

Tesis de Maestría

**SOLUCIÓN**

**PARA LA EXPLOTACIÓN EFICIENTE DE LOGS**

##### EN SISTEMAS DISTRIBUIDOS MEDIANTE TECNOLOGÍAS BIG DATA

FACULTAD DE INGENIERÍA

Lic. Rafael Urdaneta

Maestría en Tecnología de la Información

Director de Tesis

Prof. Dr. Nicolás D’Ippolito

Buenos Aires, Argentina 2018

**SOLUCIÓN**

**PARA LA EXPLOTACIÓN EFICIENTE DE LOGS**

**EN SISTEMAS DISTRIBUIDOS MEDIANTE TECNOLOGÍAS BIG DATA**

Lic. Rafael Urdaneta

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Director de Tesis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Presidente del Jurado

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jurado

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Jurado Externo

Defensa de la Tesis

Ciudad Autónoma de Buenos Aires, a los \_\_\_ días del mes de\_\_\_\_\_\_\_\_\_\_\_\_\_de \_\_\_\_\_\_\_\_.

*A mi esposa Nancy, por su apoyo incondicional en mi largo camino de aprendizaje. En los momentos de tribulación, tu acompañamiento y palabras de motivación me llenaron de fuerza para continuar adelante.*

**Reconocimientos**

Esta sección la estoy escribiendo luego de haber terminado la redacción de mi tesis. Decir mi tesis es una falacia. Es un trabajo en equipo, que incluye no solo a las personas que directamente me apoyaron desde el punto de vista técnico y académico, sino también a las que indirectamente, con sus palabras de aliento y su ejemplo sirvieron de combustible para transitar este camino lleno de aventuras, experiencias y aprendizaje.

Quiero extender mi agradecimiento y reconocimiento a mi tutor y profesor Nicolás

D’Ippolito, que con palabras precisas supo orientarme en cada momento del proceso. A Claudio Zamoszczyk, por la confianza y su apuesta firme en mi éxito. A Daniel Tokman y Adrian Nowik, que fueron dos pilares de ejemplo y constancia durante toda la maestría. A Gabriela Esperón, por su motivación a iniciar la maestría y ejemplo de superación.

Por último pero no menos importante, a Bautilio Lorenzo, por su confianza y ejemplo y a todos mis compañeros y excompañeros de labores en Argentina, México, Uruguay y Venezuela, por ser fuente infinita de aprendizaje y experiencia.

**RESUMEN DE LA TESIS**

Este trabajo propone una implementación práctica para el manejo y explotación adecuada de datos provenientes de los logs de las operaciones transaccionales de los sistemas que conforman el ciclo activo de una empresa siderúrgica de clase mundial.

Su desarrollo está enmarcado dentro de los objetivos del equipo de arquitectura de la empresa y busca resolver la necesidad de una herramienta informática que soporte el almacenamiento y explotación eficiente de grandes volúmenes de datos producidos por los logs de las aplicaciones que componen el sistema, permitiendo facilitar el diagnóstico ante fallas, suplir de información para análisis de tendencias de rendimiento de las aplicaciones, reducir los tiempos de respuesta de soporte y toma de decisiones; logrando así impactar de forma positiva agregando valor al proceso y reduciendo los tiempos y costos asociados.

Palabras claves: *big data*, arquitectura de software, mensajería, distribuida, procesamiento de logs, rendimiento, en linea.

**ABSTRACT**

This paper proposes a practical implementation for the handling of large volumes of information from the logs of the transactional operations of the active cycle of a steel company.

Its development is framed within the objectives of the architecture team of the company and seeks to solve the current need to have a computer architecture tool that supports the storage and efficient exploitation of large volumes of data produced by the logs of the applications that make up the system , Making it easier to diagnose faults, provide information for analysis of application performance trends, reduce support response times and make decisions; Thus achieving a positive impact by adding value to the process and reducing associated times and costs.

Key words: big data, software architecture, messaging, distributed, log processing, throughput, online.

**Contenido**

[1 Introducción 1](#_Toc52927)

[1.1 Objetivo general 1](#_Toc52928)

[1.2 Objetivos específicos 2](#_Toc52929)

[2 Fundamentos Teóricos 3](#_Toc52930)

[2.1 Arquitectura y diseño orientado a servicios (SOA) 3](#_Toc52931)

[2.1.1 Arquitectura Orientada a Servicios (SOA) 3](#_Toc52932)

[2.1.2 Aspectos Básicos 3](#_Toc52933)

[2.1.3 Colaboración entre Servicios 7](#_Toc52934)

[2.2 Big data 10](#_Toc52935)

[2.2.1 Propiedades del *Big Data* 12](#_Toc52936)

[2.2.2 Business Intelligence 15](#_Toc52937)

[2.3 Logs 15](#_Toc52938)

[2.4 Log en bases de datos 19](#_Toc52939)

[2.5 Logs en sistemas distribuidos 20](#_Toc52940)

[2.6 Bases de Datos NO SQL 21](#_Toc52941)

[2.6.1 Clave-Valor 23](#_Toc52942)

[2.6.2 Orientadas a Documentos 23](#_Toc52943)

[2.7 Procesamiento de Datos 25](#_Toc52944)

[2.7.1 MapReduce 25](#_Toc52945)

[2.7.2 Sistema de archivos Hadoop (*HDFS*) 28](#_Toc52946)

[2.8 Middleware orientado a mensajes (message-oriented middleware MOM) 29](#_Toc52947)

[2.9 Casos de uso típicos para un middleware orientado a mensajes 30](#_Toc52948)

[2.10 Beneficios generales de una arquitectura orientada a mensajes 33](#_Toc52949)

[2.11 Desventajas y desafíos en la implementación de arquitecturas orientadas a mensajes 36](#_Toc52950)

[2.12 Disponibilidad del Sistema 40](#_Toc52951)

[3 Caso de estudio 42](#_Toc52952)

[3.1 Análisis de la problemática actual 42](#_Toc52953)

[3.2 Arquitectura general del sistema actual 43](#_Toc52954)

[3.3 Sistema actual de generación y administración de los logs 45](#_Toc52955)

[3.4 Soluciones típicas para la recolección y explotación de logs 48](#_Toc52956)

[3.5 Propuesta de la solución general 56](#_Toc52957)

[4 Comparación y selección de herramientas big data para la solución 60](#_Toc52958)

[4.1 Capa de producción 60](#_Toc52959)

[4.2 Capa de recepción 63](#_Toc52960)

[4.2.1 Selección de un broker de mensajes 63](#_Toc52961)

[Estado del arte en arquitecturas orientadas a mensajes 63](#_Toc52962)

[Apache Kafka 66](#_Toc52963)

[Estructura General 67](#_Toc52964)

[Funcionalidades principales 70](#_Toc52965)

[RabbitMQ 73](#_Toc52966)

[Estructura general 75](#_Toc52967)

[Funcionalidades principales 78](#_Toc52968)

[Comparación de características generales entre Apache Kafka y RabbitMQ 83](#_Toc52969)

[Conclusiones preliminares y selección del broker 88](#_Toc52970)

[5 Construcción de prototipo de la solución 91](#_Toc52971)

[5.1 Capa de producción 91](#_Toc52972)

[5.2 Capa de transporte 95](#_Toc52973)

[5.3 Capa de explotación y consumo 103](#_Toc52974)

[5.4 Evaluación del modelo presentado 110](#_Toc52975)

[5.4.1 Análisis comparativo de costo de almacenamiento y procesamiento del modelo actual y el propuesto 110](#_Toc52976)

[5.4.2 Comparativa de tiempos para la obtención de información relevante en ambos modelos. 113](#_Toc52977)

[5.4.3 Análisis de valor para el área de soporte tecnológico 114](#_Toc52978)

[6 Conclusiones 116](#_Toc52979)

[6.1 Objetivos y mejoras conseguidas 116](#_Toc52980)

[6.2 Líneas futuras de investigación 117](#_Toc52981)

[6.3 Comentarios finales 118](#_Toc52982)

[7 Referencias Bibliográficas 119](#_Toc52983)

**LISTA DE TABLAS**

Tabla 1 Relación costo en disco para soportar logs productivos........................................................45

Tabla 2: Proyección de costo de storage anual para 14 aplicaciones.................................................45

Tabla 3: Evaluación de atributos clave de Apache Kafka y RabbitMQ.............................................83

**LISTA DE FIGURAS**

Figura 1: Modelo de Diseño Orientado a Servicios SOA....................................................................4

Figura 2: Mapa de interfaces entre aplicativos.....................................................................................5

Figura 3: Esquema básico de un Enterprise Service Bus (ESB)..........................................................6

Figura 4: Diagrama de Componentes SOA..........................................................................................8

Figura 5: Roles y actores en una arquitectura SOA.............................................................................9

Figura 6: Las 3 V de bigData..............................................................................................................13

Figura 7: Abstracción de secuencialidad de un log............................................................................15

Figura 8: Ejemplo de estructura básica de un log file........................................................................17

Figura 9: Referencia entre el código y el log de eventos....................................................................17

Figura 10: Ejemplo Tupla base de datos orientada a documentos......................................................23

Figura 11: Flujo de ejecución del map reduce....................................................................................25

Figura 12: Visión de un Middleware orientado a mensajes................................................................28

Figura 13: Comunicación bidireccional asincrónica entre procesos..................................................30

Figura 14: Ejemplo de duplicación de datos......................................................................................35

Figura 15: Diagrama del área de Sistemas.........................................................................................42

Figura 16: Figura 2: Diagrama de Módulos que componen el NCA.................................................43

Figura 17: Diseño arquitectura 3 capas..............................................................................................43

Figura 18: Proyección de almacenamiento necesario para 5 años de información histórica.............46

Figura 19: Patrón básico productor-consumidor................................................................................56

Figura 20: Pruebas de Performance Log4j2 Async Logging..............................................................60

Figura 21: Performance por Threads medidos en mensajes por segundo..........................................61 Figura 22: Diagrama de una arquitectura global de Apache Kafka (con un tópico, una partición y un factor de replicación 4).......................................................................................................................65 Figura 23: Arquitectura de un Tópico de Kafka.................................................................................67

Figura 24: Diagrama de balance de carga y alta disponibilidad en Kafka.........................................70

Figura 25: Diagrama simplificado de la arquitectura de RabbitMQ..................................................73

Figura 26: Ejemplo de intercambio directo (routing key = queues names = colours).......................75

Figura 27: Ejemplo de exchange de tipo fan-out................................................................................77

Figura 28: RabbitMQ Cluster con configuración de espejo de todas las colas activada....................81

Figura 29: Medición de Latencia Apache Kafka Sincrónico..............................................................85

Figura 31: Comparativa de latencia en mensajes de 1MB.................................................................85 Figura 30: Medición de Latencia RabbitMQ con Persistencia...........................................................85

Figura 31: Consumo de CPU Apache Kafka en modalidad sincrónica y asincrónica........................86

Figura 32: Comparativa de consumo de CPU RabbitMQ con persistencia y sin persistencia activada

............................................................................................................................................................87

Figura 33: Resumen de resultados de performance............................................................................87

Figura 34: Diagrama simplificado de las 3 capas funcionales de la propuesta..................................91

Figura 35: Configuración de Log4j v2.1 en el pom del proyecto......................................................92

Figura 36: Validación de librerías de log4j2 correctamente importadas al proyecto.........................93

Figura 37: Configuración de Appender de Log4j2 para RabbitMQ...................................................94

Figura 38: Iniciación del logger configurado para RabbitMQ...........................................................94

Figura 39: Código de prueba para simular el logueo de eventos de error del sistema.......................95

Figura 40: Diagrama Simplificado, enfoque en Capa de Transporte.................................................96

Figura 41: Acceso principal a la consola de administración de RabbitMQ........................................98

Figura 42: Sección principal consola de Administración RabbitMQ.................................................98

Figura 43: Creación de Exchange en RabbitMQ................................................................................99

Figura 44: Creación de Cola en RabbitMQ........................................................................................99

Figura 45: Bindeo de cola a Exchange de RabbitMQ......................................................................100

Figura 46: Colas bindeadas al exchange de RabbitMQ....................................................................100

Figura 47: Método Java de Prueba para envío de mensajes al broker mediante appender log4j2...101

Figura 48: Estadísticas de recepción de mensajes en RabbitMQ.....................................................102

Figura 49: Consulta de mensajes desde la consola de administración de RabbitMQ......................103

Figura 50: Capa de Consumo en el diagrama de componentes........................................................104

Figura 51: Creación de base de datos y colección en MongoDB.....................................................105

Figura 52: Vista de bases de datos en MongoDB.............................................................................106

Figura 53: Código fuente cliente MongoDB y ejecución de logs para envío al broker y a la bbdd.107

Figura 54: Registros provenientes del broker persistidos en MongoDB..........................................107

Figura 55: Diagrama de flujo de análisis de eventos en el SAS ESP...............................................109

Figura 56: Notificación automática al cliente telegram Desktop/Mobile vía bot.............................109 Figura 57: Estadísticas de consumo de storage para 150000 mensajes persistidos en mongoDB...110 Figura 58: Comparativa de consumo de disco (megaBytes) entre log plano y log persistido en base

NoSql MongoDB (150.000 eventos)................................................................................................111 Figura 59: Comparativa de estimado de almacenamiento para 5 años.............................................112

Figura 60: Ejemplo de consulta de un evento en MongoDB............................................................114

Figura 61: Diagrama integral del modelo propuesto........................................................................116

**LISTA DE SIGLAS**

|  |  |
| --- | --- |
| **Sigla** | **Significado** |
| ACID | Atomicidad, Consistencia, Aislamiento y Durabilidad. |
| AM | Área Mánager |
| BI | Bussiness Analytics |
| ESB | Enterprise Service Bus |
| NCA | Nuevo Ciclo Activo |
| OLTP | OnLine Transaction Processing |
| SOA | Service Oriented Architecture |
| SOC | Service Oriented Computing |
| SQL | Structured Query Language |
| TI | Tecnología en Información |
| API | Application Programming Interface |
| IoT | Internet de las cosas |
| IoE | Internet de todo |

# Introducción

En un estudio reciente de Cisco sobre internet de las cosas, se afirma que “la información de calidad es clave para el éxito” (Cisco, 2015b), ¿podemos afirmar categóricamente que hoy en día estamos actuamos sobre esa creencia? La mayoría de los gerentes, directores generales e incluso las áreas de soporte técnico de las compañías se han ocupado de la frustración de saber que tienen los datos dentro de la empresa, pero no pueden acceder a ella cuando realmente las necesitan. En ciertos contextos es difícil explicar por qué, a la luz de las costosas inversiones de la empresa en tecnología e infraestructura, los datos son de mala calidad o inaccesible. Las empresas reconocen la necesidad de información de calidad y muchos se esfuerzan por satisfacerla. Con demasiada frecuencia, sin embargo, los resultados son decepcionantes.

A menudo se subestima el poder y el valor que tienen los logs, y se tratan simplemente como un subproducto desechable; algunas personas se centran en los sistemas o los eventos transaccionales que producen de cara al cliente porque es lo que genera valor inmediato, pero ante fallas y errores del sistema, cuando es necesario investigar de forma seria utilizando los mecanismos y técnicas forenses informáticas, es cuando realmente nos damos cuenta que los logs deben ser tratarlos como información con alto valor y gran potencial de valor para el negocio.

## Objetivo general

Implementar una solución empleando tecnologías y herramientas de *big data*, que permita la recolección y explotación eficiente de alto volumen de información generada a partir de los logs de las aplicaciones de negocio, para incrementar la velocidad de análisis ante pérdida de performance y fallas del sistema.

## Objetivos específicos

* Obtener mayor información de las aplicaciones para obtener una ventaja competitiva en las áreas de soporte técnico y análisis de performance de las aplicaciones que componen el sistema.
* Exponer los beneficios que ofrecen las herramientas y tecnologías *big data* para el manejo de altos volúmenes de información, variedad de formatos y velocidad de procesamiento.
* Presentar los criterios de factibilidad, que justifican la adopción e implementación de este nuevo modelo de gestión de la información generada por las aplicaciones transaccionales.
* Unificar la metodología para el logueo de las aplicaciones involucradas mediante componentes comunes de su propia Arquitectura.
* Soportar el almacenamiento y explotación de información transaccional histórica de 5 años.
* Servir de proyecto habilitador para sistemas que promuevan la generación de conocimiento mediante la reutilización de la información y técnicas de analytics.

# Fundamentos Teóricos

## Arquitectura y diseño orientado a servicios (SOA)

### Arquitectura Orientada a Servicios (SOA)

El acrónimo SOA proviene del inglés *Service Oriented Architecture*. Se trata de un modelo de arquitectura que caracteriza el procedimiento para crear y usar los diversos procesos, reunidos en forma de servicios, que configuran un determinado Proceso de Negocio (Un proceso de negocio se puede ver como un conjunto estructurado de tareas, que contribuyen colectivamente a lograr los objetivos de una organización) (Oposiciones TIC, 2012)

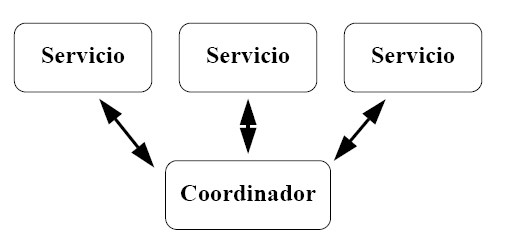
### Aspectos Básicos

Esta arquitectura define y proporciona la infraestructura necesaria para que el intercambio de información y la participación en los procesos de negocio se lleve a cabo con total independencia de la plataforma *hardware-software* sobre la que trabajan: sistema operativo, lenguaje de programación, características de los equipos, etc.

SOA es un modelo de componentes que interrelaciona las diferentes unidades funcionales de una aplicación, llamadas servicios, a través de interfaces bien definidas entre dichos servicios, ver figura 1.

***Figura 1****: Modelo de Diseño Orientado a*

*Servicios SOA*

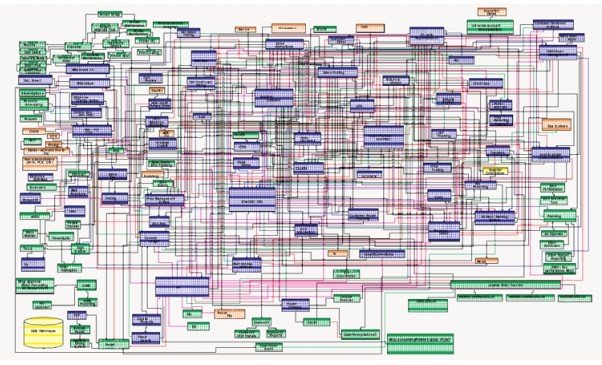


*Fuente: https://oposicionestic.blogspot.com.ar/ 2012/08/arquitectura-soa-orientadaservicios.html*

Las interfaces se definen de una manera neutral, independiente de la plataforma de *hardware*, sistema operativo, o lenguaje de programación en el que el servicio se implementa. Esto permite que los servicios, construidos sobre una gran variedad de tecnologías, puedan interactuar unos con otros de una manera uniforme y universal (Bolo, 2007).

En un SOA los diferentes servicios habitualmente no interactúan en forma directa unos con otros sino que lo hacen utilizando la mediación de un *Enterprise Service Bus[[1]](#footnote-1)*(ESB). Lo que se desea evitar es una situación tal como la que muestra la siguiente figura, que es el mapa de las interfaces entre aplicativos tomado de un caso real (una empresa de productos electrónicos de consumo masivo).

***Figura******2****: Mapa de interfaces entre aplicativos*

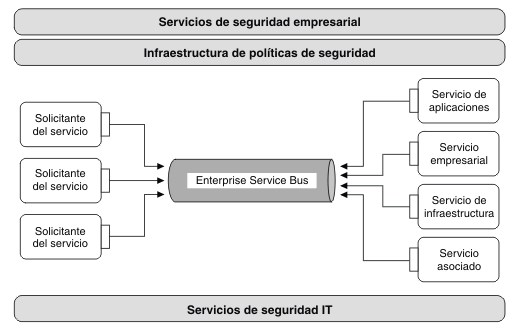


*Fuente: recuperado del sitio web*

*https://www.ibm.com/support/knowledgecenter/es/SS6RBX\_11.4.2/com.ibm.sa\_base.l egal.doc/topics/rsysarch\_overview\_base.html*

Un ESB es un *backbone[[2]](#footnote-2)* de integración, al cual se conectan los diferentes servicios y a través del cual fluyen los mensajes que permiten que aquellos interactúen, de modo que la arquitectura de integración de los servicios pasa a convertirse en algo como lo que se ilustra en la siguiente imágen:

***Figura 3:*** *Esquema básico de un Enterprise Service Bus (ESB)*



*fuente:https://www.ibm.com/support/knowledgecenter/es/SSAVUV\_7.*

*5.0/com.ibm.websphere.wbpm.scenarios.esb1.doc/topics/ cwesb\_security.html*

Esta característica de disponer de una definición neutral de las interfaces, no fuertemente ligada a ninguna implementación particular, se conoce como acoplamiento débil entre servicios. Los beneficios de un sistema débilmente acoplado consisten en su agilidad y capacidad para sobrevivir a cambios evolutivos en la estructura e implementación de cada uno de los servicios individuales que constituyen la aplicación en su totalidad. El acoplamiento fuerte, por otra parte, significa que los diferentes componentes de una aplicación están íntimamente relacionados en funcionalidad y forma, lo cual hace a estas aplicaciones muy vulnerables a los cambios evolutivos, ya que cualquier modificación a uno de sus componentes termina afectando a los demás.

### Colaboración entre Servicios

La computación orientada a servicios (SOC), es el paradigma computacional que utiliza servicios como elementos fundamentales para el desarrollo de aplicaciones y soluciones. Para construir el modelo de servicio, SOC se basa en la arquitectura orientada en servicios (SOA), que es una forma de reorganizar las aplicaciones de software y la infraestructura dentro de una colección de servicios que interactúan entre sí (Papazoglou, 2003).

La arquitectura planteada en el paradigma SOA hace un fuerte énfasis en los servicios como principal componente, el cual es usado para implementar funcionalidades propias y no propias del negocio. Otra de sus principales características, es que su arquitectura es distribuida, lo que significa que los servicios que lo componen son accesados remotamente a través de cierto tipo de protocolo para acceso remoto, por ejemplo, el *Representational State*

*Transfer* (REST)(Richards, 2006), el *Simple Object Access Protocol* (SOAP)(Rouse, 2016),

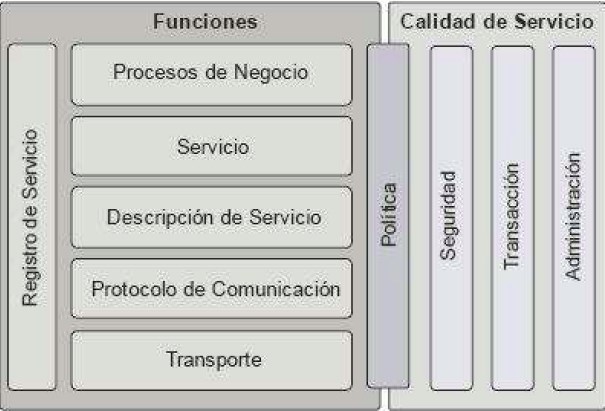
*Advanced Message Queuing Protocol* (AMQP)(Pivotal Software Inc., 2017), *Java Message*

*Service* (JMS)(Monson-Haefel & Chappell, 2001), *Microsoft Message Queuing* (MSMQ)

(Zhang & Geng, 2009)(Chun & Cho, 2006), *Remote Method Invocation* (RMI)(Grosso &

Reilly, 2001) o el *.NET Remoting(Obermeyer & Hawkins, 2001)*.

***Figura 4****: Diagrama de Componentes SOA*

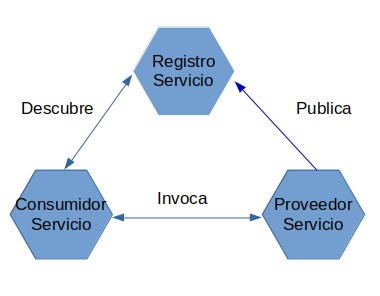
*fuente:https://oposicionestic.blogspot.com.ar/2012/08/ar quitectura-soa-orientada-servicios.html*

Las arquitecturas distribuidas ofrecen significantes ventajas sobre las arquitecturas monolíticas y las basadas en capas, que incluyen mejor escalabilidad, mejor desacoplamiento y mejor control sobre el proceso de despliegue(Mark Richards, 2015).

Los componentes dentro de una arquitectura distribuida tiende a ser auto-contenidas, permitiendo un mejor control de cambios y facilidad de mantenimiento, lo cual lleva a aplicaciones más robustas y performantes. Las arquitecturas distribuidas también tienden reducir el acoplamiento y promover la modularización.

En el contexto de arquitectura basada en servicio, la modularización es la práctica de encapsular porciones de nuestra aplicación dentro de servicios autocontenidos que pueden ser individualmente diseñados, desarrollados, probados y desplegados con poca o ninguna dependencia con otros componentes o servicios de la aplicación.

***Figura 5****: Roles y actores en una arquitectura SOA*

*fuente: adaptado de*

*https://oposicionestic.blogspot.com.ar/2012/08/ar quitectura-soa-orientada-servicios.html*

Las arquitecturas modulares también apoyan la idea de favorecer la reescritura sobre el mantenimiento, permitiendo que las arquitecturas sean refactorizadas o reemplazadas en piezas pequeñas a lo largo del tiempo a medida que crece el negocio en lugar de reemplazar o refactorizar una aplicación completa usando un enfoque de *Big Bang*.

## *Big data*

La habilidad de recolectar altos volúmenes de información, de distintos formatos y de forma rápida y eficiente ha impulsado el desarrollo de un nuevo concepto: *big data*.

*Big data* es definido como el conjunto de técnicas, nuevas tecnologías y arquitecturas que hacen posible extraer valor de altos volúmenes de datos(Katal, Wazid, & Goudar, 2013).

Gartner complementa esta definición(Sicular, 2013) agregando que las condiciones anteriormente presentadas: (variabilidad, volumen y velocidad) demandan de formas innovadoras de procesamiento rentable de información para mejorar la comprensión y la toma de decisiones, haciendo énfasis en dos aspectos del negocio: rentabilidad y valor a la toma de decisiones.

En términos generales podríamos referirnos como a la tendencia en el avance de la tecnología que ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones, la cual es utilizada para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a un base de datos relacional para su análisis(Ricardo Barranco Fragoso, 2012). De tal manera que, el concepto de *big data* aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales. Sin embargo, *big data* no se refiere a alguna cantidad en específico, ya que es usualmente utilizado cuando se habla en términos de petabytes y exabytes de datos.

Hasta hace pocos años, los grandes volúmenes de datos eran un serio problema. Cuando a principios de los años 2000 los volúmenes comenzaron a dispararse, las tecnologías de almacenamiento y CPU se vieron abrumadas por los numerosos terabytes de grandes datos, hasta el punto de que la TI enfrentó una crisis de escalabilidad de datos (Russom, 2011). Luego fuimos nuevamente impactados por la ley de Moore. Los medios de almacenamiento y CPU no sólo desarrollaron mayor capacidad, velocidad e inteligencia; también cayeron en precio. Las empresas pasaron de ser incapaces de permitirse o administrar grandes datos a gastar presupuestos en su recopilación y análisis.

Obviamente el volumen de datos es el principal atributo de *big data*. Con esto en mente, muchas definiciones de *big data* se establecen en terabytes, algunas veces en petabytes, sin embargo, también puede ser cuantificado mediante la cantidad de transacciones, registros, tabla o archivos. Otras organizaciones cuantifican *big data* en términos de tiempo, por ejemplo: debido a la prescripción de siete años en los Estados unidos, muchas empresas prefieren mantener siete años de datos disponibles para el riesgo, cumplimiento y el análisis legal.

### Propiedades del *Big Data*

*Big data* no solo se trata de alto volumen de información, sino también de diversos formatos, de velocidad de transferencia y de diversidad de frecuencias de entrega.

Si realmente queremos saber lo que sucede en nuestro negocio, necesitaremos grandes volúmenes de datos altamente detallados. Si realmente queremos ver algo que nunca hemos visto antes, ayudaría mucho el aprovechar los datos que nunca se han aprovechado para la inteligencia de negocios o Analytics(*BA[[3]](#footnote-3) Business Analytics* por su sigla en inglés). Algunos de los datos sin explotar pueden parecernos extraños, pueden provenir de sensores, dispositivos, sistemas terceros, aplicaciones Web y medios sociales. Algunas grandes fuentes de datos alimentan datos incesantemente en tiempo real. Poner todo eso en conjunto nos dará la visión de que bigdatano se trata sólo de volúmenes de datos gigantes; también se trata de una extraordinaria diversidad de tipos de datos, entregados a varias velocidades y distintas frecuencias (Russom, 2011).

