UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Implementación y optimización de un servidor Elasticsearch para la búsqueda de información y su posterior consumo mediante una aplicación frontend usando un framework JavaScript.

TRABAJO DE TITULACIÓN.

AUTOR: Zuñiga Pullaguari, Borys Alexander

DIRECTOR: Elizalde Solano, René Rolando, Mgs

LOJA – ECUADOR

2018

2018

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Magíster.

René Rolando Elizalde Solano.

DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: **Implementación y optimización de un servidor Elasticsearch para la búsqueda de información y su posterior consumo mediante una aplicación frontend usando un framework JavaScript** realizado por **Zuñiga Pullaguari Borys Alexander**, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, marzo de 2018

f) . . . . . .

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

"Yo **Zuñiga Pullaguari Borys Alexander** declaro ser autor del presente trabajo de titulación; Implementación y optimización de un servidor Elasticsearch para la búsqueda de información y su posterior consumo mediante una aplicación frontend usando un framework JavaScript, de la Titulación Sistemas Informáticos y Computación, siendo René Rolando Elizalde Solano director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad Intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad".

f .....

Autor: Zuñiga Pullaguari Borys Alexander

Cédula: 1900596535

DEDICATORIA

Dedico este proyecto a mi madre, Norma Bertha Pullaguari Japón
ya que gracias a su ejemplo, constancia y trabajo ha sido motivación
para que culmine mis estudios universitarios.

Quiero dedicárselo también a la memoria de mi padre,
Mario Alejandro Zuñiga Sarango
por estar siempre presente en cada paso de mi diario vivir y,
a mis hermanas Gabriela y Karmi.

AGRADECIMIENTO

Agradezco desde el centro de mi corazón a:

Dios todopoderoso por darme sabiduría y entendimiento

para culminar con éxito este proyecto.

A mi madrecita que tanto amo,

por brindarme apoyo no sólo económico

sino también moral a lo largo de este camino universitario.

Al magister René Elizalde por su paciencia y aporte

durante la realización del trabajo de titulación.

A mi hermana Karmi y Gabriela Zuñiga por apoyarme económicamente

en la fase final de mi carrera universitaria.

A mi primo, hermano y amigo Luis Jaramillo por ayudarme

en momentos cruciales de este proyecto y aportarme ese grado

de motivación que toda persona necesita.

A todos, **¡MUCHAS GRACIAS!**

Atentamente

Borys Zuñiga

ÍNDICE DE CONTENIDOS

RESUMEN

En este trabajo se detalla la instalación, implementación y optimización de un servidor (clúster) Elasticsearch en un entorno de pruebas, donde se indica cada paso a seguir, desde la infraestructura utilizada, la topología de red usada, así como también, la optimización del servidor para un conjunto de datos, en el cual se muestra cómo hacer un correcto modelado y posterior mapeo. Además, se utiliza de manera general para la administración del clúster la herramienta de gestión y configuración denominada Ansible.

Además, se realiza el consumo del servidor Elasticsearch con una aplicación web frontend, desarrollada con JavaScript como lenguaje de programación base del lado del servidor, construida a través del entorno de ejecución Node.js. Con la mencionada aplicación, se hace uso de la librería elasticsearch.js, la cual permite realizar desde la conexión al servidor, así como también las diferentes operaciones, ya sean estas de mapeo, indexación, lectura, eliminación o edición.

Y para finalizar, se realizan pruebas de rendimiento, las cuales permiten hacer un breve análisis y, observar el comportamiento del clúster al momento de indexar y realizar operaciones de lectura.

PALABRAS CLAVES: elasticsearch, instalación, implementación, optimización, datos, manual, ansible, aplicación, Javascript, Nodejs, elasticsearchjs, pruebas, jmeter.

ABSTRACT

This paper details the installation, implementation and optimization of a Elasticsearch server (cluster) in a test environment, where each step to be followed is indicated, from the infrastructure used, the network topology used, as well as the optimization of the server for a data set, which shows how to do a proper modeling and subsequent mapping. In addition, the management and configuration tool called Ansible is generally used for cluster administration.

In addition, the consumption of the Elasticsearch server is performed with a frontend web application, developed with JavaScript as the base programming language on the server side, built through the Node.js execution environment. With the aforementioned application, elasticsearch.js library is used, which allows to perform from the connection to the server, as well as the different operations, whether they are mapping, indexing, reading, deleting or editing.

And finally, performance tests are performed, which allow a brief analysis and observing the behavior of the cluster when indexing and performing read operations.

KEYWORDS: elasticsearch, installation, implementation, optimization, data, manual, ansible, application, Javascript, Nodejs, elasticsearchjs, tests, jmeter

INTRODUCCIÓN

En el presente trabajo de fin de titulación se propone una guía de cómo implementar, configurar y optimizar un servidor Elasticsearch para un conjunto de datos, la idea nace debido a que, en la actualidad prácticamente todo se está conectando a la red global (internet), lo que provoca que se generen datos cada vez más exorbitantes, que, vistos de una manera general, carecen de valor o significado. Por tal motivo, se cree oportuno la realización de este proyecto el cual ayudará a futuros proyectos inherentes a datos masivos.

El objeto de este proyecto es la implementación, configuración y optimización de un servidor Elasticsearch y su posterior consumo mediante una aplicación frontend; esto servirá de base para plasmar una guía que describa las configuraciones realizadas en el servidor. Así mismo, se deberá realizar la optimización de un conjunto de datos que serán almacenados en el servidor Elasticsearch y construir una aplicación web usando JavaScript para consumir los datos alojados.

La metodología usada en el proyecto fue, en principio, la instalación de la infraestructura de red, luego, la instalación de Elasticsearch como tal y configuración del servidor teniendo en cuenta los componentes disponibles; luego, se modeló y mapeó un conjunto de datos (INEC), que posteriormente fueron almacenados en el servidor, dichos datos fueron consumidos a través de la aplicación realizada, para finalmente, realizar pruebas.

Para el desarrollo del proyecto se lo ha estructurado en cuatro capítulos, los cuales están ligados uno del otro. En el primero, se detallan conceptos fundamentales que muestran en teoría como es el funcionamiento de Elasticsearch, trata temas relacionados al modelado y mapeo de datos, además, se abarca contenidos que están relacionados directamente con el desarrollo de la aplicación, siendo parte fundamental para enriquecer el léxico y lograr comprender los demás capítulos. El segundo, describe una guía detallada de la implementación, instalación y configuración del servidor Elasticsearch, incluido la utilización de Ansible. En el tercer capítulo, se aborda la solución de la aplicación web construida, la cual sirvió para almacenar en el servidor datos de forma masiva y a su vez, para consumir los mismos, además, se explica las cuestiones más relevantes del desarrollo. Finalmente, el cuarto capítulo, se enfoca en la realización de pruebas las cuales sirvieron para analizar el rendimiento del servidor Elasticsearch.

CAPÍTULO I

MARCO TEÓRICO

1.1. Introducción

El presente capítulo se centrará en repasar conceptos y definiciones, los cuales ayudaran para el posterior desarrollo del proyecto. Es por eso que se hace hincapié para que se revisen, y así lograr el entendimiento en post de continuar con los siguientes capítulos.

Dentro de los principales temas a tratar en este apartado, van desde, definiciones de Elasticsearch las cuales permitirán conocer en teoría como es la naturaleza del software, por otro lado, se conocerá acerca de cómo es el modelado en una base de datos NoSQL orientada a documentos, esto será la base para modelar los datos que serán almacenados en el servidor. Además, se repasan conceptos y definiciones de manera breve, relacionados al desarrollo de software, big data, computación distribuida, Ansible y Apache JMeter, los cuales están inherentes a este proyecto.

1.1.1. ¿Qué es elasticsearch?

Hace algún tiempo atrás la búsqueda de texto completo era un término que muy pocos ingenieros informáticos conocían, era común utilizar las bases de datos SQL para realizar las búsquedas, pero a medida que se adentraban en el tema se podían dar cuenta de las limitantes que tiene este enfoque. Kuc & Rogozinski (2014) mencionan algunas de ellas: “La falta de escalabilidad, no hay suficiente flexibilidad y falta de análisis del lenguaje”. Por tales motivos, se creó Elasticsearch, que es un proyecto de servidor de búsqueda, iniciado por Shay Banon y publicado en febrero de 2010 como código abierto. Está basado en Apache Lucene, proporcionando así a sus usuarios de un motor de búsqueda de texto completo, distribuido y con capacidades de tenencia múltiple. Además, es escalable, muy rápido y proporciona un análisis para varios idiomas.

Desde que su autor publicara su primera versión, Elasticsearch se ha convertido en un proyecto de mucha relevancia, siendo así fundamental en el campo de búsqueda, soluciones de análisis de datos, y otros proyectos de aplicaciones de búsqueda. Por otro lado, debido a su naturaleza distribuida y tiempo real, muchos lo utilizan como una tienda o almacén de documentos.

1.1.2. Conceptos de la arquitectura de datos de elasticsearch.

1.1.2.1. Índice (index).

“Es el lugar lógico donde Elasticsearch almacena datos lógicos, de modo que puede ser dividido en fragmentos más pequeños. Utilizando la analogía de una base de datos relacional, un índice es como una tabla” (Kuc & Rogozinski, 2014, p.12).

1.1.2.2. Documento (document).

Kuc & Rogozinski (2014), afirman que:

> Utilizando la analogía de una base de datos relacional, un documento, viene siendo como una fila de datos en una tabla, por supuesto, con diferente estructura.
>
> Un documento consiste en fields (campos), y cada campo puede ocurrir varias veces en un documento (cada campo es llamado multivalor). Cada campo tiene un tipo que puede ser: texto, número, fecha, etcétera. A diferencia de las bases de datos relacionales, los documentos no necesitan tener una estructura fija, cada documento puede tener un conjunto diferente de campos, y, además, los campos no tienen que ser conocidos durante el desarrollo de aplicaciones. Desde la perspectiva del cliente un documento es un objeto JSON. Cada documento se almacena en un índice y tiene su propio identificador único (que puede ser generado automáticamente por Elasticsearch) y tipo de documento. Un documento debe tener un identificador único en relación con el tipo de documento. Esto significa que, en un índice único, dos documentos pueden tener el mismo identificador único si no son del mismo tipo. (p.13)

1.1.2.3. Tipo de documento (document type).

Kuc & Rogozinski (2014), afirman que:

> En Elasticsearch, un índice puede almacenar muchos objetos con diferentes propósitos. Por ejemplo, una aplicación de blog puede almacenar artículos y comentarios. El tipo de documento nos permite diferenciar fácilmente los objetos en un índice único. Cada documento puede tener una estructura diferente, pero en implementaciones del mundo real, dividir documentos en tipos ayuda significativamente en la manipulación de datos. (p.13)

1.1.2.4. Mapping.

Kuc & Rogozinski (2014), afirman que:

> Todos los campos del documento deben ser analizados adecuadamente según su tipo. Por ejemplo, se requiere diferentes análisis para los campos numéricos (los números no deben ser ordenados por orden alfabético) y para el texto de páginas web (por ejemplo, el primer paso requerido seria omitir las etiquetas HTML ya que es información inútil,

ruido). Elasticsearch almacena información acerca de los campos en el mapeo. Cada tipo de documento tiene su propio mapping, aunque explícitamente no lo definimos (p.13).

1.1.3. Conceptos de elasticsearch.

1.1.3.1. Nodo y clúster (node and cluster).

Elasticsearch puede funcionar como un servidor de búsqueda único independiente. Sin embargo, para ser capaces de procesar grandes conjuntos de datos y lograr tolerancia a fallos y alta disponibilidad, Elasticsearch se puede ejecutar en muchos servidores cooperantes. Colectivamente, estos servidores se llaman clúster, y cada servidor formando es un nodo. (Kuc & Rogozinski, 2014, p.14)

1.1.3.2. Fragmento (shard).

Los datos pueden dividirse en partes más pequeñas llamadas fragmentos (donde cada fragmento es un índice separado de Apache Lucene). Cada fragmento puede ser colocado en un servidor diferente, y, por lo tanto, los datos se pueden propagar entre los nodos del clúster. Cuando se consulta un índice que se construye con múltiples fragmentos, Elasticsearch envía la consulta a cada fragmento relevante y combina el resultado de tal manera que la aplicación no sabe nada acerca de los fragmentos. Además, tener varios fragmentos puede acelerar la indexación de direcciones. (Kuc & Rogozinski, 2014, p.14)

1.1.3.3. Réplica.

Para aumentar el rendimiento de la consulta o para lograr alta disponibilidad, se pueden utilizar réplicas de fragmento. Una réplica es sólo una copia exacta de los fragmentos y cada fragmento puede tener cero o más réplicas. En otras palabras, Elasticsearch puede tener muchos fragmentos idénticos y uno de ellos se elige automáticamente como un lugar donde se dirigen las operaciones que modifican el índice. Este fragmento especial se llama un fragmento primario, y los otros se llaman fragmentos de réplica. Cuando el fragmento primario se pierde (por ejemplo, un servidor con que los datos del fragmento no están disponibles), el grupo promoverá la réplica que el nuevo fragmento primario. (Kuc & Rogozinski, 2014, p.14)

1.1.3.4. Puerta de enlace (gateway).

“Elasticsearch se encarga de muchos nodos. El estado del clúster se realiza por la puerta de enlace. Por defecto, cada nodo dispone de esta información almacenada localmente, que se sincroniza entre nodos” (Kuc & Rogozinski, 2014, p.15).

1.1.3.5. Indexación y búsqueda (indexing and searching).

Es aquí, donde nos estaremos haciendo la pregunta ¿Cómo se puede prácticamente unir todos los índices, fragmentos y réplicas en un entorno único? En teoría, debería ser muy difícil obtener datos del clúster cuando se tiene que saber dónde está el documento, en qué servidor y en que fragmento. Más difícil es buscar cuando una consulta puede devolver documentos de diferentes fragmentos a diferentes nodos en el cluster entero. De hecho, se trata de un problema complicado, pero afortunadamente, no debe preocupar esto, ya que es manejado automáticamente por el mismo Elasticsearch. (Kuc & Rogozinski, 2014, p.15)

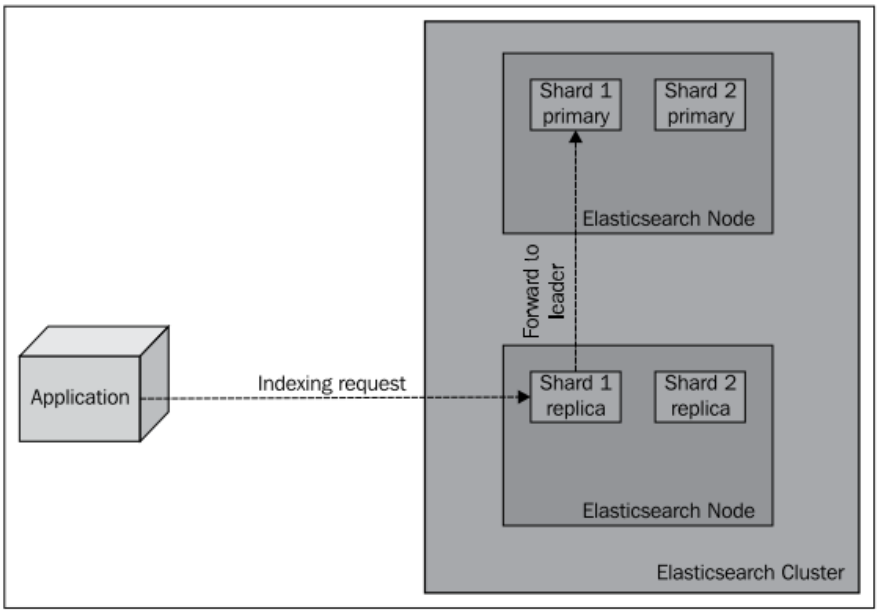


Figura 1. Indexación y búsqueda en Elasticsearch

Fuente: (Kuc & Rogozinski, 2014)

Elaboración: (Kuc & Rogozinski, 2014)

1.1.4. Comunicación en elasticsearch.

Para la comunicación con el servidor Elasticsearch puede hacerlo con varios protocolos tales como: HTTP, Native, Thrift, etc. Así mismo, Elasticsearch fue diseñado para ser usado como un servidor RESTful, por lo que utiliza el protocolo HTTP, que generalmente utiliza el puerto 9200 o superiores. La siguiente tabla muestra los principales protocolos según Paro (2015):

Tabla 1. Principales Tipos de Protocolo

Protocolo	Ventajas	Desventajas	Tipo
HTTP	• Usado frecuentemente. • API segura y tiene compatibilidad general para diferentes versiones de ES, aunque se sugiere JSON.	• Sobrecarga HTTP.	Texto
Native	• Capa de red rápida. • Programático. • Mejor para operaciones de indexación masiva.	• Si el API cambia, puede dañar las aplicaciones. • Requiere la misma versión del servidor Elasticsearch. • Sólo en JVM.	Binario
Thrift	• Similar a HTTP.	• Relacionado para el plugin Thrift.	Binario

Fuente: Paro (2015)

Elaboración: Borys Zuñiga

1.1.5. Tipos de nodos.

La configuración por defecto de Elasticsearch establece a un nodo como master y datos al mismo tiempo, pero para una configuración más avanzada del clúster se debe configurar los parámetros *node.master* y *node.data*, dentro del archivo "*elasticsearch.yml".* La siguiente tabla muestra los tipos de nodos según Paro (2015):

Tabla 2. Tipos de Nodo

Parámetros de Configuración	Descripción del nodo
node.master: true node.data: true	Esta es la configuración por defecto. Puede ser un nodo master y contener datos.
node.master: false node.data: true	Este nodo jamás podrá ser un nodo master y únicamente almacenará datos.
node.master: true node.data: false	Este nodo únicamente sirve como master, no almacena ningún dato y tiene los recursos libres. Es quien coordina el clúster.

node.master: false node.data: false	Este nodo actúa como equilibrador de carga de búsqueda (extrae datos de nodos, agrega resultados, etc.).

Fuente: Paro (2015)

Elaboración: Borys Zuñiga

1.1.6. Plugins en elasticsearch.

Una característica muy importante en Elasticsearch es que permite ampliar sus funcionalidades a través de la instalación de plugins, esto hace que las bondades de este potente software crezcan en los diversos ámbitos para el que fue construido. Existen dos tipos de plugins dentro de Elasticsearch:

Tabla 3. Tipos de Plugins

Tipo	Descripción
Plugins de Sitio	"Estos se utilizan para servir contenido estático en sus puntos de entrada. Se utilizan principalmente para crear una aplicación de gestión para la supervisión y administración de un clúster" (Paro, 2015, p.35).
Plugins Nativos	Paro (2015) afirma que: son los archivos .jar que contienen el código de la aplicación. Se utilizan para: • Rivers (plugins que permiten importar datos de otras fuentes o SGBD). • Scripting Language Engines (JavaScript, Python, Scala, y Ruby). • Puntos de entrada REST. • Soporte a nuevos protocolos (Thrift, memcache y así sucesivamente). • Soporte a nuevos almacenadores (Hadoop).

Fuente: (Paro, 2015)

Elaboración: Borys Zuñiga

1.1.7. Mapeo (mapping).

El Mapeo es un concepto muy importante en Elasticsearch debido a que es uno de los aspectos más importantes a la hora de utilizar y tratar de aprovechar este motor de búsqueda en la mayoría

de las bondades que ofrece. Según Paro (2015): “El Mapping define cómo el motor de búsqueda debe procesar un documento”. En otras palabras, el mencionado autor expresa que en el mapeo se define como vamos a indexar un documento, es decir que en este punto se precisa; el nombre de los campos, el tipo de dato de los campos, como se va a manejar los tipos de dato (integer, double, date, objeto, arreglo, etc.), así mismo, se puede definir la relación que existe entre los diferentes tipos de documentos, como manipular los campos que corresponden a metadatos, la relevancia de cada uno de los campos, etc. Dentro de Elasticsearch se puede especificar dos tipos de mapeo:

1.1.7.1. Mapeo dinámico.

Según la documentación oficial de Elasticsearch, en este tipo de mapeo no se necesita definir los campos, ni los tipos de mapeo antes de ser usado. Gracias al mapeo dinámico, los campos y tipos son agregados de forma automática, al momento de indexar un documento. Por ejemplo:

Tabla 4. Ejemplo Mapeo Dinámico

Se crea un índice *$ curl -XPUT* http://127.0.0.1:9200/test # {acknowledged":true} Se indexa un documento dinámicamente *$ curl -XPUT http://127.0.0.1:9200/test/mytype/1 -d '{"name":"Alex", "age":25}'* # {"ok":true,"_index":"test","_type":"mytype","_id":"1","_ version":1} Se hace una consulta del mapeo del tipo creado (mytype) $ curl –XGET http://127.0.0.1:9200/test/mytype/_ mapping?pretty=true

{
"mytype": {
"properties": {
"age": {
"type": "long"
},
"name": {
"type": "string"
}
}
}

}

Fuente: (Paro, 2015)

Elaboración: Borys Zuñiga

Es importante destacar que se puede configurar las reglas del mapeo dinámico para personalizar el mapeo que se utiliza para nuevos tipos y nuevos campos.

1.1.7.2. Mapeo estático.

Por otro lado en el sitio web Logz, su autor Daniel Berman (2016) afirma acerca de este tipo de mapeo lo siguiente: “en un escenario normal, sabemos bien de antemano que tipo de datos se almacena en el documento, por lo que fácilmente podemos definir los campos y sus tipos al crear el índice”. Tomando como referencia las aseveraciones del autor antes mencionado, se puede decir que este tipo de mapeo es el más indicado antes de indexar información en Elasticsearch, si el objetivo es aprovechar a Elasticsearch en su máximo potencial. Por ejemplo:

Tabla 5. Ejemplo Mapeo Estático

Se crea un índice *$ curl -XPUT http://127.0.0.1:9200/test* # {acknowledged":true} Mapeamos la estructura del documento estáticamente *$ curl -XPUT http://127.0.0.1:9200/test/mytype/1 -d '{"mappings": { "mytype": {"properties": { "age": { "type": "integer"}, "name": { "type": "string" }}}}}'* # {"ok":true,"_index":"test","_type":"mytype","_id":"1","_ version":1} Se hace una consulta del mapeo del tipo creado (mytype) $ curl –XGET http://127.0.0.1:9200/test/mytype/_ mapping?pretty=true

{
"mytype": {
"properties": {
"age": {
"type": "integer"
},
"name": {
"type": "string"
}

```
  }
 }
}
```

Fuente: (Paro, 2015)

Elaboración: Borys Zuñiga

1.1.8. Búsquedas en elasticsearch.

Elasticsearch como tal, es un motor de búsqueda, por lo que su naturaleza principal es procesar consultas y dar resultados. A continuación, se abordarán los diferentes tipos de búsquedas que se pueden realizar en Elasticsearch, y se podrá dar cuenta que Elasticsearch no únicamente busca coincidencias de documentos, sino también, es capaz de calcular información adicional necesaria para mejorar la calidad de búsqueda y, por ende, los resultados serán más acordes a lo que el usuario está buscando.

Antes de empezar es importante recalcar que Elasticsearch utiliza la biblioteca de Apache Lucene, por lo que, es el encargado de realizar las búsquedas de texto completo; y así mismo, proporciona análisis para varios idiomas.

1.1.8.1. Conceptos de apache lucene.

Previo al proceso de análisis y los tipos de búsqueda, empezaremos por dar a conocer los conceptos de esta importante biblioteca según Kuc & Rogozinski (2014):

- **Documento (Document):** es el principal soporte de datos utilizados durante la indexación y búsqueda, que consta de uno o más campos que contienen los datos que se almacenan y se obtiene de Lucene.
- **Campo (Field):** esto es una sección del documento que está formado por dos partes: el nombre y el valor.
- **Término (Term):** esto es una unidad de búsqueda que representa una palabra del texto.
- **Token:** esto es una ocurrencia de un término en el texto del campo. Consiste en el término del texto, inicio y final de una compensación, y un tipo.

1.1.8.2. Proceso de indexado de datos.

Proceso que consiste en guardar datos y tener disponible los documentos para su búsqueda. Lo que Elasticsearch hace al momento de indexar los datos es analizarlos mediante un analizador de la biblioteca de Apache Lucene que descompone un texto en términos individuales, cada uno

de ellos son normalizados en una forma estándar, y posteriormente se crea un índice invertido sobre el que se realizaran las búsquedas.

Índice Invertido (Inverted Index): Kuc & Rogozinski (2014) afirman: “Un índice invertido es una estructura de datos que asigna los términos en el índice a los documentos y no al revés como la base de datos relacional en sus tablas” (p.8). En otras palabras, un índice invertido es una tabla donde se guarda cada uno de los términos normalizados que se van a almacenar y en que documentos aparecen.

A continuación, se presenta un ejemplo de un índice invertido propuesto por Kuc & Rogozinski, (2014):

Suponiendo que se indexan los siguientes documentos:

- Elasticsearch Server 1.0 (documento 1).
- Mastering Elasticsearch (documento 2).
- Apache Solr 4 Cookbook (documento 3).

Por lo tanto, la tabla del índice invertido (de manera simplificada) puede verse como la siguiente:

Tabla 6. Ejemplo Índice Invertido

Término	Conteo (Count)	Documento
1.0	1	<1>
4	1	<3>
Apache	1	<3>
Cookbook	1	<3>
Elasticsearch	2	<1>, <2>
Mastering	1	<2>
Server	1	<1>
Solr	1	<3>

Fuente: (Kuc & Rogozinski, 2014)

Elaboración: Borys Zuñiga

Cada término señala al número de documentos en los cuales está presente. Esto permite una búsqueda muy eficiente y rápida, como las consultas basadas en el término. Además de esto, cada término tiene un número conectado a él, *count*, diciendo a Lucene con qué frecuencia el término ocurre. Por supuesto, esta tabla es netamente con fines explicativos acerca de cómo son

almacenados los datos, Lucene maneja un índice invertido mucho más complejo y avanzado, en los que pueden incluir valores del documento, vectores del término, etc.

1.1.8.3. Estructura de una petición de búsqueda.

Gheorghe, Hinman, & Russo (2016) afirman que:

> Las peticiones de búsqueda de Elasticsearch son peticiones basadas en documentos JSON o peticiones basadas en URL. Las solicitudes se envían al servidor y, debido a que todas las solicitudes de búsqueda siguen el mismo formato, es útil comprender los componentes que puede cambiar para cada solicitud de búsqueda.

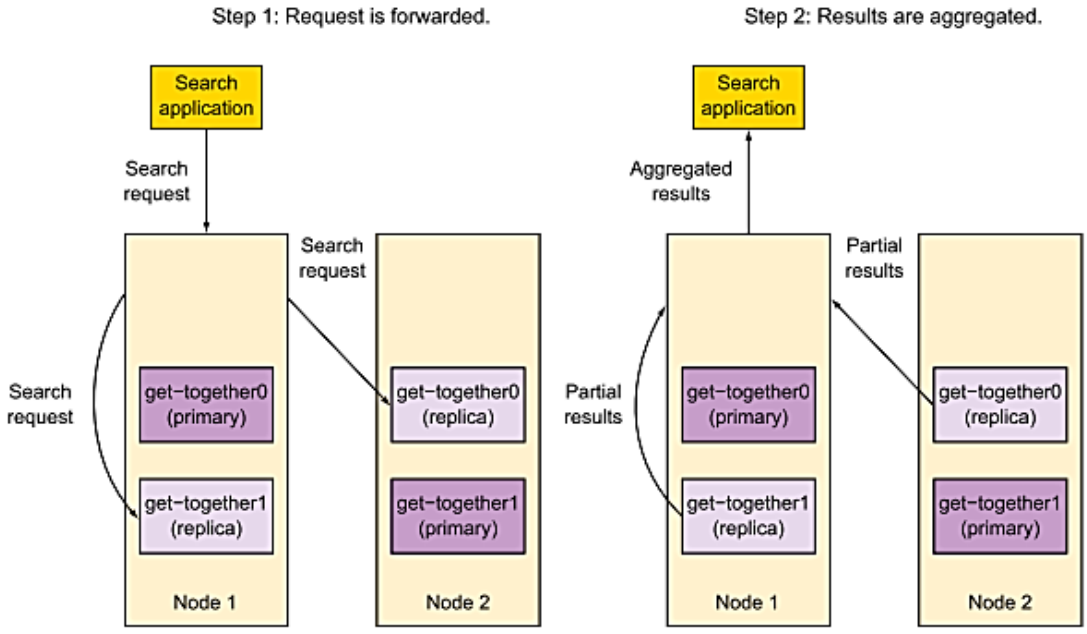


Figura 2. Cómo se encamina una solicitud de búsqueda; El índice consta de dos fragmentos y una réplica por fragmento. Después de localizar y anotar los documentos, sólo se obtienen los 10 primeros.

