

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

**Desarrollo de una aplicación
distribuida para la extracción,
almacenamiento y
procesamiento del historial de
artículos wiki basados en
MediaWiki**

TRABAJO ESPECIAL DE GRADO PARA OPTAR AL
TÍTULO DE LICENCIADO EN COMPUTACIÓN

Marvin E. Bernal P. C.I: 18.154.154
Francisco J. Delgado M. C.I: 19.608.720
Tutor: Prof. Eugenio Scalise

Mayo, 2018

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

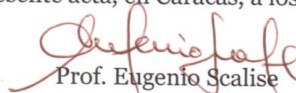
ACTA


Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los bachilleres **Marvin Enrique Bernal Perdomo** (C.I. V-18154154) y **Francisco Javier Delgado Muñoz** (C.I. V-19608720), titulado: **“Desarrollo de una aplicación distribuida para la extracción, almacenamiento y procesamiento del historial de artículos wiki basados en MediaWiki”**, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación dejan, constancia de lo siguiente:


Leído el trabajo por cada uno de los miembros del jurado, se fijó el día 17 de mayo a las 10 a.m., para que sus autores lo defendieran en forma pública en la sala PB-III de la Escuela de Computación, lo que éstos realizaron mediante una exposición oral de su contenido y luego respondieron satisfactoriamente a las preguntas que les fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela.

Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió **aprobarlo**.

En fe de lo cual se levanta la presente acta, en Caracas, a los 17 días del mes de mayo de 2018.


Prof. Eugenio Scalise
(Tutor)


Prof. María Paula Herrero
(Jurado)


Prof. Rhadamés Carmona
(Jurado)

ÍNDICE

1. Problema a Resolver y Objetivos	9
1.1. Planteamiento Del Problema	9
1.2. Justificación	12
1.3. Objetivos	13
1.3.1. Objetivo General	13
1.3.2. Objetivo Específicos	13
2. Tecnologías Utilizadas	14
2.1. MongoDB	14
2.1.1. Replicación	15
2.1.2. Particiones	17
2.1.3. PyMongo	20
2.2. Celery	21
2.2.1. Flower	22
2.3. RabbitMQ	23
2.4. Flask	24
2.5. Nginx	24
2.6. Docker	25
3. Marco Aplicativo	29
3.1. Método de Desarrollo	29
3.2. Servidor Web	31
3.3. API	31
3.3.1. Extracción de Historiales	33
3.3.2. Consultas	36
3.4. Algoritmo de Revisita	45
3.5. Almacenamiento	50
3.6. Docker	54
4. Conclusiones	58

ÍNDICE DE FIGURAS

1.	Componentes de un grupo de replicas en MongoDB y su interacción con una aplicación cliente	16
2.	Heartbeats en un grupo de replicación de MongoDB	16
3.	Proceso de elección de nuevo nodo primario	17
4.	Componentes de un cluster de particiones	18
5.	Interacción entre una aplicación cliente y una base de datos MongoDB particionada	19
6.	Range Sharding de datos en MongoDB	19
7.	Hashed Sharding de datos en MongoDB	20
8.	Zone Sharding de datos en MongoDB	20
9.	Flujo de trabajo en la asignación de tareas de Celery	22
10.	Interfaz web de Flower	23
11.	Flujo de trabajo de la emisión y suscripción de mensajes en RabbitMQ	24
12.	Atención de peticiones con el uso de proxy reverso de Nginx	25
13.	Diagrama de componentes de una Maquina Virtual	27
14.	Diagrama de componentes de un Contenedor	28
15.	Arquitectura general de la aplicación	30
16.	Algoritmo de distribución de peticiones de Nginx basado en método de Round Robin.	31
17.	Interacción entre una aplicación cliente y el API	32
18.	Parámetros para la extracción de historiales	33
19.	Respuesta a una petición de extracción de historiales de un artículo	34
20.	Respuesta a una petición para consultar de progreso de una tarea de extracción	35
21.	Respuesta a una petición de consulta de artículos	36
22.	Respuesta a una petición de consulta de revisiones	37
23.	Estructura del cuerpo de la petición a la ruta /api/v1/query	44
24.	Estructura del cuerpo de la petición a la ruta /api/v1/query	44
25.	División de la base de datos en subgrupos	51
26.	Shards de la base de datos distribuidos físicamente	52
27.	Arquitectura del cluster de base de datos	53

ÍNDICE DE TABLAS

1.	Extracción de Revisiones	35
2.	Contabilización de Revisiones	39
3.	Promedio de Revisiones	41
4.	Moda de Revisiones	43
5.	Consultas Directas a la Base de Datos	45
6.	Algoritmo de revisita de revisiones de artículos wiki	50
7.	Almacenamiento de revisiones de artículos wiki	51
8.	Configuración del cluster de MongoDB	54
9.	Configuración del levantamiento de la aplicación por medio de Docker	55

ÍNDICE DE CÓDIGO FUENTE

1.	Edición del archivo de configuración de Cron	46
2.	Edición del archivo de configuración de Cron	46
3.	Contenido del archivo Dockerfile para la imagen del API . . .	56
4.	Declaración de servicios con Docker Compose	57

RESUMEN

Con el tiempo, el uso de artículos wiki basados en MediaWiki, como Wikipedia, para la búsqueda de información, se ha incrementado de manera significativa. Existen aplicaciones y herramientas que hacen uso de estos artículos para analizar su contenido con diversos propósitos. Tal es el caso de Wiki-Metrics-UCV, cuya labor es la extracción, almacenamiento y análisis de las revisiones de dichos artículos. El almacenamiento de las revisiones de artículos wiki, puede llegar a ser una tarea compleja debido a la magnitud de los datos que se maneja y las limitaciones de hardware y espacio de almacenamiento que una máquina pueda tener. Por esta razón, se decidió crear una nueva aplicación que incorpora nuevas tecnologías que permiten escalar de manera horizontal, la capacidad de almacenamiento de datos a través de múltiples máquinas trabajando en conjunto, y de esta manera, superar las limitaciones de hardware que una sola máquina pueda presentar. Este trabajo de investigación se centra en la implementación de una aplicación distribuida, la cual permite la extracción de las revisiones de artículos wiki, su almacenamiento y la construcción de diversas consultas que permiten obtener métricas asociadas al historial de modificación de un artículo. Esta aplicación distribuida se desarrolló por medio de la aplicación de la metodología ágil llamada Desarrollo Rápido de Aplicaciones (RAD), la cual se basa en el desarrollo continuo de múltiples componentes pequeños que en conjunto conforman todo el sistema.

Palabras clave: Aplicación distribuida, Base de Datos, Nodos, API, Servicio Web, Algoritmo, Cluster, Consultas.

INTRODUCCIÓN

Hoy en día Wikipedia, la enciclopedia libre, se ha posicionado como una de las herramientas de referencia más grandes e indispensables para todo aquel en búsqueda de información.

Wikipedia cuenta con 30 millones de usuarios registrados, de los cuales mas de 130 mil son usuarios activos que contribuyen en hasta 270 idiomas generando nuevo contenido, perfeccionándolo, removiendo errores e incluso vandalismos en los artículos [4].

Cada uno de estos cambios son almacenados en un historial en conjunto con una serie de datos adicionales tales como el autor de la actualización, la fecha, notas, entre otras. Existen aplicaciones, tales como Wiki-Metrics-UCV, que tienen como objetivo la recolección, almacenamiento y procesamiento de historiales para el cálculo y visualización de métricas de sus atributos [10].

Para la fecha, Wikipedia alberga alrededor de 5.5 millones de artículos, y que cada uno de ellos existe un promedio de 20 mil actualizaciones [6]. Las aplicaciones de terceros, como Wiki-Metrics-UCV, que están enfocadas en la extracción de estos y son desarrolladas en un esquema centralizado, no tienen la capacidad de escalar sus servicios una vez alcancen los limites de sus unidades de almacenamiento.

Por suerte, existe el modelo de cómputo distribuido en el cual la carga de datos y las tareas llevadas a cabo en un sistema de software son compartidas por múltiples componentes para aumentar la eficiencia y el rendimiento.

La aplicación de los sistemas distribuidos se adapta a diversos campos, tales como: redes de telecomunicaciones, computación paralela y procesos de monitoreo en tiempo real, entre otros. El uso de este tipo de sistemas puede llegar a ser muy beneficioso por razones de rendimiento o de costo, por ejemplo, puede ser menos costoso el uso de un grupo de varios computadores con componentes de gama baja, que un solo computador con componentes gama alta. Además, un sistema distribuido puede llegar proporcionar más fiabilidad que un sistema centralizado, ya que pueden implementarse medidas de recuperación de fallos en caso de que uno de los miembros del sistema presente un error.

Con este tipo modelo es posible diseñar un sistema en donde todos sus componentes se dividan de forma equitativa las tareas necesarias para

resolver un problema común, tal como el almacenamiento y procesamiento de miles de historiales de los artículos de Wikipedia. De esta manera cada componente podría almacenar los datos recolectados en su propia unidad de almacenamiento y así garantizar que el sistema sea escalable.

Este trabajo documenta el desarrollo de un sistema distribuido, basado en Wiki-Metrics-UCV, para la extracción, almacenamiento y procesamiento del historial de artículos wikis basados en MediaWiki (por ejemplo, Wikipedia).

En el Capítulo 1 se presenta en detalle la descripción del problema, la justificación del por qué desarrollar una alternativa distribuida al mismo, y por último, los objetivos tanto generales como específicos que se llevaron a cabo.

En el Capítulo 2 se describen tecnologías utilizadas en el desarrollo del sistema, incluyendo servicios, las librerías usadas y las bases de datos elegidas.

Por último, en el Capítulo 3 se presenta el marco aplicativo del sistema desarrollado para la solución al problema planteado, en donde se describen sus componentes y el método de desarrollo.

1. PROBLEMA A RESOLVER Y OBJETIVOS

1.1. PLANTEAMIENTO DEL PROBLEMA

Un *wiki* es una página web que permite la colaboración de múltiples usuarios para la creación y modificación de contenido que hacen referencia a un tema en particular y que pueden estar relacionados entre ellos a través de hipervínculos.

Un wiki es ejecutado por el medio del uso software wiki que permite a los usuarios la edición de estas páginas. El contenido de las páginas, también llamados artículos, es almacenado en conjunto con su historial de cambios en una base de datos. De esta forma se facilita recuperar el estado anterior de cualquier artículo y visualizar diversos datos de una actualización tal como el nombre del autor o la fecha de edición.

Hoy en día el proyecto enciclopedia en línea Wikipedia es una de las páginas web basadas en wiki más populares, albergando miles de colecciones de artículos wiki en múltiples idiomas [3].

La aplicación Wiki-Metrics-UCV tiene como objetivo la extracción de estos historiales de los artículos wiki para su almacenamiento y posteriormente usar sus datos para el cálculo de diversas métricas. Wiki-Metrics-UCV hace uso de técnicas de extracción web (web scraping) sobre los artículos wiki y, a través de un análisis sintáctico, almacena los datos obtenidos en una base de datos no relacional [10].

El proceso de extracción de historiales de artículos wiki puede llegar a tomar una cantidad considerable de tiempo, no sólo por la cantidad de entradas, sino por el número de documentos que un repositorio wiki alberga.

En aplicaciones como Wiki-Metrics-UCV, que corresponden a una solución de cómputo y almacenamiento centralizado, existen diversas limitaciones que se presentan en la solución. Entre ellas es que el proceso se realiza de manera secuencial, lo cual obliga a la aplicación a terminar de procesar un artículo antes de pasar al siguiente, y como consecuencia, el tiempo empleado para procesar grandes volúmenes de historiales no se encuentra dentro de un intervalo aceptable para la cantidad de revisiones previstas a procesar.

Adicionalmente, los repositorios wiki tales como Wikipedia limitan la cantidad de peticiones HTTP que se realizan sobre sus servicios. Por lo tanto, existe la posibilidad de que un sistema distribuido lleve a cabo una cantidad de peticiones concurrentes que puedan ser vistas como un ataque de denegación de servicio (DDoS).

Dado que en la solución implementada por Wiki-Metrics-UCV se establecen tiempos definidos de espera entre petición pensados únicamente para el caso centralizado, es necesario considerar y adaptar ciertos aspectos para su adaptación en un escalamiento horizontal, tales como: la cantidad de nodos que ejecutan peticiones HTTP sobre un URL; la concurrencia y el tiempo de espera entre cada petición; y por último los permisos presentes en los términos y condiciones de uso de cada repositorio de artículos wiki en el aspecto de la extracción web.

En términos legales, y en las cláusulas de los términos y condiciones de uso, cada repositorio wiki indican como el uso de sus servicios y la extracción de sus datos públicos puede ser considerado inapropiado. En el caso de Wikipedia, por medio de sus términos de uso se indica que un individuo debe evadir cualquier actividad, ya sea legal o ilegal, que pueda comprometer su infraestructura tecnológica o violar derechos de autor [11].

