# Documentación de Análisis y Diseño de la gestión de notificaciones

**Documentación de Análisis y Diseño de Gestión de Notificaciones**
**PMPV**



## Introducción

### Propósito

El objetivo de esta documentación es describir el análisis y diseño de la gestión de notificaciones en el sistema Planificación Mantenimiento Preventivo Vehicular (PMPV).

### Alcance

La documentación aborda el análisis y diseño relativo a la gestión de notificaciones que aborda los siguientes servicios:

- Servicio Tarea
- Servicio Notificación
- Servicio Usuario

## Contexto del Proyecto

### Descripción del Proyecto

El sistema de planificación de mantenimiento preventivo vehicular tiene como objetivo principal gestionar el mantenimiento programado de una flota de vehículos. La aplicación facilitará la gestión de las tareas de mantenimiento proporcionadas por el fabricante como de aquellas generadas por el propio operador. Además, se gestionarán las revisiones de las tareas, notificando cualquier cambio a los ingenieros responsables de la gestión y planificación del mantenimiento de vehículos.

Los ingenieros crearán planes de mantenimiento para vehículos similares que al aplicarlos generarán, automáticamente, órdenes de trabajo. Estas, serán gestionadas por los mecánicos, los cuales también podrán crear nuevos trabajos debido a la detección de errores durante la realización de sus trabajos. Además, la aplicación gestionará los diferentes tipos de usuario (ingenieros, mecánicos, administradores, etc..) que podrán acceder de forma segura a las diferentes funcionalidades de esta.

Cabe destacar que el plan de mantenimiento preventivo contribuye a prevenir fallos y garantizar el correcto funcionamiento de los vehículos. De hecho, la aplicación se deberá diseñar para que gestione un crecimiento y sea escalable y esté en la nube, con procesos automatizados para minimizar errores y garantizar su calidad.

### Stakeholders

Fabricante

# Requisitos

## Requisitos funcionales

PMPV-28: US-2023-Como ingeniero quiero recibir una notificación cuando se haya creado una nueva tarea de mantenimiento por parte del fabricante FINALIZADA

PMPV-29: US-2023-Como ingeniero quiero recibir una notificación cuando se haya actualizado una tarea de mantenimiento por parte del fabricante FINALIZADA

PMPV-30: US-2023-Como ingeniero quiero recibir una notificación cuando se haya eliminado una tarea de mantenimiento por parte del fabricante que esté asociado a un plan de mantenimiento FINALIZADA

PMPV-48: US-2023-Como arquitecto quiero definir la estrategia de notificaciones FINALIZADA

## Requisitos no funcionales

PMPV-102: RS-Escalabilidad y Resiliencia TAREAS POR HACER

# Análisis

## Especificación

Teniendo en cuenta las siguientes historias de usuario acordadas anteriormente:

US-2023-Como ingeniero quiero recibir una notificación cuando se haya actualizado una tarea de mantenimiento por parte del fabricante.

US-2023-Como ingeniero quiero recibir una notificación cuando se haya eliminado una tarea de mantenimiento por parte del fabricante que esté asociado a un plan de mantenimiento

US-2023-Como arquitecto quiero definir la estrategia de notificaciones

US-2023-Como ingeniero quiero recibir una notificación cuando se haya creado una nueva tarea de mantenimiento por parte del fabricante

El fabricante puede realizar diversas solicitudes a través de los *endpoint* del Servicio de Tareas. Estas solicitudes incluyen la creación de una nueva tarea de mantenimiento, la actualización de los datos de una tarea existente o la eliminación de una tarea que ya no es necesaria. A continuación, se detallan los distintos endpoints del servicio relativos a la gestión de tareas del fabricante:

Creación de una tarea de mantenimiento por parte del fabricante:

- Método: POST
- URL: **tareas/fabricante**

Actualización de una tarea de mantenimiento por parte del fabricante:

- Método: PUT
- URL: **tareas/fabricante/{tareaId}**

Eliminación de una tarea de mantenimiento por parte del fabricante:

- Método: DELETE
- URL: **tareas/fabricante/{tareaId}**

Una vez validada y completada la acción, se genera un evento correspondiente a la creación, actualización o eliminación. Este evento es publicado en el bus de mensajes Kafka. Posteriormente, el Servicio de Notificación se conecta, recibe el mensaje, obtiene las direcciones de correo electrónico de los ingenieros y les envía un correo electrónico informándoles de dichas acciones.