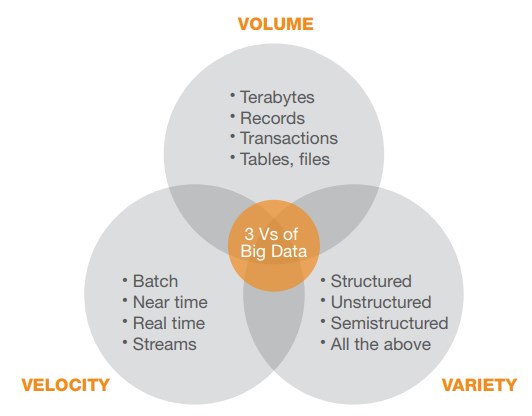
##### Volumen

El volumen en big data se refiere en términos simples, a la acumulación a gran escala de datos. Los volúmenes de información disponible actualmente suponen retos técnicos y analíticos singulares. Según las previsiones de Gartner, en 2020 más de 25 mil millones de dispositivos estarán conectados a Internet, según los pronósticos, a multiplicarse por 10 en tan solo 6 años, un crecimiento exponencial del volumen de datos contenidos en big data, muy por encima de los esfuerzos de BI más ambiciosos del pasado. Sin embargo, debemos ser conscientes de que la tecnología seguirá evolucionando con el tiempo, con lo cual, el tamaño de los *datasets* que hoy califican como big datatambién se incrementarán con el tiempo*.*

##### Velocidad

No solo representa la velocidad con la que los datos llegan, sino también la velocidad con la que estos se producen. Los sistemas tradicionales no son capaces de realizar análisis eficiente en datos que están en constante movimiento.

***Figura 6****: Las 3 V de bigData*

*fuente: https://www.researchgate.net/figure/The-3-Vsof-Big-Data\_fig1\_282287587*

##### Variedad

Los datos almacenados no necesariamente corresponde al mismo tipo o

categoría. Podemos clasificarlos por su estructura en:

* Estructurados: la cual está organizada por ejemplo en formatos y tecnologías que las hacen identificables. Por ejemplo a través de tablas que pueden ser consultadas mediante SQL[[4]](#footnote-4).
* Semi-estructurados: Son un tipo de datos que tienen una forma auto descriptiva, aun cuando no compartan una estructura formal como los de una base de datos, por ejemplo: *XML[[5]](#footnote-5).*
* No Estructurados: corresponde a datos con una estructura no identificable, por ejemplo, una imagen.

##### Variabilidad

La Variabilidad representa la inconsistencia del flujo de datos. El flujo de datos puede ser altamente incoherente, dando lugar a picos y mínimos periódicos. Las cargas de datos pico diarias, estacionales y provocadas por eventos pueden ser difíciles de gestionar, especialmente para datos no estructurados. Por ejemplo, los niveles transaccionales que se presentan durante los cierre de mes, las cargas por procesos automáticos masivos o los generados por desbordamiento durante fallas del sistema de forma no intencional aumentan el volumen de logs generados de forma abrupta y en cortos períodos de tiempo.

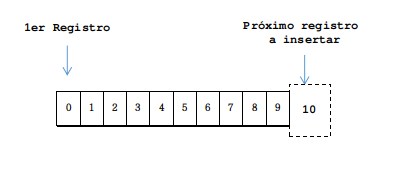
### Business Intelligence

Dos entidades técnicas se han unido: en primer lugar, hay grandes datos para cantidades masivas de información detallada. En segundo lugar, hay una analítica avanzada, que en realidad es una colección de diferentes tipos de herramientas, incluidas las basadas en análisis predictivo, minería de datos, estadísticas, inteligencia artificial, procesamiento del lenguaje natural, etc. Si los ponemos juntos obtendremos una gran analítica de datos, la nueva práctica más potente en *Business Intelligence*.

## Logs

Un log es la forma más simple de abstracción de un registro. Se trata de una agregación de secuencias de registros ordenados en el tiempo, tal como lo podemos apreciar en la siguiente imagen.

***Figura 7****: Abstracción de secuencialidad de un log*

*fuente: elaboración propia*

Los archivos de registro (*logs[[6]](#footnote-6)*), son archivos usados en modo escritura constante que registran información sobre eventos y acciones dentro de un sistema informático. Su función principal es la de registrar el qué y el cuándo de cada unos de los eventos. Son esenciales para trabajos forense digital, la detección de intrusiones y para probar el correcto funcionamiento de los ordenadores(Hartung, 2016).

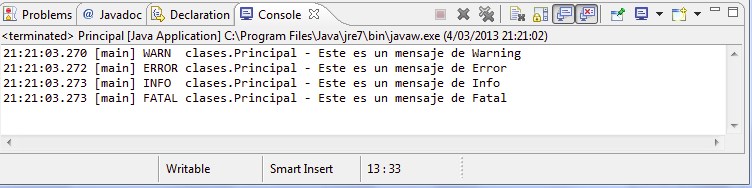
Los *logs* son la fuente primaria de información cuando se necesita determinar la operación pasada de un sistema de computación. Mantener archivos de registro correctos y precisos es importante para la investigación forense después de los hechos, así como para la administración, mantenimiento y auditoría del sistema. Sin embargo, su valor probatorio puede verse gravemente afectado si no está claro o si han sido manipulados. Por lo tanto, es imperativo proteger los archivos de registro de modificaciones no autorizadas(Department of Defense USA, 1985). Por derivación, el proceso de generación del log se le suele llamar logueo (en inglés *logging*) y al proceso o sistema que realiza la grabación en el log se le suele llamar *logger* o registrador.

Generalmente los eventos vienen registrados con:

* El momento exacto o data (fecha, hora, minuto, segundo) en el que ocurrió lo que permite analizar paso a paso la actividad.
* Una o más categorizaciones del evento registrado: usar categorías distintas para distinguir la importancia del evento estableciendo distintos niveles de registro los cuales suelen ser: depuración(*debug[[7]](#footnote-7)*), información, advertencia y error.
* La mayoría de los registros se almacenan en texto sin formato o en XML. De esta forma, el log puede ser fácilmente leído y procesado. Sin embargo, otras veces son construidos directamente por dispositivos usando campos magnéticos y pulsos electrónicos que pueden ser recolectados y analizados con herramientas y técnicas especiales(Bellare & Yee, 1997).

En la siguiente imagen se puede apreciar un ejemplo de la salida generada en un log de un programa en lenguaje Java con su estructura básica: la fecha y hora del evento, el método origen del evento, la clase, el nivel de criticidad del mensaje y el detalle del mensaje, en resumen: el qué y el cuándo.

***Figura 8****: Ejemplo de estructura básica de un log file*

*fuente: elaboración propia*

En estos ficheros los eventos vienen por orden cronológico de tal forma que la generación correcta de estos ficheros es grabar la información concatenando el nuevo contenido al final del contenido del fichero actual (Meneses, Bronevetsky, & Kalé, 2011). Típicamente se asigna una línea por evento, comenzando por el momento exacto o marca temporal (fecha, hora, minuto, segundo) en el que tuvo lugar, así como la línea de código asociada directamente al evento que se loguea, como se puede ver en la siguiente ilustración: ***Figura 9****: Referencia entre el código y el log de eventos fuente: elaboración propia*

## Log en bases de datos

El uso de los logs en las bases de datos tiene que ver con mantener sincronizados la variedad de estructuras de datos e índices en caso de falla. Para hacer esto atómico y persistible, una base de datos utiliza un log para escribir información sobre los registros que se van a modificar antes de aplicar los cambios a todas las estructuras de datos que mantiene(Kreps Jay LinkedIn, 2013).

El log es el registro de lo que sucedió y cada tabla o índice es una proyección de este historial en una estructura o índice de datos útil. Dado que el log se registra de inmediato, se utiliza como fuente autorizada en la restauración de todas las demás estructuras persistentes en caso de falla.

Con el paso del tiempo el uso del log creció de un detalle de la puesta en práctica de ACID[[8]](#footnote-8) a un método para replicar datos entre bases de datos(Xie et al., 2014). La secuencia de cambios que suceden en una base de datos es exactamente lo que se necesita para mantener una réplica de base de datos remota sincronizada. Oracle(Oracle, 2017b), MySQL(Oracle, 2017a) y PostgreSQL(“PostgreSQL: open source database”, 2017) incluyen protocolos de envío de registros para transmitir partes del registro a bases de datos de réplicas que actúan como esclavos. Oracle ha productizado el log como un mecanismo general de suscripción de datos para suscriptores de datos no Oracle con sus XStreams y GoldenGate(“Oracle GoldenGate”, 2017) e instalaciones similares en MySQL y PostgreSQL son componentes clave de muchas arquitecturas de datos.

Debido a esto, el concepto de un log legible por máquina se ha confinado en gran parte a los procesos internos de la base de datos. Esta abstracción ha sido ideal para soportar todo tipo de mensajería, flujo de datos y procesamiento de datos en tiempo real.

## Logs en sistemas distribuidos

Los dos problemas que un log resuelve: ordenar cambios y distribuir datos son aún más importantes en sistemas de datos distribuidos. El acuerdo (*commit[[9]](#footnote-9)*) sobre un pedido de actualizaciones (o de acuerdo en no estar de acuerdo(*rollback[[10]](#footnote-10)*) y hacer frente a los efectos secundarios) se encuentran entre los principales problemas de diseño para estos sistemas.

Algunos describen los usos de los logs de manera diferente. Los relacionados a base de datos generalmente diferencian entre el log físico y lógico. El log físico implica registrar el contenido de cada fila que se cambia. El log lógico implica registrar no las filas cambiadas, sino los comandos SQL que conducen a los cambios de fila (las instrucciones insert, update y delete).

En el caso de los archivos de logs, la persistencia de los eventos involucran ambos procesos, los lógicos y los físicos. Por tal motivo es necesario incorporar identificadores adicionales para segmentar las sesiones o transacciones.

## Bases de Datos NO SQL

El modelo relacional de datos fue propuesto en 1970 por Ted Codd como la mejor

solución para los problemas de DBMS[[11]](#footnote-11) del momento: procesamiento de datos empresariales. Los sistemas de base de datos relacionales iniciales incluyeron el sistema R2(Astrahan et al.,

1976), el Ingres(Stonebraker, 1976) y casi todas las implementaciones comerciales del

DBMS relacional (RDBMS) basan sus raíces en estos dos sistemas (Stonebraker & Cattell, 2011).

Definición de NoSQL: Son bases de datos de próxima generación que abordan principalmente algunos de los siguientes puntos: ser no relacional, distribuido, de código abierto y horizontalmente escalable. La intención original ha sido las modernizar las bases de datos a escala web. El movimiento comenzó a principios de 2009 y está creciendo rápidamente. A menudo más características se aplican, tales como: libre de esquemas, soporte para fácil replicación, API[[12]](#footnote-12) simple, eventualmente consistente / BASE (no ACID), un gran volumen de datos y mucho más. El término “nosql”puede resultar engañoso, por lo que la comunidad ahora lo traduce como "no sólo sql"(*not only sql* en inglés)(“NOSQL Databases”, 2011).

Entre otros aspectos, no necesariamente omnipresentes, que la mayoría de los sistemas NoSql comparten tenemos:

* Carecen de esquemas fijos.
* Evitan los j*oins* (Operación de combinar relaciones entre entidades).
* Escalamiento horizontal.

Por otro lado, los distintos sistemas NoSql buscan satisfacer diferentes necesidades. Algunos como el orientado a documentos aportan una inmensa facilidad de uso, mientras que la mayoría de los sistemas clave-valor u orientados a columnas, facilitan la distribución de datos sobre grupos de servidores (*Clusters[[13]](#footnote-13)*).

Para facilitar la comprensión de esta tecnología, describiremos a continuación algunos tipos de sistemas de almacenamiento de datos en función a su modelo:

### Clave-Valor

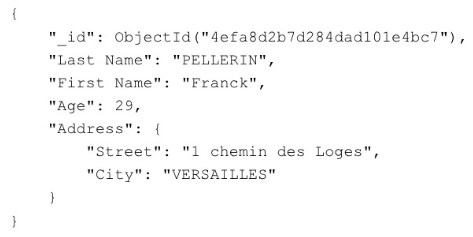
Esta clase de sistema posiblemente tenga el modelo de datos más sencillo. En lugar de tablas o documentos semiestructurados, los datos se organizan simplemente como una colección de entradas asociadas e identificadas por una clave única para identificar cada entrada. Cabe destacar que aunque tal mecanismo puede ser muy sencillo para el programador de aplicaciones, las implementaciones a menudo almacenan datos usando un modelo más tradicional detrás de las escenas. Mientras que el almacén de clave-valores se ve y actúa como una matriz asociativa, puede apoyarse en tablas, índices y otros artefactos de sistemas relacionales para ser eficiente en la práctica (Lith & Mattsson, 2010).

### Orientadas a Documentos

Las bases de datos documentales son sistemas de almacenamiento de datos semiestructurados, usualmente implementados sin tablas. La idea es permitirle a la aplicación cliente agregar y remover atributos de una tupla[[14]](#footnote-14) simple sin necesidad de perder espacio por la necesidad de crear registros con campos vacíos para las otras tuplas. Esto significa que todas las tuplas pueden contener cualquier número de campos, de cualquier longitud. Incluso si se utiliza de una manera estructurada, no hay restricciones o esquemas que limitan la base de datos para ajustarse a una estructura preestablecida. Esto significa que el programador de aplicaciones gana facilidad de uso y la posibilidad de crear datos muy dinámicos.

Una base de datos orientada a documentos se puede implementar como una capa sobre una base de datos relacional u objeto. Otra opción es implementarla directamente con un formato de archivo semiestructurado, como JSON(David Liedle, 2015), XML(Ri5, 2003) o YAML(Clark C. Evans, 2011). Conceptos tradicionales como los índices y claves se emplean a menudo en el mismo sentido que en bases de datos relacionales (Chang et al., 2006).

***Figura 10****: Ejemplo Tupla base de datos orientada a documentos*



*fuente: elaboración propia*

## Procesamiento de Datos

### MapReduce

Map reduce es un modelo programático que soporta procesamiento distribuido de grandes colecciones de datos. Fue presentado por primera vez en el paper *”MapReduce: Simplified Data Processing on Large Clusters”* por Jeffery Dean y Sanjey Ghemawatlos, ambos empleados de Google. (Dean & Ghemawat, 2004)

La función map(): se encarga del mapeo y es aplicada en paralelo para cada ítem en la entrada de datos. Esto produce una lista de pares (k2,v2) por cada llamada. Después de eso, el framework de MapReduce junta todos los pares con la misma clave de todas las listas y los agrupa, creando un grupo por cada una de las diferentes claves generadas. Desde el punto de vista arquitectural el nodo master toma el input, lo divide en pequeñas piezas o problemas de menor identidad, y los distribuye a los denominados worker nodes. Un worker node puede volver a sub-dividir, dando lugar a una estructura arbórea. El worker node procesa el problema y pasa la respuesta al nodo maestro.

Función map: *map (k1,v1) → list(k2,v2)*

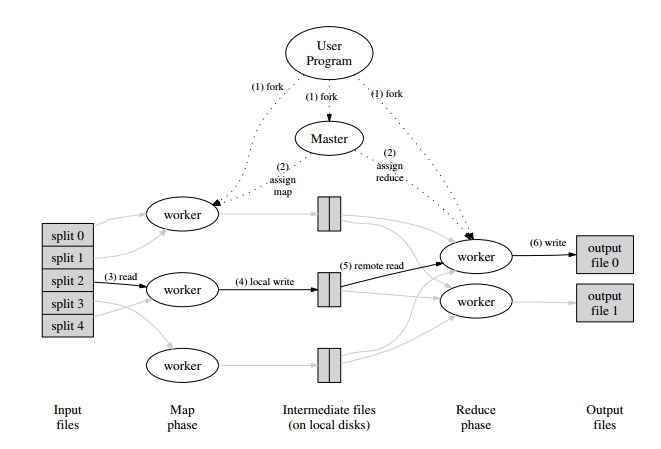
La función reduce(): cada llamada a Reduce típicamente produce un valor v3 o una llamada vacía, aunque una llamada puede retornar más de un valor. El retorno de todas esas llamadas se recoge como la lista de resultado deseado.

Por lo tanto, el framework MapReduce transforma una lista de pares (clave, valor) en una lista de valores.

Función reduce: *reduce (k2,list(v2)) → list(v2)*

La función map() se ejecuta de forma distribuida a lo largo de varias máquinas. Los datos de entrada, procedentes por regla general de un gran archivo (fichero), se dividen en un conjunto de M particiones de entrada de generalmente 16 a 64 megabytes. Estas particiones pueden ser procesadas en diversas máquinas. En una invocación de MapReduce suelen ocurrir varias operaciones:

Se procede a dividir las entradas en M particiones de tamaño aproximado de 16 a 64 megabytes. El programa MapReduce se comienza a instanciar en las diversas máquinas del cluster. Por regla general, el número de instancias se configura en las aplicaciones.

Una de las copias del programa es especial y toma el papel de "maestro". El resto de copias se denominan como "*workers*" y reciben la asignación de sus tareas desde el *master*. Se considera que existen una cantidad de M map() tareas y de R reduce(). El "maestro" se encarga de recopilar "workers" en reposo (es decir sin tarea asignada) y le asignará una tarea específica de map() o de reduce(). Un worker sólo puede tener tres estados: reposo, trabajando, completo. En la siguiente imagen se puede apreciar el flujo completo de la ejecución de los algoritmos (Chang et al., 2006). ***Figura 11****: Flujo de ejecución del map reduce fuente: https://gigadom.wordpress.com/category/map-reduce/*

Un worker que tenga asignada una tarea específica de map() tomará como entrada la partición que le corresponda. Se dedicará a parsear los pares (clave, valor) para crear una nueva pareja de salida, tal y como se específica en su programación. Los pares clave y valor producidos por la función map() se almacenan como buffer en la memoria.

Periódicamente, los pares clave-valor almacenados en el buffer se escriben en el disco local, repartidos en R regiones. Las regiones de estos pares clave-valor son pasados al master, que es responsable de redirigir a los "workers" que tienen tareas de reduce().

Cuando un worker de tipo reduce es notificado por el "maestro" con la localización de una partición, éste emplea llamadas remotas para hacer lecturas de la información almacenada en los discos duros de los diversos workers de tipo map(). Cuando un worker de tipo reduce() lee todos los datos intermedios, ordena las claves de tal modo que se agrupen los datos encontrados que poseen la misma clave. El ordenamiento es necesario debido a que, por regla general, muchas claves de funciones map() diversas pueden ir a una misma función reduce(). En aquellos casos en los que la cantidad de datos intermedios sean muy grandes, se suele emplear un ordenamiento externo.

El worker de tipo reduce() itera sobre el conjunto de valores ordenados intermedios, y lo hace por cada una de las claves únicas encontradas. Toma la clave y el conjunto de valores asociados a ella y se los pasa a la función reduce(). La salida de reduce() se añade al archivo (fichero) de salida de MapReduce (Dean & Ghemawat, 2004).

Cuando todas las tareas map() y reduce() se han completado, el "maestro" levanta al programa del usuario. Llegados a este punto la llamada MapReduce retorna el control al código de un usuario.

Se considera que ha habido un final de las tareas cuando este control se ha devuelto al usuario. Las salidas se distribuyen en un fichero completo, o en su defecto se reparten en R ficheros. Estos R ficheros pueden ser la entrada de otro MapReduce o puede ser procesado por cualquier otro programa que necesite estos datos.

### Sistema de archivos Hadoop (*HDFS[[15]](#footnote-15)*)

Es *framework[[16]](#footnote-16)* de código abierto utilizado para el almacenamiento distribuido y el procesamiento de grandes conjuntos de datos utilizando el modelo de programación *MapReduce(Dean & Ghemawat, 2004).*

Para entender cómo es posible procesar el mapReduce y ejecutarlo de forma distribuida en cientos (e incluso miles) de nodos, es necesario disponer del sistema de archivos distribuidos Hadoop (HDFS). Los datos de un cluster de Hadoop se desglosan en pedazos más pequeños (llamados bloques) y se distribuyen por todo el grupo(Zikopoulos & Eaton, 2011). De esta manera, las funciones map reduce se pueden ejecutar en subconjuntos más pequeños de sus conjuntos de datos más grandes, y esto proporciona la escalabilidad que se necesita para el procesamiento de datos grandes (*big data*).

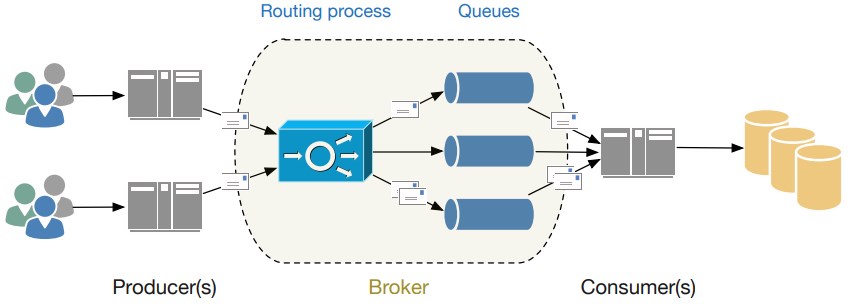
El objetivo de Hadoop es utilizar servidores comúnmente disponibles en un clúster muy grande, donde cada servidor tiene un conjunto de unidades de disco internas de bajo costo. Para un mayor rendimiento, MapReduce intenta asignar cargas de trabajo a estos servidores donde se almacenan los datos a procesar (IBM, 2017).

## Middleware orientado a mensajes (message-oriented middleware MOM)

Como su nombre lo indica, un middleware[[17]](#footnote-17) orientado a mensajes (o bus de datos) (Oracle, 2010), es una pieza de software que se ocupará de los mensajes, que son cualquier tipo de datos (por lo general, una cadena de bytes), envuelto en una estructura que cuenta con un encabezado muy simple. Los mensajes son enviados por uno o muchos procesos llamados

"productores" hacia el middleware orientado a mensajes que generalmente se llama "intermediario". El agente no procesará los mensajes (es decir nunca leerá su contenido) pero puede enrutarlos a ninguna, una o más colas de acuerdo con la información incluida en el encabezado de los mensajes.

***Figura 12****: Visión de un Middleware orientado a mensajes*



*fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

Las colas se almacenan localmente en el intermediario. Esos mensajes fueron enviados por los productores para ser procesados (por ejemplo, para realizar un cálculo, almacenar un archivo, insertar un registro en una base de datos).

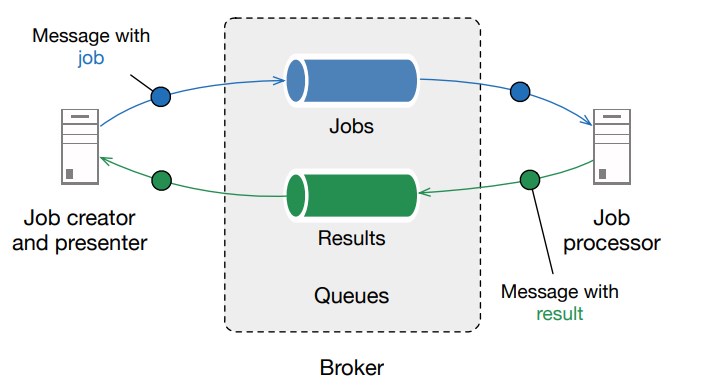
Los procesos responsables de abrir los mensajes y realizar acciones reales con su contenido son llamados "consumidores". Ellos (dependiendo del tipo de bus de datos utilizado) obtendrán mensajes enviados por el intermediario, o pueden buscar algunos de ellos consultando regularmente al broker. En el caso de uso más simple, los mensajes se descartan una vez procesados y los consumidores continúan con los siguientes.

## Casos de uso típicos para un middleware orientado a mensajes

Una diferencia entre una interacción directa proceso a proceso y una donde interviene un intermediario (broker) de mensajes, es que todas las operaciones son inherentemente asincrónicas, esto es porque cuando el productor envía un mensaje al broker no tiene garantía en absoluto con respecto al tiempo que transcurre entre la publicación del mensaje y su consumo. Adicional a lo anterior, (en casos típicos) no se notificará cuando la operación haya finalizado por completo.

Una vez producido el mensaje, su lectura no es una actividad postergable, sin embargo, procesarlo y persistirlo si lo puede ser (Nannoni, 2015). De hecho, si el valor insertado no es necesario(en el momento en que fue solicitada su creación y en el futuro cercano) por cualquier otro proceso, su inserción puede ser postergada y encolada sin ninguna implicación.

El uso más obvio de una operación de escritura diferida es cuando se escriben logs(Kreps Jay LinkedIn, 2013). Una entrada de log es una entidad standalone: ninguna otra entidad tiene un apuntador a un registro de log. Por otra parte, no es vital (en la mayoría de los ambientes productivos), que una aplicación publique sus logs en tiempo real. El único requerimiento es que la información deba quedar eventualmente persistida en alguna parte. Esto es lo que precisamente una arquitectura orientada a mensaje puede garantizar: que provisto un consumidor, un mensaje sea procesado, pero sin ninguna indicación del tiempo que pueda llevar el proceso. Otros casos de uso puede que podemos mencionar son: el envío de correos electrónicos (e-mails) o procesamiento paralelizado y computo distribuido, lo cual implica la comunicación bidireccional entre varios procesos asincrónicos tal como se puede observar en el siguiente diagrama.

***Figura 13****: Comunicación bidireccional asincrónica entre procesos fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

En este ejemplo, los jobs y los resultados son almacenados en colas en el mismo broker, lo cual no es un requerimiento mandatorio. Sin embargo, se debe considerar que todos los procesos se ejecutan de manera asincrónica y por lo tanto no hay garantía de que todos los jobs se procesen de manera exitosa. El productor del job podría implementar un mecanismo para cotejar el resultado recibido con los jobs enviados para presentar los resultados de manera correcta cuando eventualmente lleguen. Algunos brokers implementan mecanismos para identificar los mensajes con algún tipo de identificación única, pero esto no es parte de las funcionalidades por defecto de los brokers.

## Beneficios generales de una arquitectura orientada a mensajes

Sumado al principio de procesamiento asincrónico entre componentes, la arquitectura orientada a mensajes ofrece una serie de beneficios adicionales, tales como:

* **Flexibilidad y extensibilidad:**

El broker puede soportar más de un tipo de mensajes, puede tener una o más colas y si es necesario puede replicar mensajes a múltiples colas. Pero la funcionalidad más poderosa viene dada por dotar a la arquitectura del sistema de independencia entre los procesos de producción y consumo, especialmente considerando que cada unos de estos procesos normalmente están escritos en una amplia variedad de lenguajes de programación.

Al proveer a los productores y consumidores del mismo protocolo, de una misma estructura para definir el contenido de los mensajes, los procesos pueden comunicarse sin ningún problema entre ellos. Los productores y los consumidores pueden ser de varios tipos.

Múltiples consumidores pueden digerir mensajes desde una cola en un mecanismo tipo Round Robin(por ejemplo, solo un consumidor toma una copia de cada mensaje) o pueden por ejemplo realizar diferentes tareas con la misma entrada (todos reciben todos los mensajes).

* **Escalabilidad:** cualquier número de productores puede trabajar con cualquier cantidad de colas y cualquier número de consumidores pueden o no consumir simultáneamente los mensajes de estas. Esto hace que el sistema pueda escalar horizontalmente de una manera sencilla: entre más solicitudes HTTP reciban, más servidores de backend se pueden configurar, enviando la data al broker. Lo mismo se puede lograr con los consumidores, incrementando la capacidad de digerir los mensajes. El broker también puede configurarse en cluster, para lograr balanceo de carga y replicación. Sin embargo, las instancias con las que los productores y consumidores tendrían que intercambiar datos (por ejemplo, bases de datos, sistemas de almacenamiento de archivos) también podrían sobrecargarse. Por tal motivo, para implementar realmente una arquitectura de middleware de mensajería con la capacidad de escalar por completo, estos también deberían ser capaces de escalar.
* **Resistencia y tolerancia a fallas:** Estas arquitecturas deben ser robustas y resistentes ante eventuales fallas de software o de hardware. Una vez que el mensaje es enviado al broker, este puede tomar la precaución de replicarlo y almacenarlo en disco para prevenir cualquier falla que pueda causar que no pueda entregar el mensaje. Lo mismo aplica a los consumidores: si estos presentan un problema, pueden cerrar la conexión que mantienen con el broker, y esto detiene la entrega del mensaje hasta tanto se reanude el canal de comunicación, o hasta que otro consumidor retome la tarea. La notificación de la recepción del mensaje puede ser implementada del lado del consumidor, permitiéndole al broker saber cuando el mensaje ha sido efectivamente entregado y usar esta información cuando la entrega haya fallado del lado del consumidor.
* **Gestión de procesamiento en lote (batch)** Los middleware orientados a mensajes están diseñados para soportar una capacidad de producción muy alta, de modo que puedan tener un rol de amortiguador. Esta función de amortiguación otorga libertad a los arquitectos de sistemas del lado del consumidor, es decir, se pueden configurar diferentes consumidores, consumir el mismo conjunto de mensajes (todos reciben todos los mensajes), pero posiblemente no al mismo ritmo, o al mismo tiempo. Proporciona una excelente manera de transformar costosas operaciones independientes (como inserciones de bases de datos) en operaciones por lotes, esperando que lleguen suficientes mensajes antes de enviarlos a una base de datos. Además, si un proceso es más lento que otro (por ejemplo, un proceso agrega los datos en una instancia de MongoDB, mientras que el otro lo hace en un clúster de Hadoop), no interferirán ni se obstaculizarán entre sí y funcionarán lo mejor que puedan para cada caso. Esto implica en consecuencia una reducción de posibles sobrecargas en los sistemas consumidores y la oportunidad de aprovechamiento de sus recursos físicos.
* **Aislamiento de procesos** Las arquitecturas orientadas a mensajes imponen una separación de tareas de facto. Los procesos que no dependen del tiempo, son críticos o deben ser replicados, se manejan mejor fuera del servidor de aplicaciones principal.

