Diseño e Implementación de una arquitectura orientada a eventos en Kubernetes para la plataforma cognitiva MIRIAM.

Daniel Felipe Rivera Arroyave

Informe de práctica empresarial, como requisito para optar por el título de Ingeniero de Sistemas.

Asesor
Joan Andrés Hasper Tabares, Magister Administración y gestión de empresas
Martín Elias Quintero Osorio, Ingeniero de Sistemas

Universidad de Antioquia
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS
MEDELLÍN
2022

Cita	Rivera Arroyave [1]
Referencia Estilo IEEE (2020)	[1] D. Rivera Arroyave, “Diseño e Implementación de una arquitectura orientada a eventos en Kubernetes para la plataforma cognitiva MIRIAM”, Semestre de industria, Ingeniería de Sistemas, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2022.

AGRADECIMIENTOS

A la Universidad de Antioquia, porque gracias a ella hoy puedo aspirar a mi título profesional, adquirir infinidad de conocimiento y desarrollarme en un mundo que me apasiona, gracias totales.

A la empresa Guane Enterprises por el apoyo y confianza para participar en sus proyectos y así culminar mi proceso de formación profesional.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

ETL	Extract, transform and load
K8s	Kubernetes
UdeA	Universidad de Antioquia
ASB	Azure Service Bus
ABS	Azure Blob Storage
NER	Named Entity Recognition

RESUMEN

La arquitectura basada en eventos es ampliamente popular debido a su impacto en el desarrollo de software, debido a que los procesos dentro de esta arquitectura son asíncronos y distribuidos y la hace ideal para desarrollar sistemas altamente escalables, esto se acomoda perfectamente a procesos de ETL (extracción, transformación y carga) porque se puede implementar un punto de entrada que reciba los datos de un publicador y delegar los pasos siguientes a los módulos suscriptores correspondientes, así desentendiéndose del resultado del siguiente paso porque de eso se encargará otro suscriptor y cumpliendo únicamente su labor cuando le llegan tareas. Este documento aborda la implementación de un proceso ETL para contratos logísticos a través de la plataforma de gestión cognitiva MIRIAM.

Como resultado final de este proyecto, se desarrolló una nueva versión para la plataforma MIRIAM, implementando una arquitectura basada en eventos con componentes dockerizables y capaz de ejecutarse dentro de un entorno de Kubernetes, el proceso comienza con un servicio productor que toma de un tópico la información de los contratos logísticos a procesar, se encarga de enviar tareas a una cola de mensajería a la que están suscritos workers de Celery los cuales desencadenan todas las labores de procesamiento del documento para que al finalizar sea presentado a los usuarios en la interfaz gráfica. Dicho aplicativo se encuentra desplegado en un cluster de kubernetes mediante servicios en la nube ofrecidos por Microsoft Azure y se encuentra próximo a ser usado en un ambiente completamente de producción.

.

***Palabras clave*: Kubernetes, ETL, Arquitectura basada en eventos, Trabajador de celery, Desarrollo de software, Docker.**

ABSTRACT

The event-driven architecture is widely popular due to its impact on software development, because the processes within this architecture are asynchronous and distributed and makes it ideal for developing highly scalable systems, this is perfectly suited to ETL processes (extraction, transformation and load) because you can implement an entry point that receives data from a publisher and delegate the following steps to the corresponding subscriber modules, thus disregarding the outcome of the next step because that will take care of another subscriber and fulfilling only their work when tasks arrive. This paper addresses the implementation of an ETL process for logistics contracts through the MIRIAM cognitive management platform.

As a final result of this project, a new version for the MIRIAM platform was developed, implementing an event-based architecture with docker components and capable of running within a Kubernetes environment, the process starts with a producer service that takes from a topic the information of the logistic contracts to be processed, it is responsible for sending tasks to a messaging queue to which Celery workers are subscribed and which trigger all the processing tasks of the document to be presented to the users in the graphical interface at the end. This application is deployed in a Kubernetes cluster through cloud services offered by Microsoft Azure and is about to be used in a fully production environment.

***Keywords*: Kubernetes, ETL, Event-driven architecture, Celery worker, Software Developing, Docker.**

I. INTRODUCCIÓN

Guane Enterprises SAS, es una compañía que ha tenido un crecimiento significativo en los últimos años y que ofrece soluciones a problemas de la industria y automatización por medio de desarrollo de sistemas informáticos aplicando herramientas de ciencia de datos, creando así sistemas cognitivos. Esta empresa se desenvuelve en una serie de sectores económicos principales, dentro de los que destacan el sector energético y el sector logístico, en este último es donde tiene cabida este trabajo donde se partió de una aplicación ya existente y se mejoró.

Uno de los principales retos a los que se enfrentan empresas del sector logístico es la gestión documental de grandes volúmenes contratos, proceso en el que deben recibir un documento, revisarlo en primera instancia, aprobarlo, posteriormente extraer la información proveniente de este a un formato estandarizado y persistir de manera que permita a las empresa realizar sus operaciones de una forma controlada, proceso que se conoce como ETL [1], sin embargo al manejar un alto flujo de contratos y que además provienen de distintos clientes con plantillas y formatos de distintos hacen que esta requiera gran cantidad de tiempo en horas humanas. De este problema surge la plataforma MIRIAM que pretende solventarlo ofreciendo un sistema cognitivo [2] mediante el uso de técnicas de lenguaje natural y aplicación de reglas duras tiene la capacidad de realizar automáticamente los pasos iniciales a los que se someten los contratos logísticos.