## Colaboraciones entre servicios

A partir de la especificación, para comprender mejor la relación entre los diversos componentes e identificar las colaboraciones entre los diferentes servicios y dependencias, se definen los servicios, las operaciones y las colaboraciones que se ven afectados durante el proceso indicado anteriormente.

| Servicio | Operación | Colaboración |
|---|---|---|
| Servicio Tarea | crearTareaFabricante()<br>modificarTareaFabricante()<br>eliminarTareaFabricante()<br>obtenerDetalleTarea() | |
| Servicio Notificación | enviarCreacionTarea()<br>enviarModificacionTarea()<br>enviarEliminacionTarea() | Servicio Usuario:<br>obtenerCorreoIngenieros()<br><br>Servicio Tarea:<br>obtenerDetalleTarea() |
| Servicio Usuario | obtenerCorreoIngenieros() | |

Servicios Operaciones Colaboraciones

En este proceso se ven involucrados tres servicios, donde:

El **servicio de tarea** gestiona las tareas de mantenimiento manuales y del fabricante y cuyas operaciones:

*crearTareaFabricante()*: Permite al fabricante crear una nueva tarea de mantenimiento.

*modificarTareaFabricante()*: Permite al fabricante actualizar una tarea existente del fabricante.

*eliminarTareaFabricante()*: Permite al fabricante eliminar una tarea existente del fabricante.

*obtenerDetalleTarea()*: Permite obtener el detalle de una tarea en concreto.

- El **servicio de usuario** gestiona los usuarios y cuya operación:
  - *obtenerCorreoIngenieros()*: Permite recuperar las direcciones de correo electrónico de los usuarios que tienen el rol de ingeniero.

El **servicio de notificaciones** gestiona las notificaciones entre los diferentes servicios y cuyas operaciones:

*enviarCreacionTarea()*: Se encarga de enviar notificaciones cuando se crea una nueva tarea de mantenimiento.

*enviarModificacionTarea()*: Se encarga de enviar notificaciones cuando se actualiza una tarea existente.

*enviarEliminacionTarea()*: Se encarga de enviar notificaciones cuando se elimina una tarea.

El **Servicio Tarea** publica eventos de creación, modificación y eliminación de tareas en el bus de mensajes Kafka, y el **Servicio Notificación** consume estos eventos para enviar correos electrónicos a los ingenieros.

El **Servicio Notificación** colabora con el **Servicio Usuario** para obtener las direcciones de correo electrónico de los ingenieros a quienes se enviarán las notificaciones.

# Diseño

## Diseño notificaciones

A continuación, se establece el diagrama que muestra la interacción entre los servicios de tarea, notificación y usuario en una arquitectura distribuida que permite visualizar de una forma fácil como se comunican entre sí.

