UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Creación de un tablero omnicanal

AUTOR: Gustavo Reinaldi

DIRECTOR ACADÉMICO: Leonardo Corbalan

DIRECTOR PROFESIONAL: Gabriel Negri

CARRERA: Licenciatura en Sistemas

Resumen

La tesina describe el desarrollo de un software de envergadura llevado a cabo en la compañía Despegar.com: La creación de un “Tablero Omnicanal”. Este tablero permite a la compañía obtener información relevante respecto del rendimiento de todas las comunicaciones y canales abiertos con sus clientes.

Palabras Clave

Big Data, Omnichannel, Tracking de datos, cluster, key performance indicator.

Conclusiones

La creación de este tablero omnicanal, impactó directamente sobre las comunicaciones de la compañía que mejoraron tanto en calidad como en frecuencia y rendimiento (conversión de usuarios). Los datos se vislumbraron cómo un activo muy importante dentro de la organización. Los equipo participantes incorporaron el concepto de calidad de los datos como eje transversal en cada acción de nuevo desarrollo entendiendo que sin ellos la organización pierde activos.

Trabajos Realizados

Modelado y registración de datos relevantes al desarrollo del tablero. Construcción de un tablero omnicanal.

Trabajos Futuros

Agregar servicio de priorización de canal de envío con el objetivo de maximizar la apertura de comunicaciones.

Revisión del proceso de alertas para ajustar los envíos. Sumar lógica a la coordinación para aumentar la entregabilidad de comunicaciones.

Fecha de la presentación: Noviembre 2020

Creación de un tablero omnicanal

Desarrollo del sistema de tracking para nutrir el tablero omnicanal de la compañía

Índice General

Capítulo 1

Introducción

Despegar.com es una agencia de viajes en línea que cuenta con una plataforma tecnológica en la que prestadores de servicios turísticos ofrecen y comercializan sus productos hacia los usuarios. Éstos, a su vez, utilizan la misma plataforma para hacer averiguaciones sobre vuelos, alojamientos, actividades y demás servicios turísticos. Así, cada usuario, puede armar y gestionar su propio viaje, comparando las opciones disponibles, reservando y adquiriendo las prestaciones ofrecidas por separado o combinadas, de acuerdo con sus intereses personales. La agencia actúa como intermediario entre el usuario y los proveedores de servicios turísticos.

En esta tesina se describe el proceso de creación de un tablero omnicanal llevado a cabo por la compañía OTA[1] Despegar.com. Este proceso, expuesto a través de una historia de *Roadmap*[2], constituyó un esfuerzo colaborativo de varios equipos de trabajo, en el que el autor de esta tesina se desempeñó como analista, desarrollador y, muchas veces, como nexo operativo entre los distintos equipos

Luego de presentar los conceptos introductorios y el marco teórico necesario (capítulos 1 y 2), en los capítulos 3, 4, 5 y 6 de esta tesina se describen en orden temporal las tareas realizadas por los equipos involucrados en la creación del tablero omnicanal. La descripción incluye la recepción de los datos (todo aquello que puede ser procesado o transformado mediante procesos computacionales o mentales), la manipulación de información (aquellos datos que representan propiedades específicas en el dominio de acción de las personas o agentes artificiales) y finalmente la generación de conocimiento cuya gestión favorece la toma de decisiones [1].

El modelo de trabajo utilizado por la compañía fue iterativo e incremental. Los equipos participantes en la creación del tablero omnicanal fueron ejecutando varias iteraciones en un esquema de trabajo paralelo. Esto requirió la conformación de un grupo operativo en la empresa abocado a la orquestación de los objetivos parciales.

---

[1] Online Travel Agency ó Agencias de Viajes Online OTA por su definición en inglés son sitios web dedicados principalmente a la venta de servicios dentro del sector de viajes.

[2] Es una herramienta que muestra a todas las partes interesadas cómo es que un *product manager* creará un producto; sirve para delimitar un presupuesto y estrategias de desarrollo. En resumen, es un plan de cómo es que un producto cumplirá ciertos objetivos comerciales y en qué consiste la estrategia para construirlo.

La última etapa del proceso de creación del tablero omnicanal consistió en el registro y documentación del conocimiento adquirido. La sistematización de este conocimiento resulta útil en la toma de decisiones de negocio, desde las etapas más tempranas de registración de eventos, hasta las más tardías como la manipulación del conocimiento para su transformación. Una parte de estas lecciones aprendidas se describe en las conclusiones de esta tesina (capítulo 7).

1.1 Contexto histórico

En las últimas dos décadas el avance de la tecnología ha permitido el aumento del número de canales por donde se pueden comunicar las personas de manera virtual. Este proceso, que primero se apoyó en la masificación de las conexiones a Internet, se ha acelerado notoriamente en la última década. Da cuenta de ello la irrupción de nuevas formas de comunicación que han cambiado las conductas individuales y sociales.

Las empresas desde comienzos del 2000 necesitaron agilizarse para mantenerse saludables en el negocio. En particular las compañías dedicadas al Marketing se adaptaron rápidamente al cambio tecnológico y construyeron una técnica para afrontar este desafío que llamaron multicanalidad. [2]

Fue a principios de los 2000 que las empresas entendieron la necesidad de habilitar nuevas formas de contactarse con sus clientes. Esto, por supuesto, incluía fax, SMS y otros canales hoy considerados obsoletos.

Antes de estos cambios, si una persona deseaba comprar cualquier artículo, se dirigía al local que lo comercializaba, analizaba un reducido número de alternativas ofrecidas, y se decidía por aquella que mejor la conformaba. A medida que el mundo de la tecnología fue evolucionando, el hábito de compra de esa misma persona también se fue modificando, aprendiendo a gestionar una mayor cantidad de información disponible para mejorar su decisión. Las personas (clientes) dejaron de conocer productos sólo a través de otras personas (lo que conocemos como el boca a boca), sino que incorporaron otros canales de información, como revistas especializadas con análisis de opciones de productos.

Como se mencionó la llegada de Internet trajo un nuevo mundo de posibilidades. Los consumidores ya no dependían de las empresas locales para satisfacer sus necesidades. Internet les permitió navegar, investigar y comprar en línea.

La masificación de las computadoras portátiles, *tablets* y *smartphones*, permitió a los consumidores contar con múltiples opciones para satisfacer sus necesidades de compra. A su vez se abrió la posibilidad de enviar información de un nuevo producto a través del correo electrónico o por algún otro medio digital.

Más opciones implicó más oportunidades para conectar y atraer posibles clientes, pero también significó mucha más complejidad.
Las empresas se centraron en maximizar el rendimiento de cada canal, tienda, teléfono, web y móvil. Este fue el surgimiento de las estrategias multicanal.
Sin embargo, la transformación en los hábitos de compra no se detuvo. Los consumidores se volvieron más informados y exigentes respecto de los productos que deseaban consumir y de la coherencia de la información brindada por el comerciante. Es así que las estrategias multicanal comenzaron a quedar desactualizadas ante este nuevo escenario. [3]

1.2 Situación actual

El consumidor actual está cada vez más conectado, publica experiencias, opiniones y gustos y comparte sus compras. Es un consumidor que está dispuesto a moverse entre los diferentes canales, *offline* y *online*, buscar opiniones de usuarios y de especialistas, recomendaciones, garantías y todo esto hecho a gran velocidad a cualquier hora del día y con una cantidad de oferta ilimitada al instante. Es un consumidor que espera consistencia, uniformidad, integración en servicio y experiencia de los canales [4] y en este escenario de uniformidad de la experiencia surge la Omnicanalidad.
La aparición del comercio omnicanal está cambiando la ecuación del *retail*[3] [5] tanto en la forma en que los clientes recorren el proceso de compra, como en la que los *retailers* anticipan y satisfacen la demanda del cliente.
Un modelo omnicanal es un modelo basado en una estrategia transversal como eje fundamental para todas las áreas de la compañía, que permite mejorar la experiencia del cliente y sus indicadores de gestión.
Para que se haga realidad este modelo necesita de métricas que visibilicen las preferencias del usuario, su comportamiento en las distintas etapas de una compra y entender cómo afectan las comunicaciones de la empresa a estos usuarios.
La toma de mediciones cuyo objetivo es nutrir a diferentes tableros en particular al tablero omnicanal sirve también como eje central en el desarrollo de esta tesina en la que podemos distinguir tres ejes principales de abordaje

- Analizar y describir el proceso de registración de la interacción entre los usuarios y la compañía.

[3] *Retail* se refiere a las actividades relacionadas con la venta de bienes o servicios a los consumidores finales para uso personal, no comercial. El concepto *retail* es esencialmente un enfoque orientado al cliente y de toda la empresa para desarrollar e implementar una estrategia de *marketing*. Proporciona pautas que deben ser seguidas por todos los minoristas, independientemente de su tamaño, diseño de canal y medio de venta.

- Describir el proceso de extracción, transformación y carga de datos (ETL[4]) que conduce a los datos hacia entidades de información útiles al negocio.
- Describir el proceso de trabajo del equipo encargado de la Inteligencia de Negocios (*Business Intelligence* -BI-) que permite la generación del tablero.

Capítulo 2

Marco Teórico

Con el propósito de brindar al lector los instrumentos necesarios para una fácil lectura y mejor comprensión, se desarrollan brevemente los principales conceptos teóricos relacionados al negocio del turismo y del *marketing* digital que se abordarán a lo largo de la presente tesina.

El trabajo habitual de un equipo de marketing digital se centra en el uso de indicadores que ponderan el nivel de desempeño de un proceso y, desde una visión más abstracta, el estado de salud del negocio. Sin buenos indicadores es imposible saber con claridad si una campaña de marketing online es eficaz o no. Básicamente porque no se sabe qué medir. Ni qué valores son los que importan y a lo que hay que llegar. Saber extraer la información, conocer los valores estándar e importantes de cada medida, conocer cómo debe comportarse y el resultado que esperas sacar de cada una de ellas será el primer paso para comenzar a extraer información verdaderamente relevante sobre métricas y *marketing* digital.

Una vez que se tiene claramente definidos los indicadores, es importante definir un proceso que busque continuamente optimizar el rendimiento de cada uno de estos indicadores. *Conversion Rate Optimization* agrupa estrategias y herramientas para tal fin. El *A/B Testing* y el *Conversion Funnel* son conceptos importantes en este proceso que también se detallan a continuación.

2.1 Key Performance Indicator (KPI)

Cómo se mencionó un indicador es una herramienta con la que el equipo de *Marketing* mide la efectividad de sus acciones dentro de la compañía [6]. El KPI o indicador refleja un valor

[4] ETL es el proceso de extracción, transformación y carga de los datos, que es parte del ciclo de vida de una implementación de *Business Intelligence*.

medible que demuestra el grado de eficiencia con que una empresa alcanza los objetivos propuestos. Para que funcione adecuadamente un KPI debe ser *smart*, es decir, tener las siguientes características:

1. ***Specific*** (específico): Identifica qué, dónde, cómo y cuándo se mide.
2. ***Measurable*** (medible): Permite cuantificar las medidas y los beneficios que se esperan.
3. ***Attainable*** (alcanzable): Posibles. No tiene sentido elegir indicadores que sean imposibles de conseguir con los recursos asignados.
4. ***Relevant*** (pertinente): Dependiendo de los objetivos que se busquen será mejor una métrica u otra.
5. ***Time-bound*** (limitado en el tiempo): Un indicador de rendimiento clave debe estar supeditado a un periodo temporal concreto.

Existen decenas de indicadores con los que medir campañas de *marketing* digital. Algunos dependen del canal concreto por el que se establece la relación con los usuarios. Lo importante es que para que los KPIs de *marketing* digital sean eficaces deben ir acompañados de un valor. Sólo sabiendo lo que se quiere conseguir se podrán analizar los caminos adecuados para lograr esos objetivos.

Los KPIs habituales con los que el *marketing* comunicacional trabaja son:

2.1.1 *Conversion Rate*

Desarrollar un nuevo tablero tiene como objetivo más general el proveer de una herramienta de visualización de mediciones que ayuden a la toma de decisiones. El *conversion rate*, o tasa de conversión, es una métrica muy utilizada para medir resultados, principalmente en el *marketing* digital.

En un término amplio conversión es toda acción llevada a cabo por un usuario en un sitio web que genere valor de negocio. Dependiendo de la actividad y naturaleza del negocio una conversión aplica a un amplio rango de posibilidades que van desde descargarse abrir un email, hasta comprar un producto, registrarse, seguir una página, llamar a un número de teléfono o solicitar información sobre un servicio, entre otros. La primera tarea que debe llevar adelante una compañía es fijar objetivos ya que sin ellos no se puede medir el resultado de las acciones registradas. Una vez trazado un conjunto de objetivos cada uno de ellos individualmente van a generar resultados a través de la toma de mediciones y estos resultados van a generar una tasa de conversión para cada uno.

Conversion Rate = (Nº de conversiones / Nº de usuarios) x 100

Las empresas dedicadas al *e-commerce* (comercio electrónico) miran la tasa de conversión como una métrica a ser utilizada para entender la eficiencia real de una comunicación específica, también la usan como información útil para la definición de metas de un equipo comercial. El objetivo de esta métrica puede ser entender cómo está el retorno de cada inversión de la empresa en función del tiempo y el dinero invertido, con relación a las ventas generadas por esa inversión. Otro objetivo puede ser una forma de medir los resultados de una estrategia utilizada por un equipo de *Marketing* Digital como en este caso lo es la omnicanalidad.

2.1.2 *Bounce Rate*

El *bounce rate,* o tasa de rebote, es un parámetro que indica el porcentaje de los emails que no han sido recibidos por sus destinatarios. Cuando un correo es rechazado por el servidor de email del destinatario se considera *bounce*. El rechazo puede ser porque no existe más la casilla, porque el dominio es inexistente (*hard bounce*), por caída de la conexión, o porque el usuario tiene su casilla de correo o *inbox* lleno y el servicio que le gestiona el envío se ve imposibilitado de entregar el mensaje (*soft bounce*).

2.1.2 *Open Rate*

El *open rate,* o tasa de apertura, de una comunicación es una métrica que indica la cantidad de mensajes que han sido abiertos o leídos por los usuarios respecto de la cantidad de mensajes que ese mismo usuario ha recibido en su casilla de correo o *inbox*. El *open rate* se obtiene dividiendo el total de mensajes abiertos por el total de enviados.

Open rate = Nº comunicaciones abiertas / Nº comunicaciones enviadas x 100

Normalmente este indicador (KPI) se encuentra, habitualmente, en valores de entre el 15% y el 35%. El rango donde realmente se ubica, dentro de este segmento de valores, depende de ciertos factores que podemos enunciar:

- **hora del envío** muchos estudios sugieren que la tasa de apertura de las comunicaciones tiene horarios donde el indicador tiene picos en sus valores. La

figura 1, muestra un ensayo de la compañía Doppler[5], donde se evaluó sobre la base de miles de comunicaciones cómo funciona el indicador *open rate* a lo largo de las 24 hs. de un día.

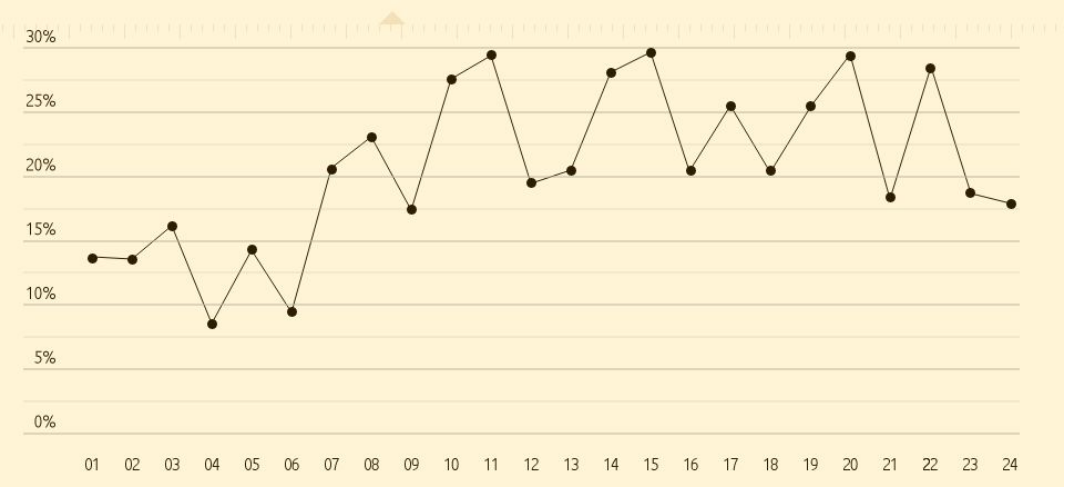

Fig. 1 - Representación del *open rate* por horario. Extraído del sitio Doppler. [7]

Normalmente hay tres grandes momentos para maximizar este indicador (KPI) enviando la comunicación a media mañana, a media tarde o bien entrada la noche (entre las 20 y 21 hs.). El mismo documento sugiere que en días laborables la métrica obtenida es más alta que en los mismos horarios pero durante el fin de semana (Fig. 2).

[5] Doppler es una compañía que desarrolla una herramienta de *email*, *automation* & *data marketing* que automatiza envíos comunicacionales.

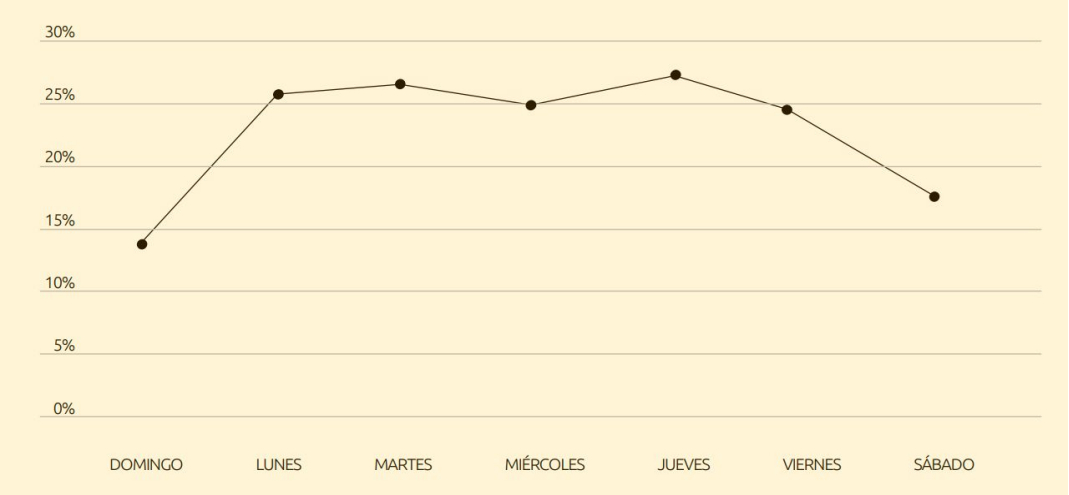

Fig. 2 - Representación del *open rate* por día de la semana. Extraído del sitio Doppler. [8]

- **País donde se realiza la campaña** Otra conclusión importante es que el indicador *open rate* tiene valores sustancialmente diferentes de acuerdo con el país donde se tome la medición. En el ensayo, Argentina, México, Colombia y Chile obtuvieron valores que duplican a los valores obtenidos en Brasil. Esta diferencia responde principalmente a la idiosincrasia de las personas en las regiones donde se obtuvieron las muestras y esta idiosincrasia determina la relación de los clientes con los medios electrónicos.

  Finalmente el indicador *open rate* varía también dependiendo de la industria donde se tome la medición como podemos observar en la gráfica de la figura 3.

Industry	Open rate
Agencies	16.10%
Arts & Entertainment	28.72%
Automotive	26.77%
Communications	28.80%
Education	28.36%
Financial Services	25.36%
Health & Beauty	25.09%
Health Care	24.83%
Internet Marketing	17.26%
Legal Services	24.65%
Non-profits	36.15%
Publishing	29.64%
Real Estate	25.48%
Restaurants & Food	33.10%
Retail	22.38%
Sports & Activities	28.99%
Technology & High Tech	24.17%
Travel	22.79%
All	22.15%

Fig. 3 - *Average results by industry*. Extraído del sitio *GetResponse*. [9]

2.1.3 *Click Through Rate* (CTR)

El *click through rate, o* tasa de clics o Ratio de cliqueo, es uno de los principales indicadores del *email marketing*. El indicador, se obtiene del número de clics que hace un usuario sobre los *links* embebidos en una comunicación respecto del número de visualizaciones de esa comunicación que él mismo efectuó. El término clic (acción de clicar sobre una URL de enlace) es propio del canal comunicacional email. El *click through rate* expone el nivel de satisfacción de un destinatario respecto de un mensaje que ha recibido y ha visualizado.

Que un usuario visualice una comunicación, depende de la capacidad que tuvo el asunto de la comunicación de despertar la atención del usuario receptor y se expone a través del indicador *open rate*. La tasa de clics, finalmente, pone de manifiesto la capacidad de una campaña para impulsar a los destinatarios a etapas más avanzadas del *conversion funnel* que se describe más adelante en esta lectura. En general, cuanto más dirigida sea una campaña, mayor será el porcentaje de clics que reciba y, por lo tanto, el valor del indicador *click through rate.*

CTR = Nº clics / Nº de visualizaciones x 100

2.1.4 *Tap Rate*

El *tap rate* es un indicador emparentado con el concepto de *push notifications.* Las notificaciones *push* son mensajes que se envían desde un servidor remoto hasta los dispositivos que tienen instalada una aplicación *mobile* capaz de procesar la recepción. El *tap rate* se obtiene dividiendo el número de *taps* (el "toque" que se le da a una *push notification*) sobre el total de envíos. Es una acción análoga a las acciones de *open* y *click* propias de la recepción y visualización de una comunicación dentro del canal de *email*.

2.2 Conversion Rate Optimization (CRO)

El *conversion rate optimization* es un término que define el conjunto de acciones enfocadas a mejorar la tasa de conversión [10]. El proceso *conversion rate optimization* consiste en iterar sobre una serie de fases:

2.2.1 Medir y analizar

Es importante conocer y analizar los resultados para tener una base de la que partir. Cuantas más métricas se analicen, se conozcan y se sepan interpretar, más información disponible se tiene acerca de las campañas comunicacionales y más fácil es detectar puntos de mejora.

Establecer metas

A partir de los datos analizados, se deben establecer las metas o KPIs que sirven como objetivo para, implementar cambios oportunos según se hayan alcanzado las metas preestablecidas o no. Es importante recalcar que los objetivos deben ser medibles y alcanzables de lo contrario su análisis será complicado y extraer conclusiones será una tarea muy difícil.

Hipótesis

Aplica al diseño de una estrategia que ayude a alcanzar los objetivos marcados, intentando valorar qué cambios pueden ayudar a ello.

Test

Realizar pruebas para comparar resultados en grupos pequeños antes de llevar a cabo grandes cambios. Para este objetivo se utiliza la técnica de *A/B Testing* que detallamos más adelante en el documento.

Implementar cambios

Una vez concluidos las pruebas, implementar los cambios con el objetivo de incrementar la tasa de conversión.

2.2.2 Establecer nuevos objetivos y volver a iterar

Establecer nuevos valores, para los indicadores tomados en cuenta en el *conversion rate optimization,* a partir de las nuevas mediciones obtenidas. El nuevo umbral que se obtiene a partir del análisis e interpretación de los valores permite volver a iniciar el proceso *de conversion rate optimization* en una nueva iteración. Aunque se hayan obtenido los resultados esperados en este ciclo, siempre se debe buscar mejorar para mantener saludable el negocio.

2.3 *A/B Test*

El *A/B Test* es una técnica que se utiliza para describir un experimento aleatorio con dos variables A y B siendo una la variable de control y la otra la dependiente. En el *marketing* comunicacional aporta una manera de comparar la eficacia de las comunicaciones enviadas a un grupo de usuarios suscriptos. Consiste en el envío de dos tipos de comunicaciones cuyo objetivo es el mismo (tienen los mismo KPIs como umbral) pero difieren en el contenido y/o *layout*[6] [11]. Generalmente los motivos principales que llevan a realizar *A/B Tests* son:

- Conseguir una mayor tasa de apertura.
- Conseguir una mejor tasa de *clicks*/ *taps*.
- Minimizar el *bounce rate*.
- Obtener mayor cantidad de ventas.
- Obtener mayor tasa de ingreso a las *landing page*.
- Optimizar el *layout* de las comunicaciones.

[6] *Layout* en el contexto del *email marketing* se refiere a la estructuración de las ofertas dentro del cuerpo del *email*. El objetivo es maximizar la conversión del indicador buscado por envío.

- Diseñar mejores *landing pages*.
- Diseñar mejores *call to action* (CTAs[7])

Estos elementos sirven para hacer campañas más efectivas y conocer mejor el comportamiento de los usuarios (una meta fundamental de la omnicanalidad).