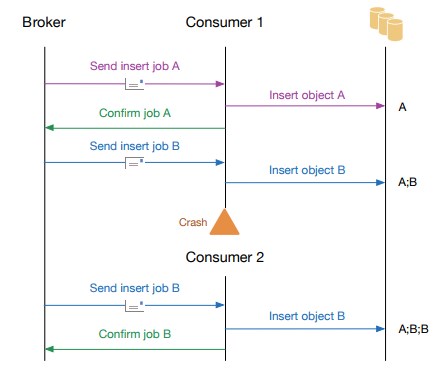
Los pequeños procesos dedicados se encargan de las tareas de forma independiente, mientras que el broker actúa como un sistema de colas de espera y planificador de tareas (*scheduling*[[18]](#footnote-18)).

* **Proceso simplificado de mantenimiento, administración y despliegue.** Junto con el aislamiento de los procesos y la resistencia que brinda, una arquitectura orientada a mensajes también es más fácil de administrar. En ambientes productivos para actividades de administración, mantenimiento o cambios de versión, las tareas internas en proceso pueden ser redirigidas hacia nuevos consumidores, que pueden apagarse y encenderse sin afectar al servidor de aplicaciones. Nuevos consumidores pueden ser instalados a la par de los ya existentes y activos sin necesidad de bajar el servicio, el broker actúa como una cola, sigue recibiendo y guardando los mensajes de los productores mientras espera por la activación de los nuevos procesos que consuman los mensajes.

## Desventajas y desafíos en la implementación de arquitecturas orientadas a mensajes

La idea básica detrás de las arquitecturas orientadas a mensajes no son complicadas de entender, sin embargo pueden presentar ciertas dificultades para implementarlas en el mundo real. Algunos de los desafíos que se pueden presentar son:

* **Multiplicación de instancias y aplicaciones a administrar y mantener.** Como una implicación directa del aislamiento que envuelven estas arquitecturas, el número total de máquinas, aplicaciones e instancias a administrar se incrementa. A diferencia de aplicaciones pequeñas, los beneficios de esta arquitectura compleja pueden verse mejor aprovechadas por aplicaciones empresariales grandes, con una carga de datos mediana a pesada y posiblemente una necesidad de replicación de datos, especialmente a la hora de agregar nuevas instancias para la ejecución de procesamiento de eventos.
* **Mensajes duplicados.** Los sistemas brokers que existen en el mercado pueden generar mensajes duplicados. Un mensaje que fue procesado por el consumidor y que no fue reportado al broker, bajo ciertas condiciones, podría causar que el mensaje pueda ser enviado nuevamente (ver Figura 14). Dependiendo de la tarea que deba cumplir, por ejemplo, persistencia en base de datos, podría generar falla de integridad por duplicidad de datos.

***Figura 14****: Ejemplo de duplicación de datos fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

Debemos tener en cuenta que esto puede suceder también entre los productores y el broker: dependiendo del sistema utilizado (ej: notificación de mensaje enviado a la cola o disponibilidad de la cola), los mensajes podrían perderse por problemas en la red y por lo tanto el mensaje podría ser enviado 2 veces o más a la cola.

* **Ordenamiento.** Dependiendo del sistema utilizado, se proporcionan diferentes garantías (si las hay) con respecto al orden en que los mensajes se almacenan en el lado del broker y como llegan a los consumidores. La importancia del orden debe tenerse en cuenta al diseñar el protocolo utilizado para intercambiar mensajes. Incluir números de secuencia puede ayudar a mitigar el problema, por ejemplo (pero puede surgir la cuestión de su sincronización entre todas las instancias). Como una remediación, se pueden aplazar las operaciones idempotentes (es decir, las operaciones que se pueden realizar en cualquier momento y aún así dar los mismos resultados) de los que no.
* **Duplicación de código o procesos:** Dependiendo del proceso diferido y de lo que se tiene que lograr, el aislamiento de los diferentes procesos que proporcionan las arquitecturas de mensajería puede conducir a la duplicación de código. Por ejemplo, posponer una inserción en una base de datos que debe realizar una consulta para registrar un atributo que corresponde a una clave foránea, antes de insertar datos en una tabla puede crear nuevos cuellos de botella en la base de datos, debido al número de lecturas simultáneas realizadas en la base de datos por los diferentes consumidores. Además, si los datos no están "listos para comer" en las colas en el broker, es probable que todos los consumidores diferentes que consumen el mismo conjunto de datos (todos reciban todos los datos para producir resultados diferentes) realicen las mismas consultas. Por lo tanto, conduce al desperdicio de recursos y la posible incoherencia del código entre los consumidores. Si los datos en las colas del broker no están listos para consumirse, es probable que todos los consumidores que consumen el mismo conjunto de datos (todos reciban todos los datos para producir resultados diferentes) realicen las mismas consultas, generando un desperdicio de recursos.
* **Mezcla de datos nuevos y viejos.** La flexibilidad en términos de la entrega y mantenimiento que la arquitectura orientada a mensajes ofrece, puede generar ciertos desafíos. Por ejemplo, asumamos que un mensaje que envuelve una entidad está listo para ser insertado en una base de datos, y que la cola está llena de esos mensajes aun sin un consumidor conectado. Si por algún motivo el modelo de la base de datos ha cambiado, (ej. se eliminó un campo), entonces la operación de inserción seguramente falle creando un problema de integridad. El código de los productores y consumidores deben ser cuidadosamente actualizados para mantener la consistencia de la data encolada. Un diseño de protocolo de mensajes inteligente pero simple puede ayudar a mitigar este tipo de problemas.
* **Demoras inducidas.** El objetivo principal de implementar un middleware orientado a mensajes es separar por completo las operaciones asincrónicas del proceso principal de la aplicación. Es probable que esto induzca un tiempo mayor antes de la finalización por completo de la tarea, ya que el número de pasos desde el inicio de la operación asincrónica en la aplicación principal hasta su ejecución completa es mayor. Además, no hay ninguna garantía de que un consumidor esté conectado y listo para procesar datos. Esta imprevisibilidad relativa puede ser un problema con operaciones cercanas a tiempo real. Para mitigar estos problemas de demora se puede considerar tener suficientes consumidores y asegurarse de que no se sobrecarguen ante la producción de alto volumen de mensajes.
* **Traslado de cuellos de botella en rendimiento.** Cuando se diseñan arquitecturas asíncronas, especialmente con sistemas de mensajería, un objetivo atractivo es asegurar el aumento de rendimiento en la aplicación principal. Por supuesto, diferir más procesos a aplicaciones más pequeñas y dedicadas aliviará al sistema principal, pero no debe verse como un reemplazo de toda una arquitectura completamente pensada. Evitar un cuello de botella debido a las capacidades limitadas de inserción de la base de datos en la aplicación principal no lo hará desaparecer por completo con estas arquitecturas: lo más probable es que vuelva a aparecer, en las mismas o incluso peores proporciones. Sin embargo, los procesos de consumo se pueden optimizar aprovechando la gran cantidad de elementos para procesar que puede recibir al mismo tiempo (por ejemplo, usar inserciones de lotes en una base de datos).

## Disponibilidad del Sistema

Las aplicaciones Web de hoy en día normalmente dependen de varios servidores, de cientos o tal vez miles de estaciones de trabajo de cliente, de comunicaciones a través de redes internas y externas, de servicios de base de datos, de procesos operativos y de una gran cantidad de otros servicios de infraestructura que deben funcionar conjuntamente y de manera uniforme. Allí donde el ideal de negocio es un flujo continuo de información, y donde una interrupción supone gastos para la compañía, la creación de aplicaciones de alta disponibilidad se convierte en una importante estrategia comercial (Network, 2014).

Las compañías que confían cada vez más en aplicaciones distribuidas basadas en Web para llevar a cabo actividades comerciales importantes necesitan una gran variedad de opciones de diseño de disponibilidad, para satisfacer los requisitos de nivel de servicio de forma rentable (Greer, Matthes, & Cannistraro, 2010). En la práctica, no es necesario que todas las aplicaciones funcionen ininterrumpidamente y proporcionen una respuesta instantánea. Es posible que algunas aplicaciones produzcan errores sin consecuencia alguna. Otras aplicaciones pueden tolerar tiempos de inactividad imprevistos, pero requieren diferentes estrategias de recuperación; y existen aplicaciones que deben proporcionar una disponibilidad muy alta a través de estrategias de replicación en suspenso, que han de garantizar una recuperación instantánea y transparente sin apenas tiempos de inactividad perceptibles.

Tal y como se describe en el tema referente a la confiabilidad (Blázquez & Martínez-

Abarca, 2005), se pueden producir errores en las aplicaciones por muchos motivos:

* Comprobación inadecuada
* Problemas relacionados con cambios en la administración
* Falta de control y análisis continuados
* Errores en las operaciones Código poco consistente
* Ausencia de procesos de diseño de software de calidad
* Interacción con aplicaciones o servicios externos
* Condiciones de funcionamiento distintas (cambios en el nivel de uso, sobrecargas máximas)
* Sucesos inusuales (errores de seguridad, desbordamientos en la difusión)
* Errores de hardware (discos, controladores, dispositivos de red, servidores, fuentes de alimentación, memoria, CPU).
* Problemas de entorno (red eléctrica, refrigeración, incendios, inundaciones, polvo, catástrofes naturales)

Cuando los desarrolladores obtienen una mejor perspectiva de lo que ocurre en el sistema, pueden escribir mejor código (Martin, 2009). Los logs son una parte clave de cualquier sistema, estos nos dan la perspectiva interna de lo que el sistema está haciendo y lo que está ocurriendo. Virtualmente cada proceso corriendo en un sistema genera logs de una forma u otra.

Usualmente estos logs son escritos en archivos físicos en disco. Cuando nuestro sistema corre en múltiples servidores, administrarlos y accederlos puede volverse complicado. Buscar un error particular a través de varios servidores, archivos y miles o millones de líneas es arduo sin buenas herramientas y un buen diseño de logueo.

# Caso de estudio

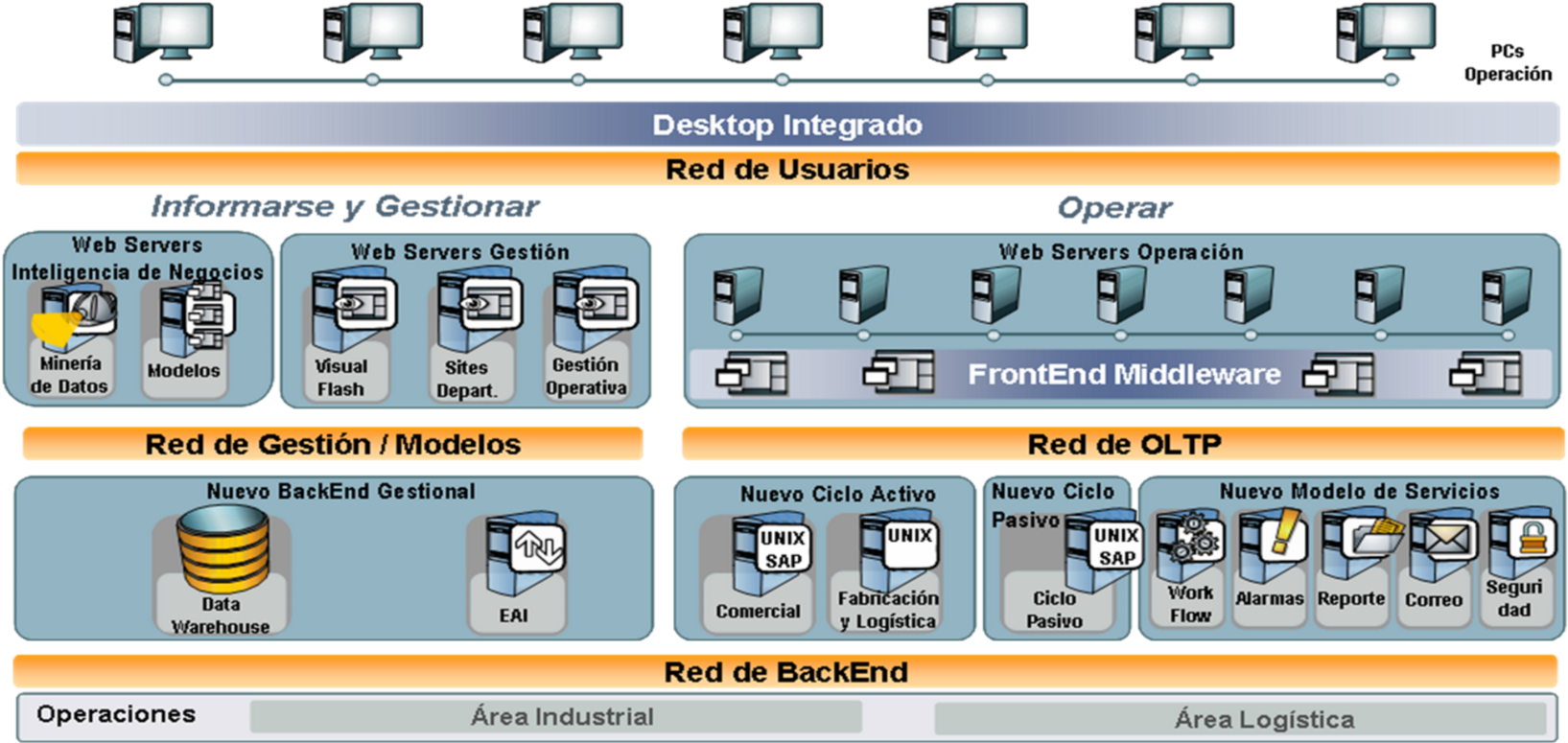
## Análisis de la problemática actual

El ciclo activo de la empresa en estudio, involucra procesos comerciales e industriales metalúrgicos: planificación, comercialización, fabricación, calidad y logística. La empresa dispone de un catálogo con más de 2500 productos metalúrgicos con un promedio de 100 variantes expresadas en especificaciones únicas. Cada una de las áreas del Ciclo Activo está soportada dentro de un único sistema integrado llamado Nuevo Ciclo Activo (NCA). Este proyecto es uno de los más ambiciosos que la empresa ha implementado ya que integra en un solo sistema los procesos de las áreas industrial, comercial, Supply Chain, calidad, planeamiento y administración, representando el core del negocio.

El Sistema NCA integra las distintas instalaciones que se encuentran distribuidas en 22 empresas enmarcadas en 2 áreas managers (AM) principales ubicadas en México (Norte) y Argentina (Sur).

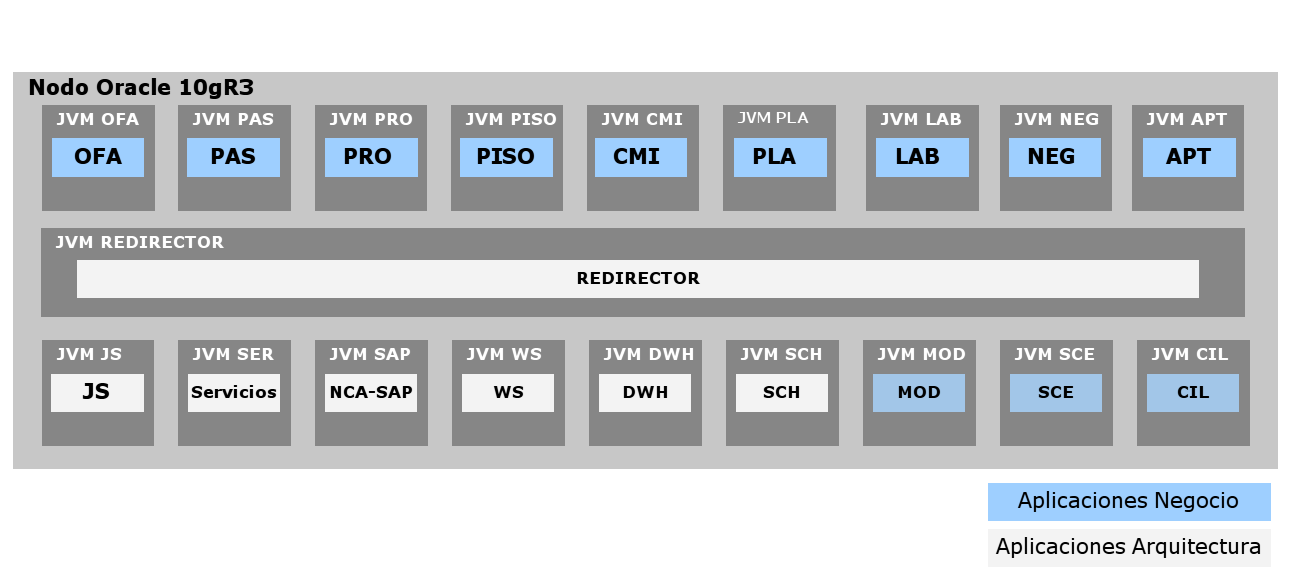
## Arquitectura general del sistema actual

En el siguiente diagrama podemos observar los principales componentes que conforman el alcance del área de Sistemas de la Compañía. ***Figura 15****: Diagrama del área de Sistemas*

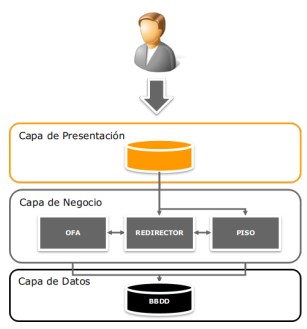


*fuente: Dirección de Sistemas.*

Cada área de proceso dispone de un catálogo de operaciones dentro del sistema que pueden ser ejecutadas y resueltas por el propio módulo invocado o puede solicitar algún servicio externo de otro frente de negocio (ver figura 1). Cada operación sobre el sistema es único y está enmarcado dentro de un modelo transaccional.

***Figura 16****: Figura 2: Diagrama de Módulos que componen el NCA fuente: Dirección de Sistemas.*

Cada módulo del sistema está construido bajo un diseño de 3 capas:

* **Capa de Presentación: *Figura 17****: Diseño arquitectura 3 capas* Desarrollada en tecnología web, sirve de

interfaz con el cliente (usuario final).

* **Capa de Negocio:**

Conformada por aplicaciones desarrolladas en Java, conforman la capa de servicios que implementan la lógica de negocio.

* **Capa de Datos:**

*fuente: Dirección de Sistemas*

Contiene el modelo de información del proyecto, cada aplicación tiene su propio esquema de base de datos.

## Sistema actual de generación y administración de los logs

Cada transacción exitosa o fallida del negocio es persistida en la base de datos transaccional (OLTP[[19]](#footnote-19) por su sigla en ingles *OnLine Transaction Processing*) y a su vez cada operación produce una serie de logs con un detalle minucioso de cada paso realizado durante la ejecución del proceso. Estos logs son generados por las aplicaciones y persistidos en archivos de texto planos. Dada la limitación de espacio y la alta velocidad de generación de logs, se configuran para rotar cada 10 megas y se generan un set de 10 archivos por aplicación. Una aplicación puede generar un archivo de 10 megas en 2 segundos, con lo cual para poder disponer de esta información por un período de tiempo mayor, se construyó un servicio que toma los archivos, los comprime y los almacena en catálogos dentro de un filesystem dedicado para tal fin.

Los volúmenes transaccionales que se manejan están en el orden de las 600.000 transacciones/hora o 15 millones/día; se calcula que en cada transacción participan en promedio 4 frentes y se realizan por lo menos 4 operaciones, resultando al final en unos 10 millones de registros por hora persistidos en la base transaccional, mientras que lo producido en logs se distribuye en 4 archivos por aplicación por nodo.

Tomando el caso de archivos logs con más alta rotación, a continuación, se muestra el costo de almacenamiento diario de una aplicación en ambiente productivo:

*Tabla 1 Relación costo en disco para soportar logs productivos*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Categoría/ métrica** | **Cantidad archivos** | **Tamaño archivo**  **(mb)** | **Archivos por minuto** | **Total (mb/minuto)** | **Total gb/día** |
| BaseLine | 1 | 10 | 1 | 10 | **14.1** |
| Real | 4 | 10 | 1 | 40 | **56.3** |

*fuente: elaboración propia*

Proyectando a 14 aplicaciones actualmente productivas, tenemos un consumo de storage estimado de 280 TeraBytes[[20]](#footnote-20) anuales. (ver Tabla 2).

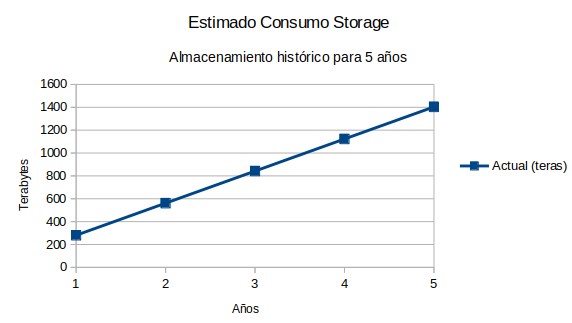
***Tabla 2****: Proyección de costo de storage anual para 14 aplicaciones*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Categoría / métrica** | **Aplicaciones** | **Instancias** | **GB/día** | **TB/año** |
| **BaseLine** | 1 | 1 | 14.1 | 5.0 |
| **Real** | 14 | 4 | 787.5 | **280.7** |

*fuente: elaboración propia*

En las condiciones actuales, con una tasa de crecimiento anual demostrada de 20% y con el objetivo de poder mantener 5 años de información, se necesitarían disponer de aproximadamente 1.4 PetaBytes[[21]](#footnote-21)[[22]](#footnote-22) de almacenamiento físico (ver gráfica 1), lo cual no solo es financiera y técnicamente inalcanzable para la administración actual de la dirección de Sistemas; sino también difícilmente sostenible.

***Figura 18****: Proyección de almacenamiento necesario para 5 años de información histórica*



*Fuente: elaboración propia*

Durante el ciclo de vida del sistema, en ambientes productivos las fallas reportadas se escalan a partir de 2 fuentes principales:

* Las reportadas por el cliente final al servicio de mesa de ayuda.
* Las detectadas de forma temprana por el equipo de IT o de Arquitectura mediante alguna de las herramientas de monitoreo actualmente disponibles en la empresa.

Desde el momento de reporte de la falla hasta su diagnóstico, se utilizan fuentes de información diversas para la recolección de información:

* Información relevada por el equipo de mesa de ayuda con el cliente.
* Orden y secuencia de los eventos en las transacciones involucradas.
* Indicadores de performance y recursos físicos de los servidores de base de datos y de aplicaciones.
* Indicadores de estado general de procesos asíncronos.
* Otros.

Sin embargo, el principal insumo de información, que posee la mayor riqueza en detalle es el log de la aplicación y es la necesidad de facilitar su almacenamiento y explotación donde descansa la principal motivación del presente trabajo.

## Soluciones típicas para la recolección y explotación de logs

La búsqueda de un error particular a través de cientos de archivos en múltiples servidores es difícil sin herramientas adecuadas. A continuación abordaremos un conjunto de herramientas y procesos que existen en el mercado bajo licencias comerciales y otras de código abierto enfocadas a ofrecer una solución a los análisis de los logs:

Una aproximación común a este problema es el diseño de una solución para logueo centralizado, de manera que múltiples logs de múltiples orígenes puedan ser agregados en una locación central, tal como la solución actual del caso de estudio.

##### Replicación de Archivos

Un método sencillo es la replicación de archivos de logs de sus registros a un servidor central con un programa cronológico. Por lo general, herramientas como *rsync[[23]](#footnote-23)* y cron[[24]](#footnote-24) se utilizan ya que son simples y fáciles de configurar. Esta solución puede funcionar por un tiempo, pero no proporciona acceso oportuno a los datos y registros. Asimismo, no almacena los registros; sólo los re ubica.

##### Syslog

Otra opción usada generalmente es *syslog*. Muchas personas usan *rsyslog* o *syslog-ng* los cuales son dos implementaciones de *syslog*. Estos servicios trabajan en segundo plano permitiendo enviar mensajes de logs a ellos y posteriormente almacenarlos en función a las políticas y configuraciones realizadas. En un sistema de logueo centralizado, un servicio principal de syslog es configurado en los nodos de la red y estos envían los mensajes al servicio central. Una buena referencia de instalación de este tipo de servicio puede ser consultado en este apartado(“Centralized Logging Using Rsyslog”, 2010).

*Syslog* es una solución interesante, dado que viene instalado por defecto en los servidores Unix y Linux como parte de la suite del sistema operativo. Con un servidor central *syslog,* el siguiente paso será configurar una arquitectura para escalar horizontalmente y llevarlo a un diseño de alta disponibilidad.

##### Recolectores de logs distribuidos

Una nueva clase de soluciones que han surgido se han diseñado para alto volumen de registro de alto rendimiento y la recopilación de eventos. La mayoría de estas soluciones son sistemas más generales de transmisión, procesamiento y registro de eventos. Todos estos tienen sus características y diferencias específicas pero sus arquitecturas son bastante similares. Generalmente se trata de capturar clientes y / o agentes en cada host específico.

Los agentes transmiten registros a un grupo de coleccionistas que a su vez reenvían los mensajes a un área de almacenamiento escalable. La idea es que el área de almacenamiento sea escalable horizontalmente creciendo con el aumento en el número de hosts y el registro de mensajes.

##### Cloud Hosted Logging Services(Servicios de logueo en la nube)

Existen en el mercado proveedores de servicios en la nube que ofrecen “logging as a service”. Entre los beneficios que representan, está el hecho de que solo necesitaríamos configurar nuestros agentes para publicar los logs y el proveedor del servicio se encargaría de los procesos de administrar la recolección, almacenamiento y acceso a los logs. Por su cuenta también van todos los temas relacionados con las configuraciones y mantenimiento de toda la infraestructura necesaria para mantener el sistema, liberándonos de esos aspectos y permitiéndonos hacer foco en nuestra aplicación. Cada servicio provee una configuración sencilla, una API y una UI para soportar las búsquedas y el análisis. Como contraparte están los costos asociados al ancho de banda, las restricciones legales por políticas propias de cada empresa para exteriorizar sus datos fuera del dominio de su red y los aspectos relacionados a las estructuras de los logs, los cuales deberán ser ajustados a las posibilidades que ofrezca cada servicio.

Entre algunos de los principales proveedores de este servicio encontramos:

* Loggly (“Log Management | Loggly”, 2017)
* Papertrail (“Papertrail - cloud-hosted log management, live in seconds”, 2017)
* Logentries (“Log Management & Analysis Software Made Easy | Logentries”, 2017)

Hasta este momento, hemos cubierto una serie de herramientas y frameworks que nos permiten resolver de manera general los aspectos relacionados a la captura y explotación de los logs. Muchas de esas herramientas apuntan a una porción del problema y otras a encararlo de manera general, lo cuál implica que necesitaremos combinar algunos de ellos para entre todos construir una solución robusta.

Los principales aspectos que necesitaremos encarar están relacionados con:

recolección, transporte, almacenamiento y análisis. En algunos casos particulares necesitaremos disponer de capacidades de notificaciones y alertas.

###### Recolección

Las aplicaciones crean logs de diferentes maneras, algunas a través de *syslog*, otras directamente en archivos. Si consideramos una típica aplicación web corriendo en un servidor linux, encontraremos más de una docena de archivos logs en la carpeta /var/log así como algunas otras en las carpetas home, entre otras.

Si estamos dando soporte a una aplicación web de cara al cliente y nuestro equipo de desarrollo o de operaciones necesita acceso a los logs de forma rápida con el propósito de resolver problemas que están ocurriendo en vivo, necesitaremos una solución que sea capaz de monitorear los cambios en los archivos de logs en *near real-time[[25]](#footnote-25)*.