En la actualidad, las arquitecturas de software tradicionales no son suficientes para cubrir por completo las necesidades que el desarrollo moderno de software requiere [3], lo cual se hace más notorio cuando el software requiere realizar complejas operaciones de procesamiento de datos que pueden tardar varios minutos en resolverse generando así problemas para realizar procesos o atender solicitudes de forma simultánea. En particular, la dificultad para escalar y mantener una aplicación construida bajo las arquitecturas tradicionales ha llevado a explorar nuevas estrategias que permitan superar las desventajas de los monolitos [4] dentro de las que se encuentran las arquitecturas orientadas a eventos.

En este trabajo se desarrolló una nueva arquitectura basada en eventos dentro de un cluster de K8s[5] capaz de soportar los procesos internos y de alto costo computacional que se ejecutan dentro de la plataforma cognitiva ya existente MIRIAM, mejorando los tiempos de procesamiento, escalabilidad y elasticidad de la misma implementando al experimentar grandes cargas de trabajo

sin afectar los tiempos de respuesta, se utilizó una metodología donde el trabajo era dividido en distintas etapas mientras que a la vez se aplicaba la metodología de desarrollo ágil SCRUM [6] , finalmente se concluyó con la migración completa de la plataforma utilizando workers de Celery [7], se realizó un perfilamiento y optimización de técnicas para el escalamiento de los componentes de la plataforma.

II. OBJETIVOS

A. Objetivo general

Diseñar e implementar una arquitectura distribuida y orientada a eventos dentro de Kubernetes apoyada en servicios cloud para la plataforma MIRIAM, encargada de la gestión de documentos y apalancada en inteligencia artificial para el análisis y trazabilidad de contratos logísticos.

B. Objetivos específicos

- Realizar una revisión bibliográfica de trabajos e implementaciones similares para comprender distintos enfoques respecto a características, arquitectura, persistencia de datos y tecnologías.
- Diseñar una arquitectura general del sistema que esté orientada a eventos y colas de mensajería, apoyada por servicios en la nube y alojada en Kubernetes que satisfaga los requisitos y necesidades del cliente.
- Planificar y desarrollar la implementación de la arquitectura, refactorización y desacoplamiento en distintos módulos y elementos.
- Validar mediante pruebas funcionales, integración el correcto desarrollo de la aplicación y beneficios de la arquitectura orientada a eventos.
- Definir mediante pruebas de carga y estrés los límites de rendimiento de la solución, factores de escalabilidad horizontal y posibles optimizaciones.

III. MARCO TEÓRICO

El desarrollo de este proyecto desde un punto de vista técnico abarca diversos temas conceptuales dentro del diseño y desarrollo de software, los conceptos más importantes son descritos a continuación para acentuar la idea y el por qué son relevantes dentro del proyecto.

Arquitectura orientada a eventos

Parte del objetivo del proyecto es diseñar e implementar una arquitectura orientada o basada en eventos, tal como describen diversos referentes en la industria del software como: RedHat o Microsoft, este patrón se compone de productores de eventos y consumidores de eventos y provee dos modelos de implementación, el primero es Pub/Sub que según RedHat “Es una infraestructura de mensajería que se basa en suscripciones a un flujo de eventos. Con este modelo, una vez que se genera o publica un evento, este se envía a los suscriptores que necesitan estar informados al respecto” [8] mientras que según Microsoft en el segundo modelo llamado Event Streaming “los eventos se escriben en un registro. Los eventos siguen un orden estricto (dentro de una partición) y son duraderos. Los clientes no se suscriben al flujo, sino que un cliente puede leer desde cualquiera de sus partes. El cliente es responsable de avanzar su posición en el flujo. Esto significa que un cliente puede unirse en cualquier momento y puede reproducir los eventos”. [9]