Dado que Wiki-Metrics-UCV no recolecta la información contenida en un wiki, ni existe una cláusula específica sobre la recolección de datos de los historiales de edición, las actividades ejecutadas por esta aplicación no violentan los términos de uso de Wikipedia mientras que las peticiones HTTP realizadas no comprometan su sistema. Actualmente Wikimedia, quien es el creador del proyecto Wikipedia, ofrece una herramienta para la recolección y consulta de datos de un wiki, incluyendo su historial, llamado MediaWiki. MediaWiki provee de una interfaz de programación de aplicaciones (Application Program Interface - API) que puede ser usada por la aplicación de un tercero para consultar estos datos. Este API se presta como alternativa a las técnicas de web scraping usadas por Wiki-Metrics-UCV y tiene ciertas limitaciones con respecto al límite de peticiones HTTP ejecutadas sobre este servicio, similares a las relacionadas con la extracción web. A pesar de que MediaWiki no define límites específicos sobre estas peticiones, sí indica que la cantidad y frecuencia de las mismas sean moderadas, especialmente si las peticiones son ejecutadas de forma paralela. En caso de que estas consultas comprometan el sistema, MediaWiki especifica que sus administradores de sistema (sysadmin) están en la potestad de bloquear la dirección IP de quien realice estas peticiones [13].

La adaptación de Wiki-Metrics-UCV a un sistema distribuido disminuye varias de las limitaciones previamente mencionadas. Con la presencia de varios nodos en el sistema es posible procesar diversos artículos de forma paralela, evitando cuellos de botella, y además, dividir la carga de almacenamiento entre los componentes del sistema.

Este escalamiento horizontal, tal como lo explica el teorema CAP[5], trae consigo problemas adicionales e intrínsecos de todo sistema distribuido. Es imposible para un servicio garantizar las tres siguientes características: consistencia, disponibilidad y tolerancia a fallos. A pesar de que estas propiedades son esenciales, es necesario identificar qué características deben ser acopladas a esta nueva solución y cuál debe ser sacrificada.

Por otra parte, teniendo en cuenta que con un escalamiento horizontal existen N nodos independientes, es innecesario mantener una comunicación entre ellos para coordinar sus acciones y cumplir con la meta establecida. Así mismo, es indispensable definir las políticas y algoritmos para la elección de coordinadores del sistema distribuido, el cual tiene como objetivo la asignación de los artículos wiki que procesa cada uno de los nodos para evitar consultas redundantes sobre un mismo URL, y la planificación de las actualizaciones en la base de datos. Adicionalmente, es importante considerar el manejo de recursos críticos dentro de la comunicación, para lograr una adecuada sincronización entre los nodos del sistema, y así evitar interbloqueos en la comunicación entre ellos. Otro aspecto a tomar en cuenta, es que a medida que la cantidad de nodos crezca (en caso del tamaño del escalamiento), la comunicación existente en el sistema también se incrementará; por ello, es necesario considerar el sistema de comunicación a emplear, y estimar su fiabilidad en casos adecuados a lo que se determinará en relación al número de nodos para la solución.

Por último, se debe tomar en cuenta la arquitectura y la forma en la cual los datos serán distribuidos entre los componentes del sistema. Determinar bajo qué estrategia los datos serán fragmentados y si será posible ofrecer tolerancia a fallos, consistencia y/o alta disponibilidad. Esto se debe a que la inclusión de miembros adicionales en un sistema trae como consecuencia un incremento en la complejidad de la infraestructura y su mantenimiento.

1.2. JUSTIFICACIÓN

En el caso de los artículos wiki de la Fundación Wikimedia, existe un campo poco explorado con respecto a las estadísticas de los artículos, las cuales permitirían resaltar características difíciles de visualizar a primera vista. De estos artículos es posible obtener información muy relevante a través del estudio de su historial de revisiones, por ejemplo: el colaborador que más ha editado un artículo en particular, las fechas y horas cuando más se ha editado un artículo o a qué país o continente pertenecen las personas que más han colaborado en los artículos. Incluso, es posible determinar ciertos patrones de interés de los usuarios sobre un tema, lo cual puede ser usado por diversos analistas para elaborar alguna estrategia de mercadeo y publicidad. La cantidad de revisiones realizadas sobre un artículo wiki reflejan el uso y la manipulación de datos sobre un recurso lo cual puede ser usado para determinar tendencias sobre el tema del artículo.

Debido a las limitaciones de un sistema centralizado, surge la necesidad implementar una nueva solución que permita ampliar las capacidades de almacenamiento y disminuir los cuellos de botella que se puedan producir durante la extracción de los historiales. Por medio de un escalamiento horizontal es posible hacer uso de un grupo de computadores que comparten la carga de todas las tareas a ejecutar e incrementar la cantidad de datos que se pueden almacenar.

El escalamiento horizontal involucra dividir los datos en grupos y almacenarlos en los múltiples nodos que se requieran agregar al sistema para incrementar su capacidad. Aunque la capacidad de uno de estos nodos sea limitada, cada uno de ellos solo maneja un subconjunto de la carga total, permitiendo de forma potencial que se obtenga una mayor eficiencia que un solo servidor de alta capacidad y potencia. Expandir la capacidad del sistema solo requiere agregar tantos nodos sea necesario, que por lo general suele ser de menor costo que adquirir componentes de alta gama en una sola máquina.

Contar con múltiples nodos permite aumentar la disponibilidad del servicio, característica definida como alta disponibilidad. Con al menos dos miembros es posible recuperar de errores tales como la detención abrupta de un servidor, en este caso, el miembro restante puede manejar la carga de trabajo mientras el primero reinicia sus operaciones. Incluso, se pueden obtener beneficios dependiendo de la ubicación geográfica de estos miembros. El consumidor de un servicio puede obtener una respuesta más rápida a sus peticiones con la implementación de servidores cache cercanos a su ubicación

geográfica.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Desarrollar una aplicación distribuida para la extracción, almacenamiento y procesamiento del historial de los artículos wiki basados en MediaWiki a través de técnicas de web scraping y el uso de MediaWiki API.

1.3.2. OBJETIVO ESPECÍFICOS

- Desarrollar de un módulo para la recolección y de consulta de los datos sobre el servicio API de MediaWiki.
- Diseñar un modelo de datos para el almacenamiento de los historiales de los artículos wiki.
- Implementar y configurar la topología de sharding de MongoDB para el sistema distribuido.
- Configurar los nodos de MongoDB dentro del sistema distribuido para la implementación de réplicas.
- Adaptar los algoritmos de revisita sobre los artículos wiki basados en la solución brindada por Wiki-Metrics-UCV.
- Diseñar y configurar los métodos de asignación de tareas para el balance de carga entre los nodos del sistema.
- Desarrollar los algoritmos de procesamiento de los datos almacenados para la visualización de estadísticas de los artículos wiki.
- Implementar un conjunto de pruebas sobre los módulos de extracción de datos, almacenamiento, replicación y procesamiento de los mismos.
- Desarrollar un API que permita a aplicaciones de terceros consultar los historiales de los artículos wiki.

2. TECNOLOGÍAS UTILIZADAS

En este capítulo se describen las tecnologías, librerías y servicios utilizados para el desarrollo del sistema.

La base de datos seleccionada para almacenar las revisiones de los artículos wiki, llamada MongoDB, y la definición de las características que facilitan el escalamiento horizontal de los datos, tales como: la réplicación, que permite brindar un servicio con alta disponibilidad; y las particiones, que se refieren a la capacidad que tiene MongoDB para dividir los datos almacenados en subgrupos entre los múltiples miembros del sistema.

Celery y RabbitMQ, que permiten la creación de una cola de tareas para que puedan ser ejecutadas de forma asíncrona, disminuyendo los cuellos de botella y así optimizar de cierta manera los tiempos de respuesta y la eficiencia general del sistema.

Flask, el microframework de Python, el cual permite el desarrollo de aplicaciones web sin la inclusión componentes innecesarios, y en este caso, permite la creación de un servicio API que aplicaciones de terceros pueden consumir para la extracción y consulta de las revisiones de los artículos wiki.

Y por último, Docker, que sirve como una herramienta para emular un ambiente distribuido en donde se van a ejecutar los múltiples nodos tanto de la base de datos, como los servicios de Celery y RabbitMQ, y las múltiples instancias del servicio API.

2.1. MONGODB

MongoDB es una base de datos NoSQL (no solo SQL) orientada a documentos. Almacena datos en una representación binaria llamada BSON (Binary JSON), la cual extiende las capacidades de un objeto JSON (JavaScript Object Notation) para representar tipos de datos como enteros, punto flotante, fechas, datos binarios, arreglos y sub-documentos o documentos embebidos. Cada documento pertenece a un grupo llamado colección, los cuales son el equivalente a las tablas de una base de datos relacional.

MongoDB posee un conjunto de características que permiten la

escalabilidad de una aplicación, entre ellas, provee la habilidad e implementar la distribución geográfica de datos (sharding). Esto permite a la base de datos ser escalada de forma horizontal con el uso de un conjunto de componentes de hardware o mediante de la nube (cloud).

Adicionalmente, MongoDB [12] está diseñado para ser ejecutado en un sistema de múltiples nodos, por lo tanto, en la presencia de un escalamiento horizontal, incluye la capacidad para la replicación y sincronización de datos entre todos los componentes del sistema. De esta manera se garantiza la posibilidad de implementar un servicio con alta disponibilidad.

2.1.1. REPLICACIÓN

La replicación permite la redundancia de datos y el aumento de disponibilidad de los mismos en aplicaciones distribuidas. Hace uso de múltiples servidores conectados entre sí y genera una réplica de los datos en cada uno de ellos, de esta manera se obtiene cierto nivel de tolerancia a fallos en caso de que un servidor de base de datos falle. También es posible mantener copias adicionales para propósitos específicos como recuperación en casos de desastre, reportes o respaldos [16].

En MongoDB se hace uso de grupos de réplica (replica sets) para almacenar las copias de datos en múltiples servidores de base de datos. Un grupo de réplica previene los tiempos de inactividad de la base de datos en caso de errores y ayuda a escalar las operaciones de lectura. La recuperación de un miembro del grupo de réplica se realiza de forma automática. En un grupo de réplica cualquier miembro puede actuar como nodo principal y el resto como secundario, por lo tanto, en caso de fallas de red o de hardware, un miembro del grupo puede tomar el puesto de nodo primario después de haber sido elegido siguiendo un conjunto de reglas predefinidas.

Un grupo de réplicas esta conformado por un nodo principal que realizará todas las operaciones de escritura, múltiples nodos secundarios y un nodo árbitro opcional, tal cual como lo muestra figura 1:

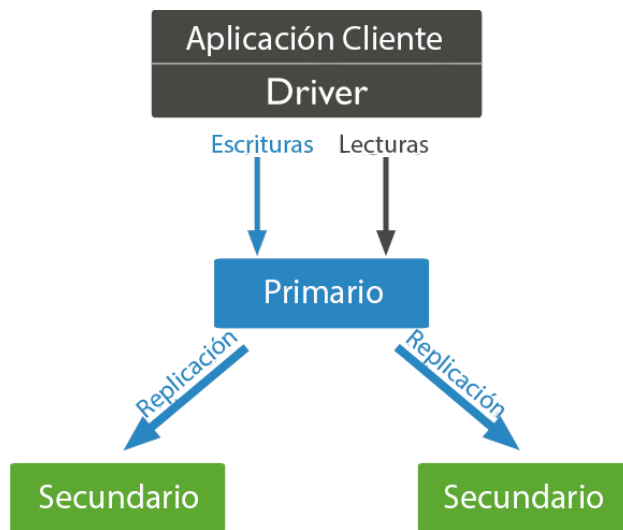


Figura 1: Componentes de un grupo de réplicas en MongoDB y su interacción con una aplicación cliente

Para poder crear estos grupos es necesario tener un mínimo de tres nodos y escalar en números impares. Esto se debe a que cuando los componentes del grupo realicen la votación para elegir el nodo principal, se elimina la posibilidad de un empate y el ganador es elegido de inmediato.

En la figura 2 se puede observar como los nodos del grupo se envían mensajes de diagnóstico (heartbeats) para determinar si sus vecinos son accesibles o no. En caso de que uno de estos heartbeat no obtenga una respuesta en 10 segundos, se marca dicho nodo como inaccesible.

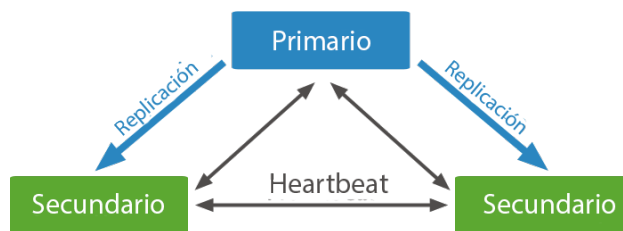


Figura 2: Heartbeats en un grupo de replicación de MongoDB

Los nodos secundarios se encargan de leer los datos presentes en el nodo primario y una copia del mismo. Si, por alguna razón, el nodo primario deja de estar disponible, un nodo secundario elegible ejecutará una votación para elegir el siguiente nodo primario, tal como lo muestra la figura 3. El primer

nodo secundario en llevar a cabo la elección y recibir la mayoría de los votos se convierte en el nuevo nodo primario.