Los *A/B Tests* conceptualmente se implementan en 3 pasos:

1. Definir un porcentaje de destinatarios que se verán afectados por el *A/B Test* dentro del conjunto de usuarios suscritos. Generalmente se afecta en una porción menor al 20% de la base total de suscriptores esto es así para evitar posibles pérdidas de ventas que afecten el rendimiento de la compañía.
2. Diseñar dos comunicaciones diferentes para un mismo objetivo. Aquí el proceso varía ligeramente dependiendo de cuál es el elemento que se pretende comparar. Por ejemplo, podría confrontarse el asunto de un *email*, en cuyo caso, trabajando con gente dedicada a la gestión de contenido, se generan dos asuntos diferentes uno para cada rama del *A/B Test.* Otro caso de utilidad podría ser determinar cuál es el diseño (*generado por un equipo de UX o User eXperience*[8]) que tiene mejor rendimiento, en cuyo caso las ramas del *A/B Test* varían en el diseño de la comunicación. También se puede considerar el *A/B Testing* sobre otros elementos constituyentes de un *email* como un CTA (*call to action*), una *landing page*, etc.
3. Generar el envío que puede ser único (una única campaña) o una campaña sostenida en el tiempo y medir resultados.

El *A/B Testing* es una de las herramientas para aumentar la tasa de conversiones [12] y consecuentemente el monto de dinero generado por *tickets*[9] (GB *Gross Booking*[10]) y el nivel de participación de las comunicaciones en ese GB de la compañía (*share*[11] GB).

2.4 *Conversion funnel*

Conversion Funnel, o embudo de conversión es una metodología de *marketing* que se ocupa de estudiar las etapas que atraviesan los usuarios en el ciclo de interacción con una compañía (Fig. 4). El *conversion funnel* permite definir los pasos que tiene que dar un

[7] Un CTA es el vínculo entre el contenido regular que el usuario consume y una página (*landing page*) con una oferta más interesante para nuestro usuario.
[8] La experiencia de usuario o *user experience*, es el conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos que dan como resultado una percepción de dicho servicio, producto o dispositivo.
[9] Un *ticket* corresponde a una venta de un producto ofrecido por la compañía.
[10] GB es el monto total pagado por los clientes por los productos y servicios de viaje reservados (incluida la parte que se transfiere al proveedor de viajes o que realiza la transacción), incluidos impuestos, tarifas y otros cargos y excluyendo IVA.
[11] *Share* refiere al porcentaje de participación que se le atribuye a una gerencia respecto del total de GB generado.

usuario para cumplir un objetivo propuesto. Esto significa, básicamente, que es una herramienta que busca que los usuarios hagan lo que un equipo comercial o de *marketing* se propuso como objetivo. [13]

En su versión más general un *conversion funnel* de trabajo para un equipo de *marketing* comunicacional debe considerar

- ***Bounces***: Los *bounces* definen la calidad de la base de datos. Nos da un valor de cuántos usuarios destinatarios de una comunicación realmente la reciben en sus casillas de *email*. En este apartado es importante priorizar la calidad sobre la cantidad asegurando que los receptores de una comunicación estén informados y deseen recibir la información que se le está enviando.
- ***Opens***: Los *opens* son una métrica que ayuda a entender cuán interesado está el receptor en recibir esa comunicación. La cantidad de comunicaciones recibidas actualmente por los usuarios puede ser abrumadora. Esto ha devenido en una feroz competencia por generar la comunicación que mejor capte la atención de un receptor. Para conseguirlo es necesario tener en cuenta las variables mencionadas para el *open rate*.
- ***Clicks*** / ***Taps***: Los clics o *taps* confirman el grado de atención del usuario receptor de una comunicación. Si el usuario llegó hasta aquí es porque la campaña publicitaria logró captar su atención. En este momento se necesita ser claro y conciso de manera que el receptor entienda que con una acción recibe un beneficio. Entran en juego elementos del *email* cómo el contenido, la forma de redacción del mensaje, el diseño y el *layout*.
- ***Return on Investment*, ROI**[12]: El retorno de inversión es una razón financiera que compara el beneficio o la utilidad obtenida en relación a la inversión realizada, es decir, representa una herramienta para analizar el rendimiento que la empresa tiene desde el punto de vista financiero [14]. En el contexto que se describe para este momento se ha invertido en que el usuario llegue a esta etapa. La idea aquí es que el usuario realice una venta para que el retorno del esfuerzo hecho (en horas, desarrollo, maquetación, envío, etc.) se vea compensado. En este punto es importante que la *landing page* destino donde se envía al usuario a través de la comunicación tenga información justa y acorde a los estímulos que recibió y que lo condujeron hasta allí.

[12] ROI *Return on Investment* hace referencia a la métrica que expresa la relación entre lo invertido en un negocio y el beneficio obtenido proveniente de dicha inversión.

- **Segmentación**: La segmentación es importante porque nos permite detectar los grupos de usuarios receptores de una comunicación que han ido quedando en las diferentes etapas del *conversion* funnel. Con esa información se pueden conocer los intereses comunes entre los usuarios de cada grupo y hacer comunicaciones customizadas.
- ***Word of mouth***: Es un término que describe la acción de discutir de manera natural y espontánea sobre un producto o una compañía. Comúnmente llamamos a esta situación como el "boca a boca". Si bien es un concepto complicado de medir, puede haber potenciales receptores de comunicaciones que esperan un estímulo para poder suscribirse. Hay herramientas como facilitar el reenvío, compartir en redes sociales, *web push notifications*, que ayudan a esta tarea.
- **Innovación**: Innovar es una acción clave en cualquier industria, el *marketing* comunicacional puede medir el interés por una innovación a través de la apertura de canales para las sugerencias, opiniones y encuestas de satisfacción.

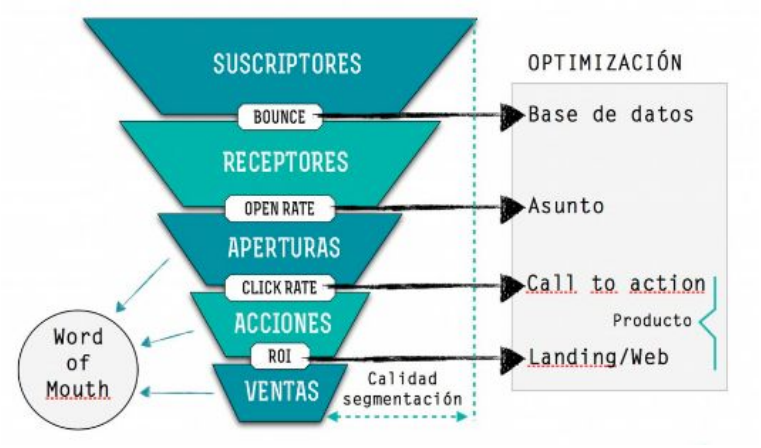

Fig. 4 - *Analytics Funnel*. Gráfica extraída del sitio Concepto 05. [15]

Capítulo 3

Primeros pasos en el proceso de construcción del tablero omnicanal

3.1 Metodología de Trabajo

El rol del equipo de *Marketing* de la compañía fue central en el proceso de construcción del tablero omnicanal. Básicamente la finalidad del *Marketing* Comunicacional es establecer contacto con potenciales clientes por medio del envío de comunicaciones que integran estrategias diseñadas para obtener los resultados esperados.
Las estrategias que se tomen deben estar en concordancia con los objetivos trazados, y estos tienen que tener en cuenta las herramientas y recursos disponibles. Es por ello que, cómo primera etapa del trabajo del equipo de *Marketing* fue necesario el aprovisionamiento de herramientas de software para *AB Testing*, tanto de asuntos y *preview text* como de diseño de *templates*, y de cada uno de los KPIs que se iban a considerar. Una vez generadas estas herramientas se comenzó con un ciclo CRO con tres ejes de trabajo principales que proveyeron las bases del proceso de omni canalización de las comunicaciones.

- Segmentación de usuarios
- flujos de trabajo
- monitorización

El equipo de desarrollo encargado de las comunicaciones creó en base a los requerimientos iniciales de *Marketing* dos plataformas, una para las pruebas en el cuerpo del mensaje y otra para el par *subject,* o asunto de email y *preview text*, o vista previa del texto.
El *subject* de un *email* es la principal herramienta de los *marketers* para mejorar el *open rate*. Una línea de asunto adecuada puede hacer que un mensaje destaque entre una multitud de *emails* promocionales dentro de una casilla de correo o dentro del espacio de notificaciones de una aplicación *mobile* [16]. El *preview text,* que se muestra en la gráfica de la figura 5, complementa el trabajo del *subject* brindando información más específica de la campaña recibida en la primera lectura de la misma. En la mayoría de las bandejas de entrada de correo electrónico después del remitente y las líneas de asunto se visualiza el contenido del mensaje, ese contenido muchas veces aparece entrecortado y carece de

sentido. Por lo tanto, el *preview text* bien pensado influye de manera positiva en el comportamiento del usuario. A modo de muestra, considérese el caso del reconocido sitio WeddingWire que en un *A/B Test* donde utilizó adecuadamente el *preview text* obtuvo un aumento del 30% en las tasas de clics CTR.

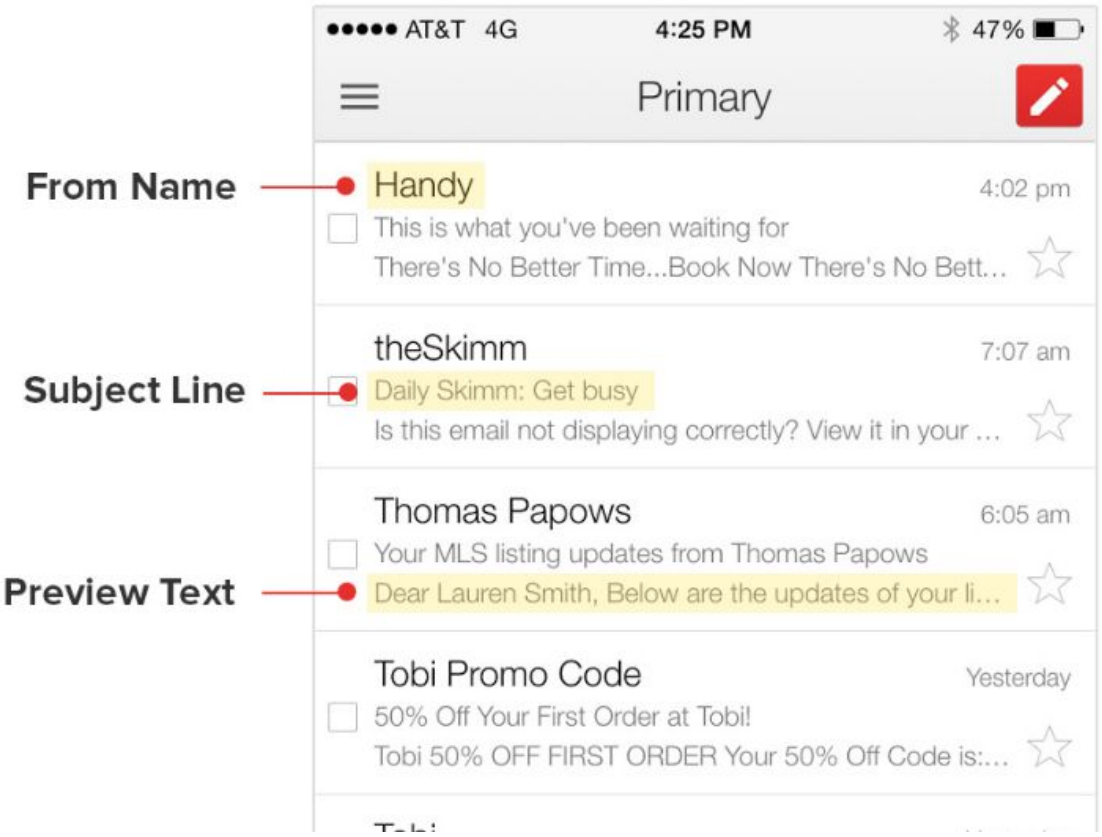

Fig. 5 - *Preview Text.* Extraído del sitio *litmus*.[17]

Luego de contar con las herramientas de *A/B Testing* el proceso de trabajo comenzó con iteraciones sobre distintos componentes de cada comunicación identificando pequeños cambios y acciones que mejoraron los KPIs.

En esta etapa fue importante construir además de las herramientas de *A/B Test* una serie de reportes donde se pudieran obtener rápidamente los resultados de los ensayos.

El proceso de *A/B Testing* logra identificar y agrupar usuarios de acuerdo con sus preferencias. Cada usuario tiene un grupo específico de preferencias comunicacionales derivadas del *tracking* del flujo de interacciones que despiertan las distintas comunicaciones a cada uno y que van a aportar al conjunto de recomendaciones de productos que se ofrecerán a ese usuario en una comunicación.

Por ejemplo, un usuario A puede recibir dos comunicaciones en una semana, la primera refiere a una sábana (grupo) de ofertas con destino de playa, la segunda se compone de una sábana de ofertas de destino de montaña. El usuario A abre ambos *emails*, en ambos decide hacer clic en alguna de las cajas con la oferta ofrecida pero luego en el destino de playa corta con el flujo y en el de montaña llega a simular la compra (habitualmente para confirmar el precio final de la oferta). En este ejemplo, el destino montaña llevó al usuario a etapas más avanzadas dentro del *funnel* de conversión. Por lo tanto, esta interacción del usuario A, si es adecuadamente registrada, va modelando las recomendaciones ideales para él.

El proceso de sondear preferencias es un proceso que se puede iterar y profundizar tanto como se quiera, el desafío consiste en encontrar el punto justo entre mejora y estancamiento por sobre análisis.

3.2 *Tracking* de datos

El primer paso para tener un tablero omnicanal es hacer el *tracking* de los datos de interacción entre la compañía y cada usuario y del comportamiento del usuario luego de recibir la comunicación.

Para llevar a cabo este proceso de *tracking*, la compañía utilizó un conjunto extenso de diferentes herramientas y tecnologías que se nombrarán a lo largo del documento y que por lo tanto es oportuno describir.

3.2.1 *Cloud Computing*

Cloud computing es una tecnología que permite acceso remoto a *software*, almacenamiento de archivos y procesamiento de datos por medio de Internet, siendo así, una alternativa a la ejecución en una computadora personal o servidor local. El modelo de *Cloud Computing* aplica tanto a computación paralela como a distribuida. Las nubes pueden ser construidas con recursos físicos o virtualizados sobre grandes centros de datos centralizados o distribuidos. Algunos autores consideran que la computación en la nube se reduce a servicios y utilerías de cómputo [18]. El *cloud computing* es masivamente usado en la compañía y para la construcción del tablero omnicanal se lo utilizó a través de una solución dual con licencia unificada para operar local y vía SAAS[13] que es provista por Qlikview.

[13] SaaS o *Software as a Service*, es un modelo de distribución de *software* en el que tanto el *software* como los datos manejados son centralizados y alojados en un único servidor externo a la empresa. Esto implica que el *software* utilizado por la empresa no se encuentra en la misma, sino que un proveedor se ocupa del *hosting* de dicho *cloud computing*, así como del mantenimiento y el soporte.

3.2.2 *Server*

El término *server* o servidor tiene dos significados en el ámbito de la informática. El primero hace referencia a una computadora que pone recursos a disposición a través de una red, y el segundo se refiere al programa que funciona en dicha computadora. En consecuencia aparecen dos definiciones de servidor:

- **Definición Servidor (*hardware*):** un servidor basado en *hardware* es una máquina física integrada en una red informática en la que, además del sistema operativo, funcionan uno o varios servidores basados en *software*. Una denominación alternativa para un servidor basado en *hardware* es "*host*" (término inglés para "anfitrión"). En principio, toda computadora puede usarse como "*host*" con el correspondiente *software* para servidores.
- **Definición Servidor (*software*):** cada servidor en un entorno cliente/servidor proporciona un conjunto de servicios compartidos a los clientes. [19]. Un servidor basado en software es un programa que ofrece un servicio especial que otros programas denominados clientes (*clients*) pueden usar a nivel local o a través de una red. El tipo de servicio depende del tipo de *software* del servidor. La base de la comunicación es el modelo cliente-servidor y, en lo que concierne al intercambio de datos, entran en acción los protocolos de transmisión específicos del servicio.

El concepto de servidor en ambas definiciones es un concepto transversal a la compañía y a la historia de *Roadmap* que conduce el desarrollo del tablero omnicanal. Es utilizado para describir cualquier acción que implica servir de una funcionalidad a un cliente de *software* que la solicite.

3.2.3 *Cluster*

Un *cluster* es un multicomputador con memoria distribuida formado por un conjunto de computadoras (nodos) conectadas a través de una red dedicada que se comportan como si fuesen un único recurso de cómputo. Su objetivo principal consiste en mejorar el rendimiento y/o la disponibilidad de un sistema, siendo más económico que una computadora de velocidad y disponibilidad comparables. Esta arquitectura surgió como alternativa a los grandes sistemas multiprocesadores de memoria compartida, y se volvieron plataformas comunes en el cómputo paralelo de problemas complejos debido a sus ventajas en cuanto a la relación costo/rendimiento. Lo podemos definir como un sistema de procesamiento paralelo o distribuido [20].

Podrían verse como objetivo de diseño de los *clusters*:

- Escalabilidad absoluta
- Escalabilidad incremental
- Alta disponibilidad
- Relación precio/prestaciones

[21]

Todas las aplicaciones desarrolladas para la construcción del tablero omnicanal fueron diseñadas para exponerlas a través de servidores dispuestos en *clusters*.

3.2.4 *Storage* (Bases de datos)

El término se refiere a una colección de datos que contiene información relevante para una empresa. Las bases de datos vienen incluidas en un sistema gestor de bases de datos (SGBD) que además de la base incluye un conjunto de programas para acceder a dichos datos [22]. Las bases de datos tienen una amplia clasificación de acuerdo con su función y también se pueden clasificar según su modelo de administración de datos. En el *stack*, o pila de herramientas incluidas en el trabajo que se describe en esta tesina se utilizan bases documentales, clave valor y relacionales.

3.2.4.1 MySQL

MySQL es un sistema de gestión de bases de datos relacionales (modelo ER) de código abierto (*RDBMS*, por sus siglas en inglés) con un modelo cliente-servidor. RDBMS es un software o servicio utilizado para crear y administrar bases de datos basadas en un modelo relacional. Implementa un modelo cliente-servidor que se comunica por medio de SQL.

3.2.4.2 *Datastore*

El *datastore* es un repositorio para almacenar y administrar de manera persistente colecciones de datos que incluyen no solo repositorios como bases de datos, sino también tipos de almacén más simples, como archivos simples y correos electrónicos. Normalmente un *datastore* es una base de datos de documentos NoSQL creada a fin de proporcionar ajuste de escala automático, alto rendimiento y facilidad para el desarrollo de aplicaciones.

3.2.4.3 Casandra

Casandra es una base de datos NoSQL, orientada a columnas y distribuida, es muy popular para la gestión de *big data*.

Proporciona tolerancia a particiones y disponibilidad, pero a cambio de ser eventualmente consistente, tal y como define el teorema de CAP (ver más adelante en este capítulo). El nivel de consistencia puede ser configurado, según nos interese, incluso a nivel de *query* o consulta.

Es distribuida, lo que quiere decir que la información está repartida a lo largo de los nodos del *cluster*. Además ofrece alta disponibilidad, de manera que si alguno de los nodos se cae el servicio no se degradará.

Escala linealmente, lo que quiere decir que el rendimiento mejora de forma lineal respecto al número de nodos que añadamos como se puede ver en la figura 6. Por ejemplo, si con 2 nodos soportamos 100.000 operaciones por segundo, con 4 nodos soportaremos 200.000. Esto da mucha predictibilidad a nuestros sistemas.

Escala de forma horizontal, lo que quiere decir que podemos escalar nuestro sistema añadiendo nuevos nodos basados en *commodity hardware*[14] de bajo coste.

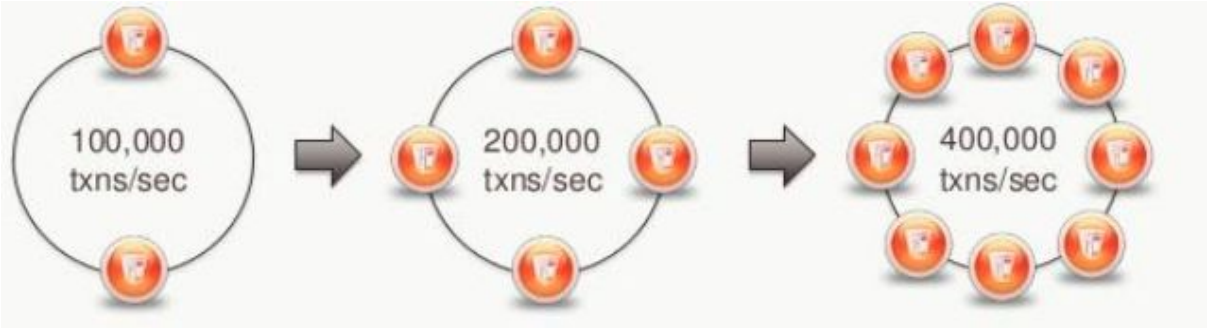

Fig. 6 - Representación del crecimiento de un Cassandra ring. [23]

La distribución de datos que llegan a través de una instrucción *insert* en los diferentes nodos de un *cluster* Cassandra se hace por medio de un componente denominado *partitioner*. *Partitioner* implementa 4 posibles esquemas de trabajo

- *Random Partitioners*

[14] *Commodity Hardware* en un contexto de IT es un dispositivo o componente de dispositivo que es relativamente económico, ampliamente disponible e intercambiable con otro *hardware* de su tipo.

    - murmur 3: es el *default* actual de Cassandra. Es una implementación del algoritmo de *hash* murmur [24] que genera *hashes* de 64 bits.
    - *random partitioner*: utilizado en versiones más viejas de Cassandra, utiliza un *bigInteger token* con una función de *hash* md5 para determinar en qué nodo del *ring* se aloja el dato.
- *Ordered Partitioners*
    - *order preserving partitioner*: utiliza una representación de string UTF-8 para determinar el lugar donde aloja el dato (orden lexicográfico).
    - *byte ordered partitioner*: trata a las *partitions keys* como *arrays* de *bytes* en vez de convertir a *strings* manteniendo el orden.

Una vez que el componente *partitioner* determina el lugar donde se aloja el dato, este se inserta en ese nodo del Cassandra *ring* y en algún otro de acuerdo con el factor de replicación configurado para el *cluster*.

3.2.4.4 MongoDB:

MongoDB es un sistema de bases de datos NoSQL orientada a documentos, desarrollada bajo el concepto de código abierto. Los documentos guardan los datos en formato JSON con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. Permite búsqueda de campos, consulta de rangos y expresiones regulares. Cualquier campo en un documento de MongoDB puede ser indexado y es posible realizar índices secundarios, similar a las bases de datos relacionales. Soporta replicación maestro esclavo y balancea la carga en múltiples servidores [25].

3.2.4.5 Teorema de CAP

Al diseñar servicios distribuidos, hay tres propiedades que comúnmente se desean: consistencia, disponibilidad y tolerancia al particionado.

- Consistencia (***Consistency***): La forma más natural de formalizar la idea de un servicio consistente es como un objeto o entidad cuya manipulación es atómica. La consistencia atómica es la condición que la mayoría de los servicios web experimentan hoy. Bajo una condición garantizada de consistencia, debe existir un orden total en todas las operaciones de manera que cada operación se vea como si

se hubiera completado en un solo instante. Esto es equivalente a exigir que solicitudes de la memoria compartida distribuida se vean como parte de una ejecución realizada en un único nodo, respondiendo a las operaciones de una en una. Esta es la garantía de consistencia que proporciona el modelo más fácil para que los usuarios entiendan, y es más conveniente para aquellos que intentan diseñar una aplicación de cliente que utiliza un servicio distribuido.

- Disponibilidad (***Availability***): Para que un sistema distribuido esté continuamente disponible, cada solicitud recibida por un nodo que no falla en el sistema debe dar como resultado una respuesta. Es decir, cualquier algoritmo utilizado por el servicio debe terminar eventualmente. Esta es una definición débil de disponibilidad ya que no limita el tiempo que el algoritmo puede ejecutarse antes de terminar y, por lo tanto, permite un cálculo ilimitado. La propiedad, cuando se califica por la necesidad de tolerancia de partición, puede verse como una definición fuerte de disponibilidad: incluso cuando haya fallas severas en la red cada solicitud debe terminar.
- Tolerancia al particionado de red (***Partition Tolerance***): Por lo general los ambientes distribuidos están divididos geográficamente, donde es normal que existan cortes de comunicación entre algunos nodos, por lo cual, el sistema debe permitir seguir funcionando aunque existan fallas que dividan el sistema. El sistema debe estar disponible así existan problemas de comunicación entre los nodos, cortes de red que dificultan su comunicación o cualquier otro aspecto que genere su particionamiento.

A partir de lo anterior y dependiendo de lo definido en cada proyecto, sólo es posible garantizar 2 de las 3 propiedades mencionadas anteriormente [26].

Para la construcción del tablero omnicanal la compañía hizo uso de un *cluster* MySQL con un modelo relacional para describir la relación entre campañas y envíos comunicacionales y utilizó los *datastores* Cassandra y MongoDB para el almacenamiento de eventos comunicacionales.

