**PRUEBA DE CONCEPTO COLA DE MENSAJES EN DOCKER UTILIZANDO KAFKA**

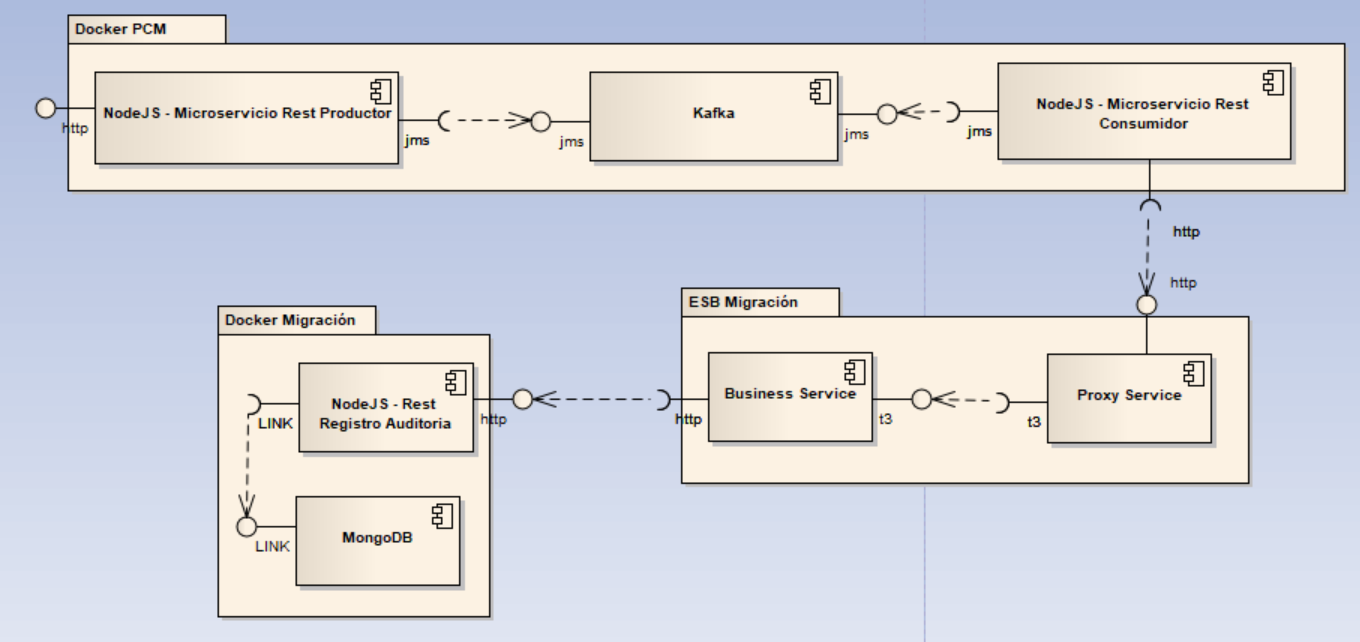
Objetivo de la prueba: Evaluar la viabilidad de crear una cola de mensajes para persistir le auditoria en todos puntos de control a través de un Docker para mejorar la calidad de la información que se persiste y eliminar dependencias con las BD de los puntos de control.

**Docker**, es una plataforma para desarrolladores y administradores de sistemas para desarrollar, implementar y ejecutar aplicaciones con contenedores ligeros y portables para que las aplicaciones de software puedan ejecutarse en cualquier máquina con **Docker** instalado. Es una tecnología para desplegar aplicaciones. Los contenedores no son nuevos, pero su uso para implementar aplicaciones fácilmente sí lo es.[[1]](#footnote-1)

**Kafka[[2]](#footnote-2),** es una plataforma de transmisión distribuida, Permite publicar y suscribirte a flujos de registros, es similar a una cola de mensajes o un sistema de mensajería empresarial. Permite almacenar secuencias de registros de una manera tolerante a fallas. Permite procesar flujos de registros a medida que ocurren.