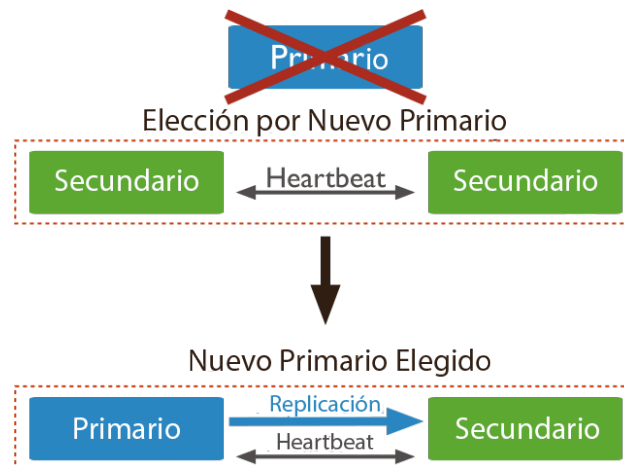


Figura 3: *Proceso de elección de nuevo nodo primario*

2.1.2. PARTICIONES

El proceso de partición (sharding) es un método de escalamiento horizontal que distribuye datos entre múltiples máquinas. Cada conjunto de datos alojado en una máquina es llamado partición (shard), y su acceso es completamente transparente para las aplicaciones. Este método permite atacar las limitaciones de hardware provenientes del uso de un solo servidor, tales como los cuellos de botella o capacidad de almacenamiento [17].

En MongoDB, un grupo de partición (sharded cluster) consiste de tres componentes:

- **Partición:** Cada partición contiene un subconjunto de los datos y pueden ser implementados como un grupo de réplica.
- **mongos:** mongos es un servicio que actúa como interfaz entre las aplicaciones cliente y el grupo de partición, también es llamado query router.
- **Servidores de Configuración:** Los servidores de configuración almacenan meta datos y los atributos de configuración del cluster, y al igual que las particiones, también pueden ser implementados como un grupo de réplica.

En la figura 4 se puede apreciar los componentes del cluster de particiones y la interacción entre los mismos:

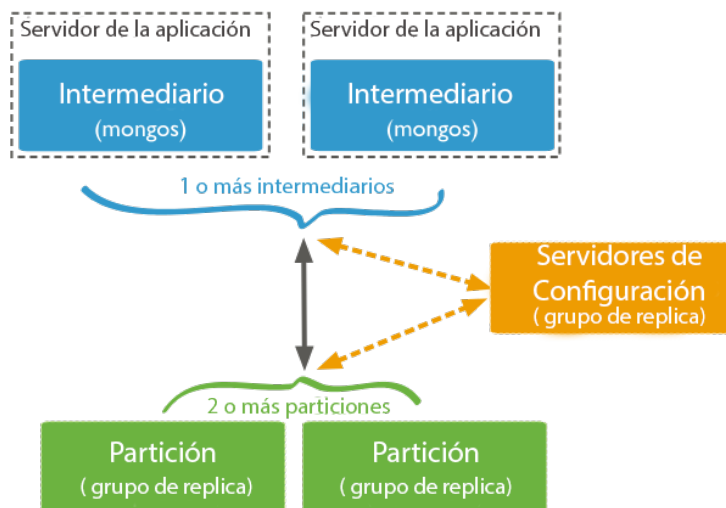


Figura 4: Componentes de un cluster de particiones

Para distribuir los documentos de una colección es necesario el uso de una llave de partición (shard key). Un shard key es uno o múltiples campos inmutables que comparten todos los documentos de una colección. Una colección particionada solo puede tener un shard key y la misma no puede ser cambiada después de haberse ejecutado el proceso de sharding.

MongoDB genera particiones de datos a nivel de colección, distribuyendo los documentos de una colección entre los miembros del cluster de particiones. Una base de datos puede tener una combinación de colecciones particionadas y no particionadas. Las colecciones particionadas son distribuidas entre cada partición del cluster, mientras que las colecciones no particionadas son almacenadas en una partición especial llamada primaria.

El acceso a una base de datos particionada en MongoDB es completamente transparente para cualquier aplicación de cliente, tan solo es necesario saber la dirección IP o nombre del equipo (hostname) y el puerto en el cual se ejecutará el query router. La figura 5 detalla la interacción entre una aplicación de cliente con una base de datos MongoDB con particiones:

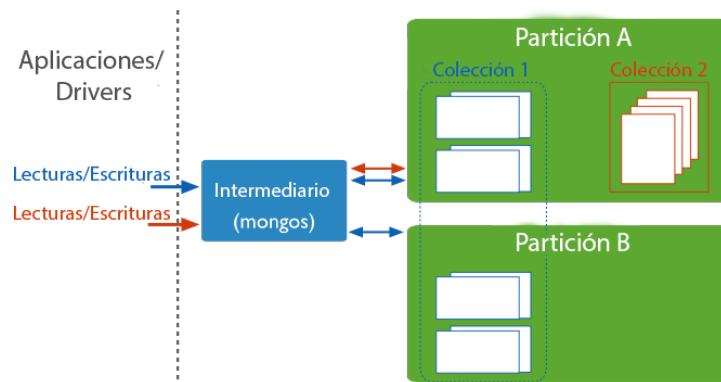


Figura 5: Interacción entre una aplicación cliente y una base de datos MongoDB particionada

MongoDB ofrece múltiples estrategias o políticas de particionamiento que permiten a un administrador de base de datos o desarrollador distribuir los datos entre los miembros del cluster.

- Range Sharding. Los documentos son divididos en rangos contiguos determinados por el valor del shard key. Se toma el valor mínimo y máximo de todas las particiones y se generan N grupos, ordenados de menor a mayor por el shard key y separados por rangos, tal cual se puede observar en la figura 6:

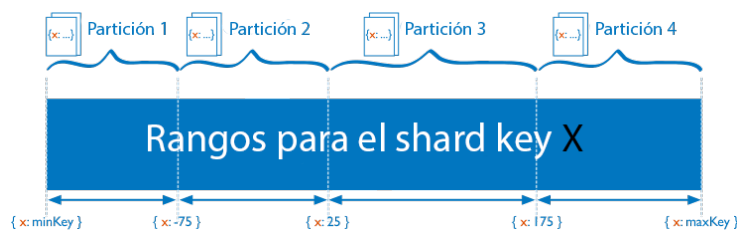


Figura 6: Range Sharding de datos en MongoDB

- Hash Sharding. Los documentos son distribuidos acorde al valor del hash MD5 del shard key. Una vez calculado el valor del hash, se dividen los datos en subgrupos separados en rangos. Este método garantiza la distribución uniforme de los documentos entre las particiones. MongoDB es quien calcula el valor del hash, por lo que las aplicaciones que interactúen con la base de datos no necesitan llevar a cabo este paso. Un ejemplo de esto se puede apreciar en la figura 7:

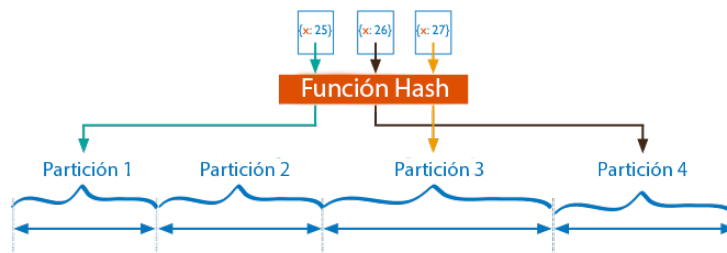


Figura 7: *Hashed Sharding de datos en MongoDB*

- **Zone Sharding.** Provee las herramientas para la definición de las reglas que definen el lugar en que serán almacenados los datos en el cluster. Se crean zonas de datos particionados basados en el shard key, y es posible asociar cada zona con uno o más particiones del cluster. Zone Sharding permite establecer políticas en donde se puede almacenar y localizar datos por región geográfica, por servicios específicos de una aplicación, e incluso por la configuración de hardware en caso de tener una arquitectura de almacenamiento por capas.

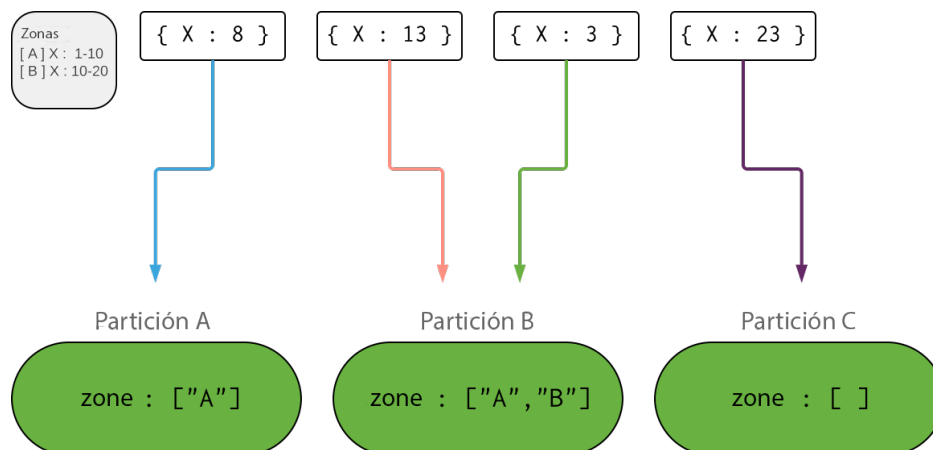


Figura 8: *Zone Sharding de datos en MongoDB*

2.1.3. PYMONGO

PyMongo es una librería de Python que permite la interacción con una base de datos MongoDB. Básicamente se encarga de mantener un canal de comunicación con la base de datos y ejecutar consultas [15].

PyMongo posee la cualidad de funcionar correctamente si es ejecutado simultáneamente por múltiples hilos, esta característica es llamada: segura

en hilos (thread-safe) [9]. PyMongo, tiene incorporado una agrupación de conexiones abiertas a la base de datos para que los hilos de una aplicación puedan reutilizar dichas conexiones al momento de ejecutar múltiples consultas o actualizaciones. Esta característica ayuda a incrementar el rendimiento puesto que PyMongo no establece una nueva conexión con la base de datos por cada que necesite ejecutar, de esta forma se evita la sobrecarga que la apertura de conexión pueda causar.

2.2. CELERY

Celery es una herramienta que permite creación de colas y tareas asíncronas basada en la transmisión de mensajes distribuidos. Esta concentrado en la ejecución de operaciones en tiempo real, y a su vez, también soporta la planificación de tareas.

Las colas de tareas son usadas como un mecanismo para la distribución de trabajo entre múltiples hilos o máquinas. Las entradas de estas colas son ejecutadas de forma concurrente en uno o más componentes llamados workers, quien constantemente verifican si existe una tarea en la cola para ser ejecutada. Estas tareas pueden ser ejecutadas en segundo plano o de forma síncrona, es decir, que esperan a que la tarea anterior culmine.

Celery realiza la transmisión de mensajes haciendo uso de un agente intermediario llamado broker. Para iniciar una tarea un cliente agrega un mensaje a la cola, el broker toma este mensaje y se lo envía a algún worker disponible, por último, una vez el worker recibe el mensaje ejecuta la tarea. Este proceso se puede observar en la figura 9:

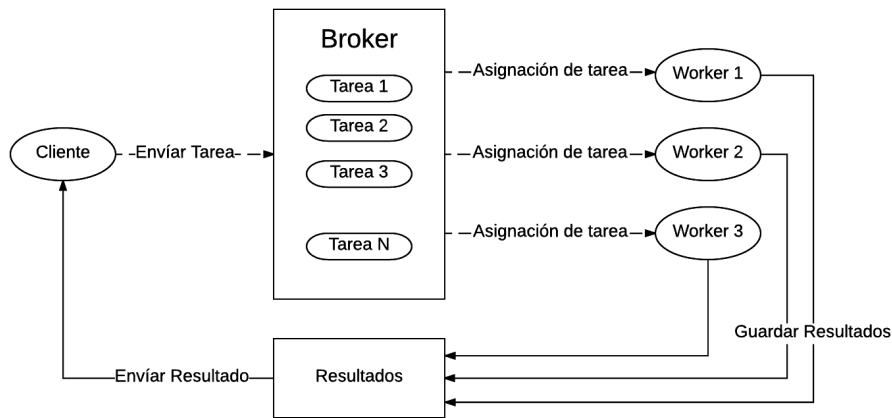


Figura 9: Flujo de trabajo en la asignación de tareas de Celery

2.2.1. FLOWER

Celery provee una herramienta de monitoreo de tareas por medio de una interfaz web, la cual hace un seguimiento de todas las peticiones recibidas por RabbitMQ y la respuesta generada por los nodos trabajadores que las atienden. Esta herramienta es llamada Flower, y permite llevar a cabo las siguientes acciones:

- Visualizar el historial de las tareas que se han ejecutado.
- Visualizar el estado y progreso de las tareas en ejecución, tareas planificadas y tareas canceladas.
- Mostrar los detalles relacionados a cada tarea, tales como: argumentos recibidos, marca de tiempo de inicio, tiempo de ejecución, entre otros.
- Cancelar tareas.
- Reiniciar y culminar la actividad de un trabajador.
- Visualizar y modificar la colas de tareas.
- Aplicar límites de tiempo y de rendimiento a cada tarea.

En la figura 10 se puede apreciar la interfaz que Flower ofrece al usuario.