3.2.5 *Big Data*

En Gartner[15] se define al *big data* como los activos de información de gran volumen, alta velocidad y / o gran variedad que exigen formas rentables e innovadoras de procesamiento

[15] Gartner, Inc. es una empresa estadounidense que realiza investigación y análisis de TI. Cuenta con una extensa base de datos de información de mercado y realiza análisis de *benchmarking* sobre TI, finanzas, ventas, *marketing* y operaciones que la posiciona líder referente en este tipo de servicios.

de información que permitan una mejor comprensión, toma de decisiones y automatización de procesos. [27]

El término *big data* se refiere a grandes cantidades de datos que sobrepasan la capacidad del software convencional para ser capturados, procesados y almacenados en un tiempo razonable.
El concepto de *big data* también engloba las infraestructuras, tecnologías y servicios que han sido creados para poder gestionar esta gran cantidad de información.
Big data no se refiere únicamente a los datos, sino sobre todo a la capacidad de poder explotarlos para extraer información y conocimiento de valor para el ámbito en que está siendo aplicado.

3.2.5.1 Hadoop

Hadoop es un *framework* de *software* que aporta la capacidad de ejecutar aplicaciones distribuidas y escalables, generalmente para el sector del *big data*. Así, permite a las aplicaciones hacer uso de miles de nodos de procesamiento y almacenamiento y petabytes de datos. Hadoop es usado en muchas empresas como plataforma central en sus *data lakes*[16], sobre la que se construyen los casos de uso alrededor de la explotación y el almacenamiento de los datos. Está diseñado para ejecutar sobre *commodity hardware* ya que la potencia del sistema se encuentra en que todos los nodos juntos actúan como un *cluster* con la capacidad de escalar horizontalmente al agregar más nodos. En los sistemas tradicionales, las tecnologías se han enfocado en traer los datos a los sistemas de almacenamiento. Hadoop cambia este enfoque y trata de acercar el procesamiento al lugar en donde se encuentran almacenados los datos y así aprovechar técnicas de paralelización, aumentando la escalabilidad y el rendimiento de los sistemas que trabajan con grandes cantidades de datos evitando altas latencias y congestión en la red. Hadoop tiene tres componentes principales.

1. **HDFS:** es el sistema de ficheros distribuido de Hadoop. Tiene capacidad para almacenar los archivos en un *cluster* de varias máquinas. Esta característica es imperante cuando se pretenden almacenar grandes cantidades de datos, puesto que en general no es posible almacenar cientos de terabytes o petabytes en una única máquina. En HDFS, los archivos que se almacenan son divididos en bloques de un mismo tamaño (128 MB) y estos se distribuyen en los nodos que forman el *cluster*.

[16] Un *data lake* es un repositorio de almacenamiento de datos que contienen una gran cantidad de datos en bruto (sin orden y sin jerarquía) que se mantienen allí hasta que sea necesario. Se describe detalladamente más adelante en este documento.

Esta característica hace que el sistema de archivos no funcione de forma óptima con archivos pequeños, por lo que deben evitarse. El tamaño de bloque es configurable. Para conseguir una alta escalabilidad, HDFS usa almacenamiento local que escala horizontalmente. Aumentar el espacio de almacenamiento solamente supone añadir discos duros a nodos existentes o añadir más nodos al sistema. Estos servidores tienen un coste reducido, al tratarse de hardware básico con almacenamiento conectado. HDFS soporta miles de nodos, lo más típico en un *cluster* es desplegar decenas o cientos de nodos y manejar cientos de terabytes, con la capacidad de escalar a decenas de petabytes. Para mantener la integridad de los datos, HDFS almacena por defecto 3 copias de cada bloque de datos. Aunque la replicación de datos no es necesaria para el funcionamiento de HDFS, almacenar solamente una copia podría suponer pérdida de datos frente a fallos o corrupción de archivos, eliminando la durabilidad del dato. La arquitectura de HDFS es de tipo maestro-esclavo. Está basada en dos componentes principales: *Name Nodes* y *Data Nodes* como se puede ver en la figura 7.

a. El *Name Node* (NN) es el maestro o nodo principal del sistema. No se encarga de almacenar los datos en sí, sino de gestionar su acceso y almacenar sus metadatos. Se asemeja a una tabla de contenidos, en la que se asignan bloques de datos a *Data Nodes*. Debido a esto, necesita menos espacio de disco pero más recursos computacionales (memoria y CPU) que los *Data Nodes*. Este componente es el único nodo que conoce la lista de ficheros y directorios del *cluster*. El sistema de ficheros no se puede usar sin el *NameNode* y para conseguir alta disponibilidad hay un *NameNode* activo primario y varios secundarios en espera que actúan como esclavos.
b. Los *Data Nodes* (DN) se corresponden con los nodos del *cluster* que almacenan los datos. Se encarga de gestionar el almacenamiento del nodo. Generalmente usan *commodity hardware*. A causa de su tipología, permiten aumentar la capacidad del sistema de una forma horizontal de forma efectiva y con un coste reducido.

   HDFS tiene un modelo *Write once read many*. Significa que no se pueden editar ficheros almacenados en HDFS, pero sí se pueden añadir datos. En las operaciones de escritura, el cliente debe comunicar la instrucción previamente al *NameNode*. El *NameNode* comprueba los permisos y responde entonces al cliente con la dirección de los *Data Nodes* en los que el cliente deberá empezar a escribir. El primer *Data Node* copiará el bloque a

otro *data node*, que entonces lo copiará a un tercero. Una vez que se hayan completado estas réplicas se enviará al cliente la confirmación de escritura. En las operaciones de lectura, el cliente pide al *name node* la localización de un fichero. Una vez que se han comprobado los permisos del cliente, el *NameNode* envía la localización de los *data nodes* que contienen los bloques que componen el fichero para el cliente. También envía un *token* de seguridad que usará en los *data nodes* como autenticación.

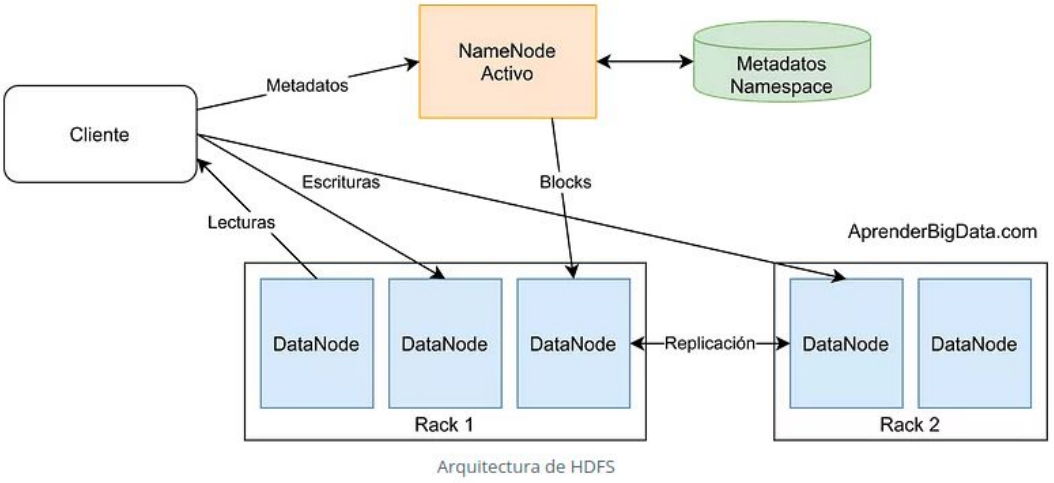

Fig. 7 - Esquema *data nodes* y *name nodes*.

2. ***MapReduce*** es un paradigma de procesamiento de datos caracterizado por dividirse en dos fases o etapas de trabajo diferenciadas: *Map* y *Reduce*. Estos subprocesos, descritos en la figura 8 y asociados a una tarea, se ejecutan de manera distribuida, en diferentes nodos de procesamiento o esclavos. Para controlar y gestionar su ejecución, existe un proceso *Master* o *Job Tracker*. También es el encargado de aceptar los nuevos trabajos enviados al sistema por los clientes. El paradigma de *MapReduce* se basa en enviar el proceso computacional al sitio donde residen los datos que se van a tratar, los cuales se coleccionan en un *cluster* Hadoop. Cuando se lanza un proceso de *MapReduce* se distribuyen las tareas entre los diferentes servidores del *cluster* y, es el propio *framework* Hadoop quien gestiona el envío y recepción de datos entre nodos. Mucha de la computación sucede en los nodos que tienen los datos en local para minimizar el tráfico de red. Una vez se han procesado todos los datos, el usuario recibe el resultado del *cluster*. *MapReduce* como modelo

de programación propone una solución práctica de algunos problemas que pueden ser paralelizados, está basado en Java.

Respecto de las mencionadas etapas de trabajo la función *Map* recibe como parámetro un par <clave, valor> y devuelve una lista de pares. Esta función se encarga del mapeo y se aplica a cada elemento de la entrada de datos, por lo que se obtendrá una lista de pares por cada llamada a la función *Map*. Después se agrupan todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas. No es un requisito que el tipo de datos para la entrada coincida con la salida y no es necesario que las claves de salida sean únicas.

Map (clave1, valor1) –> lista (clave2, valor2)

La operación de *Map* se paraleliza, el conjunto de archivos de entrada se divide en varias tareas llamado *FileSplit* con un tamaño típico de bloque de 128MB. Las tareas se distribuyen a los nodos *Task Trackers*, y estos a su vez pueden realizar la misma tarea si hiciera falta.

La función *Reduce* se aplica en paralelo para cada grupo creado por la función *Map*(). La función *Reduce* se llama una vez para cada clave única de la salida de la función *Map*. Junto con esta clave, se pasa una lista de todos los valores asociados con la clave para que pueda realizar alguna fusión para producir un conjunto más pequeño de los valores.

Reduce (clave2, lista(valor2)) –> lista(valor2)

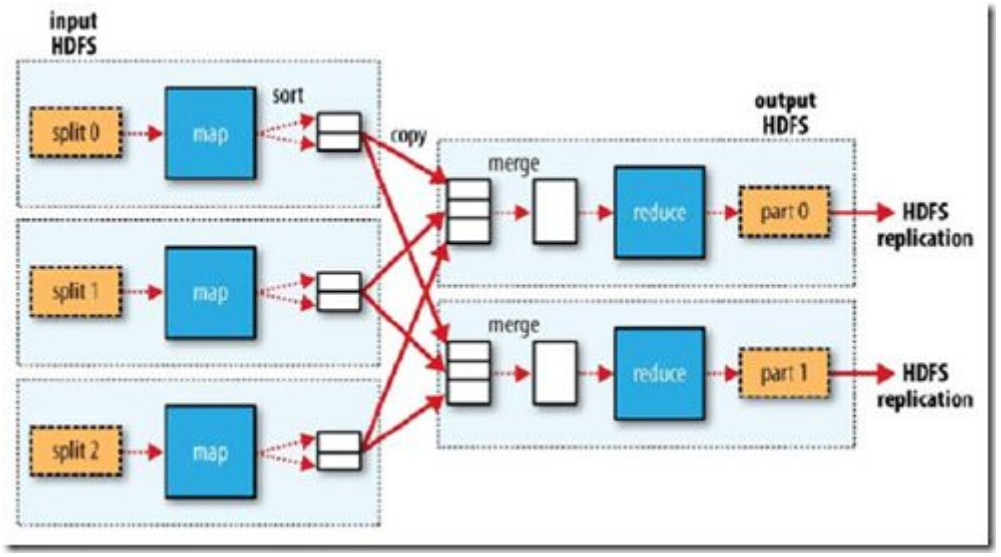

Fig. 8 - Descripción del proceso global de MapReduce. [28]

Cuando se inicia la tarea *reduce*, la entrada se encuentra dispersa en varios archivos a través de los nodos en las tareas de *map*. Los datos obtenidos de la fase *Map* se ordenan para que los pares clave-valor sean contiguos (fase de ordenación, *sort fase*), esto hace que la operación *Reduce* se simplifique ya que el archivo se lee secuencialmente. Si se ejecuta el modo distribuido estos necesitan ser primero copiados al *filesystem* local en la fase de copia. Una vez que todos los datos están disponibles a nivel local se adjuntan a una fase de adición, el archivo se fusiona (*merge*) de forma ordenada. Al final, la salida consistirá en un archivo de salida por tarea *reduce* ejecutada.

Por lo tanto, N archivos de entrada generan M mapas de tareas para ser ejecutados y cada mapa de tareas generará tantos archivos de salida como tareas Reduce hallan configuradas en el sistema.

3. **Yarn** (*Yet Another Resource Negotiator*) es el *framework* que permite a Hadoop soportar varios motores de ejecución y proporcionar un planificador agnóstico a los trabajos que se encuentran en ejecución en el *cluster*. Yarn separa las dos funcionalidades principales: la gestión de recursos y la planificación y monitorización de trabajos. Con esta idea, es posible tener un gestor global (*Resource Manager*) y un *Application Master* por cada aplicación.
    a. *Resource Manager* (RM) tiene dos componentes: El *Scheduler* y el *Applications Manager*. El *Scheduler* o planificador es el encargado de gestionar la distribución de los recursos del *cluster* de Yarn. Además, las

aplicaciones usan los recursos que el *Resource Manager* les ha proporcionado en función de sus criterios de planificación. Este planificador no monitoriza el estado de ninguna aplicación ni les ofrece garantías de ejecución. Por otro lado, el *Applications Manager* es el componente del *Resource Manager* responsable de aceptar las peticiones de trabajos, negociar el contenedor en el que ejecutar la aplicación y proporcionar reinicios de los trabajos en caso de que fuera necesario debido a errores. *El Resource Manager* mantiene un listado de los *Node Manager* activos y de sus recursos disponibles. Los clientes del sistema pueden enviar una aplicación Yarn soportada para ejecutar al *Resource Manager*.

b. *Node Manager* (NM) gestiona los trabajos con las instrucciones del *Resource Manager* y proporciona los recursos computacionales necesarios para las aplicaciones en forma de contenedores. Implementa *Heartbeats*[17] para mantener informado del estado al *Resource Manager*. Los contenedores Yarn tienen una asignación de recursos (cpu, memoria, disco y red) fija de un host del *cluster* y el *Node Manager* es el encargado de monitorizar esta asignación. Mapean las variables de entorno necesarias, las dependencias y los servicios necesarios para crear los procesos.
c. *Application Master* (AM) es el responsable de negociar los recursos apropiados con el *Resource Manager* y monitorizar su estado y su progreso. También coordina la ejecución de todas las tareas en las que puede dividirse su aplicación.

La secuencia de trabajo en Hadoop YARN, que se puede ver en la figura 9., se describe a continuación:

[17] *Heartbeat* es parte de un protocolo de comunicación entre *data nodes* y *name nodes. Data nodes* envía una señal a *name node* después de un intervalo regular de tiempo para indicar su presencia, es decir, para indicar que está vivo.
Si después de un cierto tiempo de latencia, *name node* no recibe ninguna respuesta del *data node*, entonces ese *data node* en particular se declara muerto.

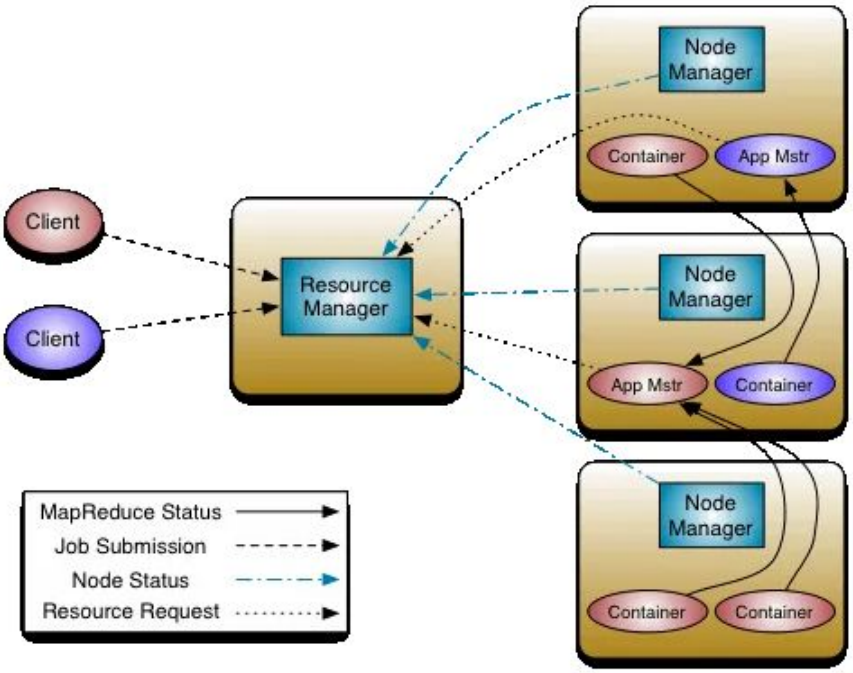

Fig. 9. Arquitectura de Hadoop Yarn. [29]

El cliente envía una aplicación Yarn.

1. *Resource Manager* reserva los recursos en un contenedor para su ejecución.
2. *Application Manager* se registra con el *Resource Manager* y pide los recursos necesarios.
3. *Application Manager* notifica al *Node Manager* la ejecución de los contenedores.
4. Se ejecuta la aplicación Yarn en el contenedor correspondiente.
5. *Application Master* monitoriza la ejecución y reporta el estado al *Resource Manager* y al *Application Manager*.
6. Al terminar la ejecución, el *Application Manager* lo notifica al *Resource Manager*.

3.2.5.2 Ecosistema Hadoop

El ecosistema Hadoop es un conjunto de componentes que te ven en la figura 10 y que son de utilidad en el mundo del *big data.*

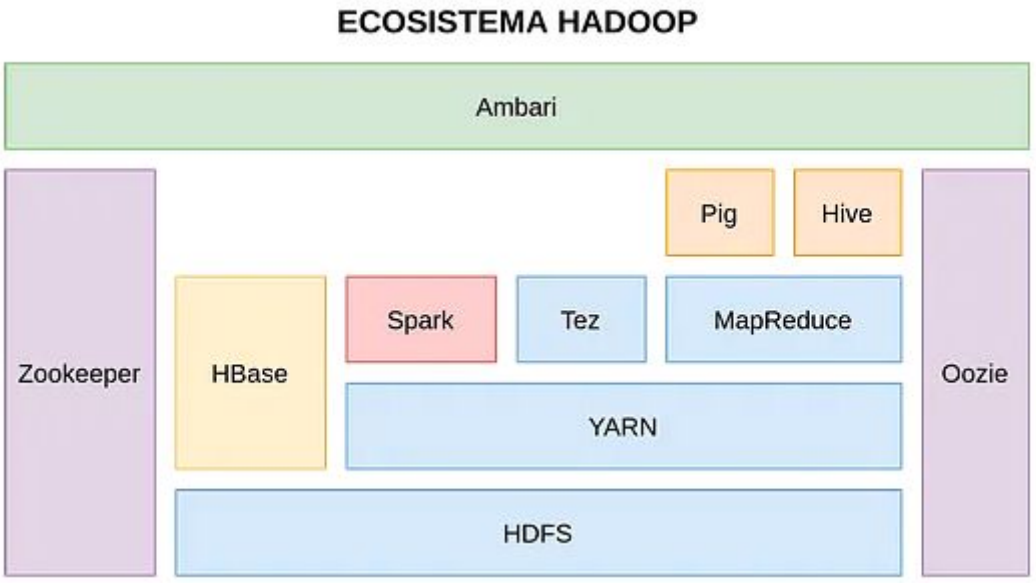

Fig. 10 - Ecosistema Hadoop y Componentes – aprenderBigData.com - site [30]

En cuanto a las distribuciones de Hadoop, la distribución más extendida y usada en la actualidad es Cloudera[18] que se ha encargado de agrupar e integrar gran cantidad de estas tecnologías para poder realizar los despliegues de lagos de datos de una manera sencilla y acorde a las necesidades.

3.2.5.2.1 *Data Lake*

Un *data lake* es un repositorio donde se almacenan todos los datos de la compañía, una gran cantidad de datos en bruto, estructurados y sin estructurar, sin ningún tipo de procesamiento previo (*raw data*) y sin ningún tipo de esquema. Se mantienen allí almacenados hasta que son necesarios para ser analizados.

La información que se almacena en el *data lake* procede de diversas fuentes de datos, por lo que guarda datos de todo tipo: procedentes de bases de datos, documentos ofimáticos, registros de servidores, recursos extraídos de Internet, redes sociales, textos, etc. con el objetivo de ser estudiados y analizados posteriormente.

Un *data lake* debe permitir la ingesta de datos estructurados y no estructurados y almacenarlos con seguridad y con las debidas protecciones de acceso, incluso en tiempo real. Estas restricciones de acceso pueden ser de lectura o de modificación al nivel del dato. Además, debe proporcionar un catálogo que permita a los analistas y profesionales descubrir los datos que contiene.

[18] Cloudera Inc. es una compañía que proporciona software basado en Apache Hadoop, soporte y servicios, y formación para grandes clientes.

Es común que existan varias referencias a los mismos tipos de datos en un único *data lake*. Puede darse el caso de que los datos almacenados tengan etiquetas diferentes pero se refieran al mismo concepto. De esta forma, se debe generar una relación para que los analistas conozcan su existencia.

3.2.5.2.2 Presto DB

Presto es un motor de consulta SQL distribuido de código abierto para ejecutar consultas analíticas interactivas contra fuentes de datos de todos los tamaños, desde gigabytes hasta petabytes. Presto permite consultar fuentes heterogéneas de datos incluyendo Hive[19], Cassandra, bases de datos relacionales o incluso almacenes de datos patentados. Presto puede combinar datos de múltiples fuentes, lo que permite construir un ecosistema *data lake* [31].

3.2.5.2.3 Apache Kafka

Apache Kafka es un sistema de transmisión de datos distribuido con capacidad de escalado horizontal, tolerante a fallos y de alto rendimiento que permite transmitir datos en tiempo real utilizando el patrón de mensajería *publish/subscribe*.

Sus principales funcionalidades son:

- Publicar y suscribirse a flujos de datos (*streams*), actuando de forma similar a un sistema de colas de mensajes pero con un alto rendimiento obteniendo latencias muy bajas en la transmisión de mensajes. Nos ofrece la posibilidad de dividir el procesamiento de datos en múltiples instancias de consumidores, lo que le permite escalar su procesamiento.
- Permite almacenar *streams* y se replican para ofrecer una tolerancia a fallos. Kafka permite a los productores esperar el reconocimiento para que una escritura no se considere completa hasta que esté completamente replicada y se garantice que persiste.
- Facilita el procesamiento de *streams* en tiempo real, pudiendo transformar los datos que se almacenan en Kafka.

Desde Apache recomiendan el uso de Kafka generalmente en dos tipos de aplicaciones:

[19] Hive es una tecnología distribuida diseñada y construida sobre Hadoop que sirve de interfaz SQL. Internamente transforma las consultas SQL en trabajos MapReduce que ejecutan en Hadoop.

- En sistemas o aplicaciones que requieren una transmisión de *streams* entre ellas de manera fiable.
- En sistemas de procesamiento a tiempo real que transforman o reaccionan a los *streams*.

Kafka trabaja con el concepto de tópico, un tópico es un flujo de datos sobre un tema en particular. Se pueden crear tantos tópicos como se necesiten y estos serán identificados por su nombre. Los tópicos pueden dividirse en particiones en el momento de su creación.
Cada elemento que se almacena en un tópico se denomina mensaje. Los mensajes son inmutables y son añadidos a una partición determinada (la elección de la partición se hace por la clave del mensaje o mediante *round-robin* en el caso de ser nula) en el orden en que fueron enviados, es decir, se garantiza el orden dentro de una partición pero no entre ellas.
Cada mensaje dentro de una partición tiene un identificador numérico incremental llamado *offset*. Aunque los mensajes se guarden en los tópicos por un tiempo limitado (una semana por defecto) y sean eliminados, el *offset* seguirá incrementando su valor.
Un *cluster* de Kafka consiste en uno o más servidores denominados Kafka Brokers. Cada miembro de Kafka Broker es identificado por un identificador entero y contiene ciertas particiones de un tópico, no necesariamente todas.
Además, permite replicar y particionar dichos tópicos balanceando la carga de almacenamiento entre los miembros del Kafka Broker. Esta característica permite que Kafka sea tolerante a fallos y escalable.

3.2.5.2.4 Apache Spark

Apache Spark es otro *framework* para procesamiento de grandes cantidades de datos. Es básicamente un motor de procesamiento distribuido en memoria diseñado para ser el sucesor de Hadoop.
Ambos sistemas tienen algunas similitudes:

- Ambos son *frameworks* para el procesamiento de *big data* que tienen arquitectura en *cluster*, es decir, que tienen múltiples nodos.
- Ambos son escalables y tolerantes a fallos.

El uso de Spark es ventajoso frente a Hadoop debido principalmente a tres razones:

- La forma de procesar los datos hace que Spark sea más rápido .
- El estilo de programación, las APIs son más sencillas de usar.

- Spark dispone de componentes específicos, como Mlib para aprendizaje automático, GraphX para grafos, Spark Streaming para tiempo real y Spark SQL.

Cabe mencionar que ambas tecnologías se pueden combinar, ya que es habitual usar HDFS como sistema de almacenamiento de ficheros y Spark para el procesamiento de datos.

Apache Spark orquesta, distribuye y monitoriza aplicaciones que constan de múltiples tareas de procesamiento de datos sobre varias máquinas de trabajo, que forman un *cluster*.

