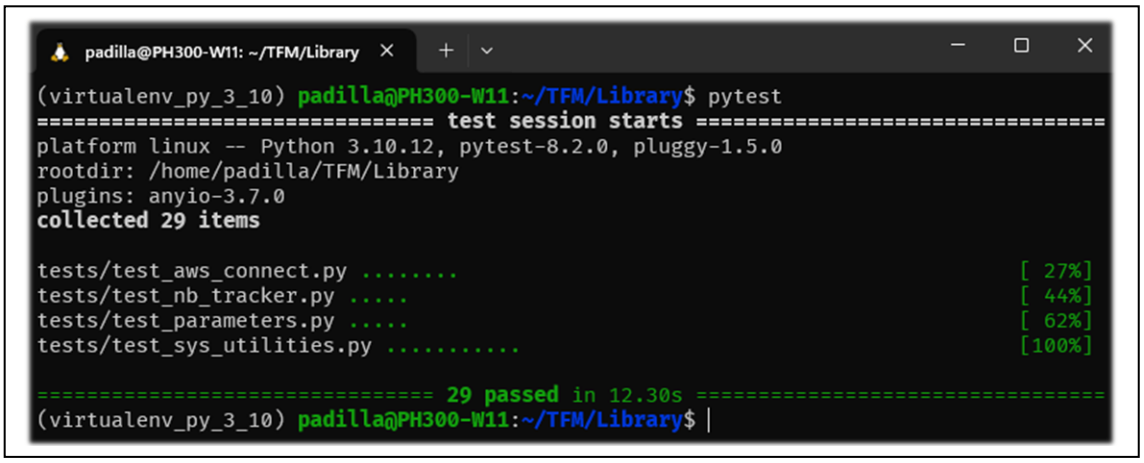

Figura 3-10: Pruebas unitarias de Parallel-Scripts desarrolladas hasta la fecha.

Es importante destacar que el número actual de pruebas no es definitivo, dado que es necesario una cobertura de pruebas más extensiva para abarcar todas las funcionalidades de la librería. Por lo tanto, está previsto continuar desarrollando y ampliando la batería de pruebas en las siguientes fases del proyecto. Con lo cual se podrá garantizar que *ParallelScripts* se mantenga actualizado frente a los cambios en las tecnologías subyacentes y las exigencias del mercado. Para acceder al código de las pruebas unitarias: *https://github.com/jlpadillas/Parallel-Script/tree/main/Library/tests*

3.5.2. Resultados de pruebas

En esta ejecución de prueba, se van a generar 16 notebooks, con una carga de trabajo ligera; para este caso de ejemplo, se van a usar multiplicaciones de matrices cuadradas donde los parámetros de entrada serán: la dimensión de las matrices y el porcentaje de ceros que tendrán las matrices. En total, los notebooks tardarán unos 25 segundos cada uno y se ejecutarán todos ellos en un único EC2, en el caso de AWS.

Además, es importante señalar el número de celdas que son de resultados, ya que se van a generar binarios por cada uno de ellos; lo que va a suponer operaciones extra para cargar y descargar dichos archivos desde y hacia el almacenamiento en S3. En este caso se guardará una variable, que es la diagonal resultante de la multiplicación de matrices y que puede resultar útil en ciertas situaciones tales como en métodos iterativos o en la resolución de sistemas lineales diagonales.

La Figura 3-11 muestra dos de los tres archivos usados en esta prueba:

a. A la izquierda, se muestra el notebook comentado antes, el cual genera dos matrices aleatorias y las multiplica para obtener la diagonal. Como se puede observar, la tercera celda está marcada con el tag "*parameters*", lo que indica que será reemplazada por los valores leídos del archivo de parámetros. Además, la última celda contiene el tag "*results*", lo que significa que la variable $diag$ se guardará como un archivo binario.

b. A la derecha, se muestra un archivo CSV que contiene los parámetros a reemplazar en cada notebook. Por ejemplo, el primer notebook tendrá los valores (mat_size = "1500", $zero\_prob$ = "0.00"), mientras que los notebooks siguientes utilizarán las líneas sucesivas del archivo.

El archivo principal que desencadena toda la ejecución es el mostrado en la Figura 3-3.

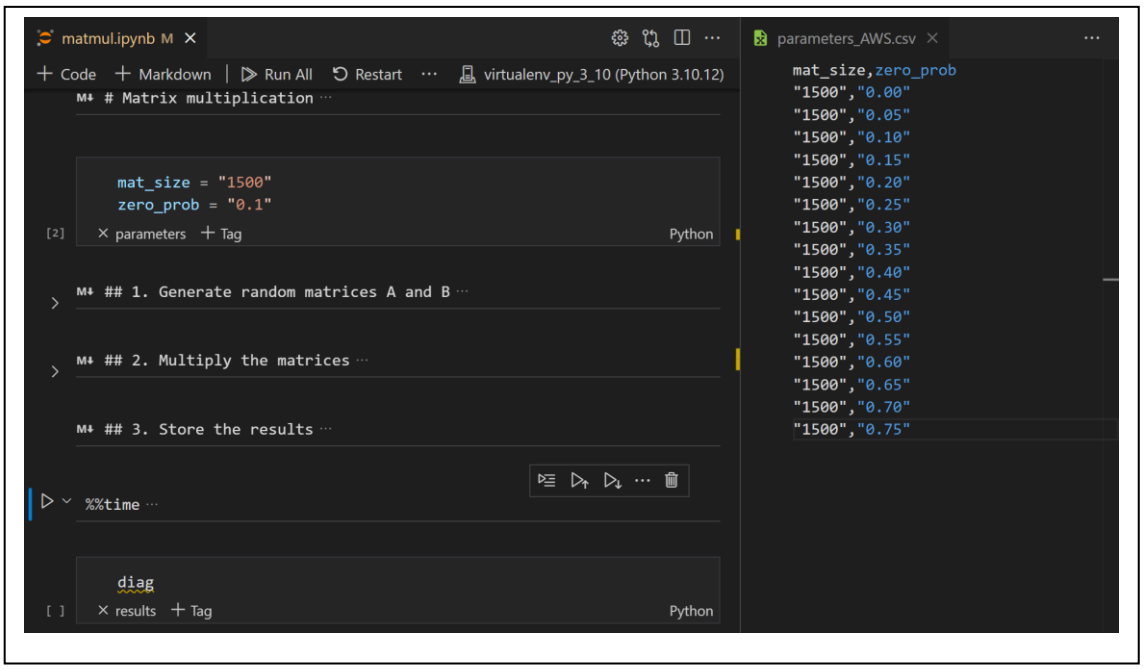

Figura 3-11: Archivos utilizados para probar la librería ejecutando en AWS.

La Figura 3-12 muestra la ejecución y los resultados del *logging*, configurado con el nivel de información "*info*". En esta figura se pueden observar las distintas etapas clave del proceso, incluyendo la generación de los notebooks, su carga en S3, la actualización del estado de las tareas en la base de datos y la descarga de los resultados una vez finalizadas las ejecuciones. Cada mensaje de log incluye una marca temporal y un nivel de detalle que facilitan el monitoreo y la depuración del flujo de trabajo de la librería en tiempo real. Por último, se muestra el contenido de la carpeta de salida para verificar que los notebooks y archivos se han generados correctamente.

