Contents

[Documentación búsqueda organigrama. 2](#_Toc88067292)

[REQUERIMIENTOS PREVIOS 2](#_Toc88067293)

[FUNCIONALIDADES Y USO 2](#_Toc88067294)

[FUNCIONALIDAD LOGICA 3](#_Toc88067295)

[LEVANTAR NODO DE KAFKA 4](#_Toc88067296)

[Creamos nuestros dos topics 5](#_Toc88067297)

# Documentación búsqueda organigrama.

El siguiente documento contiene paso a paso el proceso para realizar la practica solicitada por

Banorte con la finalidad de mostrar el demo de una API REST que consulta a un colaborador de una empresa y a su vez lista los nodos que están por arriba de el de acuerdo con el organigrama.

Dicha API será implementada con expressjs, un lenguaje de programación typescript, se

utilizarán librerías como, body-parser, log4js, nodemon, kafkajs, ts-node.

## REQUERIMIENTOS PREVIOS

Antes de empezar la práctica se definió una versión de node con cual trabajar en este caso

se optó por la versión node v12.18.0 y como gestor de paquetes npm 6.14.4.

Graphical user interface

Description automatically generated with medium confidence

Se instalo typescript de manera global ayudándome de la documentación

https://www.typescriptlang.org/

así mismo partí el proyecto instalando las librerías iniciales guiándome de la

documentación de expressjs.

<https://expressjs.com/es/starter/installing.html>.

## FUNCIONALIDADES Y USO

La práctica requería que se realizara un servicio con los siguientes requerimientos:

* El programa deberá retornar la ruta o rutas posibles de un nodo dado al nodo raíz.
* Crear al menos 30 colaboradores asignados a diferentes niveles y relacionarlos para realizar diferentes consultas.
* La dirección de las rutas siempre debe ser hacia el nivel más alto (Nodo Raíz)
* Este servicio deberá de ser capaz de soportar 15,000 peticiones por segundo.

Como datos de entrada teníamos al número del colaborador

Y este es un ejemplo de datos de salida

Text

Description automatically generated

En donde encontramos un objeto con un parámetro *resutl*, el cual nos muestra la información del usuario que se buscó y en *path* se muestra la información de sus nodos que se encuentran por arriba de él.

El organigrama que se utilizo para esta prueba es el siguiente:

Chart, box and whisker chart

Description automatically generated

En donde se muestran alrededor de 35 colaboradores donde algunos tienen una persona arriba de ellos al cual deben responder.

## FUNCIONALIDAD LOGICA

Se Configuro un nodo de kafka para estar monitoreando los mensajes de entrada y salida y del lado del API Rest se configuro el cliente con un *producer y un consumer.*

El Producer podría servir para después de cualquier actualización (eliminar, crear, actualizar un usuario) en base de datos y publicar el mensaje en el topic que se creó en este caso llamado stream-out.

El Producer manda un mensaje con el siguiente formato.

Text

Description automatically generated

Básicamente el valor de **treeProducer** es un objeto que contiene la estructura como resultado de la actualización en base de alguna base de datos del organigrama antes mencionado, pero ya con una estructura que favorece el recorrido de la información, ahorrando tiempos a la hora de consultar al coolaborador.

El consumidor actúa como escucha de los mensajes que se publican en el *topic*por lo que cada vez que se manda un nuevo mensaje este inmediatamente lo obtiene y sobrescribe la información en un fichero que se tiene en la raíz del proyecto.

Graphical user interface, text

Description automatically generated

Para así acceder a la información directamente a un fichero y evitar la consulta a alguna API o base de datos.

## LEVANTAR NODO DE KAFKA

Primero debemos de descargar el binario de Kafka en <https://kafka.apache.org>

A partir de aquí hemos arrancado primero Zookeeper y después Kafka en modo 'standalone'(Es decir, que trabajaremos con un cluster de un solo nodo):

Una vez descargados y descomprimido con los siguientes comandos levantamos zookeper.

<KAFKA\_HOME>/bin/zookeeper-server-start.sh config/zookeeper.properties

En otra terminal con los siguientes comandos levantamos kafka

<KAFKA\_HOME>/bin/kafka-server-start.sh config/server.properties

### Creamos nuestros dos topics

Ahora toca crear nuestros dos topics(Uno de entrada y otro de salida):

<KAFKA\_HOME>/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test-stream-in

<KAFKA\_HOME>/bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test-stream-out

Después hemos comprobado que realmente ambos topics se han generado correctamente:

<KAFKA\_HOME>/bin/kafka-topics.sh --list --bootstrap-server localhost:9092