Es posible leer los datos desde diferentes soluciones de almacenamiento persistente como Amazon S3 o Google Storage, sistemas de almacenamiento distribuido como HDFS (ya mencionado), sistemas clave-valor como Apache Cassandra, o buses de mensajes como Kafka.

A pesar de ello, Spark no almacena datos en sí mismo, sino que tiene el foco puesto en el procesamiento. Este es uno de los puntos que lo diferencian de Hadoop, que incluye tanto un almacenamiento persistente (HDFS) como un sistema de procesamiento (*MapReduce*) de una manera muy integrada.

Es importante hablar de la velocidad de procesamiento: la clave es la posibilidad que ofrece Spark para realizar el procesamiento en memoria. Esto, y la extensión del popular MapReduce para permitir de manera eficiente otros tipos de operaciones: *Queries* interactivas y Procesamiento en *Streaming*.

Otro punto fuerte de Spark es que ofrece una serie de APIs que permiten a usuarios con diferentes *backgrounds* poder utilizarlo. Incluye APIs de Python, Java, Scala, SQL y R, con funciones integradas y en general una performance razonablemente buena en todas ellas.

Spark además de contar con MapReduce, amplía su rendimiento gracias a varias funcionalidades: DAG y RDD.

- DAG (*Directed Acyclic Graph*)

  DAG es un grafo dirigido acíclico. Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado *cluster*. En comparación con MapReduce, el cual crea un DAG con dos estados predefinidos (Map y Reduce), los grafos DAG creados por Spark pueden tener cualquier número de etapas. Spark con DAG es más rápido que MapReduce por el hecho de que no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo. MapReduce, sin embargo, debe escribir en disco los resultados entre las etapas Map y Reduce.

  Gracias a una completa API, es posible programar complejos hilos de ejecución paralelos en unas pocas líneas de código.

- RDD (*Resilient Distributed Dataset*)

  Apache Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria. RDD permite realizar operaciones sobre grandes cantidades de datos en *clusters* de una manera rápida y tolerante a fallos. La ejecución de algoritmos iterativos y procesos de minería de datos se gestiona en memoria lo que mejora el rendimiento considerablemente [32].

Una vez que los datos han sido leídos como objetos RDD la API permite realizar dos tipos de operaciones:

- Transformaciones: tras aplicar una transformación, obtenemos un nuevo y modificado RDD basado en el original.
- Acciones: una acción consiste simplemente en aplicar una operación sobre un RDD y obtener un valor como resultado, que dependerá del tipo de operación.

Dado que las tareas de Spark pueden necesitar realizar diversas acciones o transformaciones sobre un conjunto de datos en particular, es altamente recomendable y beneficioso en cuanto a eficiencia el almacenar RDDs en memoria para un rápido acceso a los mismos. Mediante la función *cache()* se almacenan los datos en memoria para que no sea necesario acceder a ellos en disco.
Un modelo de programación típico del *framework* se comporta como sigue:

1. A partir de una variable de entorno llamada *context* se crea un objeto RDD leyendo datos de fichero, bases de datos o cualquier otra fuente de información.

2. Una vez creado el RDD inicial se realizan transformaciones para crear más objetos RDD a partir del primero. Dichas transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean uno nuevo.

3. Tras realizar las acciones y transformaciones necesarias sobre los datos, los objetos RDD deben converger para crear el RDD final. Este RDD puede ser almacenado.

Cuando el programa comienza su ejecución crea un grafo similar al de la figura 11 en el que los nodos son objetos RDD y las uniones entre ellos son operaciones de transformación.
El grafo de la ejecución es un DAG y, cada grafo es una unidad atómica de ejecución.

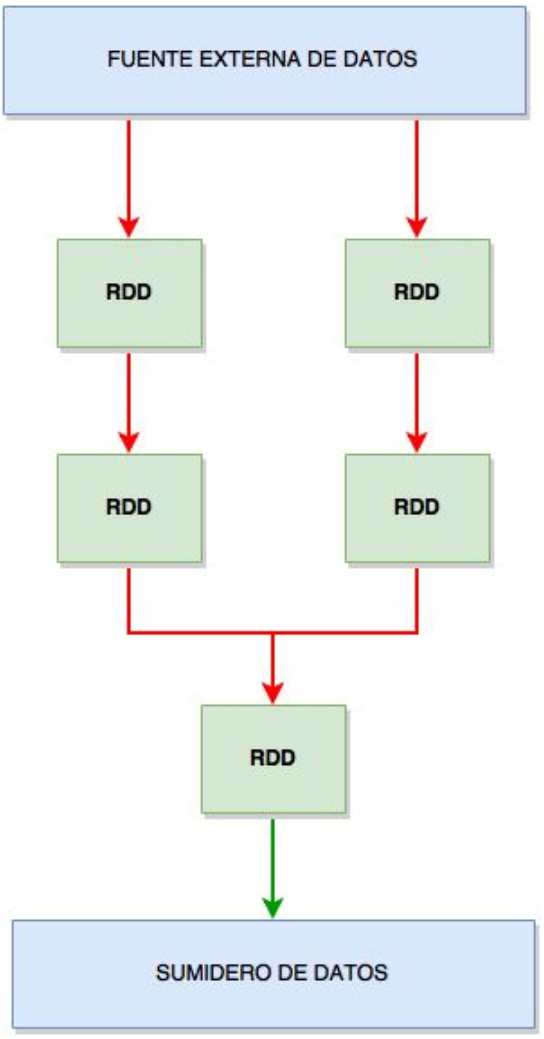

Fig. 11 - Grafo de ejecución. Extraído del sitio geekytheory.com. [33]

En la figura 11, las líneas rojas representan la transformación y las líneas verdes las operaciones.

Transformaciones

Es muy posible que los datos con los que se necesite tratar estén en diferentes objetos RDD, por lo que Spark define dos tipos de operaciones de transformación: *narrow transformation* y *wide transformation*, que se pueden ver en la figura. 12.

- *Narrow transformation*: se utiliza cuando los datos que se necesitan tratar están en la misma partición del RDD y no es necesario realizar una mezcla de dichos datos para obtenerlos todos. Algunos ejemplos son las funciones *filter*(), *sample*(), *map*() o *flatMap*().

- *Wide transformation*: se utiliza cuando la lógica de la aplicación necesita datos que se encuentran en diferentes particiones de un RDD y es necesario mezclar dichas particiones para agrupar los datos necesarios en un RDD determinado. Ejemplos de *wide transformation* son: *groupByKey*() o *reduceByKey*().

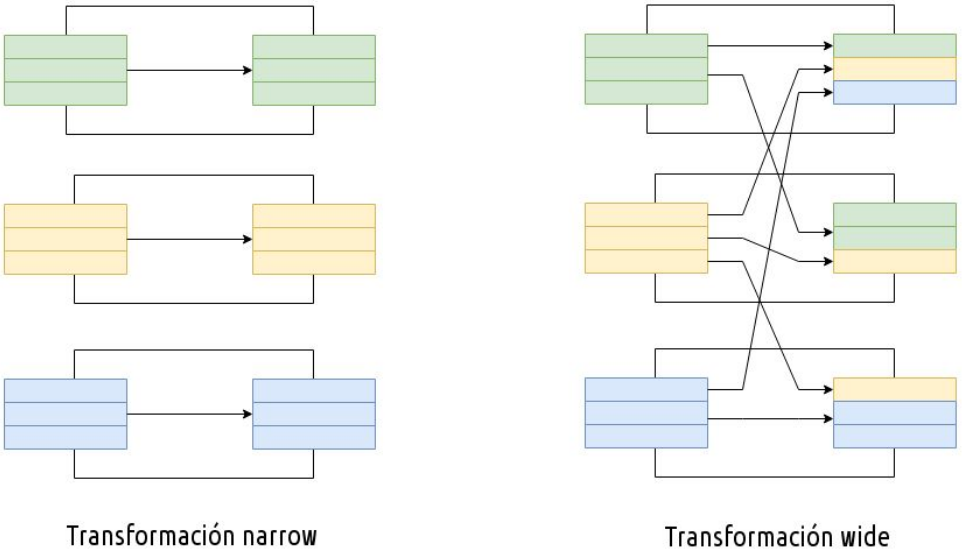

Fig. 12 - Tipo de transformaciones. Extraído del sitio geekytheory.com. [34]

Imaginemos el plan de ejecución de una operación simple

```
SELECT a, b FROM RDD1 JOIN RDD2 WHERE a>5 AND b<10
```

La operación, cómo puede verse en la figura 13, puede realizarse de dos maneras:

1. primero se realiza el JOIN entre los objetos RDD y luego se filtran los datos.
2. primero se realiza el filtrado por separado en ambos RDD y luego se hace el JOIN.

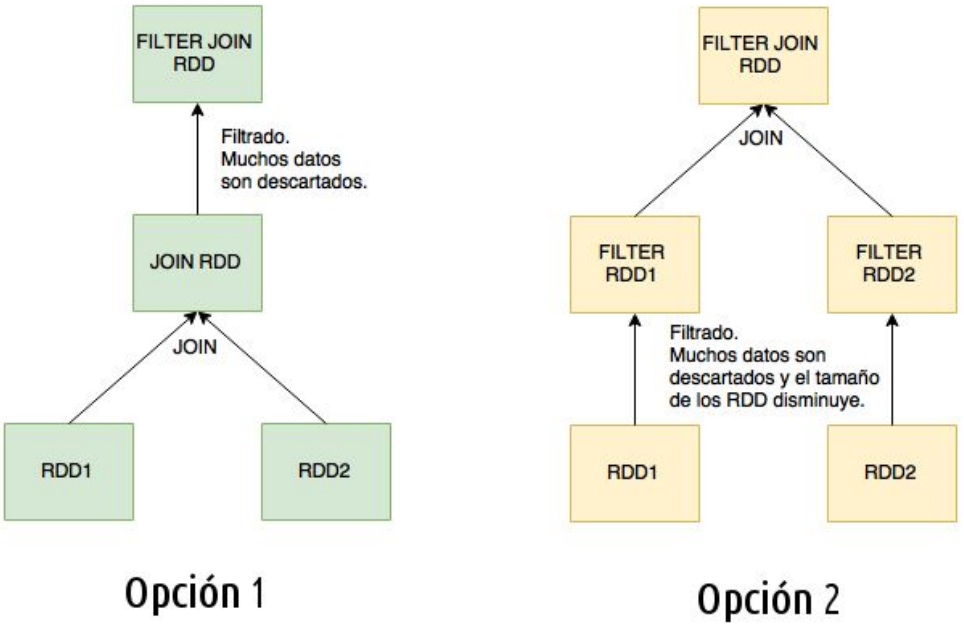

Fig. 13 - Posibles planes de ejecución. Extraído del sitio geekytheory.com. [35]

La segunda opción es más eficiente debido a que el filtrado y posterior unión de los datos se hace por separado. Es decir que el filtrado "*filtering"* de datos es paralelizado y luego el *merge* (altamente costoso) se realiza sobre un conjunto reducido de datos.
El plan de ejecución es el siguiente:

1. Primero se analiza el DAG para determinar el orden de las transformaciones.

2. Con el fin de minimizar el mezclado de datos, primero se realizan las transformaciones *narrow* en cada RDD.

3. Finalmente se realiza la transformación *wide* a partir de los RDD sobre los que se han realizado las transformaciones *narrow*.

3.2.5.2.5 Elasticsearch

Elasticsearch es un servidor de búsqueda basado en la API Apache Lucene[20]. Su especialidad son las aplicaciones que requieran indexado y análisis de texto completo. *Elasticsearch* soporta procesamiento distribuido y es fácilmente escalable, está enfocado al mundo empresarial y científico [36]. Es accesible a través de una extensa y elaborada API.

[20] Apache Lucene es una API de código abierto para recuperación de información utilizado para la creación de motores de búsqueda.

Esta herramienta permite búsquedas extremadamente rápidas haciéndolo un gran colaborador en aplicaciones de descubrimientos de datos.

Es habitual hacer trabajar a Elasticsearch conjuntamente con un agente Beat[21], con un motor de recopilación de datos llamado Logtash, y una plataforma de análisis y visualización llamada Kibana (ver Fig. 14). Los productos conforman una solución ampliamente difundida en la industria que se denomina “Elastic *Stack*”. [37]

Z

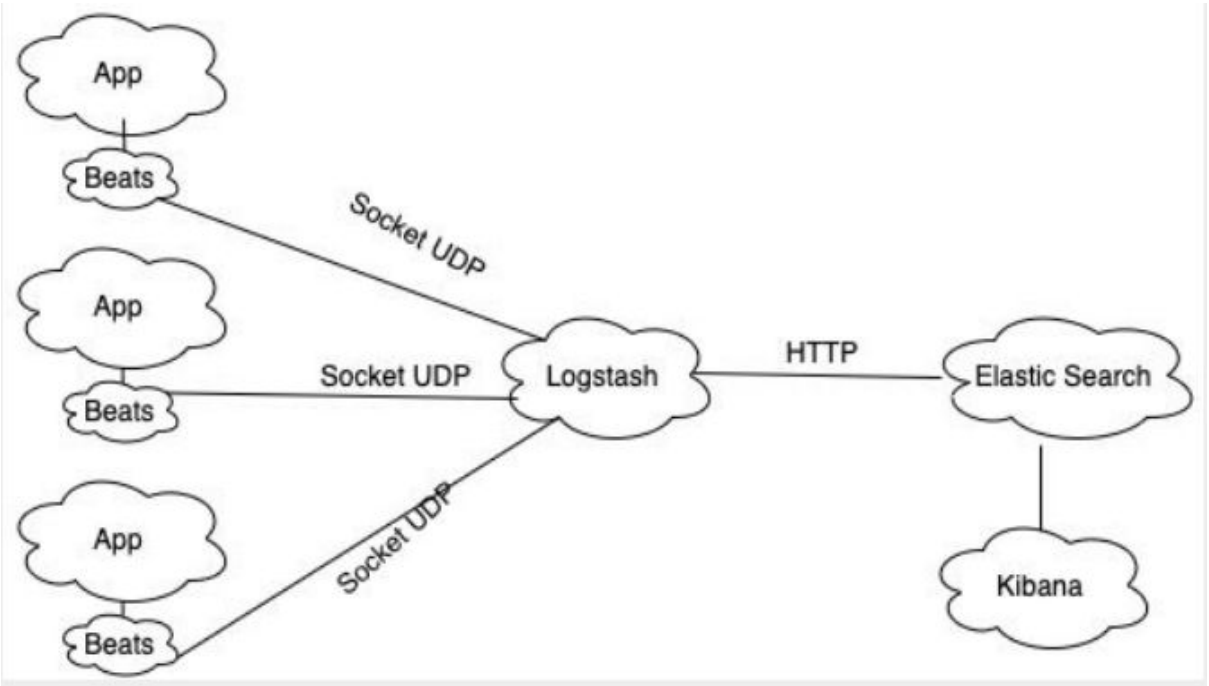

Fig. 14 - Elastic Stack

Elasticsearch cuenta con un repositorio de datos NoSQL orientado a documentos JSON, al estilo de MongoDB, esta característica hace que no necesite que se definan esquemas a la hora de insertar los datos. El *framework* permite indexar grandes volúmenes de datos, para poder consultarlos posteriormente, cada documento JSON es guardado como un par clave/valor donde las claves son cadenas de texto y los valores pueden ser cadenas, números, fechas o listas.

El proceso de añadir información a Elasticsearch se llama “indexación”. Ya que cuando insertamos datos en Elasticsearch lo que se está haciendo es insertar en los índices de Apache Lucene.

[21] Beats es una plataforma gratuita y abierta para agentes de datos con un solo propósito. Envían datos de cientos o miles de máquinas y sistemas a Logstash o Elasticsearch.

Cómo repositorio Elasticsearch es una base de datos distribuida que escala horizontalmente de manera dinámica, por lo que a mayor demanda podemos ir creciendo en nodos.

Elasticsearch también es asíncrono y concurrente (utiliza un control de concurrencia optimista OCC), lo que significa que estas solicitudes de replicación se envían en paralelo y pueden llegar a su destino fuera de secuencia, por lo expuesto necesita una forma de garantizar que una versión anterior de un documento nunca sobrescriba una versión más nueva.

Los documentos son particionados mediante técnicas de *sharding*[22]. El *sharding* distribuye los documentos en diferentes nodos (ver Fig. 15) permitiendo que las cargas de búsqueda se distribuyan por los diferentes nodos, logrando una mejora en los tiempos de respuesta del sistema.

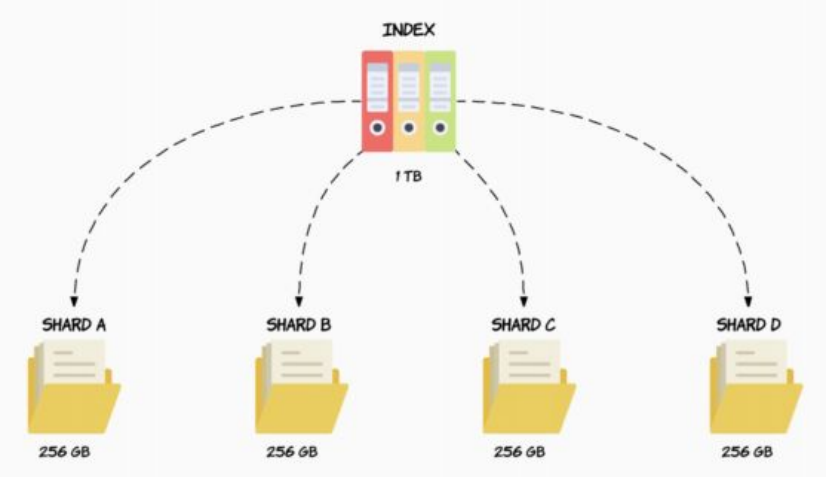

Fig. 15 - *Sharding process. Indexed logs with kibana.* Extraído del sitio Coding Explained.[38]

Al distribuir los índices por diferentes nodos dentro del *cluster* se deben tomar una serie de medidas por si alguno de los nodos falla. Es por eso que Elasticsearch nos ofrece la capacidad de realizar réplicas de los datos.

En la figura 16 por ejemplo, en un esquema de tres nodos se nos cae uno y tenemos cada nodo configurado con un *shard* y una réplica de otro entonces el *stack* no ve reflejada la caída. La configuración está pensada con este objetivo, así las réplicas de un nodo se encuentran en un nodo contiguo a donde se encuentra el *shard* original.

[22] *Sharding* refiere a *shard* (división) y es una técnica que permite en la creación de un índice indicar la cantidad de *shards* en que queremos dividirlo.

A three-node cluster—shards. Elasticsearch: The Definitive Guide. Clinton Gormley, Zachary Tong. 2011

Cluster after killing one node. Elasticsearch: The Definitive Guide. Clinton Gormley, Zachary Tong. 2011

Fig. 16 - *Cluster* shard example. [39]

3.2.5.2.6 Hazelcast:

Hazelcast es un IMDG. Un IMDG *In Memory Data Grid* es un *cluster* de computadoras que agrupa su memoria de acceso aleatorio (RAM) para permitir que las aplicaciones compartan datos con otras aplicaciones que se ejecutan en el *cluster (Fig. 17)*. Hazelcast está diseñado para el procesamiento de datos a alta velocidad. Es común su uso cuando se necesita procesamiento a gran escala que necesita más memoria RAM de la que normalmente está disponible en un solo servidor informático. Esto permite el mayor rendimiento de las aplicaciones mediante el uso de memoria RAM junto con la potencia de procesamiento de varias computadoras que ejecutan tareas en paralelo. Es especialmente valioso para aplicaciones que realizan un procesamiento paralelo extenso en grandes conjuntos de datos.

En Hazelcast todos los nodos del *cluster* generalmente se ejecutan en un mismo *data center*[23]. Esta configuración local se realiza para mantener el alto rendimiento esperado de las tecnologías en memoria, ya que la coordinación de estructuras de datos sobre computadoras geográficamente remotas puede ser un cuello de botella.

[23] Se denomina *data center* al espacio donde se concentran los recursos necesarios para el procesamiento de la información de una organización. Es una construcción que contiene equipos electrónicos necesarios para poder mantener una red de computadores funcionando con protocolos de seguridad y control.

Hazelcast funciona ejecutando *software* especializado en cada computadora de un *cluster* para coordinar el acceso a los datos para las aplicaciones. Cada computadora del *cluster* tiene su propia vista de los datos y las estructuras de datos en la memoria, pero la vista se comparte entre todas las demás computadoras. El *software* realiza un seguimiento de todos los datos en cada nodo individual, de modo que los datos se pueden compartir con cualquier otro nodo o aplicación. Esta orquestación oculta la complejidad de recuperar y actualizar datos en la red, lo que simplifica el desarrollo de aplicaciones.

La herramienta cuenta con una amplia estructura de datos para almacenar los datos provenientes de su procesamiento como mapas, listas y colas. También se incluyen tipos de datos básicos ("primitivos") como números enteros y números de coma flotante. Cada uno de estos objetos y tipos de datos se representan como variables en una aplicación, y la lógica de la aplicación hace referencia a estas variables como si residieran en la misma computadora que ejecuta la aplicación. Esto hace que el paradigma de programación sea mucho más simple que otras tecnologías en memoria, ya que el desarrollador no necesita incluir código para recuperar datos físicamente.

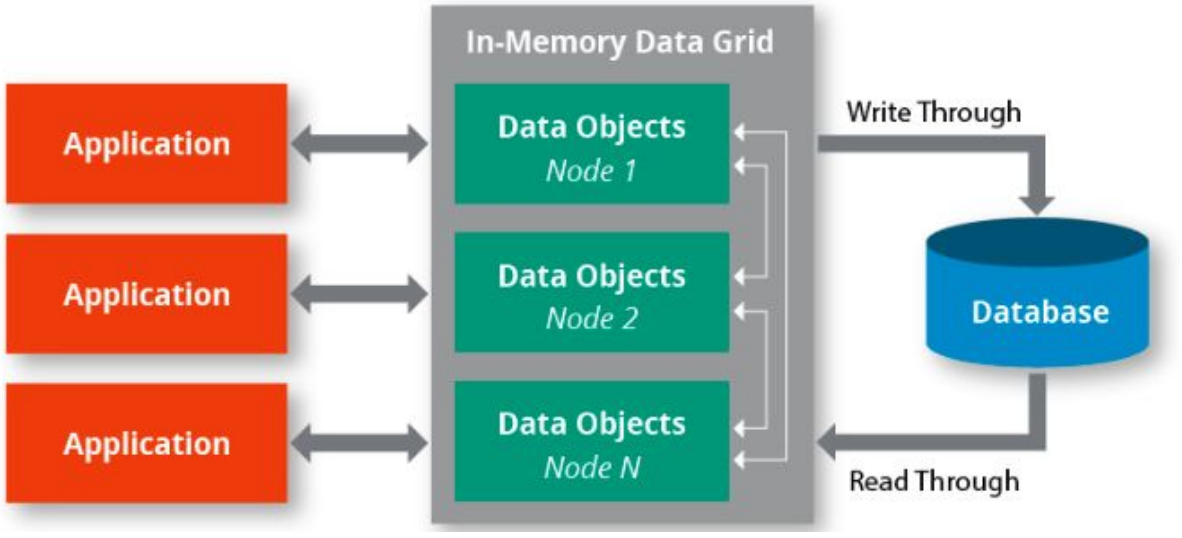

Fig. 17 - Diagrama de un *datagrid*. Extraído del oficial [40]

3.2.5.2.7 Apache Avro

Avro es un *framework* de serialización de datos y RPC[24] orientado a filas desarrollado dentro del proyecto Hadoop de Apache. Utiliza JSON para definir tipos de datos y protocolos, y serializa datos en un formato binario compacto. Su uso principal es dentro del Ecosistema de Apache Hadoop, donde puede proporcionar un formato de serialización para

[24] *Remote Procedure Call* (RPC) es un programa que utiliza una computadora para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas.

datos persistentes. Avro usa un esquema para estructurar los datos que se codifican. Tiene dos tipos diferentes de lenguajes de esquema; uno para edición humana (Avro IDL) y otro legible por máquina basado en JSON.

3.2.5.2.8 Apache Parquet

Parquet es un formato de archivo de código abierto disponible para cualquier proyecto en el ecosistema Hadoop. Apache Parquet está diseñado para un formato de almacenamiento de datos en columnas planas eficientes y de alto rendimiento en comparación con archivos basados en filas como archivos CSV o TSV. Parquet se construyó con estructuras de datos anidadas complejas y utiliza el algoritmo de ensamblaje y destrucción de registros descrito en el documento de Dremel [41] en contraposición con el enfoque de aplanamiento de espacios de nombres anidados utilizado en los archivos basado en filas [42].
El formato de Parquet está compuesto por tres piezas (Fig. 18) :

- *Row group:* es un conjunto de filas en formato columnar, con un tamaño entre 50Mb a 1Gb.
- *Column chunk*: son los datos de una columna en un grupo. Se puede leer de manera independiente para mejorar las lecturas.
- *Page*: Es donde finalmente se almacenan los datos debe ser lo suficientemente grande para que la compresión sea eficiente.

En entornos YARN es necesario indicar cuánta memoria puede utilizar un nodo para asignar recursos con el parámetro.