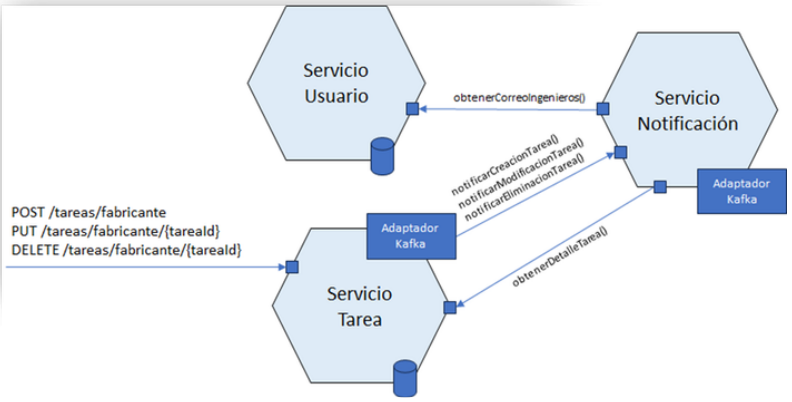



Diagrama de interacción entre servicios

Tal y como se puede observar en el diagrama, los servicios de tarea y notificación deberán tener un adaptador para poder recibir y enviar notificaciones. Sin embargo, la obtención de información entre los diferentes servicios se realizará a través de una comunicación vía REST. A continuación, se explica en detalle la estrategia que se llevará a cabo para realizar dichas notificaciones.

## Estrategia de Notificaciones

La estrategia de notificaciones debe tener en cuenta los siguientes requisitos no funcionales:

**Escalabilidad**. Permite manejar grandes volúmenes de mensajes.

**Distribución Efectiva**. Distribución que facilita la comunicación entre los diferentes servicios.

**Tolerancia a fallos**. Ofrece una solución a los fallos que se puedan producir asegurando la continuidad del proceso.

Para establecer una comunicación efectiva y atómica entre los servicios de tareas, notificaciones y usuarios, y garantizar que la arquitectura cumpla con las características indicadas anteriormente, se aplica el patrón mensajería y el patrón bandeja de salida transaccional.

El **patrón mensajería** permite la comunicación entre servicios de forma asíncrona, es decir, mediante el intercambio de mensajes a través de canales de mensajería sin necesidad de que ambos servicios estén disponibles. Para ello, se utiliza el patrón Publicador/Suscriptor que facilita que un servicio publique un mensaje para que otro u otros consumidores lo puedan recibir.

Para que el proceso sea atómico y se eviten problemas de inconsistencias de datos como, por ejemplo, cuando se crea, actualiza o elimina una tarea y al producirse un fallo de base de datos el mensaje se envía, se utiliza el **patrón bandeja de salida transaccional** o eventos de aplicación.

El patrón bandeja de salida transaccional publica un mensaje como parte de una transacción de base de datos, es decir, se envía el mensaje si se confirma la transacción en base de datos, en caso contrario no se envía. Además, los eventos se enviarán en el mismo orden en el que fueron realizados.

Por lo tanto, el servicio que envía el mensaje almacena el evento en la base de datos como parte de la transacción y luego, un proceso intermedio lo envía al bus de mensajería tal y como se puede observar en el siguiente diagrama:

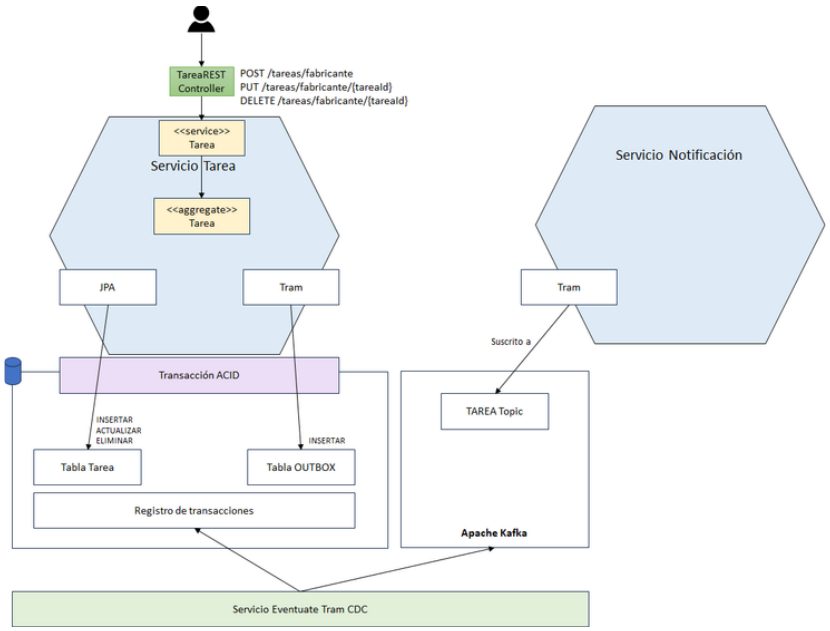



Diagrama del Patrón Bandeja de Salida Transaccional [38]