Si estamos utilizando un sistema basado en la replicación de archivos a un servidor central, mediante un sistema de tareas programadas, entonces podremos inspeccionar los datos de los logs con la frecuencia que corra la replicación. Por ejemplo, un job que corra cada un minuto para sincronizar la data puede no ser suficiente cuando nuestro sistema ha fallado causando la caída del sistema y nosotros esperando para que la data relevante de los logs termine de ser replicada.

Por otro lado, si necesitamos analizar data de los logs de forma offline para calcular métricas o realizar cualquier otro trabajo en lote, una estrategia de replicación puede adaptarse bien a las necesidades.

###### Transporte

La data de los archivos logs puede acumularse rápidamente en múltiples servidores. Transportarlo de forma segura y rápida a su ubicación central puede necesitar herramientas adicionales para poder transmitirlo de forma efectiva y garantizar que los datos no se pierdan.

*Frameworks* como Scribe, Flume, Heka, Logstash, Chukwa, fluentd y Kafka fueron diseñados para transportar altos volúmenes de datos desde un servidor a otro de forma segura y confiable. Aunque cada uno de estos *frameworks* encaran el problema del transporte, cada uno lo hace en cierta manera de diferente forma. Por ejemplo, Scribe, nsq y Kafka, requieren de un cliente para registrar la data de los logs a traves de su API. Generalmente, su código está programado para registrarse a las fuentes que producen los logs directamente y con esto reducir la latencia y mejorar la fiabilidad. Si queremos centralizar los archivos de logs, necesitaremos algún mecanismo para suscribir los distintos *streams* de logs a través de su respectiva API. Si tenemos el control de la aplicación que está logueando la data que queremos recolectar, el proceso será más eficiente dado que podremos configurar a detalle las políticas para cada aspecto productor de los logs, así como normalizar el formato de los mismos en función a nuestras necesidades.

Logstash, Heka, fluentd y Flume proveen un número de fuentes de entrada, así como soporte nativo para la depuración y filtrado seguro del contenido de los logs. Estos poseen una estrategia que se ajusta mejor para un proceso de recolección y transporte general.

Aunque rsyslog y Syslog-ng son generalmente considerados como los colectores de registro de facto, no todas las aplicaciones utilizan syslog.

###### Almacenamiento

Ahora que los archivos de logs se están transfiriendo, necesitaremos un destino. Nuestro sistema de almacenamiento centralizado debe ser capaz de manejar el crecimiento de los datos en el tiempo. Cada día se agregará una cierta cantidad de almacenamiento que es relativa al número de hosts y procesos que están generando los logs.

La manera de almacenar los logs dependerá de algunos aspectos:

* ¿Cuánto tiempo debemos almacenar?

◦ Si los registros son con el propósito de mantenerlos a largo plazo, con un propósito de simplemente archivarlos y no requieren un análisis inmediato, S3

(“Amazon Simple Storage Service (S3) — Cloud Storage — AWS”, 2017), AWS Glacier (“Amazon Glacier - Cloud Archive”, 2017) o incluso un sistema de copia de seguridad en cinta[[26]](#footnote-26) podrían ser una opción adecuada, ya que ofrecen un costo relativamente bajo para grandes volúmenes de datos. Si sólo necesitamos unos días o meses de registros, almacenarlos en algún tipo de sistema de almacenamiento distribuido como HDFS, Cassandara, MongoDB o ElasticSearch también funciona bien. Si sólo necesitamos unas pocas horas de retención para el análisis en tiempo real, Redis podría funcionar también.

* El volumen de datos del sistema:

◦ El volumen de registros de un día para Google es muy diferente al de un día de logs para un sistema promedio. El sistema de almacenamiento que elijamos nos deberá permitir escalar horizontalmente si nuestro volumen de datos es grande.

* ¿Cómo acceder a los registros?

Algunos sistemas de almacenamiento no son adecuados para el análisis en tiempo real o incluso por lotes. AWS Glacier o las copias de seguridad en cinta pueden tardar horas en cargar un archivo. Estos no funcionan si necesitamos acceso a los logs para la resolución de problemas de producción. Si planeamos realizar un análisis más interactivo de datos, el almacenamiento de datos de registro en ElasticSearch(Elasticsearch, 2017a) o HDFS(IBM, 2017) puede permitirnos trabajar con los datos sin procesar de manera más eficaz. Algunos datos de logs son tan grandes que solo pueden analizarse con frameworks orientados a lotes. El estándar de facto es este caso es Apache Hadoop(Zikopoulos & Eaton, 2011) junto con HDFS.

###### Análisis

Una vez que los logs se almacenan en una plataforma de almacenamiento centralizada, necesitaremos una manera de analizarlos. El enfoque más común es un proceso orientado por lotes que se ejecute periódicamente. Si está almacenando logs en HDFS, Hive(“Apache Hive TM”, 2011) o Pig(“Apache Pig”, 2017) podría ayudar a analizar los datos más fácilmente que escribir jobs nativos de MapReduce.

Si necesitamos una interfaz de usuario para el análisis, podemos almacenar logs analizados en ElasticSearch(Elasticsearch, 2017a) y utilizar un front-end como Kibana(Elasticsearch, 2017b) o Graylog2(“Graylog Open Source Log Management”, 2017) para consultar e inspeccionar los datos. El análisis de logs puede ser manejado por Logstash(Co, 2017), Heka(“Heka open source stream processing”, 2014) o aplicaciones de logs con JSON directamente. Este enfoque permite mejor acceso interactivo en tiempo real a los datos, pero no es realmente adecuado para un procesamiento masivo por lotes.

###### Alertas

Otro componente funcional importante es la capacidad de alertar sobre patrones de registro o métricas calculadas basadas en datos de los logs. Dos usos comunes para esto son informes de errores y monitoreo y alertas tempranas ante tendencias.

La mayoría de los datos de los logs no son interesantes, pero los errores casi siempre indican un problema. Es mucho más efectivo tener el sistema notificación por ejemplo vía correo electrónico o mobile para notificar a las partes respectivas cuando ocurren los errores en lugar de tener a alguien que vea los eventos esporádicamente. Existen varios servicios que proporcionan este tipo de información de las aplicaciones, tales como Sentry(“Sentry realtime tracking”, 2017) o HoneyBadger(“Honeybadger Exception and Uptime Monitoring”, 2017).

Estos también pueden agregar excepciones y funciones de frecuencia, que pueden darnos una idea de la frecuencia con la que está ocurriendo un error; así como informarnos cuando se entra en una situación problema y cuando se sale de ella.

Otro caso de uso es el monitoreo. Por ejemplo, es posible que tengamos cientos de servidores web y deseamos saber si comienzan a devolver códigos error 500. Si puede analizar los archivos de logs web y registrar una métrica en el código de estado, podemos activar alertas cuando esa métrica cruce un cierto umbral. Riemann(“Riemann - A network monitoring system”, 2015) o Sisense(“Business Intelligence (BI) Software | Sisense”, 2017) están diseñados para detectar escenarios como este.

## Propuesta de la solución general.

En el sistema empresarial estudiado, los logs de las aplicaciones están generándose y persistiéndose desde diferentes sistemas y en múltiples servidores dada la arquitectura de alta disponibilidad sobre la cual funcionan.

Al día de hoy, si queremos consultar el log de una aplicación en particular, debemos:

* Determinar los datos de la transacción:

◦ Aplicación de negocio que ejecutó el servicio.

◦ Servidor donde se ejecutó la transacción.

◦ Id de transacción, fecha y hora.

* Ingresar al servidor y filesystem que corresponde al área mánager:

◦ Localizar la carpeta: Servidor → Nodo → Aplicación.

◦ Localizar el/los archivos en función a la fecha y hora del evento.

◦ Copiarlo al disco local y descomprimirlo.

◦ Abrir el log con un editor de texto y buscar el id de la transacción.

◦ Extraer el stacktrace de la transacción del resto de las transacciones.

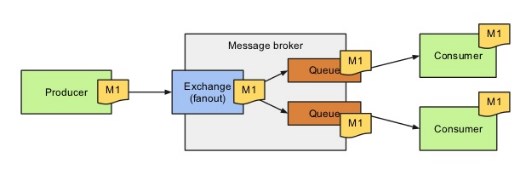
◦ Analizar el log en búsqueda de detalle para determinar causa y solución.

Esta solución fragmentada de persistencia dificulta la búsqueda de información, su monitoreo y la resolución de problemas para mantener las aplicaciones. Además de generar vacíos de información en algunos archivos de logs dado que en momentos de alta demanda transaccional se genera alta rotación de los archivos originales.

Resumiendo algunas características funcionales del proceso de generación de logs en el caso de estudio:

* La carga de trabajo por aplicación está dividida a su vez dentro de subprocesos propios de cada una y pueden ejecutarse de manera asincrónica.
* Las tareas son independientes y pueden correr en paralelo.
* El volumen transaccional del sistema es variable, lo que requiere una solución escalable.
* La solución debe proveer alta disponibilidad y debe ser resistente ante fallas posibles durante alguna de las etapas del procesamiento del mensaje.

Basado en lo anterior, se propone pasar de una arquitectura basada en colas asincrónicas de replicación de archivos batch, a una arquitectura centralizada orientada a eventos, con un enfoque integral que abarca cada etapa del proceso:

Producción, transporte, almacenamiento, explotación. ***Figura 19****: Patrón básico productor-consumidor fuente: https://www.slideshare.net/datakurre/plone-rabbit-mq-andmessaging-that-just-works*

El logueo centralizado implica la unificación del formato y almacenamiento de los logs. Estos son enviados desde las aplicaciones a un servicio, en el cual es transformado e importado a una base de datos documental para luego ser monitoreados, analizados y presentados mediante una interface.

Los principales beneficios del modelo centralizado son:

* Proporciona un sistema de nivel de carga que puede manejar amplias variaciones en el volumen de solicitudes enviadas por instancias de la aplicación. La cola actúa como un buffer entre las instancias de la aplicación y las instancias de servicio al consumidor. Esto puede ayudar a minimizar el impacto en la disponibilidad y la capacidad de respuesta tanto para la aplicación como para las instancias de servicio, tal como se describe en el patrón de nivelación de carga basado en colas.
* La gestión de un mensaje que requiere un procesamiento prolongado no impide que otras instancias del servicio al consumidor manejen otros mensajes de forma simultánea.
* Mejora la confiabilidad. Si un productor se comunica directamente con un consumidor en lugar de utilizar este patrón, pero no monitorea al consumidor, existe una alta probabilidad de que los mensajes se pierdan o no se procesen si el consumidor falla. En este patrón, los mensajes no se envían a una instancia de servicio específica. Una instancia de servicio fallida no bloqueará a un productor y los mensajes pueden ser procesados por cualquier instancia de servicio que funcione.
* No requiere una coordinación compleja entre los consumidores o entre las instancias del productor y del consumidor. La cola de mensajes garantiza que cada mensaje se entregue al menos una vez.
* Es escalable El sistema puede aumentar o disminuir dinámicamente el número de instancias del servicio al consumidor a medida que el volumen de mensajes fluctúa.
* Puede mejorar la flexibilidad si la cola de mensajes proporciona operaciones de lectura transaccional. Si una instancia de servicio al consumidor lee y procesa el mensaje como parte de una operación transaccional, y la instancia de servicio al consumidor falla, este patrón puede garantizar que el mensaje se devolverá a la cola para ser recogido y manejado por otra instancia del servicio al consumidor .
* Los logs están almacenados en una ubicación puntual. En vez de tener que recuperar logs de múltiples ubicaciones, podemos acceder a toda la información desde una simple interface.
* La data es automáticamente parseada. Los logs pueden llegar en cientos de formatos diferentes, incluyendo texto plano, json, xml. El sistema de logueo detecta automáticamente el tipo y lo convierte a un formato estándar.
* Los logs podrán ser indexados, consultados y exportados. Los logs pueden ser

exportados o redistribuidos sin tener que tocar el archivo original.

# Comparación y selección de herramientas big data para la solución

Para llevar a cabo este análisis, lo abordaremos partiendo de las capas de la solución propuesta: Capa de producción, recolección, transporte y explotación.

## Capa de producción

Esta capa representa el proceso de producción de logs por parte de las aplicaciones de negocio. Luego de investigar y analizar diferentes frameworks del mercado y considerando los sistemas de producción del sistema donde se implementará la solución, seleccionamos log4j2 como framework de logueo de las aplicaciones java.

Entre algunos de las motivaciones de esta selección tenemos:

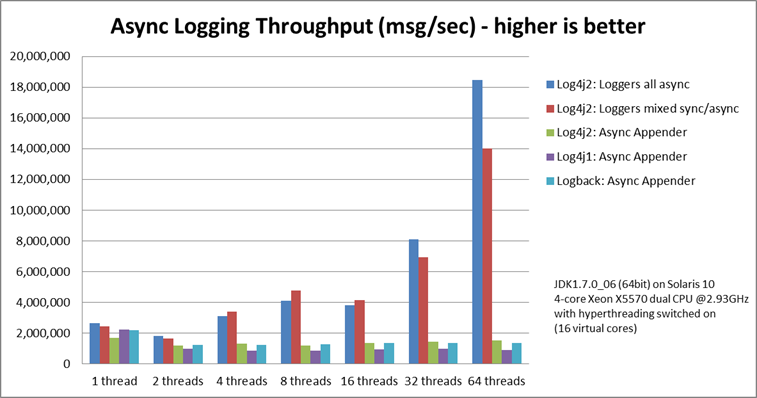
Soporte de la comunidad, Log4j 1.x no está siendo mantenida actualmente, mientras que Log4j 2 está siendo mantenida y soportada, recibe correcciones y actualizaciones. Desde agosto del 2015 Log4j 1.x está oficialmente *End of Life(MidcomData, 2014)* y es recomendado actualizar a Log4j2. Por otro lado Log4j1 no es compatible con Java 9.

Esta versión con respecto a su predecesora incluye una serie de mejoras tanto de performance como de estabilidad por la cantidad de bug’s corregidos. Esta nueva versión del log4j extiende la capacidad de los formatos disponibles en la nueva versión, entre ellos el formato JSON, el cual se ha transformado en un formato de facto para las herramientas big data (David Liedle, 2015).

Desde el punto de vista de performance, ha demostrado excelentes resultados en comparación con otros frameworks similares del mercado. A continuación se muestra una prueba de performance donde se puede observar que mientras log4j2 asincrónico y mixto loquea alrededor de 18.000.000 de mensajes por segundo, otros llegan a 1.600.000. Fuente de referencia (The Apache Software Foundation, 2017)

El logueo asincrónico puede mejorar el rendimiento de las aplicaciones al ejecutar las operaciones de E/S en un hilo separado. Log4j 2 realiza una serie de mejoras en esta área. Los logs asincrónicos son una nueva adición en Log4j 2. Su objetivo es regresar de la llamada a Logger.log a la aplicación lo antes posible. Se puede elegir entre hacer que todos los appenders sean asíncronos o utilizar una combinación de síncronos y asíncronos. Hacer todos los logs asíncronos dará el mejor rendimiento, mientras que la mezcla le da más flexibilidad.

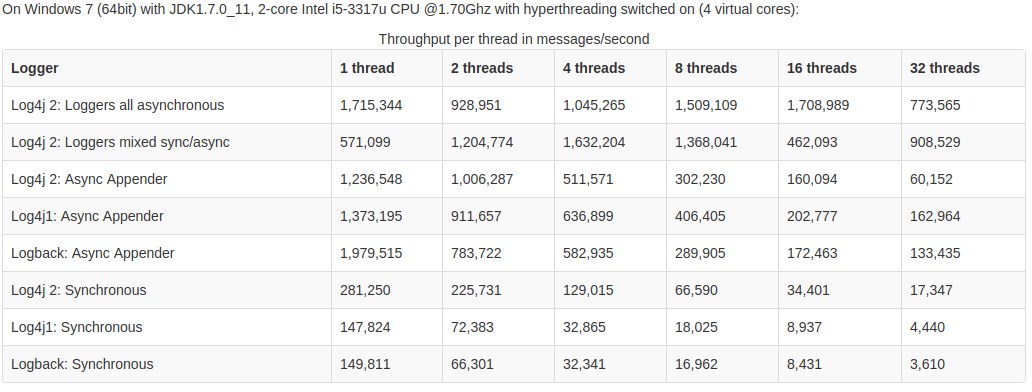
***Figura 20****: Pruebas de Performance Log4j2 Async Logging.*



*Fuente: The Apache Software Foundation, https://logging.apache.org/log4j/2.x/manual/async.html*

Los logs asíncronos internamente usan el Disruptor(LMAX-Exchange, 2011), una biblioteca de comunicación entre hilos sin bloqueo, en lugar de colas, lo que resulta en un mayor rendimiento y una menor latencia. Como parte del trabajo para los logs de Async, los apéndices asíncronos se han mejorado en la etapa de descarga al disco al final de un lote (cuando la cola está vacía). Esto produce el mismo resultado que configurar "immediateFlush = true", es decir, todos los eventos de registro recibidos están siempre disponibles en el disco, pero es más eficiente porque no necesita tocar el disco en cada evento de registro. (Los Appenders asincrónicos usan ArrayBlockingQueue internamente y no necesitan el contenedor disruptor en classpath).

***Figura 21****: Performance por Threads medidos en mensajes por segundo*



*Fuente: The Apache Software Foundation, https://logging.apache.org/log4j/2.x/manual/async.html*

El mecanismo de logueo asincrónico está implementado usando la librería para mensajería entre threads LMAX Disruptor(LMAX-Exchange, 2011). El uso de colas para transferir data entre áreas del sistema introduce latencia, por lo que el foco apuntó a optimizar esa área. Se determinó que la caché falla a nivel de CPU, y los bloqueos que requieren del arbitraje del kernel son altamente costosos, así que crearon un framework con un comportamiento orientado a la “Simpatía mecánica" por el hardware en el que se ejecuta mediante la comunicación directa entre los threads que participan del proceso lo cual evita los bloqueos entre las etapas de E/S. Las comparaciones de rendimiento interno de LMAX Disruptor con java.util.concurrent.ArrayBlockingQueue se pueden encontrar aquí (Foundation, 2012).

## Capa de recepción

Es el área donde se recibirán y procesarán los mensajes enviados por la capa de producción para ser procesados y enviados de forma asincrónica a los consumidores de los logs. Para seleccionar el broker para la solución propuesta, repasaremos cuáles son los middlewares orientados a mensajes existentes en el mercado y cuáles son sus capacidades funcionales.

### Selección de un broker de mensajes

Esta sección presenta el estado del arte en ese dominio y nos enfocaremos en dos de los principales productos disponibles en el mercado, Apache Kafka(“Apache Kafka”, 2017) y RabbitMQ(Pivotal Software Inc, 2017).

#  Estado del arte en arquitecturas orientadas a mensajes

En busca de alternativas para resolver problemas de escalabilidad y de rendimiento que implicaba las llamadas a procedimientos remotos (RPC), en los años 90 se publicaron varios artículos de middleware orientados a mensaje (Computerworld, 1996b) (Computerworld, 1996a). En ese tiempo, IBM era el competidor más serio en el campo con sus productos MQSeries, junto con algunas otras compañías como NEON (Nueva era de la red por sus siglas en inglés). Varios de estos productos de IBM todavía existen, renombrado como WebSphere MQ, pero son comerciales y propietarios, y por lo tanto son no de código abierto. Otras herramientas aparecieron en los años 2000. La fundación Apache originó varios proyectos, como ActiveMQ en 2004 y Qpid en 2005 (Apache Fundation, 2015).

A partir del 2014, ActiveMQ es uno de los servidores de mensajes de código abierto más popular y potente, de acuerdo con el sitio web del proyecto. Tanto ActiveMQ como Qpid también son compatibles con JMS o Java Message Service, que es una API de Java que permite que la aplicación Java se conecte a cualquier broker que lo implemente, sin definir realmente nada más que interfaces para ser implementado por los sistemas de middlewares de mensajes que cumplan con JMS.

El panorama para los middlewares orientados a mensajes cambió con el inicio de un proceso abierto estándar: Advanced Message Queuing Protocol (AMQP) en 2003 (Vinoski, 2006). Originado en JPMorgan Chase en Londres, en un esfuerzo por lograr una interoperabilidad simplificada entre operadores del mercado financiero, se convirtió en un trabajo de la industria global a la que 23 empresas hicieron contribuir (incluyendo Barclays, Cisco Systems, Microsoft Corporation y VMware). AMQP define una forma estándar para construir middleware orientado a mensajes (por ejemplo, en términos de soporte de confirmación de recepción de mensajes, entrega de mensajes push and pull, etc.). La versión final 1.0 fue finalmente lanzada en octubre de 2011.

Qpid fue la primera implementación de AMQP, mientras que RabbitMQ (creado por

Rabbit Technologies) llegó en el 2007 implementando el mismo protocolo, pero utilizando

Erlang en su lugar de Java y C++ como lenguaje de programación. Al día de hoy, "RabbitMQ es utilizado por todos, desde pequeñas startups en Silicon Valley hasta algunos de los nombres más grandes en Internet"(Videla & Williams, 2012). Cuenta con un fuerte apoyo de la comunidad, completamente compatible con AMQP, y puede aprovechar las capacidades que Erlang proporciona para facilitar el trabajo sobre clusters y procesamiento distribuido. También es de código abierto, parece rápido según algunos puntos de referencia, y es utilizado por grandes empresas como Google, VMware y Mozilla (VMware, 2011). Por estos motivos, se eligió RabbitMQ como uno de los middleware de mensaje para evaluar en este estudio. Sin embargo, AMQP no es el único camino a seguir. Otros protocolos como STOMP

(Streaming / Protocolo de mensajería orientada a texto simple) y MQTT (telemetría MQ

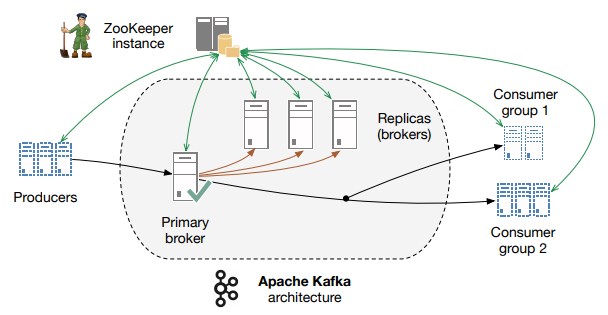
Transporte) también tienen como objetivo normalizar los intercambios entre los procesos (Stomp, 2012), (IBM, s. f.).

Otros framework orientados a mensajes y que poseen un mecanismo de optimización de persistencia es Apache Kafka y HornetQ. Estos son dos brokers de mensajes que fueron específicamente diseñados para ser eficientes cuando se escriben datos de la cola al disco (“Apache Kafka”, 2017), (RedHat, 2015). Ellos se soportan en métodos inteligentes de caché para evitar grandes impactos en el rendimiento debido a la importante cantidad de pequeñas operaciones de escritura que dicho sistema debería realizar al recibir miles de mensajes por segundo. Apache Kafka es una pieza reciente de software, creado en LinkedIn, se convirtió posteriormente en código abierto y ahora es mantenido por la fundación de software Apache. Dadas las posibilidades que ofrece mediante un esquema de almacenamiento de mensajes orientado a big data, es un producto que puede conformar soluciones interesantes y por tal motivo fue elegido para ser el segundo competidor en ser parte de este estudio.

#  Apache Kafka

Apache Kafka (“Apache Kafka”, 2017) es un middleware de mensajería de mensajes de fuente abierta, que se soporta en Apache ZooKeeper(Apache Software Fundation, 2010), ambos mantenidos por Apache Software Fundación, fue creado en LinkedIn en 2011. Diseñado teniendo en mente el rendimiento y persistencia, Apache afirma que "un solo broker de Kafka puede manejar cientos de megabytes de lecturas y escrituras por segundo desde miles de clientes". Fue desarrollado en Scala (un lenguaje de programación funcional orientado a objetos que se compila como bytecode de Java y se ejecuta en JVM(Scala, 2017)).

***Figura 22****: Diagrama de una arquitectura global de Apache Kafka (con un tópico, una partición y un factor de replicación 4)*



*Fuente: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apachekafka*

Kafka persiste los mensajes en disco tan pronto como los recibe. De hecho, los mensajes no son transmitidos al consumidor si no son previamente persistidos en el disco, específicamente al ingresar en el kernel’s pagecache de kafka. Por lo tanto una vez que el broker primario recibe el mensaje, no sería posible que lo pierda, incluso si hay una falla importante en el broker. Adicionalmente los brokers tienen réplicas, y un mensaje puede ser opcionalmente considerado como “encolado” una vez que el mensaje ha sido replicado en todas las réplicas del grupo. Esta arquitectura diseñada con orientación a persistencia, provee de un nivel máximo de seguridad ante fallas críticas del software y no debería perjudicar la performance en comparación con el modelo de encolamiento en memoria (Apache Software Fundation,

2017a).

#### *Estructura General*

Debemos tener en cuenta que Kafka no es auto suficiente, él depende de la infraestructura de Apache ZooKeeper, el cual actúa como un programador de tareas (Scheduler) y provee una base de datos compartida jerárquica clave/valor en la cuál se persisten los cambios de estado de los componentes que participan del intercambio de mensajes. En consecuencia, Apache Kafka no está disponible como un framework completo de por si, sino que se requiere de ZooKeeper para su funcionamiento, sin embargo, en la distribución de Kafka se proporciona una versión obsoleta de ZooKeeper, por lo cuál debe ser actualizada por separado.

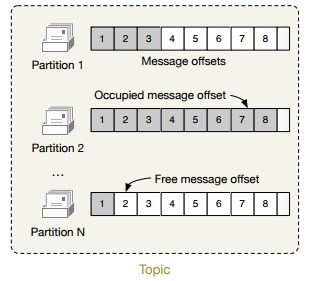
Kafka al igual que otros middlewares orientados a mensajes, posee el concepto de productor y consumidor de mensajes. Los productores envían el mensaje al broker, mientras que los consumidores los procesarán. Sin embargo, Kafka introduce un nuevo concepto que no comparte con los otros middlewares:

• **Tópicos y particiones.**

Un tópico puede ser considerado como un equivalente a una cola. Este comprende la entidad de agrupamiento más grande. Sin embargo los mensajes no solo son enviados por el productor a un tópico, sino también pueden ser enviados a una partición específica. Una partición es una subdivisión de un tópico y un broker tiene la responsabilidad de {0..N} particiones, donde N es el número de particiones en las cuales el tópico será dividido.

***Figura 23****: Arquitectura de un Tópico de*

*Kafka*



*Fuente: http://cloudurable.com/blog/kafkaarchitecture-topics/index.html*

Cada partición tiene su propia contabilidad en cuanto a capacidad de mensajes. Significa que los mensajes pueden ser ingresados de forma desigual a las particiones dentro de un tópico. Sin embargo, dado que las particiones son la entidad más pequeña de la que un broker puede ser responsable (y, como tal, la más pequeña entidad replicada), podría no llegar a ser una configuración prudente. En los casos típicos, en una partición round-robin la selección de qué partición o tópico ya está hecha. Tengamos en cuenta que la elección de qué partición debe ser entregado un mensaje depende del productor que envía el mensaje.

* Offset del mensaje

Tal como se observa en la figura anterior, cada mensaje que se almacena en una partición recibe un identificador único de tipo entero numérico. Como consecuencia, los mensajes que se encuentran en cola en una partición se almacenan (y por lo tanto se consumen) en el orden en que se enviaron. Tener un identificador único e incremental también permite a los consumidores "rebobinar" y reproducir mensajes con compensaciones de mensajes pasados.

* Grupos de consumidores

Kafka proporciona una forma inteligente de distribuir la carga en los consumidores a través de diferentes instancias al tiempo que también permite el consumo simultáneo de un mismo mensaje. Introduce para este propósito el concepto de grupo de consumidores.

El concepto de grupo de consumidores es una forma de lograr dos cosas: tener consumidores como parte del mismo grupo implica poder proporcionar el patrón de "consumidores competidores" con quienes los mensajes de las particiones de tópicos se distribuyen entre los miembros del grupo. Cada consumidor recibe mensajes de una o más particiones ("automáticamente" se le asigna) y los otros mensajes no los recibirán (asignados a diferentes particiones). De esta manera, podemos escalar el número de consumidores hasta el número de particiones (teniendo un consumidor leyendo solo una partición); en este caso, un nuevo consumidor que se una al grupo estará inactivo hasta ser asignado a alguna

partición.

En segundo lugar el concepto implica también el poder proporcionar el patrón "publicación / suscripción" donde los mensajes de las particiones/tópicos se envían a todos los consumidores de los diferentes grupos. Significa que dentro del mismo grupo de consumidores, tendremos las reglas explicadas anteriormente, pero a través de diferentes grupos, los consumidores recibirán los mismos mensajes. Es útil cuando los mensajes dentro de un tópico son de interés para diferentes aplicaciones que los procesarán de diferentes maneras. Queremos que todas las aplicaciones interesadas reciban todos los mismos mensajes del tópico.