Flower Dashboard Tasks Broker Monitor							Logout	Docs	Code
Show 10 entries		Search:							
Name	UUID	State	args	kwargs	Result	Received	Started		
app.tasks.app_tasks.extract_article	44f1fde9-4dd2-4c18-83c4-c05bceb500e6	SUCCESS	('Parque_nacional_El_Ávila', 'es', None)	{}	{'status': 'Task completed', 'result': '597 revisions extracted'}	2018-04-27 14:30:52.363	2018-04-2 14:30:52.3		
app.tasks.app_tasks.mode_task	e1656868-be9a-4d07-a2a9-93bdd577a80c	SUCCESS	(({'title': 'React'}, 'user'))	{}	{'status': 'Task completed', 'result': '[\"WikiPediaAid\"]'}	2018-04-27 14:29:30.194	2018-04-2 14:29:30.1		
app.tasks.app_tasks.count_task	415ac046-247e-42f5-bf1a-5e179a418283	SUCCESS	(({'user': 'Ta bu shi da yu'},))	{}	{'status': 'Task completed', 'count': '1'}	2018-04-27 14:27:43.976	2018-04-2 14:27:43.9		
app.tasks.app_tasks.count_task	bad166f1-6e5f-432c-b594-fe5250e1d40d	SUCCESS	(({'user': 'Jahaza'},))	{}	{'status': 'Task completed', 'count': '0'}	2018-04-27 14:25:16.304	2018-04-2 14:25:16.3		
Showing 1 to 4 of 4 entries							Previous	1	Next

Figura 10: *Interfaz web de Flower*

2.3. RABBITMQ

RabbitMQ es un intermediario para la transmisión de mensajes que sirve como plataforma para que los mensajes enviados en una aplicación sean almacenados de forma temporal durante el proceso de comunicación [19].

El uso de la mensajería en una aplicación permite la conexión entre múltiples componentes de la misma, ya sea para enviar notificaciones, datos, tareas asíncronas o suscripción a un servicio. Un mensaje puede incluir cualquier tipo de información, por ejemplo, puede contener datos relacionados con un proceso o tarea, una notificación, e incluso el resultado de una operación. RabbitMQ almacena estos mensajes y los coloca en una cola hasta que alguna aplicación se conecta a él y toma dicho mensaje de la cola.

RabbitMQ hace uso del protocolo AMPQP (Advanced Message Queuing Protocol), el cual esta diseñado para la transmisión de mensajes de aplicación entre sistemas usando, entre otras, una combinación de técnicas como: almacenamiento y reenvío, en donde los datos son almacenados en un componente intermediario de forma temporal y después enviados al destinatario final u a otro intermediario [7]; y el patrón publicación-suscripción, en donde el emisor del mensaje no envía datos a un destinatario en específico, sino que existe un miembro llamado suscriptor quien decide o no si recibir los mensajes publicados por el emisor [8].

En la figura 11 se puede apreciar un flujo de trabajo simple en la emisión y suscripción de mensajes usando RabbitMQ:



Figura 11: *Flujo de trabajo de la emisión y suscripción de mensajes en RabbitMQ*

2.4. FLASK

Flask es un framework (microframework) para el desarrollo aplicaciones web. Esta característica se refiere a que tiene como objetivo mantener el núcleo de la aplicación de simple pero extensible [21]. Es decir, no tiene limitaciones ni ofrece herramientas en el uso de servidores de bases de dato a usar, motores de plantillas, manejadores de correos o validaciones de formulario. En cambio, es capaz de soportar extensiones y librerías que agregan estas funcionalidades a la aplicación de tal forma que parecieran ser implementadas por Flask.

El objetivo de Flask es permitir la construcción de una base sólida para todas las aplicaciones, y que cualquier otro requerimiento o módulo necesario, que sea implementado por el desarrollador o por la inclusión de extensiones [20].

2.5. NGINX

Nginx es un servidor HTTP que también funciona como: un proxy reverso, servidor proxy IMAP/POP3 e inclusive como software para el manejo de balance de carga [16]. Puede ser usado para servir contenido HTTP estático y dinámico en la red haciendo uso de FASTCGI, manejadores SCGI para scripts, servidores de aplicación WSGI o módulos de Phusion Passenger.

Nginx no utiliza múltiples hilos para manejar solicitudes, en cambio, hace uso de un solo hilo y la implementación del paradigma de la programación dirigida por eventos, en donde el manejo de las peticiones recibidas por este servidor van determinados por los sucesos que ocurran en el sistema. La arquitectura modular dirigida por eventos de Nginx permite proveer un rendimiento predecible en ambientes de alta carga de trabajo.

La capacidad de Nginx de ser utilizado como proxy reverso tiene diversos usos. El uso de un proxy es típicamente usando para distribuir la carga de peticiones entre múltiples servidores, mostrar contenido de múltiples servicios web de forma transparente, e incluso transferir peticiones a un servidor por protocolos diferentes a HTTP [18].

Un ejemplo de como seria el flujo de trabajo para atender una petición con un proxy reverso se puede apreciar en la figura 12:

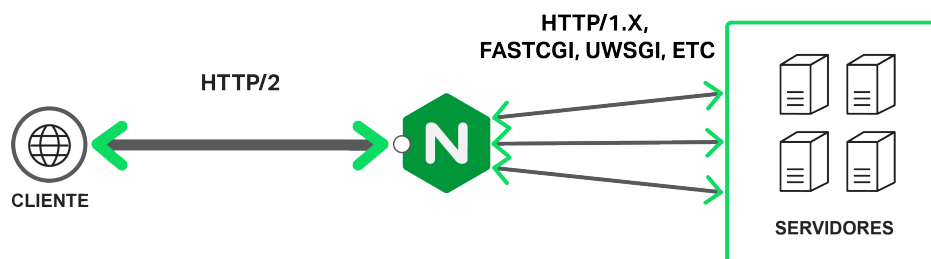


Figura 12: Atención de peticiones con el uso de proxy reverso de Nginx

Dado una aplicación cliente que emite una petición al servicio, es Nginx es quien recibe la petición y procede a elegir cual servidor atenderá la petición. Cada servidor ejecuta la aplicación haciendo uso de tecnologías como UWSGI, FASTCGI, SCGI, etc. Una vez que el servidor atiende la petición, envía la respuesta devuelta a Nginx y este último la reenvía al cliente.

2.6. DOCKER

Docker es una plataforma para la automatización en el despliegue de aplicaciones dentro de paquetes llamados contenedores de software, los cuales contienen todos los elementos que permiten su ejecución en cualquier sistema operativo.

Un contenedor de software es una manera de empaquetar software en un formato en el cual puede ser ejecutado de forma aislada en un sistema operativo. Solo se empaqueta el código, sus librerías, dependencias y configuraciones necesarias para que el software funcione de forma adecuada sin ejecutar un sistema operativo por completo. De esta manera se generan sistemas independientes, eficientes y ligeros que garantizan que el software siempre se ejecutará de la misma manera sin importar el ambiente [14].

Los contenedores de software están conformados por dos componentes: una imagen y un contenedor.

- Imagen: es un paquete ejecutable aislado que incluye todo lo necesario para ejecutar un software, incluyendo código, librerías, variables de entorno y archivos de configuración.
- Contenedor: es una instancia de una imagen. Su ejecución, por defecto, esta aislada del ambiente de la máquina anfitrión, de tal forma que solo accede a los archivos y/o puertos del anfitrión si fue configurado para ello.

Los contenedores ejecutan aplicaciones directamente en el kernel de la máquina anfitrión, por lo que tienen mejor rendimiento que una máquina virtual que solo tiene acceso virtual a los recursos del anfitrión.

La figura 13 muestra cómo las máquinas virtuales ejecutan sistemas operativos completos sobre la máquina anfitrión:

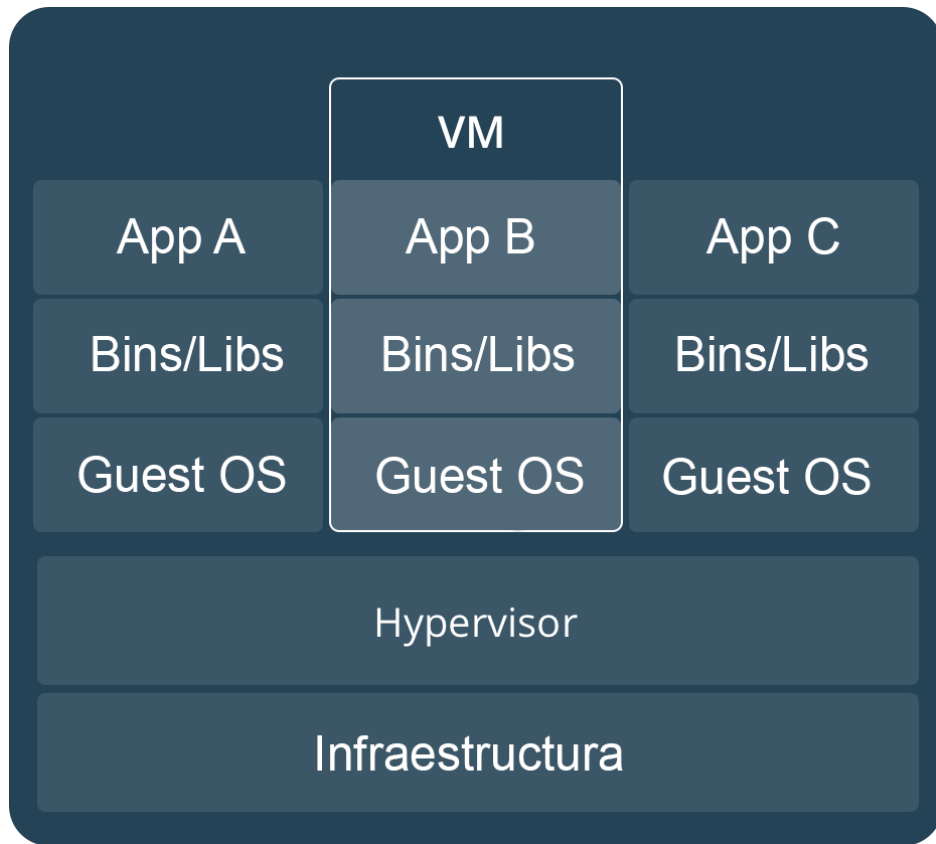


Figura 13: *Diagrama de componentes de una Máquina Virtual*

En cambio, los contenedores pueden compartir un mismo kernel, la única información que necesita tener la imagen del contenedor es el ejecutable y sus dependencias. En la figura 14 se puede apreciar como cada contenedor se ejecutará como una aplicación aislada sobre el sistema operativo:



Figura 14: *Diagrama de componentes de un Contenedor*

Docker permite la creación de redes y espacios de almacenamiento, llamados volúmenes, que pueden ser asignados a uno o múltiples contenedores a la vez. De esta forma, es posible conectar varios contenedores entre sí mediante una red y compartir archivos. Esto permite gestionar múltiples contenedores para que se ejecuten como un sistema distribuido, en donde es posible, y si los recursos lo permiten, desplegar nuevos nodos que cumplan las funciones como: un miembro en una réplica de base de datos, un worker de celery e incluso un servicio como RabbitMQ.

3. MARCO APLICATIVO

3.1. MÉTODO DE DESARROLLO

Para llevar a cabo el cumplimiento de los objetivos, es necesario definir un esquema o metodología de trabajo que permita la elaboración de un conjunto de lineamientos de manera estructurada y organizada. Por lo tanto, se hace uso de el método de Desarrollo Rápido de Aplicaciones (RAD), el cual permite la iteración rápida y continua de pequeños objetivos para alcanzar la meta final.

RAD es un proceso de desarrollo de software que integra un conjunto de técnicas, lineamientos y herramientas que permiten llevar a cabo, en cortos periodos de tiempo, la implementación de funcionalidades en un sistema, de tal manera que satisfacen las necesidades del cliente [2]. Siguiendo esta metodología, el software evoluciona y crece durante el proceso de desarrollo en base a la retro alimentación que se tiene con el cliente. De esta manera, se realizan múltiples entregas de una tarea que contiene las nuevas funcionalidades esperadas. Cada una de estas tareas cuenta con instrumento de documentación al cuál llamaremos historias. Estas historias describen los detalles técnicos e categorizan las tareas en subgrupos.

Por medio de RAD, fue posible la implementación de la solución al problema previamente planteado a través del desarrollo de múltiples componentes relacionados entre si, para brindar un servicio web como un API. La figura 15 muestra la arquitectura general del servicio web, y como interactuá con las aplicaciones clientes que lo consumen.

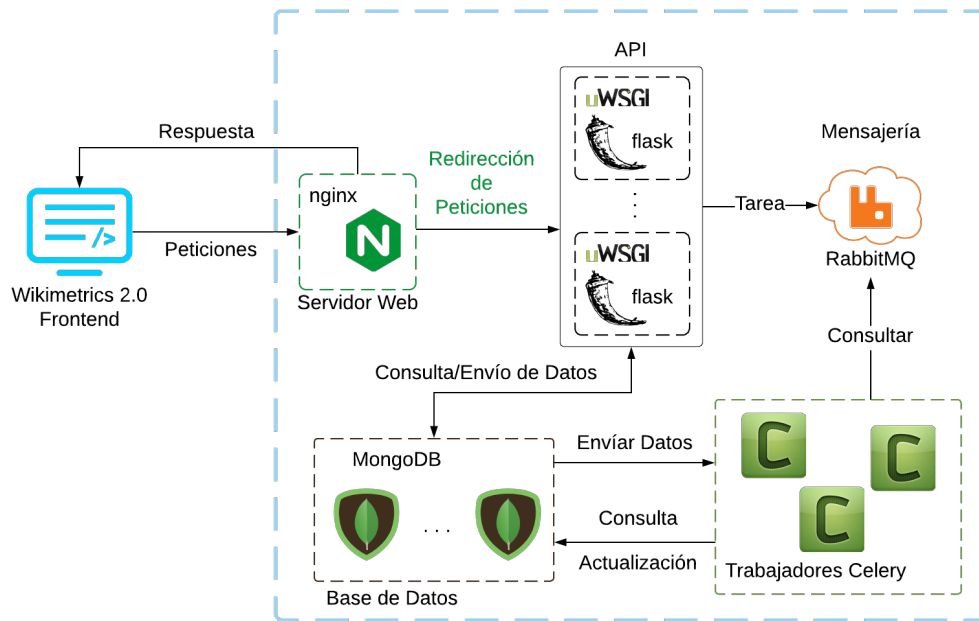


Figura 15: *Arquitectura general de la aplicación*

Los componentes que conforman este servicio son los siguientes

- Un servidor web (Nginx) que atiende, en primera instancia, las peticiones de las aplicaciones clientes y balancea la carga entre múltiples nodos que almacenan el API.
- Una o más aplicaciones Flask, almacenadas en máquinas separadas, que ejecutan el API e inician las tareas de consulta y extracción de revisiones.
- Un servicio de mensajería (RabbitMQ) que recibe las tareas iniciadas por el API y las encola para su posterior ejecución.
- Uno o más nodos que ejecutan un proceso trabajador de Celery para la ejecución de las tareas almacenadas en RabbitMQ. Estos nodos mantienen una comunicación con RabbitMQ para actualizar el estado y progreso de las tareas.
- Y, por último, un componente de almacenamiento de artículos wiki y sus revisiones. Para ello se hace uso de MongoDB y está implementado como un cluster de base de datos distribuidos entre múltiples máquinas conectadas entre sí que permiten la fragmentación de los datos.