Para Migración Colombia es importante contar con una auditoria confiable, que permita buscar de manera fácil y rápida la información enviada a través de los diferentes servicios que consumen las aplicaciones, en especial en esos puntos de control donde la conexión es intermitente, como mejora en el proceso de auditoria se propuso crear una cola de mensajes instalada de manera local en cada punto de control en el que se enviaran mensajes de los datos enviados de cada máquina que ejecute un movimiento migratorio, ya que muchas veces en la auditoria de la BD no se visualiza el consumo de los servicios, por problemas de comunicación en la red el consumo del servicio muchas veces ni siquiera sale de la aplicación local al Bus de Servicios, como se instalara de manera local, se garantiza la persistencia de los mensajes.

El modelo planteado se describe a groso modo en la siguiente imagen:

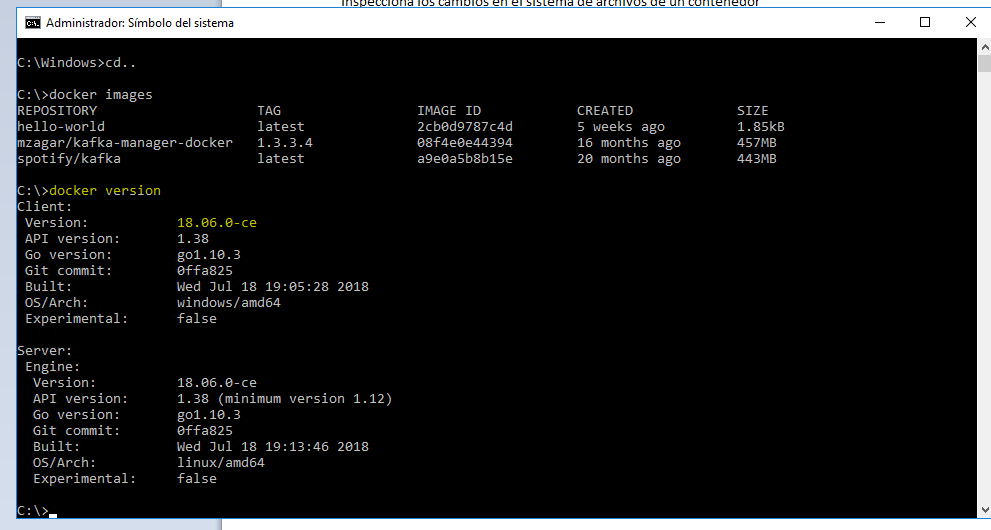


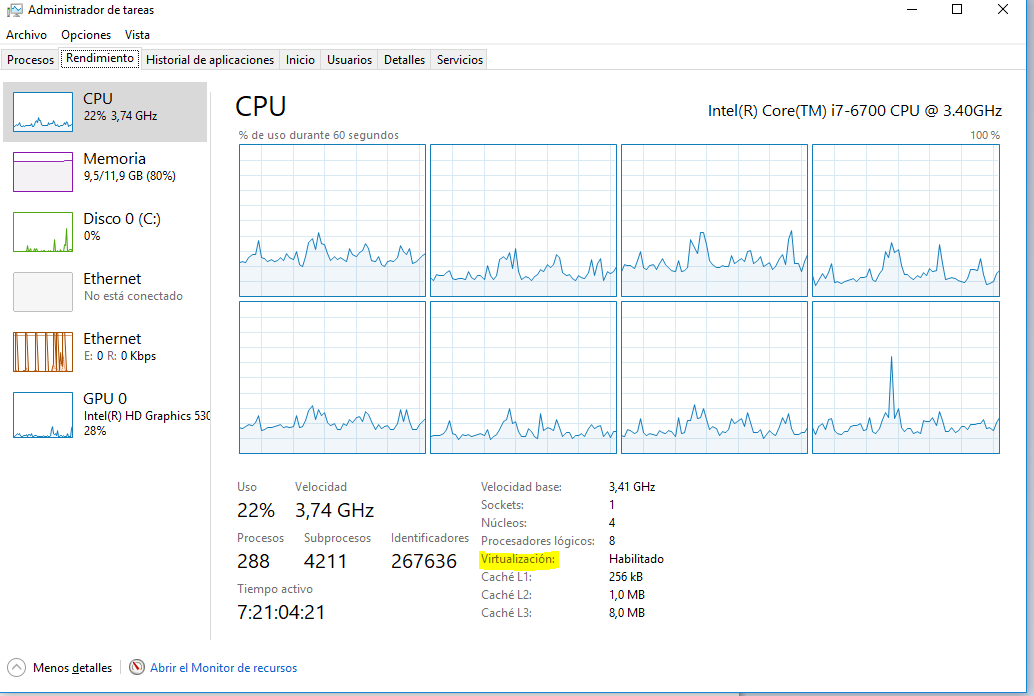
**Inicialmente cada una de las maquinas en un punto de control tendrá acceso a un microservicio mediante el cual se enviara un mensaje con la información necesaria de auditoria que se decida enviar, el microservicio (que actúa como productor) enviara todos los mensajes a la pila de mensajes (Kafka). Posteriormente y a través de otro microservicio (el consumidor) se enviara cada uno de los mensajes encolados al bus de servicio donde se procesan y se persisten en una BD MongoDB.**