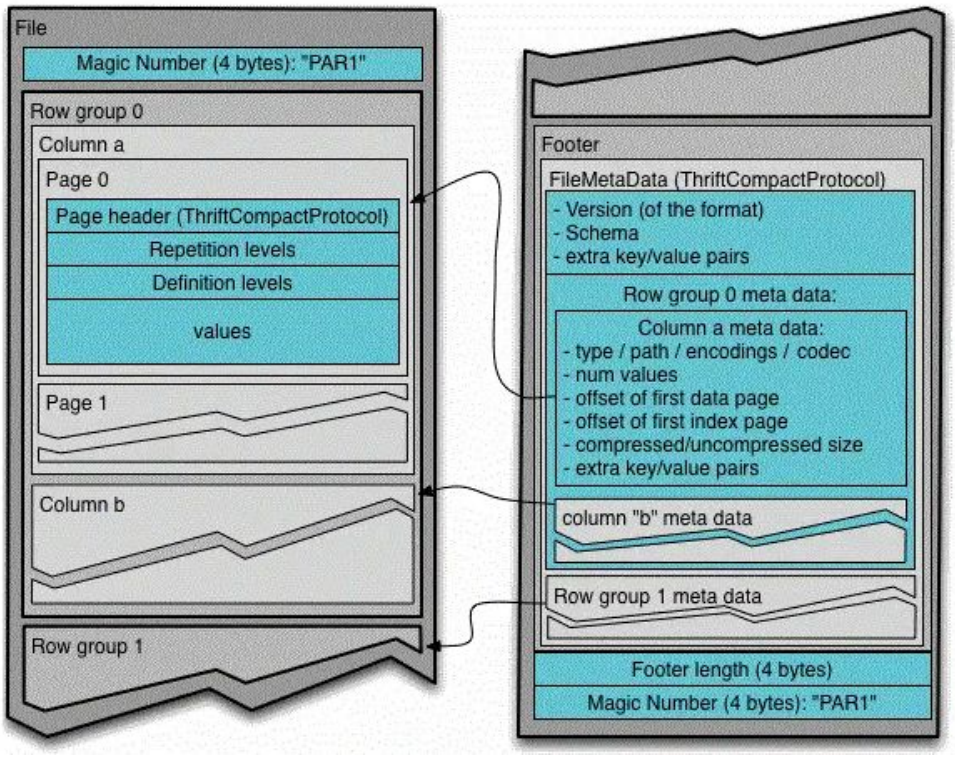

Fig. 18 - Estructura de un archivo parquet. Extraído del de Diego Calvo [43]

El ecosistema Hadoop es utilizado en todas las etapas de trabajo (recolección, ETL y presentación de datos) que conducen a la exposición de información a través del tablero omnicanal. Los procesamientos de spark con funciones *map* y *reduce* en todas las etapas, hazelcast para el procesamiento de la información que nutre las herramientas de asuntos de *email*, *Zookeeper*[25] para distribuir el trabajo que tiene la coordinación, Avro y Parquet para el gobierno de datos, etc.

[25] ZooKeeper es un proyecto Apache de código abierto que proporciona un servicio centralizado para proporcionar información de configuración, nomenclatura, sincronización y servicios agrupados en *clusters* de sistemas distribuidos. El objetivo es hacer que estos sistemas sean más fáciles de administrar con una propagación de cambios mejorada y más confiable.

Capítulo 4

Reestructuración del proceso de registro de actividades de usuario.

4.1 Arquitectura de las aplicaciones

La arquitectura propuesta para las aplicaciones nuevas creadas para reestructurar el proceso de registración de usuario ha sido una arquitectura basada en microservicios. La arquitectura de microservicios es un método de desarrollo de aplicaciones *software* que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa. Cada microservicio, que puede codificarse en un lenguaje de programación diferente desempeña una función específica. Los microservicios se comunican entre sí a través de RESTful APIs [26], y cuentan con sistemas de almacenamiento propios lo que evita la sobrecarga y caída de la aplicación.

4.2 Proceso de registración

El proceso de registrar eventos comunicacionales comienza con la necesidad de enviar la comunicación promocional por parte de un cliente. Un cliente, en este contexto, es un equipo de *Marketing*, Comercial o Posventa que quiere enviar una comunicación a un grupo de usuarios.

Generalmente la comunicación es una respuesta a una acción de usuario (comunicación reactiva), o es un alerta creada por un usuario (que le permite recibir comunicación de concordancia de precio entre lo que él pidió y la oferta publicada) o es un evento comunicacional programado que se envía a usuarios suscriptos (normalmente con distintas frecuencias y audiencias[27]).

Ante la necesidad de comunicación a sus usuarios el cliente remitente invoca a un servicio que provee de utilidades para envío de comunicaciones. Este servicio es una aplicación escrita en Java y distribuida en nodos dentro de un *cluster* distribuido, la aplicación expone

[26] Una API que describe la forma en que los programas o los sitios webs intercambian datos. Rest (**RE**presentational **S**tate **T**ransfer) es un tipo de arquitectura que se apoya en el protocolo http y se compone de una lista de reglas que se deben cumplir en el diseño de la arquitectura de una API**.** Restful es una capa de servicios que cumplen la arquitectura Rest

[27] En *Marketing* Digital, la audiencia es el conjunto de individuos que recibe o puede recibir una comunicación. Las audiencias suelen dividirse o segmentarse en función de diversas variables como la edad, el sexo, sus intereses o la forma en la que reciben e interactúan con las comunicaciones.

una API para comunicarse con ella y provee servicios de *encoding*[28], coordinación, envío y registración.

4.2.1 Coordinación

El punto de entrada del proceso que finalizará en la construcción del tablero omnicanal es el envío comunicacional a un usuario y es por eso que surge la necesidad de comenzar con la registración de datos. El mensaje enviado puede ser un envío manual proveniente de una iniciativa del equipo de *Marketing* que crea una campaña, diseña su imagen (esto a través del trabajo de un equipo de UX[29] propio) o puede derivar de una campaña automatizada.

Antes de realizarse el envío del mensaje, éste pasa por varias etapas de trabajo. En la primera etapa se realiza una serie de validaciones verificando que el *request* de envío tiene los datos mínimos para que pueda ser enviado. La siguiente etapa es ejecutada por un componente que se denomina *Coordinator*. El coordinador es una aplicación que expone un servicio que determina si un mensaje puede ser enviado a un usuario.

El coordinador es una aplicación que fue pensada para no abrumar a los usuarios con demasiadas comunicaciones. Uno de los pilares de la omnicanalidad es “ponerse en los zapatos del usuario”. Consecuentemente, evitar el *spam* (mensaje recibido no deseado), fue un objetivo primordial en todos los equipos que trabajaron en el proyecto.

El coordinador, es una aplicación desarrollada en Java que posee una capa de servicios *RESTful* y un *datastore* Cassandra que responde sobre la viabilidad del envío de una comunicación a un usuario.

Básicamente el coordinador, cada vez que recibe un *request* con un identificador de usuario, busca en su modelo de datos. Las tablas de Cassandra, contienen eventos aceptados y eventos rechazados que son permisos de solicitud de envío de comunicación aceptada o rechazada respectivamente. Cada evento (sea aceptado o rechazado) tiene además del identificador del usuario, el identificador de la comunicación, el *brand*[30] y un dato de tipo *timestamp* que determina el momento en que sucedió ese evento. La tabla tiene un TTL (*Time To Live,* o tiempo de vida) de varios días. Así la búsqueda que se ejecuta trae el historial de eventos sucedidos para ese usuario en el lapso de tiempo tenido en cuenta. Con esta información se nutre un motor de decisión que tiene una serie de reglas de coordinación y determina si se puede realizar el envío por el que se ha hecho el *request*.

---

[28] *Encoding* implica nutrir las URLs con parámetros codificados.

[29] El trabajo de un equipo de UX hace referencia a mejorar un producto/servicio y que sea funcional, lo que dejará en el usuario una buena experiencia al momento de consumirlo/usarlo.

[30] El *branding* de marca o de empresa es el proceso mediante el cual se construye una marca (*brand*), comprendiendo este como el desarrollo y mantenimiento de un conjunto de atributos y valores inherentes a la marca y por la que esta será identificada por su público.

Si el motor de decisión determina que se acepta el envío, entonces se devuelve una respuesta afirmativa, se modela un nuevo evento de aceptación y se lo inserta en la tabla de eventos aceptados. Si por el contrario el motor de decisión determina que se rechaza el envío entonces se envía una respuesta negativa, se modela un nuevo evento de rechazo y se lo inserta en la tabla de eventos rechazados.

La coordinación mejoró notoriamente las métricas de *Open Rate* y *Click Through Rate* y permitió entender el comportamiento de los usuarios respecto de la frecuencia de envíos y generar audiencias agrupadas por frecuencia.

4.2.2 *Encoding*

Una vez superada la etapa de coordinación, si el envío del mensaje fue aceptado, la siguiente etapa es la de trabajar en la codificación de las URLs y *deeplinks*[31] (esto depende del canal de comunicación que se utilice) contenidos en el *payload*[32] de las comunicaciones que se están por enviar.

Desde el punto de vista de la solución de *software* la aplicación expone una API que propone un *payload* que se utiliza en el *request* que se hace al recurso que ofrece servicios de envío. Dentro del *payload* hay una jerarquía de datos necesarios para hacer *tracking*, para Google *Analytics*, para nutrir el modelo de atribución[33] y para obtener el rendimiento de las comunicaciones. Con estos datos y haciendo uso del patrón "*Chain of Responsibility*[34]" se van a codificar cada uno de los *deeplinks* de una comunicación.

Cada *link* luego de la cadena de procesamiento, va a quedar codificado y con los parámetros y redirecciones necesarias para que cada *open*, *click* / *tap* de los usuarios dirija el tráfico hacia una aplicación que registre el evento antes de enviar al destino propuesto por el equipo comercial.

4.2.3 Envío y Registración

Una vez concluido el *encoding* de las URLs y *deeplinks*, la aplicación toma dos trabajos en paralelo, por un lado negocia con una aplicación *sender* (que se encarga del envío puntual) y por otro lado registra el envío comunicándose con una aplicación que se encarga de las registraciones de mensajería.

---

[31] Un *deeplink* es un enlace que lleva a un contenido específico dentro de una app o web.

[32] En informática, el payload es la parte de datos transmitidos que es efectivamente el mensaje real previsto. El concepto de *payload* no incluye la información enviada con ella como los encabezados o metadatos. Se suele traducir como carga útil.

[33] Un modelo de atribución es un conjunto de reglas por las cuales se asigna un determinado valor a los distintos canales por los que un usuario ha pasado antes de realizar una acción que interpretamos como una conversión.

[34] *Chain Of Responsibility* es un patrón de diseño mencionado en el libro Patrones de Diseño de Eric Gamma. [44]

Envío

La aplicación *sender* tiene la responsabilidad de enviar las comunicaciones. La estrategia de envío depende del canal por donde se despache la comunicación:

- ***email***: si el mensaje que se está por enviar es un *email,* el *sender* se va a encargar de encolar mensajes en distintos puertos SMTP de salida. En este punto fue muy importante mantener la reputación de las direcciones IP asociadas a los puertos, una buena reputación aumenta la entregabilidad. La entregabilidad, que es una métrica que se desprende de tres de los indicadores KPIs que se trazaron al comienzo del trabajo, es el resultado de restarle a los eventos enviados aquellos que fueron rebotados.

#delivered = #sents - (#soft bounce + #hard bounce)

Los proveedores de *email* y de ISP[35] de los usuarios destinatarios son cada vez más sofisticados a la hora de decidir qué *emails* deben ser bloqueados, filtrados como *spam*, entregados sin imágenes o entregados en la bandeja de entrada. Es así cómo el dominio y la reputación IP se han convertido en algo crítico de las comunicaciones de *email*. Manteniendo el dominio y un grupo de IPs con buena reputación, se baja la tasa de mensajes filtrados por *spam*, y se suben tanto *Open Rate* como *Click Through Rate*. Para mantener alta la reputación es necesario considerar una gran cantidad de ítems: diseño del *email*, calidad del contenido, asunto, *preview text*, frecuencia de envío, etc. El *sender* encola en los puertos con más alta reputación a las mejores comunicaciones, es decir, las que destacan por su calidad de contenido o puntualidad (por ejemplo, un *email* que reacciona a una búsqueda intensiva de un usuario). Por otro lado, las comunicaciones más genéricas o dirigidas a segmentos de usuarios con menos *engagement*[36], son identificadas por el *sender* para ser enviadas por los puertos con menor reputación.

Además de elegir el puerto de salida más adecuado, el *sender* se ocupa de gestionar los *bounces* que se dan en el transcurso del envío. Un *bounce* o rebote es una dirección de *email* que no puede recibir *emails* debido a un error, es decir, los emails son rechazados. Ese error puede ser temporal (*soft bounce*) o permanente

[35] El proveedor de servicios de Internet, (ISP, por las siglas en inglés de *Internet Service Provider*) es la empresa que brinda conexión a Internet a sus clientes.
[36] *Engagement* es un término original del inglés que, en español, se usa para determinar el compromiso que se establece entre una marca y su audiencia en las distintas comunicaciones que producen entre sí.

(*hard bounce*). Mientras que los *soft bounces* normalmente vienen de direcciones que tienen la bandeja de entrada llena o un fallo de acceso temporal, los *hard bounces* suelen proceder de direcciones que ya no existen o fueron abandonadas. El *sender* recibe los *emails* rebotados ya sea porque se comunica con el SMTP de salida o el del cliente y con esa información genera un evento de *bounce* y lo registra enviándole un *request* POST a una aplicación encargada de las registraciones.

- ***push***: si la comunicación a enviar es un *push mobile,* el *sender* va a negociar la entrega con APNS[37] si el dispositivo destinatario es IOS o con FCM[38] si es Android. En este canal la gestión de *bounces* tiene como participante al sistema operativo del dispositivo destinatario quien recepciona el mensaje e intenta comunicarse con la aplicación móvil correspondiente. Si la aplicación no se encuentra disponible por un cambio en la configuración del usuario o por una desinstalación entonces el sistema operativo informa al remitente del evento y éste (el *sender*) genera el evento de *bounce* y lo envía a la aplicación que registra los eventos.
- ***web push***: otro canal preparado para las notificaciones es el *web push notification*, en este canal la negociación se da entre el *sender* y el *service worker*[39]. Existe la dependencia con un *service worker* para implementar *web push messages*. La razón de esto es que cuando se recibe un mensaje *push*, el navegador puede iniciar un *service worker*, que se ejecuta en segundo plano sin que se abra una página, y enviar un evento para que pueda decidir cómo manejar ese mensaje *push*. De manera análoga a las notificaciones *push* aquí el *service worker* puede informar de un navegador, que bloqueó la recepción de mensajes o que se ha desinstalado. En cualquier caso, el *service worker,* informa el evento al *sender* para que este cree y registre un evento de *bounce*.

El ciclo de vida del *sender* incluye en resumen la gestión de envíos y de registración de *bounces*.

Registración de los envíos

La aplicación encargada de la registración de los eventos provenientes de la interacción entre el usuario y el site expone diferentes recursos REST a través de una API. Esta API cubre todos los tipos de eventos que pueden generarse y lo hace para la totalidad de los

[37] El Apple Push Notification Service es un servicio móvil creado por Apple Inc, que usa la tecnología *push* para enviar notificaciones de los servidores de aplicaciones de terceros para sus productos, las notificaciones pueden incluir logos, canciones o alarmas de texto personalizadas.

[38] FCM es una tecnología que permite el envío de *push notifications* a dispositivos Android.

[39] Un *service worker* es una secuencia de comandos que responden a una API que un navegador ejecuta en segundo plano, separado de una página web, abriéndose la puerta a funciones que no necesitan una página web ni interacción de usuario.

canales posibles. La aplicación que orquesta el flujo de *software* que se encarga del envío y registro de una comunicación, negocia paralelamente con el *sender* para que realice efectivamente el envío y solicita, por medio de un *request*, el registro de estos eventos de *sent* al subsistema que se encarga de la registración de eventos comunicacionales.

Un evento de envío o "*sent event*" es el primer tipo de evento que se registra en orden temporal para un identificador único de mensaje. Llega al punto de entrada del servicio de registración y es modelado y registrado en diferentes aplicaciones *datastore* cuya creación responde a diferentes objetivos. Cada *datastore* es un *cluster* distribuido que expone servicios de *storage* y recupero de datos a la aplicación que expone la API de registración:

- **Cassandra *Datastore***: el *datastore* vive en un *cluster* distribuido con 5 nodos. El modelo de datos construido responde a la metodología *query-driven data modeling*[40]. Se tienen dos tablas para la ingesta de datos en línea una cuyo modelo está orientado al mensaje (tabla "*messages*") y otra cuyo modelo está orientado al evento (tabla "*events*"). La tabla "*messages*" contiene elementos como el asunto, el diseño, el *preview text*, el legal incluido y un campo multivalor para contener información centrada en el mensaje enviado. La tabla de "*events*" guarda datos orientados al tipo de evento como, el *user agent* con el que se interactúa, la compañía remitente, el *referer*[41], etc. Registrar mensajes permite hacer procesamiento útil para tomar mediciones sobre asuntos, *preview text,* diseños, uso de AMP[42]. Esta información por cuestiones de rendimiento serán procesadas por un proceso *data batch* que luego se describe.

  Los eventos registrados sirven como *storage* de datos que serán utilizados en todo el ciclo de vida de los eventos. La tabla de eventos tiene como *primary key* el identificador único de mensaje. Este identificador asignado a cada envío por la aplicación que provee de las comunicaciones, acompaña a los distintos tipos de eventos generados a partir de una necesidad comunicacional. Teniendo como *primary key* este identificador, las escrituras y lecturas de eventos se realizan con alta velocidad.

  Ya se ha descrito en esta tesina cómo una comunicación codifica todos sus *deeplinks* en una de las etapas de trabajo. Precisamente uno de los parámetros que

[40] Es una metodología que propone que el diseño se realice en función de las necesidades de consultas sobre la base. [45]
[41] Un HTTP *referer*, o referenciador HTTP, se define como el origen del visitante a una web, generalmente una URL. El término "*Referer*" o referenciador se utiliza debido a un error ortográfico en la especificación HTTP original, y este error ortográfico se ha convertido en el valor predeterminado.
[42] AMP (*Accelerated Mobile Pages*) es un marco de trabajo de código abierto de componentes web que permite crear fácilmente sitios, historias, correos electrónicos y anuncios en los que se les da prioridad a los usuarios y con un tiempo de carga casi instantáneo.

se codifica es este identificador único de mensaje al que hace referencia el párrafo anterior.

En este momento es oportuno considerar un análisis sobre el ciclo de vida de una comunicación. En un flujo temporal de acontecimientos, el primer evento será siempre un evento de *sent*. Luego, en algunos casos, podrá suceder un evento de *bounce* o, en un flujo más habitual, un evento de *open* que puede derivar en un evento de clic. Debe considerarse que, una vez ocurrido el primer *open*, se pueden suceder en cualquier orden distintas aperturas y clics).

La registración de un evento de tipo *sent* se toma como el instante inicial de registración de un evento.

A partir de esa registración cualquier otro evento recepcionado que esté relacionado con el sent registrado será nutrido con los datos que se recuperen del evento de sent guardado en el *datastore* Cassandra.

Una vez nutrido el *payload,* el evento recepcionado necesita ser registrado en los distintos *datastores* disponibles.

Cassandra tiene, de acuerdo a lo descrito en los párrafos anteriores, la función principal de guardar eventos para nutrir otras bases, y almacenar información sobre mensajes que van a servir para tomar mediciones y hacer estadística en un procesamiento que será descrito más adelante en esta lectura.

- **MongoDB *Datastore***: MongoDB registra en sus documentos información relativa a eventos y mensajes. El cambio de base en el *datastore* según el canal responde principalmente a que MongoDB se adapta mejor al guardado de datos cuya estructura no tiene una definición clara. Los usuarios de aplicaciones *mobile* tienen características que hacen que los datos cambien constantemente. En primer lugar es muy común instalar la app bajo demanda y desinstalar ni bien termine el tiempo de uso intensivo. La vida media útil de un dispositivo móvil ronda los dos años, mientras que el *email* se mantiene inalterable. Este último punto tiene una consecuencia importante: para el canal *email* la dirección de correo de un usuario funciona como clave única. Esto es posible porque una comunicación mantiene una relación uno a uno con una casilla de correo. Sin embargo, en el caso *mobile*, el diseño se planteó de manera diferente. Una comunicación mantiene una relación de uno a muchos con los dispositivos móviles del usuario y, por lo tanto, la clave única será la superclave (identificador de mensaje, *token* de dispositivo móvil). Se deja abierta la posibilidad de registrar comportamiento y horarios de uso de un usuario

por dispositivo que permiten tener mejor *timing* a la hora de enviar las comunicaciones.

- **MariaDB *Datastore***: un *cluster* MariaDB es otro *storage* que gestiona el equipo encargado del envío de comunicaciones. El *cluster* cuenta con 5 nodos (la cantidad de nodos siempre debe ser impar), uno de ellos es el primario que se encarga de coordinar y sincronizar a los otros nodos que se unen al *cluster*.

  Pueden existir situaciones que deban ser resueltas mediante votaciones de los nodos (*quórum*). Por ejemplo ante alguna discrepancia de datos entre los miembros del *cluster* o ante la necesidad de elegir un nuevo nodo primario ante la caída del actual. Por este motivo es importante tener un número impar de nodos para evitar situaciones de *votaciones divididas* (*split-brain*).

  Los datos en este *cluster*, a diferencia de Casandra, conservan las sumas de las ocurrencias de cada tipo de evento. MariaDB, como vemos en la figura 19, se diseñó con un modelo de datos entidad relación "E-R" que representa cada evento comunicacional en una entidad. Paralelamente el diseño separa las entidades y sus relaciones por canal comunicacional. Los diseños por canal son análogos y tienen una entidad central que establece relación con cada una de las entidades restantes del modelo que en la entidad Campaña[43]. Cada evento tiene un contador (es una columna iniciada en cero para tal fin) que acumula un valor por cada evento recibido. La solución es muy rápida para el caso en que sólo se necesite una suma total de un evento para una campaña específica.

[43] Una campaña es un conjunto de estrategias comerciales que tienen como objetivo dar a conocer el producto o servicio que se ofrece. Para este escenario una campaña se define con un país, un nombre, un *brand* y un grupo de *tags* (marcas) que la identifican unívocamente.

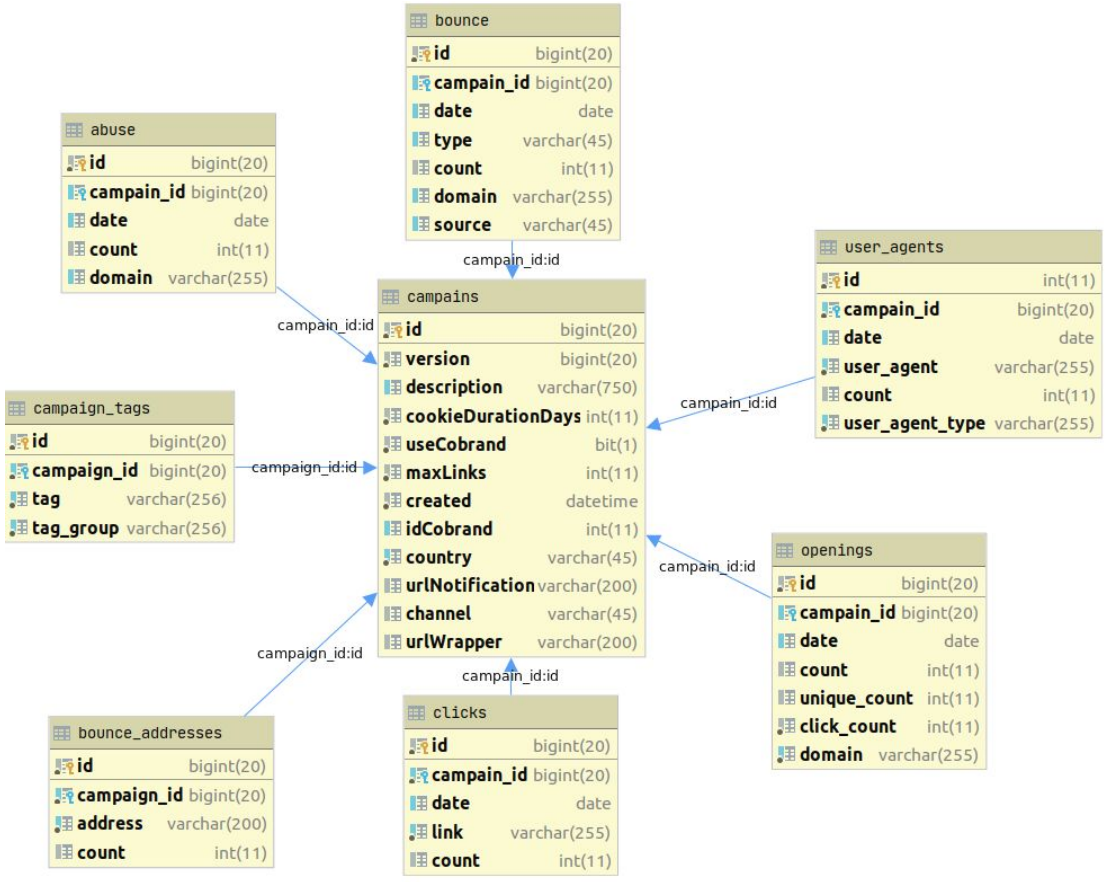

Fig. 19 - Esquema E-R para el canal *email*.