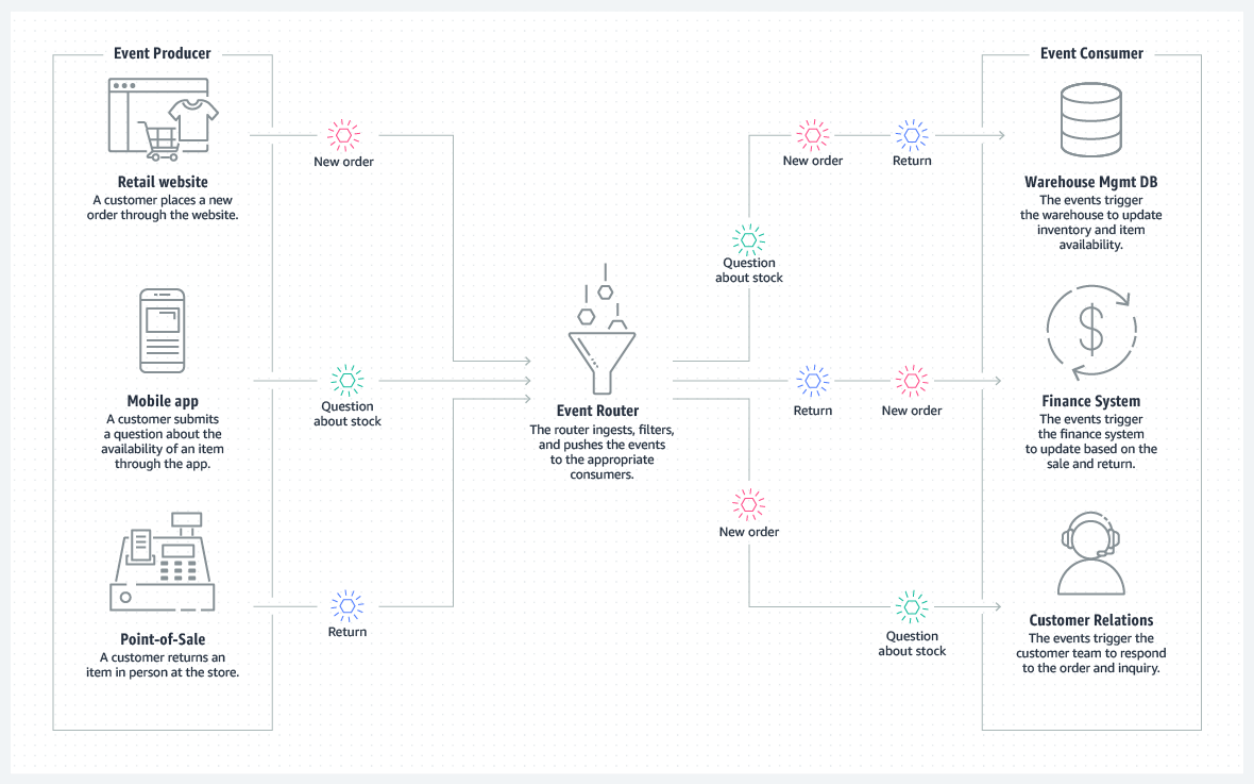

Fig 1. Ejemplo arquitectura orientada a eventos

Nota: Tomado de [10]

Patrón de diseño competencia de consumidores

Uno de los patrones de diseño que serán de utilidad en la implementación para dividir y distribuir la carga de trabajo es “Competing Consumers” [11] el cuál plantea que cuando es necesario atender múltiples solicitudes y estas toman alto tiempo de procesamiento la mejor estrategia es dar lugar a un servicio consumidor mediante un sistema de tareas y mensajería, donde el consumidor se encargue de realizar el procesamiento de una forma asíncrona y liberar de esa carga al primer servicio, de esta manera el primer servicio no estará bloqueado mientras se realicen las instrucciones de procesamiento. El potencial de este patrón radica en la posibilidad de tener varios consumidores activos y a la espera de procesar las tareas correspondientes a la misma cola de mensajería y de esta forma puede procesar de forma simultánea distintas tareas más de una tarea, mejorando así la escalabilidad.

Uso de contenedores y Kubernetes

El uso de contenedores ofrece múltiples ventajas, dentro de las que se destaca que "ofrecen un mecanismo de empaquetado lógica en el que las aplicaciones se pueden abstraer del entorno en que se ejecutan" [12] lo que brinda versatilidad para su inclusión en arquitecturas modernas. Sin embargo, cuando se utilizan gran cantidad de ellos, se hace necesario contar con un sistema de gestión y administración de los procesos ejecutados por ellos mismos, es allí donde sistemas como Kubernetes, una plataforma de orquestación de contenedores de código abierto, cumple un papel muy importante en la industria del software. Kubernetes [5] ofrece un entorno de administración centrado en contenedores, de manera que orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo no dependan directamente del soporte humano de los usuarios, esto quiere decir que Kubernetes proporciona un ecosistema de componentes y herramientas que facilitan el proceso de desplegar, mantener, escalar, interconectar y administrar aplicaciones.

Celery workers

Celery es una biblioteca para el lenguaje de programación Python para la gestión de colas de tareas distribuidas. Es maduro, rico en funciones y debidamente documentado. Es muy adecuado para servicios Back-End de Python escalables debido a su naturaleza distribuida, lo cual lo hace altamente acoplable con la propuesta arquitectónica, ayudando en el desarrollo de los módulos correspondientes a productores y suscriptores con su implementación a alto nivel para enviar y recibir tareas, para su correcto funcionamiento es necesario el uso de un bróker [13] que se ocupa de la infraestructura sobre la cual se almacenan los mensajes de tareas y de la información.

IV. METODOLOGÍA

En el proyecto se implementó un modelo de casada dividido en 4 fases tal como se muestra en la Fig. 3.