Otra gran ventaja de la agrupación de consumidores es la función de reequilibrio.

Cuando un consumidor se une a un grupo, si todavía hay suficientes particiones disponibles (es decir, no hemos alcanzado el límite de un consumidor por partición), se inicia un reequilibrio y las particiones se reasignarán a los consumidores actuales, más el nuevo. Del mismo modo, si un consumidor deja un grupo, las particiones se reasignarán a los consumidores restantes.

#### *Funcionalidades principales*

Persistencia de los mensajes: La fortaleza principal de Apache Kafka, comparado con la mayoría de los sistemas de mensajes disponibles es que impone la persistencia del mensaje como regla: lo hace por diseño y no se puede elegir trabajar solo con colas en memoria. Por lo tanto, está optimizado para el trabajo intensivo sobre operaciones de lectura y escritura (I/ O) sobre disco(Kreps, Corp, Narkhede, & Rao, 2011). La persistencia de mensaje también implica la persistencia de la cola, lo que significa que las colas están previamente definidas por el administrador del broker y que existirán hasta tanto sean explícitamente borradas por

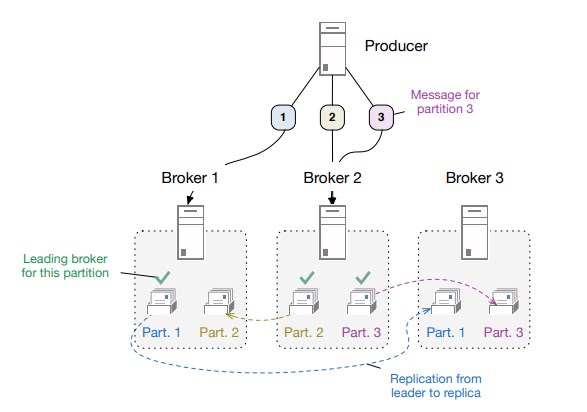
él.

Replicación y balance de carga por partición:

Kafka ofrece una manera fácil y conveniente de realizar el balanceo de carga y mejorar la tolerancia a fallas a nivel de broker: cada broker es responsable de algunas particiones y guardará una copia de ellas. En la siguiente figura se muestra cómo un sistema de 3 agentes con 3 particiones podría dividir la carga entre los diferentes brokers.

***Figura 24****: Diagrama de balance de carga y alta disponibilidad en*

*Kafka*



*Fuente: https://antonmry.github.io/talk-vigotech-2017-introductionto-apache-kafka/#/e-ti-de-quen-ves-sendo*

Debemos tener en cuenta que el broker 3 no recibe ningún mensaje de los productores (y no envía ningún mensaje a los consumidores): solo el nodo establecido como líder puede enviar y recibir mensajes. Es su deber mantener sus réplicas sincronizadas después. Tenga en cuenta también que un broker puede ser el broker líder para ninguna, una o más particiones de las que sea responsable. La distribución de partición en los brokers se realiza de forma predeterminada, pero también se puede realizar manualmente cuando se crean los tópicos.

Productor asíncrono:

A diferencia de otros middleware orientados a mensajes, Apache Kafka ofrece una forma conveniente e integrada para permitir a los productores publicar al broker mensajes de manera asincrónica. Este comportamiento se denomina envío de mensajes por lotes. Los productores tienen una cola local en la que publican mensajes y cuando un cierto número mensajes ha sido puesto en cola, o después de un cierto tiempo, se vacía la cola enviando en lote los mensajes al broker. Este mecanismo optimiza el uso de la red y el de los recursos globales de la infraestructura.

Confirmación de mensajes:

Los productores pueden requerir que el agente confirme una vez que los mensajes han sido encolados exitosamente. Este proceso se le denomina: confirmación de publicación. Si bien se puede optar por no hacerlo, luego se corre el riesgo de perder mensajes (por ejemplo, debido a problemas en la red). Se dan múltiples esquemas de confirmación, entre ellos uno simple, confirmando cuando los mensajes son recibidos y encolados por el broker líder, y una confirmación de "*commit*", enviado solo cuando el líder y todas sus réplicas tienen una copia del mensaje.

Compresión de los registros

Como Kafka almacena todo en el disco, la cantidad de datos que mantiene constantemente aumentan y puede alcanzar valores muy altos. Mantener todos los mensajes puede no ser necesario. Sumado a esto también otros componentes como los topics y las particiones generan logs de procesos. Estos pueden limpiarse para reclamar espacio. Hay dos modos disponibles: un modo de eliminación o un modo de compactación.

El método de borrado es simple: después de un período de tiempo determinado, o cuando el tópico llega a un tiempo de vida definido o un límite de tamaño, el registro se elimina. Se usa el modo de compactación para limpiar los registros manteniendo por ejemplo solo el último registro de interés o conservar solo los últimos mensajes con clave valor distintos.

Si los tópicos están diseñados para almacenar mensajes grandes, puede ser una buena idea habilitar la compresión de datos. Kafka proporciona configuraciones integradas que permiten la compresión usando ya sea *gzip* o el algoritmo de compresión *Snappy* (Apache Software Fundation, 2017b).

#  RabbitMQ

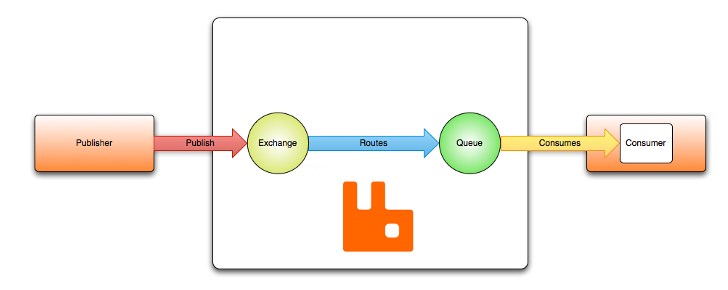
Es un software de negociación de mensajes de código abierto creado en el 2007, y entra dentro de la categoría de middleware de mensajería. Implementa el estándar Advanced

Message Queuing Protocol (AMQP)(Vinoski, 2006). El servidor RabbitMQ está escrito en

Erlang(Erlang.org, 2016) y utiliza el framework Open Telecom Platform (OTP)(Fady Sedrak, 2007) para construir sus capacidades de ejecución distribuida y conmutación ante errores. Rabbit Technologies Ltd., la compañía que lo desarrolla, fue adquirida en abril de 2010 por la división SpringSource de VMWare(VMware, 2017). El código fuente está liberado bajo la licencia Mozilla Public License(Mozilla, 2017). A partir de Mayo del 2013, por una unión de empresas, Pivotal es el nuevo patrocinante del proyecto.

La mayoría de las operaciones se realizan en memoria. RabbitMQ no está "orientado a persistencia en disco". Los brokers reciben los mensajes a través de un componente denominado E*xchange* (es decir, un punto de entrada lógico que decide en función de algunos criterios en qué colas del broker debe colocar un mensaje) y luego enviarlo a los consumidores registrados.

***Figura 25****: Diagrama simplificado de la arquitectura de RabbitMQ*



*Fuente: https://www.rabbitmq.com/tutorials/amqp-concepts.html*

Como se muestra en la figura anterior , RabbitMQ conecta a los productores que generan mensajes y los envían a un Exchange. Los Exchanges aplican reglas de enrutamiento en el mensaje (posiblemente en función de una clave de enrutamiento que el productor puso en el encabezado del mensaje) para decidir si el mensaje debe entregarse a cero, una o más colas (luego, duplica el mensaje). Los consumidores tienen una conexión permanente con el broker, que, por lo tanto, sabe qué consumidores están disponibles y a qué colas se suscribieron. El broker envía los mensajes a los consumidores siempre que es posible (es decir, hasta que se alcanza el recuento de captación previa o el consumidor rechaza los mensajes).

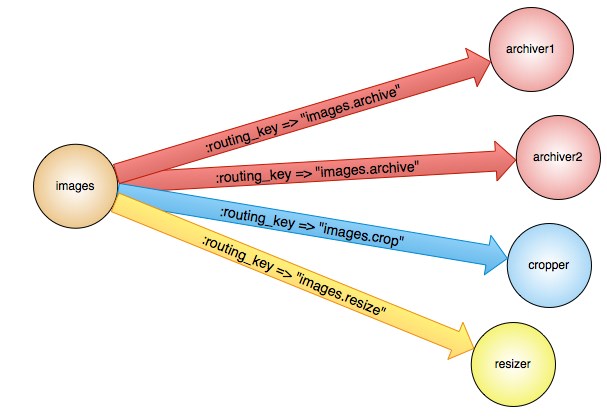
Una diferencia principal con Apache Kafka es que los mensajes (así como las colas y los Exchanges) no son persistentes por defecto. Todos los elementos que administran los brokers no sobreviven (con la configuración predeterminada) si el broker se reinicia o falla. Todo se mantiene en la memoria, que es una diferencia fundamental con la forma en que funciona Kafka. Afortunadamente, RabbitMQ ofrece configuraciones convenientes para hacer que tanto las colas como los mensajes sean duraderos. Los propios productores pueden etiquetar los mensajes como duraderos o no, de modo que puedan elegir mensajes por mensaje para no persistir algunos mensajes no críticos enviados a una cola persistente.

En lo que respecta al orden de los mensajes, la sección 4.7 de la especificación básica AMQP 0-9-1 explica las condiciones bajo las cuales se garantiza el orden: los mensajes publicados en un canal, que pasan por un Exchange y una cola se recibirán en el mismo orden en que se enviaron. RabbitMQ ofrece garantías más sólidas desde la versión 2.7.0. Los mensajes pueden devolverse a la cola utilizando métodos AMQP que tienen un parámetro de repetición (basic.recover, basic.reject y basic.nack), o debido a un cierre de canal mientras se mantienen mensajes no reconocidos. En versiones anteriores a la 2.7.0, cualquiera de estos escenarios causaban que los mensajes se reenvíen a la cola para relanzamientos de RabbitMQ. Desde la versión 2.7.0 de RabbitMQ, los mensajes siempre se mantienen en la cola en orden de publicación, incluso en presencia de reabastecimiento o cierre de canal(Pivotal Software Inc., 2017b).

#### *Estructura general*

• *Exchanges*: son partes constitutivas del protocolo AMQP. Un productor envía un mensaje a un Exchange específico (y no a una cola), que a su vez decidirá si debe redirigir, duplicar o descartar el mensaje. Los diferentes tipos de Exchanges están estandarizados y RabbitMQ los admite a todos (Pivotal Software Inc., 2017). Los productores pueden enviar en la cabecera del mensaje una clave que permita al broker identificar cuál debe ser el enrutamiento que debe seguir ese mensaje. El enrutamiento más simple es el denominado intercambio directo (*direct exchange*) como se muestra en la siguiente imagen:

*Figura 26: Ejemplo de intercambio directo (routing key = queues names = colours)*



*Fuente: https://www.rabbitmq.com/tutorials/amqp-concepts.html*

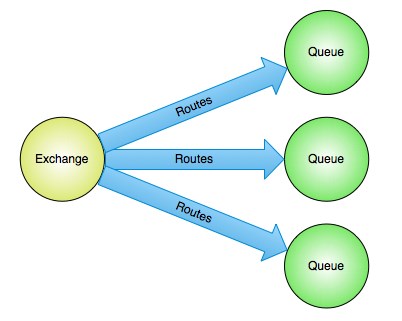
Este modelo de enrutamiento por defecto. Si el productor no envía una clave de enrutamiento, el exchange enviará el mensaje a todas las colas creadas en el sistema, donde cada cola tendrá una clave de enrutamiento por defecto igual al nombre de la cola. Por lo tanto, al enviar un mensaje al exchange predeterminado con la clave de enrutamiento "appgrid" enrutará el mensaje a la cola llamada "appgrid", si esta existe.

Otro modelo de enrutamiento interesante es el *fan-out.* Un exchange de fanout enruta mensajes a todas las colas que están vinculadas a él y se ignora la clave de enrutamiento. Si N colas están vinculadas a un intercambio de fanouts, cuando se publica un nuevo mensaje en ese intercambio, se entrega una copia del mensaje a todas las N colas.

Los intercambios de Fanout son ideales para el enrutamiento de difusión de mensajes(*broadcast*). Debido a que un exchange de fanouts entrega una copia de un mensaje a cada cola, los casos de uso son bastante similares: pueden usarse para actualizaciones de marcadores u otros eventos globales, por ejemplo: en los juegos masivos multijugador en línea. Los sitios de noticias deportivas pueden usar exchanges de fanouts para distribuir actualizaciones de puntaje a clientes móviles casi en tiempo real.

Los sistemas distribuidos pueden transmitir varias actualizaciones de estado y configuración. Los chats grupales pueden distribuir mensajes entre participantes utilizando un intercambio de fanouts. Un intercambio de fanout se puede representar gráficamente de la siguiente manera:

***Figura 27****: Ejemplo de exchange de tipo fan-out*



*Fuente: https://www.rabbitmq.com/tutorials/amqpconcepts.html*

Se puede lograr un enrutamiento más complejo utilizando otros tipos de exchanges (con encabezados y Exchanges topics), que no se explicarán en este documento. Para mayor información sobre otras topologías y formas de enrutamiento se pueden consultar las referencias bibliográficas indicadas (Pivotal Software Inc., 2017a).

Colas (*queues*):

Las colas en RabbitMQ son los puntos finales de todos los mensajes que se enrutaron correctamente (es decir, que no fueron eliminados por un exchange). Un mensaje en una cola se entregará (idealmente) solo una vez, a un solo consumidor que se suscribió a la cola, pudiendo a la vez múltiples consumidores estar subscritos a una cola conformando un grupo de consumidores que serán tratados como una entidad única.

Canales (*Chanels*):

Los canales son un componente de bajo nivel de AMQP, implementado en RabbitMQ, que se usa para multiplexar conexiones del mismo productor o consumidor al mismo broker, pero que son utilizadas por diferentes hilos, por ejemplo. En lugar de tener múltiples conexiones TCP establecidas entre el par y el broker, solo una se instancia y se mantiene activa, y RabbitMQ realiza la multiplexación para que todos los canales compartan esta única conexión TCP.

#### *Funcionalidades principales*

RabbitMQ es un sistema de colas de mensajes bastante estándar y comparte la mayoría de los conceptos generales de un Middleware orientado a mensajes. Sin embargo, aunque sigue el estándar AMQP, que en sí mismo contiene funcionalidades particulares, su equipo desarrolló algunas extensiones del protocolo que pueden ser útiles en el caso de uso estudiado:

* Almacenamiento en memoria y durabilidad opcional.

Como ya se mencionó en la sección anterior, RabbitMQ almacena todos los elementos que trata en la memoria, al tiempo que proporciona indicadores(*flags*) que pueden colocarse en las colas y en los mensajes para marcarlos como duraderos. Sin embargo, la durabilidad lograda no es comparable a la que presenta Apache Kafka, ya que la durabilidad provista es solo temporal: los mensajes se graban en disco solo si no hay un consumidor que pueda procesar el mensaje de inmediato. Los mensajes (indistintamente del flag de durabilidad) también se pueden persistir el disco si la memoria del broker se agota. Una vez que se ha entregado el mensaje (o que se confirmó su consumo si las confirmaciones están habilitadas), el mensaje se elimina, incluso si se etiquetó como duradero. RabbitMQ no proporciona una forma de aplicar almacenamiento permanente de forma similar a lo que hace Apache Kafka, y como tal no permite a los consumidores reproducir mensajes antiguos.

* Confirmación de entrega y rechazo de mensajes.

Los mensajes entregados a los consumidores pueden requerir confirmación de aceptación: esto garantiza que los mensajes se hayan consumido con éxito previo a que se eliminen de la memoria del broker. De manera predeterminada, no se necesita confirmación, y una entrega exitosa a uno de los consumidores registrados de una cola es suficiente para que el broker se olvide del mensaje. Las confirmaciones de consumo ayudan a reducir la cantidad de mensajes sin procesar, al tiempo que brindan la oportunidad a los procesos del consumidor de notificar al broker que un mensaje no se pudo procesar. De hecho, pueden rechazar un mensaje y potencialmente volver a ponerlo en cola para su nueva entrega. Un caso de uso que puede encontrar esta funcionalidad muy conveniente es por ejemplo una base de datos que temporalmente no está disponible y, por lo tanto, impide que se escriba el contenido de un mensaje, mientras que la conexión con

RabbitMQ no presenta ningún error.

* Colas y Exchanges declarados dinámicamente:

Si bien el administrador del broker puede crearlos con anticipación, se pueden crear nuevas colas y exchanges directamente en el código de productores y consumidores. De hecho, esta es una muy buena práctica para simular su creación y garantizar que los mensajes no se envíen a colas y exchanges inexistentes (Pivotal Software Inc., 2017f).

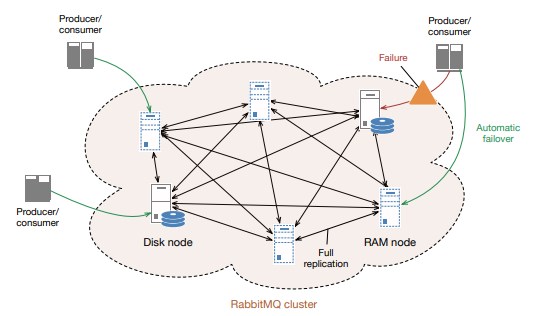
* Administración de permisos y soporte de SSL.

RabbitMQ requiere que los productores, consumidores y administradores se autentiquen antes de que puedan realizar cualquier operación en el broker (Pivotal Software Inc., 2017a). Se pueden otorgar diferentes permisos a diferentes usuarios, lo que permite la aplicación de un control de acceso eficiente y de gran precisión. SSL también es compatible y se puede utilizar para autenticar y cifrar los datos transmitidos a través de los canales entre los brokers, los productores y consumidores.

* Múltiples alternativas para replicación y balanceo de carga.

RabbitMQ posee diferentes alternativas para replicar configuraciones y contenido de las colas. Uno de ellos es la de conformar un Cluster de RabbitMQ, tal como se muestra en el siguiente diagrama:

***Figura 28****: RabbitMQ Cluster con configuración de espejo de todas las colas activada*



*Fuente: https://www.rabbitmq.com/clustering.html*

En este ejemplo, existe una replicación completa permanente entre todos los miembros. Se presentan dos tipos de nodos, nodos de tipo disco y RAM, que entre ellos forman el clúster.

Un nodo de tipo disco escribe la información del exchange, colas, sus contenidos, etc., a disco, mientras que un nodo RAM mantiene todo en la memoria (excepto los contenidos de la cola, si se les configura explícitamente que son duraderos o se establece un límite en tamaño). Los nodos RAM son por lo general más rápido, y los nodos de disco permiten la recuperación del clúster en caso de apagado completo (al menos uno es obligatorio). Los pares pueden insertar y extraer datos de cualquiera de los nodos, en cualquier momento, siempre que el nodo al que se enviaron los mensajes push esté todavía en funcionamiento (es decir, los mensajes se almacenan solo en el nodo destino del mensaje. Los mensajes no se replican en los nodos de manera predeterminada). Los enlaces entre los nodos que forman el clúster deben ser altamente confiables con bajas latencias para que funcione correctamente y se debe mantener una conexión bidireccional entre todos los nodos (Pivotal Software Inc., 2017c).

Una de las fortalezas inyectadas por el lenguaje Erlang (Erlang.org, 2016), es el manejo de la concurrencia dentro de las políticas de duplicación que ofrece RabbitMQ: el contenido de la cola se puede replicar a ninguno, una cierta cantidad fija o todos los miembros de un conjunto de réplicas, y puede proporcionar un nodo maestro en automático.

Se pueden agregar nuevos nodos sobre la marcha, ya sea para ponerse al día naturalmente (sin aprovisionamiento previo: a medida que las colas se vacían en los nodos antiguos, los nodos nuevos tendrán eventualmente los mismos contenidos que los otros nodos) o ser aprovisionado con la cola de datos antes de que el nodo se ponga en funcionamiento. Sin embargo, el aprovisionamiento puede implicar un tiempo de indisponibilidad, ya que el nodo debería tener exactamente los mismos contenidos que los demás. Debemos considerar que esta función de espejo no hace balanceo de carga por sí mismo: los pares publicarán y consumirán mensajes solo desde el nodo maestro, las réplicas se usan solo en caso de falla.

RabbitMQ también proporciona otras dos alternativas para compartir la carga entre varios brokers. Estos mecanismos alternativos reciben el nombre de Federación y pueden usarse para compartir o copiar mensajes de un broker a otro. Los mensajes pueden almacenarse temporalmente en el punto de entrada en caso de fallo del enlace de red y enviarse cuando vuelva a subir. Esas alternativas permiten un balance de carga parcial: uno puede federar solo algunas de las colas y mantener otras locales. Se puede encontrar más información sobre estos enfoques en las referencias (Pivotal Software Inc., 2017d).

#  Comparación de características generales entre Apache Kafka y RabbitMQ

En la siguiente tabla, se resumen las características de Apache Kafka y RabbitMQ documentadas en las secciones anteriores, consolidando los principales atributos de ambos productos. Los colores verdes y rojos fueron agregados para facilitar visualmente las mejores y peores valoraciones de cada atributo respectivamente, en blanco los que se consideran indistintos a efectos del alcance del proyecto.

***Tabla 3:*** *Evaluación de atributos clave de Apache Kafka y RabbitMQ*

|  |  |  |
| --- | --- | --- |
| **Característica** | **Apache Kafka** | **RabbitMQ** |
| Año creación | 2011 | 2007 |
| Licencia | Apache (Open Source) | Mozilla Public License  (Open Source) |
| Soporte Enterprise | No | Si |
| Lenguaje de Programación | Scala | Erlang |
| Sigue las normas del AMPQ | No | Si |
| Cobertura de Clientes Soportados | +18, incluyendo Java, .Net,  Python y Node.js | +24, incluyendo Java, .net, Python  y Node.js |
| Documentación, ejemplo y soporte de la comunidad[[27]](#footnote-27) | 6/10 | 9/10 |
| Principal soporte de almacenamiento | Disco | Memoria (RAM) |
| Persistencia en disco | Completo y obligatorio | Temporal y opcional |
| Borrado de mensajes | Luego de alcanzar cierto límite o cierto tamaño | Inmediatamente luego de confirmar el consumo del mensaje |
| Compresión mensajes | Si | No |
| Colas predefinidas en el broker | Si | Si |
| Múltiples consumidores del mismo set de datos | Si | Si |
| Balance de carga entre consumidores | Si | Si |
| Definición remota de Colas | No | Si |
| Enrutamiento avanzado de mensajes | Solo a particiones | Si |
| Soporte de lista de control de acceso | No | Si |
| Soporte SSL | Si | Si |
| Soporte Cluster | Si | Si |
| Autosuficiente | No, requiere de Apache  Zookeeper | Si |
| Interface de administración y monitoreo | JMX y Clientes de terceros | Si, Web y cliente de terceros |
| Facilidad de Instalación y configuración | 6/10 | 8/10 |

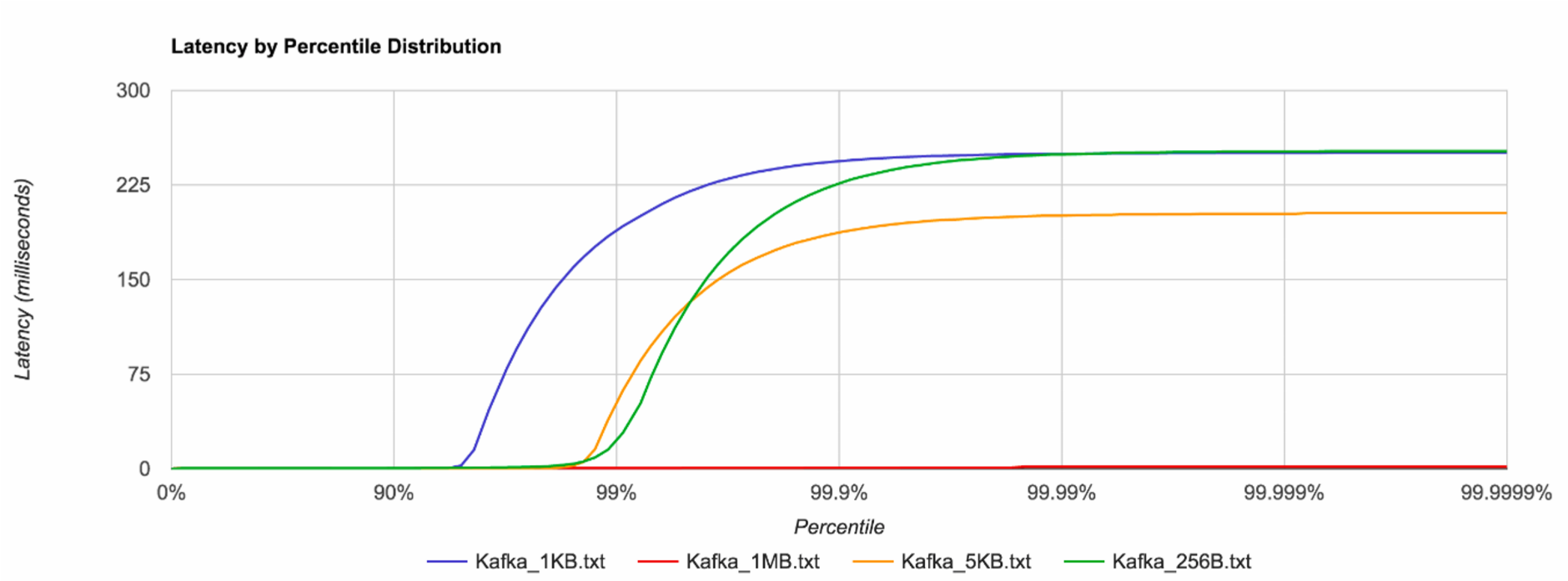
*Fuente: elaboración propia*

**Performance**:

Apache Kafka posee un rendimiento de procesamiento de mensajes en el orden de los 100k/seg. Si bien esto podría ser clave para la selección del broker, las tasas de mensajes por segundo son difíciles de establecer y cuantificar ya que dependen tanto de su entorno y hardware, el tipo de carga de trabajo, las garantías de entrega que se utilizan (por ejemplo, el persistente es costoso), etc. Por otro lado, en RabbitMQ se pueden enviar fácilmente 20K mensajes por segundo a través de una sola cola, de hecho, con pocas garantías. La cola está respaldada por un único subproceso liviano de Erlang que se programa de forma cooperativa en un grupo de subprocesos nativos del sistema operativo, por lo que se convierte en un cuello de botella natural, ya que una sola cola nunca hará más trabajo que los ciclos de la CPU (Pivotal Software Inc., 2017).