Una vez registrado un evento de *sent* para un identificador los demás eventos asociados a ese identificador son opcionales y pueden llegar en cualquier orden.

La aplicación que registra las comunicaciones, al tener comunicación diferencial con los *datastores* asociados al equipo de Comunicaciones, expone a su vez un servicio de información sobre remitentes. Cada equipo o aplicación que necesite hacer un envío masivo por medio de una campaña puede obtener los datos del remitente deseado asociado al envío.

Para completar su tarea esta aplicación envía información de cada evento recibido a una serie de tópicos[44] expuestos por el equipo de *Data* que es responsable de la siguiente etapa dentro del *pipeline* de trabajo[45].

[44] Refiere al Kafka Topic donde el equipo de Comunicaciones actúa como productor de un tópico o lista de información por canal y el equipo de *Business Intelligence* actúa como suscriptor.

[45] Un *pipeline* o tubería es un conjunto de elementos procesadores de datos conectados en serie, en donde la salida de un elemento es la entrada del siguiente. *Pipeline* es sinónimo de segmentación de trabajo que se ejecuta generalmente en paralelo.

4.3 Procesamiento en memoria

Se ha descrito como esta etapa de trabajo, que incluye el envío de comunicaciones y su registro de eventos en almacenes de datos propios, se encarga también de pasar información procesada a otros equipos. Para el trabajo de manipular la información recolectada a través del registro de eventos comunicacionales, la aplicación ejecuta una serie de procesamientos en memoria que transforma los datos en información útil a otras aplicaciones. Para el procesamiento en memoria se eligieron dos tecnologías que trabajan en conjunto para lograr la mejora buscada en el rendimiento del sistema: Spark y Hazelcast.

4.3.1 Spark

Arriba del *cluster datastore* Cassandra se tiene una capa extra de *software* de Apache Spark. El *cluster* Spark, que vemos en la figura 20, se dedica a procesar en memoria estadística de asuntos, *A/B Tests* y ratio de aperturas.

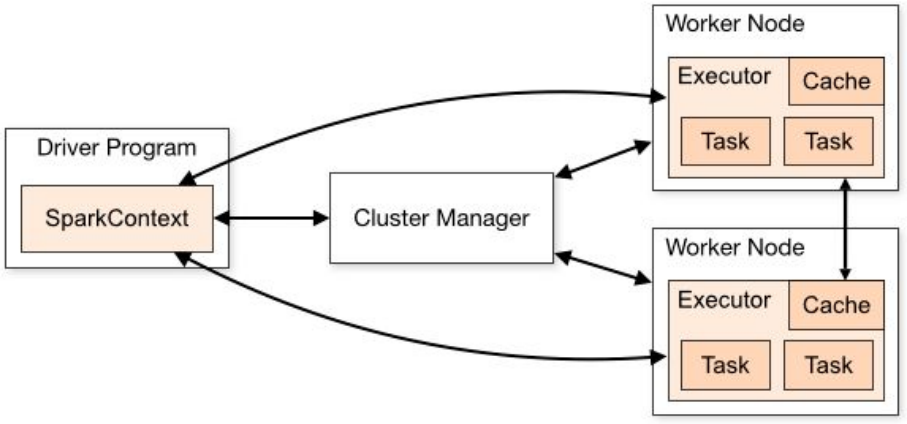

Fig. 20 - Arquitectura de trabajo de Spark

Spark por medio de un objeto llamado *Sparkcontext* se conecta con la administración del *cluster* Java donde reside y se encarga de que se creen ejecutores o executors encargados de la computación en los nodos del *cluster*. Los ejecutores son los procesos de los nodos trabajadores encargados de ejecutar *tasks* o tareas individuales en un trabajo Spark determinado. Se inician al comienzo de una aplicación Spark y normalmente se ejecutan durante toda la vida útil de una aplicación. Cada tarea Spark crea una estructura de datos RDD[46] leyendo datos de la base, una vez creado el RDD inicial se realizan transformaciones

[46] *Resilient Distributed Datasets* representan una colección inmutable y particionada de elementos sobre los que se puede operar de forma paralela. Los RDD aceptan transformaciones (*map*()) donde cada elemento pasa por una determinada función

para crear más objetos RDD a partir del primero. Las transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean uno nuevo. Luego de que Spark realice las acciones y transformaciones necesarias sobre los datos, los objetos RDD deben converger para crear el RDD final. Este RDD, que hace estadística con los datos obtenidos del envío de campañas, nutre distintas aplicaciones. En el caso particular de datos provenientes del ratio de apertura, el que se nutre del trabajo de procesamiento realizado por Spark es Hazelcast (Fig. 21)

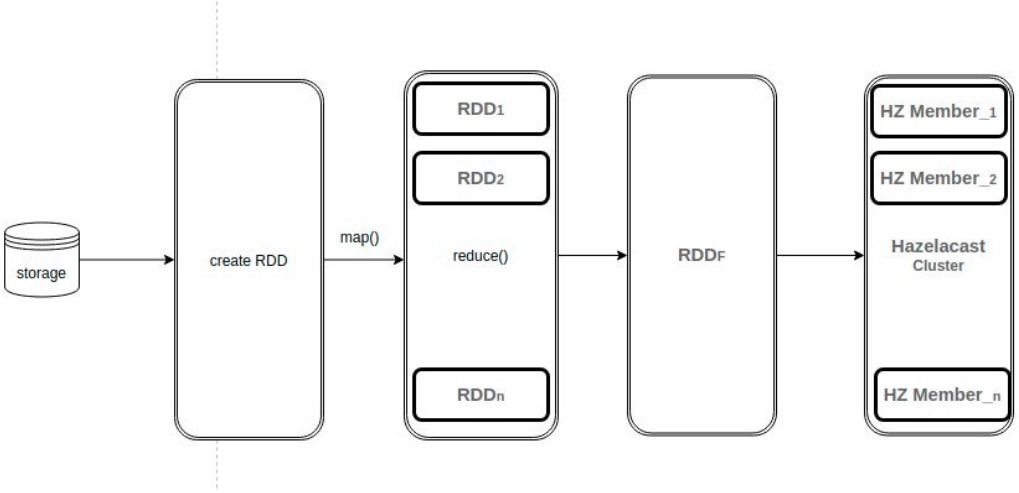

Fig. 21 - Integración Apache Spark, Hazelcast.

4.3.2 Hazelcast

El *datastore* Cassandra tiene la responsabilidad principal de registrar eventos y cómo ya se mencionó provee de la información de remitentes de campañas a los clientes que necesitan despachar comunicaciones. Tener acceso a los datos posibilitó brindar información necesaria para la coordinación, el coordinador necesita información del *rate* de aperturas de comunicaciones por campaña, *brand* y país para un usuario específico. Esto es así porque la función de validación para un usuario está determinada por la entrada de estos datos.

El uso de Hazelcast tuvo como objetivo descargar de trabajo al *backend* de la aplicación que centraliza la registración de eventos. Como se puede ver en la figura 22, Hazelcast está embebido en la aplicación Java y responde prácticamente sin demora a cada *request recepcionado* que solicita servicio de coordinación. Una de las principales características de Hazelcast es que no cuenta con un nodo maestro. Cada nodo está configurado de la misma manera en términos de funcionalidad. Un cliente puede conectarse directamente a cualquier nodo del *cluster* y, por lo tanto, ayuda a una disponibilidad más rápida de los datos. Otra

y devuelve un nuevo RDD con el resultado y acciones(*reduce*()) que devuelven valores al *cluster* después de llevar a cabo una computación sobre el conjunto de datos.

característica importante de Hazelcast es que los datos en la caché se copian varias veces en los nodos, lo que ayuda a manejar la condición de conmutación por error. Si algún nodo falla, otro nodo proporciona datos al cliente hasta que se recupere el nodo fallido, por lo que proporciona alta disponibilidad.

La tarea de Hazelcast es exponer en una caché los datos procesados por Spark, eventos de open agrupados por tipo de campaña (clave de la lógica de coordinación) y usuario.

Para su objetivo mantiene un mapa distribuido donde por cada clave <*user_id*; *coordination_token*> se tiene un ratio de apertura. El ratio se hace procesando los eventos de *sent* y de *open* agrupados por tipo de campaña. Tanto el procesamiento como el mapa con la información se gestionan en la memoria distribuida de Hazelcast y los tiempos de respuesta son inmediatos.

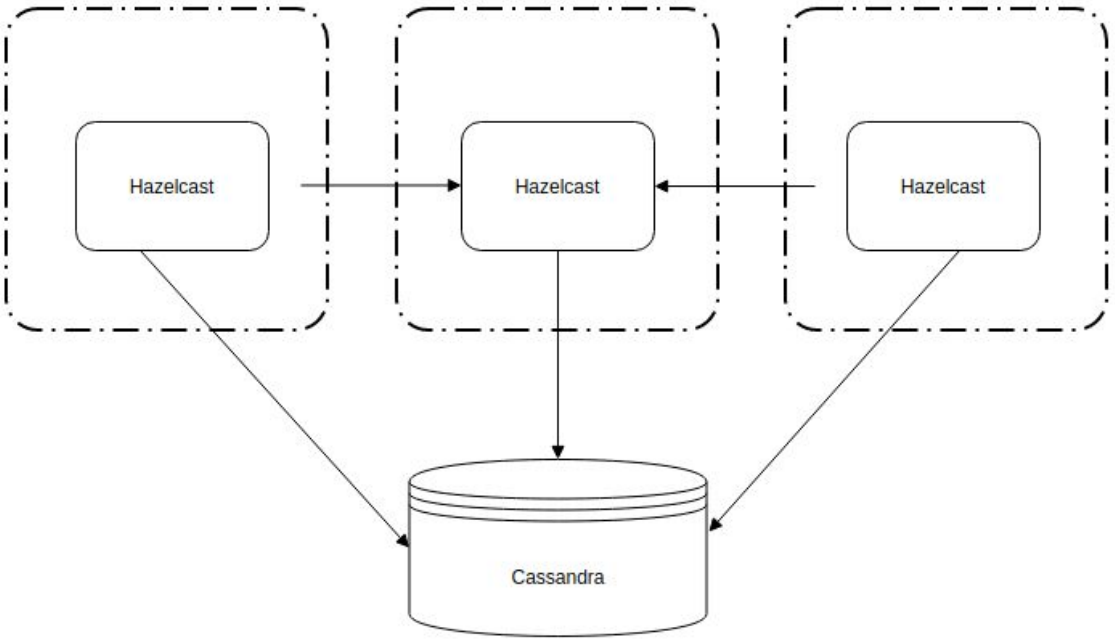

Fig. 22 - Hazelcast embebido en la aplicación Java en comunicación con Cassandra *datastore*

Debido a que la aplicación de *tracking* está construida en Spring Boot[47] agregar Hazelcast sólo requirió sumar dos dependencias al archivo de configuración pom.xml una para soporte de Spring y otro para incluir el IMDG.

Después de eso, la solución necesita configurar la instancia de Hazelcast para esto se agregó un *bean* a la clase *@SpringBootApplication*, esta clase hace un *scan* y va creando

[47] Spring Boot es una herramienta que simplifica el proceso de declaración de dependencias y de *deploy* dentro del proceso de desarrollo de aplicaciones basadas en el *framework* Spring Core

clases de configuración, que están agrupadas lógicamente y las mete al Spring *Container* que gestiona su ciclo de vida.

4.4 Segmentación

El equipo que se encarga de las comunicaciones tiene a su vez la tarea de exponer servicios que proveen segmentación de usuarios. La segmentación fue pensada cuando se trazaron las primeras definiciones sobre la omnicanalidad. La segmentación es una herramienta que permite clasificar a los usuarios de la aplicación (para este caso usuarios suscriptores a la recepción de comunicaciones) en grupos pequeños acorde a características como edad, género, ubicación, etc. con el objetivo de crear campañas o estrategias dirigidas. Tener bases segmentadas por distintos criterios permite a la compañía lanzar campañas más específicas.

Un cliente, habitualmente un integrante del equipo de *marketing online*, que desea enviar una campaña, tiene la posibilidad, por medio del segmentador, de crear un segmento de usuarios destinatarios de una comunicación partiendo de un comportamiento común de los mismos. La aplicación que se encarga de la creación de segmentos de usuarios, ofrece una interfaz de usuario desarrollada en el *framework* Angular[48] (Fig.23) a través de un punto de menú de un *back office*. Inicialmente la aplicación permite crear una serie de segmentos predeterminados como por ejemplo “buscador de un producto”, “comprador de un producto”, “buscador de un destino”, etc. Al mismo tiempo la herramienta de *back office* permite ajustar los segmentos predeterminados a partir de una serie extra de filtros disponibles.

[48] AngularJS (comúnmente llamado Angular.js o AngularJS 1), es un *framework* de Javascript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página.

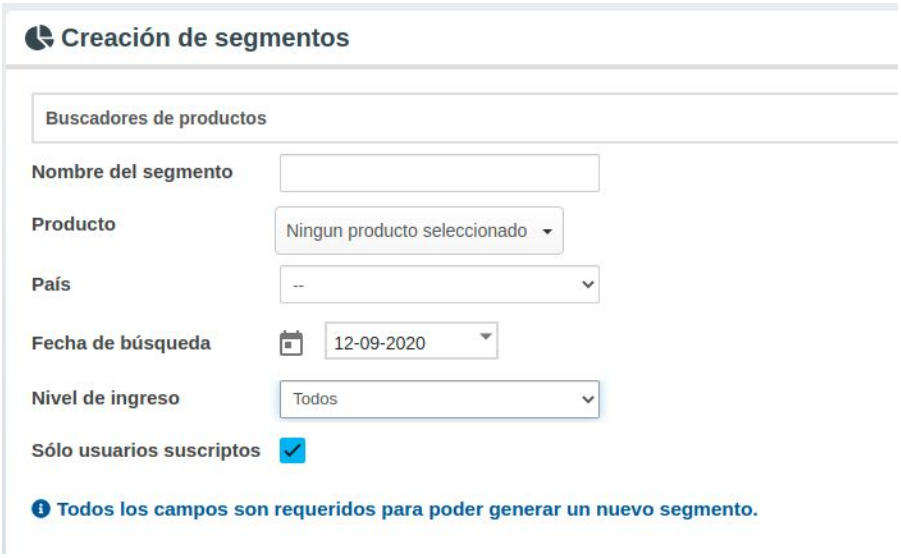

Fig. 23 - Pantalla inicial de creación de un segmento de usuarios.

Es continuo el trabajo de análisis sobre los criterios que se aplican para la segmentación en el *marketing* comunicacional. Los resultados muestran que una comunicación certera y ajustada a la preferencia del usuario es percibida cómo una preocupación por sus intereses y siempre es compensada desde el punto de vista del negocio.

4.5 Capa presentación

Finalmente, el equipo encargado de despachar las comunicaciones y registrar los eventos relacionados al envío, expone una serie de tableros que ayudan a monitorear esta etapa del proceso de trabajo que describe esta tesina y que tiene por objetivo la exposición de información en un tablero omnicanal.

Diferentes herramientas permiten tomar métricas propias de esta etapa del pipeline de trabajo y que sirven para chequear la salud general del proceso. Los datos expuestos en estos tableros son útiles también para verificar la consistencia entre los datos propios de esta etapa y los esperados en las etapas posteriores de trabajo.

El conteo de eventos y el conteo de ventas se expone en un tablero de Kibana que expone lo registrado en Elasticsearch.

Kibana es muy versátil en la construcción de gráficas con lo que se ofrecen cantidad de tableros algunos son:

- eventos (Fig. 24)
- *bounces*
- instalaciones/desinstalaciones *mobile*

- instalaciones/desinstalaciones *browser*
- comportamiento de las encuestas enviadas (Fig. 25)
- agentes de usuario utilizados
- *click map* sobre *emails*
- rendimiento de diferentes *call to action*
- comportamiento sobre *geofences*

Fig. 24 - Vista parcial del tablero de eventos enviados. Acumulado diario y estadística de ratios.

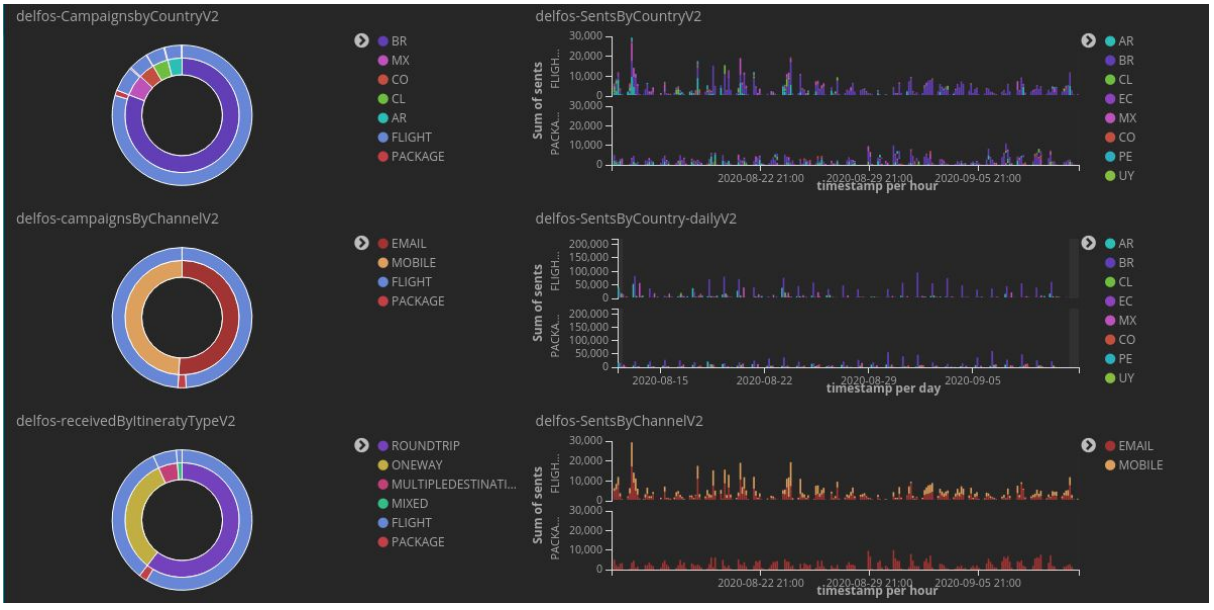

Fig. 25 - Distintos cortes para la información de envíos de un tipo de comunicación.

Ya se ha mencionado en esta tesina que Elasticsearch es uno de los *storages* utilizados para almacenar eventos comunicacionales. A diferencia de los *storages* Cassandra o MongoDB los eventos se guardan agrupados por tipo de evento. Esta característica de tener un contador por tipo de evento hace que Kibana tenga una respuesta más rápida ante una consulta que los otros *storages* y resulta muy útil para la visualización online de resultados de un envío de una campaña.

Si se desea obtener información más desagregada, a nivel de evento, por ejemplo explorar solo eventos con una determinada marca, se cuenta con un reporte que se opera desde un *back office* desarrollado en Java + Angular. (Fig. 26)

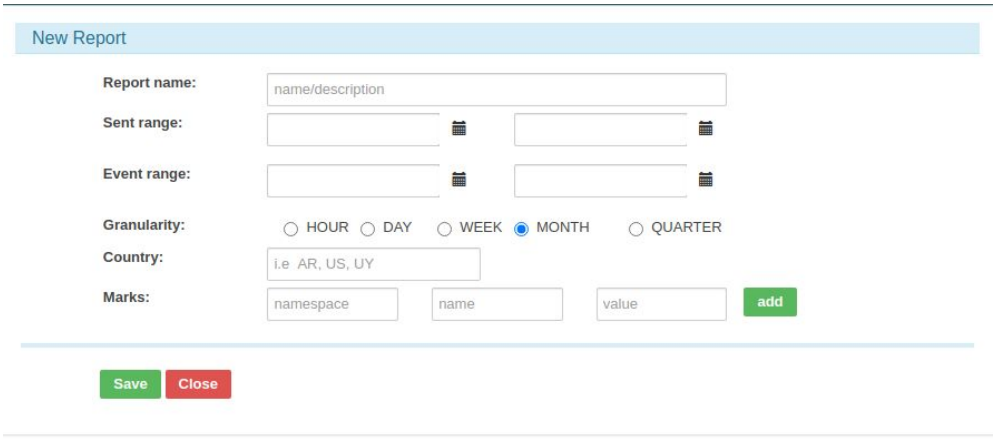

Fig. 26 - Vista inicial para la creación de un reporte individualizado por eventos.

Otro reporte que se ofrece a través del *back office* es sobre los experimentos que se llevan adelante sobre asuntos en *emails* (Fig. 27). El *back office* permite la creación de experimentos en *A/B Testing* sobre asuntos de *emails*, se pueden crear tantas ramas del *A/B Test* como se deseen sobre una misma comunicación. Una vez establecido el *A/B Test* se puede activar su uso en producción e inmediatamente se comienzan a ver las métricas de *open rate* cada una de las ramas constituyentes del *A/B Test*.

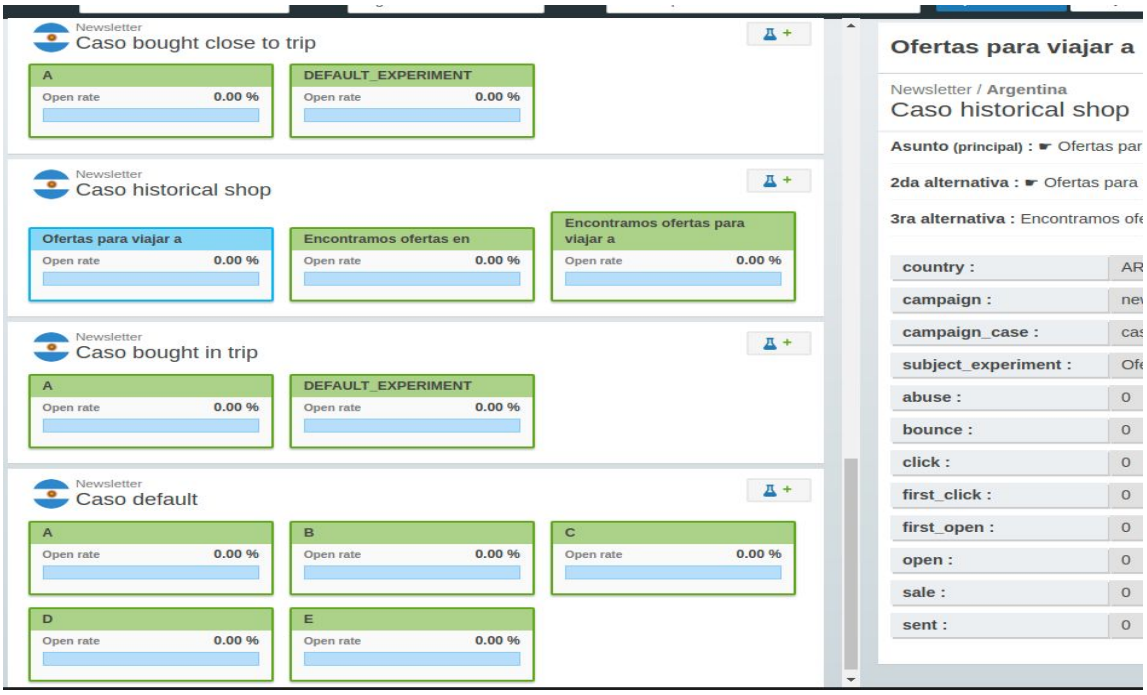

Fig. 27 - Pantalla de creación de experimentos sobre asuntos de comunicaciones, edición y visualización de resultados

Capítulo 5

Manipulación de la información registrada.

Esta etapa del pipeline de trabajo se ocupa de monitorear, medir y registrar los ingresos de las fuentes de tráfico y de la atribución de las transacciones.

Respecto de los ingresos de las fuentes de tráfico realiza los cálculos necesarios para determinar cuántos ingresos hay en el sitio y cuantos usuarios entran por esos ingresos. Se tiene un control sobre la cantidad de personas que acceden al *site* por ingreso directo, por Google (SEO/SEM[49]), por Instagram o por una comunicación propia (gestionada por el equipo encargado de las comunicaciones cuyas tareas fueron descritas en el apartado anterior).

Por atribución se entiende al uso de la información procesada de las fuentes de tráfico que permite atribuir un valor de peso (importancia) al ingreso que se considera responsable de

[49] Estrategias de *marketing online* para mejorar la visibilidad o posicionamiento de un determinado sitio *web*.SEM se refiere a la promoción de un sitio *web* en los buscadores mediante el uso de anuncios de pago a través de plataformas como Google Ads o Bing Ads mientras que SEO se refiere al trabajo de optimización y de aumento de la popularidad de un sitio web, con el objetivo de que dicho sitio sea rastreable por los motores de búsqueda, indexado correctamente y suficientemente relevante para que algunas o muchas de las páginas sean mostradas en las primeras posiciones de los buscadores para determinadas consultas de búsqueda de los usuarios.

una acción de compra llevada a cabo por un usuario. Este sistema de pesos, que se relaciona con un porcentaje del dinero total de la venta asignado a un ingreso, determina el flujo de dinero invertido por una compañía en función de sus intereses.