Para poder llevar a cabo lo indicado anteriormente, se utilizarán las siguientes tecnologías las cuales se deberán integrar con Spring Boot:

**Apache Kafka** se utiliza como un sistema de mensajería para el intercambio de eventos entre diferentes servicios de una aplicación distribuida. Además, permite gestionar flujos masivos de datos de una forma eficaz. Algunas de sus características principales son:

Permite la publicación y subscripción de eventos en tiempo real gracias al patrón Publicación/Subscripción.

Los eventos se almacenan permitiendo su reprocesamiento.

Es altamente escalable y permite manejar grandes volúmenes de eventos.

Garantiza la replicación de los eventos entre los diferentes nodos del clúster.

**Apache Zookeeper** se emplea para coordinar y gestionar la configuración de los nodos del clúster de Kafka facilitando su gestión y mejorando su eficacia.

A continuación, se muestra como Apache Kafka y Zookeeper llevan a cabo la transmisión de mensajes.

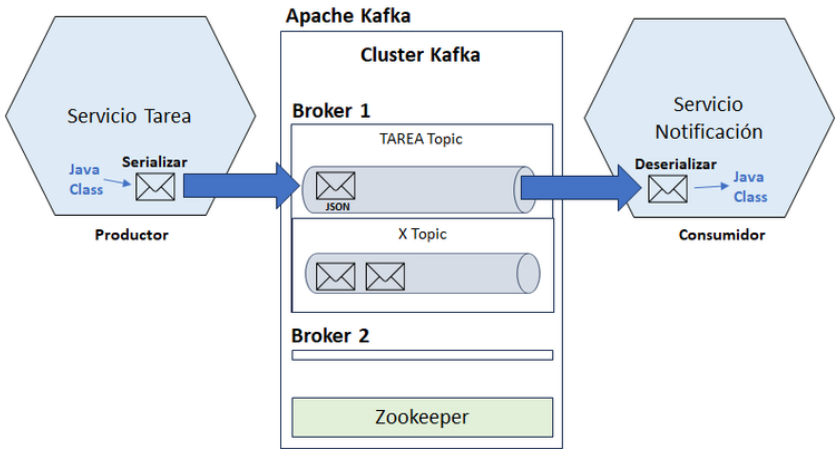


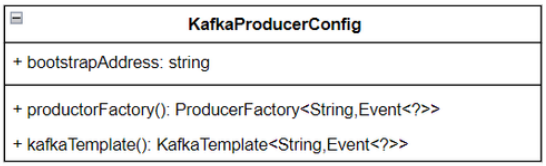

Diagrama de comunicación de mensajes del Servicio Tarea al Servicio Notificación

Para que el Servicio Tarea actúe como un **publicador** o **productor** de Kafka e interactúe de forma efectiva con Kafka, se tiene que configurar la siguiente infraestructura:

**KafkaProducerConfig**. Una clase de configuración que permite el envío de mensajes utilizando Apache Kafka. Además, debe tener los siguientes métodos:

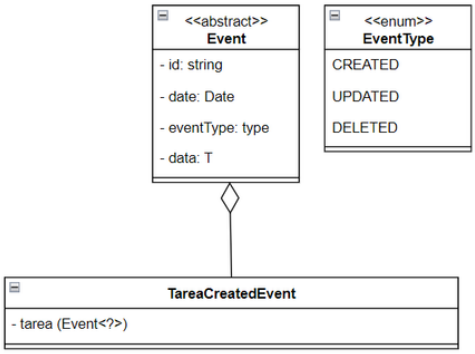
**ProducerFactory**(): Crea y devuelve una instancia de ProducerFactory, el cual es responsable de producir instancias de productores de Kafka. Esta clase incluye la configuración necesaria para establecer la conexión con los servidores de Kafka, como la dirección del bus de mensajes de Kafka, la clave con la que se serializan los mensajes y el serializador para convertir los objetos Java en formato JSON.