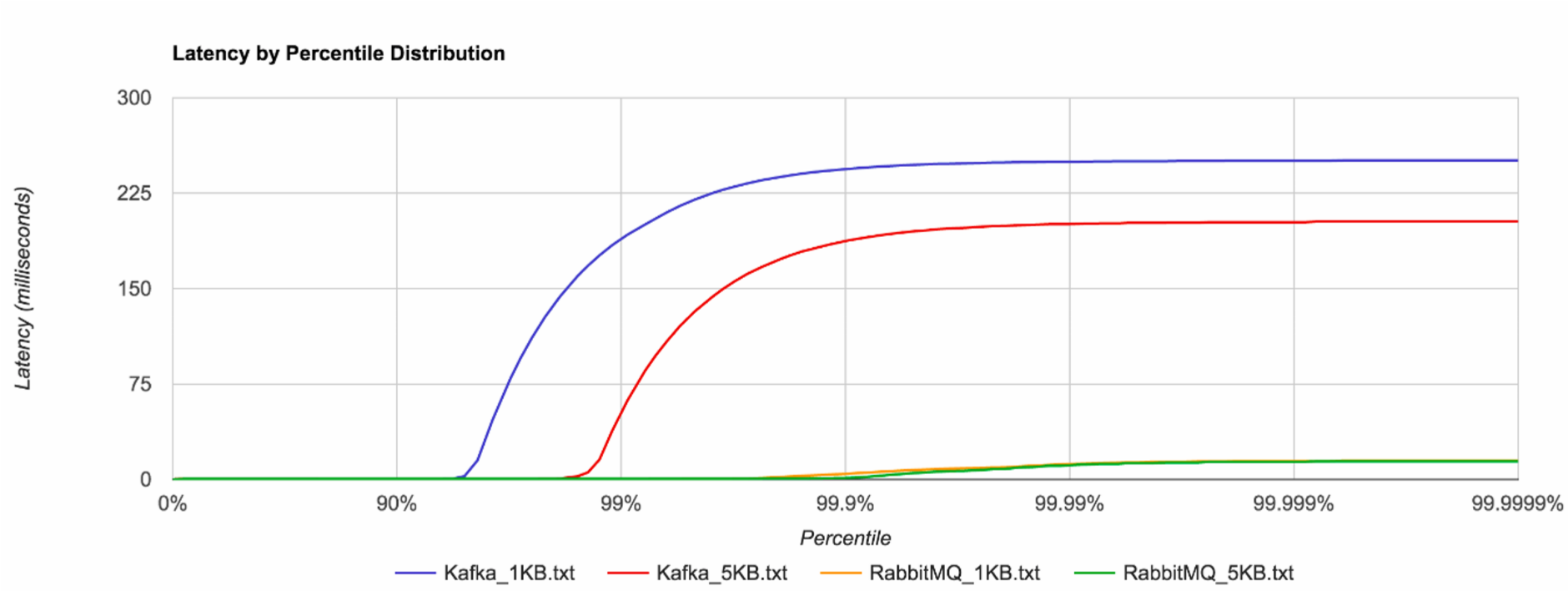
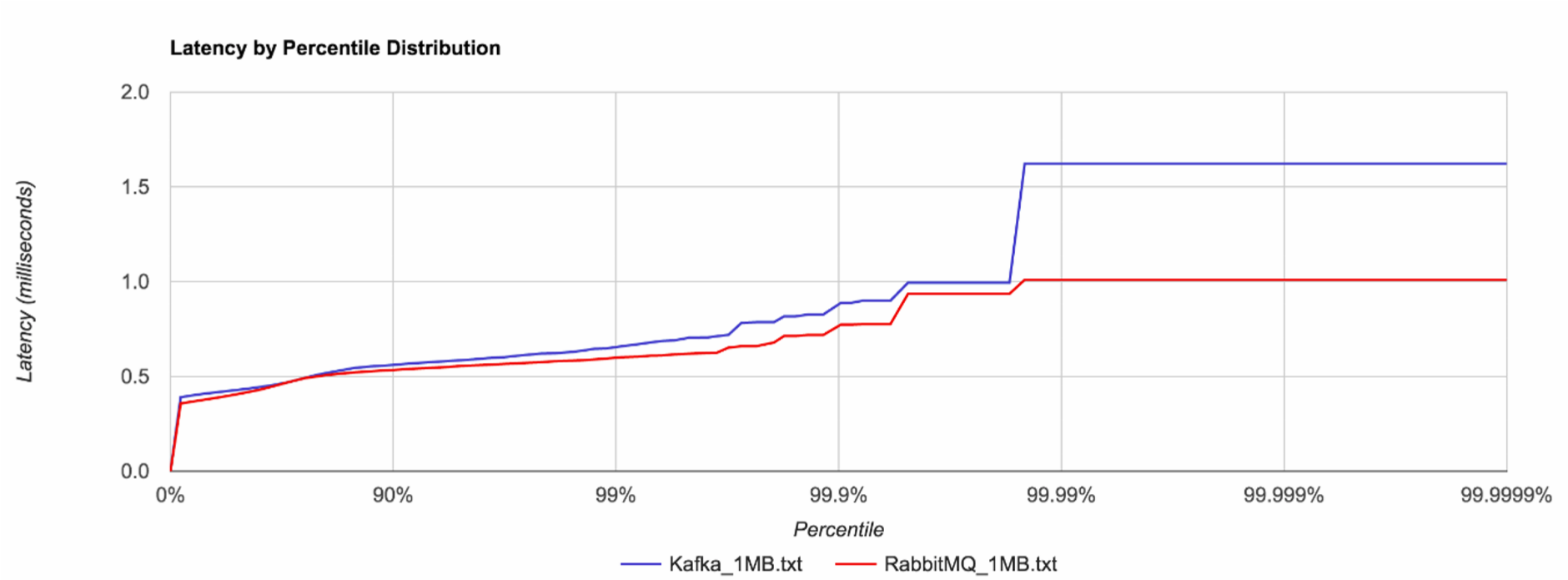
Se han realizado y publicado mediciones de latencia comparando ambos productos. En la siguientes imágenes se puede observar las mediciones publicadas de latencia con mensajes de tamaño variable (Tyler Treat, 2016), con y sin persistencia activada:

***Figura 29****: Medición de Latencia Apache Kafka Sincrónico*



*Fuente: https://bravenewgeek.com/tag/benchmarking/*

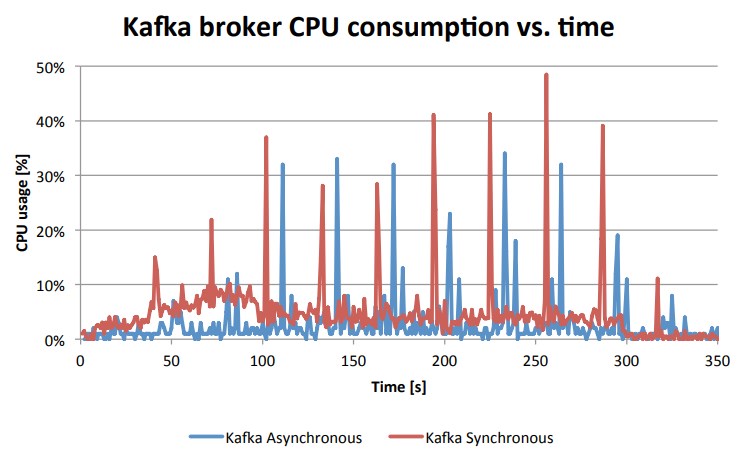
***Figura 301****: Medición de Latencia RabbitMQ con PersistenciaComparativa de latencia en mensajes d 1MB*



*Fuente: https://bravenewgeek.com/tag/benchmarking/Fuente: https://bravenewgeek.com/tag/benchmarking/*

En la gráfica anterior se puede observar una diferencia de comportamiento cuando los mensajes son más grandes (1 MB). Esto puede deberse en principio por el mecanismo de buffer a nivel sistema operativo o incluso en el mismo producto, los mensajes más grandes producirán mayor frecuencia de vaciado del buffer, tal como se puede observar en los picos de la siguiente gráfica para la medición de consumo de CPU de Kafka:

***Figura 31****: Consumo de CPU Apache Kafka en modalidad sincrónica y asincrónica*

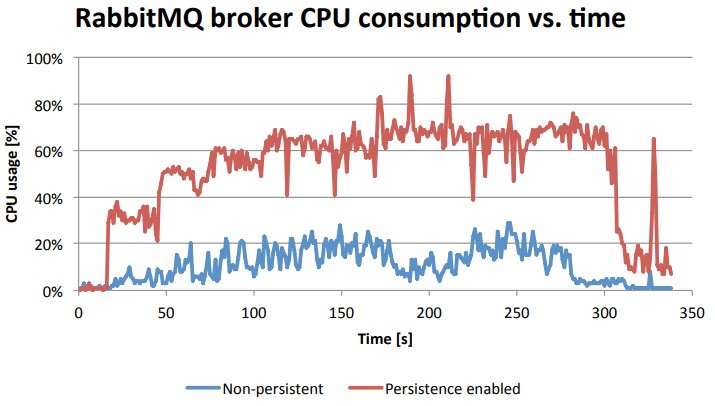


*Fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

Kafka no genera alta latencia por persistir en disco, dado que los mensajes son previamente llevados a la caché y posteriormente persistidos en disco de forma asincrónica, cómo lo indica Nicolás Nanonni en su tesis de maestría(Nannoni, 2015), la mejor performance se obtiene utilizando RabbitMQ sin persistencia.

Para ilustrar el punto anterior, se muestra en el siguiente gráfico la diferencia de consumo de CPU de RabbitMQ con persistencia y sin persistencia activada:

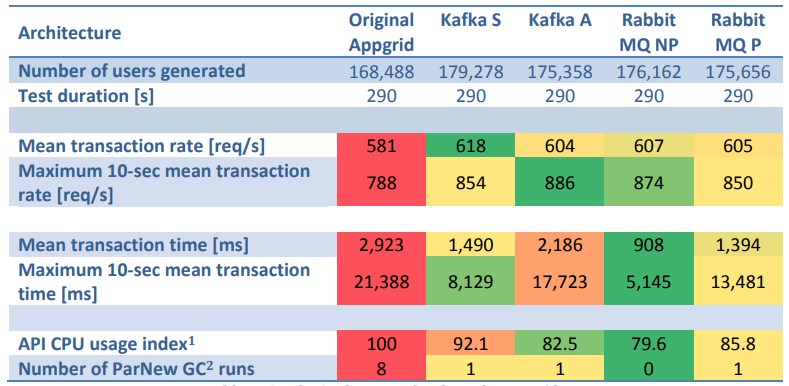
***Figura 32****: Comparativa de consumo de CPU RabbitMQ con persistencia y sin persistencia activada*



*Fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

A continuación se muestra una síntesis de test de performance de referencia(Nannoni, 2015) que compara los distintos valores obtenidos con 4 configuraciones: Kafka S(sincrónico), Kafka A(asincrónico), RabbitMQ NP (sin persistencia), RabbitMQ P (con persistencia). Los mejores resultados en verde, los menos favorables en rojo.

***Figura 33****: Resumen de resultados de performance.*



*Fuente: N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures,” 2015.*

Aumentar los mensajes por segundo a menudo se reduce a explotar adecuadamente el paralelismo disponible en el entorno haciendo cosas tales como interrumpir el tráfico a través de múltiples colas mediante el enrutamiento inteligente (para que se puedan ejecutar diferentes colas al mismo tiempo). En ambas tecnologías, incrementar la capacidad de mensajes básicamente se logra mediante el uso de Clusters, en el caso de RabbitMQ la recomendación va por formar cluster’s de entre tres y siete nodos (Pivotal Software Inc., 2017).

Los indicadores de performance presentados no representan una evidencia determinante para predecir cómo se comportarán los productos evaluados en un ambiente productivo; no solo por las diferencias en capacidades de la infraestructura en sí, sino también por la diferencia en el comportamiento generado por los sistemas productores de información que emiten los mensajes a procesar por los brokers de forma aleatoria y con picos de concurrencia propias del negocio.

#  Conclusiones preliminares y selección del broker

Apache Kafka y RabbitMQ son dos Middlewares orientados a mensajes que resuelven de manera sólida los requerimientos básicos que se necesitan para la recepción, transporte y distribución de alto volumen de mensajes desde los productores a los consumidores. En cuanto a la performance documentada ambos poseen excelentes resultados y estos están dentro de los volúmenes de mensajes que se manejarán en la solución. Sin embargo a efectos del proyecto en particular con el escenario de negocio y problemática planteada, hay otros aspectos determinantes que nos orientan en la selección de RabbitMQ como el broker para el proyecto.

Estos aspectos son:

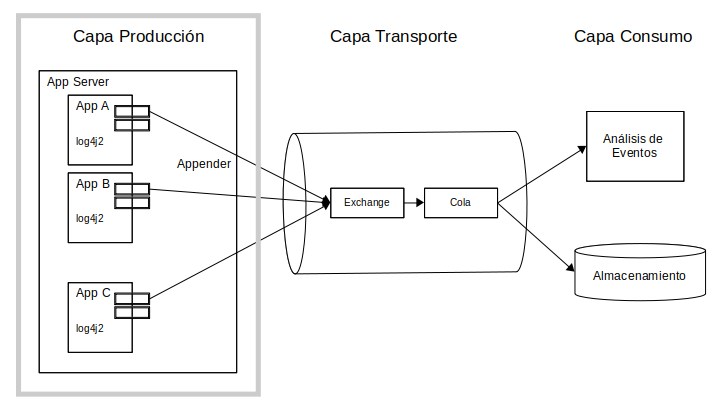
* Manejo de mensajes en memoria por sobre disco: Una de los valores agregados que se quiere explotar es el manejo de mensaje en RAM por sobre disco. Esto inyectará mayor ahorro de recursos a la solución integral, delegando posteriormente la persistencia en disco a agentes especializados como bases de datos orientadas a documentos y manejadores de reglas para el análisis de los mensajes.
* Soporte Empresarial: En el marco cultural de la empresa donde estará funcionando, es determinante el nivel de soporte de las soluciones planteadas. Esto facilitará la aceptación del proyecto mitigando riesgos y generando mayor confianza en los aspectos de soporte técnico sobre la herramienta.
* Interfaces de administración y monitoreo/ Control de acceso: Ambos componentes son de alto valor dentro de un ambiente empresarial corporativo. Las políticas de las áreas de tecnología requieren de herramientas de federación para la administración, monitoreo y control de la herramienta así como para su gestión desde el área de Seguridad Informática para los accesos y permisos.
* Documentación y soporte por la comunidad: Si bien ambas se encuentran documentadas en la web, en el caso de Apache Kafka, el depender de un segundo producto (Zookeeper), hace una curva de aprendizaje más grande con respecto a RabbitMQ, y por consiguiente más complejo su instalación, administración, control y monitoreo. Por tal motivo, la documentación precisa de los componentes y sus funcionalidades, de su instalación y operatoria, es un factor muy importante en el inicio y en el post puesta en marcha para el tunning y performance y por consiguiente para el éxito del proyecto.
* Enrutamiento de mensajes: La capacidad de poder incorporar reglas de enrutamiento de los mensajes en función de su contenido es una funcionalidad de RabbitMQ que le agrega valor a la solución propuesta para el manejo de los logs de las aplicaciones y su delivery posterior para procesamiento de los eventos.

# Construcción de prototipo de la solución

La solución se divide en tres capas funcionales: producción, transporte y consumo. Se detalla a continuación los componentes, su instalación, configuración y ensamblado para el prototipo de la solución propuesta.

Para ubicarnos en el contexto de la solución, se diagrama de forma simplificada las tres capas y sus componentes. Iniciaremos desarrollando la capa de producción:

***Figura 34****: Diagrama simplificado de las 3 capas funcionales de la propuesta*



*Fuente: elaboración propia*

## Capa de producción

Es la capa representada por las producción de logs por parte de las aplicaciones de negocio y arquitectura mediante la ejecución de procesos, funciones y consultas sobre el sistema propias del negocio que representan. Estos logs contienen información detallada y contextual: parámetros de entrada, nombre de los métodos y clases que intervienen, detalle de las líneas del código donde se produce una excepción, el tipo de excepción, la posición dentro de la estructura del código, tiempos de duración, entre otros. A continuación se listan y detallan las actividades a realizar para esta capa.

• Se instala y configura log4j2 en el proyecto como framework de logueo (Apache

Software Fundation, 2014):

Importaremos al pom del proyecto las librerías de log4j2 versión estable: 2.1

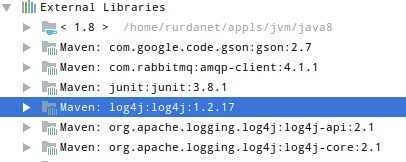
***Figura 35****: Configuración de Log4j v2.1 en el pom del proyecto*



*Fuente: elaboración propia*

Posterior a la modificación del pom, validaremos que las dependencias indicadas fueron descargadas a la sección de librerías del proyecto:

***Figura 36****: Validación de librerías de log4j2 correctamente importadas al proyecto*



*Fuente: elaboración propia*

Se configura el log4j para recibir los logs de la aplicación y enviarlos al broker(RabbitMQ). Esta configuración se realiza en el archivo log4j.xml que localizaremos a efectos de la POC en la carpeta resources del proyecto para que sea importado automáticamente al classpath del mismo al compilarse.

Como puntos importantes de la configuración caben resaltar: la configuración del método asincrónico de logueo especificado en la línea 29, las características del Exchange del broker (línea 34 a la 44) al cuál se enviarán los mensajes, y la plantilla (layout) con el

manejo del formato de salida del String para ser tratado en formato JSON

personalizado(líneas 45 a 48), tal como se muestra a continuación:

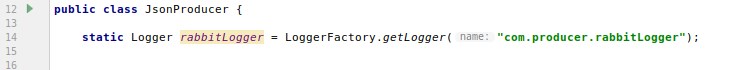
***Figura 37****: Configuración de Appender de Log4j2 para RabbitMQ*



*Fuente: elaboración propia*

En la clase donde se desee loguear mediante esta nueva estrategia, se instancia el logger de la clase con el configurado en el log4j.xml, tal como se muestra a continuación:

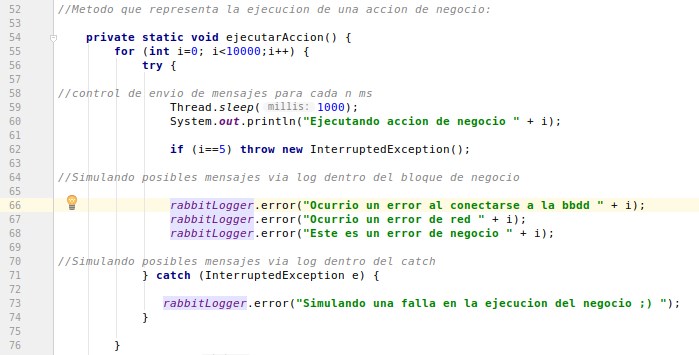
*Figura 38: Iniciación del logger configurado para RabbitMQ*



*Fuente: elaboración propia*

En el método y línea que deseemos capturar un mensaje para loguear simplemente invocaremos al método que implementa log4j para este fin, en este caso la notificación de un error:

***Figura 39****: Código de prueba para simular el logueo de eventos de error del sistema*



*Fuente: elaboración propia*

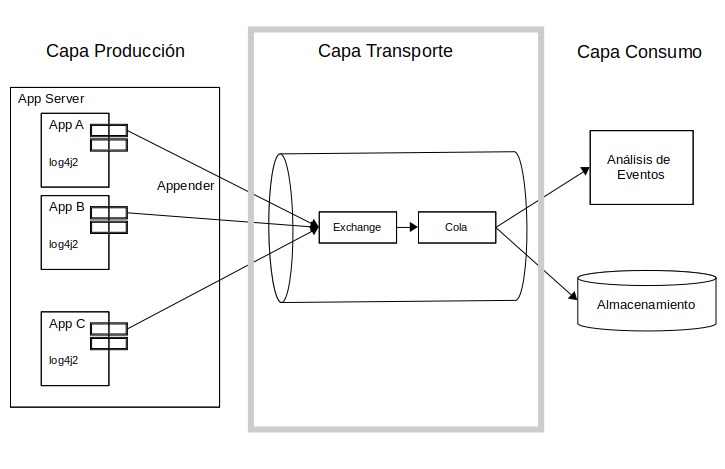
En este punto ya tenemos lo necesario para producir los mensajes y enviarlos al broker. En la siguiente sección, el detalle de la capa de transporte para recibir y procesar los mensajes a enviar.

## Capa de transporte

En esta capa es donde participa el broker como componente receptor de todos y cada uno de las líneas de logs producidas en la Capa I. Por su rol, es un componente que está diseñado y construido para poder recolectar altos volúmenes de información con baja

latencia.

***Figura 40****: Diagrama Simplificado, enfoque en Capa de Transporte*



*Fuente: elaboración propia*

A continuación se detalla la lista de actividades necesarias para la instalación, configuración y puesta en funcionamiento del broker.

###### Instalación del broker (RabbitMQ)

Desde la web oficial de pivotal, se puede acceder a la sección de instalación del broker para las distintas plataformas posibles (Pivotal Software Inc., 2017e). A efectos de nuestro prototipo, lo instalaremos siguiendo las instrucciones para Linux distribución Ubuntu. RabbitMQ está incluido en los repositorios oficiales de Debian y Ubuntu. Con el siguiente comando se instala tanto el broker como sus dependencias necesarias:

sudo apt-get install rabbitmq-server

Realizado este paso, lo siguiente es activar el broker como servicio:

sudo service rabbitmq-server start

Iniciado el servicio, procederemos a la activación de la consola de administración web de rabbitMQ:

sudo rabbitmq-plugins enable rabbitmq management

El broker crea por defecto un usuario y password (guest). Este sirve a efectos de pruebas y accesos locales (localhost). Para ambientes productivos será necesario la creación y configuración de seguridad requerida para cada caso particular.

Para consultar el estado del servicio, se puede ejecutar la siguiente instrucción:

rabbitmqctl status

Para detenerlo simplemente ejecutaremos:

rabbitmqctl stop

**Configuración del broker:**

El siguiente paso será entrar a la consola de administración, previamente instalada en el paso anterior y procederemos a configurar los principales componentes: el Exchange y la cola (queue por su nombre en inglés) .

Para eso, abriremos nuestro browser e ingresaremos la dirección: localhost:15672 Ingresaremos el usuario y contraseña creados por defectos en la instalación (guest/guest)

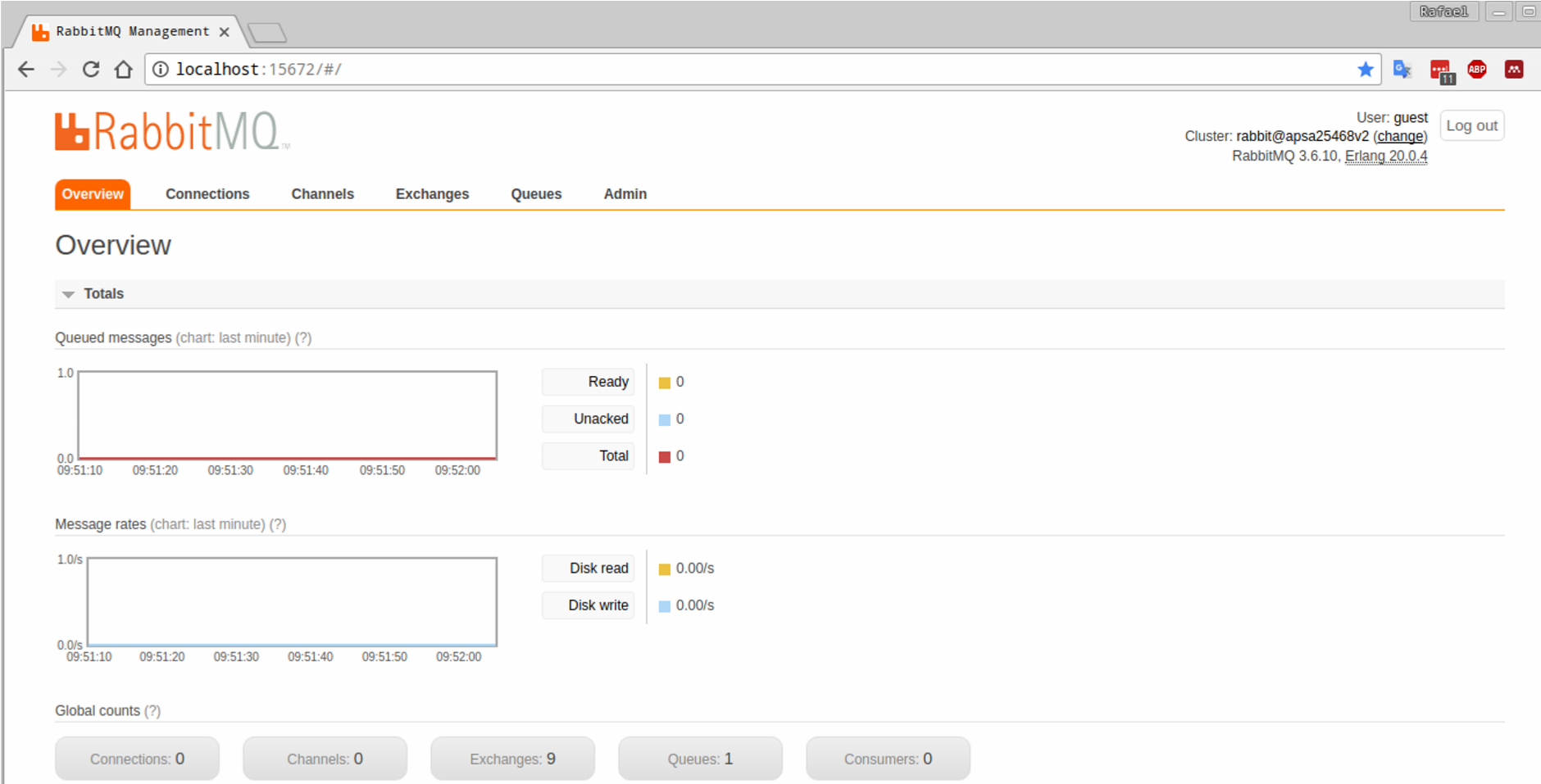
***Figura 41****: Acceso principal a la consola de administración de RabbitMQ*



*Fuente: elaboración propia*

Una vez dentro de la consola veremos una sección de vista general que muestra las estadísticas básicas del sistema que se actualizan de forma automática:

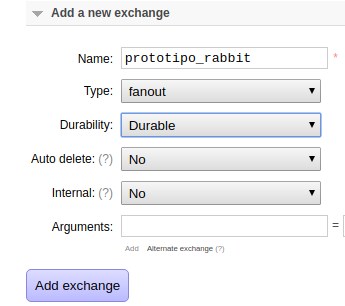
***Figura 42****: Sección principal consola de Administración RabbitMQ*



*Fuente: elaboración propia*

Para configurar el componente del broker que servirá de punto de entrada seleccionamos en el menú principal la sección de Exchanges y luego veremos un área para crear uno nuevo. A efectos de la POC ingresamos los valores como se muestra a continuación y presionamos el botón Add exchange:

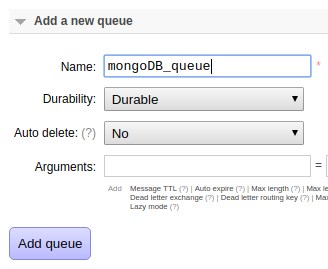
***Figura 43****: Creación de Exchange en RabbitMQ*



*Fuente: elaboración propia*

Una vez configurado el Exchange, procederemos a configurar y asociarle una cola. Para ellos seleccionaremos la opción en el menú llamado “Queues” y luego “Add a new queue”, ingresaremos los datos que se muestran a continuación:

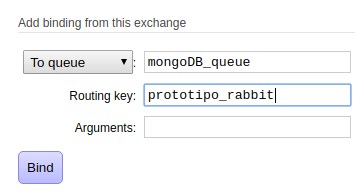
***Figura 44****: Creación de Cola en RabbitMQ*



*Fuente: elaboración propia*

Esto creará la cola. Luego para asociarla al exchange, seleccionamos la cola creada y en la sección de Bindeo, llenamos los datos como se muestra a continuación, indicando el nombre del Exchange y la clave de enrutamiento:

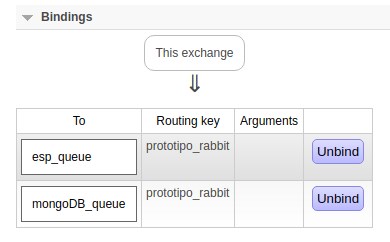
***Figura 45****: Bindeo de cola a Exchange de RabbitMQ*



*Fuente: elaboración propia*

Una vez realizado este procedimiento, haremos lo mismo para una segunda cola llamada esp\_queue, que será consumida por un segundo cliente. Al finalizar veremos el status de las colas asociadas al exchange de la siguiente manera:

*Figura 46: Colas bindeadas al exchange de RabbitMQ*

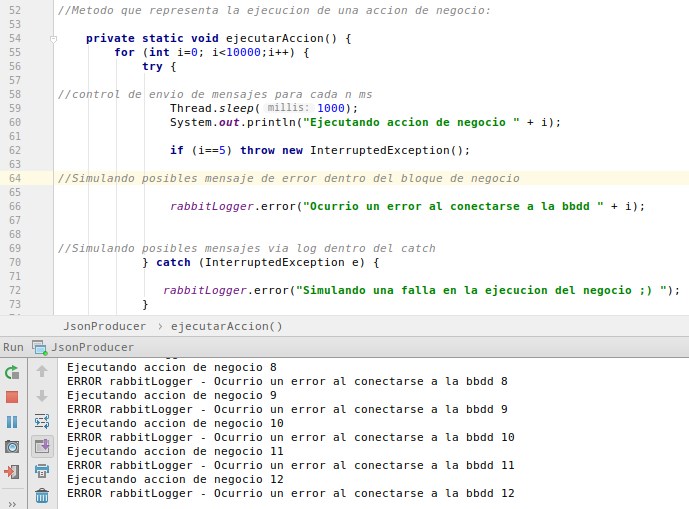


*Fuente: elaboración propia*

En esta parte de la instalación, podemos hacer una primera prueba de conectividad desde el productor hacia la capa de transporte (broker). Para ello ejecutaremos una clase java de test enviando de forma controlada mensajes cada 1 segundo, de manera de poder verificar la recepción y las estadísticas que ofrece rabbitMQ.