En caso de que alguna tarea falle, el responsable de la librería puede acceder a la interfaz de AWS Batch y revisar la cola de tareas, donde se listan todas las tareas generadas por la librería. La Figura 3-13 muestra esta cola de trabajos al finalizar la ejecución de prueba. En esta interfaz, se pueden consultar detalles importantes como las horas de creación, inicio y finalización, el tiempo de ejecución, y el estado de cada tarea. Esta vista es especialmente útil, ya que permite relanzar manualmente una tarea o acceder a los logs para investigar la causa del fallo.

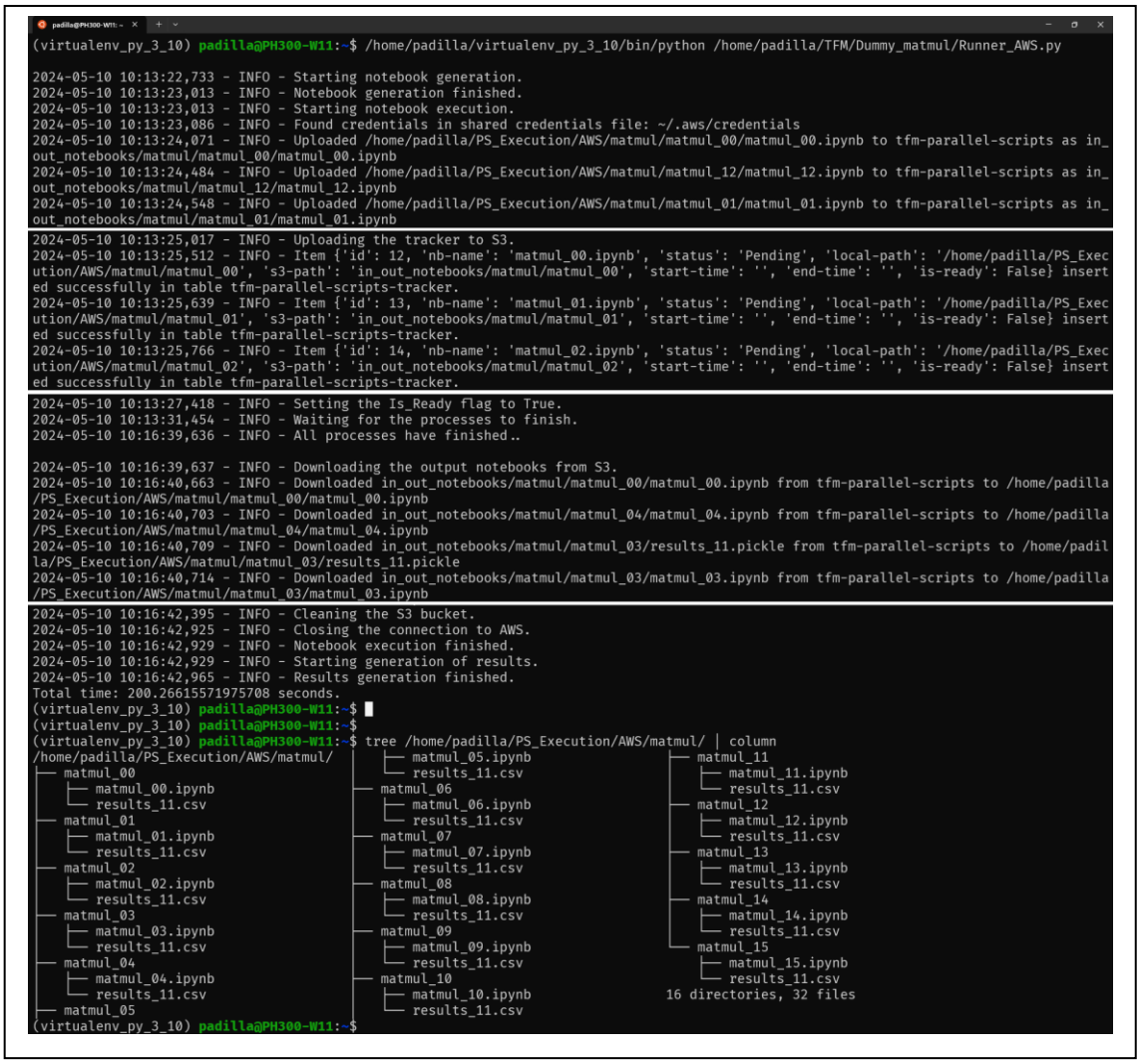

Figura 3-12: Traza generada durante la ejecución de la librería y comprobación de archivos generados.

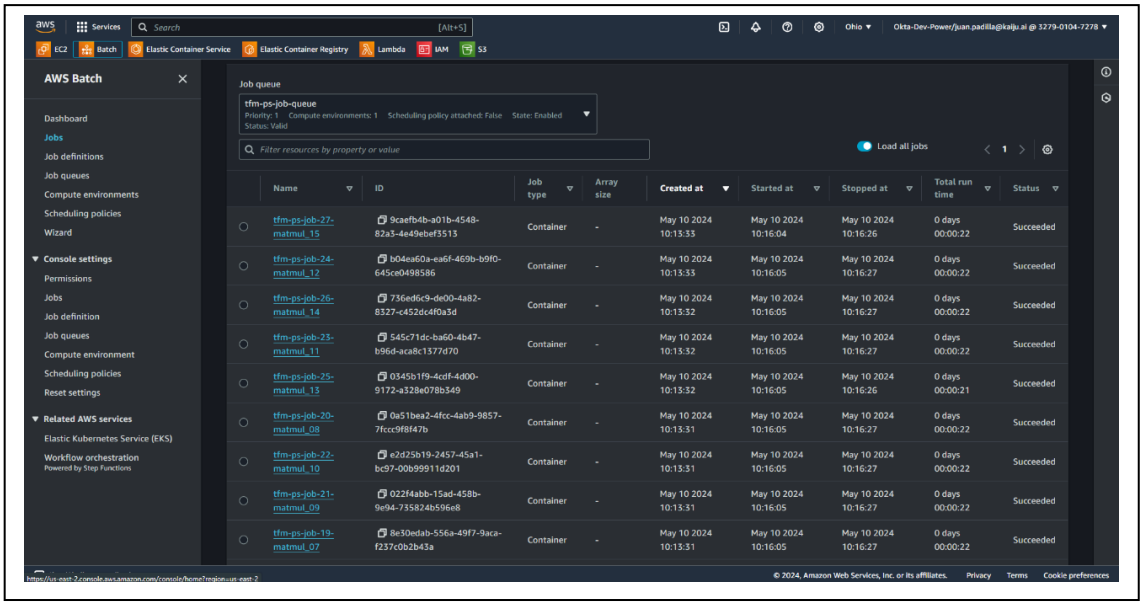

Figura 3-13:Cola de trabajos de AWS Batch con las ejecuciones finalizadas.