Para eso se inicia una prueba de concepto utilizando las herramientas antes mencionadas, se intentó instalar el Docker en una maquina con Windows 7, infortunadamente la configuración para este sistema operativo tiene demasiadas restricciones.

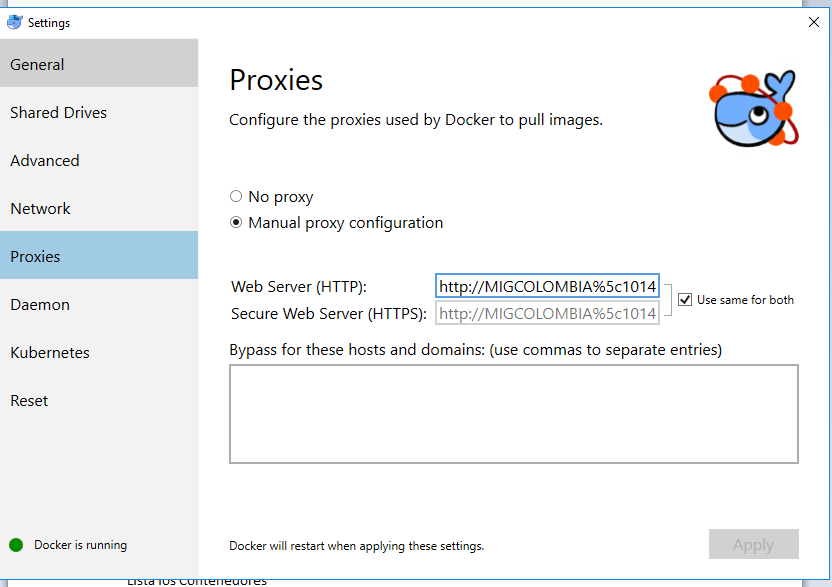
Se decidió seguir con la prueba de concepto en máquinas Windows 10, donde la única configuración que solicito la instalación fue activar el Hyper-V (tecnología que, mediante software y aprovechando los procesadores de 64-bits, permite virtualizar un segundo o tercer sistema operativo, siempre y cuando el hardware lo permita.)

Se valida que el Docker esté instalado y corriendo.





Posteriormente se configuro el Proxy en el Docker



Después de instalado y de ver que el Docker ya se estaba ejecutando se procedió a realizar la investigación de Kafka para Docker en Windows.

***Apache Kafka[[3]](#footnote-3)*** es una plataforma de transmisión distribuida, que le permite construir un canal de mensajería confiable entre varios sistemas. Su funcionalidad admite el modelo de publicación / suscripción, la persistencia de mensajes confiable, el procesamiento de flujo en tiempo real y muchos más.

Dentro de la investigación encontramos una Imagen que contiene Kafka y Zookeeper[[4]](#footnote-4) para Docker y procedimos a instalarla mediante la creación del Docker compose.[[5]](#footnote-5)