Fuente: (Gheorghe, Hinman, & Russo, 2016)

Elaboración: (Gheorghe, Hinman, & Russo, 2016)

1.1.8.4. Especificación de un ámbito de búsqueda.

Todas las solicitudes de búsqueda REST utilizan el punto final *_search* y puede ser una solicitud GET o POST. Se puede buscar toda la información de un clúster en una sola petición de búsqueda o se puede limitar especificando el nombre del índice y tipo en la URL de solicitud búsqueda.

Las siguientes URL de búsqueda son ejemplos que limitan el alcance de las búsquedas:

Tabla 7. Ejemplos URL de búsqueda limitada

URL de Búsqueda	Acción
# curl 'localhost:9200/_search' -d '...'	Busca en todo el clúster
# curl 'localhost:9200/get-together/_search' -d '...'	Busca en el índice *get-together*
# curl 'localhost:9200/get-together/event/_search' -d '...'	Busca el tipo *event* en el índice *get-together*
# curl 'localhost:9200/_all/event/_search' -d '...'	Busca todos los tipos de *event* en todos los índices
# curl 'localhost:9200/*/event/_search' -d '...'	Busca todos los tipos de event en todos los índices
# curl 'localhost:9200/get-together,other/event,group/_search' -d '...'	Busca los tipos *event* y *group* en los índices *get-together* y o*ther.*
# curl 'localhost:9200/+get-toge*,-get-together/_search' -d '...'	Busca todos los índices que empiezan con *get-toge* pero no los índices *get-together.*

Fuente: (Gheorghe, Hinman, & Russo, 2016)

Elaboración: Borys Zuñiga

1.1.8.5. Componentes básicos de una petición de búsqueda.

Luego de seleccionar los índices y tipos en los que se va a buscar (punto anterior), se debe configurar los componentes más importantes de la petición de búsqueda. Estos componentes son los encargados de limitar la cantidad de documentos a devolver, seleccionar los documentos que concuerdan con los criterios de búsqueda y descartar los que no se desean. Gheorghe, Hinman, & Russo (2016) los detallan de la siguiente manera:

- **query:** El componente más importante para la solicitud de búsqueda, esta parte calcula los mejores documentos a devolver basados en una puntuación, así como los documentos que no desea devolver.
- **size:** representa la cantidad de documentos a retornar.
- **from:** junto con el tamaño (*size*), *from* se utiliza para hacer la paginación.
- **_source:** especifica cómo se devuelve el campo *_source*. El valor predeterminado es devolver el campo *_source* completo. Al configurar *_source*, se filtran los campos que se

devuelven. Esto se usa si los documentos indexados son grandes y no se necesita el contenido completo en el resultado.

- **sort:** la clasificación por defecto se basa en la puntuación de un documento. Si usted no se preocupa acerca de la puntuación o espera un montón de documentos con la misma puntuación, agregar una clase le ayuda a controlar qué documentos se ha devuelto.

1.1.8.6. Tipos de peticiones de búsqueda.

A continuación se muestran dos tipos de peticiones de búsqueda según Gheorghe, Hinman, & Russo (2016):

- **Basadas en URL:** pretende ser útil para peticiones rápidas basadas en *curl*. No todas las funciones de búsqueda se exponen mediante la búsqueda basada en URL. Ejemplos:

Tabla 8. Ejemplos Peticiones basadas en URL

Petición de coincidencia de todos los documentos con *size* y *from* enviado como parámetros en la URL.
curl 'localhost:9200/get-together/_search?from=10&size=10'
Solicitud de coincidencia de todos los documentos, pero devolviendo por defecto los 10 primeros de todos los resultados ordenados por fecha en orden ascendente.
curl 'localhost:9200/get-together/_search?sort=date:asc'
Solicitud de coincidencia de todos los documentos, pero por defecto 10 de todos los resultados ordenados por fecha en orden ascendente. Se desea sólo dos campos en la respuesta: título y fecha.
curl 'localhost:9200/get-together/_search?sort=date:asc&_source=title,date'

Fuente: (Gheorghe, Hinman, & Russo, 2016)

Elaboración: Borys Zuñiga

- **Basadas en el cuerpo de la solicitud:** Al ejecutar búsquedas más avanzadas, el uso de búsquedas basadas en el cuerpo de la solicitud le ofrece más flexibilidad y más opciones. Incluso cuando se utilizan búsquedas basadas en el cuerpo de solicitud, algunos de los componentes también se pueden proporcionar en la URL. Ejemplos:

Tabla 9. Ejemplos de peticiones basadas en el cuerpo de la solicitud

Busca la segunda página del índice cuando todos los documentos coinciden. Esto equivale al primer ejemplo del punto anterior.

```
% curl 'localhost:9200/get-together/_search' -d '{
  "query": {
    "match_all": {}
  },
  "from": 10,
  "size": 10
}'
```

Returns results starting from the 10th result

Returns a total of max 10 results

Devuelve los campos nombre y fecha de cada grupo coincidente.

```
% curl 'localhost:9200/get-together/_search' -d '{
  "query": {
    "match_all": {}
  },
  "_source": ["name", "date"]
}'
```

Returns the name and date fields with the search response

Devuelve los resultados ordenados primero por fecha de creación, empezando por el más antiguo; luego por el nombre del grupo de encuentro, en orden alfabético inverso; Y finalmente por el *_score* del resultado.

```
% curl 'localhost:9200/get-together/_search' -d '{
  "query": {
    "match_all": {}
  },
  "sort": [
    {"created_on": "asc"},
    {"name": "desc"},
    "_score"
  ]
}'
```

Sorts first by the creation date, starting from the oldest to newest

Then sorts by name of the group, in reverse alphabetical order

Finally, sorts by the relevancy of the result (its _score)

Fuente: (Gheorghe, Hinman, & Russo, 2016)

Elaboración: Borys Zuñiga

1.1.8.7. Tipos de búsquedas.

Según Kuc & Rogozinski (2014):

- **query_then_fetch:** En el primer paso, se ejecuta la consulta para obtener la información necesaria para ordenar y clasificar los documentos. Este paso se ejecuta contra todos los fragmentos. Entonces, sólo los fragmentos pertinentes se preguntan por el contenido de los documentos. A diferencia de *query_and_fetch*, el número máximo de resultados devueltos por este tipo de consulta será igual al parámetro *size*. Este es el tipo de búsqueda que utiliza por defecto si no se ha proporcionado ningún tipo de búsqueda con la consulta, y este es el tipo de consulta que se ha descrito anteriormente.
- **query_and_fetch:** Generalmente se trata de la aplicación del tipo de búsqueda más rápido y más simple. La consulta se ejecuta contra todos los fragmentos (por supuesto, se le preguntó solamente una sola réplica de un determinado fragmento primario) en paralelo y todos los fragmentos que devuelve el número de resultados igual al valor del parámetro tamaño. El número máximo de documentos devueltos será igual al valor de tamaño (*size*) multiplicado por el número de fragmentos.
- **dfs_query_and_fetch:** Esto es similar al tipo de búsqueda *query_and_fetch*, pero que contiene una fase adicional en comparación con la *query_and_fetch*. La parte adicional es la fase inicial de la consulta que se ejecuta para calcular frecuencias de término distribuido para dejar resultados más precisos de los documentos devueltos y así más resultados relevantes de la consulta.
- **dfs_query_then_fetch:** Al igual que con el tipo de búsqueda anterior dfs_query_and_fetch, el tipo de búsqueda dfs_query_then_fetch es similar a su contraparte: query_then_fetch. Sin embargo, contiene una fase adicional en comparación con query_then_fetch, al igual que dfs_query_and_fetch.
- **count:** Este es un tipo de búsqueda especial que sólo devuelve el número de documentos que coincidió con la consulta. Si usted sólo necesita obtener el número de resultados, pero no se interesa por los documentos, debe utilizar este tipo de búsqueda.
- **scan:** Utilice únicamente si la consulta va a devolver una gran cantidad de resultados. Se diferencia un poco de las habituales consultas porque después de enviar la primera solicitud, Elasticsearch responde con un identificador de desplazamiento, similar a un cursor en bases de datos relacionales. Todas las consultas deben ejecutarse en el punto final REST de *_search/scroll* y es necesario enviar el identificador de desplazamiento (*scroll*) devuelto en el cuerpo de la solicitud.

1.1.8.8. Búsquedas y consultas.

Una vez que se ha mapeado los índices en Elasticsearch, y post al indexado de los datos, éstos se pueden buscar. En este apartado se destacará los más principales tipos de consultas y filtros. Elasticsearch le permite utilizar Domain Specific Language (DSL), un lenguaje de sintaxis diseñado para la búsqueda, que cubre todas las necesidades comunes, desde una consulta estándar hasta un filtrado de Geoshape complejo.

A continuación, se muestra una consulta:

Tabla 10. Ejemplo de consulta en Elasticsearch

Desde la línea de comandos, se ejecuta lo siguiente: # curl -XGET 'http://127.0.0.1:9200/test-index/test-type/_search' -d '{"query":{"match_all":{}}}' En este caso, se ha utilizado una consulta *match_all*, lo que significa que todos los documentos serán devueltos.

El resultado de la consulta seria lo siguiente:
{
"took": 0,
"timed_out": false,
"_shards": {
"total": 5,
"successful": 5,
"failed": 0
},
"hits": {
"total": 3,
"max_score": 1.0,
"hits": [
{
"_index": "test-index",
"_type": "test-type",
"_id": "1",
"_score": 1.0,
"_source": {
"position": 1,

```
        "parsedtext": "Joe Testere nice guy",
        "name": "Joe Tester",
        "uuid": "11111"
      }
    },
    {
      "_index": "test-index",
      "_type": "test-type",
      "_id": "2",
      "_score": 1.0,
      "_source": {
        "position": 2,
        "parsedtext": "Bill Testere nice guy",
        "name": "Bill Baloney",
        "uuid": "22222"
      }
    },
    {
      "_index": "test-index",
      "_type": "test-type",
      "_id": "3",
      "_score": 1.0,
      "_source": {
        "position": 3,
        "parsedtext": "Bill is not\n nice guy",
        "name": "Bill Clinton",
        "uuid": "33333"
      }
    }
  ]
 }
}
```


took: este es el tiempo, en milisegundos, requerido para ejecutar la consulta.

time_out: esto indica si se ha producido un tiempo de espera durante la búsqueda. Esto está relacionado con el parámetro timeout de la búsqueda. Si se produjo un tiempo de espera, obtendrá resultados parciales o no.

_**shards:** Este es el estado de los fragmentos, que se pueden dividir en los siguientes:

- **total:** éste es el número total de fragmentos (shards).
- **successful:** éste es el número de fragmentos en los que se realizó correctamente la consulta.
- **failed:** éste es el número de fragmentos en los que falló la consulta, porque se produjo algún error o excepción durante la consulta.

hits: esto representa los resultados y se compone de lo siguiente:

- **total:** éste es el número total de documentos que coinciden con la consulta.
- **max_score:** Este es el resultado de la coincidencia del primer documento. Normalmente esto es 1 en caso de no coincidir la puntuación se calcula, por ejemplo, para ordenar o filtrar.
- **hits:** esta es una lista de los documentos resultantes.

El documento resultante tiene muchos campos que siempre están disponibles y otros campos que dependen de los parámetros de búsqueda. Los siguientes son los campos más importantes:

_**index:** este es el índice que contiene el documento.

_**type:** este es el tipo del documento.

_**id:** este es el identificador del documento.

_**source:** esto contiene el origen (campos mapeados) del documento (por defecto se devuelven, pero se puede desactivar).

_**score:** este es el resultado de consulta del documento.

sort: Estos son los valores que se utilizan para ordenar, si los documentos están ordenados.

highlight: Estos son los segmentos destacados, si highlight fue solicitado.

fields: Esto indica que algunos campos pueden recuperarse sin la necesidad de recuperar todos los objetos de _source.

Fuente: (Paro, 2015)

Elaboración: Borys Zuñiga

1.2. Modelado de datos en elasticsearch

El modelado de datos en Elasticsearch, es de vital importancia porque es el punto en donde se analiza y define como van a estar estructurados los datos, así como también las relaciones que existen entre ellos, por supuesto, en un escenario en donde previamente se ha hecho un análisis de los diferentes casos de uso.

Un concepto clave a la hora de modelar una base de datos NoSQL orientada a documentos es la denormalización. Según (Akdoğan, 2015), “La denormalización es el proceso de optimizar el rendimiento de lectura de una base de datos al agregar datos redundantes.” Acotando lo que argumenta el mencionado autor, este proceso se enfoca en el almacenamiento de datos repetidos para obtener un beneficio de rendimiento especialmente a la hora de hacer consultas, al hacer esto se ahorra tiempo de respuesta al evitar tener que hacer los famosos “joins” para obtener información entre varias entidades, pudiendo obtener toda la información en una sola consulta.

Como es de conocimiento, Elasticsearch utiliza Apache Lucene para indexar y recuperar información, por lo que está libre de esquemas, lo que impide que se pueda usar los conceptos de algebra relacional que son aplicados para modelar las bases de datos relacionales. Por lo tanto, en este punto, se tomará como referencia ciertas pautas de algunos expertos, para modelar una base de datos NoSQL orientada a documentos y de esta manera dar respuesta a la gran interrogante ¿Cómo se van a almacenar los datos?

Para empezar a modelar los datos se deberá tratar cada entidad como un documento independiente que será representado en un objeto JSON.

Para un mejor entendimiento de algunos conceptos se lo hará mediante el siguiente ejemplo tomado de un blog web de (Krishna R., 2016):

Figura 3. Ejemplo de relaciones de una base de datos relacional

Fuente: (Krishna R., 2016)

Elaboración: (Krishna R., 2016)

En la imagen anterior se muestra las relaciones que existen para el modelado de una base de datos relacional, la mayoría está familiarizado con ellas. En este ejemplo, una persona puede tener varios registros de información de contacto y también varios registros de dirección, adicional a esto, se extrae un tipo de contacto, que podría ser doméstico o empresarial.

Para recuperar toda la información de una persona con sus respectivos contactos y direcciones se deben usar conexiones o “joins” entre tablas, de la siguiente manera:

```
SELECT p.FirstName, p.LastName, a.City, cd.Detail
FROM Person p
JOIN ContactDetail cd ON cd.PersonId = p.Id
JOIN ContactDetailType on cdt ON cdt.Id = cd.TypeId
JOIN Address a ON a.PersonId = p.Id
```

Figura 4. Ejemplo de consulta en una base de datos relacional

Fuente: (Krishna R., 2016)

Elaboración: (Krishna R., 2016)

Ahora, se pasa a demostrar cómo sería el modelado de los mismos datos, pero en una base de datos orientada a documentos.

1.2.1. Incrustación o datos embebidos.

```
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "addresses": [
        {
            "line1": "100 Some Street",
            "line2": "Unit 1",
            "city": "Seattle",
            "state": "WA",
            "zip": 98012
        }
    ],
    "contactDetails": [
        {"email": "thomas@andersen.com"},
        {"phone": "+1 555 555-5555", "extension": 5555}
    ]
}
```

Figura 5. Ejemplo de un documento en formato JSON con datos embebidos

Fuente: (Krishna R., 2016)

Elaboración: (Krishna R., 2016)

En la imagen anterior se muestra un único documento denormalizado del registro de “Persona” en el que se ha incrustado o embebido toda la información relacionada a una persona, como sus contactos y direcciones.

Según Krishna R., 2016 se deben incrustar los datos, cuando:

- Existen relaciones de inclusión entre entidades.
- Existen relaciones de uno a algunos entre entidades.
- Existen datos incrustados que cambian con poca frecuencia.
- Existen datos incrustados que no crecerán sin límites.
- Existen datos incrustados que se integran en los datos en un documento.

1.2.2. Datos de referencia.

```
Person document:
{
    "id": "1",
    "firstName": "Thomas",
    "lastName": "Andersen",
    "holdings": [
        { "numberHeld":  100, "stockId": 1},
        { "numberHeld":  50, "stockId": 2}
    ]
}

Stock documents:
{
    "id": "1",
    "symbol": "zaza",
    "open": 1,
    "high": 2,
    "low": 0.5,
    "vol": 11970000,
    "mkt-cap": 42000000,
    "pe": 5.89
},
{
    "id": "2",
    "symbol": "xcxc",
    "open": 89,
    "high": 93.24,
    "low": 88.87,
    "vol": 2970200,
    "mkt-cap": 1005000,
    "pe": 75.82
}
```

Figura 6. Ejemplo de varios documentos en formato JSON referenciando datos

Fuente: (Krishna R., 2016)

Elaboración: (Krishna R., 2016)

Ahora considere otro ejemplo, como se muestra en la imagen anterior, varios documentos que representan la cartera de acciones de una persona. En este caso se hace referencia al artículo comercial en la cartera, debido a que el artículo comercial cambia frecuentemente durante el día, el único documento que se debería modificar sería el documento de acciones único.

Según Krishna R., 2016 se deben referenciar los datos, cuando:

- Se realiza una representación de relaciones de uno a varios.

- Se realiza una representación de las relaciones de muchos a muchos.
- Los datos relacionados cambian con frecuencia.
- Puede cancelarse el límite de los datos de referencia.

1.3. Lenguajes de programación y herramientas de desarrollo de software

Los lenguajes de programación son fundamentales para desarrollar proyectos que involucran desarrollo de software. Así mismo, se expondrá a detalle las herramientas utilizadas para la programación que son fundamentales para minimizar y optimizar el esfuerzo en la codificación, tales como: entorno de ejecución, frameworks, librerías, entre otros.

1.3.1. Lenguaje de programación para el desarrollo web.

Hoy en día, existen muchos lenguajes de programación orientados al desarrollo web y cada vez siguen apareciendo nuevas tendencias que van ganando mercado. En este caso en particular se hablará de JavaScript.

- **JavaScript y Node.js**

Crockford (2008) define a JavaScript como un lenguaje de desarrollo importante porque es el lenguaje del navegador Web. Está asociado con los navegadores más populares y de hecho este es lenguaje de programación del lado del cliente más popular del mundo.

Por su parte Sphinx (2016) afirma que:

> Javascript es un lenguaje interpretado orientado a objetos de peso ligero, con funciones de primera clase, y es mejor conocido como el lenguaje de script para páginas web, pero ha sido utilizado en muchos entornos no-navegador también. Es un lenguaje de scripts basado en prototipos y multi-paradigma que es dinámico y admite estilos de programación orientados a objetos, imperativos y funcionales.

Analizando las definiciones de ambos autores, se puede decir que, el primero se enfoca en resaltar el uso de JavaScript en los navegadores, por otro lado, el segundo se centra en destacar características de las fortalezas del lenguaje de programación, y hace mención en que es usado en entornos “no-navegador”, por ejemplo, Node.js.

JavaScript nace a principios de los noventas, especialmente para resolver una problemática de validación de formularios en los navegadores. Pero con el transcurso de los años, este lenguaje de programación ha ido evolucionando de manera muy positiva, a tal punto de ser usado no sólo del lado del cliente sino también ahora en entornos BackEnd o del lado del servidor.

Se destaca la definición de la Fundación Node.js (2009) la cual menciona que, “es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente”.

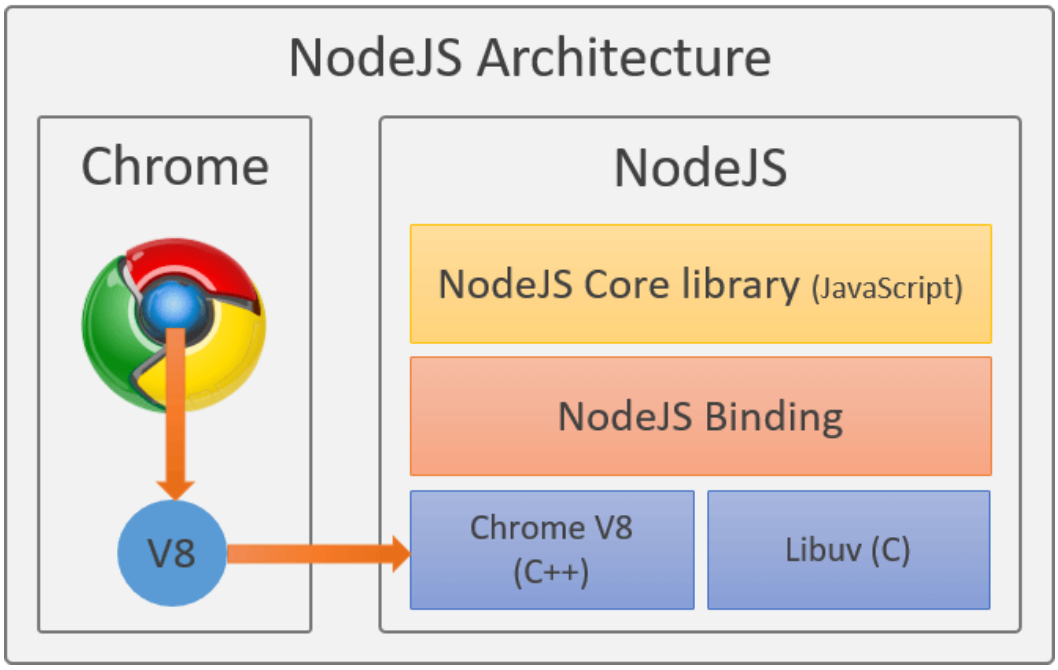


Figura 7. Arquitectura de Node.js

Fuente: (Blancarte, 2017)

Elaboración: (Blancarte, 2017)

En el sitio web Tutorials Point se destaca lo siguiente acerca del paradigma de la programación orienta a eventos:

> Node.js usa los eventos en gran medida y es también una de las razones por las que es bastante rápido en comparación con otras tecnologías similares. Tan pronto como el nodo inicia su servidor, simplemente inicia sus variables, declara funciones y espera a que ocurra el evento.
>
> En una aplicación impulsada por eventos, generalmente hay un bucle principal que escucha eventos y luego activa una función de devolución de llamada cuando se detecta uno de esos eventos.

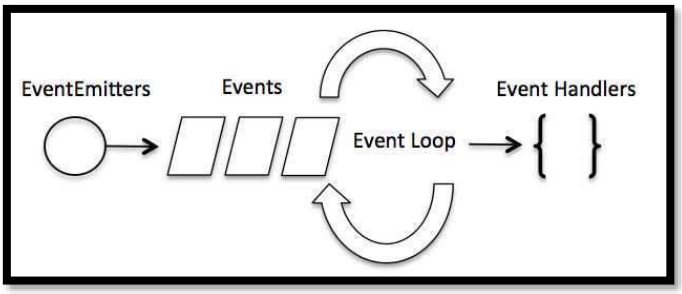


Figura 8. Programación Orienta a Eventos

Fuente: (Tutorials Point, 2015)

Elaboración: (Tutorials Point, 2015)

Además Node.js utiliza NPM, que es el manejador de paquetes oficial, mediante este gestor se puede descargar e instalar las dependencias de la aplicación en desarrollo, así como también, ejecutar o detener el entorno.

1.3.2. Framework express.js.

Según los creadores Express, (2014), es un framework de desarrollo de aplicaciones web “rápida, minimalista y flexible para Node.js”

Figura 9. Estructura del framework Express.js

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

La principal fortaleza de este framework radica en lo sencillo y fácil que es crear una aplicación web en tan solo minutos, además de ser publicado como código abierto.

1.3.3. Aplicación google charts.

Según los desarrolladores Google Developers (2007), “Google Charts proporciona una forma perfecta de visualizar datos en su sitio web. Desde simples gráficos de líneas hasta complejos mapas de árboles jerárquicos, la galería de gráficos proporciona una gran cantidad de tipos de gráficos listos para usar.”

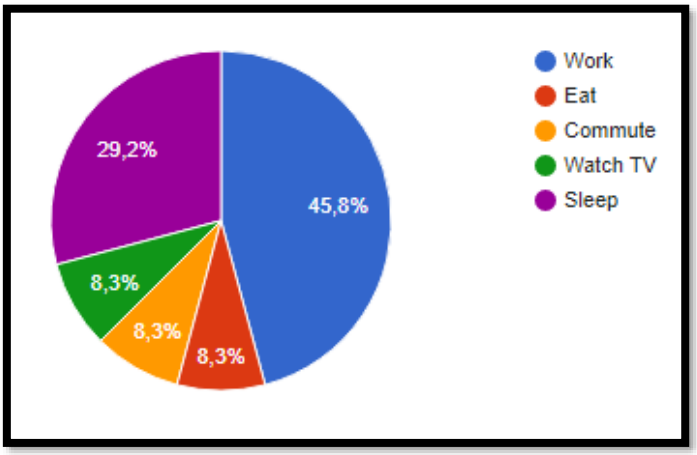


Figura 10. Ejemplo de grafico creado con Google Charts

Fuente: (Google Developers, 2007)

Elaboración: (Google Developers, 2007)

Luego de analizar detenidamente todo lo antes mencionado donde se destacan puntos importantes acerca de JavaScript, Express.js y Google Charts, se cree conveniente usar todo esto en conjunto para desarrollar la aplicación Frontend, no sólo porque se ajusta a las necesidades del proyecto sino también porque son de código abierto, es decir, no se necesita de licencias para su utilización.

1.4. Apache JMeter

Según los creadores de Apache JMeter (2016), “es un software de código abierto, una aplicación Java 100% pura, diseñada para cargar el comportamiento funcional de la prueba y medir el rendimiento.”

Aunque en principio Apache JMeter fue diseñado para probar aplicaciones Web, hoy en día también permite simular una carga masiva en un servidor o grupo de servidores, red u objeto para probar su fortaleza.

A través de la interface GUI de Apache JMeter permite diseñar, crear y ejecutar un plan de pruebas.

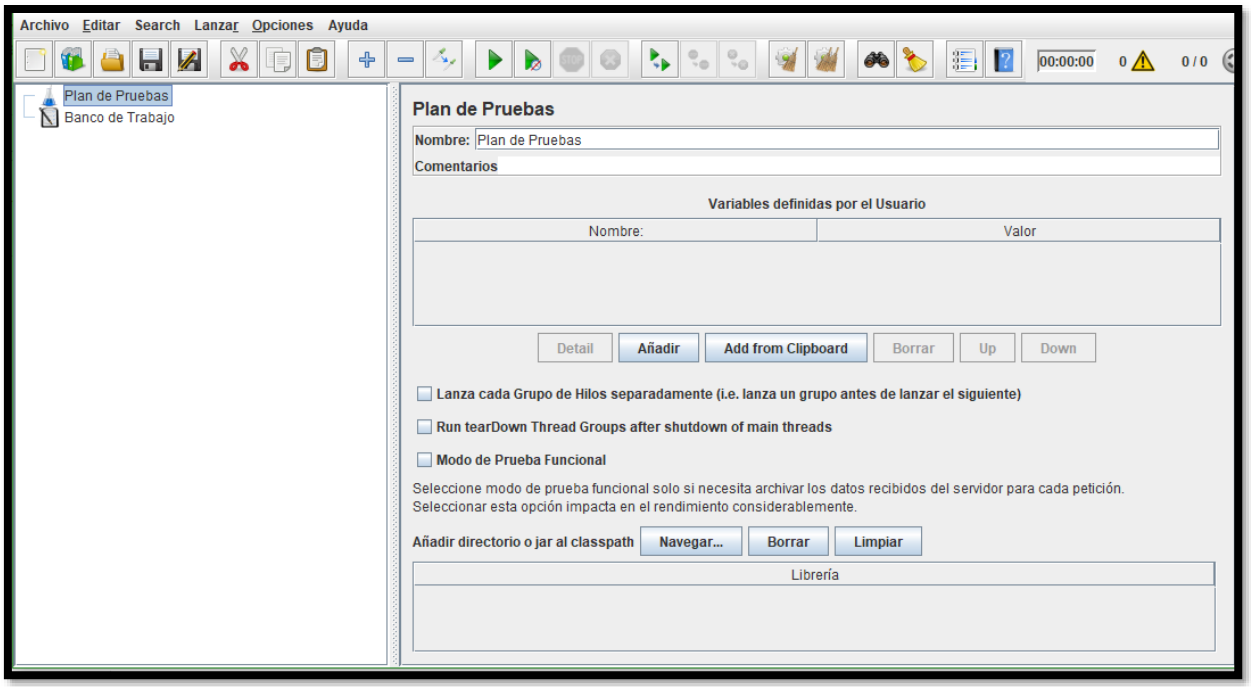


Figura 11. Interfaz gráfica de Apache JMeter

Fuente: Apache JMeter (2016)

Elaboración: Borys Zuñiga

1.5. Ansible

“Ansible es una herramienta de automatización de TI. Puede configurar los sistemas, implementar software y orquestar tareas más avanzadas como despliegue continuo” (Red Hat, 2016).

Ansible sólo necesita ser instalado en las máquinas que se usa para administrar la infraestructura. No necesita un cliente para ser instalado en el equipo administrado, ni necesita infraestructura de servidor para configurar antes de que se puede utilizar.