4 Análisis de costes

Una de las consideraciones más críticas de este proyecto es la gestión y previsión de costos. El traslado de toda la infraestructura de cálculo de un entorno *cloud* a otro local solamente tiene sentido si los beneficios económicos son más que evidentes. En este capítulo, nos hemos limitado a realizar un análisis preliminar de costos, cuyo objetivo es obtener cifras aproximadas (con varias simplificaciones) para entender de manera rápida el orden de magnitud en el que nos movemos. Este análisis preliminar pretende ofrecer una visión general de las implicaciones financieras tanto a corto como a largo plazo, comparando los costos asociados con la ejecución en AWS y en una infraestructura propia ubicada en un centro de *colocation*.

4.1. Coste de ejecución en AWS

El uso de servicios en la nube como AWS ofrece flexibilidad y escalabilidad, pero también implica costos que deben ser cuidadosamente calculados y monitoreados. En esta sección, se detalla el proceso de estimación de costos para la ejecución de notebooks utilizando la librería *Parallel-Scripts* en AWS.

4.1.1. Metodología para la estimación de costes

Aunque Amazon ofrece una herramienta gratuita llamada *AWS Pricing Calculator*, que permite a los usuarios estimar el costo del uso de determinados servicios, esta herramienta no resulta completamente adecuada para nuestros propósitos en lo que respecta a estimar el costo anticipado de ejecutar la librería *Parallel-Scripts*. El *AWS Pricing Calculator* facilita la estimación de costos para servicios como EC2, *DynamoDB* y otros, basándose en configuraciones específicas y en un período determinado, proporcionando una idea del gasto mensual asociado a esos servicios.

Es por ello por lo que se ha decidido desarrollar una calculadora de gastos utilizando Excel y valores constantes obtenidos mediante mediciones experimentales. Esta solución es viable dado que se tiene un conocimiento detallado de las interacciones de la librería con los distintos servicios de AWS, lo que permite obtener una estimación bastante precisa del costo real de una ejecución.

4.1.2. Ejemplo y verificación de estimación de costes

Los precios utilizados provienen directamente del sitio web oficial de AWS [14][15][16] y se han analizado y calculado minuciosamente para asegurar su precisión. Por ejemplo, se puede considerar una ejecución con las características de la Tabla 4-1:

Tabla 4-1: Parámetros usados para probar los costos de AWS.

Notebooks:	16
Exec. Time /s:	25
Result cells per-Notebook:	1
Num. EC2s:	1

Overhead start EC2 /s:	160
Overhead stop EC2 /s:	60
Lambda exec. Time /ms:	2500
Polling time /s:	15

La ejecución es la descrita en la Sección 3.5.2, y, dado que las instancias EC2 van a estar apagadas por defecto, se ha considerado un tiempo adicional correspondiente al periodo que las instancias tardan en encenderse y apagarse. Y estos valores hay que tenerlos en cuenta dado que Amazon cobra por el tiempo que una instancia está en funcionamiento, incluyendo el tiempo que toma iniciar y detener la instancia.

A continuación, se detallan los costos asociados a cada función que realiza la librería, y a cada servicio utilizado, proporcionando una visión clara de los gastos incurridos.

Tabla 4-2: Interacción con los servicios de AWS y el costo de cada uno de ellos, usando información de la Tabla 4 1.

	Services	**Function**	**Price /USD**
1.	S3	*Upload notebooks*	8,00E-05
2.	DynamoDB	*Create rows*	2,00E-05
3.	DynamoDB	*Set 'is_ready' to true*	2,00E-05
4.	Lambda	*Create Batch jobs*	5,25E-06
4.1.	DynamoDB	*Read modified rows*	4,00E-06
4.2.	DynamoDB	*Register start time*	2,00E-05
5.	EC2	*Warm-up and run notebooks*	5,49E-02
5.1.	S3	*Download and upload notebooks*	8,64E-05
5.2.	S3	*Upload results*	8,00E-05
5.3.	DynamoDB	*Polling*	4,99E-05
6.	Lambda	*Register end time*	2,00E-05
7.	S3	*Download notebooks and results*	1,28E-05
8.	S3	*Storage costs*	3,49E-11
9.	S3	*Clean the s3 bucket*	1,28E-05
		TOTAL:	**0,055 USD**

En la Tabla 4-2 se puede ver que la estimación total de gastos es de aproximadamente $0.055 USD. Para verificar la precisión de estos cálculos, se ha ejecutado la prueba en AWS y, utilizando el servicio *Cost Explorer*, se han revisado los costos incurridos. Es importante no confundir el *Cost Explorer* con el *Pricing Calculator* de AWS. Mientras que el *Pricing Calculator* se utiliza para estimar costos futuros, el *Cost Explorer* permite analizar y visualizar los gastos reales en los servicios de AWS, permitiendo filtrar por proyecto, servicio y periodo, como días o meses específicos.

Todos los servicios utilizados en este proyecto se han etiquetado con un tag específico: $Project: Padilla-thesis$, lo que facilita el seguimiento de los costos asociados al proyecto. Como se observa en la Figura 4-1, el cálculo realizado inicialmente no está muy desviado de los costos reales reportados por AWS, teniendo en cuenta que AWS muestra los precios con precisión hasta los centavos de dólar.