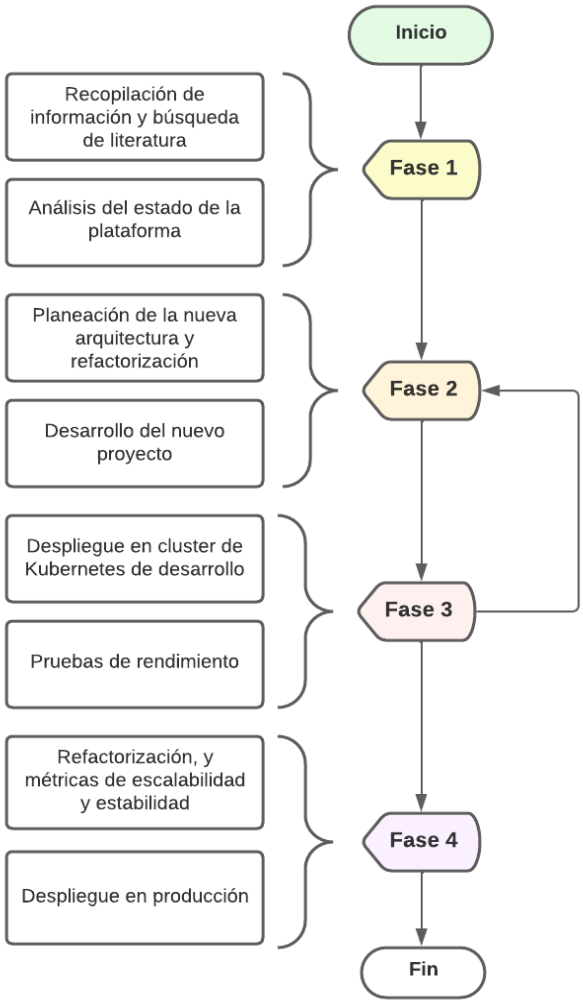

Fig 2. Metodología del proyecto

Fase 1: Estado del arte y comprensión de la plataforma

La primera fase comenzó con una contextualización completa sobre el proyecto, los modelos implementados y las reglas de negocio que se cumplen en este, además de comprender los principales limitantes que poseía y la razón de ser de la idea de implementar una nueva

arquitectura, se realizó una búsqueda de la literatura sobre plataformas similares y posibles soluciones a los problemas que presentaba, desde el equipo de arquitectura se plantearon unas posibles soluciones tecnológicas por lo que se procedió a comprender el funcionamiento de dichas tecnologías, tales como el uso de Docker, K8s, arquitecturas basadas en eventos y uso de Celery junto a RabbitMQ [13], entre otros.

Fase 2: Planeación y desarrollo del proyecto

Antes de idear la arquitectura con las nuevas tecnologías se analizó el proyecto, a partir del análisis se llegó a la conclusión que lo óptimo de implementar en este proyecto era un sistema de colas de tareas. Donde cada trabajador asíncrono se encargará de un paso del proceso. de esta manera se puede crear réplicas de los trabajadores asíncronos que lo requieran y operen de manera distribuida sin tener que escalar toda la aplicación en conjunto.

Una vez constituida formalmente la nueva arquitectura se ideó un plan de desarrollo y mejoramiento y comenzó y comenzó con el desarrollo siguiendo una metodología de SCRUM para su implementación, se separó el proyecto en distintos módulos donde cada uno era un suscriptor asíncrono con la lógica de negocio desacoplada.

Fase 3: Entorno de desarrollo y evaluación de rendimiento

A mediados de la fase 2, conforme se fueron cumpliendo las historias de usuario e implementando la arquitectura se aprovisionó con un cluster de Kubernetes en Azure en donde se validaron las funcionalidades. Esta fase fue complementaria para la fase 2, y produjo un proceso reiterativo donde al final de cada sprint de la fase 2, se trabajaba en esta tercera sucesivamente hasta culminar el proceso inicial de desarrollo. Finalmente, con la fase 2 completada se procedió a realizar un perfilamiento completo del desarrollo dentro de la infraestructura de desarrollo para validar los tiempos de mejora, estabilidad y elasticidad de la plataforma, pruebas end to end, se concluyó con un reporte con los resultados y aspectos a mejorar.

Fase 4: Elasticidad de la plataforma y puesta en producción

Partiendo de los resultados de la fase anterior, se designó una primera parte de esta fase para realizar pequeñas mejoras y pagar la deuda técnica resultante de la fase 2, se establecieron los límites de recursos de CPU y RAM para cada uno de los módulos, así como los recursos de K8s

que permiten que la plataforma experimente un adecuado escalamiento horizontal y vertical, posteriormente se elaboró una guía para que el cliente pudiera realizar el despliegue en producción de la plataforma dentro de su propia infraestructura.

V. RESULTADOS

Arquitectura implementada

Un diagrama a alto nivel de la arquitectura implementada se presenta en la Fig. 4 y se procede a dar una explicación de la misma.