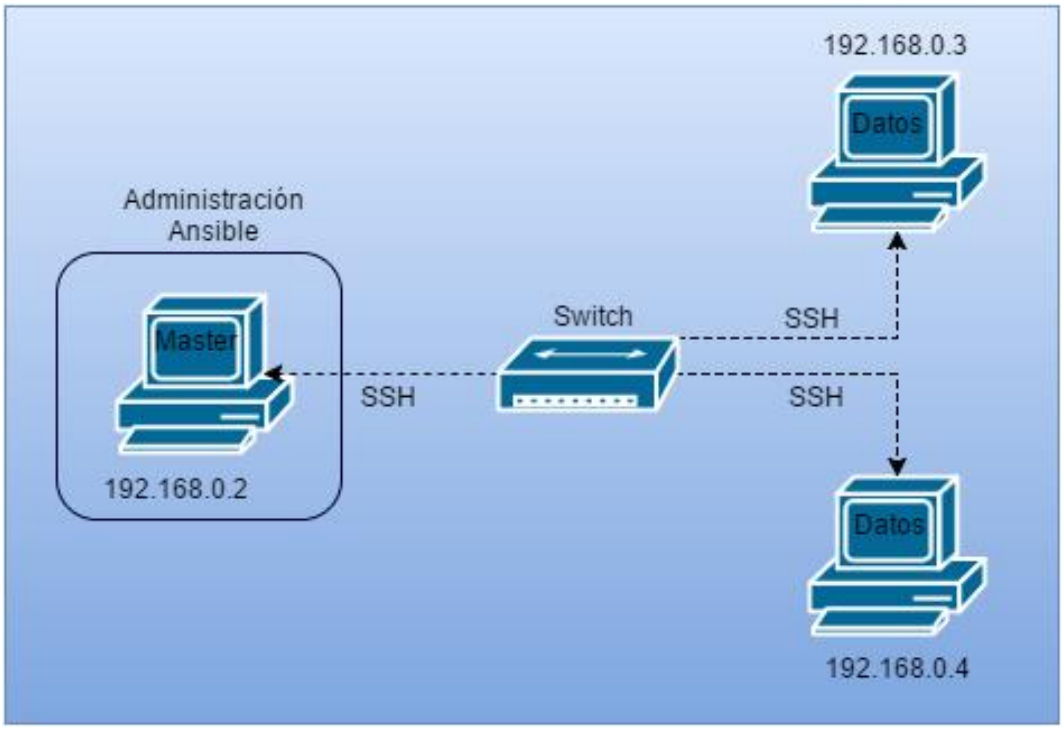


Figura 12. Despliegue de Ansible

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

1.5.1. Playbooks.

Para empezar a utilizar los módulos básicos de Ansible se puede utilizar la línea de comandos. Sin embargo, “su principal poder radica en la utilización de scripts, ya que en las máquinas administradas se suele hacer más de una cosa a la vez, es por eso que Ansible incorpora un concepto denominado *playbook* para hacer esto” (Hall, 2013, p.17). El uso de playbooks permite realizar muchas acciones a la vez, a través de múltiples sistemas. Permiten orquestar las implementaciones, garantizar una configuración coherente o realizar una tarea en común.

Los playbooks se expresan en YAML, en su mayor parte, ya que Ansible utiliza un analizador YAML estándar.

> Los playbooks también abren muchas oportunidades. Permiten llevar el estado de un comando a otro. Por ejemplo, se puede tomar el contenido de un archivo en una máquina, registrarlo como una variable y luego utilizar el valor en otra máquina. Esto permite realizar mecanismos de despliegue complejo que serían imposibles sólo con los comandos de Ansible. (Hall, 2013, p.17)

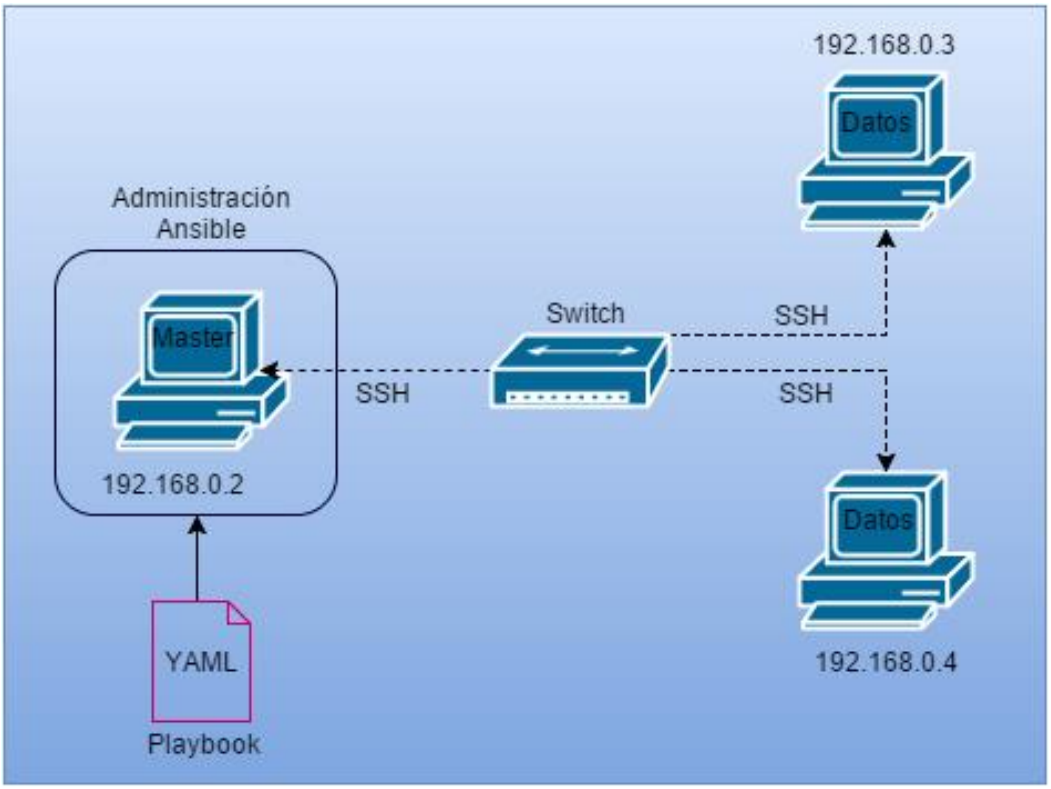


Figura 13. Incorporación de Playbooks en Ansible

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

1.5.2. Composición de playbooks.

Según Hall (2013), un playbook está compuesto por uno o más *plays.* Un play está compuesto de tres secciones:

- **Sección de destino (Target section):** define los hosts que se llevará a cabo el play, y cómo se ejecutará. Aquí es donde montamos el nombre de usuario SSH y otros ajustes relacionados con SSH.
- **Sección variable:** define las variables, que estarán disponibles para el play mientras se está ejecutando.
- **Sección Tarea (Task section):** enumera todos los módulos en el orden que queremos que se ejecuten por Ansible.

Se puede incluir tantos playbooks como se desee en un sólo archivo YAML. Los archivos YAML inician con ---, utilizan indentación para indicar anidamiento variable al analizador y para facilitar la lectura del archivo. A continuación, se presenta un ejemplo de un playbook:

```
---
- hosts: webservers
  user: root
  vars:
    apache_version: 2.6
    motd_warning: 'WARNING: Use by ACME Employees ONLY'
    testserver: yes
  tasks:
    - name: setup a MOTD
      copy:
        dest: /etc/motd
        content: "{{ motd_warning }}"
```

Figura 14. Playbook en Ansible

Fuente: (Hall, 2013)

Elaboración: (Hall, 2013)

1.6. Computación distribuida

1.6.1. Definiciones.

“Es un conjunto de computadores independientes, interconectados a través de una red y que son capaces de colaborar con el fin de realizar una tarea” (M.L.Liu, 2004, p.1).

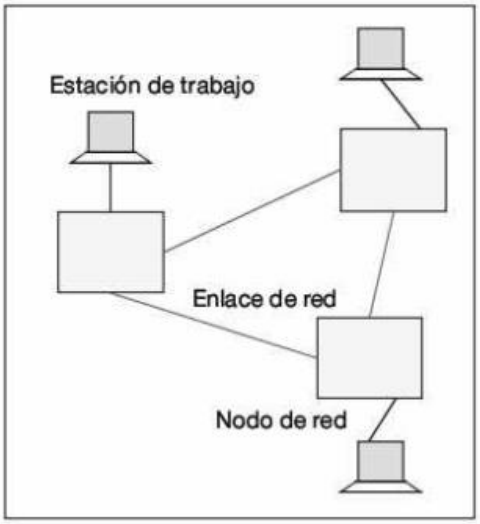


Figura 15. Computación Distribuida

Fuente: (M.L.Liu, 2004)

Elaboración: (M.L.Liu, 2004)

“Un sistema distribuido es aquel en el que los componentes localizados en computadores, conectados en red, comunican y coordinan sus acciones mediante el paso de mensajes” (Coulouris, Dollimore, & Kindberg, 2007, p.1)

En cualquier caso, varios equipos (nodos) conectados a una red, ya sea en un mismo rack o en un sitio diferente del mundo, hacen parte de un sistema distribuido.

1.6.2. Ventajas de la computación distribuida.

1.6.2.1. Heterogeneidad.

Coulouris, Dollimore, & Kindberg (2007) afirman que, “internet permite que los usuarios accedan a servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadores.” Esta heterogeneidad (es decir, variedad y diferencia) se aplica a todos los siguientes elementos:

- Redes.
- Hardware de computadora.
- Sistemas Operativos.
- Lenguajes de programación.
- Implementaciones de diferentes desarrolladores.

1.6.2.2. Extensibilidad.

“Es la característica que determina si el sistema puede ser extendido y re implementado en diversos aspectos” (Coulouris, Dollimore, & Kindberg, 2007, p.17).

1.6.2.3. Escalabilidad.

“Se dice que un sistema es escalable si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios” (Coulouris, Dollimore, & Kindberg, 2007, p.19).

1.6.2.4. Tolerancia a fallos.

“Los fallos en un sistema distribuido son parciales; es decir, algunos componentes fallan mientras otros siguen funcionando” (Coulouris, Dollimore, & Kindberg, 2007, p.20).

Los sistemas distribuidos proporcionan un alto grado de *disponibilidad* frente a los fallos del hardware. La disponibilidad de un sistema mide la proporción del tiempo en que está utilizable.

1.6.2.5. Concurrencia.

“Tanto los servicios como las aplicaciones proporcionan recursos que pueden compartirse entre los clientes en un sistema distribuido. Existe por lo tanto una posibilidad de que varios clientes intenten acceder a un mismo recurso compartido a la vez” (Coulouris, Dollimore, & Kindberg, 2007, p.21).

1.7. Big data

Barranco Fragoso (2012), afirma que:

> En términos generales *Big Data* se podría referir como la tendencia en el avance de la tecnología que ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones, la cual es utilizada para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a un base de datos relacional para su análisis. Big Data no se refiere a alguna cantidad en específico, ya que es usualmente utilizado cuando se habla en términos de petabytes y exabytes de datos.

Tabla 11. Unidades de información

Gigabyte = 10^9 = 1,000,000,000
Terabyte = 10^{12} = 1,000,000,000,000
Petabyte = 10^{15} = 1,000,000,000,000,000
Exabyte = 10^{18} = 1,000,000,000,000,000,000
Zettabyte = 10^{21} = 1,000,000,000,000,000,000,000

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

1.7.1. ¿Qué tipo de datos debo explorar?

Actualmente existe una gran variedad de datos a analizar, por lo que las empresas y organizaciones no se deben centrar en hacerse la pregunta “¿Qué tipo de datos debo explorar?”, sino más bien “¿Qué problema necesito resolver?”.

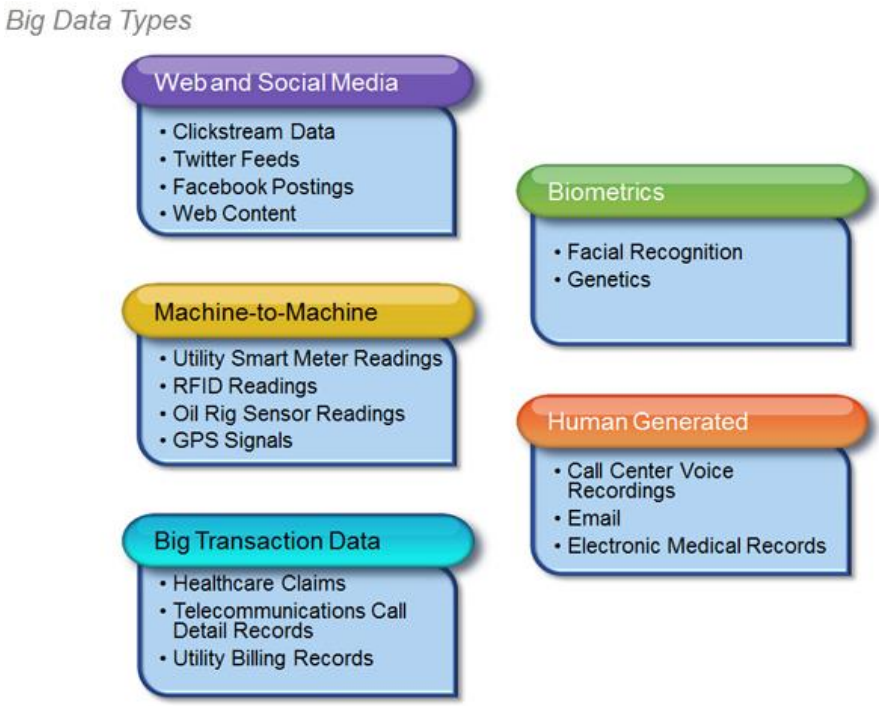


Figura 16. Tipos de datos de Big Data

Fuente: (Soares, 2012)

Elaboración: (Soares, 2012)

En el sitio web de IBM, Barranco Fragoso (2012) afirma:

- *Web and Social Media*: Incluye contenido web e información que es obtenida de las redes sociales como Facebook, Twitter, LinkedIn, etc, blogs.
- *Machine-to-Machine (M2M)*: M2M se refiere a las tecnologías que permiten conectarse a otros dispositivos. M2M utiliza dispositivos como sensores o medidores que capturan algún evento en particular (velocidad, temperatura, presión, variables meteorológicas, variables químicas como la salinidad, etc.) los cuales transmiten a través de redes alámbricas, inalámbricas o híbridas a otras aplicaciones que traducen estos eventos en información significativa.
- *Big Transaction Data*: Incluye registros de facturación, en telecomunicaciones registros detallados de las llamadas (CDR), etc. Estos datos transaccionales están disponibles en formatos tanto semiestructurados como no estructurados.
- *Biometrics*: Información biométrica en la que se incluye huellas digitales, escaneo de la retina, reconocimiento facial, genética, etc. En el área de seguridad e inteligencia, los datos biométricos han sido información importante para las agencias de investigación.

- *Human Generated*: Las personas generamos diversas cantidades de datos como la información que guarda un *call center* al establecer una llamada telefónica, notas de voz, correos electrónicos, documentos electrónicos, estudios médicos, etc.

1.7.2. Análisis de big data y pre procesamiento.

Los conjuntos de datos recopilados a gran escala se encuentran en diferentes ubicaciones con diferentes formatos. Así, los conjuntos de datos reunidos están generalmente en un estado crudo con mucha redundancia, inconsistencia o información inútil. Por lo tanto, la gran cantidad de datos semiestructurados y no estructurados hacen imposible encajar en las bases de datos relacionales con tablas ordenadas de columnas y filas. Las bases de datos NoSQL se están convirtiendo en la tecnología alternativa para los grandes datos. Para evitar espacio de almacenaje innecesario y asegurar la eficacia de procesamiento, los datos deben ser previamente procesados para estar listo para el análisis de los datos, antes de que se transmita a los sistemas de almacenamiento…Tres técnicas comunes de procesamiento de datos son: integración, limpieza y eliminación de redundancia. (He et al., 2016)

1.7.3. Predicciones del tráfico IP para el año 2020

El sitio web oficial de Cisco (2016), en una iniciativa que realiza, predice cómo será el tráfico IP para el año 2020, y en el cual se puede destacar lo siguiente:

- **El tráfico IP global anual superará el umbral de Zettabyte (ZB; 1000 exabytes [EB]) en el año 2016 y llegará a 2.3 ZB en 2020.** El tráfico IP global alcanzará 1.1 ZB por año o 88.7 EB (1 billón de gigabytes [GB]) por mes en el año 2016. En 2020, el tráfico IP global llegará a 2.3 ZB por año, o 194 EB al mes.
- **El tráfico de teléfonos inteligentes superará el tráfico de PC en 2020.** En 2015, los PC representaron el 53 por ciento del tráfico IP total, pero en 2020 los PC, serán responsables únicamente el 29 por ciento del tráfico. Los teléfonos inteligentes representarán el 30 por ciento del total del tráfico IP en 2020, por encima del 8 por ciento en 2015. El tráfico PC originado crecerá a una tasa compuesta anual del 8 por ciento, mientras que los televisores tabletas teléfonos inteligentes, módulos, y máquina-a-máquina (M2M) tendrá las tasas de crecimiento de tráfico de 17 por ciento, 39 por ciento, 58 por ciento, y 44 por ciento, respectivamente.
- **El número de dispositivos conectados a redes IP será tres veces mayor que la población mundial en el año 2020.** Habrá dispositivos de red 3.4 per cápita en 2020,

frente a los dispositivos de red 2.2 por habitante en 2015. Acelerados en parte por el aumento de los dispositivos y las capacidades de los dispositivos, el tráfico IP per cápita llegará a 25 GB per cápita en 2020, frente a los 10 GB per cápita en 2015.

Haciendo uso de la herramienta VNI Forecast Highlights, se puede observar las predicciones para el año 2020:

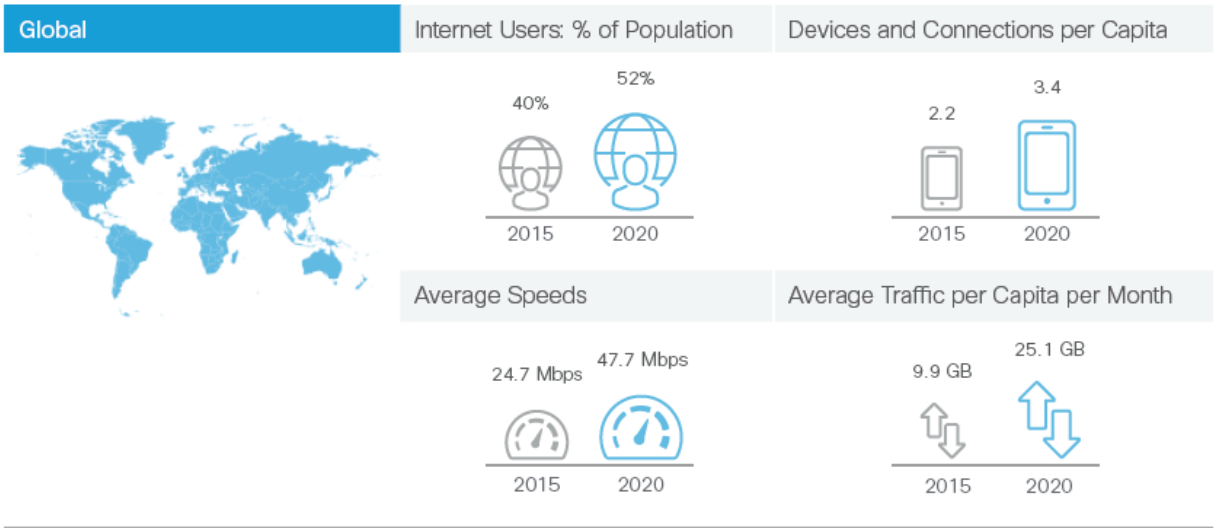


Figura 17. Previsión del 2020

Fuente: (Cisco, 2016b)

Elaboración: (Cisco, 2016b)

En estos momentos se deberá estar haciendo la pregunta ¿Y qué tiene que ver todo el tráfico IP con Big Data? La respuesta a esta pregunta es; mucho, ya que son conceptos que están ligados directamente. En resumen, a lo antes mencionado, el que se genere día a día más tráfico IP a nivel mundial, se debe a que cada día se conectan nuevos dispositivos a la red mundial (Internet), y esto a su vez hace que genere datos exorbitantes, los cuales, al crecer en su mayoría se hacen intratables, en otras palabras, al generarse datos de manera masiva se hacen cada vez menos analizables lo cual no genera un valor real a las organizaciones. Por tal motivo, la realidad del mundo actual, hace que se tengan que buscar soluciones viables, para el tratamiento, procesamiento y el correcto análisis de los datos que se generan masivamente.

1.8. Interfaz de programación de aplicaciones y protocolo HTTP

1.8.1. API REST.

Denominado así por sus siglas en inglés *Representational State Transfer.* Arkhipkin, Lauret, & Shanmuganathan, (2015) definen a una API REST como “un modelo de diseño arquitectónico

que permite desarrollar servicios escalables y de alto rendimiento utilizando un conjunto de interfaces sencillas y amigables para la web.”

Por su parte Chen, Ji, Fan, & Zhan, (2017) destacan lo siguiente, “es una especificación técnica sobre intercomunicación heterogénea para aplicaciones web. La adopción de REST puede conducir a una arquitectura simple, escalable, efectiva, segura y confiable.” REST está basada en recursos, que propiamente dicho por los autores antes mencionados, “…un recurso es un tipo de información a la que se puede acceder, como un objeto de aplicación, un registro de base de datos, un algoritmo, etc.”

Los mencionados recursos en los que se basa REST, están identificados por un URI (Universal Resource Indentifier), los cuales están representados a través de direcciones, por ejemplo: “indice/_mapping”. Sobre estos recursos se pueden ejecutar las diferentes operaciones HTTP tales como: GET, PUT, POST, DELETE, HEADER y OPTIONS.

Otra característica importante de REST, es que el lado del servidor mantiene “*stateless*” o sin estado entre múltiples interacciones, lo que significa que cada operación solicitada por el cliente es independiente, es decir, que cada servidor en los clústeres puede servir al cliente en cada solicitud.

En resumen, una API REST es una interfaz a través de la cual un cliente puede acceder a determinados recursos de un servidor a por intermedio de operaciones como: GET, PUT, POST, DELETE, etc.

1.8.2. Protocolo HTTP.

De sus siglas en inglés *Hypertext Transfer Protocol.* Se destaca la definición de Gonzalez, (2017) el cual argumenta lo siguiente:

> Es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada, por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así una página web completa, resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts...

1.9. Trabajos relacionados

1.9.1. T1: Monitorización como un servicio para las aplicaciones en la nube de computación científica que utilizan el ecosistema ElasticSearch.

- **Introducción**

  El centro de computación INFN-Torino alberga una infraestructura privada en la nube, que brinda servicios de infraestructura como servicio (IaaS) y plataforma como servicio (PaaS) a diferentes aplicaciones informáticas científicas. La implementación de máquinas virtuales se gestiona con el controlador de nube OpenNebula.

  El principal interesado de la instalación es un sitio WLCG Tier-2, que sirve principalmente a ALICE. Aproximadamente el 60% de las instancias virtuales que se ejecutan en la nube de Torino son trabajadores de la red de nodos. La segunda aplicación más grande es un sitio separado de la red DIRAC para la colaboración BESIII, que cuenta con alrededor de 200 núcleos dedicados. Los recursos se asignan estáticamente a estas aplicaciones.

  En el nivel de PaaS, el centro alberga ALICE Virtual Analysis Facility (VAF), es decir un clúster Proof on Demand (PoD) basado en μ-cernvm y HTCondor. El clúster puede ajustar automáticamente su tamaño de acuerdo con las solicitudes de los usuarios. Además, Torino Cloud aloja máquinas virtuales individuales personalizadas y adaptadas a las necesidades de casos de uso específicos (es decir, simulaciones de plantas nucleares y detectores de silicio ultrarrápidos). En el nivel IaaS, la instalación proporciona varios entornos de espacio aislado donde los inquilinos puede crear instancias de granjas de lote privadas (Bagnasco et al., 2015).

- **Problema**

  Un sistema de monitoreo debería proporcionar información casi en tiempo real sobre la utilización de los recursos y la salud de la infraestructura. Además, debería ser capaz de generar alarmas y desencadenar contramedidas. El objetivo es lograr una interfaz de monitoreo uniforme y fácil de usar como punto de entrada único para estos datos misceláneos (Bagnasco et al., 2015).

- **Implementación**

  En la implementación actual, los datos heterogéneos se envían a diferentes bases de datos MySQL y luego se envían a Elasticsearch (ES) a través de Logstash. Se configura un conjunto de paneles Kibana con consultas predefinidas para mostrar la información relevante en cada caso.

Los datos de monitoreo se almacenan en un servidor MySQL de alta disponibilidad antes de ser insertados en el motor de ES. Esto podría verse como un paso redundante ya que los datos están parcialmente duplicados entre la base de datos y ES. De hecho, ES en sí mismo se puede configurar como un clúster de base de datos de pleno derecho (NoSQL) utilizando sus funciones incorporadas de alta disponibilidad. Sin embargo, para asegurar flexibilidad y modularidad en el sistema de monitoreo, se elige usar ES como un motor de búsqueda puro con una configuración simple de un solo nodo y para delegar la preservación de datos a la base de datos MySQL. Con el uso de una solución de back-end estándar y ampliamente utilizada, se podría alejar sin problemas de la pila ELK sin perder datos históricos, en caso de que demuestre que no es la herramienta óptima (Bagnasco et al., 2015).

1.9.2. T2: Monitoreo del rendimiento de una infraestructura informática altamente distribuida y compleja en LHCb.

- **Introducción**

  DIRAC interware es un framework que ofrece una solución completa para administrar recursos informáticos heterogéneos distribuidos, como Grid, Cloud y permite a los usuarios el uso eficiente de estos recursos. LHCbDIRAC es la extensión LHCb de DIRAC, que implementa los requisitos informáticos específicos de LHCb. Dentro de LHCb se están ejecutando más de 134 servicios y 146 instancias de agentes, lo que requiere monitoreo en tiempo real para descubrir problemas y tomar diferentes medidas, por ejemplo: reiniciar un servicio, detener una determinada producción o notificar a un determinado sitio en caso de falla, eliminar un cierto elemento de almacenamiento de la producción o notificar a la persona adecuada. Además, el monitoreo de diferentes actividades como análisis de datos, procesamiento de datos, replicación de datos, fusión de datos, eliminación de datos y simulación de Monte Carlo también es crucial para poder monitorear cada actividad y hacer un seguimiento del uso de los recursos informáticos (Mathe, Haen, & Stagni, 2017).

- **Problema**

  DIRAC interware requiere monitoreo en tiempo real para descubrir problemas y tomar diferentes medidas, por ejemplo: reiniciar un servicio, detener una determinada producción o notificar a un determinado sitio en caso de falla, eliminar un cierto elemento de almacenamiento de la producción o notificar a la persona adecuada (Mathe et al., 2017).

- **Implementación**

  El resultado es el nuevo Monitoring System para monitoreo y análisis en tiempo real, que recolecta datos de todos los componentes: agentes, servicios y tareas de DIRAC; y permite crear informes a través de Kibana y la aplicación web, que se basa en el marco web DIRAC. La aplicación web de es la interfaz de usuario oficial, porque:

  - ✓ Es fácil de usar; proporciona selectores muy simples.
  - ✓ Está integrado a DIRAC; no requiere aprender una herramienta adicional.
  - ✓ Proporciona todas las funcionalidades que tiene Kibana

  Además de estas ventajas, permite crear parcelas de alta calidad. En este artículo se presenta el nuevo sistema de monitoreo, que está basado en el motor analítico y de búsqueda distribuido Elasticsearch y diseñado para almacenar datos de series de tiempo. Se implementa en python utilizando diferentes bibliotecas: Elasticsearch-dsl para interactuar con la biblioteca de trazado Elasticsearch, Matplotlib para crear gráficos, el software RabbitMQ Message Broker para la conmutación por error y otras funcionalidades proporcionadas por DIRAC. Se utiliza un módulo ElasticsearchDB que es un envoltorio alrededor de Elasticsearch-dsl, que expone las funcionalidades de la base de datos a través de una interfaz amigable para desarrolladores. Este módulo es utilizado por la utilidad Monitoring DB que implementa consultas específicas utilizadas para recuperar los datos (Mathe et al., 2017).