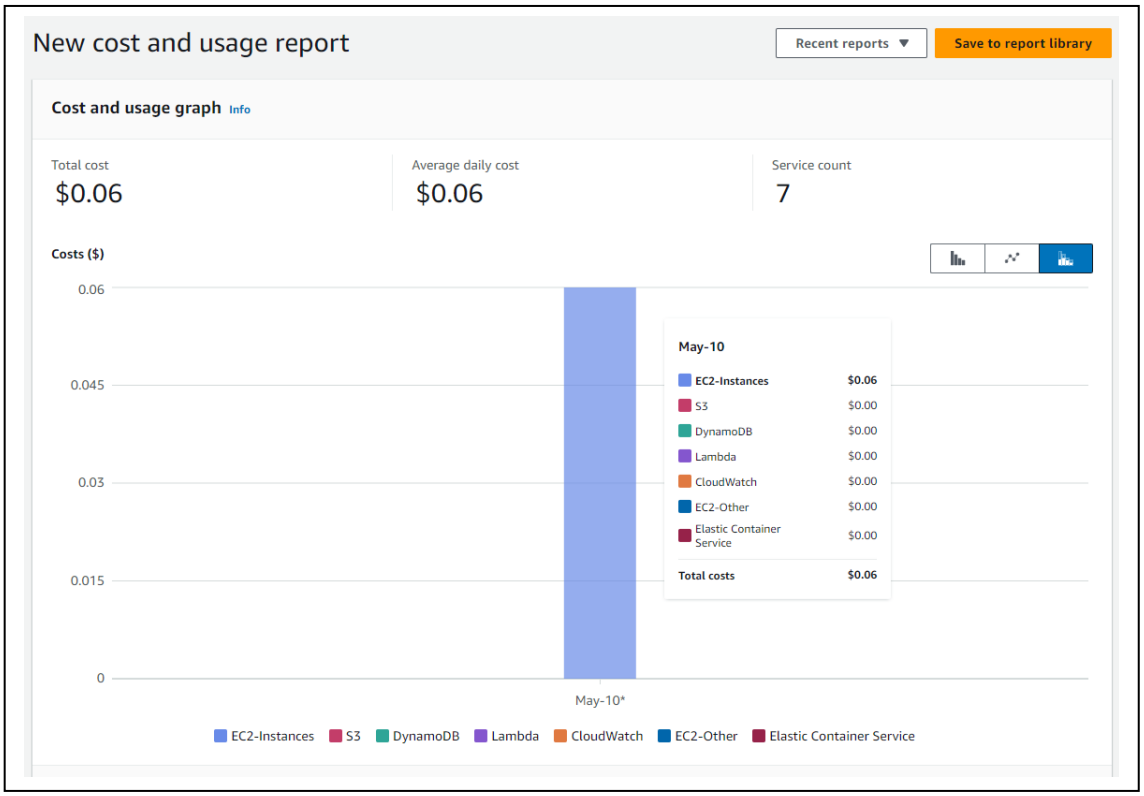

Figura 4-1: Costos asociados en AWS a una ejecución con los parámetros de la Tabla 4-1.

Otro dato relevante son los servicios que aparecen listados por AWS. Por ejemplo, se puede ver que entra en juego *Elastic Container Service* (ECS), el cual es la manera elegida para lanzar los contenedores a través de *AWS Batch*. Si bien, este servicio no es usado directamente, se ha podido detectar que AWS lista servicios en la factura como una forma de mostrar todos los servicios a los que se ha tenido acceso, incluso si no incurrieron en costos directos.

Prueba de ello, es también el "servicio" llamado *EC2-Other*, que puede incluir costos misceláneos asociados con EC2 que no son directamente las instancias. Esto puede incluir:

- *Elastic IP Addresses*. Si se tienen direcciones IP elásticas (direcciones IPv4 estáticas) que estén siendo usadas o no. Esto puede incurrir en gastos adicionales.
- *EBS* (*Elastic Block Store*) *Snapshots*: Son instantáneas de los discos duros usados que pueden servir para respaldo o recuperación de datos; y, en caso de usarlo, también genera gastos.
- Datos de transferencia. AWS cobra por la transferencia de datos de salida desde EC2 a internet o a otras regiones.

Ninguno de ellos es usado por *Parallel-Scripts*, por tanto, no debería haber ningún costo asociado a ello, pero como se puede ver, sí que aparece reflejado como mera información.

Con la Tabla 4-3 se pretende mostrar que más del 99% de los gastos se deben a las EC2. Es decir, si bien entran en juego otros servicios de AWS, la ejecución de las instancias

virtuales son las que incurren en la mayor parte del gasto. Esto parece una constante para 1 o 512 notebooks a ejecutarse en la nube.

Tabla 4-3: Desglose de costos por servicio de AWS durante la ejecución de diferentes cantidades de notebooks.

Notebooks	S3 /$	DynamoDB /$	Lambda /$	EC2 /$	TOTAL /$
1	1,70E-05	1,67E-05	6,50E-06	0,18	**0,182**
2	3,40E-05	3,33E-05	7,75E-06	0,18	**0,182**
4	6,80E-05	6,67E-05	1,03E-05	0,18	**0,182**
8	1,36E-04	1,33E-04	1,53E-05	0,18	**0,183**
16	2,72E-04	2,67E-04	2,53E-05	0,18	**0,183**
32	5,44E-04	5,33E-04	4,53E-05	0,36	**0,366**
64	1,09E-03	1,07E-03	8,53E-05	0,73	**0,731**
128	2,18E-03	2,13E-03	1,65E-04	1,46	**1,462**
256	4,35E-03	4,27E-03	3,25E-04	2,84	**2,849**
512	8,70E-03	8,53E-03	6,45E-04	5,24	**5,258**
1024	1,74E-02	1,71E-02	1,29E-03	10,04	**10,076**

Esta tabla se puede obtener a través del mismo Excel, el cual se puede encontrar en el repositorio de Git para que cualquier persona pueda hacer las mismas estimaciones y ver que, con ciertos parámetros fijados, se puede obtener cuánto puede costar el ejecutar la librería en AWS e incluso ver qué servicio genera más gasto.

Para dar mayor validez a esta calculadora, se han realizado ejecuciones reales que confirman que el costo estimado está en línea con los cálculos realizados por AWS, tal como se aprecia en la Figura 4-2. De este modo, se puede afirmar que la calculadora desarrollada ofrece valores que son razonablemente consistentes con los costos generados en AWS por el uso de la librería.

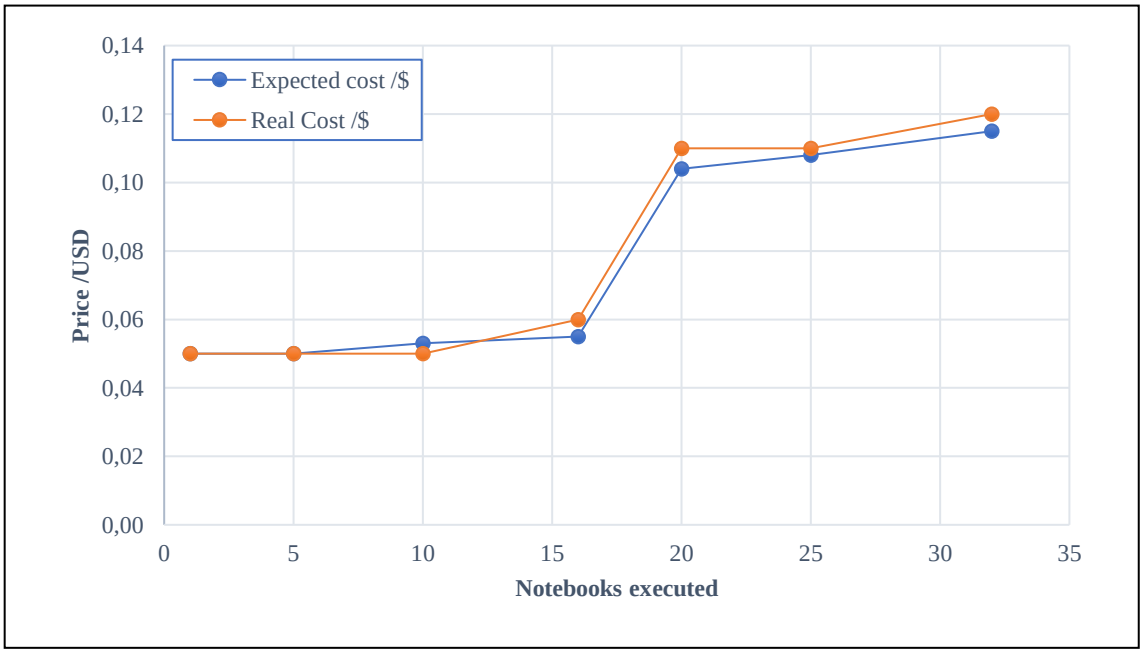

Figura 4-2: Comparación de los precios estimados y reales al ejecutar un distinto número de notebooks.