**KafkaTemplate**(): Crea y devuelve un objeto KafkaTemplate que utiliza la configuración de ProducerFactory. Este objeto proporciona el método enviar que permite enviar mensajes al bus de mensajes de Kafka.



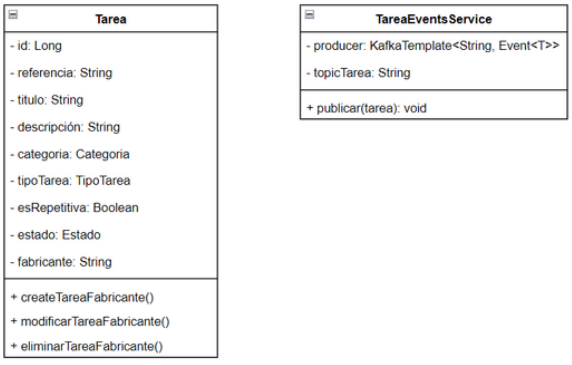

Configuración Productor Kafka

Tal y como se puede observar, en este caso, se puede enviar cualquier mensaje que cumpla la clase abstracta *Event* en la cual se informa un identificador, una fecha, el tipo de acción que es (creación, actualización o eliminación) y la propia tarea.




Evento que se envía a otro servicio

Una vez configurado, es necesario inyectar el objeto KafkaTemplate en el lugar donde se quiere enviar el evento, en este caso, se crea una clase que contine el método que lo publicará y que será llamado desde el servicio de la tarea cuando se cree, actualice o elimine una tarea de fabricante.




Servicio que publica el evento

Para enviarlo a través del método *SEND* de KafkaTemplate se debe especificar el topic al que se enviará y el evento que se enviará. Este último, extiende de la clase Event donde se incluye el tipo de acción (creación, actualización o eliminación), el identificador del evento, la fecha de creación y la tarea.

Por lo tanto, con el topic y los datos de configuración del productor como es la dirección de bus de mensajería, se produce el envío del evento a la correspondiente “tubería” o topic de Kafka en la cual se publica. Es importante resaltar que existen herramientas que permiten ver los mensajes que se han enviado en los diferentes topics de Kafka como, por ejemplo, Offset Explorer.

Finalmente, los consumidores subscritos al topic podrán leer los mensajes. Pero, para ello, es necesario que el servicio Notificación sea configurado como consumidor y, por lo tanto, debe tener la siguiente configuración:

**KafkaConsumerConfig**. Una clase de configuración que permite conectarse y leer los mensajes de Apache Kafka. Esta clase tiene los siguientes métodos:

**ConsumerFactory**(): Crea y devuelve una instancia de ConsumerFactory, el cual es responsable de producir instancias de consumidores de Kafka. Esta clase incluye la configuración necesaria para establecer la conexión con Kafka como es la dirección del bus de mensajería. Además, de proporcionar,  el deserializador de la clave y del evento que permitirá transformar el JSON en una clase de Java.

**KafkaListenerContainerFactory**(): Crea y configura un objeto ConcurrentKafkaListenerContainerFactory que crea contenedores de escucha y que utiliza la configuración de ConsumerFactory.


Configuración Consumidor Kafka

En resumen, se establece la configuración necesaria para consumir mensajes desde un servidor Kafka y transformar los mensajes JSON en objetos tipo Event.

Finalmente, para que se ejecute el método que lea los eventos que llegan a un topic de Kafka es necesario configurar lo siguiente:

**TareaEventsService:** Clase que contiene el método **consumidor**, el cual, es marcado como consumidor de Kafka a través de la anotación **@KafkaListener** donde se especifican los siguientes parámetros:

- El topic sobre el que escucha
- El contenedor de Kafka que se usa para leer el mensaje, en este caso KafkaListenerContainerFactory.
- El grupo de consumidores que lo escuchan.