1.9.3. Características de los trabajos relacionados

Dentro de los trabajos relacionados se puede identificar algunas características las cuales se pasan a describir a continuación:

- **Monitoreo:** el proyecto se basa la utilización de Elasticsearch para la observación, seguimiento y estudio de un programa en particular.
- **Búsqueda de texto completo:** Elasticsearch es utilizado para hacer uso de sus características de búsqueda de texto completo (por ejemplo: permite realizar consultas que buscan una o varias palabras, frases específicas, frases que empiezan por un texto determinado, etc).
- **Flexibilidad:** dentro del proyecto en el cual Elasticsearch es usado, también es susceptible de ser cambiado.
- **Elasticsearch (ES) como base de datos:** dentro del proyecto Elasticsearch es usado como base de datos.

- **Kibana:** herramienta para visualizar y explorar los datos que están indexados dentro de Elasticsearch.
- **Logstash:** herramienta que permite administrar logs de aplicaciones.
- **Disponibilidad:** acceso por parte de los usuarios a los datos almacenados en Elasticsearch la mayor parte del tiempo.

Tabla 12. Resumen de características de trabajos relacionados

	Monitoreo	**Búsqueda texto completo**	**Flexibilidad**	**ES como BD**	**Kibana**	**Logstash**	**Disponibilidad**
T1	✓	✓	✓	x	✓	✓	x
T2	✓	✓	x	✓	✓	x	x

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Luego de analizar los trabajos relacionados, se puede decir que, Elasticsearch está siendo utilizado para monitorizar en tiempo real otras tecnologías, en las cuales existe un alto tráfico generado por agentes, servicios, tareas, etc.

En el proyecto T1 no se utiliza Elasticsearch como una base de datos independiente, por motivos de flexibilidad, sino más bien, como una base de datos alternativa, dejando únicamente su utilización para el análisis y búsqueda de texto completo. Además, se utilizan productos adicionales de la pila ELK como Logstash para la indexación y Kibana para consultar los datos históricos almacenados, dejando una única instancia de nodo para realizar las búsquedas.

En el proyecto T2, crean un sistema de monitoreo en el cual se utiliza Elasticsearch para recolectar datos basados en el tiempo de diversos componentes, así mismo, utilizan Kibana para generar informes de los datos recolectados. Para la creación del sistema de monitoreo a través de python utilizan librerías que facilitan el trabajo de indexación, consulta, monitoreo y manejo de errores.

En ambos casos, Elasticsearch es utilizado de una manera secundaria para almacenar y posteriormente monitorear el tráfico de otras aplicaciones, para generar ya sea avisos, visualizar el rendimiento de las mismas y tomar correctivos. Por lo que, en el presente proyecto se dejará como constancia la utilización de Elasticsearch, a través de un proceso que detalla desde implementación de red, la configuración de un clúster, no sólo para la búsqueda sino también

para el almacenamiento independiente de datos; y la utilización del mismo a través de la creación de una aplicación web, todo esto validado mediante de pruebas de rendimiento y disponibilidad.

CAPÍTULO II

IMPLEMENTACIÓN, INSTALACIÓN Y CONFIGURACIÓN DE ELASTICSEARCH

2.1. Introducción

En el capítulo anterior se trató sobre los conceptos que son indispensables para lograr el entendimiento de lo que es Elasticsearch, en los puntos posteriores de este apartado se configurará un clúster, el cual finalmente será consumido con una aplicación web.

En el presente capítulo se muestra una guía detallada de cómo instalar un clúster Elasticsearch. Paso a paso se muestra desde la topología de red usada, la manera en que se encuentran distribuidos los nodos para optimizar los recursos disponibles y la configuración de cada uno de los nodos que formaran parte del clúster. Por otro lado, se dejará listo para optimizar el clúster con un conjunto de datos en el siguiente capítulo.

Es importante destacar un plus adicional para este proyecto, la utilización de la herramienta de administración de configuración *Ansible (Ver 1.5)*, así mismo se mostrará como instalar y configurar este software para la administración del clúster Elasticsearch.

Es relevante también recalcar que, la guía que se mostrará a continuación está elaborada para el sistema operativo Ubuntu, si utiliza otra distribución de Linux, OS X o Windows, deberá consultar con la guía de instalación para cada distribución.

2.2. Distribución del clúster elasticsearch

Elasticsearch está orientado para trabajar con grandes volúmenes de datos. En el siguiente gráfico se muestra la distribución del clúster a ser configurado, el mismo que consta de 3 nodos, el primero denominado “master”, que será encargado netamente de la administración general de todo el clúster; y 2 nodos denominados “datos”, los cuales estarán enfocados al procesamiento de los datos.

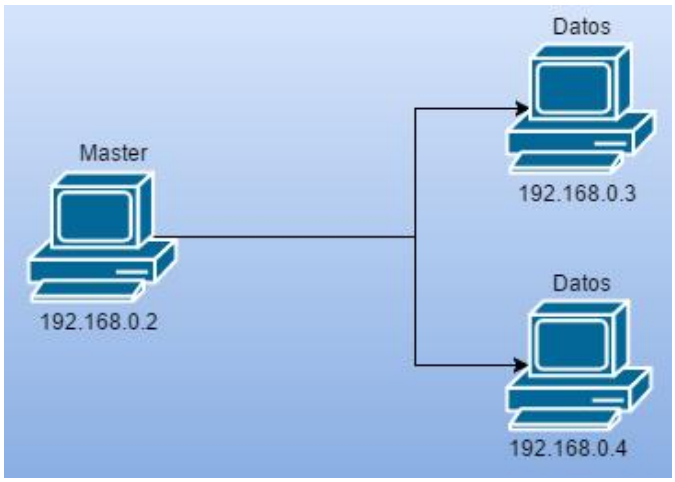


Figura 18. Distribución de los nodos del clúster Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.2.1. Topología física de la red.

Para la instalación del clúster Elasticsearch, se creará una red LAN (*Local Area Network*). Para lo cual se necesitará de los siguientes implementos:

- ✓ Un switch.
- ✓ 4 cables directos.

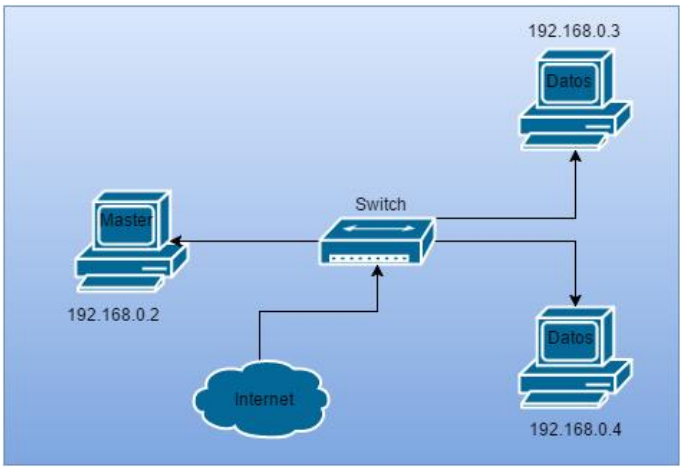


Figura 19. Distribución de los nodos del clúster Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.2.2. Requerimientos de hardware.

Como se mostró en el punto anterior se hará uso de tres hosts, los cuales tienen las siguientes características:

Tabla 13. Características Nodo 1(Master)

Procesador	Intel Core 2 Duo, 2.93 GHz x 2
Disco Duro	389.5 GB
Memoria RAM	3.8 GB
Kernel (Núcleo)	GNU/Linux
Sistema Operativo	Ubuntu 16.04 LTS
Plataforma	64 bits
Interface Ethernet	1

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Tabla 14. Características Nodo 2 (Datos)

Procesador	Intel Core 2 Duo, 2,93 GHz x 2
Disco Duro	395 GB
Memoria RAM	4 GB
Kernel (Núcleo)	GNU/Linux
Sistema Operativo	Ubuntu 12.04 LTS
Plataforma	32 bits
Interface Ethernet	1

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Tabla 15. Características Nodo 3 (Datos)

Procesador	Intel Core i5, 2,27 GHz x 4
Disco Duro	500 GB
Memoria RAM	4 GB
Kernel (Núcleo)	GNU/Linux

Sistema Operativo	Ubuntu 16.04 LTS
Plataforma	64 bits
Interface Ethernet	1

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Elasticsearch no obliga a tener requerimientos básicos de hardware, ya que la admisión de heterogeneidad de componentes es una de sus principales características. Esto es importante para el procesamiento de la información, ya que, si se dispone de mejores recursos de hardware, mejor será el rendimiento y los tiempos de respuesta a las solicitudes serán más óptimos.

2.3. Instalación de Java

Para empezar con Elasticsearch, debido a que está desarrollado en Java, el primer paso que se debe hacer es instalar un entorno Java. Elasticsearch necesita ejecutar como mínimo la versión 6 o superior. Para este manual se instalará Java SE Development Kit 8 (JDK).

2.3.1. Ubuntu.

- Primero agregue los repositorios ejecutando el siguiente comando:

  $ sudo add-apt-repository ppa:webupd8team/java

- Ahora, actualice los repositorios:

  $ sudo apt-get update

- Instalamos la versión de Java que necesitamos:

  $ sudo apt-get install oracle-java8-installer

- Si se desea verificar la versión de Java instalado, ejecute el siguiente comando:

  $ java -version

2.4. Instalación de elasticsearch

Es importante recalcar que, para efectos de pruebas, en este primer punto se descargará Elasticsearch y se ejecutará directamente (instalación rápida), dejando la instalación de Elasticsearch “como un servicio” para después.

- ✓ Para empezar con Elasticsearch diríjase a la página oficial: https://www.elastic.co/downloads, y descargue la última versión estable.

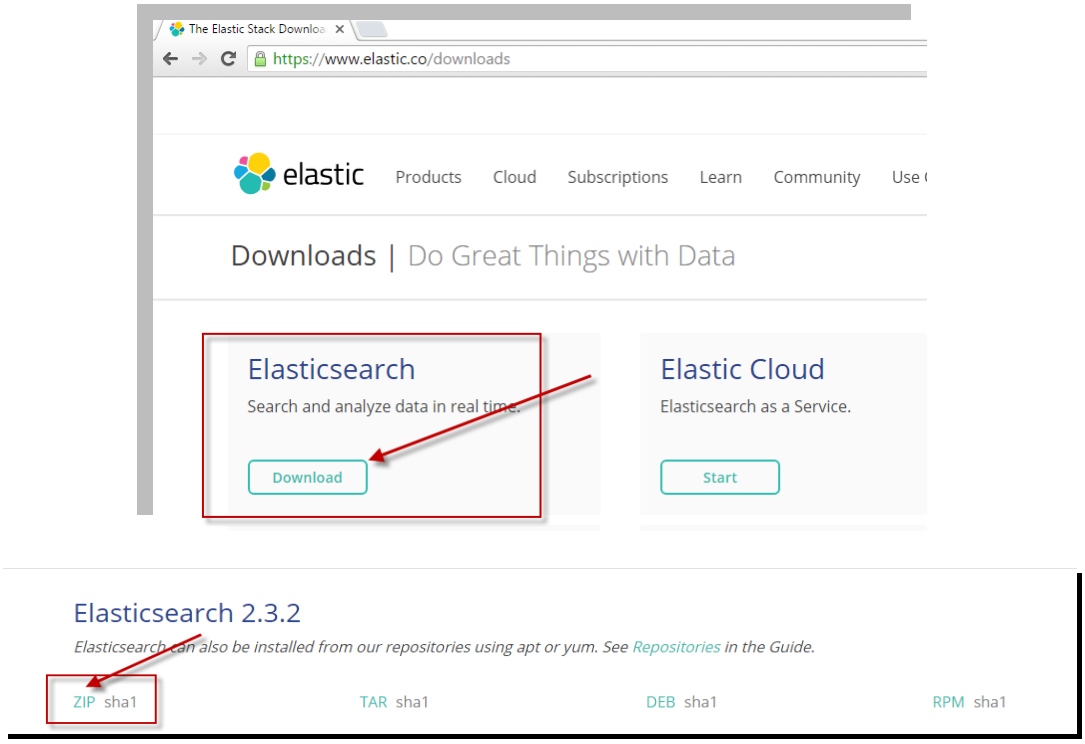


Figura 20. Descarga Elasticsearch

Fuente: (Elasticsearch, 2016)

Elaboración: Borys Zuñiga

- ✓ Luego se descomprime el archivo .zip descargado.
- ✓ Bien, ahora ingrese a la carpeta “bin” y ejecute el archivo “elasticsearch”. Elasticsearch debe empezar a levantarse.

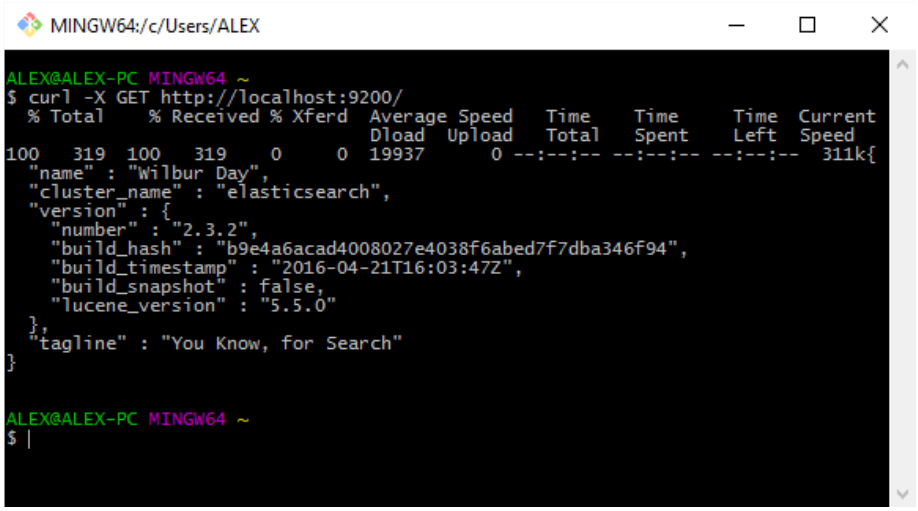


Figura 21. API REST de Elasticsearch desde consola

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la imagen anterior se muestra que Elasticsearch ha sido instalado correctamente al devolver información básica en un objeto JSON, al establecer una comunicación con el API REST de Elasticsearch. Para comprobarlo, ejecute el comando *curl -X GET http://localhost:9200/* en consola o, en el navegador se dirige a *http://localhost:9200/* y se visualizará de la siguiente manera:

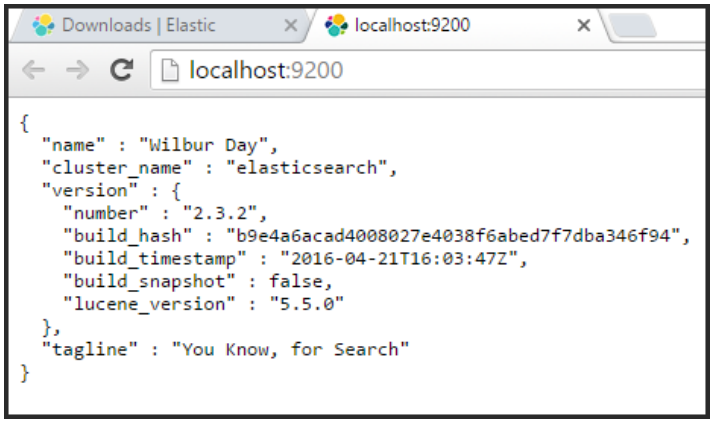


Figura 22. API REST de Elasticsearch desde el navegador

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

La interface principal de Elasticsearch está basada en el protocolo HTTP y REST, lo que significa que se puede usar un navegador para hacer algunas consultas básicas, pero para consultas más

avanzadas se necesitará de los comandos cURL o de una aplicación que facilite el uso del API REST.

Importante: Si utiliza alguna distribución de Linux o Mac OS, el paquete de los comandos cURL ya están disponibles pero si utiliza Windows, se necesitará instalar el paquete que se lo puede descargar de la siguiente dirección: http://curl.haxx.se/download.html.

2.5. Instalación de elasticsearch como un servicio

Bien, ahora ya se ha probado la "instalación rápida" de Elasticsearch donde seguramente se pudo haber dado cuenta de las principales bondades que ofrece este software, pero si se quiere utilizar de una manera más profesional, es importante instalarlo como un servicio del sistema operativo. Para el desarrollo, se instalará en una distribución de Linux (Ubuntu).

Nota Importante: los siguientes comandos (sudo) se deben ejecutar en todos los nodos que componen el servidor de Elasticsearch.

Elasticsearch se puede instalar con un administrador de paquetes, pero también se puede descargar directamente de www.elasti.co en paquetes zip, tar.gz, deb o rpm. Para Ubuntu se recomienda usar el paquete *.deb* (Debian) porque instalará todo lo que necesita para ejecutarse.

- Al momento de desarrollar este proyecto la última versión más estable (abril de 2016) de Elasticsearch fue la 2.4.3. Se descarga en el directorio de su preferencia con el siguiente comando:

  $ wget https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-2.4.3.deb

- Luego se instala de la forma más habitual de Ubuntu, con el comando *dpkg*:

  $ sudo dpkg -i elasticsearch-2.4.3.deb

- Ahora ya se puede ejecutar para probar si la instalación fue correcta:
  Si la versión de Ubuntu es la LTS 16.04:

  $ sudo /etc/init.d/elasticsearch start

  Si la versión de Ubuntu es la LTS 14.04 o 12.04:

  $ sudo -i service elasticsearch start

2.6. Configuración de elasticsearch

Para lograr el entendimiento básico de las funcionalidades de Elasticsearch, basta con ejecutar el software con las configuraciones que tiene por defecto, dichos valores son razonables para entornos sencillos y sin lugar a dudas resulta fácil, por tal motivo dicho proyecto está ganando mucha popularidad. Sin embargo, nuestros objetivos van más allá de aquello, para lo cual es necesario conocer algunas de las opciones disponibles y donde se deberán efectuar los siguientes cambios.

Si la instalación que se eligió fue “la rápida” pase a explorar el directorio por defecto de Elasticsearch, para observar cómo están distribuidos los archivos. Céntrese en el directorio “*config*”, que es donde Elasticsearch almacena la configuración completa. Se puede ver dos archivos: “*elasticsearch.yml*” y “*loggin.yml*”. El primer archivo, es el responsable de establecer los valores predeterminados para la configuración del servidor. Esto es importante porque algunos de estos valores pueden cambiarse en tiempo de ejecución y pueden conservarse como parte del estado del clúster, por lo que los valores de este archivo pueden no ser precisos. Dos de los valores que no se pueden cambiar en tiempo de ejecución son “cluster.name” y “node.name”. Ahora se procede a abrir el archivo *“elasticsearch.yml”* con la ayuda de un editor de texto cualquiera, y luego se editará con las configuraciones personalizadas.

Si instaló Elasticsearch como un servicio del sistema, entonces vale recalcar que para tener acceso y modificar el archivo “*elasticsearch.yml*”, se lo debe hacer con permisos de “super usuario” con la ayuda de algún editor de texto como “vi” o “nano”, dicho esto, se abre el archivo de la siguiente manera:

$ sudo nano /etc/elasticsearch/elasticsearch.yml

2.6.1. Nombre del clúster.

La propiedad “cluster.name” establece un nombre a nuestro clúster. Su importancia radica en que separa diferentes grupos entre sí, es decir, todos los nodos como el mismo nombre en esta propiedad intentaran formar un clúster. Esto es fundamental cuando se desee tener varios clústeres en la misma red.

Para que esta propiedad se habilite descomentamos *cluster.name.* Por ejemplo, este clúster llevará por nombre “*mi-tesis*”.

```
19  # ---------------------------------- Cluster -----------------------------------
20  #
21  # Use a descriptive name for your cluster:
22  #
23  cluster.name: mi-tesis
24  #
```

Figura 23. Configuración del nombre del clúster Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.6.2. Nombre del nodo.

La segunda propiedad "*node.name*" establece el nombre para el nodo. Cada vez que inicia la instancia, Elasticsearch elige un nombre de forma automática, y su nombre puede variar por cada reinicio, claro, esto pasa porque de forma predeterminada está configurado para que esto suceda así.

Definir un nombre a la instancia de nodo, puede facilitar a la hora de referirse a un nodo en concreto cuando se haga uso de la API o cuando se hace uso de una herramienta para monitoreo de los nodos del clúster.

Para que esta propiedad se habilite descomentamos "*node.name".* La variable ${HOSTNAME}, establece como nombre de nodo el nombre del equipo que se está utilizando. Para efectos de este proyecto se asignará los nombres a los hosts: nodo1, nodo2, nodo3; siendo nodo1, el nodo master del clúster.

```
25  # ------------------------------------ Node ------------------------------------
26  #
27  # Use a descriptive name for the node:
28  #
29  node.name: ${HOSTNAME}
30  #
```

Figura 24. Configuración del nombre de un nodo del clúster Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.6.3. IP del nodo.

La propiedad "*network.host*", establece la dirección IP del nodo. De manera predeterminada Elasticsearch establece el nodo con la dirección 127.0.0.1, es decir localhost.

Previa edición en el archivo "*elasticsearch.yml*", se debe realizar la respectiva configuración en la Interface Ethernet de cada nodo, con **IP estática**, tal y como se mostró en la topología física de

la red. Al nodo master se establece la IP: 192.168.0.2, nodo de datos 1 con la IP: 192.168.0.3 y nodo de datos 2 con la IP: 192.168.0.4. En la siguiente imagen se muestra la configuración de IP estática para el nodo master (Ver también la Figura 18).

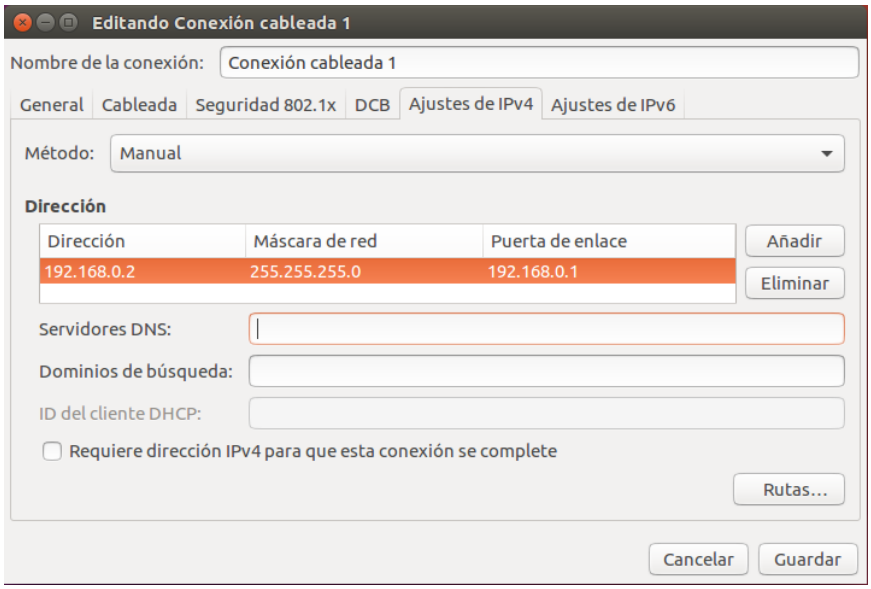


Figura 25. Configuración de dirección IP estática en Ubuntu

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Bien, ahora se pasa a descomentar la propiedad “network.host” y se establece la dirección IP del host.

```
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: 192.168.0.2
```

Figura 26. Configuración de dirección IP del nodo Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Si se quiere utilizar también de manera local el nodo entonces se configura de la siguiente manera:

$ *network.host: ["192.168.0.2", _local_]*

Si se quiere utilizar la dirección IP que es asignada por DHCP (Dynamic Host Configuration Protocol, esto para evitar estar cambiando la IP en repetidas ocasiones) y también de manera local el nodo entonces se configura de la siguiente manera:

$ *network.host: [_site_, _local_]*

2.6.4. Discovery

Elasticsearch permite elegir el tipo de descubrimiento (discovery) ya sea *multicast* o *unicast*. En redes de computadoras **multicast**, es la entrega de un mensaje a un grupo de hosts en una sola transmisión, mientras que **unicast,** es la entrega de un mensaje en la red a un sólo host a la vez.

Por defecto elasticsearch tiene el tipo de discovery multicast, es por eso que, si ejecuta el programa en varios hosts en una red, estos formaran un clúster sin realizar ninguna configuración previa.

Por seguridad, para este caso se utilizará el tipo de discovery unicast, y se lo configura de la siguiente manera. Se especifica la dirección IP de todos los nodos que conforman el clúster, de esta manera se está limitando el clúster.

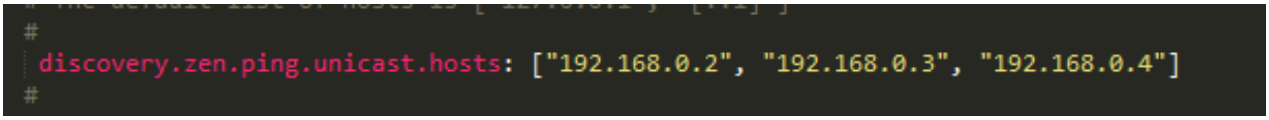


Figura 27. Configuración de la propiedad Discovery

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.6.5. Tipo de Nodo

El tipo de nodo esta descrito en el capítulo anterior en el punto 1.1.5, por lo que no cabe mayor explicación y se pasa directamente a configurar de acuerdo a la topología de red.

Para el nodo master:

$ node.master: true

$ node.data: false

Para los nodos de datos:

$ node.master: false

$ node.data: true

Importante: Los mismos pasos de todo este capítulo, se deberán ejecutar en cada uno de los hosts que conformaran el clúster Elasticsearch, respetando la topología de red que se ha planificado previamente.

Otro punto muy importante a la hora de establecer la configuración de un servidor Elasticsearch es la variable *discovery.zen.minimum_master_node,* ubicada dentro del mismo archivo, dicha variable permite establecer un número de nodos maestros mínimos, y principalmente ayuda a evitar el denominado "*split-brain*" (cerebro dividido, existencia de dos nodos maestros en un sólo grupo). Se debe tener mucho cuidado al modificarla porque, así como puede ser de mucha ayuda para darle estabilidad alg clúster, también puede resultar contraproducente llegando a provocar perdida de datos. Esta variable es modificable principalmente en escenarios donde varios nodos hacen las funciones de un nodo master y de datos a la vez, solo ahí, se deberá establecer un número a la variable en base a (número total de nodos maestros elegibles / 2 + 1). En este proyecto esta variable no se utiliza porque se tiene un nodo master y dos nodos de datos dedicados, configurados en el punto anterior.

2.7. Administración del Clúster Elasticsearch

2.7.1. Requerimientos de hardware

Ansible se utilizará desde la terminal de una computadora que se la denominará "máquina controladora", desde la cual se configurará y administrará otro ordenador al cual se lo denominará "máquina gestionada o administrada". Desde la óptica de estos dos conceptos se necesita al menos dos ordenadores, no existiendo ningún problema porque en este proyecto se dispone de tres.

Este poderoso software de gestión y administración no pone muchos requerimientos en la máquina controladora y menos aún en la maquina administrada.

2.7.2. Requerimientos de software

Ansible actualmente está disponible únicamente para una máquina controladora Linux o OS X, pero para las maquinas gestionadas está disponible para sistemas operativos tipo Unix, OS X y Windows.

2.7.2.1. Máquina controladora.

Los requerimientos para la máquina controladora son los siguientes:

- ✓ Python 2.6 o superior.

- ✓ Paramiko.
- ✓ PyYAML.
- ✓ Jinja2.
- ✓ httplib2.
- ✓ Sistema Operativo basado en Unix.

2.7.2.2. Máquina administrada.

La máquina administrada necesita:

- ✓ Python 2.4 o superior.
- ✓ Simplejson.

Nota: Cuando se instale Ansible más adelante para la máquina controladora**,** automáticamente instalará el software que requiere, mientras que para la máquina administrada deberá verificar que estén instalados los requerimientos antes mencionados.

2.7.3. Instalación de ansible.

Las distribuciones más modernas incluyen un gestor de paquetes que maneja automáticamente dependencias de paquetes y actualizaciones. Esto hace que la instalación de Ansible a través de su gestor de paquetes en gran medida sea la forma más fácil para empezar con Ansible. Para instalar el software se ejecuta el siguiente comando:

$ apt-get install ansible

Por defecto Ansible se instala en el directorio del sistema /etc/, para lo cual diríjase al mismo a través del siguiente comando:

$ cd /etc/ansible/

En el directorio /ansible/ se encuentran dos archivos: “*ansible.cfg*” y “*hosts*”, se procede a abrir el segundo fichero con el editor de texto por consola “nano”.