3.2. SERVIDOR WEB

El servidor web que atiende las solicitudes en primera instancia es Nginx, el cual funciona como un intermediario entre el cliente y el API que responde dicha solicitud. Nginx permite balancear la carga de trabajo entre múltiples servidores que contienen la aplicación mediante el algoritmo de round robin, de tal forma que cada servidor es utilizado de forma equitativa, tal como se muestra en la figura 16.

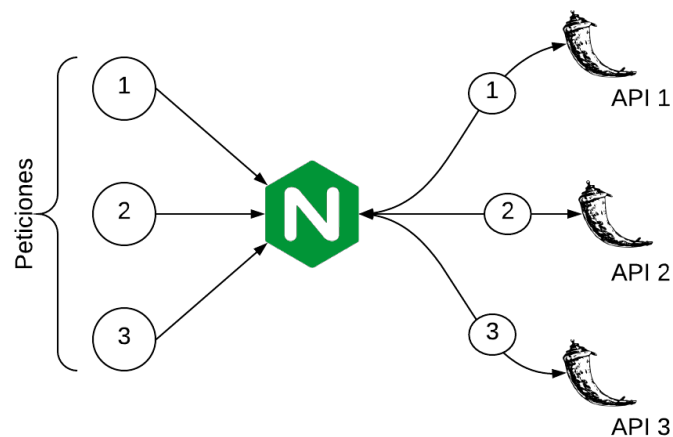


Figura 16: *Algoritmo de distribución de peticiones de Nginx basado en método de Round Robin.*

Una vez la petición ha sido reasignada, es atendida por medio del servidor web UWSGI, y en conjunto con Flask y el resto de las tecnologías incluidas en el API, genera una respuesta de vuelta al cliente.

3.3. API

El API diseñado para realizar las labores de extracción y consulta de las revisiones, hace uso del lenguaje de programación Python 2.7.10 y el uso del framework Flask v0.12. Ambas tecnologías permiten la asignación de rutas específicas para cada módulo y la atención de peticiones.

La interacción entre el servicio y una aplicación cliente puede ser apreciada en la figura 17.

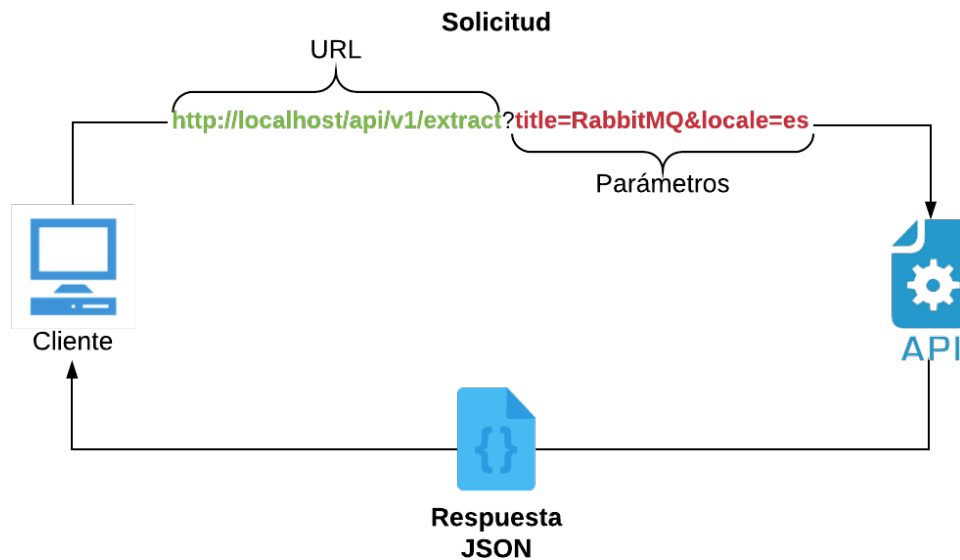


Figura 17: Interacción entre una aplicación cliente y el API

Esta interacción se basa en la petición de un recurso o de un proceso, por parte de una aplicación cliente, a una ruta del servidor web (endpoint). Cada petición puede incluir parámetros opcionales para la paginación, filtrado de resultados, como por ejemplo, el idioma del artículo a extraer.

Las rutas de acceso que ofrece el API son las siguientes:

- `/api/v1/articles`. Listado de todos los artículos extraídos.
- `/api/v1/revisions`. Listado de las revisiones de los artículos extraídos.
- `/api/v1/extract`. Extracción de revisiones de artículos wiki.
- `/api/v1/avg`. Cálculo del promedio de revisiones de un artículo por rango fechas.
- `/api/v1/count`. Cálculo de número de revisiones de un artículo por fecha.
- `/api/v1/mode`. Cálculo de las revisiones mas extraídas por rango de fecha.

- `/api/v1/status`. Indica el estado del proceso de extracción.
- `/api/v1/query`. Permite la ejecución de consultas personalizadas a la base de datos.

3.3.1. EXTRACCIÓN DE HISTORIALES

Para la extracción de historiales se hace uso de la ruta de acceso `/api/v1/extract`, la cual tiene como parámetros: `title`, que se refiere al título del artículo; `url`, representa la ruta completa de un artículo, por ejemplo: `https://en.wikipedia.org/wiki/The_Lord_of_the_Rings`; y por último, `locale`, el cual indica el idioma del artículo y es completamente opcional, puesto que el sistema asume el inglés como lenguaje por defecto o lo extrae del parámetro `url`. Es necesario proporcionar el parámetro `url` o `title` de forma obligatoria para poder llevar a cabo la extracción.

En la figura 18 se puede apreciar dos ejemplos del uso de los parámetros `url`, `title` y `locale`:

`http://localhost/api/v1/extract?title=RabbitMQ&locale=es`
 Título Idioma
`http://localhost/api/v1/extract?url=https://en.wikipedia.org/wiki/The_Lord_of_the_Rings`
 Idioma Título

Figura 18: *Parámetros para la extracción de historiales*

La ejecución de esta tarea se lleva a cabo por medio de Celery y RabbitMQ. Con estas tecnologías, múltiples computadores ejecutan un proceso de Celery, los cuales están conectados entre sí gracias a RabbitMQ, escuchando constantemente los mensajes entrantes de este servicio. Una vez recibida la petición por el API, se genera una tarea de extracción por medio de Celery, la cual tiene un identificador único. Esta tarea es encolada en una lista de espera que es atendida por uno de los trabajadores conectados a RabbitMQ y es ejecutada en segundo plano. Posteriormente, el API crea una respuesta a la petición generando una ruta para consultar el progreso del proceso de extracción, la cual luce como se muestra en la figura 19:

```

{
  "Location": "http://localhost/api/v1/status/34842158-fb95-456b-9284-ada8c31b5fd4?name=extract_article"
}

```

Figura 19: *Respuesta a una petición de extracción de historiales de un artículo*

Durante el proceso de extracción, se realizan peticiones hacia el API de Wikipedia. Se toma como URL por defecto la dirección <https://en.wikipedia.org/w/api.php>, en conjunto con diversos parámetros, tales como:

- **action:** tipo de petición que se realiza sobre este API, en este caso se hace uso del valor **query** para indicar que solo se realizará una consulta.
- **format:** formato de la respuesta obtenida, la cual puede ser XML o JSON.
- **props:** la propiedad del artículo al cual se desea consultar, en este caso se usa el valor **revisions** para poder extraer el historial de modificaciones del mismo.
- **rvprop:** atributos de la propiedad extraída, los incluidos para esta extracción son los siguientes: ids, flags, timestamp, user, userid, size, sha1, contentmodel, comment, parsedcomment, content, tags.
- **rvlimit:** permite la paginación de los resultados e indica la cantidad de resultados que se quieren obtener por página.
- **newer:** permite ordenar las revisiones de forma ascendente o descendiente acorde a la fecha de creación. Se ordenan de forma descendente usando el valor **newer**, de esta manera es más rápido extraer las revisiones mas recientes.

Justo antes de enviar la solicitud al API de wikipedia, se verifica que el artículo a consultar ya esté almacenado en la base de datos, en caso de no estarlo, se almacena indicando la fecha de extracción. Posteriormente, se extraen sus revisiones, y puesto que la respuesta del API coloca las revisiones mas recientes al principio, el extractor culmina su ejecución una vez determina que alcanzó una revisión que ya ha sido almacenada. Al almacenar las revisiones en la base de datos, se actualiza el documento del artículo con la fecha de la última extracción y el identificador de la revisión

extraída, y por último, se actualiza el progreso de la tarea. Por cada página de revision extraída, el sistema espera un segundo para ejecutar la siguiente llamada al API, de esta manera se respetan los lineamientos recomendados por Mediawiki para la consulta de los datos y evitar la sobrecarga de su servicio.

En la figura 20 se puede apreciar un ejemplo de la consulta del progreso de las tareas:

```
▼ {
  "state": "IN PROGRESS",
  "status": "100 revisions extracted"
}
```

Figura 20: Respuesta a una petición para consultar de progreso de una tarea de extracción

La tabla 1 muestra la historia asociada a la extracción de revisiones.

Historia	
Desarrollador	Marvin Bernal
Nombre	Extracción de revisiones por API
Sección	Extracción
Descripción	Extracción de todas las revisiones de un artículo wiki dado su título a través del API que ofrece mediawiki Se realiza una petición sobre el URL https://en.wikipedia.org/w/api.php y se agregan parámetros extra, incluyendo el título del artículo, para obtener todos los metadatos posibles de una revisión. Cada petición al API obtiene un máximo de 50 revisiones se llevan a cabo con una diferencia de 2 segundos.
Observaciones	Se extraen los siguientes datos: id de la revisión, tipo de revisión, fecha, nombre de usuario, id de usuario, tamaño, comentarios, contenido y etiquetas

Tabla 1: Extracción de Revisiones

3.3.2. CONSULTAS

El API consta de métodos de acceso para consultar las revisiones extraídas y los artículos asociados a través de las rutas `/api/v1/revisions` y `/api/v1/articles` respectivamente.

En estas rutas de consulta es posible hacer uso de diversos parámetros de búsqueda, entre ellas están: los parámetros de paginación `page` y `page_size`, y los parámetros de búsqueda que corresponden a los atributos del artículo, tales como: `title`, `first_extraction_date`, `last_extraction_date`, `last_revision_extracted` y `locale`. Adicionalmente, existe el parámetro `sort`, que permite ordenar los resultados a través del valor de un atributo de la colección a consultar, ejemplo, por la fecha de extracción de un artículo.

La ruta `articles` tiene como finalidad mostrar la información pertinente a los artículos que han sido extraídos a través del API, para lo cual se realiza una consulta a la colección `articles` almacenada en la base de datos. Un ejemplo de la respuesta obtenida por esta ruta puede ser apreciada en la figura 21.

```
▼ {
  "ns": 0,
  "pageid": 4606,
  "title": "Battle of the Nile",
  "first_extraction_date": "2017-09-28T02:36:34.798000",
  "last_extraction_date": "2017-09-28T02:40:03.942000",
  "last_revision_extracted": 802628501,
  "_id": {
    "$oid": "59cc6032523748000e361d6e"
  }
},
▼ {
  "ns": 0,
  "pageid": 3787053,
  "title": "RabbitMQ",
  "locale": "es",
  "first_extraction_date": "2017-11-10T05:57:17.455000",
  "last_extraction_date": "2017-12-14T05:14:21.635000",
  "last_revision_extracted": 87575870,
  "_id": {
    "$oid": "5a053fbd446506000e40be1d"
  }
},
```

Figura 21: Respuesta a una petición de consulta de artículos

Por su parte, la ruta `revisions` permite consultar el contenido y los datos pertinentes a las revisiones que han sido extraídas. En la figura 22 se puede apreciar una respuesta del servidor a la consulta de revisiones.

Figura 22: *Respuesta a una petición de consulta de revisiones*

Durante el cálculo de estas métricas se toman como parámetros múltiples atributos, tales como: el título del artículo o la fecha de extracción. Dichos atributos son usados para reducir o filtrar las revisiones que se van a consultar en el cálculo de métricas.