Todo ello, permite que el método se ejecute cada vez que llegue un mensaje al topic especificado, lo lea y envíe los correos electrónicos a todos aquellos usuarios cuyo perfil sea ingeniero.


Servicio subscrito al topic de Kafka

A continuación, se muestra el diagrama de actividad donde se puede ver el flujo completo del proceso.

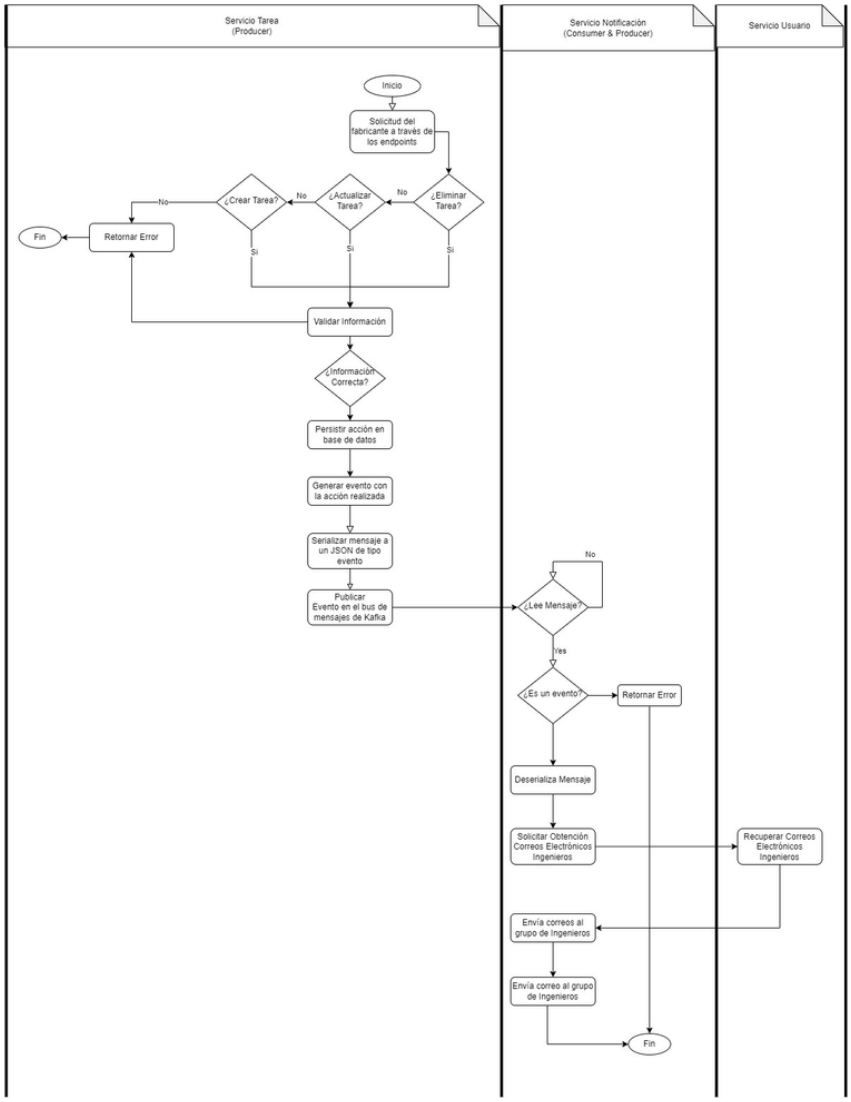



Diagrama de actividad de la creación de tareas del fabricante

Es importante resaltar que la decisión de este estilo arquitectónico ha sido derivada por la adaptabilidad que tiene a los cambios, es decir, la notificación no contiene la información de la tarea o de los usuarios, sino que el servicio notificación tiene que obtenerlo del correspondiente servicio. De esta forma, existe un mayor acoplamiento entre servicios y aumentaría la gestión de contratos, pero, sin embargo, se adaptaría mejor a posibles cambios que son muy probables dado que estamos en una fase inicial de la aplicación.