4.1.3. Costo de AWS con instancias reservadas de EC2

AWS ofrece descuentos significativos para aquellas empresas que puedan anticipar sus necesidades de recursos y optan por reservar ciertos servicios con antelación. Por ejemplo, en este proyecto, se puede prever la necesidad de 16 instancias m4.4xlarge, lo que permite acceder a un precio especial mediante la reserva anticipada. En la Figura 4-3 se muestra una estimación de costos, usando el *Pricing Calculator* de AWS, para la reserva de estas instancias durante 3 años. Como se puede observar, el gasto estimado asciende a unos $139136 USD, lo que, según los cálculos de AWS, podría representar un ahorro de hasta un 72%[5].

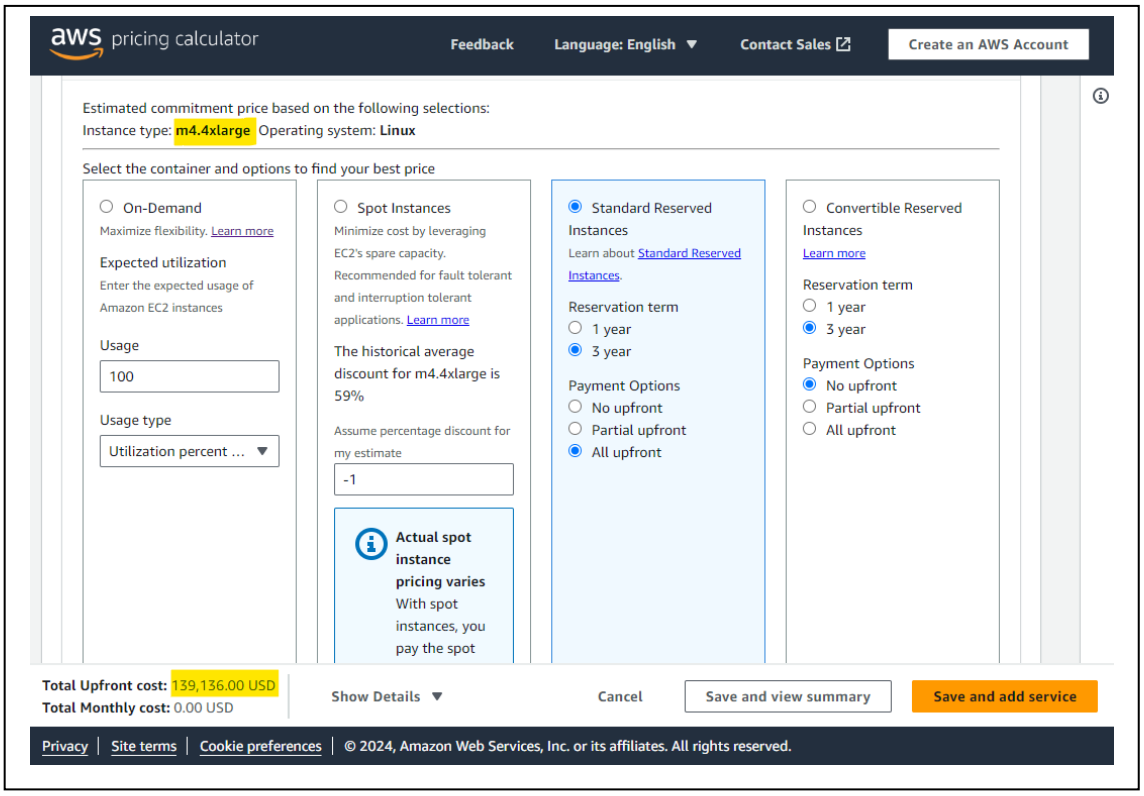

Figura 4-3: Estimación de costos utilizando la calculadora de AWS para la reserva de 16 instancias durante 3 años, con pago total anticipado.

Este precio corresponde exclusivamente al uso de las instancias EC2 durante un periodo de 3 años, independientemente de si se utilizan las instancias de manera continua o no. Es importante tener en cuenta que este costo no incluye los servicios adicionales que la librería requiere para funcionar correctamente. Sin embargo, para simplificar los cálculos, se ha decidido no incluir estos servicios adicionales, ya que, como se ha demostrado previamente en la Tabla 4-3, su impacto en el costo total es mínimo en comparación con el de las instancias EC2.

4.2. Coste de infraestructura propia en *colocation*

La opción de utilizar un clúster propio en un centro de *colocation* representa una alternativa que, aunque requiere una inversión inicial significativa, podría ofrecer ahorros a

[5] https://aws.amazon.com/es/ec2/pricing/reserved-instances/

largo plazo. En esta sección, se detallan los costos asociados con la adquisición, implementación y operación de una infraestructura propia.

4.2.1. Costos de Hardware y adquisición

Se planea adquirir un clúster con características equivalentes a las instancias m4.4xlarge de AWS, como se detalla en la Tabla 3-1. Al analizar los precios en el mercado, se observa que la adquisición de servidores con estas especificaciones requiere un desembolso inicial considerable. Este costo puede fluctuar según el fabricante, la configuración detallada del servidor y los componentes adicionales necesarios, pero en general representa una inversión significativa en infraestructura de alto rendimiento. En la Tabla 4-4 se detallan los componentes, especificaciones y precios estimados para los diferentes elementos necesarios para construir un clúster con características similares a las deseadas. Es importante destacar que estos precios han sido obtenidos en la fecha de redacción de esta memoria y reflejan valores estándar del mercado en ese momento.

Tabla 4-4: Componentes, especificaciones y precios para montar un clúster local comparable al de AWS.

Componente	Especificaciones	Cantidad	Costo por Unidad /USD	Costo Total /USD
Procesador	Intel Xeon E5-2686 v4 2.3GHz 18 Core 45MB	8	71	568
Placa base	Supermicro X10DRi	4	180	720
Memoria RAM	32GB DDR4 2133MHz	18	60	1080
Almacenamiento	2TB Internal SSD SATA	9	170	1530
Chasis/Rack Unit	2U Server Chassis	4	2052	8208
Otros	Cables, enfriamiento, ventilación, etc.	8	200	1600
		TOTAL	**2733**	**13706**

Como se puede apreciar, para adquirir un clúster con características similares a las instancias utilizadas en Amazon, se estima que sería necesario un desembolso inicial de aproximadamente 14,000 dólares. Si cada servidor ocupa 2U y se tiene un total de 4 *blades*, estos ocuparían 8U en total; por lo tanto, podrían alojarse cómodamente en un único rack, dado que un rack estándar ofrece 42U de espacio disponible.