$ sudo nano hosts

Ahora edite el archivo, colocando las direcciones IP que conforman el clúster, luego guarde y cierre, como se muestra en la siguiente imagen:

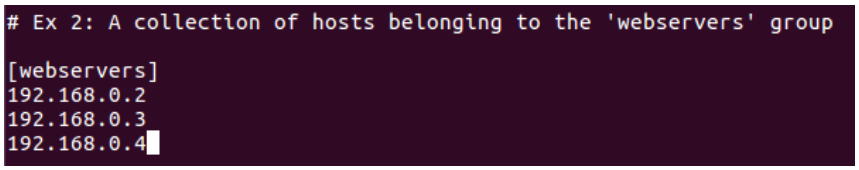


Figura 28. Configuración inventario de hosts de Ansible

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Ansible fue diseñado para ser simple, su conectividad con cada una de las máquinas administradas es a través de SSH. Esto significa que Ansible no necesita instalarse en las máquinas administradas.

2.7.4. Crear autenticación por llave SSH.

Para tener acceso remoto a las máquinas administradas, se debe disponer de una llave SSH, para lo cual, se ejecuta en un terminal de la máquina controladora el siguiente comando:

$ ssh-keygen

Una vez hecho esto, se dispone de dos ficheros: *~/.ssh/id_rsa* y *~/.ssh/id_rsa.pub*

Se procede a copiar la llave pública a todos los nodos que se van a administrar, como se muestra a continuación:

$ ssh-copy-id –i ~/.ssh/id_rsa.pub root@192.168.0.3

2.7.5. Usando algunos módulos de Ansible.

Los módulos de Ansible usualmente corren en los llamados *playbooks* (guías), pero eso no significa que no se los puedan usar desde la terminal. Para ejecutar Ansible desde la línea de comandos; primero, se necesita un patrón de host para la máquina que desea aplicar el módulo. En segundo lugar, se necesita proporcionar el nombre del módulo que desea ejecutar y opcionalmente cualquier argumento que desea dar al módulo. Para el patrón de host, puede utilizar un nombre de grupo, un nombre de máquina o la dirección IP. A continuación, se probará algunos de ellos básicamente para chequear que Ansible esté funcionando correctamente.

2.7.5.1. Módulo ping.

Este módulo de Ansible fue diseñado para verificar si se tiene comunicación con un host determinado. Ejecute el siguiente comando:

$ ansible 192.168.0.102 -u root -k -m ping

La respuesta al comando anterior debería ser la siguiente:

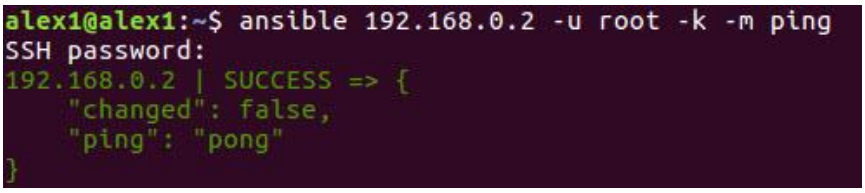


Figura 29. Módulo ping de Ansible

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Tenga en cuenta que se necesita el acceso *root* a la máquina administrada, por lo que se debería disponer de la contraseña, de no ser así, cámbiela por una nueva.

2.7.5.2. Problemas comunes al establecer comunicación.

- **Servicio SSH no instalado**

Algunas versiones de Ubuntu como, por ejemplo: Ubuntu 12.04 LTS y Ubuntu 14.04 LTS, ya tienen instalado de manera predeterminada el servicio SSH, pero en la versión Ubuntu 16.04 LTS no viene instalado por lo que se debería prestar atención a lo que se muestra a continuación.

Si la respuesta es negativa como la siguiente:

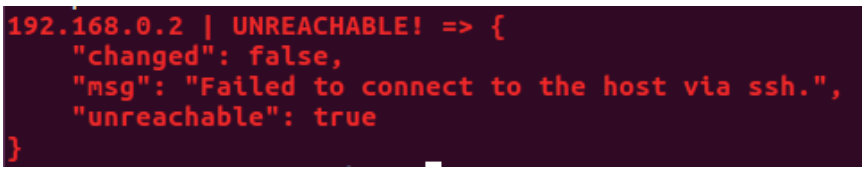


Figura 30. Error servicio SSH no instalado

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Esto se debe a que no tiene instalado el servicio SSH en la máquina remota. Ejecute el siguiente comando para verificarlo:

$ sudo service ssh status

Si la respuesta al comando anterior es negativa, entonces se instala el servicio SSH:

$ sudo apt-get install openssh-server

Bien, ahora se reinicia el servicio:

$ sudo service ssh restart

- **Conexión por SSH no agregada**

Otro error que suele presentarse a la hora de usar el módulo ping de Ansible es el siguiente:

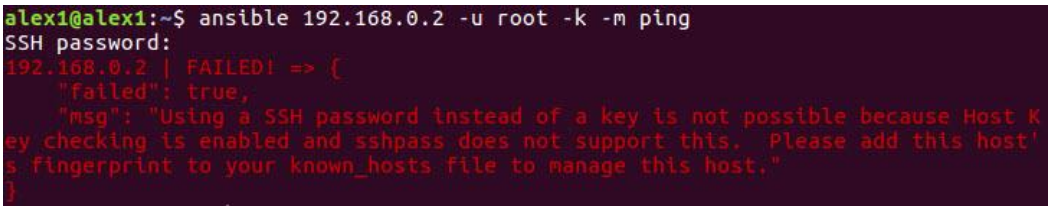


Figura 31. Error conexión SSH no agregada

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Esto se debe a que la dirección IP a la cual se está intentando conectar no está almacenada en el archivo “known_hosts”, que es el encargado de almacenar los hosts a los cuales se accede de manera remota vía SSH.

Para solucionar esto, simplemente se debería conectar primero de manera remota a través de SSH para que se almacene la conexión en el archivo que se mencionaba anteriormente.

$ ssh root@192.168.0.2

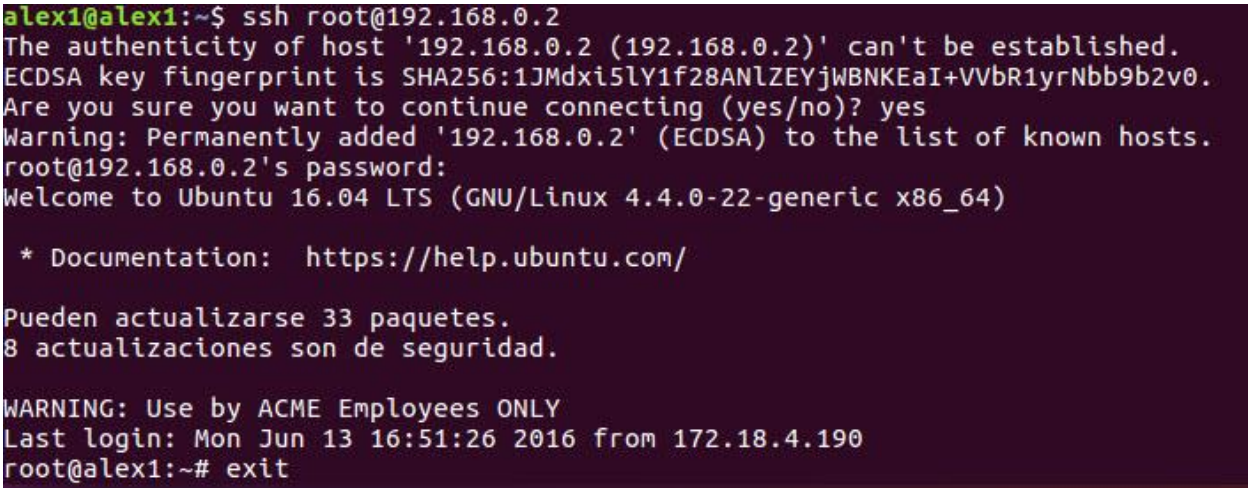


Figura 32. Conexión SSH

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Es importante aclarar que esto sólo se deberá hacer una sola vez, porque la conexión establecida quedará registrada de manera permanente. Ahora si ejecute el módulo ping de Ansible como se mostró en el punto 2.7.5.1.

- **Programa Sshpass no instalado**

Otro error muy común al querer utilizar este módulo por primera vez es el siguiente:

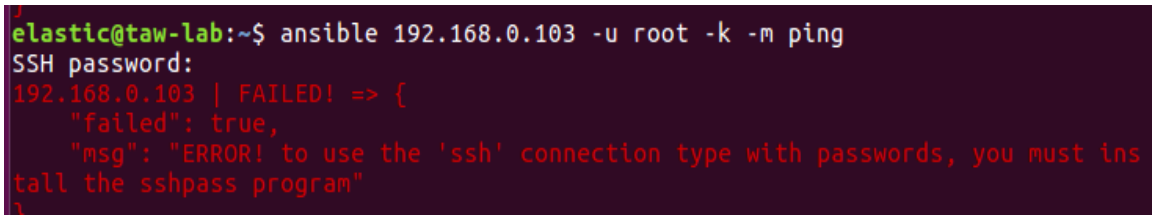


Figura 33. Error Sshpass no instalado

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Instálelo con el siguiente comando:

$ sudo apt-get install sshpass

2.7.5.3. Módulo setup.

El módulo *setup*, se conecta al nodo configurado, recoge los datos del sistema y retorna dichos valores. Esto no es particularmente práctico si se ejecuta desde la línea de comandos, pero si se los usa en un *playbook* puede ser muy útil si se lo usa con otros módulos. A continuación, se muestra el comando para usar el módulo *setup*:

$ ansible 192.168.0.2 -u root -k -m setup

La respuesta al comando es la siguiente:

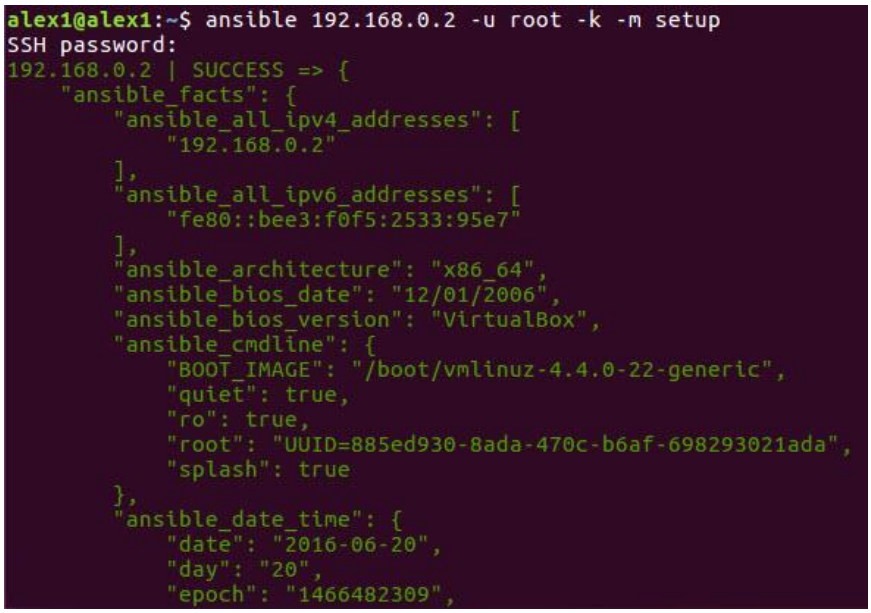


Figura 34. Respuesta módulo *setup* de Ansible

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

2.7.6. Playbooks.

Como se explicó en el punto 1.5, la principal fortaleza de este poderoso software de despliegue es la utilización de scripts (playbooks), por lo que a continuación se muestran algunos ejemplos.

Los siguientes comandos son ejecutados desde la carpeta donde está instalado Ansible:

$ *cd /etc/ansible.*

2.7.6.1. Playbook para apagar los nodos.

Se crea un archivo con el siguiente comando:

$ sudo nano

Y se escribe lo siguiente:

```
---
- hosts: webservers
  gather_facts: no
  tasks:
    - name: Apagando nodos elasticsearch
      command: /sbin/shutdown -h now
```

- name: Espere...

local_action: wait_for host={{ ansible_ssh_host }} port=22 state=stopped

Se procede a guardar el archivo YAML con extensión g.*yml*, en este caso se lo nombrará *shutdown.yml*

2.7.6.2. Playbook para reiniciar los nodos.

Se crea un archivo con el siguiente comando:

$ sudo nano

Y se escribe lo siguiente:

```
---
- hosts: webservers
  gather_facts: no
  tasks:
    - name: Reiniciando nodos elasticsearch
      command: /sbin/reboot
    - name: Espere...
      local_action: wait_for host={{ inventory_hostname }} search_regex=OpenSSH port=22
timeout=300
```

Se procede a guardar el archivo YAML con extensión *.yml*, en este caso se lo nombrará *reboot.yml*

2.7.6.3. Playbook para detener el servicio de elasticsearch.

El siguiente playbook detiene el servicio de elasticsearch de todas las máquinas administradas.

Se crea un archivo con el siguiente comando:

$ sudo nano

Y se escribe lo siguiente:

```
---
- hosts: elastic
```

```
  gather_facts: no
  become: yes
  tasks:
   - service:
      name: elasticsearch
      state: stopped
```

Se procede a guardar el archivo YAML con extensión *.yml*, en este caso se lo nombrará *parar_serv.yml*

2.7.6.4. Playbook para reiniciar el servicio de elasticsearch.

El siguiente playbook reinicia el servicio de elasticsearch de todas las máquinas administradas.

Se crea un archivo con el siguiente comando:

$ sudo nano

Y se escribe lo siguiente:

```
 ---
 - hosts: elastic
   gather_facts: no
   become: yes
   tasks:
    - service:
       name: elasticsearch
       state: restarted
```

Se procede a guardar el archivo YAML con extensión *.yml*, en este caso se lo nombrará *reiniciar_serv.yml*

2.7.6.5. Ejecutar un playbook.

Para ejecutar un playbook se lo hace con el siguiente comando:

$ ansible-playbook shutdown.yml –u root

Siendo *shutdown.yml* el nombre de cualquier playbook creado.

CAPÍTULO III

DESARROLLO DE APLICACIÓN WEB

3.1. Introducción

Hasta este punto del proyecto se ha revisado que es Elasticsearch conceptualmente, se ha instalado y configurado el clúster, quedando disponible para su utilización. Adicional a esto, se ha mostrado también de la misma forma la interesante herramienta de despliegue y configuración, Ansible.

Otro objetivo de la realización de este proyecto es la utilización del clúster Elasticsearch, que hasta el capítulo anterior se ha instalado, configurado y desplegado, por lo que, en el capítulo que se presenta a continuación, se hará uso del clúster de una manera ejemplificable, donde se cargará datos masivamente; y posterior a ello, se los consumirá con una aplicación que se realizará en JavaScript, no sin antes, realizar una optimización y limpieza de los datos que se disponen para este proyecto.

3.2. Objetivo de la aplicación

El objetivo general es crear una aplicación web con Node.js, que permita cargar masivamente los datos del Censo de Población y Vivienda del año 2010, clasificarlos de manera general (todas las provincias), por provincia, por cantón o por parroquia, y mostrar de manera gráfica el resumen de los datos, con la finalidad de probar el clúster Elasticsearch.

3.3. Alcance de la aplicación

Dentro del alcance de desarrollo de la aplicación web se contempla:

- Desarrollar la aplicación con JavaScript como lenguaje de programación del lado del servidor.
- Consumir el servidor Elasticsearch instalado, para la realización de diversas operaciones de lectura de datos tales como: consultar datos de manera general (todas las provincias), por provincia, por cantón, por parroquia.
- Indexar masivamente los datos (.csv) desde la aplicación web.
- No montar la aplicación web en un servidor de producción real, porque es requerida básicamente para consumir e indexar datos en el servidor Elasticsearch.
- No utilizar una metodología de desarrollo de software, debido a que el producto final es principalmente por cuestiones ejemplificativas de prueba y consumo del servidor Elasticsearch, pero; se explicará los componentes más relevantes del desarrollo tales como: mapeo de datos, indexación de datos, funcionamiento de la aplicación web (manual de usuario) y consultas realizadas.

3.4. Conjunto de datos (DataSet)

Los datos con los que se trabajaron fueron 3’815.528, resultado del Censo de Población y Vivienda realizado en el Ecuador en el año 2010, disponibles en: http://www.ecuadorencifras.gob.ec/base-de-datos-censo-de-poblacion-y-vivienda-2010/.

Es importante destacar que los datos descargados de la base de datos del Instituto Nacional de Estadísticas y Censos, fueron en paquete estadístico *Statistical Package for the Social Sciences (SPSS)*.

Los conjuntos de datos tienen la extensión *“.sav”,* por lo que, para la posterior carga masiva de los mismos se debió convertir a un formato más sencillo y general, como por ejemplo: archivo separado por comas (,) o *“.csv”*. Los conjuntos de datos fueron descargados por cada provincia del Ecuador. Y se detallan en el ANEXO A.

3.4.1. Descripción de los datos.

Para visualizar los datos se lo hizo con la ayuda de *RStudio*, y se los describe en el ANEXO B.

Si bien es cierto, las variables de los archivos *(.sav)* son 34, se ha decidido trabajar con las más representativas (19, detalladas en el ANEXO B) y que se considera que tienen más significado. Además, en el ANEXO B, también se describen los demás datos que contienen los archivos con los que se trabajó, para relacionar correctamente los datos y obtener información representativa.

3.4.2. Conversión de los archivos.

Como se aclaró anteriormente, los datos descargados directamente del INEC en principio se encontraban en formato *(.sav)*, por lo que trabajar en dicho formato resulta complejo, debido a que deben ser almacenados en el clúster Elasticsearch. Se ha optado por convertir dichos archivos en un formato más general y factible para la extracción mediante un mecanismo de programación y posterior carga.

Para la conversión se ha utilizado el software *RStudio*. Con este programa se importa los archivos SPSS y se convierte con el siguiente código en la consola de la aplicación:

$ write.csv(SPSS_Azuay_Hogar, file = "SPSS_Azuay_Hogar.csv", na = "", row.names = FALSE, quote = FALSE)

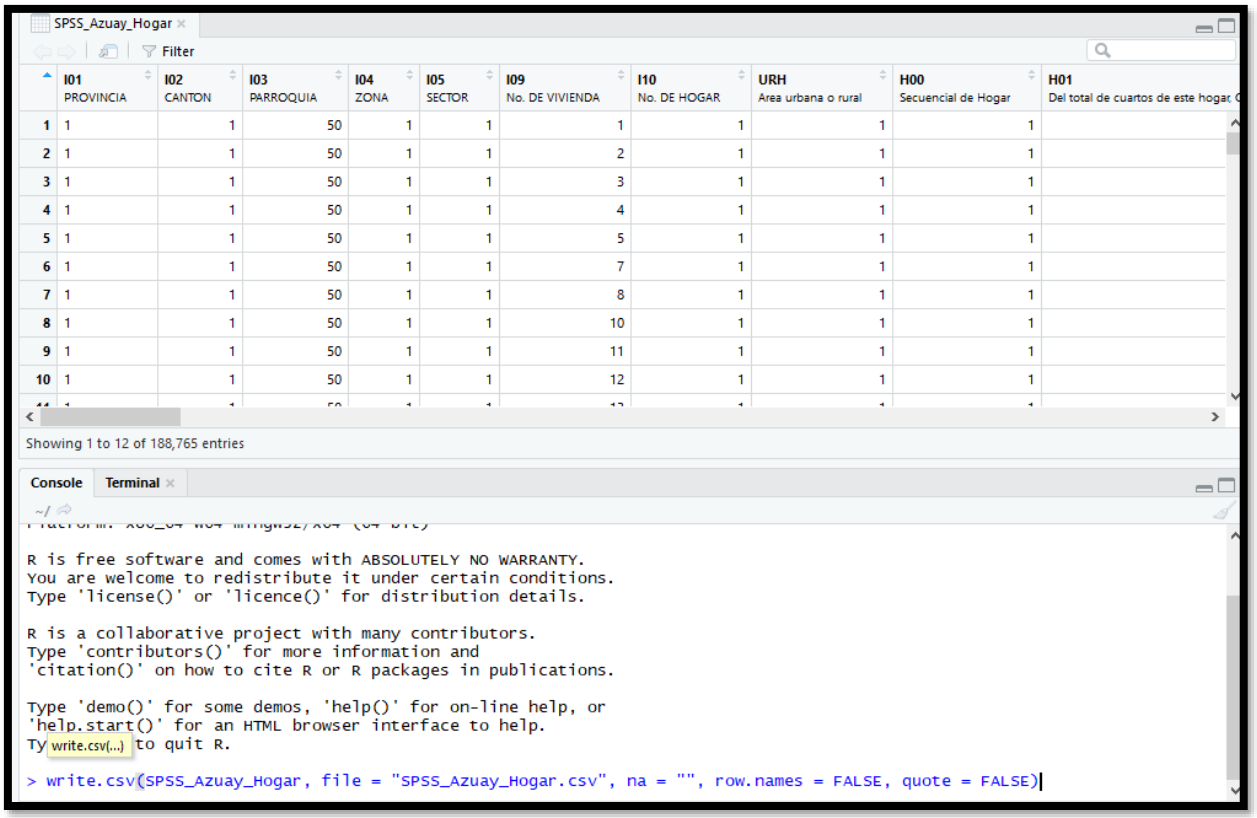


Figura 35. Captura de la importación un archivo SPSS a RStudio

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

3.5. Solución para la aplicación web

En la siguiente figura se mostrará los componentes que integran la aplicación web, y la comunicación que existe.

En esta solución intervienen tres componentes, el servidor Elasticsearch, que actuará como base de datos y también para la búsqueda de información. El servidor de aplicación web, que será creado con el lenguaje de programación *JavaScript* a través de la utilización de un *Framework BackEnd* (*Express.js*) y, la interface del cliente, que será creada mediante el uso de *HTML* y *JavaScript*.

Las principales librerías y middlewares que se utilizarán son: *elasticsearch.js*, para realizar las peticiones CRUD al servidor *Elasticsearch*, y, *Google Charts* del lado del cliente, para mostrar la información de forma gráfica.

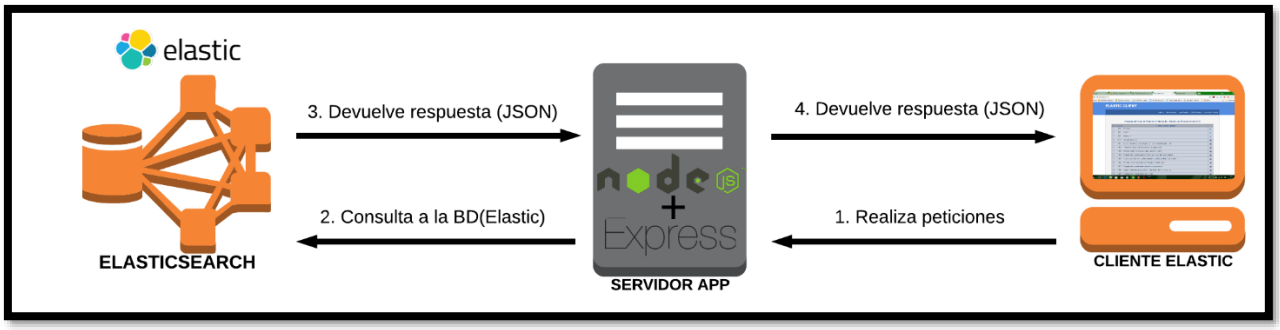


Figura 36. Solución para Aplicación Web (Elastic Client)

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

3.6. Diagrama general de la solución

En el siguiente diagrama se detallan los componentes mencionados en el punto 0.

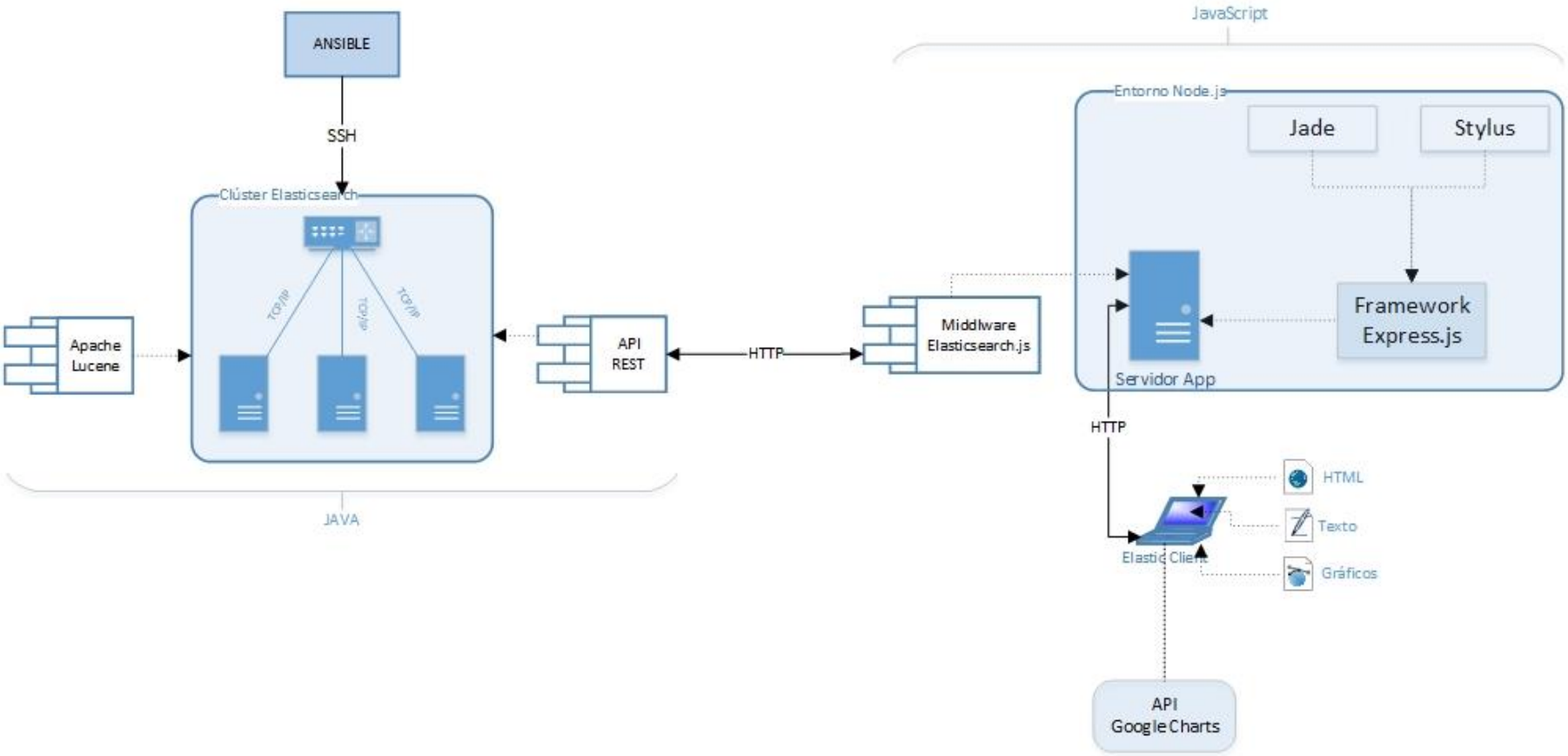


Figura 37. Diagrama general de la solución

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

3.7. Diagrama de arquitectura de software

Este diagrama ofrece más detalles sobre la relación que existe entre los componentes y las capas de software.

El diagrama (Figura 38), muestra la relación que tiene el usuario con la aplicación, el cual interactúa a través de interfaces (GUI), posterior a esto, la aplicación se comunicará con Elasticsearch realizando peticiones de indexado y consulta.