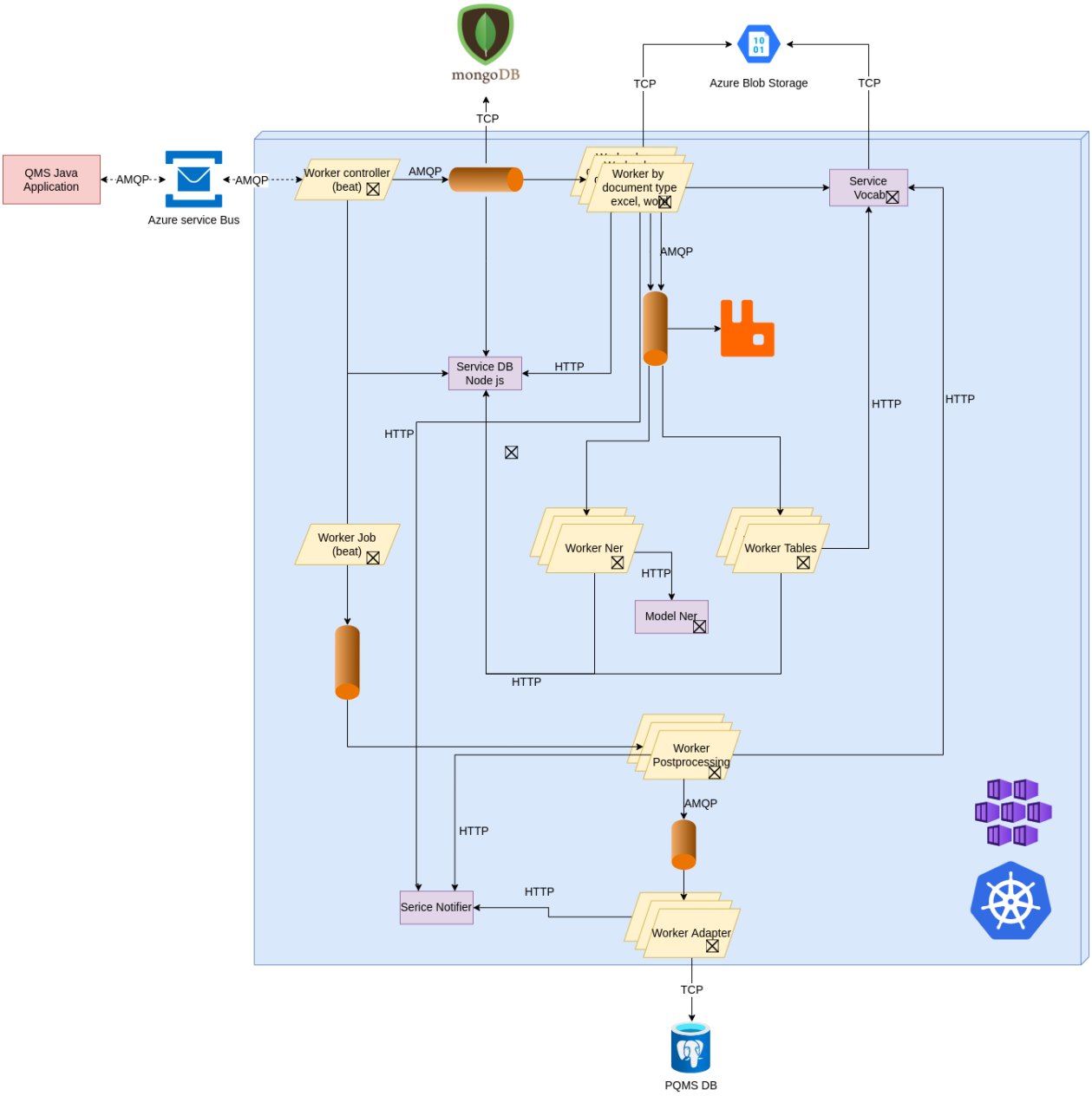

Fig 3. Arquitectura a alto nivel de la plataforma

Desde la aplicación FrontEnd se carga un contrato, la aplicación QMS de Java recibe el contrato y se encarga de subirlo a Azure Blob Storage, una vez cargado se envía un mensaje a través un tópico en Azure Service Bus, dicho mensaje es leído por el "Worker controller" que se ejecuta cada 20 segundos y captura los últimos mensajes, crea un registro inicial en la base de datos de mongo y procede a enviar las tareas al siguiente worker "worker preprocessing (document type)" Excel y Word el cual descarga el archivo del Azure Blob Storage, capturando internamente dos estructuras definidas, (Tablas y Párrafos) a estas entidades se les aplican funciones de similaridad para una clasificación de tipo con base a un vocabulario alojado también en ABS, una vez clasificado se almacenan ambos tipos de entidades en la base de datos de mongo y se procede a enviar todas las tablas al "Worker tables" que segmenta horizontalmente y realiza un preprocesado adicional a estas, mientras que los párrafos son enviados cada uno al "Worker ner" donde se consume un modelo de reconocimiento de entidades nombradas y se agregan nuevos atributos a estas entidades.

Adicional a esto, está el "Worker Job" se ejecuta cada 20 segundos y consulta los documentos que no se enviaron a procesar y sus respectivas tablas/ párrafos. Los documentos que satisfagan las consultas son enviados al "Worker postprocessing" donde se recopilan todas las tablas y párrafos y mediante operaciones matriciales se aplican reglas de negocio se constituye la información del template final, al finalizar este worker envía la información al "Worker adapter" que recopila la información y ajusta a un formato acordado y que soporta las entidades de la base de datos de PQMS Java, este se encarga de realizar las respectivas inserciones en la base de datos de PQMS, finalmente se comunica con el “Service notifier” el cual es encargado de centralizar la comunicación del flujo de la plataforma cognitiva con la aplicación de PQMS Java informando que ya se terminó el proceso de ETL para el contrato cargado, este último servicio también es consumido al presentarse algún tipo de excepción dentro de la plataforma, tales excepciones pueden deberse a que los contratos se encuentren corruptos, no estén disponibles en el bucket, errores de acceso a base de datos y similares.

Perfilamiento de recursos y réplicas

Para garantizar la estabilidad y confiabilidad de un sistema, es necesario asignar adecuadamente los recursos para que las solicitudes necesarias puedan procesarse sin exceder los costos involucrados.