El equipo que se encarga del proceso de extracción, transformación y carga de datos (equipo de *Data*) se ocupa de recopilar las interacciones que hace un usuario dentro de la compañía. Una vez que el usuario accede al *site* se monitorea su actividad, por ejemplo búsquedas realizadas, ingresos a un *checkout*, ingresos a su perfil, visualización de notificaciones guardadas, etc. Toda esa información alimentará a la aplicación de ingesta de datos desarrollada por este equipo de trabajo.

Procesar esta información tiene el objetivo de tomar valores de las métricas de conversión establecidas para participar del *funnel* interno al *site*. Dentro del *site*, un usuario puede estar visualizando algún destino, conducido por medio de una búsqueda realizada desde la *home page* o bien por medio de un enlace externo que lo condujo directamente. El usuario puede seguir avanzando hacia el detalle (*detail*) de un producto buscado, incluso llegar hasta el paso previo a la compra (el *checkout*, donde se ingresan los medios de pago) y eventualmente puede concluir y efectuar compra (*thanks*).

El equipo de *Data*, también presta servicios de *data streaming.* Este es un servicio de mensajería *online* que permite a un equipo de trabajo, enviar datos que pueden ser consumidos por diferentes equipos en tiempo real o de manera sincrónica.

5.1 - Ingesta y validación de datos

Ya se ha mencionado en esta tesina que la arquitectura de trabajo de las aplicaciones que le dieron nuevas funcionalidades al *stack* y que permiten un avance hacia el proceso de omni canalización se basó en microservicios. Los microservicios han cambiado el panorama del desarrollo en los últimos años. Las arquitecturas basadas en microservicios reducen las dependencias. Por ejemplo, cada aplicación cuenta con su propio *storage* y baja el porcentaje de bases de datos compartidas, en ese sentido desacoplar implica procesos de desarrollos más ágiles. [46]

No obstante, las aplicaciones distribuidas necesitan algún tipo de integración para compartir datos. Una opción de integración muy popular es conocida como el método síncrono que se basa en el uso de interfaces de programación de aplicaciones (APIs) para compartir datos entre los diferentes usuarios.[47]

El equipo de *Data*, que interviene luego del equipo que envió las comunicaciones, recibe datos en un *storage* masivo y los procesa agrupándolos en nuevas tablas con mayor

significancia en el negocio de la OTA (*Online Travel Agency*). Los datos así procesados podrán ser consumidos por el equipo de Business Intelligence.

El ecosistema que describimos y que se encarga del *storage* masivo de datos heterogéneos cuenta con varias etapas de trabajo. En la primera estación de trabajo se expone una API de integración sincrónica que ingesta todo tipo de eventos en la infraestructura de datos. La API se respalda con una aplicación desarrollada con Spring Boot y JavaEE es parte de un subsistema de microservicios que interactúan para hacer avanzar por el *pipeline* a los datos que se ingresan.

Los datos que recibe del equipo propietario de las aplicaciones que despachan las comunicaciones, son el elemento clave para llevar adelante la historia de *Roadmap* que guía el desarrollo del tablero omnicanal. Estos datos primarios son importantes ya que posibilitan el envío de tráfico de compra hacia etapas más avanzadas del *funnel* y que redundan en una mejora global de los KPIs de la compañía.

Los datos recepcionados son fuentes de ingreso de un anuncio, un *email*, un *push notifications* o un *web push notifications* y lo que específicamente se registra son los clics del usuario. Para la toma de estas mediciones cada evento de clic que dirige originalmente hacia un lugar específico a través de un *link* URL sufre una serie de redirecciones antes de dirigir al usuario al destino pensado para esa comunicación.

La primera redirección, de las mencionadas en el párrafo anterior, conduce la información asociada al evento hacia una aplicación gestionada por el equipo encargado del despacho de las comunicaciones. La aplicación del equipo encargado de las comunicaciones, luego de registrar datos que le son de utilidad, dirige el flujo hacia el equipo encargado de la manipulación de los datos.

La información de un *open*, de un clic o de un *tap* recepcionada por la aplicación de ingesta de datos trae consigo información sobre el destino del usuario, el identificador de la campaña que gestiona la comunicación, el identificador único de la comunicación, el agente de usuario que refiere el flujo, entre otros datos.

Esta aplicación, ni bien recepciona el evento, interactúa con otra aplicación propiedad del mismo equipo cuyo objetivo es validar que el ingreso de datos tenga la estructura esperada. Esta segunda aplicación que provee servicio de validación de estructuras recepcionadas utiliza una solución basada en Apache Avro que expone un catálogo de eventos. La aplicación de catálogos gestiona los esquemas de registración de manera que los eventos ingestados sean solo los adecuados.

La solución arquitectural Apacke Kafka / Avro es de uso común en las arquitecturas orientadas a eventos donde el volumen de información que se produce y se consume no

para de crecer. En este escenario es importante centralizar (por medio del uso de esquemas de datos) y ofrecer más control sobre el dato (*Data Governance*[50]) [48].

El uso de un esquema de datos permite hacer responsable al propietario del dato de definir la compatibilidad (*backward* o *forward*) del esquema, aliviando de tareas de validación a los consumidores y productores que usen dicho dato.

Por ejemplo, cuando se usa JSON para compartir la información, la compatibilidad de ese dato es "*backward*" por defecto. Porque cuando se introduce un nuevo campo, los consumidores antiguos lo ignoran. Pero este último escenario descrito no es del todo correcto, y genera demasiada diversidad de consumos sobre un mismo dato y se pierde el control del mismo. Esta es una posible razón para empezar a usar esquemas y su proceso de validación.

Los esquemas se usan cuando se leen y se escriben datos. Los esquemas deben ser compartidos entre los servicios que envían el dato y los servicios que lo consumen. Ya sea para el uso de esquemas JSON o Avro se necesita un recurso global donde controlar los datos.

Como ya se ha visto en este trabajo Kafka es un *software* para el transporte de datos entre servicios y aplicaciones que en este equipo se usa como una plataforma de *streaming* de Datos. La plataforma de *streaming* se apoya en los productos:

- **Kafka Streams**: Para transformar un *stream* de datos y publicarlos otra vez en un tópico de Kafka
- **Kafka Connect**: Kafka Connect se utiliza para recibir datos desde una fuente externa (Elasticsearch, Mongo, IBM MQ, PostgreSQL, etc) y para enviar datos a una fuente externa.
- **Schema Registry:** Para registro de esquemas que serán validados mediante Avro.

Las herramientas descritas se implementan teniendo un tópico Kafka por equipo productor, *n-esquemas* Avro por tópico (todos los esquemas donde el productor volcará sus datos agrupados por criterios funcionales), y una estructura columnar (Archivo Parquet sobre Presto DB) que persiste los datos (Fig. 28).

[50] El gobierno de datos o *data governance* se refiere a la capacidad que tiene una organización de cuidar por la calidad de los datos, garantizar la protección y gestionar el ciclo de vida de la información.

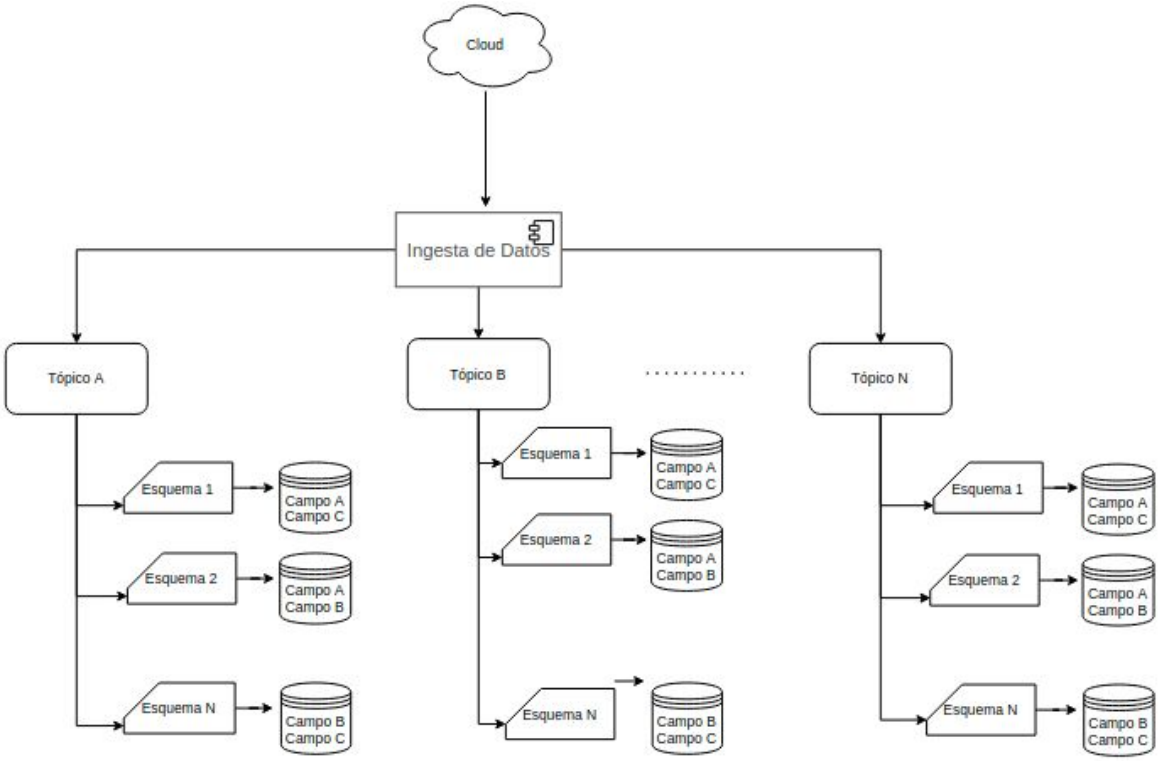

Fig. 28 - Estructura de tópicos/ esquemas y columnas.

En resumen, como se puede ver en la figura 29, cada vez que un evento se publica en la API de ingesta, el evento recibido se valida contra el catálogo para determinar si está correcto, si tiene algún error o si es un evento desconocido. Una vez que esa validación está concluida, se inicia una nueva etapa del *pipeline* enfocada en la persistencia de los datos.

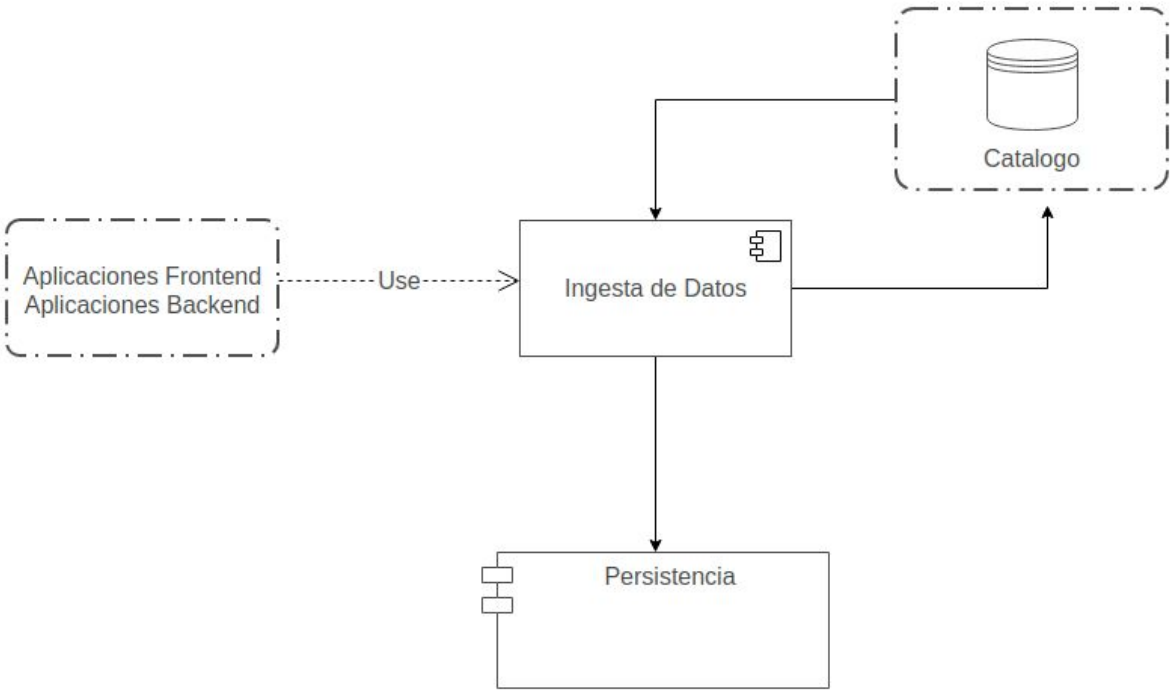

Fig. 29 - Arquitectura de la primer etapa de trabajo

5.2 - Persistencia de datos

Una vez que la información fue recepcionada y validada, la aplicación encargada de la ingesta publica en un recurso que se va a encargar de persistir la información en diferentes *storages* (Fig. 30).

Este componente, responsable de la gestión de la persistencia, genera dos espacios de almacenamiento con distintos criterios y funcionalidades. Por un lado, se genera una estructura donde se almacena información en tiempo real que se conserva por 30 días a partir de su recepción. Esta información se guarda y consume en una solución Elastic Stack y por lo tanto se permite el seguimiento del flujo de datos que va hacia el equipo de *Business Intelligence BI* para determinar la salud del proceso. Elastic Stack utiliza Kibana para ofrecer distintos tableros personalizables para la visualización del flujo mencionado.

Por otro lado, para visualizar información con más tiempo de guardado desde su recepción se cuenta con dos herramientas: Dbeaver y Apache Zeppelin que se conectan al procesador de datos Presto.

Presto interpreta las consultas del binomio Zeppelin/Dbeaver y obtiene resultados desde el *data lake*.

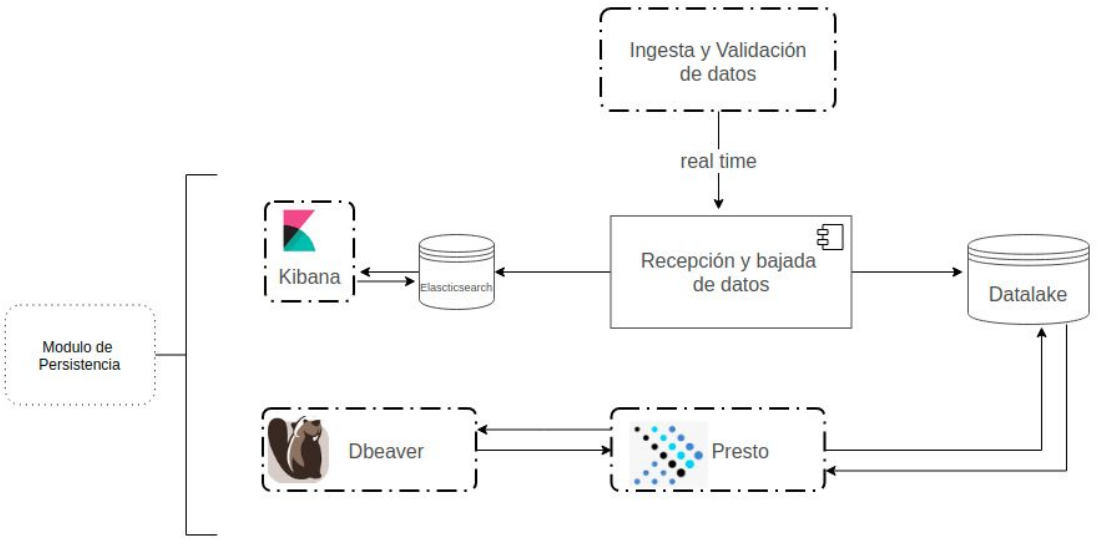

Fig. 30 - Mapa de la gestión de persistencia de datos.

5.3 - Recopilación de eventos relacionados

Hasta aquí se describe el proceso de ingesta y persistencia de eventos comunicacionales. Pero para registrar una venta y poder atribuirle participación en el acto de venta al evento comunicacional es necesario registrar más datos.

Cada evento comunicacional recepcionado almacena una *cookie*[51] en el dispositivo donde se produjo el evento. de manera que si en algún momento se realiza una compra (en este caso en el *site*), se pueda contar con un "*tracking pixel*[52]".

La página de *thanks*[53], al ser visitada por el usuario comprador, registra cierta información que junto con la cookie "*tracking pixel*" permiten obtener la información necesaria para rastrear los eventos relacionados con la venta.

La información recolectada del *tracking pixel* y de la visita a la página de *thanks* es registrada como un evento de *thanks* por medio de la aplicación de ingesta de datos que describimos más arriba en esta tesina.

[51] Las *cookies* son archivos de texto de tamaño pequeño, con etiquetas de identificación que se almacenan en el directorio del navegador de su computadora o en las subcarpetas de datos del programa. Las *cookies* se crean cuando un usuario usa un navegador compatible que realiza un seguimiento de los movimientos de ese usuario dentro del sitio con el objetivo de ayudarlo a seguir acciones desde donde las dejó, recordar su inicio de sesión registrado, selección de tema, preferencias y otras funciones de personalización. [49]

[52] Un *tracking pixel* es un objeto incrustado en una página web o correo electrónico, generalmente invisible para el usuario, que permite comprobar que un usuario ha visitado la página o leído el correo electrónico. Los *web bug* se suelen implementar mediante una imagen incrustada transparente de 1×1 pixel, denominada *tracking pixel* (*pixel de seguimiento* en inglés).

[53] Una *thanks page* o página de gracias es el lugar donde al usuario se le entrega la oferta prometida en la *landing page*. En esta página finaliza el proceso de conversión.

Capítulo 6

Construcción del enfoque multicanal en la inteligencia de negocio

El equipo *Business Intelligence* se encarga de hacer un nuevo proceso de ETL sobre los datos que se recolectaron y procesaron con la aplicación que ingesta datos.
Una vez hecho este nuevo procesamiento, el equipo *Business Intelligence* construye distintas vistas sobre esta información que terminará generando nuevo conocimiento de negocio. Este conocimiento surge del uso que gente idónea hace de los tableros para contribuir en una mejora del proceso de toma de decisiones.

6.1 - Extracción, Transformación y Carga de datos

El equipo de *Business Intelligence* nutre su procesamiento ETL con los datos expuestos por los equipos de Comunicaciones y de *Data*.
Directamente del equipo de Comunicaciones toma datos de las campañas desde el modelo Entidad-Relación contenido en el *datastore* MariaDB. Los datos restantes que le son de utilidad, cómo datos de atribución, de ingresos de usuario al *site* por diferentes fuentes de tráfico, etc, los obtiene de la estructura de archivos Parquet expuesta por el equipo de *Data*.
El equipo de Business Intelligence también consume por el canal de *streaming* información de los ingresos a páginas de "*thanks*" para obtener detalle de los productos que convierten y para mantener un catálogo propio de productos vendidos.
BI trabaja su procesamiento ETL, descrito en la figura 31, con aplicaciones Spark. Cada día corre una aplicación que ejecuta consultas sobre los esquemas asociados a los tópicos de *email*, *push notifications* y *web push notifications* y les hace un *join* con la tabla de eventos atribuidos propiedad del equipo de *Data*.
A su vez consume información en tiempo real con la API Spark *Streaming*. Esta tecnología toma un flujo de datos continuo y lo convierte en un flujo discreto denominado DStream[54], para que el *core* de Spark lo pueda procesar. Spark *Streaming* consume datos de las transacciones (ingresos de usuarios a las páginas de "*thanks*") desde el Kafka expuesto por el equipo de *Data*.

[54] DStream es una abstracción que representa a una secuencia de RDDs ordenados en el tiempo que cada uno de ellos guarda datos de un intervalo concreto. Con esta abstracción se consigue que el core Spark haga su trabajo sin enterarse de que está procesando un flujo de datos, ya que el trabajo de crear y coordinar los RDDs es realizado por Spark Streaming.

Toda esta información es relacionada y guardada en un *storage* de su propiedad.

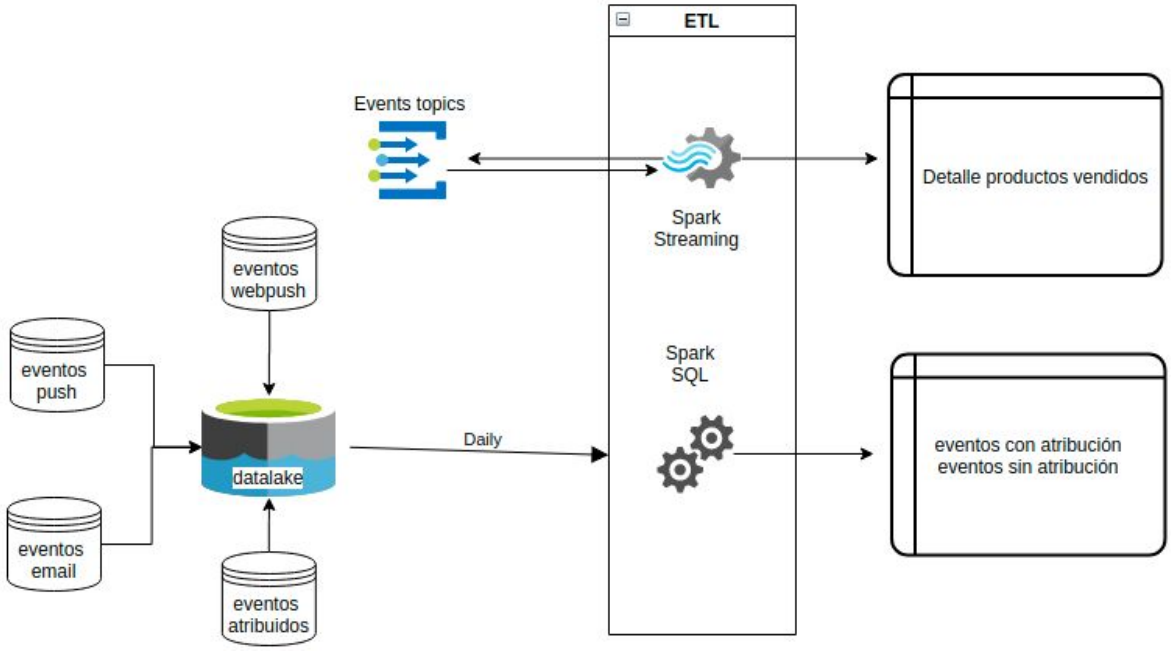

Fig. 31 - Descripción proceso ETL

A partir de este procesamiento inicial ahora el equipo de *Business Intelligence* dispone de un almacenamiento propio con la totalidad de eventos comunicacionales recepcionados (Fig. 32). Si esos eventos estuvieran asignados a alguna atribución entonces también cuenta con los datos de esa transacción incluyendo el detalle de los productos asociados a ella.

Este cúmulo de información recibe un nuevo procesamiento para resumir eventos por tipo (*sent*, *bounce*, *open*, *click*, *sale*), campaña y por categoría comunicacional.

Es importante recordar que muchos de los eventos registrados por Comunicaciones, son expuestos al equipo de *Business Intelligence* por la gestión y trabajo del equipo de *Data*. Estos eventos tienen información de la campaña, la fecha de sucedido, el producto asociado a cada *link* de la comunicación, el país, y una colección dinámica de datos que por ejemplo permiten el *A/B Testing*.

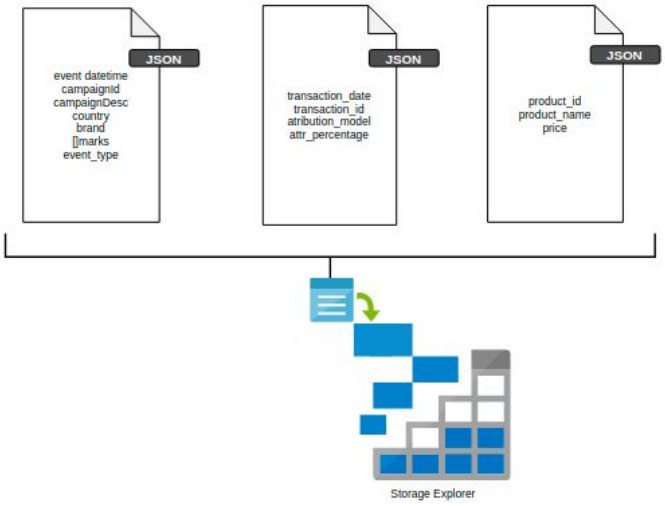

Fig. 32 - Esquema de eventos y volcado al *storage* propio de BI

El *storage* de exploración es un *datamart*[55] cuyo propietario es el equipo de *Business Intelligence*. Es el último almacenamiento antes de trabajar la capa de presentación y se construye sobre la base del procesamiento Spark. En él, los datos pueden ser agrupados, explorados y propagados de múltiples formas para que diversos grupos de usuarios realicen la explotación de los mismos de la forma más conveniente según sus necesidades.

Los grupos de usuarios son parte del equipo de analistas de BI que se encargan de convertir la información dentro del *datamart* en conocimiento que pueda ser aplicado por los usuarios finales del tablero. Hay analistas encargados de la parte de eventos comunicacionales y sus KPIs que generan el tablero omnicanal pero el *datamart* brinda información útil a analistas de tableros con información más agregada y útiles a sectores más gerenciales de la compañía.