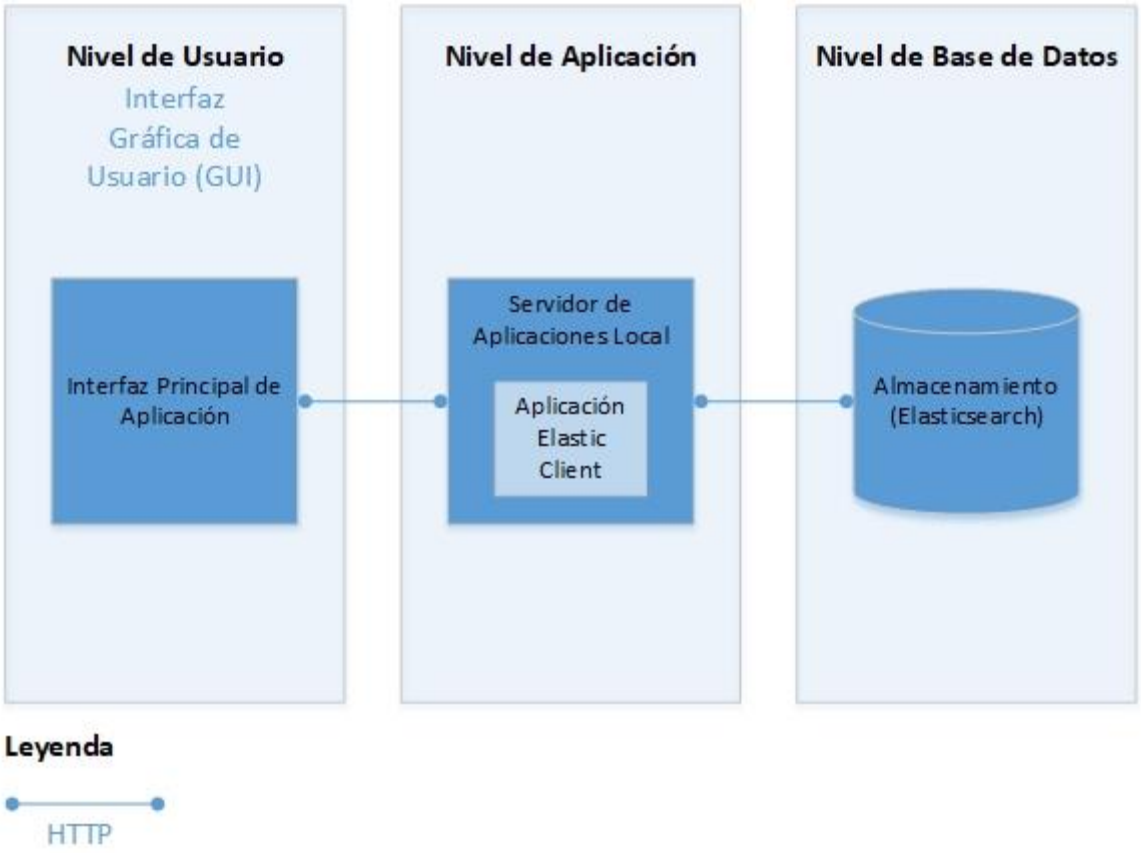


Figura 38. Diagrama de arquitectura de software de la aplicación web

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

3.8. Modelado de datos

En la actualidad no existen reglas, procedimientos u operaciones que estandaricen como se debe modelar los datos que se van a almacenar en una base de datos NoSQL orientada a documentos, por lo tanto, para realizar el modelado de datos de este proyecto se lo hizo en base a la experiencia y recomendación de personas que llevan más tiempo haciendo uso de base de datos

orientadas a documentos; además, se aplican ciertos conceptos como: denormalización, embebido y referencia de datos, dependiendo de la estructura de nuestros datos.

A continuación, se detalla cómo están estructurados los datos del proyecto y se muestra de manera general el índice, los tipos y sus relaciones.

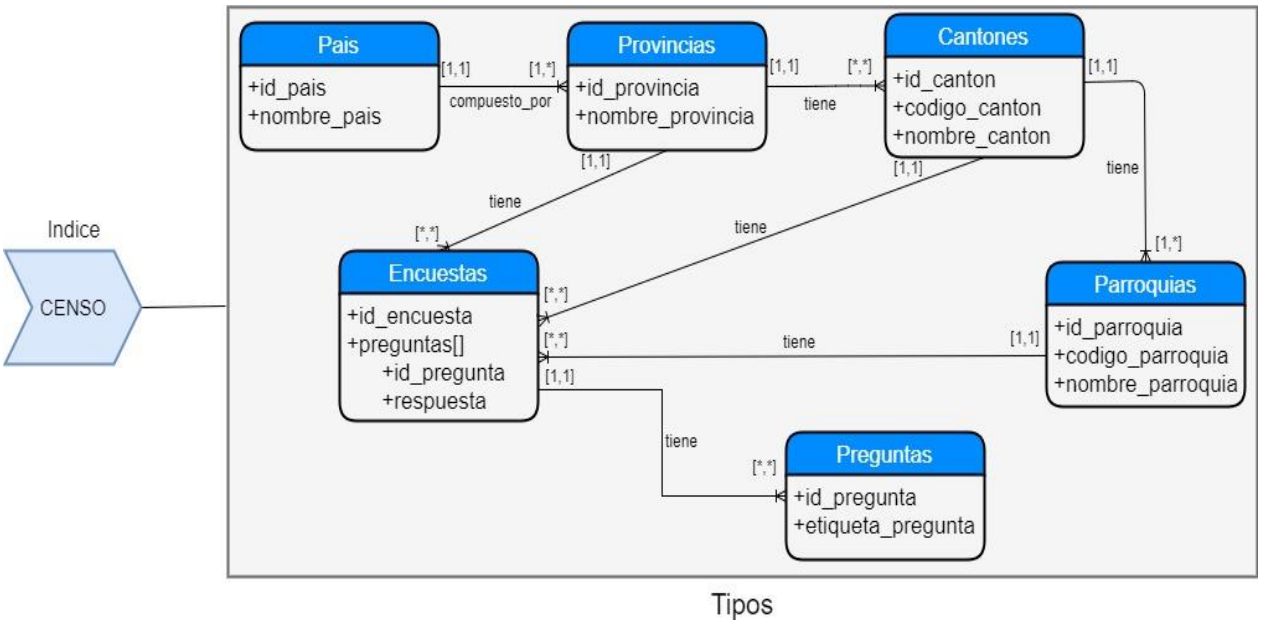


Figura 39. Estructura de los datos del proyecto (Censo de Población y Vivienda del año 2010)

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Si bien es cierto *Elasticsearch* es un motor de búsqueda, no hay que olvidarse que utiliza *Apache Lucene* para el indexado y consulta. *Lucene* utiliza el concepto de documentos, por lo que implícitamente se sobrentiende que es una base de datos NoSQL. Pero el hecho de que sea así, no significa que no se puedan relacionar los datos dentro de Elasticsearch, por el contrario, permite trabajar con cualquier tipo de datos, ya sea estructurados o no estructurados; siempre y cuando se puedan extraer, esa flexibilidad es precisamente uno de los puntos a favor que ha hecho tan famoso al concepto de manejar documentos con grandes cantidades de datos.

En el ANEXO C, se detalla el modelado de los datos expresados en formato JSON. Se puede decir entonces, que los documentos de los tipos “provincias”, “cantones” y “parroquias”, utilizan el concepto de **datos referenciados**, debido a que se hace referencia a datos de otros tipos dentro del mismo índice a través de identificadores (códigos), todo esto, para establecer las diferentes relaciones que existen entre estos tipos como se mostró en el diagrama de la Figura 39. ¿Por qué se utiliza el concepto de referencia de datos y no, datos embebidos? Bueno,

considere la relación “provincia-cantón”, tanto las provincias como los cantones están dados por un número fijo, por lo que no resulta “costoso” a la hora de hacer consultas, por ejemplo: ¿Qué cantones pertenecen a una provincia?

En el ejemplo del documento de tipo “encuestas”, se utiliza el concepto de **datos embebidos** para establecer la relación que existe entre “encuestas” y “preguntas”, todo esto, con el fin de agilizar y minimizar los tiempos de respuesta de las consultas, debido a que se trata de millones de encuestas. En este punto es donde también se aplica la denormalización porque se está redundando datos del tipo “preguntas”, pero es necesario ya que es un concepto ligado a las bases de datos orientas a documentos y es precisamente aquí donde se marca la diferencia en milisegundos en los tiempos de respuesta a las consultas. Al embeber los datos se ahorra muchos milisegundos, por ejemplo, para responder a la consulta ¿Devuelva todas las encuestas con sus respectivas preguntas y respuestas?, bastaría con hacer una sola consulta al tipo “encuestas”, por otro lado, si se utiliza datos referenciados, habría que hacer un “*join*” a la consulta, lo que aumentarían los tiempos de respuesta.

3.9. Mapeo de datos

Previo a la revisión de los conceptos de Elasticsearch en el punto 1.1.7, aquí se define cómo el motor de búsqueda va a tratar los documentos al momento de indexar, tomando como base la estructura que se estableció previamente en el modelado de datos en el punto 3.8.

Para mapear la estructura del índice completamente se utiliza el API REST de Elasticsearch, haciendo uso ya sea de comandos *cURL* por consola, mediante la ayuda de una herramienta externa a Elasticsearch, complementos para monitorear Elasticsearch disponibles en la web como*: Elasticsearch Head, Kopf, Elastic HQ*, etc, o en este caso; con la ayuda de la *API elasticsearch.js*, para Node.js, utilizándola directamente desde la aplicación a crear. En el ANEXO D se muestra el mapeo estático realizado para los datos, es importante revisarlo porque se explica más a detalle este proceso.

3.10. Indexación de datos

Una vez que se tiene el índice y tipos completamente mapeados acorde a la estructura de los datos, el siguiente paso para este proyecto es la indexación de los datos, para ello, también se hará uso de la API elastisearch.js.

Los datos están en archivos con extensión *.csv,* por lo que ahora a través de la aplicación se deberá extraer los datos e irlos almacenando en Elasticsearch. A continuación, se muestra un extracto de código con el cual se extrae los datos y se almacena en el clúster:

```
var csv = require('ya-csv');
```

El API “ya-csv”, permite extraer datos de archivos .csv.

cargar.js

```
21          type: "encuestas2",
22          requestTimeout: 700000
23      },function (error, response){
24          i = response.count;
25          //Modulo para extraer ficheros csv
26          var archivoEncuestas = "./CSV/"+ req.files.archivoEncuestas.name;
27          //Se valida que el tipo de archivo es un .csv sino se presenta mensaje de error
28          if(req.files.archivoEncuestas.mimetype == "text/csv"){
29              //Leemos archivo separado por comas csv
30              var encuestas = csv.createCsvFileReader(archivoEncuestas);
31              //Leemos cada columna del archivo csv
32              encuestas.setColumnNames(['I01','I02','I03','I04','I05','I09','I10','URH','H00','H01','H01N','H02','H03','H04','H05
33              //arreglo para almacenar temporalmente todas las encuestas, para posterior enviar en una solictud Bulk al servidor
34              var arrayEncuestas = [];
35              //leemos cada fila del archivo csv encuestas
36              encuestas.on('data', function(filas) {
37                  if (i != 0) {
38                      //Agregando al arrayEncuestas todas las encuestas del archivo leido
39                      //Primero se envia los campos de Metadata como index, type, y el id
40                      arrayEncuestas.push({index: {_index: index, _type: 'encuestas2', _id: i}});
41                      //Luego se envian los atributos a guardar el documento
42                      arrayEncuestas.push({id_encuesta: i, encuestas2: [ {id_pregunta: "I01", respuesta: filas.I01 }, {id_pregunta
43                  };
44                  i++;
45              }).on('end', function() {
46                  i=0;
47                  //Se envia a traves del metodo bulk(carga masiva), una peticion bulk con el arreglo que contiene todas las encu
48                  elasticsearch.elasticCliente().bulk({
49                      requestTimeout: 700000,
50                      body: arrayEncuestas
51                  }, function (err, resp) {
52                      if (err){
53                          ensajeEnv = "No se pudo cargar el archivo porque existió un error";
54                          console.log(err);
55                          res.render('cargarArchivo', { title: "Elastic Client", mensaje: mensajeEnv});
56                      }else{
57                          mensajeEnv = "Los datos del archivo enviado, fueron indexados correctamente";
58                          console.log(mensajeEnv);
59                          res.render('cargarArchivo', { title: "Elastic Client", mensaje: mensajeEnv});
60                      }
61                  });
```

Figura 40. Código para extraer los datos de los archivos .csv

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Línea 26: se captura el archivo enviado a través del método POST desde la interfaz GUI del cliente.

Línea 30: se lee el archivo .csv y se almacena en un objeto.

Línea 32: se lee todas las columnas del archivo.

Línea 34: se crea arreglo que almacena temporalmente, objetos JSON con cada “encuesta”.

Línea 36: evento que almacena los datos del archivo .csv.

Línea 40: se agrega primer al arreglo los campos de *metadata* del índice.

Línea 42: luego se agrega al arreglo los datos de los archivos .csv, con la estructura mapeada para los documentos de las encuestas.

Línea 45: evento que determina que se terminó de leer todas las filas del archivo .csv.

Línea 48: se utiliza el método *bulk(),* para la carga masiva de datos. Este método permite enviar en una sola solicitud al servidor Elasticsearch muchas solicitudes ya sea de indexación, edición o eliminación de datos.

Línea 50: se envía en el *body,* el arreglo que contiene los objetos JSON con todos los datos de las encuestas.

Línea 51: en una función de retorno, se recibe ya sea una respuesta o error por parte del servidor Elasticsearch. Las siguientes líneas se encargan de dar una la respuesta al cliente dependiendo de lo antes mencionado.

Importante: Teniendo en cuenta el mapeo previo realizado para almacenar los documentos de este tipo (encuestas) y luego de realizar la indexación de las mismas; al momento de monitorear los documentos indexados con alguna herramienta resulta que el número de documentos es mayor al que se envió previamente a almacenar, esto es debido al mapeo del índice y tipo. Cuando se mapea un documento como un arreglo de objetos (*nested*), Apache Lucene trata cada objeto como un documento independiente, esto es lo que hace posible que luego se puedan realizar las búsquedas sobre un documento anidado. Por ejemplo: en este caso se enviaron 3’815.528 filas de datos provenientes de los archivos, pero se almacenaron en el servidor 133’544.470 millones de documentos.

3.11. Aplicación web (Elastic Client)

Para el desarrollo de la solución web, que consumirá los datos almacenados anteriormente en Elasticsearch, se ha empleado un editor de texto denominado *Bracktes*. La interfaz gráfica de la herramienta se muestra a continuación:

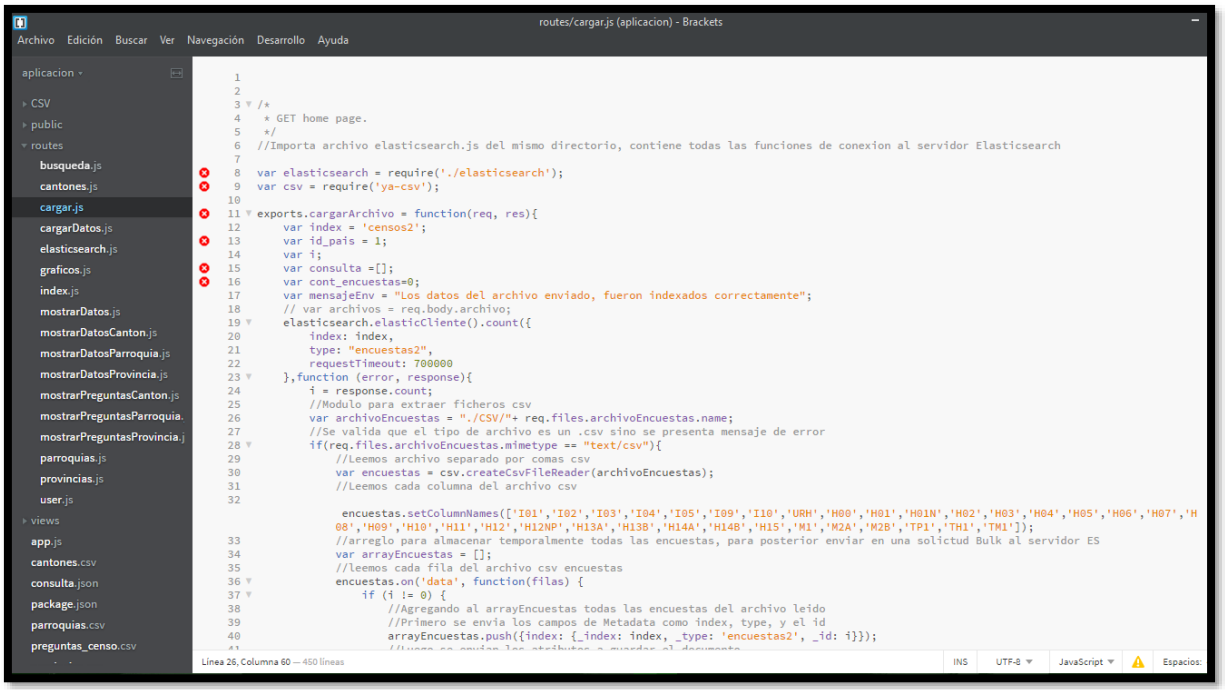


Figura 41. Interfaz gráfica *Brackets*.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

- **Lenguaje de programación y entorno de ejecución**

Uno de los objetivos del presente proyecto es realizar la aplicación web mediante JavaScript como lenguaje de programación base "*BackEnd*", para ello, se ha utilizado *Node.js,* un entorno de ejecución del lado del servidor que utiliza el motor V8, utilizado para interpretar el código JavaScript en el navegador Google Chrome.

Por otro lado, Node.js utiliza NPM, mediante este importante gestor de paquetes se puede descargar de manera automática las dependencias (*middlewares*) que utilizará la aplicación en desarrollo, además, permite ejecutar instrucciones como iniciar o apagar el entorno.

El lenguaje de programación Javascript también se ha utilizado del lado del cliente, para extraer la respuesta enviada desde el servidor y, para generar las estadísticas web con la aplicación Google Charts.

- **Framework**

Un framework a breves rasgos se puede definir como, una estructura tecnológica a nivel de software, la cual contiene artefactos o módulos que sirven de base para el desarrollo de software.

En el universo del desarrollo web existen muchos frameworks, sin embargo, para desarrollar este proyecto se utilizó *Express.js,* porque se ajusta a los requerimientos de la aplicación y, además, por ser una tecnología de código abierto. Adicional a esto, es importante destacar que, *Express.js* utiliza el *API Jade*, que es un lenguaje de plantillas HTML, permitiendo así simplificar las etiquetas. Otro punto a tomar en cuenta es la utilización de *Stylus* por parte del framework en mención, este importante preprocesador ayuda a crear de hojas de estilo de manera eficiente, rápida y dinámica.

- **Diseño de las interfaces graficas de usuario (GUI)**

El diseño de las interfaces gráficas de usuario es importante en cuanto a la facilidad de uso de la aplicación, ya que, de ello depende en su totalidad que la aplicación sea intuitiva para el usuario.

El diseño de la parte principal de la aplicación, lista las preguntas realizadas en el Censo de Población y Vivienda del Ecuador en el año 2010, además permite al usuario consultar y mostrar el resumen de los datos tanto numérica como gráficamente de una pregunta en particular. Esta misma filosofía de consulta se maneja en toda la aplicación, con la diferencia que, también se puede resumir los datos ya sea por provincia, por cantón y por parroquia. El manual de usuario se encuentra disponible en el ANEXO E, donde se explica más a detalle cada interfaz y uso de la misma.

3.12. Lectura de datos (consultas realizadas)

Como se podrá haber dado cuenta, previo a la revisión del punto anterior, el funcionamiento de la aplicación web se basa principalmente en la consulta de datos, donde, la interacción del usuario es requerida básicamente para seleccionar dentro de varias opciones, que datos quiere mostrar. A continuación, en el ANEXO F se explica las diversas consultas realizadas dentro de la aplicación para consultar el resumen de los datos, agrupados ya sea: de manera general, por provincia, por cantón y por parroquia, así como también, consultas básicas realizadas para mostrar: preguntas del censo, provincias, cantones y parroquias.

CAPITULO IV

PRUEBAS Y VALIDACIÓN DE RESULTADOS

4.1. Introducción

Una vez que se ha montado el servidor y se ha realizado la aplicación que consume los datos almacenados en Elasticsearch, es necesario realizar pruebas no funcionales que determinen especialmente el rendimiento y disponibilidad del clúster. El presente capítulo está dedicado a la realización de pruebas especialmente para medir eventualmente la carga masiva, así como las diversas solicitudes de lectura que se hacen al servidor Elasticsearch, para observar su comportamiento.

Entre los objetivos del proyecto no está contemplado instalar y montar el servidor en un ambiente de producción, por lo que, las pruebas que se realizarán serán únicamente para verificar la disponibilidad y rendimiento del servidor. También se hace un análisis de los tiempos de respuesta obtenido de las pruebas para sacar algunas conclusiones.

Para el diseño, creación y ejecución de las pruebas se utilizará Apache JMeter (Ver 1.4), que permite realizar pruebas de rendimiento.

4.2. Objetivo de las pruebas

Las pruebas a realizar tienen como finalidad medir el rendimiento a la hora de indexar datos de forma masiva, medir a través de métricas los tiempos de respuesta a las solicitudes de lectura y, validar la disponibilidad del clúster Elasticsearch ante una eventual caída de un nodo.

4.3. Plan de pruebas

El siguiente plan de pruebas será empleado para medir el rendimiento y la disponibilidad del clúster Elasticsearch.

Tabla 16. Plan de pruebas para medir el rendimiento y disponibilidad del clúster

ID. Prueba	Componente a ser probado	Tipo de Prueba	Objetivo	Número de Nodos	Hilos recurrentes	Datos a enviar o recibir
PR1	Carga masiva de datos (API Bulk de Elasticsearch a través de la aplicación)	Carga	Determinar la cantidad máxima de solicitudes de indexación de encuestas.	3	1	30.000 datos a enviar
PR2	Carga masiva de datos (API	Carga	Determinar la cantidad	3	1	50.000 datos a enviar

	Bulk de Elasticsearch a través de la aplicación)		máxima de solicitudes de indexación de encuestas.			
PR3	Carga masiva de datos (API Bulk de Elasticsearch a través de la aplicación)	Estrés	Determinar la cantidad máxima de solicitudes de indexación de encuestas.	3	1	70.000 datos a enviar
PR4	Solicitud de consulta a todos los datos de las encuestas mediante API REST	Rendimiento	Determinar los tiempos de respuesta a la solicitud con los 3 nodos.	3	1	133’544.470 Documentos a recibir
PR5	Solicitud de consulta a todos los datos de las encuestas mediante API REST	Rendimiento y Disponibilidad	Determinar los tiempos de respuesta a la solicitud con Nodo1 y Nodo2	2	1	133’544.470 Documentos a recibir
PR6	Solicitud de consulta a todos los datos de las encuestas mediante API REST	Rendimiento y Disponibilidad	Determinar los tiempos de respuesta a la solicitud con Nodo1 y Nodo3	2	1	133’544.470 Documentos a recibir

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Es importante aclarar que el plan fue diseñado para, la indexación masiva de datos, para lo cual, no se necesitan muchos hilos recurrentes (usuarios conectados simultáneamente), ya que el API Bulk es la que permite realizar los miles de solicitudes de indexación al servidor, es por ello que la petición HTTP se hace directamente a la aplicación, que es la encargada de hacer este proceso. Así mismo, para las consultas (Rendimiento y Disponibilidad) tampoco se necesita realizar pruebas con varios usuarios recurrentes, debido a que se busca medir y observar el

rendimiento del servidor utilizando, ya sea, ambos nodos de datos a la vez o uno a la vez. No se olvide que el clúster está compuesto de tres nodos, uno dedicado a atender las solicitudes y dos dedicados a procesar las mismas.

4.4. Diseño de pruebas en apache JMeter

El diseño del escenario de pruebas en Apache JMeter, es posterior a la elaboración del plan de pruebas a ejecutar. Consta de 3 partes, El Grupo de Hilos, que hace referencia a la cantidad de usuarios que realizan peticiones recurrentes al servidor, La petición HTTP, que hace referencia a la conexión mediante dicho protocolo hacia el componente a probar. Los receptores, que son los encargados de almacenar los resultados luego de la ejecución de las pruebas.

Es importante destacar que, se hace la petición HTTP directamente al API REST del servidor Elasticsearch, debido a que se busca evitar la mayor cantidad de tiempo en latencia. El único escenario en que se necesitará de la aplicación web, será para la carga masiva de datos, debido a que por intermedio de la misma se extrae los archivos .csv y se carga los datos mediante el API Bulk, que a su vez realiza la conexión con el servidor Elasticsearch.

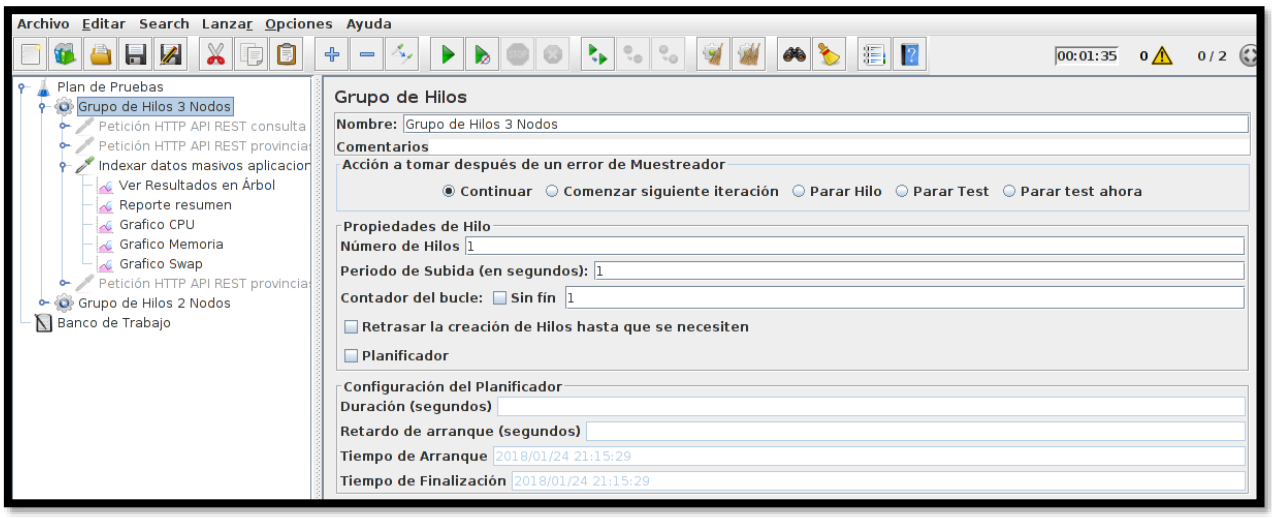


Figura 42. Diseño de pruebas en Apache JMeter

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

4.5. Resultados de las pruebas

4.5.1. Prueba PR1

Petición HTTP a la ruta *cargarArchivo* de la aplicación, que realiza la carga masiva con 30.000 datos.

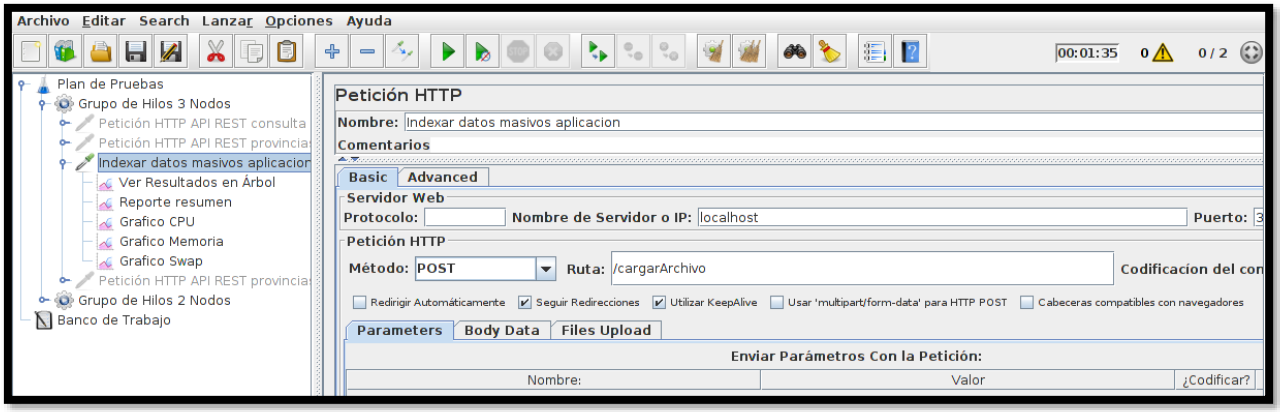


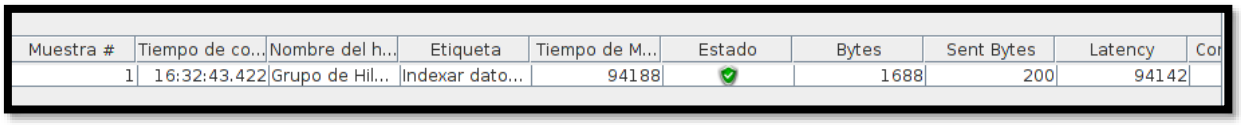
Figura 43. Petición HTTP a la ruta */cargarArchivo*

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta */cargarArchivo* de la aplicación web. Dicha ruta hace una solicitud por el método POST, el cual permite realizar la carga masiva de datos.

Resultados en Árbol



Muestra #	Tiempo de co...	Nombre del h...	Etiqueta	Tiempo de M...	Estado	Bytes	Sent Bytes	Latency	Cor
1	16:32:43.422	Grupo de Hil...	Indexar dato...	94188		1688	200	94142	

Figura 44. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros.

Rendimiento de la CPU de los 3 nodos.