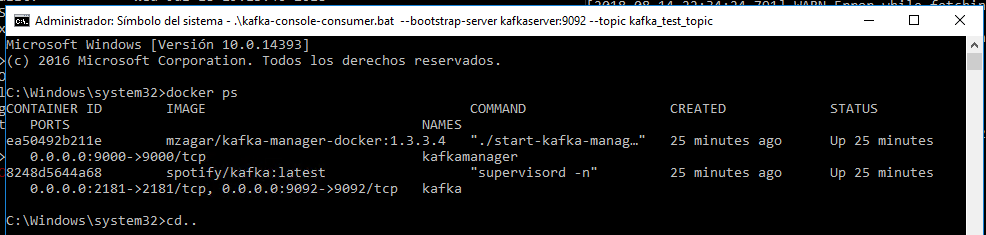
En el archivo se declara primero el **kafkaserver** que ejecuta la imagen de Kafka desarrollada por **Spotify[[6]](#footnote-6)** (incluye Zookeeper). Se establece un puerto y un host de escucha y posteriormente en un segundo contenedor se descarga el kafka\_manager ejecuta la herramienta kafka-manager, esta necesitara conectarse al servidor antes configurado.

[[7]](#footnote-7)

Después de creado, se debe ejecutar desde una terminal.

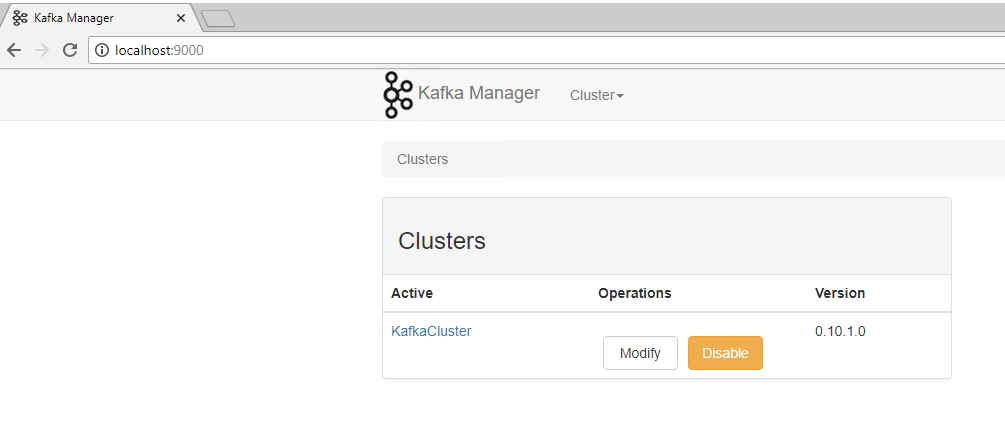


Posteriormente, validamos la lista de contenedores para validar que estén creados.



Para validar que el administrador de Kafka haya quedado instalado y en línea accedemos a