4.2.2. Costos de energía y colocation

Además del costo de adquisición, en la infraestructura propia necesitamos calcular el costo de operación, derivado del consumo energético de las máquinas, así como de su refrigeración. Finalmente, habrá que añadir el costo del alquiler de espacio en el centro de *colocation*.. Esto es necesario porque algunas empresas de *colocation* cobran por el consumo energético realizado a lo largo del mes y el ancho de banda utilizado como parte de gasto, a parte del espacio que ocupa en sus instalaciones. La estimación precisa de costos de esta sección es un proceso extremadamente complejo. Debido a que nuestro interés es obtener un valor aproximado que nos permita entender si es conveniente seguir adelante con el proyecto, se harán ciertas aproximaciones que permitan simplificar el proceso de manera importante.

Con el Hardware seleccionado, se va a calcular el consumo que va a suponer el uso del clúster para distintos porcentajes de carga. Si bien el equipo va a mantenerse encendido siempre, no siempre va a tener trabajo que realizar. Generalmente, los distintos componentes de un sistema informático consumen una cantidad de energía relativamente constante, independientemente de la carga de trabajo que estén manejando; la excepción notable es el procesador y los sistemas de refrigeración asociados. A medida que aumenta la carga de la CPU, no solo su consumo energético se incrementa, sino también el consumo de los sistemas de refrigeración necesarios para mantener la temperatura adecuada del sistema. Por esta razón, en la estimación del consumo energético, se asume un consumo constante para todos los componentes excepto para la CPU y el sistema de refrigeración, como se detalla en la Tabla 4-5. La categoría "Otros consumos" incluye el consumo energético de las placas base, módulos de memoria RAM, unidades SSD, chasis, sistemas de refrigeración, cables y otros componentes que mantienen el clúster operativo. Estos valores son estimativos y pueden variar dependiendo de las condiciones específicas de operación, pero representan una aproximación bastante precisa del consumo energético del clúster.

Tabla 4-5: Estimación del consumo energético del clúster en diferentes niveles de carga del sistema.

System Load /%	***CPU Consumption /W***	***Other consumptions /w***	***Total Consumption/W***
10	*116*	901	**1017**
25	*290*	1075	**1365**
50	*580*	1365	**1945**
75	*870*	1655	**2525**
100	*1160*	1945	**3105**

Se ha determinado que el TDP (*Thermal Design Power*) de la CPU es de 145W cuando opera a plena carga (100% de uso). Para estimar el consumo en otros estados de carga, se ha calculado proporcionalmente en función de este valor máximo. Para un procesador en estado idle, se asume un consumo aproximado del 20% de su TDP máximo. Por otro lado, el consumo de los demás componentes del clúster, como las placas base, módulos de memoria RAM y unidades de almacenamiento, ha sido estimado en aproximadamente 800W.

Además, se ha supuesto, de manera simplista, que la refrigeración consume un valor equivalente al 100% del consumo de la CPU. Sin embargo, es importante destacar que esta estimación puede no reflejar con precisión la realidad, ya que el consumo de refrigeración varía según diversos factores, como la eficiencia del sistema de enfriamiento, las condiciones ambientales y el tipo de infraestructura utilizada en el clúster.

Con toda esta información, se procede a encontrar empresas de *colocation* que ofrezcan precios competitivos para alojar un chasis de 8U con un consumo entre 1.1 y 3.1kWh. Se ha contactado con varias empresas especializadas, solicitando una simulación o presupuesto para el alojamiento de un clúster con estos recursos. Sin embargo, muchas de estas compañías requieren información adicional, como detalles específicos de la em-

presa, el propósito del uso o incluso un compromiso previo. Una de las compañías que proporciona acceso público a sus precios es *ServerMania*[19], una empresa canadiense con más de una década de experiencia en el suministro de plataformas de alojamiento de infraestructura de alto rendimiento para empresas de todo el mundo. Por esta razón utilizaremos sus datos como valores de referencia.

La Figura 4-4 muestra el presupuesto para una configuración en Montreal, que incluye 1 cabina o rack de 48U con un consumo de hasta 3 kWh. Por un costo adicional de 75 dólares al mes, se ofrece un ancho de banda constante de 1 Gbps sin medición precisa ni limitación basada en percentil (que es la opción gratuita y predeterminada). En total, el primer pago sería de 3850 dólares, con un costo mensual posterior de 850 dólares. Este precio es orientativo y puede ser negociado dado que no se va a usar el rack en su totalidad y, al optar por una facturación anual, se puede obtener una mejor oferta.

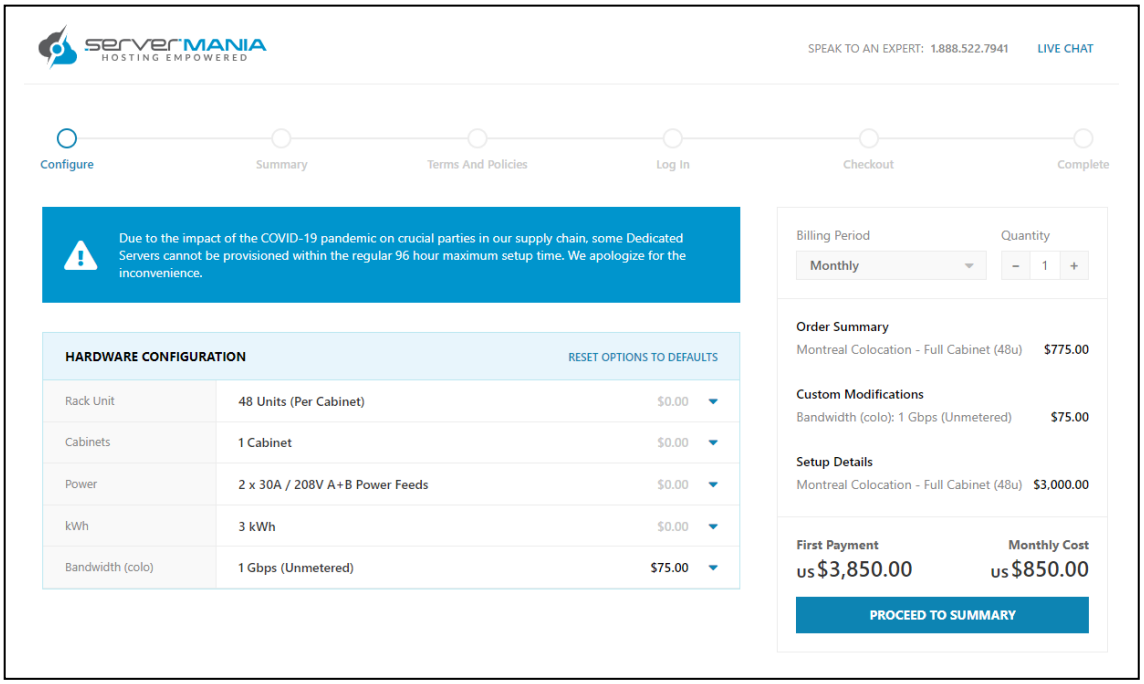

Figura 4-4: Presupuesto de colocation para 1 cabina (48 U) en Montreal por parte de ServerMania.

En caso de que se requiera más energía adicional a los 3 kWh asignados, como es nuestro caso, es necesario contactar directamente con el proveedor para obtener una cotización personalizada. No obstante, este aspecto se ha omitido en este análisis, asumiendo que el ahorro potencial por contratar el servicio de *colocation* de forma anual (estimado en un 10%) compensaría los costos adicionales derivados del aumento en el consumo energético.