El método de test, simulará el envío de mensajes tanto de negocio como de errores, en intervalos controlados de 1 segundo (1000 ms), tal como se muestra en la siguiente imagen:

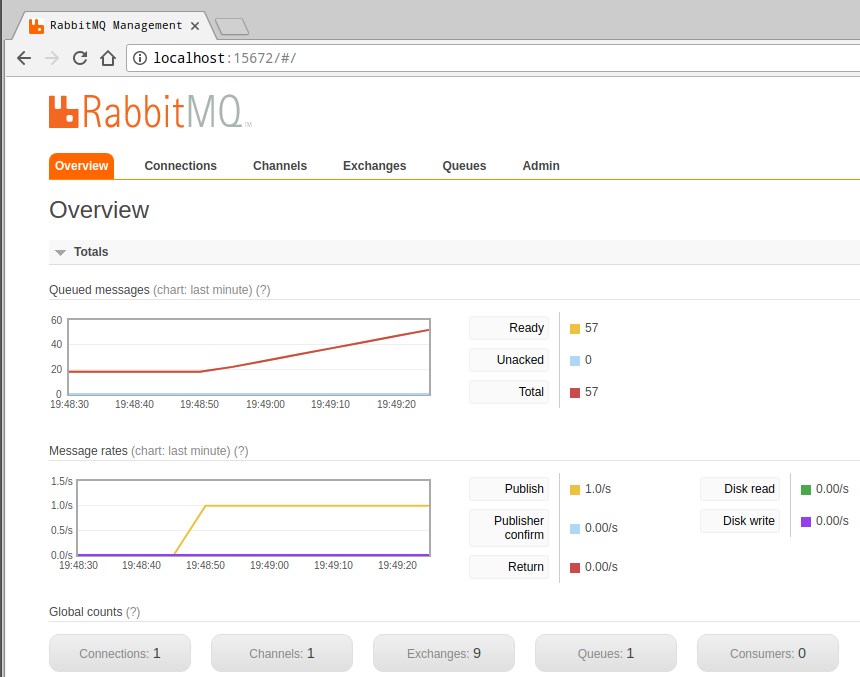
***Figura 47****: Método Java de Prueba para envío de mensajes al broker mediante appender log4j2*



*Fuente: elaboración propia*

Mientras se ejecuta el método de test, consultamos la sección de Overview del panel de administración de RabbitMQ y podemos observar las estadísticas de recepción de mensajes al broker:

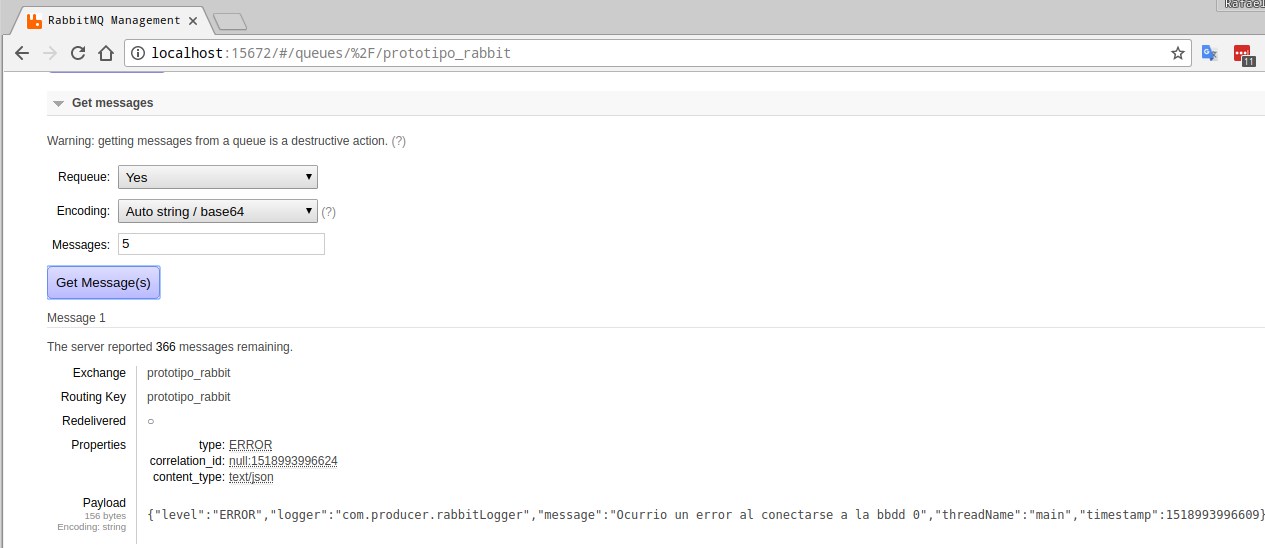
***Figura 48****: Estadísticas de recepción de mensajes en RabbitMQ*



*Fuente: elaboración propia*

En la sección de colas (Queues), seleccionaremos la creada y consultaremos algunos mensajes enviados para comprobar tanto su forma como su contenido, indicando la cantidad de mensajes a consultar de la cola y presionando el botón Get Message(s):

*Figura 49: Consulta de mensajes desde la consola de administración de RabbitMQ*



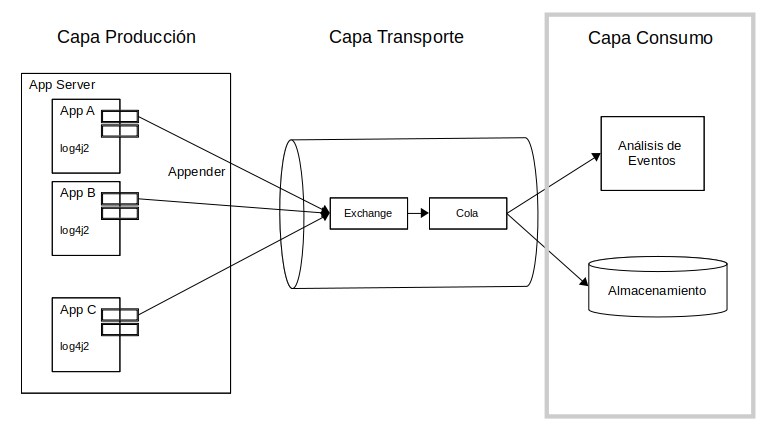
*Fuente: elaboración propia*

Como podemos apreciar en la imagen anterior, el mensaje llegó con el formato Json esperado, así como con los atributos necesarios para almacenarlo y consumirlo posteriormente en la capa de almacenamiento y explotación que detallaremos en la siguiente sección.

## Capa de explotación y consumo

Para la capa de explotación y consumo, a efectos de la POC, instalaremos y configuraremos dos clientes con alto potencial de valor para la solución: Una base de datos noSql (mongoDB) para analizar y consultar los eventos de forma independiente del resto del sistema y un manejador de eventos para analizar de forma automatizada los mensajes y emitir determinadas alertas en función a su contenido y las reglas que definamos para ello. La selección de estas herramientas son para uso exclusivo de la POC y buscan únicamente mostrar el potencial de este tipo de herramientas, de ninguna manera implica una valoración con respecto a otras disponibles en el mercado.

***Figura 50****: Capa de Consumo en el diagrama de componentes*



*Fuente: elaboración propia*

Instalación de MongoDB:

El equipo de MongoDB indica en la sección de instalación (MongoDB Inc., 2017b), que la versión que se encuentra en los repositorios de Ubuntu no son mantenidas por ellos, por lo que la sugerencia es instalarlos desde los siguientes repositorios, agregándolos de la siguiente manera:

* Importamos la clave pública de MongoDB:

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 –recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5

* Agregamos el repositorio

echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu

xenial/mongodb-org/3.6 multiverse" |

sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list

* Posteriormente se realiza el update del repositorio y la instalación del paquete:

sudo apt get-update sudo apt install mongodb-org

* Finalizada la instalación, procederemos a levantar el servicio de mongoDBsudo service mongod start

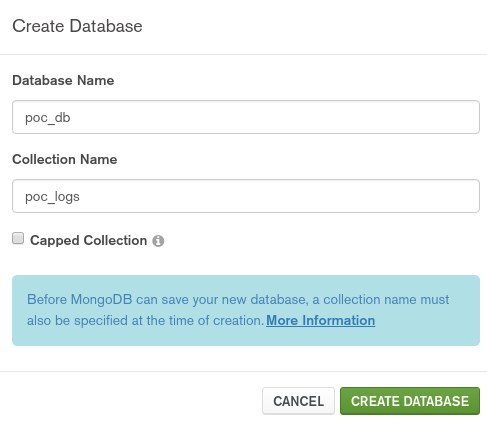
A efectos de la POC crearemos una base de datos y una colección en MongoDB. Para interactuar con el motor de la Base de datos usaremos MongoDB Compass versión Comunity, la cual puede ser descargada e instalada de forma gratuita desde el link de la referencia (MongoDB Inc., 2017a).

Seleccionaremos un nombre para la base de datos y un nombre para la colección:

poc\_db y poc\_logs respectivamente:

***Figura 51****: Creación de base de datos y colección en*

*MongoDB*

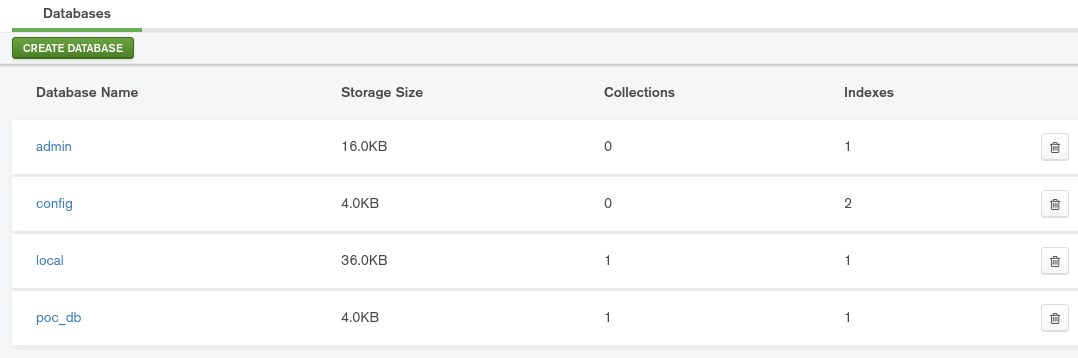


*Fuente: elaboración propia*

Una vez creada la veremos en la lista de las bases de datos configuradas en

MongoDB:

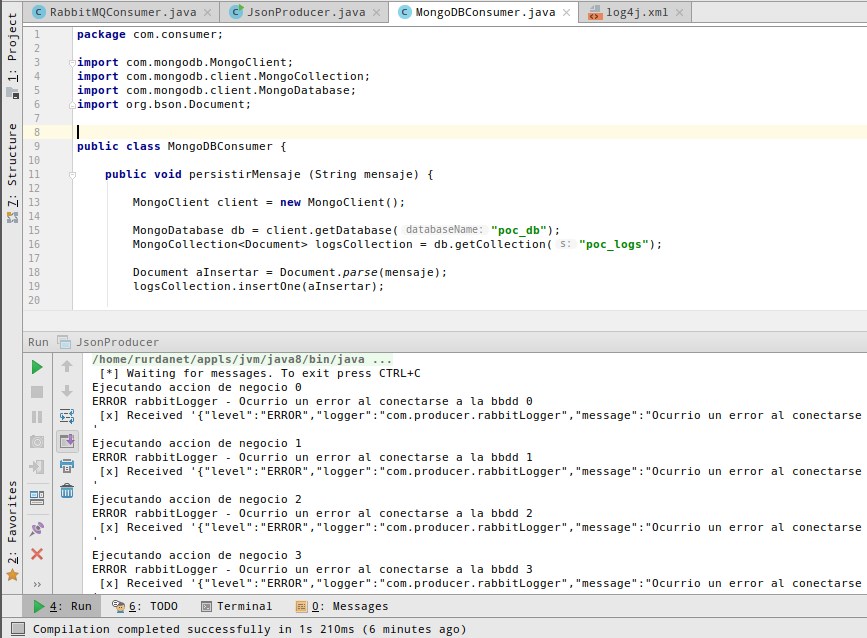
***Figura 52****: Vista de bases de datos en MongoDB*



*Fuente: elaboración propia*

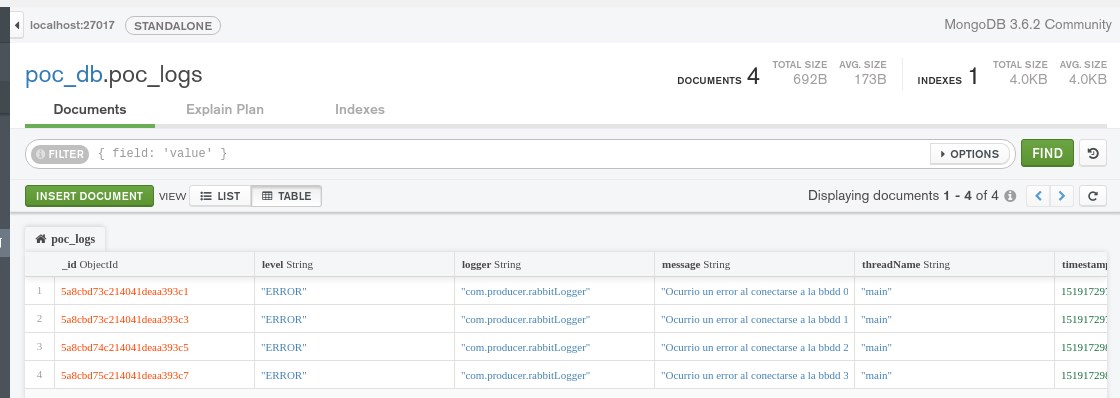
Ahora ejecutaremos nuestra clases de test para capturar los mensajes de la cola de salida de RabbitMQ y enviarlos a la base de datos de MongoDB y así validar el envío desde el productor al broker y desde el broker a la base de datos noSql. A continuación parte principal del código fuente del cliente para MongoDB y el resultado del envío desde el productor al broker y a MongoDB:

*Figura 53: Código fuente cliente MongoDB y ejecución de logs para envío al broker y a la bbdd*



*Fuente: elaboración propia*

***Figura 54****: Registros provenientes del broker persistidos en MongoDB*



*Fuente: elaboración propia*

En este punto tenemos ya un primer cliente escuchando y consumiendo los mensajes provenientes del broker. Este mecanismo a efectos de la solución propuesta representa uno de los valores a agregar para el análisis posterior de eventos y logs.

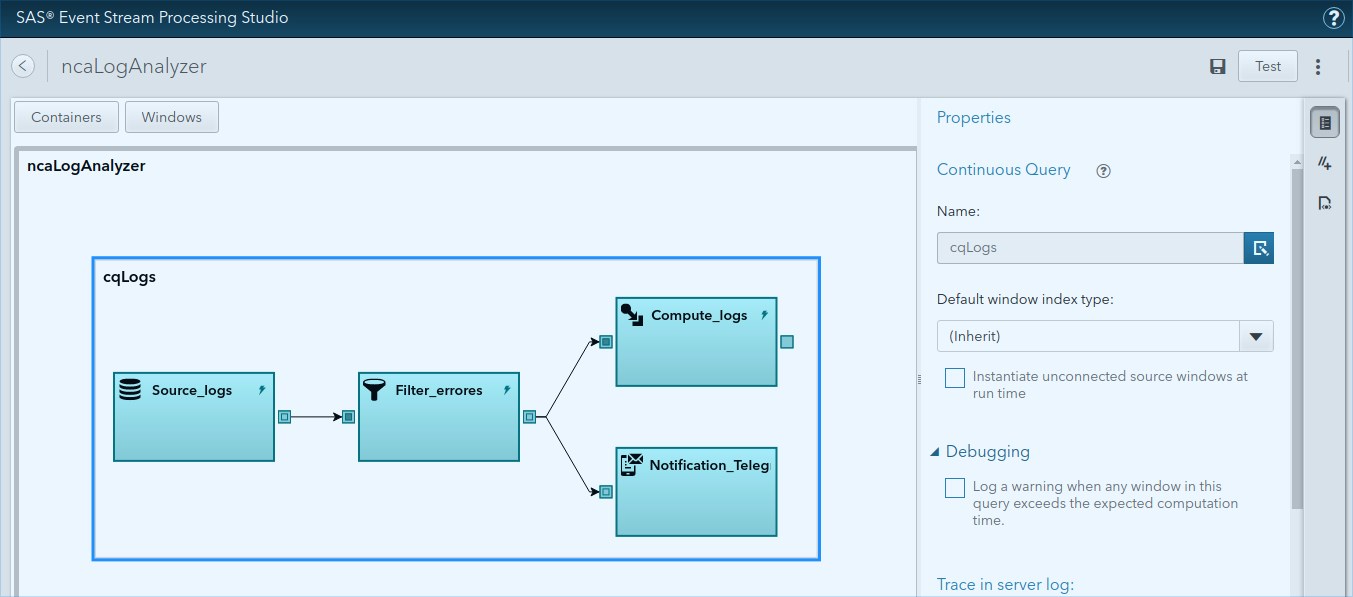
Disponer de un broker nos brinda la capacidad de poder incluir varios clientes consumidores que reciban una copia del mensaje generado y que puedan tomar distintas acciones al respecto en función a una estrategia. En esta línea de ideas, incorporaremos un segundo cliente que reciba el mensaje, analice el contenido y en función a ciertas reglas ejecute una acción, que a efectos de la POC será el envío de un mensaje de alerta a un dispositivo mobile informando que ha ocurrido una falla importante en el sistema y que debe ser atendido.

Para esto reutilizaremos una herramienta disponible en la organización del caso de estudio encargada de recibir eventos y procesarlos en función a una regla determinada. Este producto es el Event Processing de SAS (SAS Inc., 2017). Dado que la herramienta ya está disponible, obviaremos el proceso de configuración e instalación, sin embargo en la referencia puede encontrarse documentación al respecto.

El diseño del flujo para el análisis de los mensajes incluye una ventana de entrada, un filtro para analizar el contenido en busca de mensajes que contengan cierto contenido, una ventana de computo para procesar la salida hacia un mecanismo de persistencia propio del manejador de eventos y una ventana de notificación a un bot de telegram(Telegram.org,

2017):

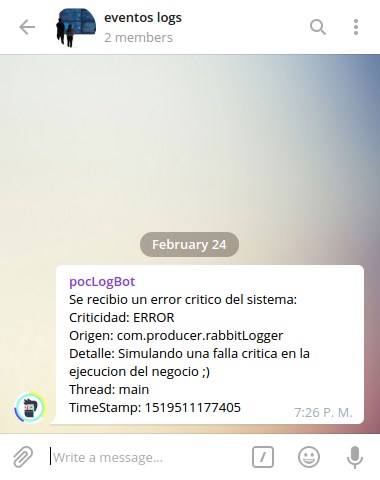
*Figura 55: Diagrama de flujo de análisis de eventos en el SAS ESP*



*Fuente: elaboración propia*

El analizador de eventos podrá disponer de distintas estrategias de comunicación y acción en función al contenido de los mensajes recibidos de los logs de los productores, en el ejemplo de la POC, podemos observar una ventana emergente donde un Bot comunica en near real time el evento crítico encontrado en el sistema a un grupo de Telegram (chat), en el que pueden estar incluidos el personal de mesa de ayuda o de algún otro nivel de soporte dependiendo del tipo de criticidad o área de negocio que se encuentre afectada por el mensaje.

***Figura 56****: Notificación automática al cliente telegram Desktop/Mobile vía bot*



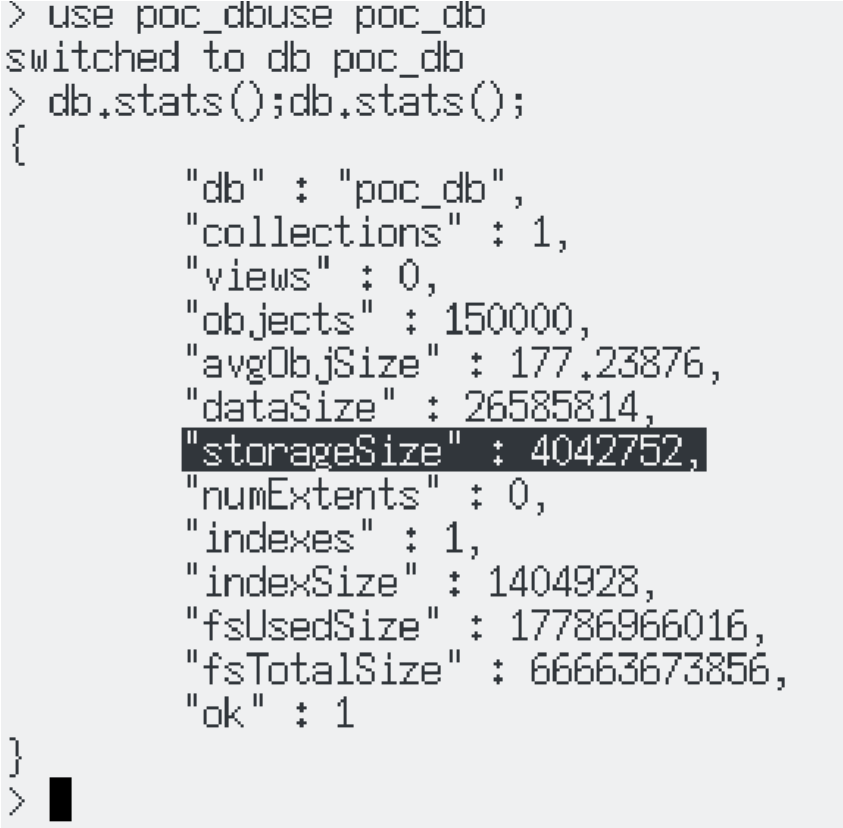
*Fuente: elaboración propia*

## Evaluación del modelo presentado

### Análisis comparativo de costo de almacenamiento y procesamiento del modelo actual y el propuesto.

Para este análisis tomaremos como base un archivo de logs que se genera en el sistema actual. Como vimos en la sección 3.3, los logs rotan cada 10 mb.

Analizando el contenido del mismo,

***Figura 57****: Estadísticas de consumo de storage para 150000 mensajes persistidos en mongoDB* encontramos un promedio de 150.000 líneas de mensajes. Para la primera comparación de almacenamiento, se simuló la generación de la

misma cantidad de mensajes, y su

correspondiente persistencia en la base de datos noSql de MongoDB. Posteriormente se solicitó la información estadística al motor de Mongo, y los resultados obtenidos como se muestra en la

*Fuente: elaboración propia* siguiente imagen, nos indican que para la misma cantidad de registros, el motor necesitó consumir 4.042.752 bytes, es decir 4 megas:

**DataSize:** representa el tamaño en bytes de los datos sin comprimir almacenados en esta base de datos.

**StorageSize:**

Es la cantidad total de espacio en disco asignado a todas las colecciones en la base de datos. En resumen, la diferencia de formato, sistema y modelo representa una reducción aproximada de consumo de disco de un 60%.

***Figura 58****: Comparativa de consumo de disco (megaBytes) entre log plano y log persistido en base NoSql MongoDB (150.000 eventos)*

Consumo de Disco

Log en archivo Plano Vs Base NoSql MongoDB

Storage(mb)

0

2

4

6

8

10

12

FileSystemLog

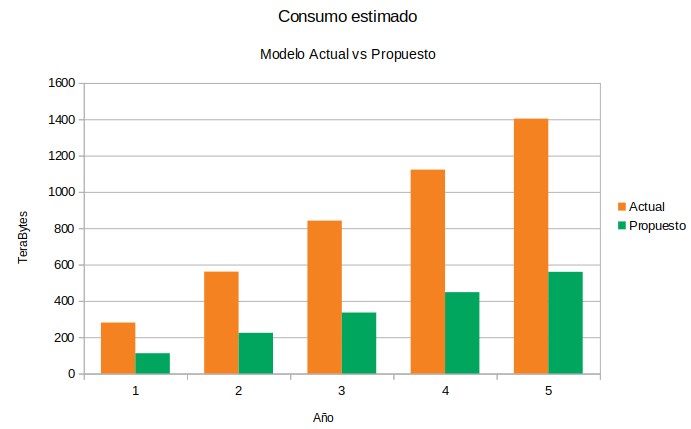
MongoDB

*Fuente: elaboración propia*

El nivel de ahorro de almacenamiento en disco y en RAM va a depender del engine y el algoritmo de compresión que se configure en el motor de base de datos(MongoDB Inc., 2017). En este caso estamos utilizando el que viene por defecto en el motor.

Haciendo el ejercicio para estimar el espacio en disco necesario para lograr el objetivo de disponer información histórica almacenada por 5 años, los valores en comparación con el modelo propuesto se reducen un 60%, tal como se muestra a continuación en el siguiente gráfico:

***Figura 59****: Comparativa de estimado de almacenamiento para 5 años*



*Fuente: elaboración propia*

El almacenamiento en disco es un recurso crítico para cualquier sistema de base de datos escalable. El rendimiento de la base de datos basadas en disco dependerá de cómo se gestionen los datos en el disco. El servidor MongoDB admite varios motores de almacenamiento conectables que manejan la gestión del almacenamiento y almacenan inicialmente todos los documentos secuencialmente. A medida que la base de datos crece y se ejecutan varias operaciones de escritura, este espacio contiguo se fragmenta en bloques más pequeños con espacios libres entre ellos. La solución típica es aumentar el tamaño del disco; sin embargo, existen alternativas que pueden ayudarlo a recuperar el espacio libre sin tener que escalar el tamaño del disco. Una cosa importante a tener en cuenta son las estadísticas de almacenamiento de MongoDB y cómo puede compactar o reparar la base de datos para manejar la fragmentación.

### Comparativa de tiempos para la obtención de información relevante en ambos modelos.

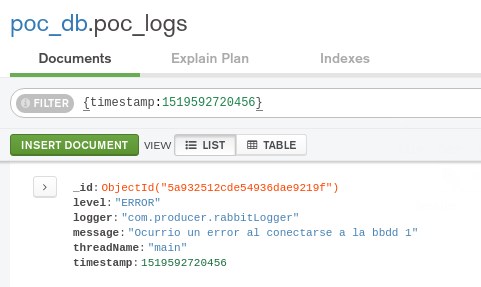
Para obtener información relevante en el modelo actual, se deben ejecutar no menos de 8 acciones descriptas en la sección 3.5. Con la práctica adecuada, los tiempos para obtener la información requerida puede llevar entre 10 y 15 minutos, y en caso de no tener la experiencia necesaria, podría llevar 30 minutos aproximadamente.

La solución propuesta involucra dos estrategias base para la obtención de información relevante de los eventos:

1. Procesamiento en modelo de datos orientado a Documentos (noSql) para los eventos
2. Análisis, detección y notificación automática de eventos críticos del sistema.

Analizar el log en búsqueda de detalle para determinar causa y solución en el caso 1 implicaría el uso de una simple query que recupere en milisegundos los datos relevantes del log persistido. Las consultas pueden ser por un campo específico o por la coincidencia de algún valor o parte de él en el registro, tal como se muestra en la siguiente imagen:

***Figura 60****: Ejemplo de consulta de un evento en MongoDB*



*Fuente: elaboración propia*

En el caso 2 (detección automática) se agrega un factor de digitalización de procesos inexistente en el sistema actualmente, pasando de una acción reactiva al escalamiento de incidentes a una pro-activa, donde se puede notificar y accionar sobre un incidente prácticamente en simultáneo a su detección en los ambientes productivos.

Evitar los procesos manuales, además del ahorro de tiempo, implica un ahorro de energía y reduce el desgaste humano subjetivo pero inminente, permitiendo invertir el tiempo usado para la búsqueda de información de los eventos en el análisis y corrección de fallas.

### Análisis de valor para el área de soporte tecnológico

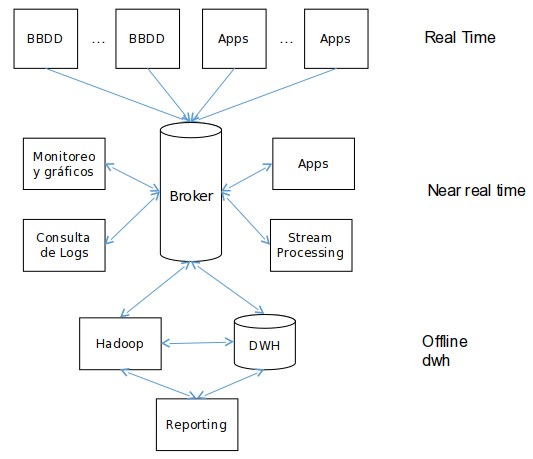
Para el área de soporte tecnológico la solución presentada aporta ventajas competitivas globales como solución integral e individuales desde la perspectiva y función de cada componente:

Como solución integral:

* Se reduce significativamente el tiempo y esfuerzo para obtener información relevante para el diagnóstico y análisis de fallas en el sistema, evitando procesos artesanales desgastantes para el soportista.
* Disponibiliza información semi-estructurada para análisis, facilitando el cruce información entre eventos mediante automatismos y herramientas especializadas.
* Disponibiliza de forma automática y simplificada los eventos más relevantes dirigiéndolos a un área especializada particular sin necesidad de intervención humana.
* Permite mantener información disponible por mayor tiempo, facilitando el análisis de tendencias e incidencias en períodos de tiempo superior.
* Incrementa la capacidad de entendimiento del sistema simplificando información compleja y ofuscada en registros ordenados y semiestructurados, permitiendo que no se requiera alta especialización para entender y analizar con su contenido.
* Incorpora mejoras de performance al sistema actual de aplicaciones que conforman el backend del negocio mediante la actualización de la capa de logueo (log4j2)

En el siguiente diagrama se grafica las correspondencias de estrategias de big data con respecto a las capas de la solución propuesta:

***Figura 61****: Diagrama integral del modelo propuesto*



*Fuente: elaboración propia*

# Conclusiones

## Objetivos y mejoras conseguidas

Con la solución presentada, logramos obtener mayor información de las aplicaciones, filtrando y exponiendo los elementos claves del mensaje, estructurándolo de forma dinámica y disponibilizándolo para ser explotado de forma ágil. Estos agregados son la esencia clave para agilizar el análisis las áreas de soporte técnico y análisis de performance de las aplicaciones que componen el sistema.