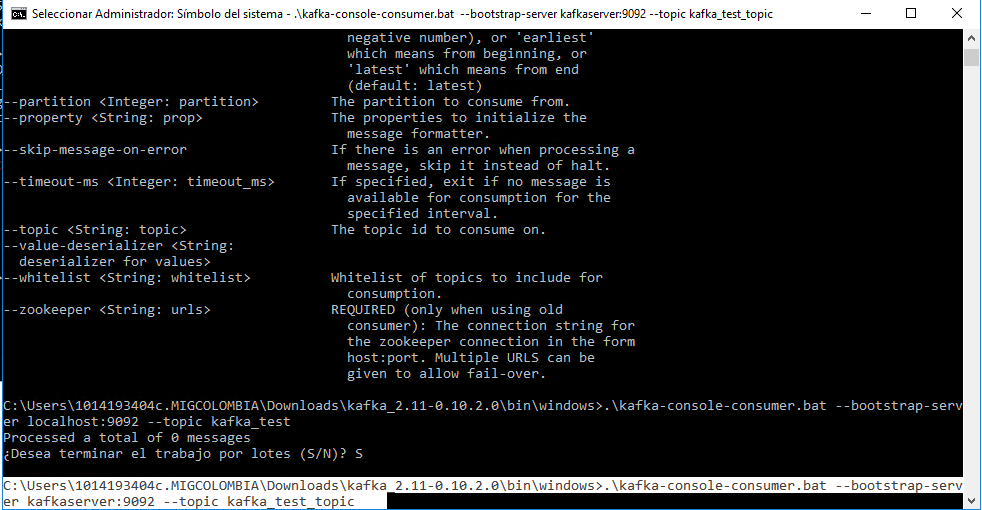
http://localhost:9000

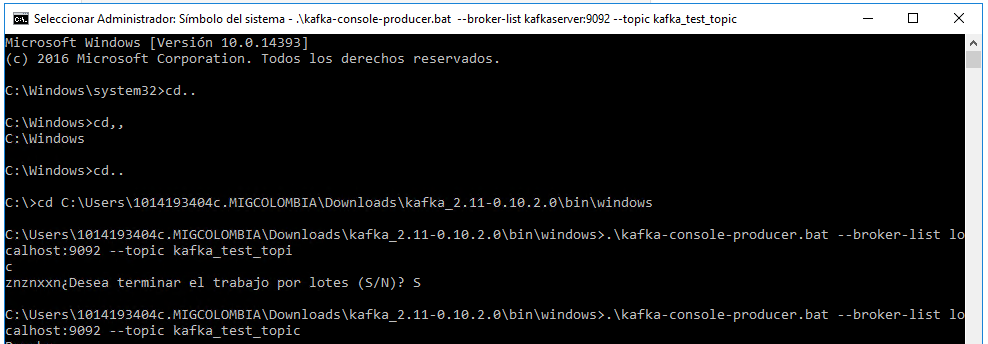


Con esto, nos damos cuenta de que quedo correctamente instalado. Ahora hay que probarlo, para ello utilizamos unos scripts provistos por Apache y simplemente lanzamos dos de ellos, uno para el productor y otro para el suscriptor.

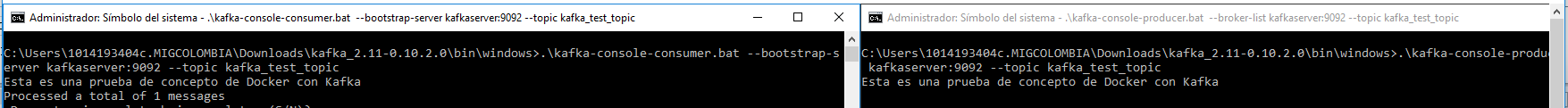
Estos scripts son públicos y se pueden descargar de la página: <https://www.apache.org/dyn/closer.cgi?path=/kafka/0.10.2.0/kafka_2.11-0.10.2.0.tgz>

Ejecutamos el script de productor y consumidor únicamente:

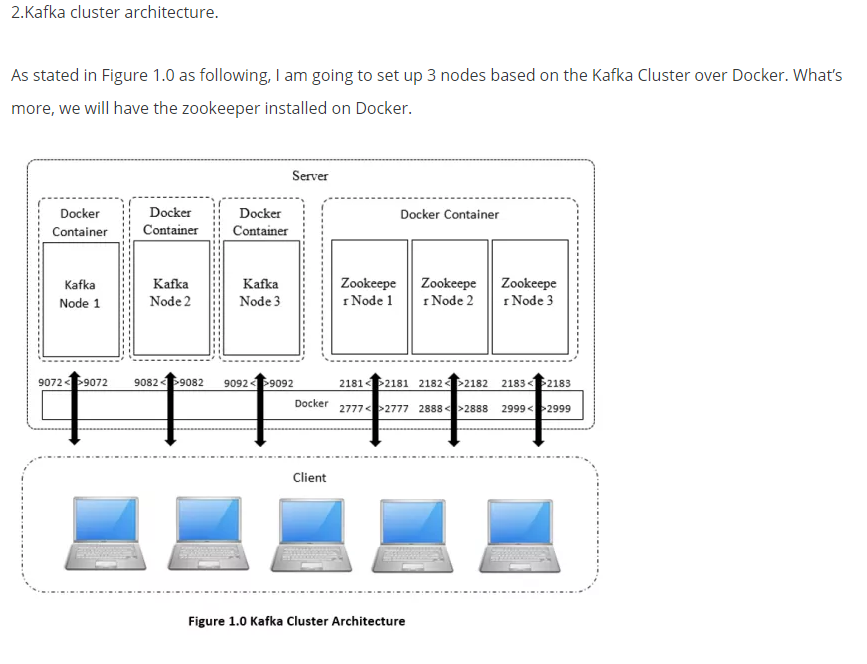




En este caso el consumidor está “escuchando” y el productor está listo para enviar mensajes: probamos y el resultado es el esperado.