4.3. Comparativa de costes totales

En la Figura 4-5 se muestra el coste mensual de la infraestructura de cálculo para las diferentes alternativas de implementación, ya sea en la nube (AWS) o en *colocation* (Local). El eje Y representa los costes normalizados para distintos niveles de carga, desde un 100% (todos los procesadores en uso durante todo el mes) hasta un 10% (todos los procesadores en uso solo el 10% del tiempo).

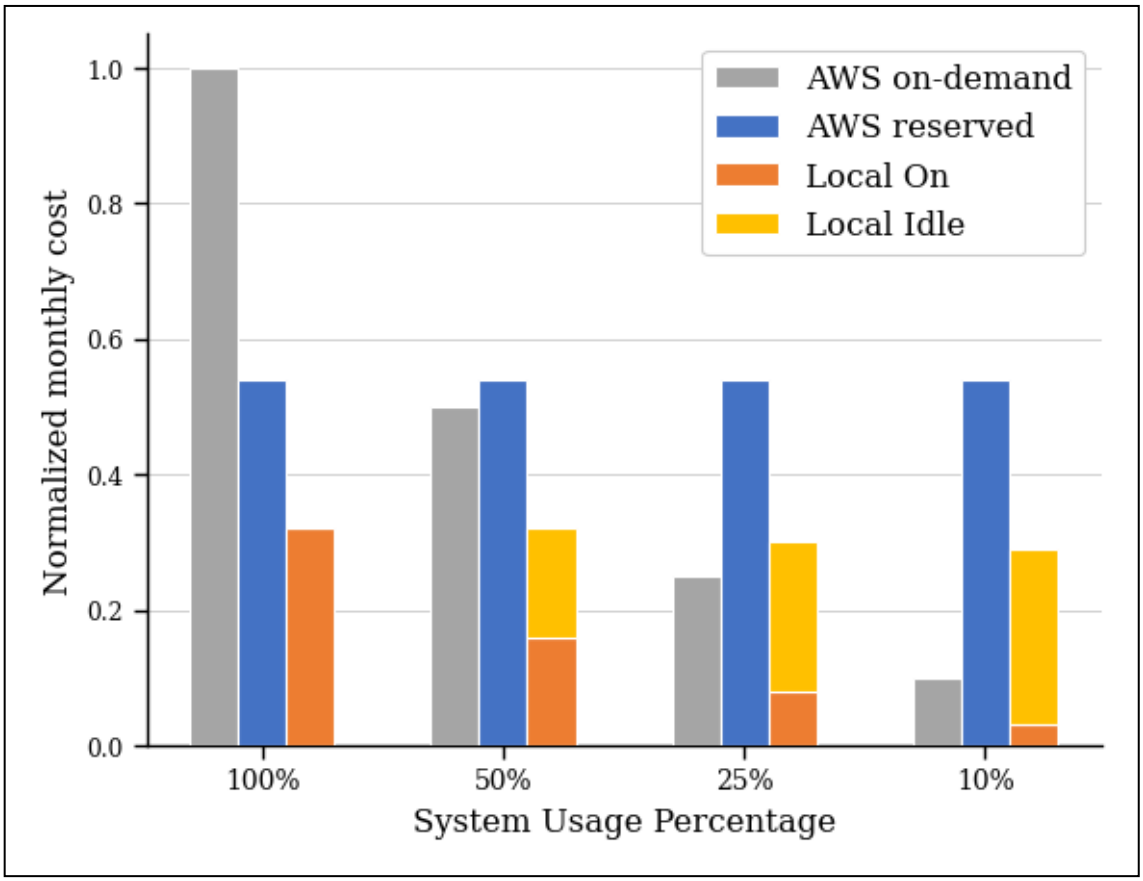

Figura 4-5: Comparación de los precios normalizados mensuales entre AWS y un clúster en local con colocation, según distintos porcentajes de uso del sistema.

El valor correspondiente a *AWS on-demand* refleja los gastos estimados utilizando la calculadora desarrollada (ver Sección 4.1.2) para un mes completo. Por ejemplo, ejecutar 144 notebooks durante 10 minutos tiene un costo aproximado de $1,65 USD. La mayor parte de este gasto (más del 99%) está relacionado con el uso de las instancias EC2. Extrapolando estos costos al número total de minutos de un mes, se obtiene el 100% de uso del sistema. En consecuencia, si el sistema solo se utiliza un 50% del tiempo, el costo será proporcional, pagando solo por el tiempo en que las EC2 están en uso.

Por otro lado, como se explicó en la Sección 4.1.3, el uso de instancias reservadas ofrece un ahorro significativo al pagar las instancias por un periodo anual. En este caso, el porcentaje de utilización del sistema es irrelevante, ya que cada mes se paga una fracción fija del costo acordado al reservar las instancias.

Para el cálculo local, la gráfica muestra por separado el costo de adquisición y operación. El costo de adquisición mensual se ha calculado en base a los valores de la Tabla 4-4, asumiendo un tiempo de vida útil de 5 años. Como se menciona en la Sección 4.2.2, el costo de operación incluye el consumo energético y los costes de *colocation* (ver Figura 4-4). A partir de estos valores, se ha estimado el costo total de operar el clúster de manera anual y mensual. Es importante destacar que existe un coste fijo asociado a la operación (representado por las barras amarillas y naranjas en el gráfico). Este

coste está presente tanto si el sistema se utiliza completamente (100%) como si está en estado de inactividad parcial (10%).

Los resultados obtenidos son coherentes con lo esperado: la ventaja principal de los entornos en la nube radica en que el costo es estrictamente por uso. Por lo tanto, en entornos donde las máquinas no se utilizan la mayor parte del tiempo, las infraestructuras locales son más caras. Sin embargo, con nuestros cálculos preliminares de costos, podemos observar dos aspectos clave: primero, la ubicación aproximada del punto de inflexión en términos de rentabilidad, y segundo, la cota máxima de ahorro. Aproximadamente, cuando el uso del sistema cae por debajo del 50%, la opción en la nube se vuelve más económica. En cambio, con un uso completamente eficiente de la infraestructura local (100%), el coste que la empresa dedica al cálculo se podría reducir a algo menos de la mitad, en una estimación conservadora.

En conclusión, la infraestructura local resulta ser más eficiente en términos de costos cuando se opera a plena capacidad (100% de uso), ya que presenta un menor coste en comparación con las opciones de AWS. Sin embargo, a medida que la carga de trabajo disminuye (hacia el 50% o menos), la nube se vuelve más competitiva, especialmente cuando se utilizan instancias reservadas. Esto sugiere que, para empresas con cargas de trabajo fluctuantes, AWS puede ser una opción más rentable a largo plazo.

Volviendo al objetivo principal, donde *Parallel-Scripts* se ha desarrollado para que los *quants* puedan aplicar sus modelos de datos sobre el universo de compañías que cotizan en la bolsa de valores, es fundamental analizar los escenarios de uso.