Posteriormente, se genera una tarea que será procesada en segundo plano por medio de Celery, con el tipo de consulta a realizar y los parámetros de búsqueda.

El formato de algunos atributos utilizados en el proceso de búsqueda de revisiones o artículos, tales como, fecha de extracción y tamaño de la revisión, es alterado para facilitar la consulta de revisiones por un rango o intervalo determinado. Para los atributos que representan una fecha, se condiciona la búsqueda en base a una fecha única o un intervalo de fechas por medio de los parámetros recibidos en la petición. Para atributos que representan el tamaño se ejecuta un proceso similar, donde se considera un segundo argumento opcional que indica el intervalo a evaluar para condicionar la consulta.

En la tabla 2 se puede apreciar, con detalle, y por medio de su historia, las condiciones de búsqueda aceptadas por dicha consulta.

Historia	
Desarrollador	Marvin Bernal
Nombre	Endpoint para el cálculo de la cantidad de revisiones
Sección	Consulta

Tabla 2 – continuación de la página anterior

	Historia
Descripción	<p>Parte del API de la solución distribuida donde el endpoint <code>/count</code> recibe como parámetros los atributos para restringir o filtrar el rango a tomar en cuenta como resultado. Los argumentos disponibles hasta el momento son:</p> <ul style="list-style-type: none"> • <code>title</code>: título del artículo extraído. • <code>pageid</code>: id del artículo extraído. • <code>user</code>: nombre del usuario que realiza la revisiones. • <code>userid</code>: id del usuario que realiza la revisiones. • <code>tag</code>: una etiqueta determinada que contenga las revisiones. • <code>size</code>: el tamaño de la revisión realizada. • <code>sizematch</code>: valor que acompaña a <code>size</code>. Si el valor es positivo, se filtrarán todas las revisiones de mayor tamaño que el valor de <code>size</code>. Si es negativo, se filtrarán todas las revisiones de menor tamaño que el valor de <code>size</code>. Si el valor es 0 o no se encuentra en los parámetros de la solicitud, se filtrarán todas las revisiones cuyo tamaño sea exactamente el valor de <code>size</code>. • <code>date</code>: la fecha exacta en que fueron realizadas las revisiones. El formato de fecha utilizado es: <code>YYYY-MM-DD</code>. • <code>datestart</code>: la fecha inicial a partir de la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: <code>YYYY-MM-DD</code>. En caso de que no exista el parámetro <code>dateend</code> en la solicitud, la fecha final del intervalo será la fecha actual. • <code>dateend</code>: la fecha final hasta la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: <code>YYYY-MM-DD</code>. En caso de que no exista el parámetro <code>datestart</code> en la solicitud, la fecha inicial del intervalo será la fecha de la primera revisión del artículo.
Observaciones	<p>Primero se realiza una comprobación de los parámetros de la solicitud contra una lista blanca con los parámetros permitidos. Luego, con los parámetros resultantes, se realiza el procesamiento a través de una tarea asignada, para luego devolver el resultado.</p>

Tabla 2: *Contabilización de Revisiones*

La consulta del promedio revisiones se calcula haciendo uso de intervalos de tiempos (en este caso, días), determinados por el usuario en la petición al servicio. En la tabla 3 se presentan con mayor con detalle, por medio de su historia, los filtros aceptados, así como los argumentos obligatorios de la consulta.

Historia	
Desarrollador	Marvin Bernal
Nombre	Endpoint para el cálculo del promedio de revisiones
Descripción	<p>Parte del API de la solución distribuida donde el endpoint <code>/avg</code> recibe como parámetros los atributos para restringir o filtrar el rango a tomar en cuenta como resultado, adicionalmente es necesario un intervalo de fechas para el cálculo del promedio. Los argumentos disponibles hasta el momento son:</p> <ul style="list-style-type: none"> • <code>title</code>: título del artículo extraído. • <code>pageid</code>: id del artículo extraído. • <code>user</code>: nombre del usuario que realiza la revisiones. • <code>userid</code>: id del usuario que realiza la revisiones. • <code>tag</code>: una etiqueta determinada que contenga las revisiones. • <code>size</code>: el tamaño de la revisión realizada. • <code>sizematch</code>: valor que acompaña a <code>size</code>. Si el valor es positivo, se filtrarán todas las revisiones de mayor tamaño que el valor de <code>size</code>. Si es negativo, se filtrarán todas las revisiones de menor tamaño que el valor de <code>size</code>. Si el valor es 0 o no se encuentra en los parámetros de la solicitud, se filtrarán todas las revisiones cuyo tamaño sea exactamente el valor de <code>size</code>. • <code>datestart</code>: la fecha inicial a partir de la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: <code>YYYY-MM-DD</code>. Este argumento es obligatorio. • <code>dateend</code>: la fecha final hasta la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: <code>YYYY-MM-DD</code>. Este argumento es obligatorio.
Observaciones	<p>Primero se realiza una comprobación de los parámetros de la solicitud contra una lista blanca con los parámetros permitidos. Luego, con los parámetros resultantes, se realiza el procesamiento a través de una tarea asignada, para luego devolver el resultado.</p>

Tabla 3 – continuación de la página anterior

Historia

Tabla 3: *Promedio de Revisiones*

Las consultas sobre la ruta `/api/v1/mode` permiten calcular el valor mas frecuente de un atributo determinado. Dicha consulta se realiza sobre un conjunto de revisiones que pueden ser filtradas con parámetros de búsqueda, los cuales pueden ser apreciados en la tabla 4. Este tipo de consulta permite, por ejemplo, calcular cuál es el mayor contribuyente a un artículo en un año determinado.

Historia	
Desarrollador	Marvin Bernal
Nombre	Endpoint para el cálculo de la moda de las revisiones
Sección	Consulta
Descripción	<p>Parte del API de la solución distribuida donde el endpoint <code>/mode</code> recibe como parámetros los atributos para restringir o filtrar el rango a tomar en cuenta como resultado, adicionalmente se usa el parámetro <code>attribute</code> para señalar el atributo sobre el cual calcular la moda, que consiste en el valor con mayor número de repeticiones entre el conjunto de valores obtenidos. Entre los valores posibles que puede asumir <code>attribute</code> se encuentran:</p> <ul style="list-style-type: none"> • <code>title</code>: el título del artículo con mayor cantidad de revisiones. • <code>pageid</code>: el id del artículo con mayor cantidad de revisiones. • <code>user</code>: nombre del autor que mas revisiones haya realizado. • <code>userid</code>: id del autor que mas revisiones haya realizado. • <code>date</code>: la fecha en la cual se hayan realizado mas revisiones. • <code>size</code>: el tamaño de revisión mas repetido.

Tabla 4 – continuación de la página anterior

	Historia
Descripción	<p>Adicionalmente, se pueden utilizar argumentos de filtrado, para especificar el conjunto de revisiones sobre el cuál se realiza el cálculo. Los argumentos disponibles hasta el momento son:</p> <ul style="list-style-type: none"> • title: título del artículo extraído. • pageid: id del artículo extraído. • user: nombre del usuario que realiza la revisiones. • userid: id del usuario que realiza la revisiones. • tag: una etiqueta determinada que contenga las revisiones. • size: el tamaño de la revisión realizada. • sizematch: valor que acompaña a size. Si el valor es positivo, se filtrarán todas las revisiones de mayor tamaño que el valor de size. Si es negativo, se filtrarán todas las revisiones de menor tamaño que el valor de size. Si el valor es 0 o no se encuentra en los parámetros de la solicitud, se filtrarán todas las revisiones cuyo tamaño sea exactamente el valor de size. • date: la fecha exacta en que fueron realizadas las revisiones. El formato de fecha utilizado es: YYYY-MM-DD. • datestart: la fecha inicial a partir de la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: YYYY-MM-DD. En caso de que no exista el parámetro dateend en la solicitud, la fecha final del intervalo será la fecha actual. • dateend: la fecha final hasta la cual fueron realizadas las revisiones en un intervalo de tiempo. El formato de fecha utilizado es: YYYY-MM-DD. En caso de que no exista el parámetro datestart en la solicitud, la fecha inicial del intervalo será la fecha de la primera revisión del artículo.

Tabla 4 – continuación de la página anterior

	Historia
Observaciones	<p>Primero se realiza una comprobación de los parámetros de la solicitud contra una lista blanca con los parámetros permitidos. Luego, con los parámetros resultantes, se realiza el procesamiento a través de una tarea asignada, para luego devolver el resultado.</p> <p>Se recomienda precaución al seleccionar los argumentos de filtrado, para evitar resultados redundantes. Por ejemplo, si se selecciona el atributo <code>user</code> como filtro y <code>attribute</code> tiene como valor <code>'user'</code>, el resultado será en el mejor caso el valor recibido por el filtro <code>user</code>, pues el conjunto seleccionado solo consistirá en revisiones realizadas por dicho usuario.</p>

Tabla 4: *Moda de Revisiones*

Por último, el API provee la ruta `/api/v1/query` que permite ejecutar consultas a la base de datos directamente de los parámetros recibidos por la petición. El cuerpo de las peticiones realizadas sobre esta ruta debe tener un formato JSON, en conjunto con el atributo de cabecera `content-type` que debe tener como valor `application/json`. La estructura del cuerpo de la petición debe coincidir con el formato aceptado para la creación de consultas por medio de PyMongo. Un ejemplo de este formato puede ser apreciado en la figura 23, en donde cada clave de la dupla clave-valor del archivo JSON representa una función de agregación válida de MongoDB:

```

{
  "$match": {
    "extraction_date": "2017-09-28T02:36:33.452000",
    "pageid": 4606
  },
  {
    "$project" :{
      "pageid": 1 , "timestamp" : 1
    }
  },
  { "$limit" : 5 }
}

```

Figura 23: Estructura del cuerpo de la petición a la ruta `/api/v1/query`

Adicionalmente, la ruta permite especificar la colección de la base de datos a la cual consultar, y de forma opcional, el formato de fecha de alguna columna en caso de ser necesario. Un ejemplo, del uso de estos parámetros es el que se muestra en la figura 24, en donde se realiza una consulta de la colección llamada `revisions`

`http://localhost/api/v1/query?collection=revisions&date_format=%Y-%m-%dT%H:%M:%S.%f`

Figura 24: Estructura del cuerpo de la petición a la ruta `/api/v1/query`

Por medio de esta ruta es posible realizar cualquier tipo de consulta sobre la base de datos, y así, obtener métricas adicionales para su análisis posterior. En la tabla 5 se puede ver mas detalles con respecto a los parámetros.

Historia	
Desarrollador	Francisco Delgado
Nombre	Endpoint para la consultas directas a la base de datos
Sección	Consulta

Tabla 5 – continuación de la página anterior

	Historia
Descripción	<p>Parte del API de la solución distribuida donde el endpoint <code>/query</code> recibe el contenido de la consulta a realizar en formato JSON, a través del cuerpo de la solicitud que se realiza. Dicho cuerpo debe tener un formato adecuado para realizar la consulta en base al módulo Pymongo.</p> <p>Además de esto, existen 2 argumentos opcionales:</p> <ul style="list-style-type: none"> • <code>collection</code>: la colección sobre la cuál se hará la consulta. Su valor por defecto es la colección <code>revisions</code>. • <code>date.format</code>: el formato de fecha que será usado en la consulta. Su valor por defecto es <code>%Y-%m-%dT%H:%M:%S</code>. Donde cada letra excepto la T representa el valor de unidad de tiempo correspondiente en inglés, por ejemplo Y para Año (Year). La letra T representa la finalización de la sección de la fecha y el inicio de la sección del tiempo en el formato.
Observaciones	

Tabla 5: *Consultas Directas a la Base de Datos*

3.4. ALGORITMO DE REVISITA

Una vez el proceso de extracción esta completo, es necesario mantener una actualización constante de los historiales del artículo. Para ello se hace uso de un algoritmo, al cual llamaremos algoritmo de revisita, que acorde a ciertos coeficientes, evalúe si es necesarios ejecutar una petición de extracción de los historiales de un artículo. Este algoritmo es ejecutado en segundo plano como una tarea programada, y para ello se hace uso del programa llamado Cron. Cron es un planificador de tareas, basado en tiempo, para sistemas operativos basados en Unix, que permite la ejecución de programas o comandos de forma automática en una fecha o frecuencia de tiempo determinado [1].

El código fuente 1 muestra el proceso de edición del archivo de configuración previamente mencionado, en donde se hace uso del comando `crontab -e` para acceder fácilmente al archivo, y ademas, se muestra el formato aceptado por Cron para asignar la frecuencia de ejecución.

```

$ crontab -e
#----- Minutos (0 - 59)
| #----- Horas (0 - 23)
| | #----- Dia del mes (1 - 31)
| | | #----- Mes (1 - 12)
| | | | #---- Dia de la semana (0 - 6) (Domingo=0)
| | | | |
* * * * * ejemplo-script.sh

```

Código Fuente 1: Edición del archivo de configuración de Cron

La configuración de Cron para el algoritmo de revisita puede ser apreciado en el código fuente 2, en el cual el comando `python -m app.cronjobs.revisit` se ejecuta de forma diaria a las 12:00AM.

```

$ crontab -e
0 0 * * * python -m app.cronjobs.revisit

```

Código Fuente 2: Edición del archivo de configuración de Cron

El algoritmo de revisita hace uso de un modelo probabilístico que calcula el coeficiente del tiempo revisita de cada artículo wiki de manera independiente haciendo uso de la distribución probabilística exponencial.