A continuación se describe el diagrama de arquitectura de Kafka para un mejor entendimiento.



TO DO:

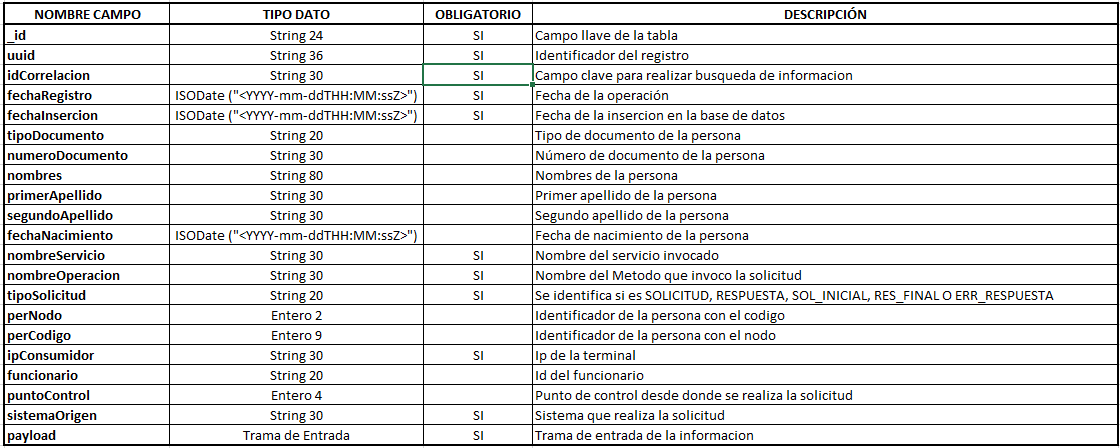
Estas son las actividades que nos hacen falta para poder terminar nuestra prueba de concepto.

* La implementación de los Microservicios con Node.JS:
* Para el productor es necesario que se defina la estructura del mensaje que se quiere enviar.
* Se deben hacer pruebas de carga que consuman el servicio y garantizar que se envíen mensajes a Kafka.
* Para el consumidor, Se debe capturar los mensajes encolados en Kafka, se debe realizar la conexión al OSB, en el caso de no poder generar conexión o se genere read timeout, se debe reinyectar en Kafka para que no se pierda el mensaje.
* Establecer la forma en que se van a realizar los reintentos de conexión al bus.
* Pruebas de rendimiento y de Integración.

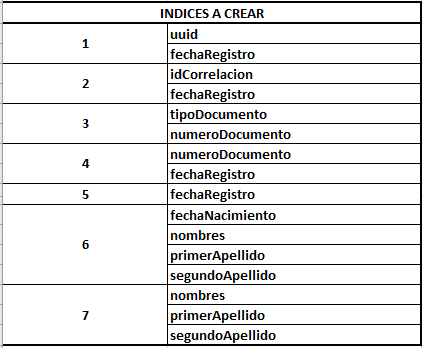
Y estas son las actividades adicionales, que no requieren prueba de concepto porque tenemos certeza de que funcionan.

* El OSB debe exponer un servicio Rest que reciba las peticiones del microservicio consumidor.
* El OSB debe consumir el microservicio que procesa y envía la información al MogoDB.
* Se debe definir la estructura de la BD Mongo.