Para el perfilamiento fue utilizado un AKS Cluster con 5 Nodos de tipo Standard B4ms (4 vcpus, 16 GB de memoria) en los servicios en la nube de Microsoft Azure. Fueron montados todos los servicios y Workers que componen el proceso de ETL de un contrato ya sea en formato xlsx (Microsoft Excel) o docx (Microsoft Word). Hemos perfilado los workers encargados del preprocesamiento, tablas y párrafos-ner, así como el posprocesamiento a partir de las siguientes premisas: (i) el efecto en la estabilidad y velocidad de la plataforma es mínimo por parte del worker controller y worker dbjob dada la baja complejidad y la mínima carga de trabajo de estos, (ii) Worker NER trabaja incorpora el Modelo NER.

El procedimiento adoptado fue el siguiente:

- Se tomaron siete documentos de Excel y seis documentos de Word que se consideran grandes para la operación, tanto en términos de tamaño de disco (Mb) como en la cantidad de párrafos y tablas que generan cuando se preprocesan, tal como se presenta en la siguiente tabla.
- Los contenedores se han desplegado variando cada vez uno de sus parámetros de límite de recursos (RAM y CPU) y midiendo el tiempo que se tarda en cada una de estas configuraciones en procesar un documento en cada uno de los contenedores.

En función del tiempo de ejecución y los recursos necesarios, las réplicas de pod se optimizan para tener los tiempos de procesamiento mínimos posibles sin exceder los recursos disponibles dentro del cluster.

TABLA 1
CARACTERIZACIÓN DE DOCUMENTOS USADOS PARA EL PERFILAMIENTO

File	Tamaño	Párrafos	Tablas
CMA_21-2950 amd 70 (1).xlsx	521kb	1135	23
CMA_21-0765 amd 136.xlsx	2.8Mb	2369	74
CMA_21-1754 AMD 25.xlsx	1.6Mb	2050	67
CMA_MF_21-3262 amd 29.xlsx	3.1Mb	2235	23
CMA_SC#21-2691_AMD#30.xlsx	916.6kb	2128	50
CMA_SC#21-2691_AMD#34a.xlsx	916.7kb	2083	50
CMA 21-1484-64.xlsx	3.3Mb	2322	90
T0097678 - CHN21020-17 Carotrans International Inc_.docx	417.2kb	1260	32
T0092788 - OOLU (1).docx	605.8kb	1334	86
T0097661 - atth_2021341707655_1614894129.docx	365.6kb	555	19
T0098103 - atth_2021382206924.docx	515.7kb	1154	60
T0098103 - atth_2021382206424.docx	510.4kb	725	53
T0098020 - atth_2021381509458_docx	660.0kb	775	63

Para la creación de perfiles de preprocesamiento de archivos de Excel, se utilizaron conjuntos de límites de (megabytes de RAM, milicores de CPU) como los siguientes (500, 500), (500, 750), (1000, 500), (1000, 750), (1000, 1000). Como podemos ver en la Fig. 5 tanto el segundo como el tercer conjunto de parámetros nos llevan a tiempos de procesamiento similares; sin embargo, en este caso, es preferible el primero, indicado por la banda vertical, ya que utilizaremos menos recursos permitiéndonos utilizar esta RAM para una posible réplica más. Con esta configuración, tenemos un tiempo medio de preprocesamiento de Excel de unos 80 segundos.

Del mismo modo, para el perfilado del preprocesamiento de Word, se utilizaron conjuntos de límites de (megabytes de RAM, milicores de CPU) como los siguientes (500, 500), (500, 750), (800, 750), (1000, 500), (1000, 750). En este caso, la Fig. 6 muestra que un contenedor con un límite de RAM inferior a 500Mb no es capaz de procesar los documentos. Ahora, son el tercer y el quinto conjunto de parámetros los que nos llevan a un tiempo de procesamiento similar; así, es

preferible el que tiene menos RAM, indicado por la banda vertical, porque permite un pod más. Con esta configuración, tenemos un tiempo medio de preprocesamiento de palabras de unos 60 segundos.