El objetivo es buscar un valor esperado de tiempo que mida la distancia entre dos revisiones para ejecutar una revisita de un artículo wiki. Para ello, se plantea utilizar la fórmula de la esperanza matemática proveniente de la distribución de probabilidad exponencial, la cual es adecuada para estos casos, puesto que la misma estudia la longitud de los intervalos de una variable.

Siendo x una variable aleatoria que sigue una distribución exponencial, la cual mide el intervalo de tiempo transcurrido entre dos revisiones de un artículo de wiki; y sea λ el promedio de revisiones por unidad de tiempo, podemos obtener valor medio o esperanza matemática de x con la siguiente fórmula:

$$E(x) = 1/\lambda$$

Para calcular el promedio de revisiones por unidad de tiempo, se toma una muestra de las revisiones almacenadas en el sistema. Es necesario que se tome en cuenta que cada artículo wiki tiene una frecuencia de revisiones que la distingue de las demás. Por ejemplo, para eventos deportivos recurrentes como las olimpiadas o copas mundiales de fútbol, la cantidad de revisiones que se ejercen sobre dichos artículos disminuyen drásticamente una vez culminado el evento, por lo que su frecuencia varía dependiendo de qué intervalos de tiempo se tome en cuenta a la hora de evaluarlos. Por lo tanto, para poder obtener el valor más adecuado a la frecuencia actual de los artículos, se toma como muestra el mayor número entre: las últimas 20 revisiones del artículo y el más reciente 10 % de la cantidad total de revisiones del artículo.

Una vez extraída la muestra, es posible calcular λ con la siguiente fórmula:

$$\lambda = n/(t1 - t0)$$

Siendo n el número de revisiones de la muestra; $t1$ la fecha de la revisión más reciente en la muestra; y $t0$ la fecha de la revisión más antigua en la muestra.

Por último, se toma la fecha de la última revisión y se calcula la diferencia entre la fecha actual. Si dicho valor es mayor a la esperanza matemática $E(x)$, entonces se volverá a visitar el artículo. A medida que el λ se eleve, menor es el intervalo de tiempo entre cada ocurrencia, y por lo tanto, el intervalo de tiempo entre cada revisita del artículo es menor.

Usemos 2 casos de ejemplo:

- El artículo A tiene 280 revisiones almacenadas en el sistema.

1. Se determina el valor más alto entre 20 y el 10 % de 280.

$$\begin{aligned}
100 \% &= 280\text{revisiones} \\
10 \% &= x \\
x &= (280\text{revisiones} \times 10 \%)/100 \% \\
x &= 28\text{revisiones}
\end{aligned}$$

Se toma 28 como el número de la muestra a evaluar.

Se calcula el intervalo de tiempo entre la primera y la última revisión ($t_1 - t_0$) de la muestra de 28 revisiones. Asumamos que dicho valor es de 14 días, con el cual podemos calcular λ de la siguiente manera:

$$\begin{aligned}
\lambda &= n/(t_1 - t_0) \\
\lambda &= 28\text{revisiones}/14\text{días} \\
\lambda &= 2\text{revisiones}/\text{día}
\end{aligned}$$

2. Ahora, para calcular el valor de la esperanza:

$$\begin{aligned}
E(x) &= 1/\lambda \\
E(x) &= 1/(2\text{revisiones}/\text{día}) \\
E(x) &= 0,5(\text{días}/\text{revisión})
\end{aligned}$$

3. A continuación, podemos convertir la esperanza a otra unidad de tiempo más conveniente, por ejemplo horas, haciendo una conversión básica:

$$\begin{aligned}
E(x) &= (0,5\text{días}/\text{revisión}) * (24\text{horas}/1\text{día}) \\
E(x) &= 0,5 * 24\text{horas}/\text{revisión} \\
E(x) &= 12\text{horas}/\text{revisión}
\end{aligned}$$

4. Si han pasado más de 12 horas desde la fecha de la última revisión, el artículo será encolado para su extracción.

- El artículo B tiene 1350 revisiones almacenadas en el sistema.

1. Se determina el valor más alto entre 20 y el 10 % de 1350.

$$\begin{aligned}
 100 \% &= 1350 \text{revisiones} \\
 10 \% &= x \\
 x &= (1350 \text{revisiones} * 10 \%) / 100 \% \\
 x &= 135 \text{revisiones}
 \end{aligned}$$

2. Se toma 135 como el número de la muestra a evaluar.
3. Se calcula el intervalo de tiempo entre la primera y la última revisión ($t1 - t0$) de la muestra de 135 revisiones. Asumamos que dicho valor es de 13 días, con el cual podemos calcular λ de la siguiente manera:

$$\begin{aligned}
 \lambda &= n / (t1 - t0) \\
 \lambda &= 135 \text{revisiones} / 13 \text{días} \\
 \lambda &= 10,38 \text{revisiones} / \text{día}
 \end{aligned}$$

4. Ahora, para calcular el valor de la esperanza:

$$\begin{aligned}
 E(x) &= 1 / \lambda \\
 E(x) &= 1 / (10,38 \text{revisiones} / \text{día}) \\
 E(x) &= 0,096 (\text{días} / \text{revisión})
 \end{aligned}$$

5. A continuación, podemos convertir la esperanza a otra unidad de tiempo más conveniente, por ejemplo horas, haciendo una conversión básica:

$$\begin{aligned}
 E(x) &= (0,096 \text{días} / \text{revisión}) * (24 \text{horas} / 1 \text{día}) \\
 E(x) &= 0,096 * 24 \text{horas} / \text{revisión} \\
 E(x) &= 2,31 \text{horas} / \text{revisión}
 \end{aligned}$$

Con esto, se puede verificar que en intervalos de mayor actividad, el algoritmo de revisita puede mantener un buen rendimiento.

La tabla 6, mostrada a continuación, representa la historia asociada al algoritmo de revisita.

Historia	
Desarrollador	Marvin Bernal
Nombre	Algoritmo de Revisita
Sección	Extracción
Descripción	<p>Algoritmo para la extracción de nuevas revisiones de los artículos almacenados. Calcula la esperanza matemática de cada artículo en base a la función probabilística exponencial, para estimar cuando un artículo tenga una o mas revisiones pendientes no almacenadas en la base de datos.</p> <p>Para el cálculo de la esperanza se toma el 10% más reciente de revisiones, y el tiempo transcurrido entre estas. Con estos datos, se calcula la esperanza y se verifica si el tiempo desde la última revisión es mayor a la misma. En caso positivo, el artículo es encolado para su extracción.</p>
Observaciones	Se ejecuta en segundo plano por medio de un cronjob diariamente.

Tabla 6: *Algoritmo de revisita de revisiones de artículos wiki*

3.5. ALMACENAMIENTO

Como fue mencionado en el Capítulo 2, la base de datos utilizada para almacenar las revisiones y artículos wiki es MongoDB. A través del proceso de extracción se almacenan dos colecciones: **revisions**, que representa las revisiones o historial de modificaciones del artículo wiki; y **articles**, que se refiere a los artículos de dichas revisiones. La arquitectura elegida para esta solución contiene la cantidad de elementos mínimos para implementar un cluster de base de datos en MongoDB, y cuenta con: dos sets de réplicas de shards, un set de réplicas para el servidor de configuración, y por último, un servidor de consultas.

La tabla 7 representa la historia asociada al almacenamiento de los datos en MongoDB.

Historia	
Desarrollador	Francisco Delgado
Nombre	Almacenamiento de Revisiones

Tabla 7 – continuación de la página anterior

Historia	
Sección	Base de Datos
Descripción	Conjunto de funciones para la conexión, consulta e inserción de datos en la base de datos (MongoDB). Realizada con el lenguaje Python y definidas dentro de una clase, se encarga de brindar una capa de interacción entre MongoDB y todas las operaciones de lectura/escritura de revisiones de un artículo wiki.
Observaciones	Las operaciones de escritura detectan si la revisión a insertar es un duplicado por medio del identificador único de la revisión (revid), y posteriormente procede a actualizar los datos ya existentes con los de la nueva entrada. Se genera una nueva conexión con la base de datos por cada operación que se requiera ejecutar en vez de utilizar una o más conexiones en común para llevar a cabo las tareas.

Tabla 7: Almacenamiento de revisiones de artículos wiki

Las colecciones están distribuidas entre dos fragmentos o shards que contienen un subgrupo de los datos fragmentados en el cluster, y la unión de estos subgrupos conforman la totalidad de los datos almacenados por la aplicación. Esta separación de los datos puede ser apreciada en la figura 25.

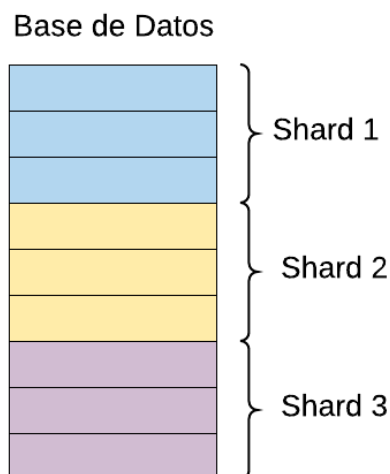


Figura 25: División de la base de datos en subgrupos

Cada uno de estos subgrupos está siendo almacenado en un servidor físico diferente, garantizando el escalamiento horizontal, tal como lo demuestra la figura 26. Para la creación de estos grupos, el sharding se lleva a cabo a través del Hashed Sharding, de esta manera es posible alcanzar una distribución homogénea de los datos entre los servidores.

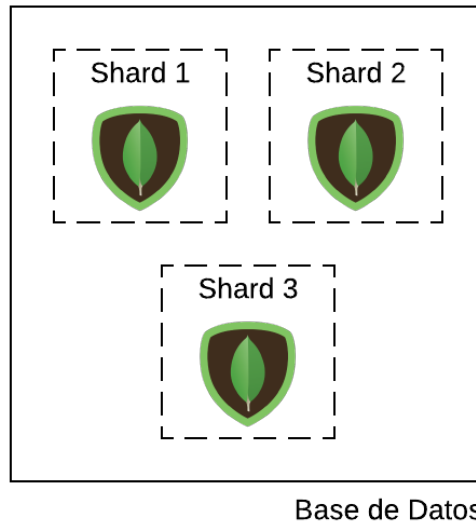


Figura 26: *Shards de la base de datos distribuidos físicamente*

Para poder crear un shard es necesario que este pertenezca a un set de réplica de un mínimo de tres miembros, por lo tanto, por cada shard se tienen tres servidores físicos que permiten la replicación de ese subgrupo de datos, garantizando así la alta disponibilidad de los mismos. Para poder fragmentar la base de datos, se necesita de un mínimo de dos shards, por lo que se tiene un total mínimo de 6 servidores que solo cumplen las labores de fragmentación y replicación.

Para poder coordinar las funciones de estos shards, como la autenticación, almacenamiento de los meta datos y consultas sobre el cluster de shards, es necesario hacer uso del servicio de un servidor de configuración. Este servidor, al igual que los shards, debe pertenecer a un grupo de réplicas, sumando tres servidores físicos más, que además, tienen la capacidad de recuperarse en caso de fallos, y nuevamente, permitir la alta disponibilidad.

Finalmente, se tiene un último servidor de base de datos cuya única labor es la ejecución de operaciones de lectura y escritura sobre la base de

datos. Este servidor es denominado servidor de consultas y no es necesaria su replicación. Las aplicaciones clientes que deseen realizar alguna consulta sobre la base de datos, realizan sus peticiones únicamente a este servidor. Así mismo, una vez es recibida la petición de consulta, este se comunica con servidor de configuración para que este le brinde apoyo a la hora de ejecutar las operaciones. Con este último servidor, el conteo total de servidores de MongoDB para esta arquitectura es de diez nodos.

La figura 27, que se muestra a continuación, demuestra el diagrama de la arquitectura del cluster de base de datos implementado:

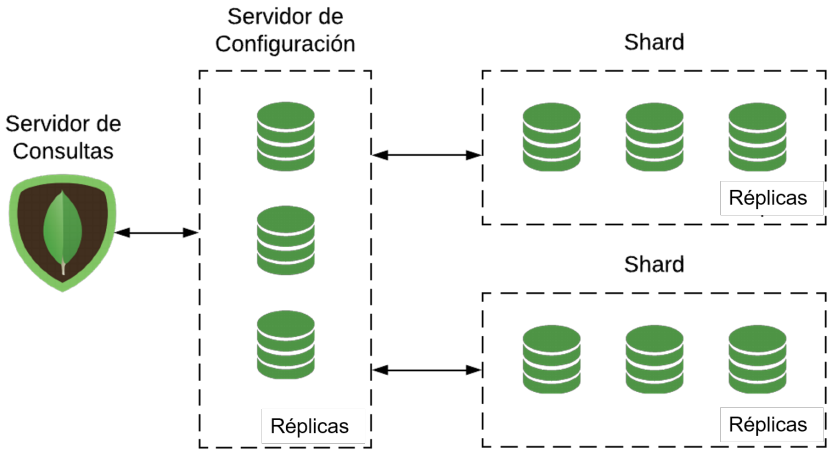


Figura 27: *Arquitectura del cluster de base de datos*

En tabla 8 se puede apreciar la historia asociada a la configuración del cluster de MongoDB.