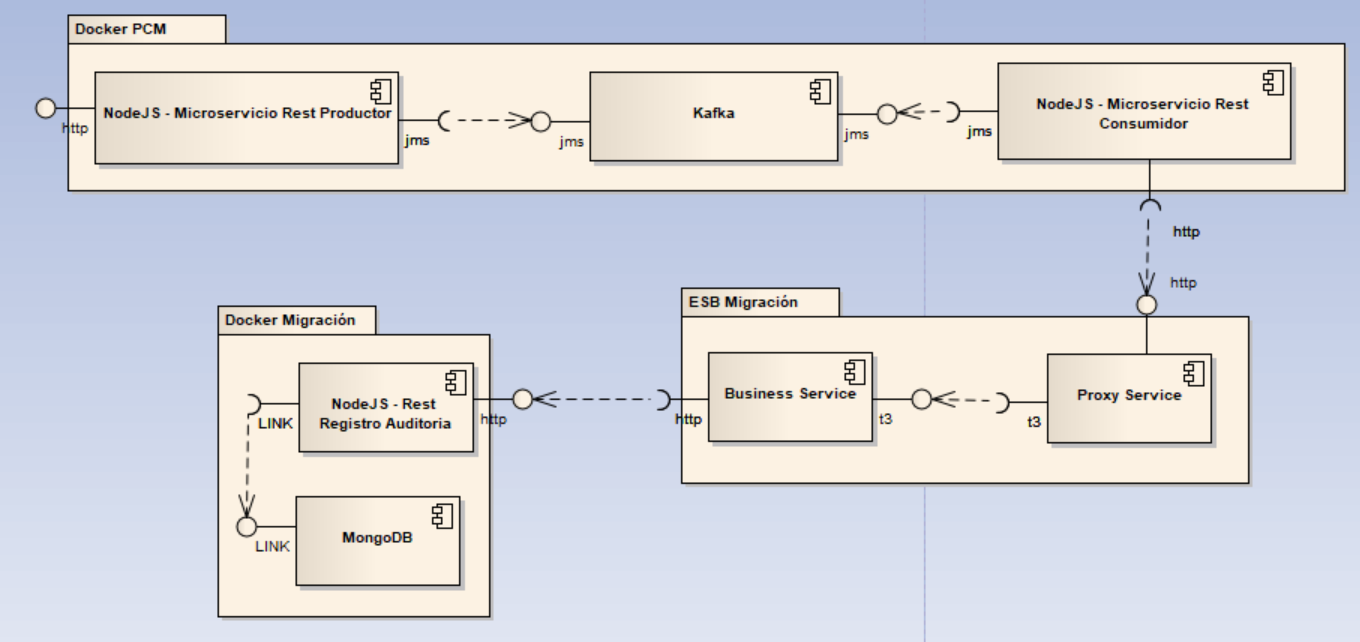
Estructura de Tabla de Auditoria Servicios en MONGO DB



Índices para consulta de información

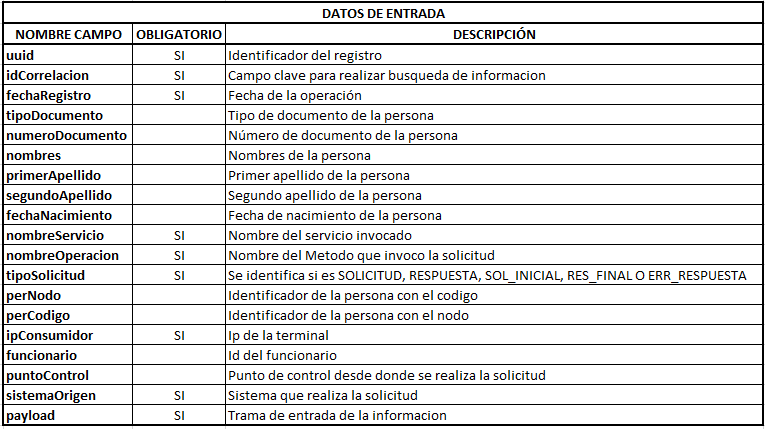


Diseño de componentes del proceso

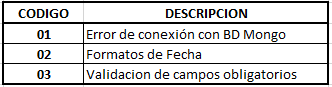


Creación de servicio que guarda información en la base de datos Mongo.

Se va realizar un servicio en Node js el cual recibe los parámetros siguientes:



Este servicio debe retornar:



1. https://docs.docker.com/get-started/ [↑](#footnote-ref-1)
2. http://kafka.apache.org/intro [↑](#footnote-ref-2)
3. https://zablo.net/blog/post/setup-apache-kafka-in-docker-on-windows [↑](#footnote-ref-3)
4. https://zookeeper.apache.org/ [↑](#footnote-ref-4)
5. https://docs.docker.com/compose/overview/ [↑](#footnote-ref-5)
6. https://hub.docker.com/r/spotify/kafka/ [↑](#footnote-ref-6)
7. Este es el Docker-Compose ejecutado [↑](#footnote-ref-7)