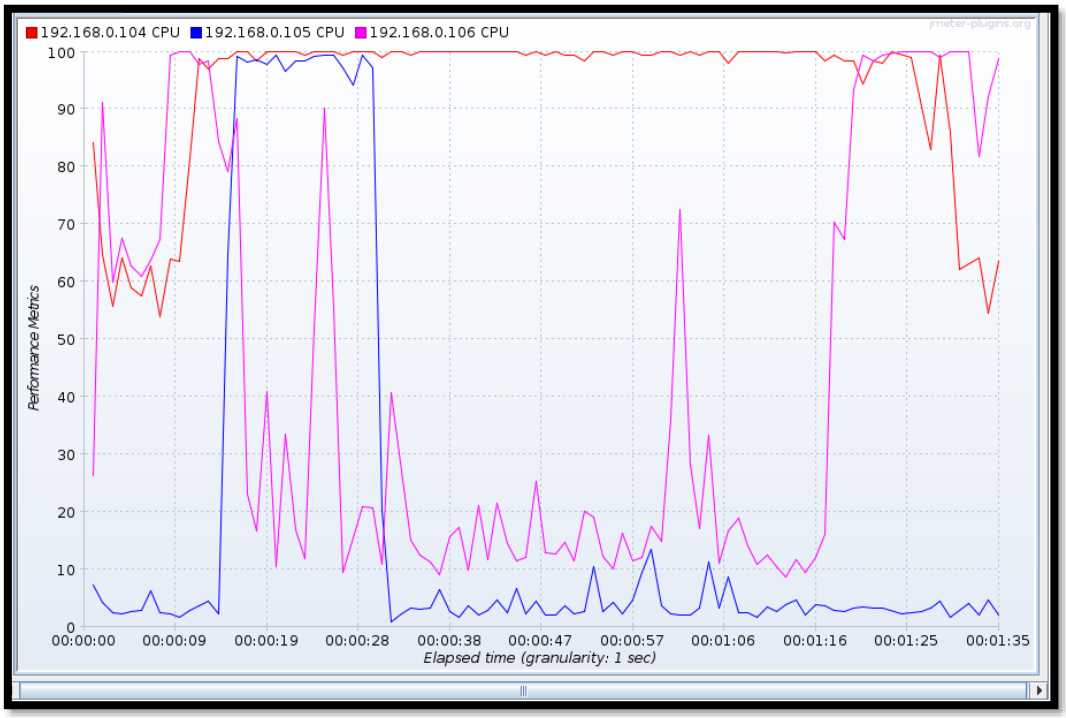


Figura 45. Rendimiento CPU, durante carga masiva con 30.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

El gráfico de la figura anterior, muestra una comparación de la ocupación de la CPU de los 3 nodos a lo largo de la petición de indexación con 30.000 datos. Se puede observar que, el Nodo2 está trabajando más que Nodo3 de datos y Nodo1 maestro.

Uso de la memoria RAM durante la carga masiva.

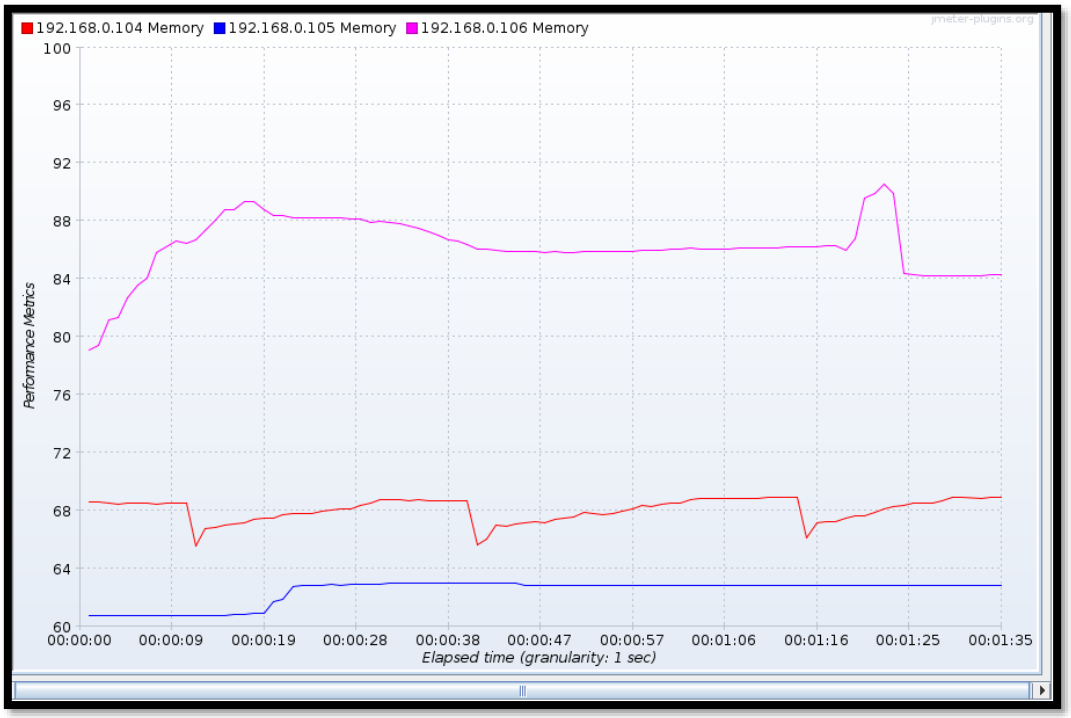


Figura 46. Uso de la memoria RAM, durante carga masiva con 30.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico la figura anterior, se muestra una comparación del uso de la memoria RAM de los 3 nodos. Donde se puede divisar que, el Nodo1 master, tiene un mayor uso de la memoria durante la indexación. Esto es debido a, que es el encargado de distribuir las operaciones de indexación a todos los nodos del clúster para que éstos las procesen.

Uso de la Swap durante la carga masiva con 30.000 datos.

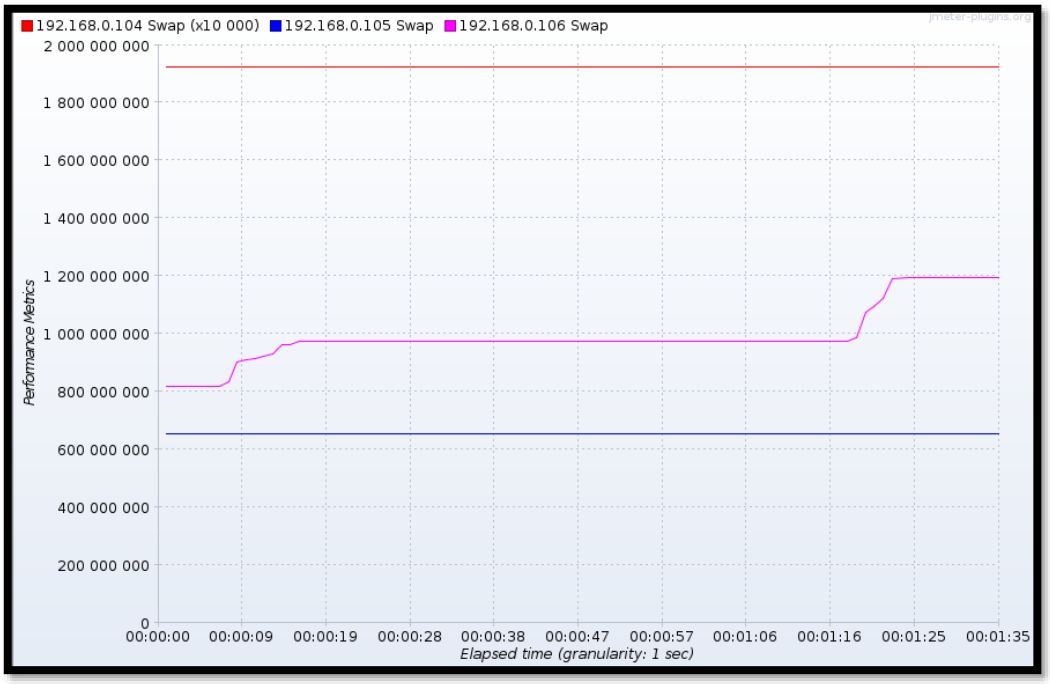


Figura 47. Uso de la Swap, durante carga masiva con 30.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico de la figura anterior, se muestra una comparación del uso de la *swap,* es decir el entorno JAVA asignado para Elasticsearch. Se puede observar que Nodo2 (rojo) hace mayor uso respecto a los otros dos nodos.

4.5.2. Prueba PR2

Petición HTTP a la ruta */cargarArchivo* de la aplicación, que realiza la carga masiva con 50.000 datos.

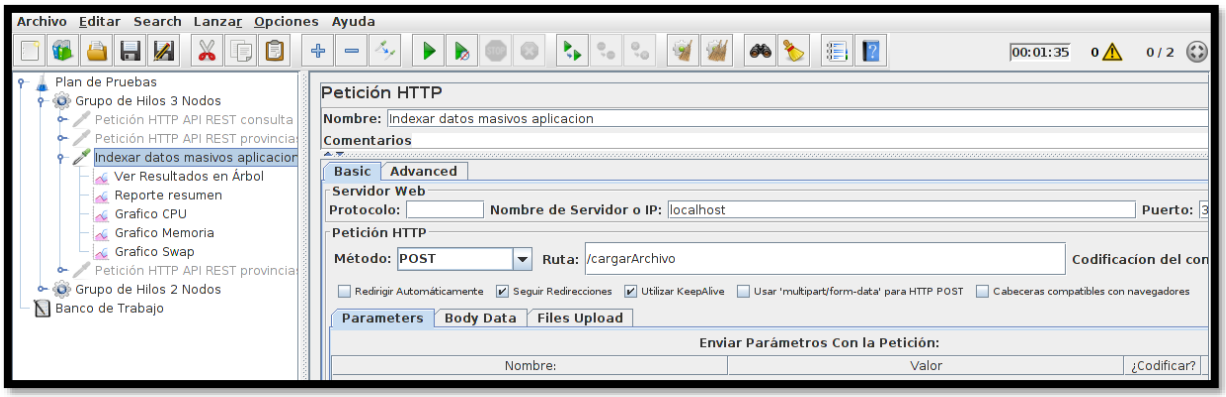


Figura 48. Petición HTTP a la ruta */cargarArchivo*

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta */cargarArchivo* de la aplicación web. Dicha ruta hace una solicitud por el método POST, el cual permite realizar la carga masiva de datos.

Resultados en Árbol

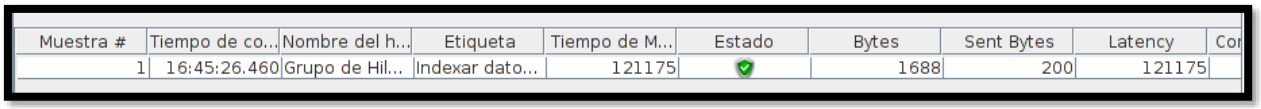


Figura 49. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros.

Rendimiento de la CPU de los 3 nodos.

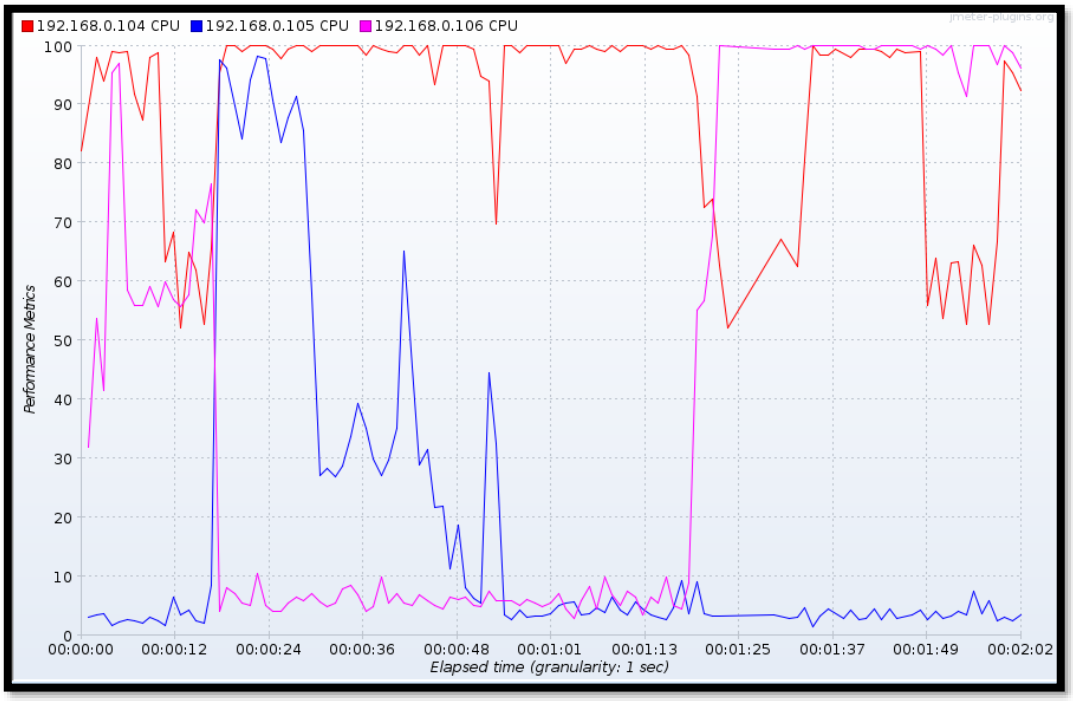


Figura 50. Rendimiento CPU, durante carga masiva con 50.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

El gráfico de la figura anterior, muestra una comparación de la ocupación de la CPU de los 3 nodos a lo largo de la petición de indexación con 50.000 datos. Se puede observar que, el Nodo2 está trabajando más que Nodo3 de datos y Nodo1 maestro. Aunque para este caso, también se observa que Nodo3 realiza un esfuerzo mayor que la prueba anterior.

Uso de la memoria RAM durante la carga masiva.

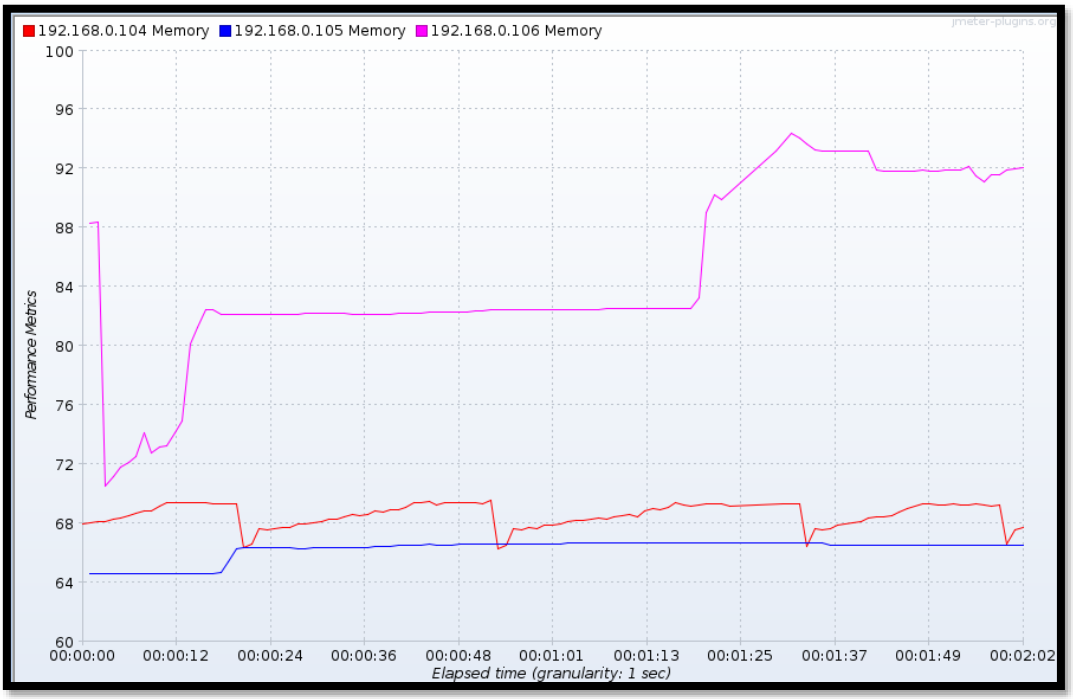


Figura 51. Uso de la memoria RAM, durante carga masiva con 50.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico la figura anterior, se muestra una comparación del uso de la memoria RAM de los 3 nodos. Donde se puede divisar que, el Nodo1 master, tiene un mayor uso de la memoria durante la indexación. Esto es debido a, que es el encargado de distribuir las operaciones de indexación a todos los nodos del clúster para que éstos las procesen.

Uso de la Swap durante la carga masiva con 50.000 datos.

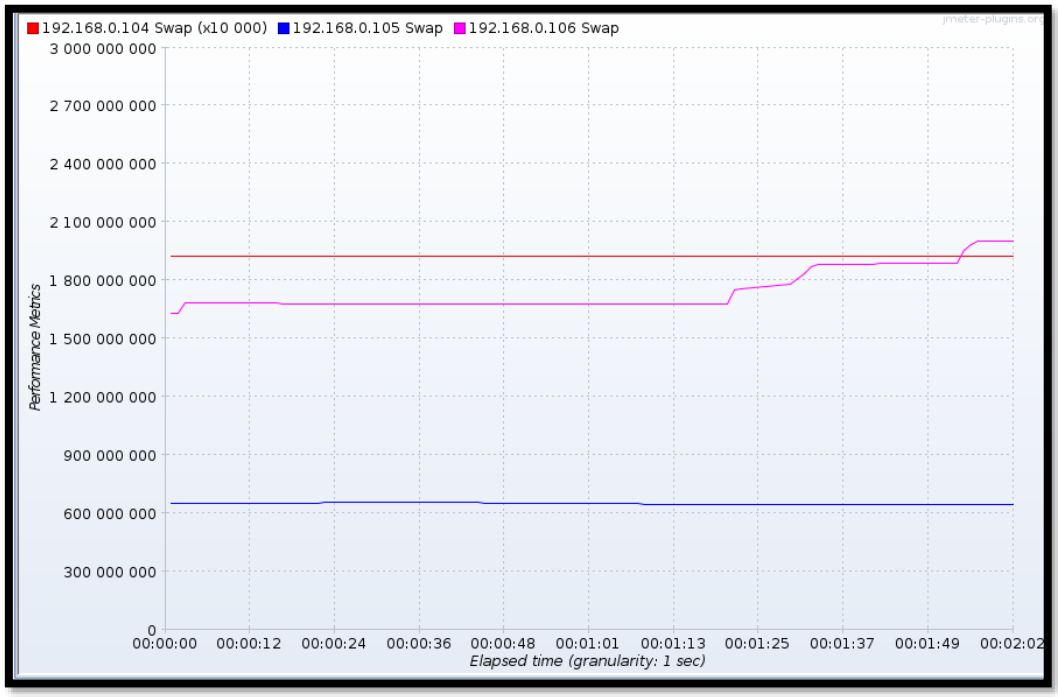


Figura 52. Uso de la Swap, durante carga masiva con 50.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico de la figura anterior, se muestra una comparación del uso de la *swap,* es decir el entorno JAVA asignado para Elasticsearch. Se puede observar que Nodo2 (línea roja) hace mayor uso respecto a los otros dos nodos.

4.5.3. Prueba PR3

Petición HTTP a la ruta */cargarArchivo* de la aplicación, que realiza la carga masiva con 70.000 datos.

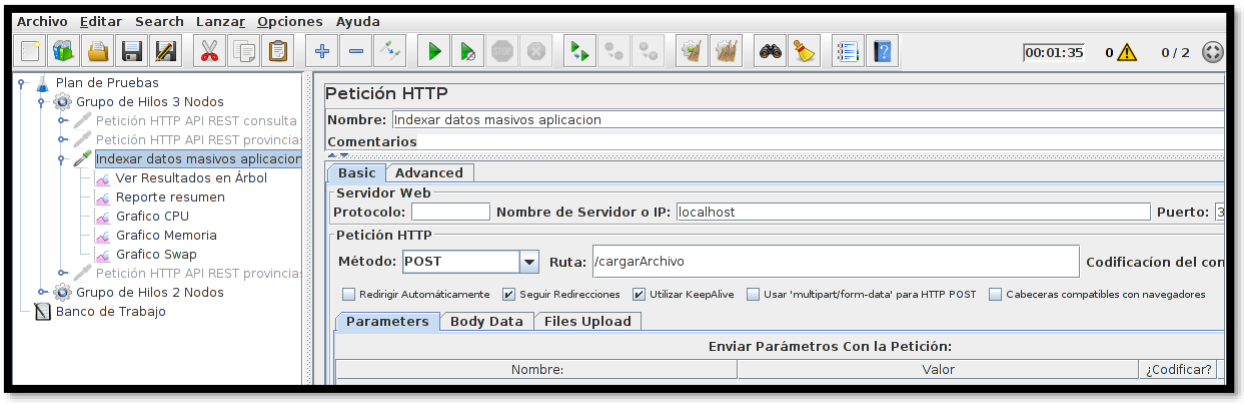


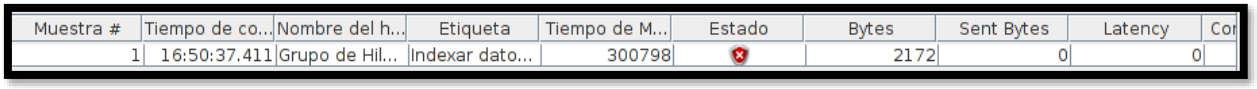
Figura 53. Petición HTTP a la ruta */cargarArchivo*

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta */cargarArchivo* de la aplicación web. Dicha ruta hace una solicitud por el método POST, el cual permite realizar la carga masiva de datos.

Resultados en Árbol



Muestra #	Tiempo de co...	Nombre del h...	Etiqueta	Tiempo de M...	Estado	Bytes	Sent Bytes	Latency	Cor
1	16:50:37.411	Grupo de Hil...	Indexar dato...	300798		2172	0	0	

Figura 54. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros. El estado de esta prueba da como resultado un error.

Rendimiento de la CPU de los 3 nodos.

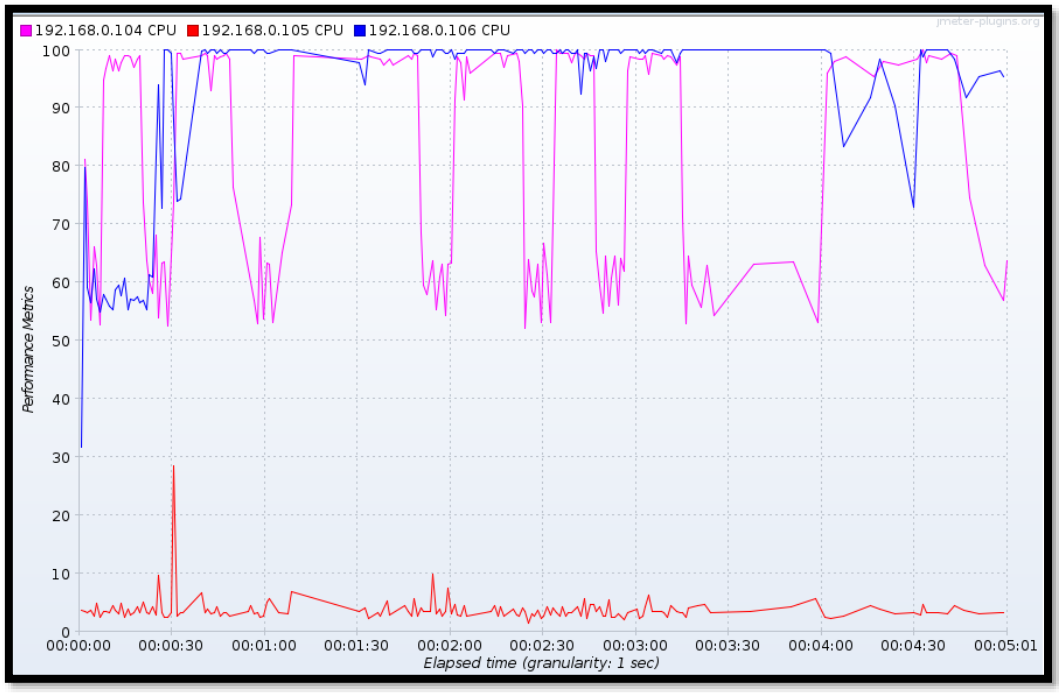


Figura 55. Rendimiento CPU, durante carga masiva con 70.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

El gráfico de la figura anterior, muestra una comparación de la ocupación de la CPU de los 3 nodos a lo largo de la petición de indexación con 70.000 datos. Se puede observar que, tanto Nodo2 (rosado) y Nodo1 (azul) han llegado a su límite de procesamiento. Como resultado ha esto es un error.

Uso de la memoria RAM durante la carga masiva.

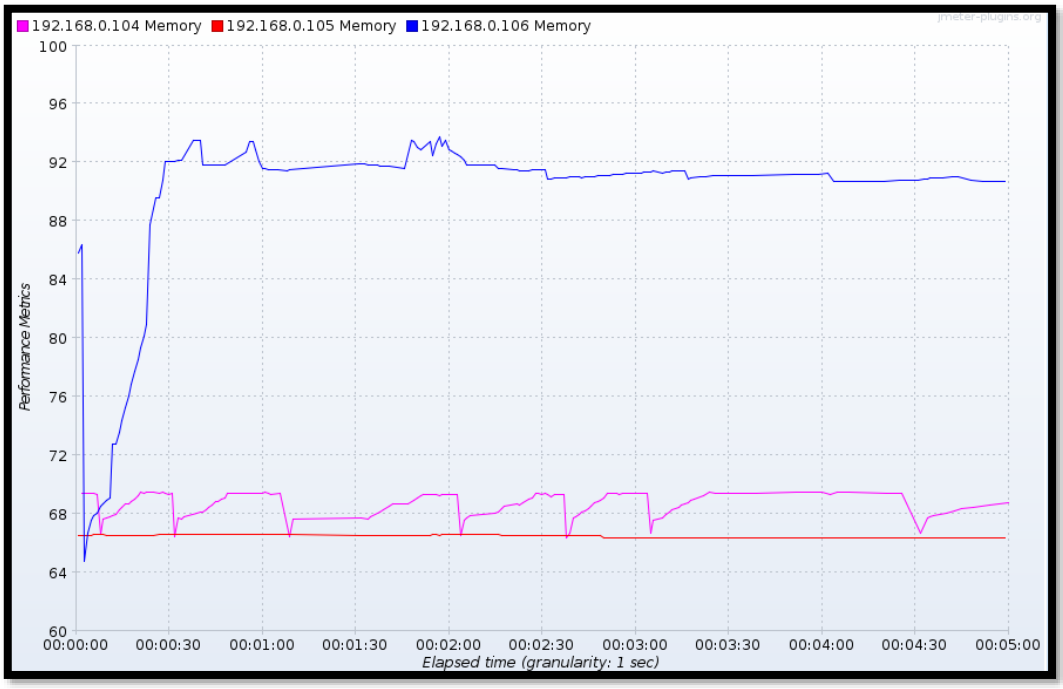


Figura 56. Uso de la memoria RAM, durante carga masiva con 70.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico la figura anterior, se muestra una comparación del uso de la memoria RAM de los 3 nodos. Donde se puede divisar que, prácticamente ambos nodos de datos (rosada y roja) ya no reciben las peticiones de indexación debido al error.

Uso de la Swap durante la carga masiva con 70.000 datos.

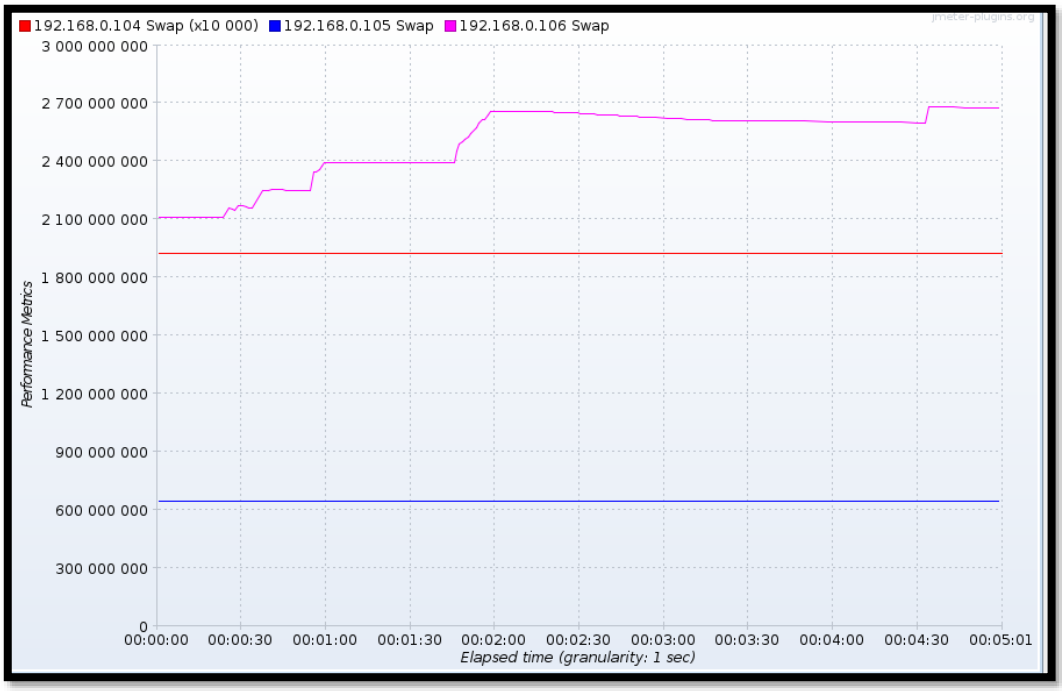


Figura 57. Uso de la Swap, durante carga masiva con 70.000 datos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En el gráfico de la figura anterior, se muestra una comparación del uso de la *swap,* es decir el entorno JAVA asignado para Elasticsearch. Se puede observar que Nodo1 (línea rosada) hace mayor uso respecto a los otros dos nodos.