Historia	
Desarrollador	Francisco Delgado
Nombre	Configuración de Shards y Réplicas
Sección	Base de Datos

Tabla 8 – continuación de la página anterior

Historia	
Descripción	<p>Configuración de los diversos nodos que incluidos dentro del cluster de la base de datos. Se configuran dos servidores para la fragmentación de datos por medio de Hashed Sharding. Cada uno de ellos forma parte de un grupo de réplicas, por lo que se hace uso de tres servidores por nodo de fragmentación.</p> <p>Se elige un nodo principal por cada grupo de réplica, en este nodo se inicializa la configuración del grupo por medio del comando <code>rs.initiate()</code>. Posteriormente se agregan los dos miembros restantes del grupo usando <code>rs.add('mongors1n2:27018')</code> y <code>rs.add('mongors1n3:27018')</code>. Luego, con estos grupos de réplicas ya creados, se generan los shards con el siguiente comando: <code>sh.addShard('mongors1/mongors1n1:27018')</code>. Y, finalmente, se habilita el sharding por medio del comando <code>sh.enableSharding('wiki_history_extractor')</code></p> <p>Una vez las réplicas y shards, han sido configurados, se genera una réplica extra para el servidor de configuración, el cual ayuda a la coordinación y consulta de los shards.</p>
Observaciones	<p>Para los grupos de réplicas no fue implementado un algoritmo de selección de nodo maestro, por lo cual se hace uso del comportamiento por defecto de MongoDB para elegirlo.</p>

Tabla 8: *Configuración del cluster de MongoDB*

3.6. DOCKER

Para poder poner en funcionamiento esta aplicación distribuida y el cluster de la base de datos, se hace uso de la herramienta Docker. Docker permite levantar servicios que emulan un servidor físico, los cuales pueden ser conectados en una red para que puedan comunicarse y llevar a cabo todas las tareas de extracción y consultas.

Adicionalmente, permite generar ambientes tanto de producción como de

desarrollo, en este caso, se hace uso de la versión 17.12.0-ce en conjunto con Docker-Compose para facilitar el levantamiento de los servicios en ambos ambientes.

En la tabla 9, que se presenta a continuación, se puede apreciar la historia asociada al uso de docker para el levantamiento de la aplicación.

Historia	
Desarrollador	Francisco Delgado
Nombre	Automatización de levantamiento de la aplicación
Sección	Configuración de aplicación distribuida
Descripción	Configuración de las imágenes y contenedores de docker para el levantamiento de la aplicación, distribución de las tareas entre diversos nodos, y levantamiento del cluster de MongoDB.
Observaciones	<p>La configuración de los contenedores se realiza por medio del archivo <code>docker-compose.yml</code>. Se implementó un archivo por cada ambiente de desarrollo, los cuales son: Digital Ocean, el cual es un servicio de alojamiento web y sirve como ambiente de producción sin la inclusión de las réplicas de mongo. Desarrollo, para el ambiente para desarrollo continuo de la aplicación; y por ultimo, Réplica, el cual define la configuración completa del cluster de MongoDB y sirve como ambiente de producción.</p> <p>Para la aplicación, es asignada una Máquina Virtual en el Centro de Computación, la cual posee los requerimientos para instalar y configurar la aplicación remotamente. Una vez instalado y levantado el servicio en ese entorno, el mismo puede responder peticiones provenientes de la misma red.</p>

Tabla 9: Configuración del levantamiento de la aplicación por medio de Docker

En primera instancia se genera una imagen para la aplicación flask, los trabajadores de Celery y el resto de los servicios. El nombre de las imágenes está definido bajo el siguiente formato: `nombre:version`, y pueden ser generadas por medio de un archivo llamado Dockerfile, cuyo contenido puede ser apreciado a continuación:


```
# Dockerfile
FROM python:2.7-alpine

RUN mkdir /app
WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD python manage.py runserver --host 0.0.0.0 --port=80
```

Código Fuente 3: *Contenido del archivo Dockerfile para la imagen del API*

Para los servicios de Nginx, RabbitMQ y MongoDB se hace uso de imágenes provenientes del repositorio oficial llamado Docker Hub, las imágenes utilizadas son las siguientes: `rabbitmq:3.6.11`, `mongo:3.6.2` y `nginx:1.13.3`.

La configuración de los contenedores se realiza a través del archivo `docker-compose.yml`, en el cual se especifican las imágenes a utilizar por cada servicio, variables de entorno, nombre de la red, entre otros.

Las imágenes de RabbitMQ y MongoDB tienen la característica que permite utilizar variables de entorno para establecer el usuario y clave de acceso sin ningún tipo de configuración manual por parte del desarrollador, por lo tanto, las credenciales usadas para ambos servicios son agregadas en el archivo `docker-compose.yml` para restringir su acceso. Por ejemplo, para RabbitMQ las credenciales son las siguientes: `RABBITMQ_DEFAULT_USER=wiki` y `RABBITMQ_DEFAULT_PASS=wiki123`. Adicionalmente, se establecieron una serie reglas adicionales, como por ejemplo, declaración de volúmenes para mantener la persistencia de datos y la exposición de los puertos de RabbitMQ, MongoDB y Nginx; lo cual permite a las aplicaciones externas acceder a estos servicios, y en el caso de Nginx, permite que otras aplicaciones puedan ejecutar peticiones al API.

Parte de la configuración de los servicios y contenedores de Docker puede ser apreciada a continuación:

```
version: '3'

services:

  rabbit:
    image: rabbitmq:3.6.11
    restart: always
    hostname: rabbit
    environment:
      - RABBITMQ_DEFAULT_USER=wiki
      - RABBITMQ_DEFAULT_PASS=wiki123
    ports:
      - "5673:5672"
    networks:
      - wiki_network
    volumes:
      - 'wiki_rabbit:/data'
```

Código Fuente 4: *Declaración de servicios con Docker Compose*

4. CONCLUSIONES

El análisis de los historiales de los artículos wiki basados en MediaWiki permite descubrir diversos indicadores y características que no pueden ser detalladas a simple vista. Para llevar a cabo estos análisis, es necesario almacenarlos en una base de datos. Sin embargo, debido a la gran cantidad de datos que puede envolver, resulta en una tarea complicada de ejecutar debido a las limitaciones de hardware de la máquina. Estas limitaciones pueden variar desde el espacio de almacenamiento hasta disponibilidad de servicios y recuperación de fallos, para los casos de las aplicaciones o servicios como Wiki-Metrics-UCV. Hoy en día, la necesidad de diseñar aplicaciones con una arquitectura distribuida ha incrementado, puesto que, en muchos casos, reduce o elimina estas limitantes a un menor costo económico que, por ejemplo, mejorar la capacidad individual de una sola máquina.

Durante el desarrollo de este Trabajo Especial de Grado fue posible llevar a cabo los objetivos planteados en la fase de investigación por medio de la metodología de desarrollo rápido de aplicaciones. Los objetivos fueron desarrollados, de forma progresiva, a través de la inclusión de nuevos módulos y funcionalidades al sistema en múltiples iteraciones de desarrollo. A su vez, esta metodología facilitó el proceso de retroalimentación de cada iteración, lo cual conllevó a un proceso más adaptable a la hora de cubrir las necesidades y problemas que surgieron continuamente.

Aunque la implementación de una arquitectura distribuida puede ser complicada, actualmente existen herramientas que facilitan esta tarea en gran medida. Tal es el caso del sistema de administración de bases de datos MongoDB, cuya gestión de datos permite implementar, de manera sencilla, aspectos como la replicación y fragmentación de la base de datos entre múltiples máquinas. Esta característica influye directamente en la habilidad de prestar un servicio de alta disponibilidad.

El uso de MongoDB, en conjunto con herramientas como Celery y RabbitMQ, proporcionan una gran ayuda a la hora de mitigar las limitaciones que conlleva el uso de un sistema centralizado, y el cumplimiento de los objetivos planteados para este Trabajo Especial de Grado.

Por medio de Celery es posible ejecutar múltiples tareas, como la extracción de métricas de los datos almacenados, evitando, por ejemplo, el colapso de un servicio web, e incluso, brinda de manera moderada la habilidad de recuperarse de fallos en tiempo de ejecución, puesto que gracias

a RabbitMQ, es posible llevar un registro de su progreso y de las tareas pendientes a ejecutar.

El desarrollo de un nuevo algoritmo para la actualización automática de las revisiones ya almacenadas, toma en cuenta nuevos parámetros que permiten una evaluación más adecuada de cada artículo. De esta forma se redujo la cantidad de peticiones innecesarias sobre el API de MediaWiki, incrementando la eficiencia del sistema.

La inclusión de Flask, para el desarrollo del API, permite a las aplicaciones de terceros consultar y extraer los historiales del artículo wiki deseado, además de brindar las herramientas necesarias para la ejecución de consultas más especializadas y la visualización de métricas previamente definidas ya incluidas en el sistema.

La arquitectura del sistema distribuido, descrito en este documento, brinda una solución escalable a los problemas previamente planteados, en donde la capacidad de un sistema centralizado es insuficiente ante la demanda progresiva de las aplicaciones de extracción, consulta y procesamiento de historiales de artículos wiki.

En caso de tener una gran cantidad de revisiones que extraer, los lineamientos de uso del API de MediaWiki limitan el uso de múltiples nodos para agilizar el proceso. A pesar de que existen múltiples nodos de extracción trabajando de forma paralela, puede que la duración del proceso de extracción se extienda demasiado, dado que existe 1 segundo de intervalo por cada página de revisiones.

Por otro lado, la ejecución de las tareas de Celery depende del servicio que provee RabbitMQ, el cual es centralizado. Por lo tanto, en caso de que este servicio presente fallas, el resto de los nodos involucrados en el sistema se paraliza.

Con respecto a la base de datos, dado que la misma se encuentra particionada entre diversos nodos separados geográficamente, el tiempo de respuesta para cada consulta puede ser elevado. Lo mismo se aplica para los diversos nodos que ejecutan el API, su ubicación en referencia al origen de las peticiones de consultas, afecta el tiempo de respuesta.

Tomando en cuenta estas limitaciones, se propone implementar nuevos algoritmos para la partición de datos entre los nodos que conforman las particiones de MongoDB, y así mejorar su distribución.

Adicionalmente, se propone la inclusión de las técnicas y diseño de bases de datos caché para mejorar la escalabilidad con nuevos nodos de menor capacidad de almacenamiento que atiendan la consulta de los datos más usados, y además, mejorar su disponibilidad y la velocidad con la que son servidos.

Finalmente, se recomienda la descentralización de RabbitMQ a través de un escalamiento horizontal que permita la tolerancia a fallos, y a su vez, incrementar la disponibilidad del servicio.

REFERENCIAS

- [1] cogNiTioN. *Cron*. 1990. URL: <http://www.unixgeeks.org/security/newbie/unix/cron-1.html>.
- [2] Ellen Gottesdiener. «RAD REALITIES: BEYOND THE HYPE TO HOW RAD REALLY WORKS». En: *Application Development Trends* (1996).
- [3] Wikipedia. *Wiki*. [En Línea]. 2001. URL: <https://en.wikipedia.org/wiki/Wiki>.
- [4] Wikipedia. *Wikipedians*. [En Línea]. 2001. URL: <https://en.wikipedia.org/wiki/Wikipedia:Wikipedians>.
- [5] S. Gilbert N. Lynch. «Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services». En: *ACM SIGACT News* (2002).
- [6] Wikipedia. *Statistics*. [En Línea]. 2004. URL: <https://en.wikipedia.org/wiki/Special:Statistics>.
- [7] Alberto Prieto Julio Ortega Mancia Anguita. *Arquitectura de computadores*. 2005. ISBN: 9788497322744.
- [8] John O'Hara J. «Queue - API Design». En: *ACM* 5.4 (2007), págs. 48-55.
- [9] Eric Lippert. *What is this thing you call "thread safe"*. 2009. URL: <https://blogs.msdn.microsoft.com/ericlippert/2009/10/19/what-is-this-thing-you-call-thread-safe/>.
- [10] B. Worwa R. D' Apuzzo. «Métodos Y Técnicas Para El Cálculo Y Visualización De Métricas De Historiales En Artículos De Wikis». En: 2012.
- [11] Wikipedia Foundation. *Terms of Use - Wikimedia Foundation*. 2014. URL: https://wikimediafoundation.org/wiki/Terms_of_Use.
- [12] MongoDB Inc. *MongoDB Architecture Guide*. 2017.
- [13] Wikipedia Foundation. *API:Etiquette - MediaWiki*. URL: <https://www.mediawiki.org/wiki/API:Etiquette>.
- [14] Docker Inc. *What is Docker?* URL: <https://www.docker.com/what-docker>.
- [15] MongoDB Inc. *PyMongo 3.5.1 documentation*. URL: <http://api.mongodb.org/python/current/tutorial.html>.

- [16] MongoDB Inc. *Replication*. URL: <https://docs.mongodb.com/manual/replication/>.
- [17] MongoDB Inc. *Sharding*. URL: <https://docs.mongodb.com/manual/sharding/>.
- [18] NGINX Inc. *NGINX REVERSE PROXY*. URL: <https://www.nginx.com/resources/admin-guide/reverse-proxy/>.
- [19] Pivotal Software Inc. *What can RabbitMQ do for you?* URL: <https://www.rabbitmq.com/features.html>.
- [20] Armin Ronacher. *Design Decisions in Flask*. URL: <http://flask.pocoo.org/docs/0.11/design/#design>.
- [21] Armin Ronacher. *Flask (A Python Microframework)*. URL: <http://flask.pocoo.org>.