Soportar el almacenamiento y explotación de información transaccional histórica de 5 años es ahora técnica y financieramente posible para la empresa, pasando de un estimado de

1400 Terabytes para soportar 5 años, a una fracción que representa un 40 %, es decir, unos 560 terabytes, logrando una reducción de un 60% de almacenamiento.

Mediante la solución presentada se expusieron los beneficios que ofrecen las herramientas y tecnologías big data para el manejo de altos volúmenes de información, variedad de formatos y velocidad de procesamiento. La solución habilitó la solución a problemáticas similares en otros dominios dentro de la compañía, sirviendo como modelo base para otros proyectos big data de protocolo B2B[[28]](#footnote-28) para notificaciones en near real time entre sistemas y áreas de supervisión industrial y comercial de la organización.

## Líneas futuras de investigación

Esta sección tiene como objetivo formular algunas pautas e identificar nuevas perspectivas para continuar con este trabajo en tecnologías orientadas a big data, como el título de la tesis sugiere, para contribuir a la construcción de una arquitectura robusta y escalable para la explotación eficiente de logs en sistemas distribuidos complejos.

A continuación se enumeran algunos de estos aspectos:

* Analizar patrones ocultos para detectar posibles relaciones y tendencias entre eventos.
* Usar filesystem orientados al procesamiento paralelo como hadoop, para complementar el modelo actual presentado e incrementar la velocidad de procesos de lectura y escritura de información.
* Analizar la performance y niveles de almacenamiento necesarios empleando algoritmos y mecanismos de compresión para la transferencia de los mensajes entre los distintos componentes del sistema.
* Realizar análisis de factibilidad de costo y performance para el uso de tecnologías sobre plataformas Cloud, híbridas y onPremise para el escalamiento de la arquitectura presentada.
* Establecer estrategias de hidratación de los mensajes a partir de información complementaria de sistemas y procesos alternos.
* Emplear estrategias de IoT[[29]](#footnote-29) (*Internet of Things*) e IoE[[30]](#footnote-30) (*Internet of Everything*) a partir de mensajes provenientes de entidades de negocio para análisis y seguimiento de procesos core de la empresa.

## Comentarios finales

Actualmente existen soluciones tecnológicas prácticamente para cualquier necesidad empresarial, por compleja que sea. A la hora de elegir la más adecuada, conviene distinguir entre las tecnologías "de moda" y las que ofrecen valor a largo plazo.

El modelo ha sido presentado con un diseño simple y estándar, siguiendo los patrones documentados del estado del arte de la arquitectura orientada a eventos y dentro de las metodologías de big data. La arquitectura demostró su eficacia y practicidad con la organización, estructura y notificación básica de alertas tempranas de eventos críticos, logrando no solo disponibilizar la información relevante de forma simple e indexable sino también permitiendo escalar el almacenamiento histórico para análisis y diagnósticos que así lo requieran.

Para el éxito de este tipo de proyectos, es importante administrar las espectativas sobre los resultados esperados, dado que son proyectos que a diferencia de un proyecto común de desarrollo, implica actividades como pruebas ensayos y error constante, una mejora continua y constante parametrización de los componentes del sistema. A la vez, es vital equivocarse rápido, aplicando circuitos cortos e iteraciones ágiles con los prototipos de cada componente del sistema, de manera de poder disponer de más tiempo para los aspectos que más insumen dedicación, como la observación, el análisis y la integración de los sistemas.

# Referencias Bibliográficas

Amazon Glacier - Cloud Archive. (2017). Recuperado 13 de mayo de 2017, a partir de https://aws.amazon.com/glacier/?

sc\_channel=PS&sc\_campaign=acquisition\_AR&sc\_publisher=google&sc\_medium=eng lish\_glacier\_b&sc\_content=glacier\_p&sc\_detail=aws glacier&sc\_category=glacier&sc\_segment=176370234601&sc\_matchtype=p&sc\_countr y=AR&s\_kwcid=AL!4422!

Amazon Simple Storage Service (S3) — Cloud Storage — AWS. (2017). Recuperado 13 de mayo de 2017, a partir de https://aws.amazon.com/s3/? utm\_source=AWSBlog&utm\_medium=socialmedia&SM\_AWS\_Blog\_S3Glacier

Apache Fundation. (2015). Home - Apache QpidTM. Recuperado 28 de enero de 2018, a partir de https://qpid.apache.org/

Apache Hive TM. (2011). Recuperado 13 de mayo de 2017, a partir de https://hive.apache.org/

Apache Kafka. (2017). Recuperado 7 de mayo de 2017, a partir de https://kafka.apache.org/

Apache Pig. (2017). Recuperado 13 de mayo de 2017, a partir de https://pig.apache.org/

Apache Software Fundation. (2010). Apache ZooKeeper - Home. Recuperado 28 de enero de 2018, a partir de https://zookeeper.apache.org/

Apache Software Fundation. (2014). Maven, Ivy, and Gradle Artifacts - Apache Log4j 2. Recuperado 17 de febrero de 2018, a partir de https://logging.apache.org/log4j/log4j-2.1/ maven-artifacts.html

Apache Software Fundation. (2017a). Apache Kafka Documentation. Recuperado 28 de enero de 2018, a partir de http://kafka.apache.org/documentation.html#diskandfs

Apache Software Fundation. (2017b). Compression - Apache Kafka - Apache Software Foundation. Recuperado 10 de febrero de 2018, a partir de https://cwiki.apache.org/confluence/display/KAFKA/Compression

Astrahan, M. M., Blasgen, ht W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., … Watson, V. (1976). System R: Relational Approach to Database Management.

Recuperado a partir de http://daslab.seas.harvard.edu/reading-group/papers/astrahan1976.pdf

Bellare, M., & Yee, B. S. (1997). Forward Integrity For Secure Audit Logs. Recuperado a partir de http://www-cse.ucsd.edu/users/mihir.

Beller, M. J., & Alan Barnett. (2009). Business analytics Types of analytics Basic domains within analytics. Recuperado a partir de

http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W101126B.pdf

Blázquez, A., & Martínez-Abarca, J. A. (2005). Infraestructuras para el desarrollo e implantación de aplicaciones de red escalables. *Desarrollo de Grandes Aplicaciones Distribuidas Sobre Internet*, 11–44. Recuperado a partir de http://search.ebscohost.com/login.aspx? direct=true&db=iih&AN=43759645&lang=es&site=ehost-live

Bolo, M. (2007). Arquitectura de integración orientada a servicios. *Universidad de Lima - Interfases*, *0*(1), 19–46.

Business Intelligence (BI) Software | Sisense. (2017). Recuperado 13 de mayo de 2017, a partir de https://www.sisense.com/

Centralized Logging Using Rsyslog. (2010). Recuperado 7 de mayo de 2017, a partir de https://www.urbanairship.com/blog/centralized-logging-using-rsyslog

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E. (2006). Bigtable: A Distributed Storage System for Structured Data. Recuperado a partir de https://static.googleusercontent.com/media/research.google.com/es//archive/ bigtable-osdi06.pdf

Chun, J. K., & Cho, S. H. (2006). Performance and stability testing of MSMQ in the .NET environment. En *Proceedings - Third IEEE International Workshop on Electronic Design, Test and Applications, DELTA 2006*. https://doi.org/10.1109/DELTA.2006.72

Cisco. (2015a). Internet of Everything (IoE) - Cisco. Recuperado 16 de marzo de 2018, a partir de http://ioeassessment.cisco.com/en-gb/learn

Cisco. (2015b). Internet of Things (IoT) - Cisco. Recuperado 16 de marzo de 2018, a partir de https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html

Clark C. Evans. (2011). The Official YAML Web Site. Recuperado 13 de enero de 2018, a partir de http://yaml.org/

Co, E. (2017). Logstash. Recuperado 7 de mayo de 2017, a partir de https://www.elastic.co/products/logstash

Computerworld. (1996a). Delta relies on IBM’s MQSeries to tie systems. *vol. 30, no. 49, p.*

*53*. Recuperado a partir de https://books.google.com.ar/books?

id=l6NBBE8\_9IUC&pg=PA53&lpg=PA53&dq=“Delta+relies+on+IBM’s+MQSeries+t o+tie+systems.&source=bl&ots=BgxZKejty9&sig=Wfs9BUoWP0kwf6RHIWyPEQCd

Bbc&hl=es-419&sa=X&ved=0ahUKEwi\_9OnjwPvYAhVFxpAKHRTkCYQ6AEIKTAA#v=o

Computerworld. (1996b). Middleware safeguards data transactions over the Web. *vol.30*,

p.28. Recuperado a partir de https://books.google.com.ar/books?

id=323lGEzHOJcC&pg=PA28&lpg=PA28&dq=Middleware+safeguards+data+transacti ons+over+the+Web.+(New+Era+of+Networks+Neonweb+messageoriented+middleware)&source=bl&ots=4TcDeqCzSR&sig=y7auZ3UJ38j3xZ10Aa66xG GljW8&hl=es-419&sa=X&v

David Liedle. (2015). Why JSON is the Best Application Log Format. Recuperado 10 de diciembre de 2017, a partir de https://www.loggly.com/blog/why-json-is-the-bestapplication-log-format-and-how-to-switch/

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. Recuperado a partir de

https://static.googleusercontent.com/media/research.google.com/es//archive/mapreduceosdi04.pdf

Department of Defense USA. (1985). Trusted Computer System Evaluation Criteria.

Recuperado a partir de http://csrc.nist.gov/publications/history/dod85.pdf

Elasticsearch. (2017a). Elasticsearch. Recuperado 13 de mayo de 2017, a partir de https://www.elastic.co/products/elasticsearch

Elasticsearch. (2017b). Kibana. Recuperado 13 de mayo de 2017, a partir de https://www.elastic.co/products/kibana

Erlang.org. (2016). Erlang Programming Language. Recuperado 12 de febrero de 2018, a partir de https://www.erlang.org/

Fady Sedrak. (2007). Erlang: Open Telecom Platform. Recuperado 13 de enero de 2018, a partir de http://fadylog.blogspot.com.ar/2007/09/erlang-open-telecom-platform-part-

3.html

Foundation, T. A. S. (2012).

https://github.com/LMAX-Exchange/disruptor/wiki/Performance-Results.

Graylog Open Source Log Management. (2017). Recuperado 7 de mayo de 2017, a partir de https://www.graylog.org/

Greer, K. M. L., Matthes, P. C., & Cannistraro, V. F. (2010). Biometric social networking. Google Patents.

Grosso, W., & Reilly, P. O. (2001). *Java RMI*. *October*.

Hartung, G. (2016). Secure Audit Logs with Verifiable Excerpts – Full Version. Recuperado a partir de https://eprint.iacr.org/2016/283.pdf

Heka open source stream processing. (2014). Recuperado 13 de mayo de 2017, a partir de https://hekad.readthedocs.io/en/v0.10.0/

Honeybadger Exception and Uptime Monitoring. (2017). Recuperado 13 de mayo de 2017, a partir de https://www.honeybadger.io/

IBM. (s. f.). IBM MQ. Recuperado 30 de abril de 2017, a partir de http://www-03.ibm.com/software/products/en/ibm-mq

IBM. (2017). IBM - What is the Hadoop Distributed File System (HDFS) - United States. Recuperado 7 de mayo de 2017, a partir de

http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/

Katal, A., Wazid, M., & Goudar, R. H. (2013). Big data: Issues, challenges, tools and Good practices. En *2013 Sixth International Conference on Contemporary Computing (IC3)*

(pp. 404–409). IEEE. https://doi.org/10.1109/IC3.2013.6612229

Kreps, J., Corp, L., Narkhede, N., & Rao, J. (2011). Kafka: a Distributed Messaging System for Log Processing. Recuperado a partir de http://notes.stephenholiday.com/Kafka.pdf

Kreps Jay LinkedIn. (2013). The Log: What every software engineer should know about realtime data’s unifying abstraction | LinkedIn Engineering. Recuperado 1 de mayo de 2017, a partir de https://engineering.linkedin.com/distributed-systems/log-what-everysoftware-engineer-should-know-about-real-time-datas-unifying

Lith, A., & Mattsson, J. (2010). Investigating storage solutions for large data. Recuperado a partir de http://publications.lib.chalmers.se/records/fulltext/123839.pdf

Liu, C. L., & Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a HardReal-Time Environment. *Journal of the ACM*, *20*(1), 46–61. https://doi.org/10.1145/321738.321743

LMAX-Exchange. (2011). Disruptor by LMAX-Exchange. Recuperado 12 de enero de 2018, a partir de http://lmax-exchange.github.io/disruptor/

Log Management & Analysis Software Made Easy | Logentries. (2017). Recuperado 7 de mayo de 2017, a partir de https://logentries.com/

Log Management | Loggly. (2017). Recuperado 7 de mayo de 2017, a partir de https://www.loggly.com/

Mark Richards. (2015). *Microservices vs. Service-Oriented Architecture -*. Recuperado a partir de https://play.google.com/books/reader? printsec=frontcover&output=reader&id=rXcMCAAAAEAJ&pg=GBS.PA15.w.0.0.0.1

Martin, R. C. (2009). *Clean code : a handbook of agile software craftsmanship*. Prentice Hall.

Meneses, E., Bronevetsky, G., & Kalé, L. V. (2011). Evaluation of Simple Causal Message Logging for Large-Scale Fault Tolerant HPC Systems. Recuperado a partir de http://charm.cs.illinois.edu/newPapers/11-04/paper.pdf

MidcomData. (2014). End of Service Life. Recuperado 27 de marzo de 2018, a partir de http://www.midcomdata.com/printer-end-of-service-life/

MongoDB Inc. (2017). Install Compass — MongoDB Compass stable. Recuperado 20 de febrero de 2018, a partir de

https://docs.mongodb.com/compass/master/install/#download-compass

MongoDB Inc. (2017). Install MongoDB Community Edition on Ubuntu — MongoDB Manual 3.6. Recuperado 18 de febrero de 2018, a partir de

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/#overview

MongoDB Inc. (2017). WiredTiger Storage Engine — MongoDB Manual 3.6. Recuperado 25 de febrero de 2018, a partir de

https://docs.mongodb.com/manual/core/wiredtiger/#compression

Monson-Haefel, R., & Chappell, D. a. (2001). *Java Message Service Richard*. *Sun Microsystems Inc.*

Mozilla. (2017). Mozilla Public License. Recuperado a partir de https://www.mozilla.org/enUS/MPL/2.0/

Nannoni, N. (2015). Message-oriented Middleware for Scalable Data Analytics Architectures. Recuperado a partir de

http://kth.diva-portal.org/smash/get/diva2:813137/FULLTEXT01.pdf

Network, M. D. (2014). Introducción a la Disponibilidad. Recuperado 10 de abril de 2016, a partir de https://msdn.microsoft.com/es-es/library/aa291543(v=vs.71).aspx

NOSQL Databases. (2011). Recuperado 1 de mayo de 2017, a partir de http://nosqldatabase.org/

Obermeyer, P., & Hawkins, J. (2001). Microsoft .NET Remoting: A technical overview. *Microsoft Corporation*.

Oposiciones TIC. (2012). Arquitectura SOA orientada a servicios ~ Oposiciones TIC. Recuperado 4 de marzo de 2017, a partir de

https://oposicionestic.blogspot.com.ar/2012/08/arquitectura-soa-orientada-servicios.html

Oracle. (2010). Message-Oriented Middleware (MOM) (Sun Java System Message Queue

4.3 Technical Overview). Recuperado 21 de enero de 2018, a partir de https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html Oracle. (2017a). MySQL. Recuperado 1 de mayo de 2017, a partir de https://www.mysql.com/

Oracle. (2017b). Oracle | Database. Recuperado 1 de mayo de 2017, a partir de https://www.oracle.com/index.html

Oracle GoldenGate. (2017). Recuperado 1 de mayo de 2017, a partir de http://www.oracle.com/us/products/middleware/data-integration/goldengate/overview/ index.html

Papazoglou, M. P. (2003). Service -oriented computing: Concepts, characteristics and directions. En *Proceedings - 4th International Conference on Web Information Systems Engineering, WISE 2003* (pp. 3–12). IEEE Comput. Soc. https://doi.org/10.1109/WISE.2003.1254461

Papertrail - cloud-hosted log management, live in seconds. (2017). Recuperado 7 de mayo de 2017, a partir de https://papertrailapp.com/

Pivotal Software Inc. (2017). RabbitMQ - open source message broker. Recuperado 13 de enero de 2018, a partir de https://www.rabbitmq.com/

Pivotal Software Inc. (2017). RabbitMQ - AMQP 0-9-1 Model Explained. Recuperado 7 de febrero de 2018, a partir de https://www.rabbitmq.com/tutorials/amqp-concepts.html

Pivotal Software Inc. (2017). RabbitMQ - Broker Semantics. Recuperado 7 de febrero de 2018, a partir de https://www.rabbitmq.com/semantics.html

Pivotal Software Inc. (2017). RabbitMQ - Clustering Guide. Recuperado 12 de febrero de

2018, a partir de https://www.rabbitmq.com/clustering.html

Pivotal Software Inc. (2017). RabbitMQ - Distributed RabbitMQ brokers. Recuperado 12 de febrero de 2018, a partir de http://www.rabbitmq.com/distributed.html

Pivotal Software Inc. (2017). RabbitMQ - Downloading and Installing RabbitMQ.

Recuperado 18 de febrero de 2018, a partir de https://www.rabbitmq.com/download.html

Pivotal Software Inc. (2017). RabbitMQ - RabbitMQ tutorial - “Hello World!”; Recuperado 12 de febrero de 2018, a partir de https://www.rabbitmq.com/tutorials/tutorial-onejava.html

Pivotal Software Inc. (2017). Understanding When to use RabbitMQ or Apache Kafka. Recuperado 13 de febrero de 2018, a partir de

https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

PostgreSQL: open source database. (2017). Recuperado 1 de mayo de 2017, a partir de https://www.postgresql.org/

RedHat. (2015). HornetQ - putting the buzz in messaging - JBoss Community. Recuperado

28 de enero de 2018, a partir de http://hornetq.jboss.org/

Ri5. (2003). ¿Qué es XMLy Para Que Sirve? Recuperado 23 de abril de 2017, a partir de http://www.ri5.com.ar/ayuda07.php

Ricardo Barranco Fragoso. (2012). ¿Qué es Big Data? Recuperado 9 de abril de 2017, a partir de https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/

Richards, R. (2006). Representational State Transfer (REST). *Pro PHP XML and Web*

*Services SE - 17*. https://doi.org/10.1007/978-1-4302-0139-7\_17

Riemann - A network monitoring system. (2015). Recuperado 13 de mayo de 2017, a partir de http://riemann.io/

Rouse, M. (2016). SOAP ( Simple Object Access Protocol ). *techtarget*.

Russom, P. (2011). Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, 6. Recuperado a partir de

ftp://129.35.224.12/software/tw/Defining\_Big\_Data\_through\_3V\_v.pdf

Sacha Krakowiak. (2009). What is Middleware. Recuperado a partir de http://ligmembres.imag.fr/krakowia/Files/MW-Book/Chapters/Preface/preface.html

SAS Inc. (2017). SAS Event Stream Processing | SAS. Recuperado 20 de febrero de 2018, a partir de https://www.sas.com/en\_us/software/event-stream-processing.html

Scala. (2017). The Scala Programming Language. Recuperado 28 de enero de 2018, a partir de http://scala-lang.org/

Sentry real-time tracking. (2017). Recuperado 13 de mayo de 2017, a partir de https://sentry.io/welcome/

Sicular, S. (2013). Gartner’s Big Data Definition Consists of Three Parts, Not to Be Confused with Three “V”s.

Stomp. (2012). STOMP. Recuperado 28 de enero de 2018, a partir de http://stomp.github.io/

Stonebraker, M. (1976). The Desing and Implementation of INGRES. Recuperado a partir de http://files.eric.ed.gov/fulltext/ED118130.pdf

Stonebraker, M., & Cattell, R. (2011). 10 rules for scalable performance in “simple operation” datastores. *Communications of the ACM*, *54*(6), 72. https://doi.org/10.1145/1953122.1953144

Telegram.org. (2017). Bots: An introduction for developers. Recuperado 25 de febrero de

2018, a partir de https://core.telegram.org/bots

The Apache Software Foundation. (2017). Log4j – Log4j 2 Lock-free Asynchronous Loggers for Low-Latency Logging - Apache Log4j 2. Recuperado 12 de enero de 2018, a partir de http://logging.apache.org/log4j/2.x/manual/async.html

Tyler Treat. (2016). Benchmarking Message Queue Latency – Brave New Geek. Recuperado 13 de febrero de 2018, a partir de https://bravenewgeek.com/benchmarking-messagequeue-latency/

Videla, A., & Williams, J. J. W. (2012). *RabbitMQ in action : distributed messaging for everyone*. Manning.

Vinoski, S. (2006). Advanced message queuing protocol. *IEEE Internet Computing*.

https://doi.org/10.1109/MIC.2006.116

VMware. (2011). vCenter Orchestrator AMQP Plug-in is now Generally Available - VMware vCenter Orchestrator Blog - VMware Blogs. Recuperado 28 de enero de 2018, a partir de https://blogs.vmware.com/orchestrator/2011/08/vcenter-orchestrator-amqp-plug-in-isnow-generally-available.html

VMware. (2017). VMware – Official Site. Recuperado 13 de enero de 2018, a partir de https://www.vmware.com/

Wikipedia. (2017). ¿Qué es el marketing industrial? Definición, características y estrategia. Recuperado 26 de febrero de 2018, a partir de https://www.titular.com/blog/que-es-elmarketing-industrial

Xie, C., Su, C., Kapritsos, M., Wang, Y., Yaghmazadeh, N., Alvisi, L., & Mahajan, P. (2014).

Combining ACID and BASE in a Distributed Database Salt: Combining ACID and BASE in a Distributed Database. Recuperado a partir de

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/xie

Zhang, Y., & Geng, H. (2009). A new solution of distributed simulation calculation based on .NET Remoting and MSMQ. En *2009 International Conference on Future Computer and Communication, FCC 2009*. https://doi.org/10.1109/FCC.2009.72

Zikopoulos, P., & Eaton, C. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.

1. Bus de Servicio Empresarial (ESB por sus siglas en inglés) es un modelo de arquitectura de software que gestiona la comunicación entre servicios web. Es un componente fundamental de la Arquitectura Orientada a Servicios. [↑](#footnote-ref-1)
2. La palabra backbone se refiere a las principales conexiones troncales de una red. [↑](#footnote-ref-2)
3. Business Analytics (BA) se refiere a las habilidades, tecnologías, prácticas para la exploración iterativa continua y la investigación del desempeño del negocio pasado para obtener información y conducir la planificación de negocios(Beller & Alan Barnett, 2009) [↑](#footnote-ref-3)
4. SQL: SQL (por sus siglas en inglés *Structured Query Language;* en español lenguaje de consulta estructurada) es un lenguaje específico del dominio que da acceso a un sistema de gestión de bases de datos relacionales que permite especificar diversos tipos de operaciones en ellos. [↑](#footnote-ref-4)
5. XML Por su sigla en ingles (eXtensible Markup Language), traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible (Ri5, 2003). [↑](#footnote-ref-5)
6. Log: El uso de este anglicismo es debido a las traducciones del inglés de la jerga informática. En inglés el término log designa la bitácora en la cual el comandante de un navío consigna los acontecimientos que sobrevienen durante la travesía [↑](#footnote-ref-6)
7. Debug: técnicas comunes de depuración para detectar anomalías, corregir funcionalidades y optimizar código fuente. [↑](#footnote-ref-7)
8. ACID: En bases de datos se denomina ACID a las características de los parámetros que permiten clasificar las transacciones de los sistemas de gestión de bases de datos (Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.). [↑](#footnote-ref-8)
9. Commit: En el contexto de la Ciencia de la computación y la gestión de datos, commit (acción de comprometer) se refiere a la idea de consignar un conjunto de cambios "tentativos" de forma permanente. [↑](#footnote-ref-9)
10. Rollback: En tecnologías de base de datos, un rollback o reversión es una operación que devuelve a la base de datos a algún estado previo. [↑](#footnote-ref-10)
11. DBMS: Database Managment System: Sistema Gestor de Base de datos, es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. [↑](#footnote-ref-11)
12. API: La interfaz de programación de aplicaciones, abreviada como API del inglés: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. [↑](#footnote-ref-12)
13. El término cluster (del inglés cluster, "grupo" o "racimo") se aplica a los conjuntos o conglomerados de ordenadores unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen una única computadora. [↑](#footnote-ref-13)
14. Tupla: un registro (también llamado fila) representa un objeto único de datos implícitamente estructurados en una tabla. [↑](#footnote-ref-14)
15. HDFS, por sus siglas en inglés Hadoop File System. [↑](#footnote-ref-15)
16. Framework o marco de trabajo, es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. [↑](#footnote-ref-16)
17. En sistemas distribuidos, un Middleware es una capa de software ubicada entre el sistema operativo y las aplicaciones de cada parte del sistema. Facilita el desarrollo de software abstrayendo la heterogenedidad de funcionalidades y encapsulando sus recursos adyacentes.(Sacha Krakowiak, 2009) [↑](#footnote-ref-17)
18. En informática, el *scheduling*(por su término en inglés) es el método por el cual el trabajo especificado por algún medio es asignado a los recursos que completan el trabajo (Liu & Layland, 1973) [↑](#footnote-ref-18)
19. OLTP: Procesamiento de Transacciones En Línea, es un tipo de procesamiento que facilita y administra aplicaciones transaccionales, usualmente para entrada de datos y recuperación y procesamiento de transacciones [↑](#footnote-ref-19)
20. Terabyte (TB), equivalente a 1012 (1.000.000.000.000 — un billón —) de bytes o 1024 *gigabytes* [↑](#footnote-ref-20)
21. Un petabyte es una unidad de almacenamiento de información que equivale a 1015 bytes = [↑](#footnote-ref-21)
22. .000.000.000.000.000 de bytes, 1.000 terabytes o 1.000.000 gigabytes. [↑](#footnote-ref-22)
23. *rsync* es una utilidad de código abierto que provee un mecanismo de sincronización rápido e incremental de archivos. Disponible en las ditribuciones linux y Unix como parte de la GNU General Public License y es actualmente mantenido por Wayne Davison. [↑](#footnote-ref-23)
24. *Cron* es un servicio que ejecuta tareas de manera programada basado en la configuración de la orden crontab. Cumple esta tarea despertando cada minuto y revisando la existencia de trabajos de cron en las crontabs de los usuarios. [↑](#footnote-ref-24)
25. Near Real Time (Casi en tiempo real), se refiere al lapso de tiempo introducido entre la ocurrencia del evento y el la disponibilidad de la data producida por el evento. [↑](#footnote-ref-25)
26. La cinta magnética es un tipo de medio o soporte de almacenamiento de datos que se graba en pistas sobre una banda plástica con un material magnetizado, generalmente óxido de hierro o algún cromato. [↑](#footnote-ref-26)
27. Evaluación basada en la experiencia del autor, ponderación del 1 al 10, donde 1 es muy mala y 10 es excelente [↑](#footnote-ref-27)
28. B2B (Del inglés *business-to-business*) hace referencia a las transacciones comerciales entre empresas o entre componentes de sistemas que representan distintos negocios.(Wikipedia, 2017) [↑](#footnote-ref-28)
29. IoT (*Internet of Things por sus siglas en ingles)* incluye cualquier tipo de objeto físico o virtual o entidad que pueda hacerse direccionable y que tenga la capacidad de transmitir datos sin la entrada de persona a máquina (Cisco, 2015b). [↑](#footnote-ref-29)
30. IoE (*Internet of Everything por sus siglas en ingles*)implica la interconexión de personas, cosas, procesos y datos de manera de lograr un aprovechamiento más relevante de la información (Cisco, 2015a)*.*  [↑](#footnote-ref-30)