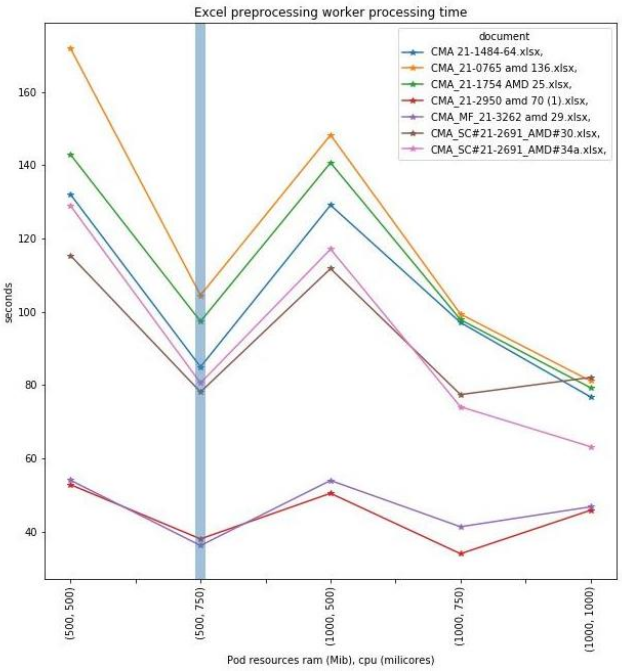

Fig 4. Estimación de recursos worker preprocessig Excel

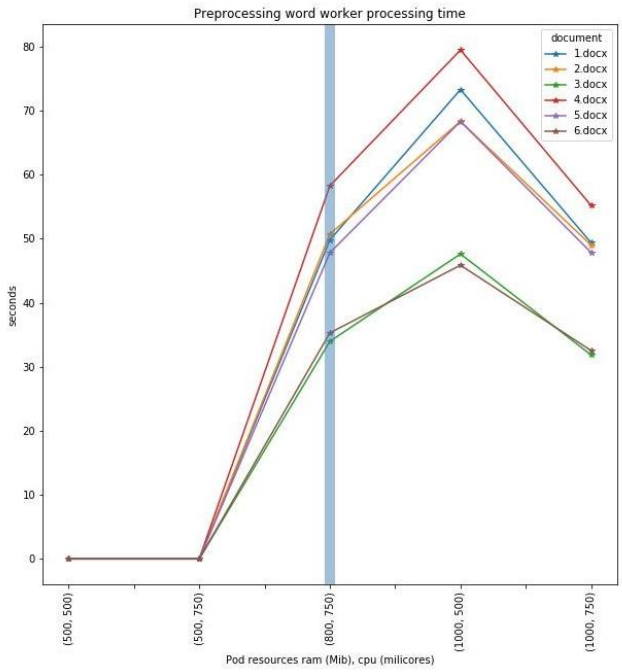

Fig 5. Estimación de recursos worker preprocessig Word

Por último, para el worker de posprocesamiento, utilizamos (megabytes de RAM, milicores de CPU) parámetros de (500, 1000), (900, 1000), (900, 750), (900, 1000). Para este contenedor, los recursos de la CPU son de suma importancia porque tiene un alto costo computacional. Como vemos en la configuración con 750 milicores se genera un cuello de botella mientras que al aumentar a 1250 milicores no se genera una diferencia apreciable respecto al caso de sólo 1000 milicores. Por lo tanto, se toma la configuración (900, 1000) para este contenedor, generando un tiempo medio de postprocesamiento de unos 206s por contrato.

Definición de réplicas y escalabilidad

Con los resultados anteriores y teniendo en cuenta el requisito no funcional de poder procesar al menos 50 contratos por hora, podemos estimar el número de pods que hay que crear

en otros servicios y trabajadores para cada pod de preprocesamiento. Para cada Excel, pod de preprocesamiento y con una media de 2000 párrafos y 80 tablas por contrato, tenemos lo que se presenta en la tabla 2.

TABLA 2

MULTIPLICADOR DE CONTENEDORES POR ARCHIVO SIMULTANEO

Worker	**Réplicas por Archivo word**	**RAM (Mb)**	**CPU (milicores)**
Worker Preprocessing Excel	1	500	750
Worker NER	3	150	200
Model NER	1,5	150	100
Tables	3,5	200	200
Postprocess	1	500	1000
Adapter	1	1000	1000
Service DB	3	200	200
DB job	1, sin importar el número de workers de preprocesamiento.	100	50
Controller	1, sin importar el número de workers de preprocesamiento.	100	50
Service vocab	1, sin importar el número de workers de preprocesamiento.	100	100

Como se mencionó al principio, uno de los beneficios y razones para implementar esta arquitectura es lo apropiada para servicios que deben ser altamente escalables, con los recursos correctamente estimados y el factor de réplicas definidos se hizo uso de Keda [14] que es un escalador automático basado en eventos para Kubernetes, el cual nos brinda la posibilidad de escalar y des-escalar horizontalmente un recurso de Kubernetes como lo son los pod donde se alojan los workers, todo esto se hace con base a distintas métricas que se pueden configurar como lo son la cantidad de mensajes en una cola, para el propósito de este proyecto fue altamente pertinente y permitió implementar lo definido en la tabla anterior.

A continuación, en la Fig. 7, se presenta un ejemplo de manifiesto de configuración de Keda donde se define el mínimo y máximo de réplicas, las políticas para des-escalar y en específico que se debe escalar de a una réplica por cada 10 mensajes en la cola de tareas.