Consideremos el peor de los casos, donde el modelo de datos se ejecuta constantemente durante el horario de apertura del mercado, es decir, de 9:30 a.m. a 4:00 p.m., de lunes a viernes. Para un mes con 21 días hábiles, esto supone aproximadamente un 20% del tiempo total mensual. Si añadimos el tiempo dedicado al desarrollo y ajuste (*fine-tuning*) del modelo, podríamos estimar un uso adicional del 20%, lo que deja a la empresa financiera utilizando la infraestructura local (y la librería) un 40% del mes, en un escenario optimista. Este nivel de uso generaría costos similares en ambos entornos.

Por lo tanto, la conclusión más sensata sería la de no tomar el riesgo de optar por una infraestructura local a menos que la carga de trabajo de la empresa fuera constante y altamente predecible. En el escenario planteado, los ahorros que ofrece el clúster local no son lo suficientemente significativos como para justificar el riesgo, la falta de flexibilidad y los costos fijos elevados que trae consigo. La capacidad de escalar rápidamente, el modelo de pago por uso y la eliminación de riesgos asociados con el mantenimiento de hardware en AWS ofrecen una mayor tranquilidad y mejor alineación con el crecimiento o los cambios en la demanda de la empresa.

Por lo tanto, AWS sería la opción preferida debido a su flexibilidad y capacidad de adaptación, especialmente si la carga de trabajo de la empresa no es elevada, fluctúa o es impredecible. Si bien la opción local puede parecer más barata en algunos escenarios específicos, la flexibilidad y la reducción de riesgos que ofrece AWS justifican el gasto adicional, y protegerían mejor a la empresa ante cambios inesperados en la demanda.

5 Conclusiones y líneas futuras

En este trabajo se ha desarrollado una herramienta, *Parallel-Scripts*, que permite la generación y ejecución automatizada de *Jupyter Notebooks* en múltiples entornos, incluyendo local, en la nube (AWS) y en un clúster propio (UC). Esta librería, que se encuentra disponible en GitHub, ha sido sometida a diversas pruebas y se ha demostrado que puede desplegarse en ambos entornos con relativa facilidad. A través del análisis comparativo de costos, se ha demostrado que, si bien AWS ofrece una solución robusta, escalable y flexible, el clúster en *colocation* puede representar una alternativa más económica para cargas de trabajo intensivas y de larga duración.

Las pruebas realizadas confirman que la mayor parte del costo en AWS proviene de las instancias EC2, lo que subraya la importancia de optimizar el uso de estas instancias para reducir gastos. Por otro lado, aunque la infraestructura en *colocation* requiere una inversión inicial importante, muestra una reducción significativa en los costos a largo plazo, especialmente con una alta utilización. No obstante, se ha identificado que, en el mejor de los casos, la empresa usaría la infraestructura local durante un 40% del tiempo, lo que generaría costos similares en ambos entornos.

En este contexto, si la carga de trabajo es variable o impredecible, AWS resulta más flexible y seguro. Su modelo de pago por uso y capacidad de escalar recursos lo hacen una opción más eficiente frente a una infraestructura local que, en este caso, no ofrece ahorros suficientes para compensar sus costos fijos.

A lo largo del desarrollo de la librería se han incorporado mejoras clave para optimizar el rendimiento y gestión de recursos. Por ejemplo, se ha implementado la agrupación de llamadas a la API de AWS (mediante boto3) para eliminar los archivos residuales en S3. Sin embargo, otras mejoras no son triviales y requieren un estudio más exhaustivo. Un caso particular es la decisión de utilizar un contenedor para ejecutar cada notebook, donde la elección óptima depende estrechamente de las operaciones realizadas por cada notebook. Algunos son exhaustivos en términos de cómputo, otros en I/O y otros incluso pueden aprovechar el paralelismo. Lo ideal sería encontrar un equilibrio y ajustar el número de notebooks por contenedor, una idea que se propone como trabajo para futuras investigaciones y desarrollos de la librería.

Este trabajo representa una primera aproximación, y se recomienda ampliar las pruebas y establecer contacto con otras empresas de *colocation* y proveedores de servidores para obtener valores más precisos. Sin embargo, la aproximación inicial sugiere que el uso de esta librería tendría costos menores en un clúster propio en *colocation*. Además, se propone la valoración y evaluación de entornos híbridos que combinen lo mejor de ambos mundos: la escalabilidad de AWS y el costo reducido de una infraestructura propia en *colocation*. Este enfoque podría proporcionar mayor flexibilidad y optimización de costos.

6 Referencias

[1] V. Mayer-Schönberger, K. Cukier, “Big Data: A Revolution that Will Transform how We Live, Work and Think”, John Murray, 2013, ISBN: 1848547919, 9781848547919.

[2] "The Importance of Big Data and Analytics in Business", Forbes, accedido por última vez el 15 de junio de 2024.

[3] T.H. Davenport, J.G. Harris, "Competing on Analytics: The New Science of Winning", Harvard Business Review Press, 2007, ISBN: 1422103323, 9781422103326.

[4] “Dropbox Saved $75M Moving from Cloud to Colocation”. http://www.investopedia.com/terms/s/sp500.asp, consultado por última vez el 16 de julio de 2024.

[5] “Standard & Poor's 500 Index - S&P 500”. Investopedia: https://www.hostdime.com/blog/move-from-cloud-to-colocation/sp, consultado por última vez el 11 de junio de 2024.

[6] https://aws.amazon.com/es/s3/, accedido por última vez el 11 de junio de 2024.

[7] https://aws.amazon.com/es/lambda/, accedido por última vez el 11 de junio de 2024.

[8] https://aws.amazon.com/es/dynamodb/, accedido por última vez el 11 de junio de 2024.

[9] https://aws.amazon.com/es/batch/, accedido por última vez el 11 de junio de 2024.

[10] https://aws.amazon.com/es/ec2/, accedido por última vez el 11 de junio de 2024.

[11] https://aws.amazon.com/es/ecr/, accedido por última vez el 11 de junio de 2024.

[12] https://slurm.schedmd.com/quickstart_admin.html, accedido por última vez el 16 de agosto de 2023.

[13] https://aws.amazon.com/es/ec2/instance-types/, accedido por última vez el 14 de Julio de 2024.

[14] https://aws.amazon.com/es/s3/pricing/?nc=sn&loc=4, accedido por última vez el 14 de Julio de 2024.

[15] https://aws.amazon.com/dynamodb/pricing/on-demand/, accedido por última vez el 14 de Julio de 2024.

[16] https://aws.amazon.com/lambda/pricing/?p=pm&c=la&z=4, accedido por última vez el 14 de Julio de 2024.

[17] "Slurm Workload Manager Overview", SchedMD, https://slurm.schedmd.com/overview.html, accedido por última vez el 1 de septiembre de 2023.

[18] "Paramiko Documentation", https://docs.paramiko.org/en/stable/, accedido por última vez el 1 de septiembre de 2023.

[19] ServerMania. "Colocation Hosting Services." ServerMania, https://www.servermania.com/colocation, accedido por última vez el 3 de julio de 2024.

[20] .