6.2 - Carga de datos en memoria

La presentación de la información se hace con QlikView[56]. En esta etapa del procesamiento, los datos contenidos en el *datamart* son de significancia para la generación del tablero omnicanal. Esta significación permite realizar un análisis de la interacción de usuarios con el *site* a través de los eventos comunicacionales y relaciona esto con las transacciones generadas. La etapa de carga en memoria de esta información tiene como objetivo dejar disponible la información procesada y recolectada en las bases propias del equipo de

[55] Los *data marts* son pequeños *data warehouse* centrados en un tema o un área de negocio específico dentro de una organización. En nuestro contexto es un subconjunto del *data lake* específico y procesado.
[56] QlikView es un *software* de *Business Intelligence* de visualización y análisis de datos.

Business Intelligence, para que sea usada por los distintos tableros construidos y que colaboren con la toma de decisiones a nivel empresa.

QlikView es una herramienta SSBI de visualización y análisis de datos. *Self Service Business Intelligence* (SSBI) es un enfoque para el análisis de datos, que permite que usuarios no técnicos puedan elaborar vistas sobre la información existente. El uso de QlikView libera así el ancho de banda de los equipos de desarrollo BI y deja la responsabilidad de crear los tableros de uso gerencial a recursos con preparación más cercana al dominio funcional donde se utiliza el tablero.

Los datos del *datamart* de BI son ingresados a QlikView por un script SQL Spark[57]. Una vez ingresados a QlikView este los comprime y los mantiene almacenados en la memoria RAM. El almacenamiento en memoria permite que los cálculos se hagan con alta velocidad.

Varios usuarios pueden acceder inmediatamente a los datos almacenados en la memoria para su exploración. Y para el caso de conjuntos de datos, que son demasiado grandes para caber en la memoria, QlikView puede conectarse directamente a la fuente de datos.

6.3 - Exploración de datos en memoria

Con la información en memoria el equipo de analistas de BI comienzan el proceso de exploración de datos.

La exploración de datos es una de las etapas con más interacción humana del proceso. Este equipo guía el trabajo de desarrollo, ya que evalúa si los datos que llegan a esta etapa son de utilidad o necesitan algún cambio.

Los indicadores que se registran son evaluados y agrupados con diferentes criterios en busca de nuevas interpretaciones que desafían una interpretación inicial (Paradoja de Simpson[58]) [50].

Como podemos ver en la Fig. 33, QlikView está construido sobre una arquitectura cliente-servidor. El usuario que crea y manipula los datos para la construcción del tablero trabaja en el *back-end* generando con su tarea unos archivos con extensión ".qvw" que serán pasados por la componente QlikView Publisher al *front-end* de la aplicación.

QlikView Publisher es el componente de *back-end* que se utiliza como servicio de distribución para distribuir los documentos .qvw a varios servidores y usuarios de la arquitectura de *front-end* QlikView. QlikView Publisher también realiza la carga de datos

[57] Spark SQL es una utilería que brinda soporte nativo para SQL a Spark. Spark SQL procesa los datos almacenados tanto en RDD (conjuntos de datos distribuidos de Spark) como en fuentes externas haciendo transparente el trabajo con los RDD y las tablas relacionales.

[58] La paradoja de Simpson se refiere al cambio de sentido de una comparación o de una asociación cuando datos de distintos grupos se combinan en un solo grupo.

desde la fuente *datamart* de BI y es responsable de mantener el acceso y los privilegios de los usuarios.

El equipo de analistas de BI utiliza QlikView Desktop[59] para la creación de los tableros (aplicación GUI[60]). Esta herramienta provee, para la construcción de un tablero, de asistentes de *software* para la exploración de datos y herramientas *drag and drop*.

Los diferentes archivos procesados con el IDE[61] QlikView Desktop se almacenan nuevamente con la extensión .qvw. pero en el *front-end* de la aplicación por medio del componente de *front-end* QlikView Server que recibe la información procesada a través del componente propio del *stack* Qlikview el "*publisher*".

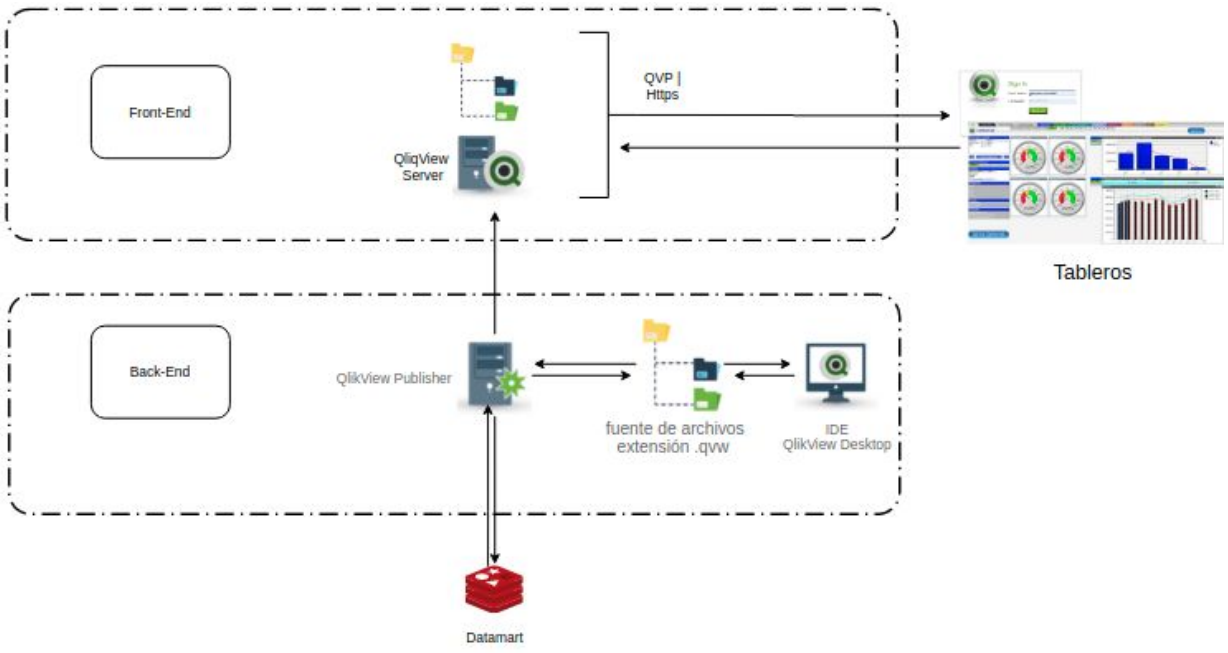

Fig 33 - Detalle arquitectura QlikView.

QlikView Server expone los archivos con extensión .qvw para que un cliente http pueda consumirlos desde Https o a través del protocolo QVP[62].

Finalmente el *dashboard* o tablero de control es una herramienta que se utiliza para obtener una visión general de todos los procesos comunicacionales de la compañía y sus resultados. La visión del *dashboard* es fácil de interpretar y por lo tanto ayuda a la toma de decisiones.

---

[59] Es una herramienta de escritorio basada en Windows y embebida en el *backend* QlikView.

[60] GUI (*Graphical User Interface*) se refiere a una aplicación que proporciona un entorno visual sencillo para permitir la comunicación con el *core* de un programa.

[61] *Integrated Development Environment es una aplicación que proporciona servicios integrales a las tareas de desarrollo normalmente un IDE consiste de un editor de código fuente, herramientas de construcción automáticas, un depurador, un compilador y un intérprete.*

[62] QVP es un protocolo de comunicación propiedad de QlikView que viene encriptado por defecto con RSA 1024 bits.

La información más importante para la consecución de los objetivos de negocio está disponible a través de gráficos. Mapas y KPIs, lo que facilita su visualización e interpretación.

El trabajo de construcción del tablero se planteó tres objetivos principales:

- presentar la información gráfica, resumida y concisa (gráficos y KPIs). La idea no es abrumar sino solo los KPIs necesarios.
- mostrar principalmente información relevante para la compañía. Tiene que ayudar a conocer cómo está el negocio en una primera observación.
- mostrar análisis sobre lo ocurrido, recomendaciones y su potencial impacto sobre el negocio.

El *dashboard,* con sus diferentes vistas (Fig.34), debe ayudar a hacer predicciones y medir las tendencias que permiten al equipo de *marketing* tomar decisiones asertivas en el momento adecuado.

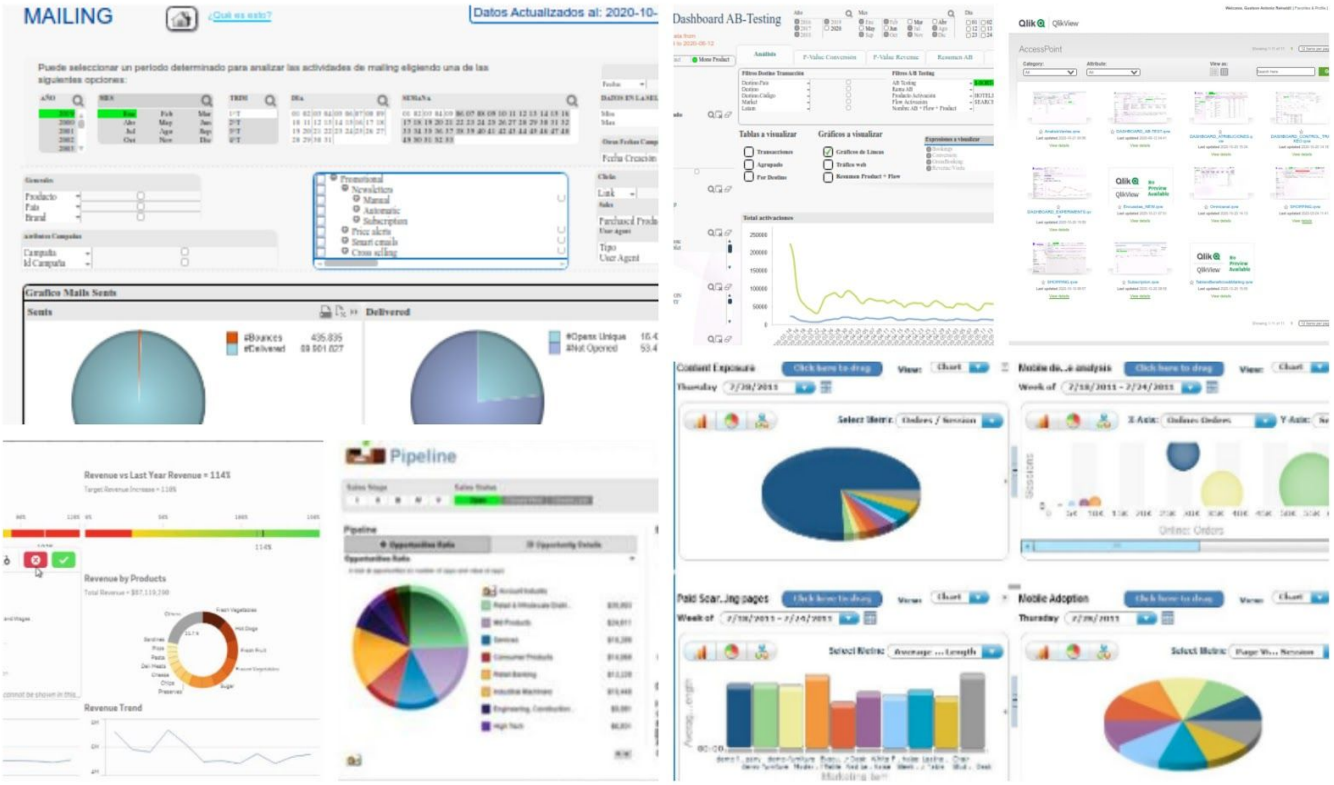

Fig. 34 - Vista de diferentes tableros, Mailing, *A/B Testing*, KPIs y puntos de entrada.

Capítulo 7

Conclusiones y Trabajos futuros

7.1 - Conclusiones

Esta tesina ha sido elaborada en el marco del Programa de Apoyo al Egreso de Profesionales en Actividad (PAEPA). Aquí se describe un desarrollo de *software* de envergadura llevado a cabo en la compañía Despegar.com: La creación de un "Tablero Omnicanal". Este tablero permite a la compañía obtener información relevante respecto del rendimiento de todas las comunicaciones y canales abiertos de comunicación con sus clientes.

La ejecución de este proyecto de desarrollo constituyó un esfuerzo colaborativo de varios equipos de trabajo, en el que el autor de esta tesina se desempeñó como analista, desarrollador y, muchas veces, como nexo operativo entre los distintos equipos.

Ha sido pretensión del autor que la lectura de este documento ilustre cabalmente el esfuerzo de trabajo que implicó la creación del tablero omnicanal. Todo ello sin abrumar al lector con demasiados detalles técnicos, pero al mismo tiempo, exponiendo los suficientes que permitan apreciar la magnitud del trabajo realizado.

La creación de este tablero omnicanal, impactó directamente sobre las comunicaciones de la compañía que mejoraron tanto en calidad como en frecuencia y rendimiento (conversión de usuarios). Sin embargo, las consecuencias de este desarrollo fueron más allá del ámbito de las comunicaciones. La mejora de rendimiento atravesó a otros sectores de la compañía que debieron adecuarse a este nuevo escenario. La obtención de información de calidad permitió identificar datos relevantes para calcular el retorno de la inversión ROI y la eficiencia de cada una de las acciones y campañas que emprende el equipo de *Marketing* Digital. También fue posible mejorar los KPIs comunicacionales y de otras áreas que se vieron afectadas por la suba en el rendimiento de las comunicaciones.

A lo largo de esta tesina se describe la forma en que se llevó adelante el proceso de trabajo que ocupó a los equipos de *Marketing*, Comunicaciones, *Data* y *Business Intelligence*. Construir una herramienta de análisis de datos como el tablero omnicanal requiere de una

gran cantidad de tareas. Orquestación y coordinación son acciones claves para cumplir con los objetivos propuestos.

Este proceso no ha sido un plan de unas pocas semanas. Requirió de esfuerzos sostenidos en el tiempo, con un enfoque y un plan flexible para poder llevarlo adelante.

La planificación a través de la herramienta de *Roadmap* sirvió para delimitar la construcción del tablero dentro de un presupuesto y con una estrategia de desarrollo definida de antemano. El uso de un *Roadmap* ágil posibilitó que los equipos involucrados puedan trabajar aspectos diferentes del tablero sin perder de vista el objetivo final. A su vez permitió contar con la suficiente flexibilidad para afrontar problemáticas que atraviesan transversalmente espacios de acción de equipos con enfoques de trabajo y aptitudes disímiles.

Es bastante común encontrar equipos de desarrollo que llevan tiempo en proyectos de software que nunca terminan o terminan mucho después de la estimación inicial, con el consecuente costo extra en el desarrollo. La razón de esto generalmente no es la falta de pericia o *expertise* del equipo involucrado sino más bien el escenario de no contar con una fecha precisa de lanzamiento del producto. Este escenario propicia el trabajo de mejora indefinido. El *Roadmap* no permite esta flexibilización de los plazos al establecer límites de tiempo con precisión.

El proceso de trabajo guiado por la historia de *Roadmap* condujo a los equipos a centrarse en el producto. Propició diseñar y desarrollar cada solución sin desviar la atención ni los objetivos principales de cada tarea adoptando el pragmatismo en las cercanías de las fechas límites establecidas para cada tarea [51].

Los plazos iniciales propuestos para la entrega de la primera versión del producto viable mínimo (MVP) fueron cumplidos. Es así como, en el seno de los equipos técnicos y gerenciales de la compañía, el proceso guiado por la herramienta de *Roadmap* ágil se ha considerado un éxito.

La realización de la historia de *Roadmap* también permitió entender la importancia de los datos en la compañía. Los datos se vislumbraron cómo un activo muy importante dentro de la organización del trabajo del equipo de *marketing*. Se logró que los equipos participantes incorporen el concepto de calidad de los datos como eje transversal en cada acción de nuevo desarrollo entendiendo que sin ellos la organización pierde activos.

Paralelamente, el proceso de desarrollo de este tablero significó horas de lecturas sobre tecnologías orientadas a brindar solución a cada uno de los problemas que se fueron planteando con su posterior discusión en mesas de trabajo. Este análisis, elección e

implementación de nuevas tecnologías aportó conocimiento y *expertise* a los equipos técnicos involucrados.

7.2 - Trabajos futuros

En esta sección se listan las distintas extensiones que se planean aplicar al presente trabajo, con el fin de ampliar y mejorar el proceso de adquisición de conocimiento para la toma de decisiones.

7.2.1 - Rendimiento por usuario y media

Actualmente se registran los eventos de apertura y de clic. Se pensó en una aplicación que provea de un servicio de recomendación del canal por donde comunicar una notificación a un usuario. La idea es que dado un usuario y un tipo de comunicación[63] el servicio devuelva una lista priorizada de las medias por donde ese usuario rinde mejor respecto de los KPIs de apertura y clic.

Si por ejemplo consideramos una campaña reactiva (enviada a un segmento de usuarios que están buscando un destino turístico), el servicio debe devolver, para cada usuario del segmento, una lista priorizada de canales por donde las comunicaciones enviadas a ese usuario tiene mejor tasa de apertura.

La registración ya está disponible. Restan dos tareas:

(1) crear el servicio que se encargue del procesamiento,

(2) desarrollar la API que permita su utilización.

7.2.2 - Ampliar el servicio de coordinación

Actualmente el servicio de coordinación determina si se permite el envío de una comunicación a un usuario por un canal de acuerdo con una matriz lógica. La idea es permitir que el usuario de *Marketing* o Comercial pueda generar su propio modelo de coordinación. Además de poder crear su propio modelo lógico de coordinación sería de utilidad contar con una herramienta de *A/B Testing* que permita al equipo que crea un modelo nuevo, compararlo con uno existente. Estas nuevas funcionalidades buscan ampliar la autonomía de los equipos de *Marketing* y Comercial manteniendo un proceso ágil de obtención de métricas, de resultados y toma de decisiones.

[63] El tipo de comunicación es un concepto adaptable a las preferencias del equipo de *marketing*. Podría darse considerando el producto ofrecido en una comunicación, o en el tipo de motivación que se use para enviar un mensaje (proactivo o reactivo a un evento de usuario), entre otras posibilidades.

7.2.3 - Revisión del proceso de envío de alertas

Alertas de precio es una funcionalidad ofrecida a los usuarios a través del *site* y de la aplicación *mobile* de la compañía. Los alertas permiten realizar un seguimiento de los cambios de precio de una ruta de vuelo, o de un hotel en un rango de fechas específico.

Cuando un usuario crea un alerta de precios, si el precio del producto alertado entra en concordancia con el precio del alerta, entonces el usuario recibe una notificación sobre la existencia del producto alertado.

La propuesta es procesar datos de los alertas creados, las comunicaciones de concordancia de alertas enviadas y las ventas asociadas.

Con estos datos se pretende obtener el tiempo promedio de creación de alertas por parte del usuario.

Por otra parte, si el usuario crea un alerta para un evento con una anticipación de varios meses, entonces es válido preguntarse el momento de ese tiempo en que el usuario terminará convirtiendo.

La idea es ofrecer información de que sucede en el espacio de tiempo entre que el usuario crea el alerta y finalmente el alerta termina (sea porque el producto alertado ya se venció o sea porque el usuario terminó comprando el producto).

Esta información permitirá enviar comunicaciones con un texto más ajustado de acuerdo al tiempo de vida del alerta (este trabajo es habitual en el equipo de *marketing* encargado del contenido de las comunicaciones) y ajustar la coordinación para permitir mayor cantidad de envíos en la zona donde estadísticamente ese usuario convierte mejor.

8 - Bibliografía

[1] Knowledge management tool for integrated decision-making in industry.E. Muñoz, L. Puigjaner, A. Espuña. 2012.

[2] Challenge of Understanding Multichannel Customer Behavior in the 21st Century. Kim Soohyun, Ahn Insook. 2014

[3] Why Multichannel Retail is Obsolete. Forbes. Brian Walker. 2011

[4] Introduction to the Special Issue Information Technology in Retail: Toward Omnichannel Retailing. Cuthbertson & Wojciech, 2014.

[5] Handelsmanagement. N. Pop, A. Dabija. 2008.
[6] Key Performance Indicators. David Parmenter. 2010.
[7] Doppler *site*. Promedio de aperturas por horario. https://www.fromdoppler.com/benchmarks/. Accedido el 18-11-2020.
[8] Doppler *site*. Promedio de aperturas por día. https://www.fromdoppler.com/benchmarks/. Accedido el 18-11-2020.
[9] Get Response site. *Average results by industry*. https://www.getresponse.com/resources/reports/email-marketing-benchmarks. Accedido el 18-11-2020.
[10] Conversion Rate Optimization. Greg Ahern. 2019.
[11] A / B Testing: The Most Powerful Way to Turn Clicks Into Customers.Dan Siroker, Pete Koomen. 2013.
[16] Improving Conversion Rates for Fashion e-Commerce with A/B Testing. R.Rahutomo, Y. Lie,A. Samsinga Perbangsa. 2020.
[17] Litmus *site. What is Preview Text?* https://www.litmus.com/blog/the-ultimate-guide-to-preview-text-support/. Accedido el 18-11-2020.
[18] Marketing Funnel. Rober Baker. 2019.
[14] Return On Investment – Indicator for Measuring the Profitability of Invested Capital. M. Zamfir, M. Manea, L. Ionescu. 2016.
[15] Concepto 05 *Site* . *Funnel Analytics* Https://www.concepto05.com/2013/03/redescubriendo-el-email-marketing-funnel-analytics/. Accedido el 18-11-2020.
[12] Email Marketing by the Num8ers. Chris Baggott. 2007
[13] Distributed and Cloud Computing from Parallel Processing to the Internet of Things. Hwang-Fox-Dongarra. 2012.
[19] Sistemas operativos (5° Edición), William Stalling. Pearson - Prentice Hall.
[20] Presto Documentation. Presto web site.
[21] Operating Systems - Internals and Design Principles - Stalling. 2005
[22] NoSQL: What's in a name?. Evans, Eric.
[23] Apache Cassandra and DataStax Enterprise Explained. Peter Halliday. 2014
[24] Murmur 3 in the dark. Gaurav Karkhanis. 2014
[25] MongoDB in Action. Kyle Banker. 2011.
[26] Brewer's Conjecture and the Feasibility of Consistent, Available, Vaish, "Getting Started with NoSQL". 2013.

Partition-Tolerant Web Services. Seth Gilbert, Nany Lynh.

[27] Gartner Glossary - Big Data - Gartner site. 2020.

[28] Solid Q *site.* https://blogs.solidq.com/es/business-analytics/que-es-mapreduce/. Accedido el 18-11-2020.

[29] Aprender *Big Data site.* https://aprenderbigdata.com/hadoop-yarn/. Accedido el 18-11-2020.

[30] Aprender *Big Data site.* https://aprenderbigdata.com/hadoop/. Accedido el 18-11-2020.

[31] Data Lake. Superando las limitaciones del Data Warehouse. Powerdata. 2020.

[32] Learning Spark. H. Karau, A. Konwinski, P. Wendell, M. Zaharia. 2015.

[33] Geeky Theory *site.* Modelo de programación. https://geekytheory.com/apache-spark-que-es-y-como-funciona. Accedido el 18-11-2020.

[34] Geeky Theory *site.* Tipo de transformaciones. https://geekytheory.com/apache-spark-que-es-y-como-funciona. Accedido el 18-11-2020.

[35] Geeky Theory *site.* Tipo de operaciones en las transformaciones. https://geekytheory.com/apache-spark-que-es-y-como-funciona. Accedido el 18-11-2020.

[36] Mastering Elastic Stack. R. Gupta- Y. Gupta. 2017.

[37] Machine Learning with Elastic Stack. R. Collier, B. Azarmi. 2019.

[38] Coding Explained *site.* https://codingexplained.com/coding/elasticsearch/understanding-sharding-in-elasticsearch. Accedido el 18-11-2020.

[39] Elasticsearch: The Definitive Guide. Clinton Gormley, Zachary Tong. 2011.

[40] Hazelcast *site.* https://hazelcast.com/glossary/in-memory-data-grid/. Accedido el 18-11-2020.

[41] Dremel: Interactive Analysis of Web-Scale Datasets. S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis. 2010.

[42] Presto Documentation. Presto web site.

[43] Diego Calvo *site.* https://www.diegocalvo.es/en/file-formats-big-data/. Accedido el 18-11-2020.

[44] Patrones de diseño. Erich Gamma. 2002.

[45] A Big Data Modeling Methodology for Apache Cassandra. Chebotko. 2015.

[46] Integración asíncrona con Apache Kafka. Redhat site.

[47] Software Systems Architecture. Nick Rozanski, Eoin Woods. 2011.

[48] Data Governance: How to Design, Deploy, and Sustain an Effective Data Governance Program. John Ladley 2012.

[49] All about cookies *site. What is a cookie?.* https://www.allaboutcookies.org/cookies/. Accedido el 18-11-2020.

[50] Estadística aplicada básica. 2° edición. David S. Moore - 2005.

[51] Guía metodológica de RoadMapping para proyectos de innovación. Jordi Rodríguez, Joaquim Lloveras. XIV International congress on project engineering. Madrid 2010.