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: performance-rabbit
  namespace: MIRIAM
spec:
  scaleTargetRef:
    name: worker-ner-deployment
  pollingInterval: 10
  cooldownPeriod: 100
  minReplicaCount: 1
  maxReplicaCount: 10
  advanced:
    restoreToOriginalReplicaCount: true
    horizontalPodAutoscalerConfig:
      behavior:
        scaleDown:
          stabilizationWindowSeconds: 60
          policies:
            - type: Percent
              value: 20
              periodSeconds: 60
  triggers:
    - type: rabbitmq
      metadata:
        host: http://<user-rabbit>:<password-rabbit>@rabbitmq-hos.default.svc.cluster.local:15672
        queueName: worker-ner-prod
        mode: QueueLength
        value: "10"
```

Fig 6. Manifiesto de hpa con Keda

VI. CONCLUSIONES

- Una de las principales ventajas de Kubernetes es su capacidad para permitir que las aplicaciones sean altamente escalables, sin embargo, es necesario planear una estrategia que permite un balanceo adecuado de las cargas de trabajo y así evitar que sean desaprovechados los recursos que se aprovisionan; debido a la configuración del cluster este habilita una nueva máquina o nodo temporal en los momentos en que se presenta mayor uso de recursos físicos como RAM o CPU. El uso de Keda en este proyecto permitió escalar horizontalmente los Workers en base a la longitud en las colas de tareas, que debido a la naturaleza del problema entre más larga sea, mayor es la carga de trabajo; lo cual hace coincidir el escalamiento de la infraestructura como el de los Workers, logrando que estos últimos se vayan ubicando y repartiendo equitativamente a través de la infraestructura nueva y con menos carga de trabajo. Finalmente, cuando la cola de tareas se va vaciando y es necesaria una menor cantidad de réplica de Workers, estos se des-escalan y liberan las máquinas o nodos nuevos permitiendo ahorrar costos.

- Se logró una mejora sustancial en los tiempos de respuesta del sistema, al procesar simultáneamente diferentes documentos, esto fue posible gracias a que se diseñó e implementó apropiadamente una arquitectura orientada a eventos utilizando un sistema de colas de tareas que permite orquestar todos los procesos de trabajo del sistema, de tal manera que las tareas se ejecutan en el instante que el Worker esté disponible y no es un proceso bloqueante como podría ser una API monolítica.

- Debido a la nueva segmentación en Workers de todas las operaciones del sistema se percibió una gran mejora en diversos atributos de calidad de software como lo son la mantenibilidad, testabilidad y principalmente escalabilidad y rendimiento. Permitiendo un menor tiempo de desarrollo de nuevas funcionalidades o corrección de bugs, observar de manera precisa todo el flujo de la información correspondiente al proceso de ETL y ser resiliente a las distintas cargas de trabajo que se puedan presentar.

- La implementación de este tipo de arquitectura suele ser significativamente más compleja que una arquitectura tradicional debido a la forma de comunicación entre los distintos procesos, la cola de tareas, base de datos y costos que pueden tanto computacionales como económicos para poder soportar la infraestructura requerida. Adicional a desafíos técnicos que conlleva este tipo de arquitecturas con procesos asíncronos y/o simultáneos como condiciones de carrera que se genera en el instante que dos procesos intentan usar y modificar el mismo recurso al mismo tiempo o sincronización de la información para avanzar a los siguientes pasos.

REFERENCIAS

[1] raunakjhawar, “Extracción, transformación y carga de datos (ETL) ,” *Azure Architecture Center*. https://docs.microsoft.com/es-es/azure/architecture/data-guide/relational-data/etl (accessed Jun. 02, 2022).

[2] R. A. Manjarrés-Betancur and M. M. Echeverri-Torres, “Asistente virtual académico utilizando tecnologías cognitivas de procesamiento de lenguaje natural,” *Rev. politec.*, vol. 16, no. 31, pp. 85–95, May 2020, doi: 10.33571/rpolitec.v16n31a7.

[3] J. Bosch, “Software Architecture: The Next Step,” in *Software Architecture*, vol. 3047, F. Oquendo, B. C. Warboys, and R. Morrison, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 194–199. doi: 10.1007/978-3-540-24769-2_14.

[4] A. A. Abbasi, A. Abbasi, S. Shamshirband, A. T. Chronopoulos, V. Persico, and A. Pescape, “Software-Defined Cloud Computing: A Systematic Review on Latest Trends and Developments,” *IEEE Access*, vol. 7, pp. 93294–93314, 2019, doi: 10.1109/ACCESS.2019.2927822.

[5] “What is Kubernetes?,” *Kubernetes*. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (accessed Jun. 02, 2022).

[6] V. Hema, S. Thota, S. Naresh Kumar, C. Padmaja, C. B. Rama Krishna, and K. Mahender, “Scrum: An Effective Software Development Agile Tool,” *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 981, no. 2, p. 022060, Dec. 2020, doi: 10.1088/1757-899X/981/2/022060.

[7] M. Lunacek, J. Braden, and T. Hauser, “The scaling of many-task computing approaches in python on cluster supercomputers,” in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2013, pp. 1–8. doi: 10.1109/CLUSTER.2013.6702678.

[8] “¿Qué es la arquitectura basada en eventos?” https://www.redhat.com/es/topics/integration/what-is-event-driven-architecture (accessed Jun. 02, 2022).

[9] EdPrice-MSFT, “Event-driven architecture style,” *Azure Architecture Center*. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven (accessed Jun. 02, 2022).

[10] “Event-Driven Architecture,” *Amazon Web Services, Inc.* https://aws.amazon.com/event-driven-architecture/ (accessed Jun. 02, 2022).

[11] EdPrice-MSFT, “Patrón de consumidores de la competencia - Azure Architecture Center,” *Azure Architecture Center*. https://docs.microsoft.com/es-es/azure/architecture/patterns/competing-consumers (accessed Jun. 02, 2022).

[12] “What are containers?,” *Google Cloud*. https://cloud.google.com/learn/what-are-containers (accessed Jun. 02, 2022).

[13] V. John and X. Liu, “A Survey of Distributed Message Broker Queues,” 2017, doi: 10.48550/ARXIV.1704.00411.

[14] P. G. López, A. Arjona, J. Sampé, A. Slominski, and L. Villard, “Triggerflow: trigger-based orchestration of serverless workflows,” in *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, Montreal Quebec Canada, Jul. 2020, pp. 3–14. doi: 10.1145/3401025.3401731.