4.5.4. Prueba PR4

Petición HTTP a la ruta */censos/encuestas2/_search* de la API REST de Elasticsearch, que realiza la consulta de todas las encuestas con un filtro de la pregunta "Área urbana o rural".

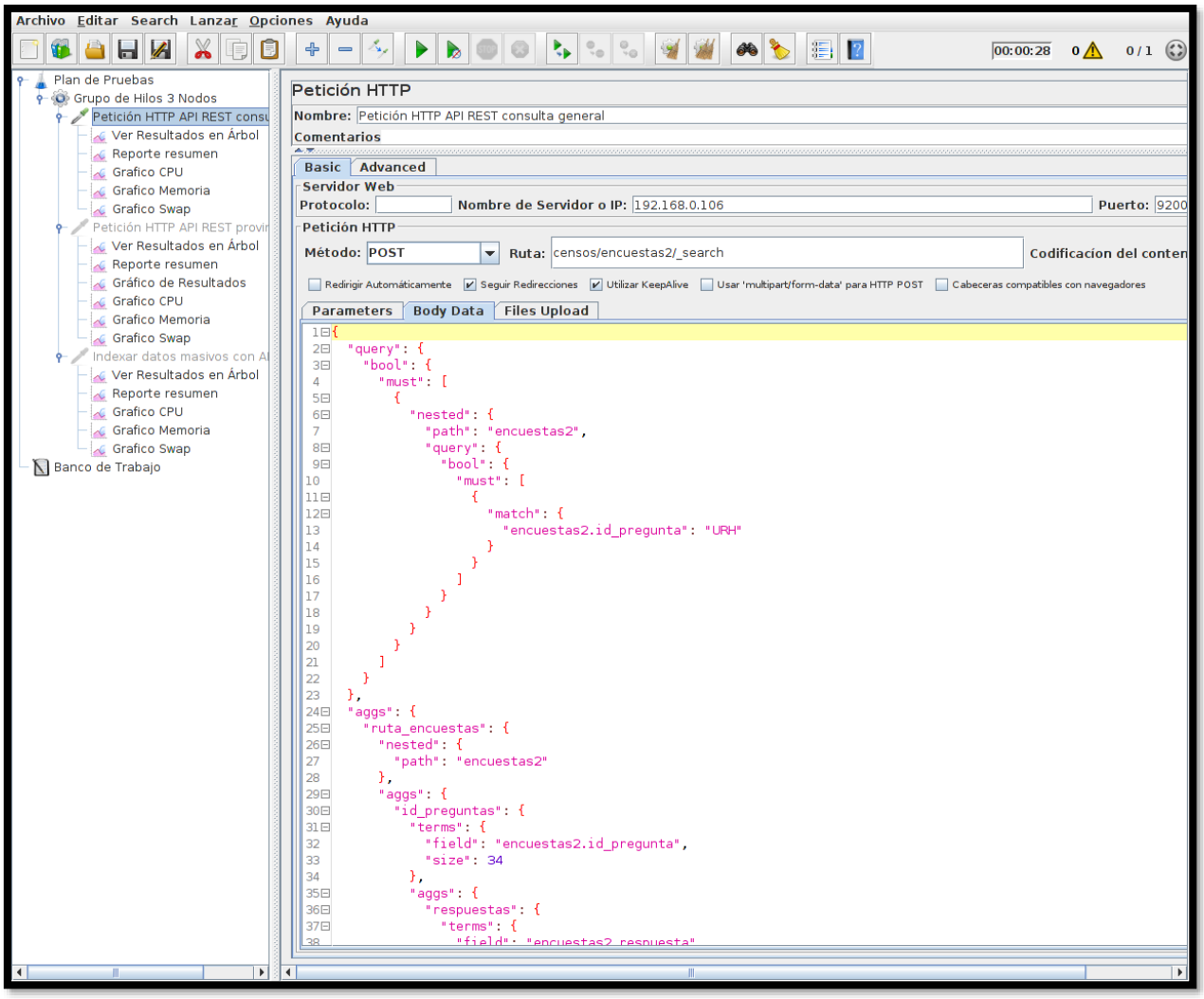


Figura 58. Petición HTTP a la ruta */censos/encuestas2/_search* de la API REST de Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta *censos/encuestas2/_search* del API REST de Elasticsearch. Dicha ruta hace una solicitud al servidor por el método POST, la cual permite realizar una consulta que solicita todas las encuestas de la pregunta “URH” o “Área urbana o rural”.

Resultados en Árbol

Muestra #	Tiempo de co...	Nombre del h...	Etiqueta	Tiempo de M...	Estado	Bytes	Sent Bytes	Latency	Conne
1	15:55:01.170	Grupo de Hil...	Petición HTT...	14458		19742	1156	14458	

Figura 59. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros.

Rendimiento de la CPU de los 3 nodos, al realizar la consulta.

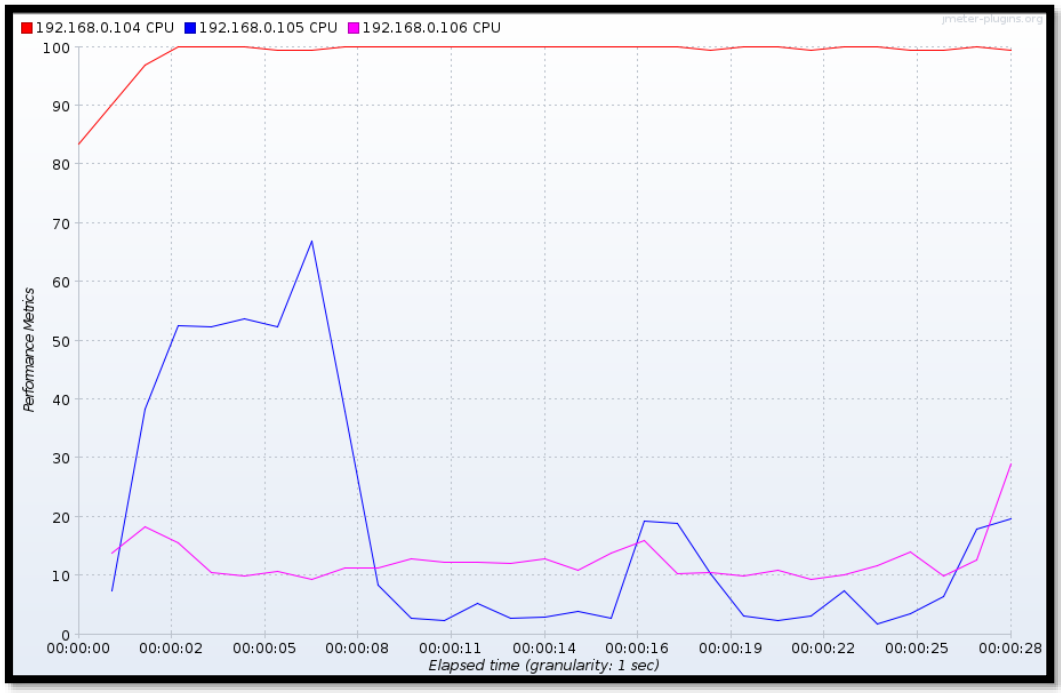


Figura 60. Rendimiento CPU, durante la consulta con 3 nodos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

El gráfico de la figura anterior, muestra una comparación de la ocupación de la CPU de los 3 nodos durante la consulta realizada, en el cual se puede observar que Nodo2 (rojo) de datos realiza un mayor esfuerzo para procesar la solicitud a diferencia de Nodo3 (azul) que llega a un máximo del 68% de utilización. Cabe recalcar que, Nodo1 (morado) es el nodo master por lo que

su función no es procesar la consulta sino enviar copias de la misma hacia todos los fragmentos que se encuentran en los nodos de datos.

Uso de la memoria RAM durante la consulta.

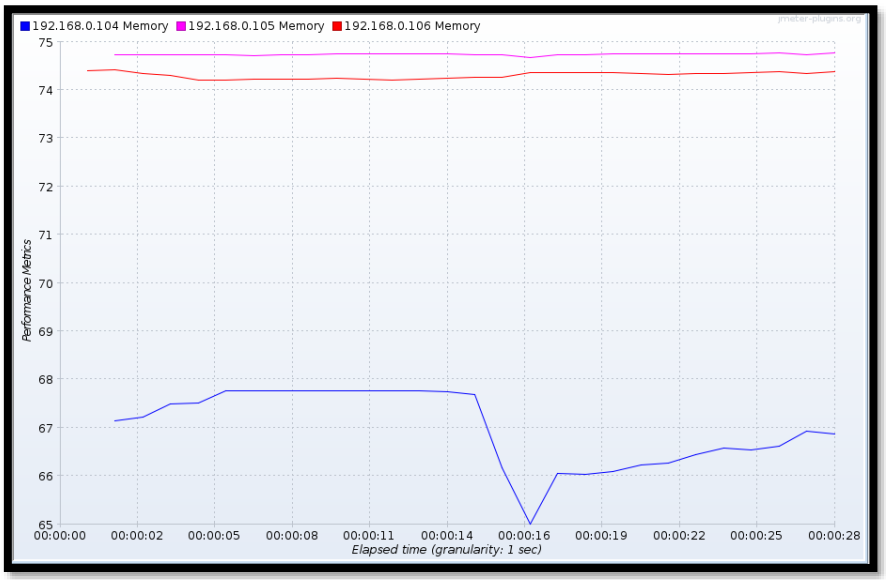


Figura 61. Uso de la memoria RAM, durante la consulta con 3 nodos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra una comparación del uso la memoria RAM de todos los nodos durante la consulta. Como se puede observar, la memoria de Nodo2 de datos fue usada en mayor porcentaje y tiempo (75%).

Uso de la Swap durante la consulta.

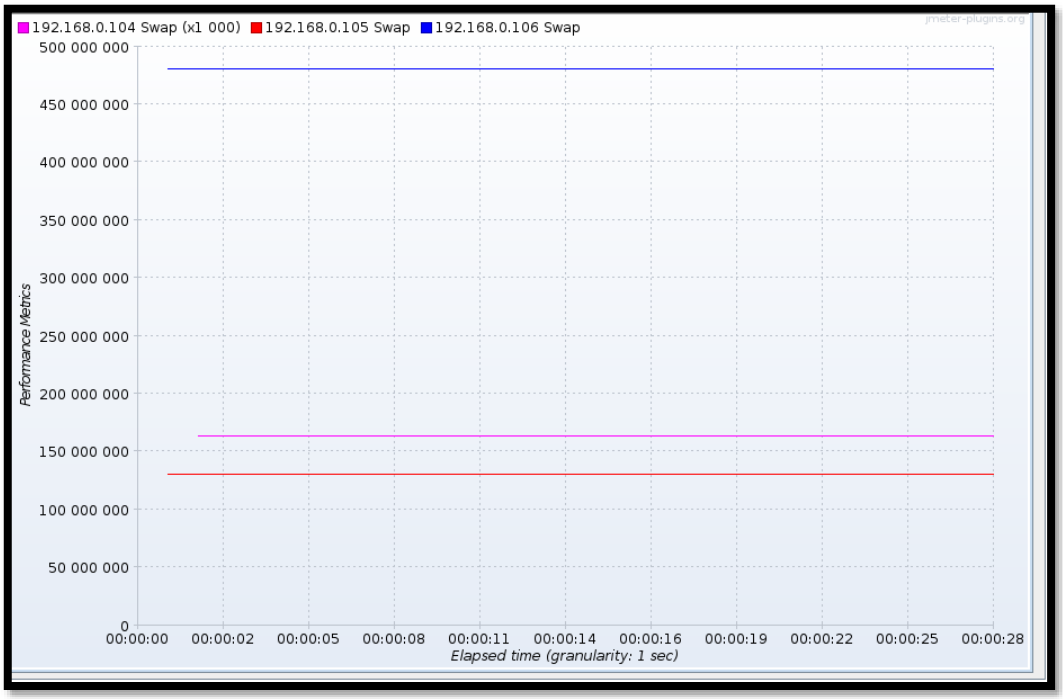


Figura 62. Uso de la Swap, durante consulta con 3 nodos.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra una comparación de la swap, recuerde que es el entorno JAVA asignado para Elasticsearch. Como se puede observar, la swap de Nodo1 (azul) está siendo usada en mayor porcentaje, esto es esperado, debido a que el nodo master es quien se encarga de distribuir las solicitudes a todos los nodos, esperar las respuestas de los nodos de datos y devolver los resultados obtenidos al cliente, por eso Nodo1 master ocupa la swap en mayor medida que los nodos de datos.

4.5.5. Prueba PR5

Esta prueba requiere únicamente del Nodo1 master y Nodo2 de datos, por lo que, la imagen que se muestra a continuación, es una prueba del monitoreo a través del API de Elasticsearch de los 5 fragmentos del indice “censos” asignados únicamente al Nodo2 de datos. Para este caso el Nodo3 de datos está fuera de servicio.

```
censos 3 p STARTED    26732520 762.8mb 192.168.0.104 nodo2
censos 3 r UNASSIGNED
censos 2 p STARTED    26720830 701.6mb 192.168.0.104 nodo2
censos 2 r UNASSIGNED
censos 4 p STARTED    26678994 719.9mb 192.168.0.104 nodo2
censos 4 r UNASSIGNED
censos 1 p STARTED    26722913 688.5mb 192.168.0.104 nodo2
censos 1 r UNASSIGNED
censos 0 p STARTED    26689213 695.7mb 192.168.0.104 nodo2
censos 0 r UNASSIGNED
```

Figura 63. Monitoreo de los fragmentos del indice censos, sólo al Nodo2

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Petición HTTP a la ruta *censos/encuestas2/_search* de la API REST de Elasticsearch, que realiza la consulta de todas las encuestas con un filtro de la pregunta "Área urbana o rural".

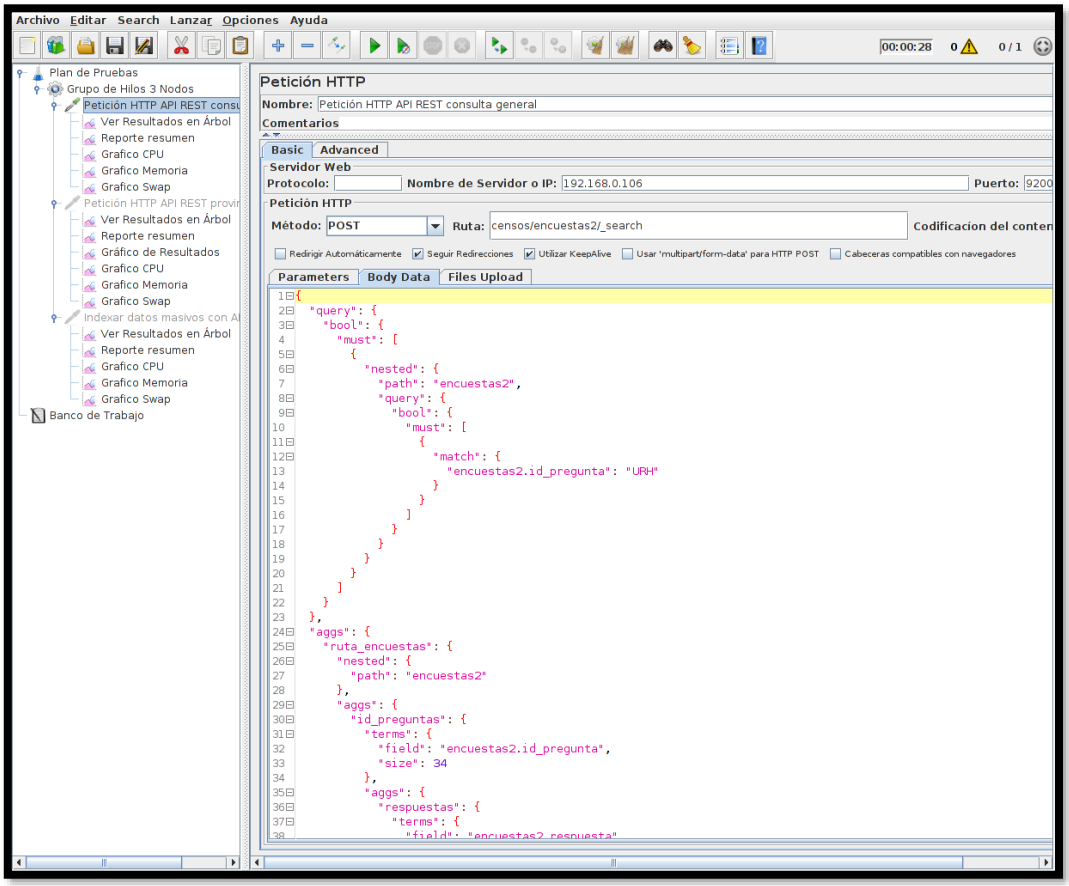


Figura 64. Petición HTTP a la ruta */censos/encuestas2/_search* de la API REST de Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta *censos/encuestas2/_search* del API REST de Elasticsearch. Dicha ruta hace una solicitud al servidor por el método POST, la cual permite realizar una consulta que solicita todas las encuestas de la pregunta “URH” o “Área urbana o rural”.

Resultados en Árbol

Muestra #	Tiempo de co...	Nombre del h...	Etiqueta	Tiempo de M...	Estado	Bytes	Sent Bytes	Latency	Conne
1	16:06:58.912	Grupo de Hil...	Petición HTT...	52933		19742	1156	52933	

Figura 65. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros.

Rendimiento de la CPU de Nodo1 y Nodo2, al realizar la consulta.

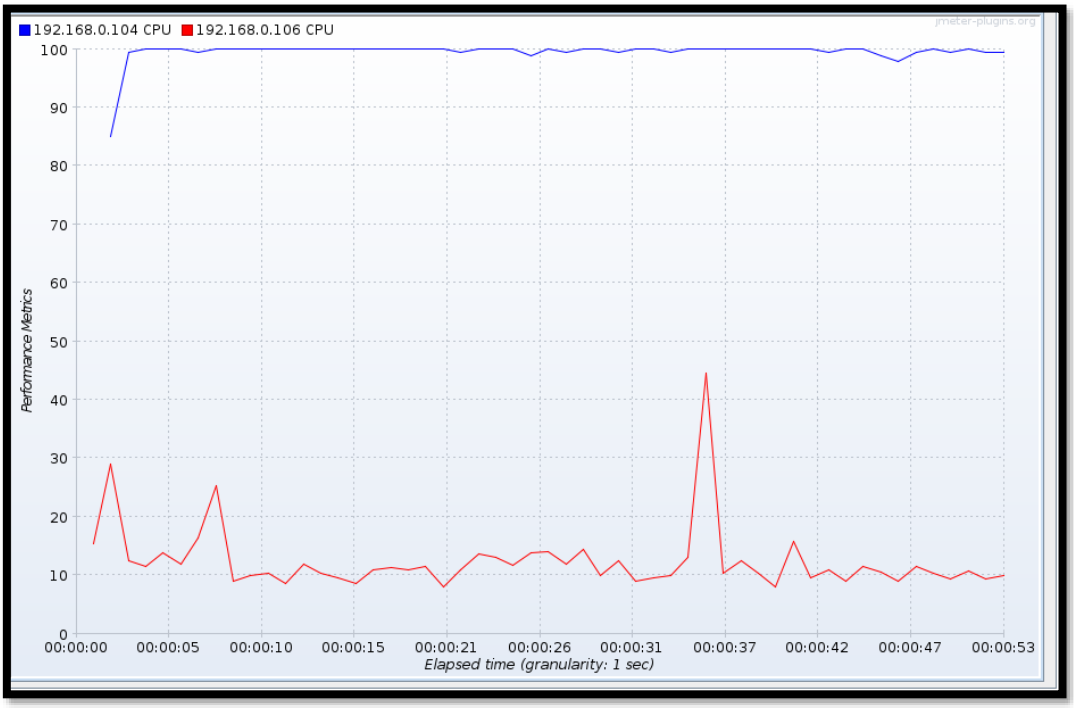


Figura 66. Rendimiento CPU, durante la consulta con Nodo1 y Nodo2.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra una comparación gráfica del uso de la CPU del Nodo1 (rojo) y Nodo2 (azul), que son los que están trabajando en la prueba. Como se puede observar el procesamiento es mayor (100%) para el Nodo2 debido ya que, es el único que se encarga de realizar la búsqueda.

Uso de la memoria RAM durante la consulta.

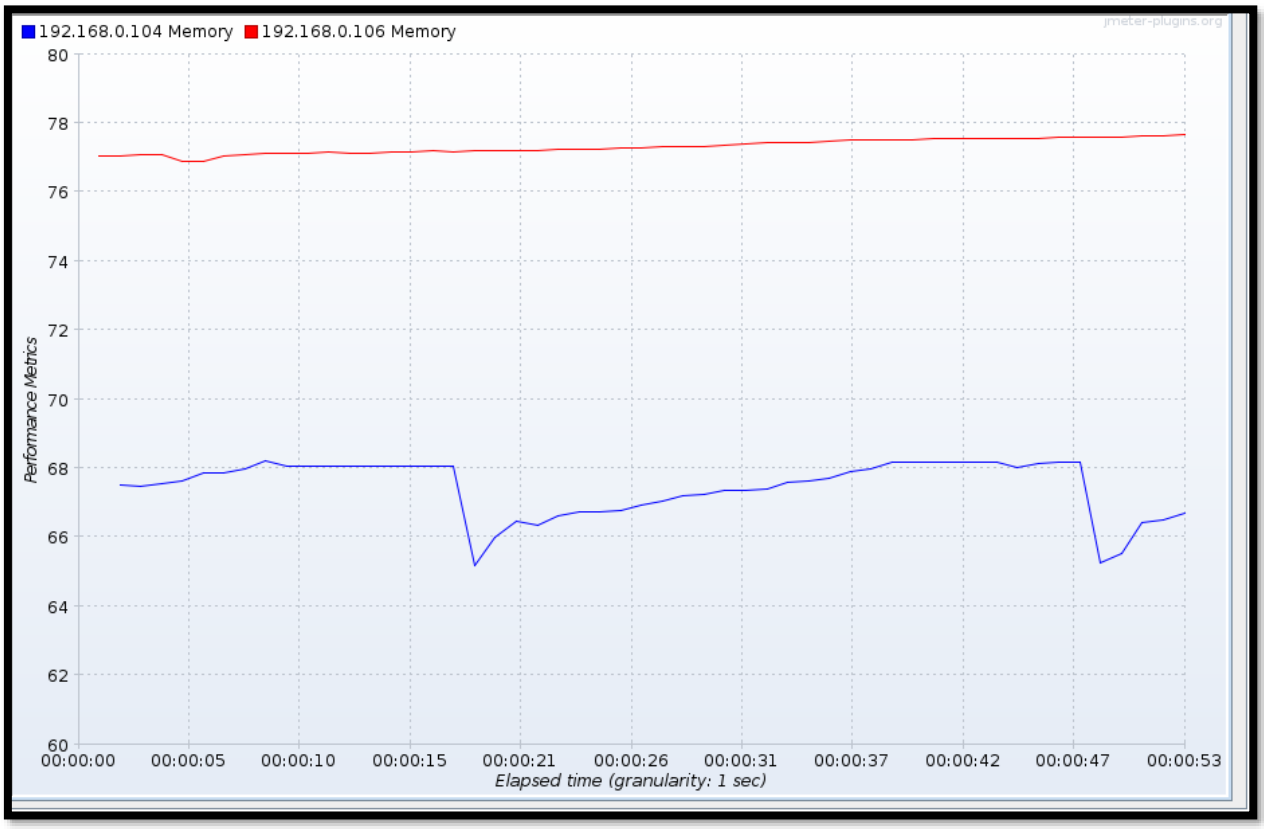


Figura 67. Uso de la memoria RAM, durante la consulta con Nodo1 y Nodo2.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la imagen anterior, se observa una comparación gráfica del uso de la memoria RAM por parte de Nodo1 (rojo) y Nodo2 (azul). En este caso se puede observar que, Nodo1 utilizó en mayor medida la memoria.

Uso de la Swap durante la consulta.

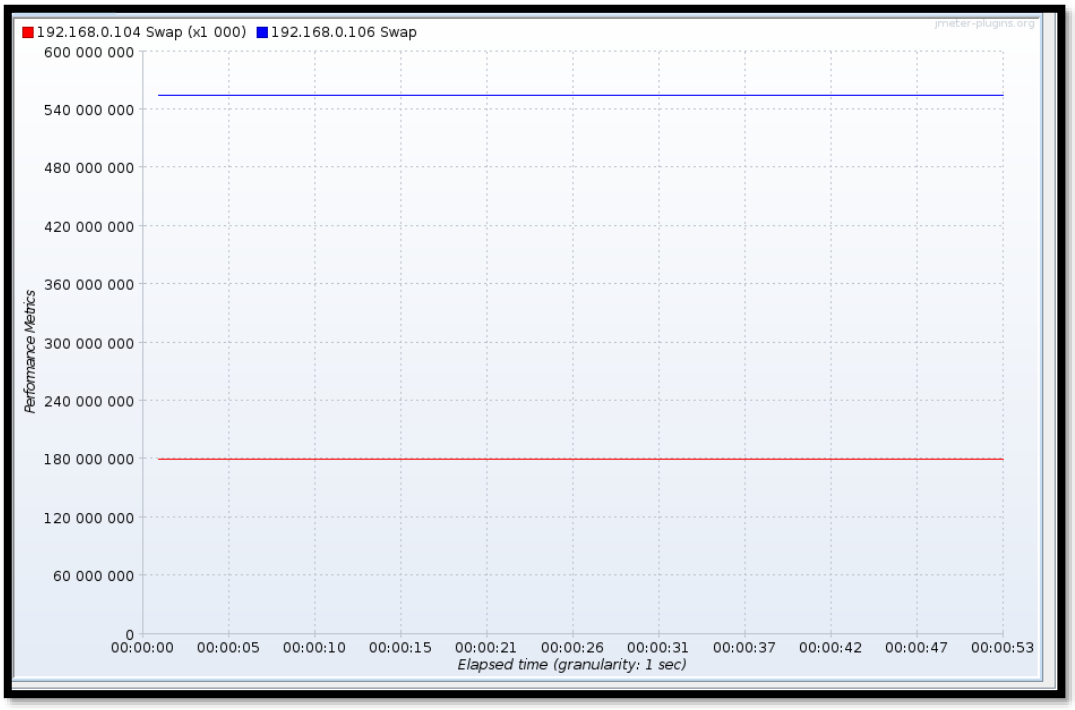


Figura 68. Uso de la Swap, durante consulta con Nodo1 y Nodo2.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la imagen anterior, se observa una comparación gráfica del uso de la swap por parte de Nodo1 (azul) y Nodo2 (rojo). En este caso se puede observar que, Nodo1 utilizó en mayor medida la swap.

4.5.6. Prueba PR6

Esta prueba requiere únicamente del Nodo1 master y Nodo3 de datos, por lo que, la imagen que se muestra a continuación, es una prueba del monitoreo a través del API de Elasticsearch de los 5 fragmentos del indice “censos” asignados únicamente al Nodo3 de datos. Para este caso el Nodo2 está fuera de servicio.

```
censos 3 p STARTED    26732520 762.8mb 192.168.0.105 nodo3
censos 3 r UNASSIGNED
censos 2 p STARTED    26720830 701.6mb 192.168.0.105 nodo3
censos 2 r UNASSIGNED
censos 4 p STARTED    26678994 719.9mb 192.168.0.105 nodo3
censos 4 r UNASSIGNED
censos 1 p STARTED    26722913 688.5mb 192.168.0.105 nodo3
censos 1 r UNASSIGNED
censos 0 p STARTED    26689213 695.7mb 192.168.0.105 nodo3
censos 0 r UNASSIGNED
```

Figura 69. Monitoreo de los fragmentos del indice censos, sólo al Nodo2

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

Petición HTTP a la ruta */censos/encuestas2/_search* de la API REST de Elasticsearch, que realiza la consulta de todas las encuestas con un filtro de la pregunta "Área urbana o rural".

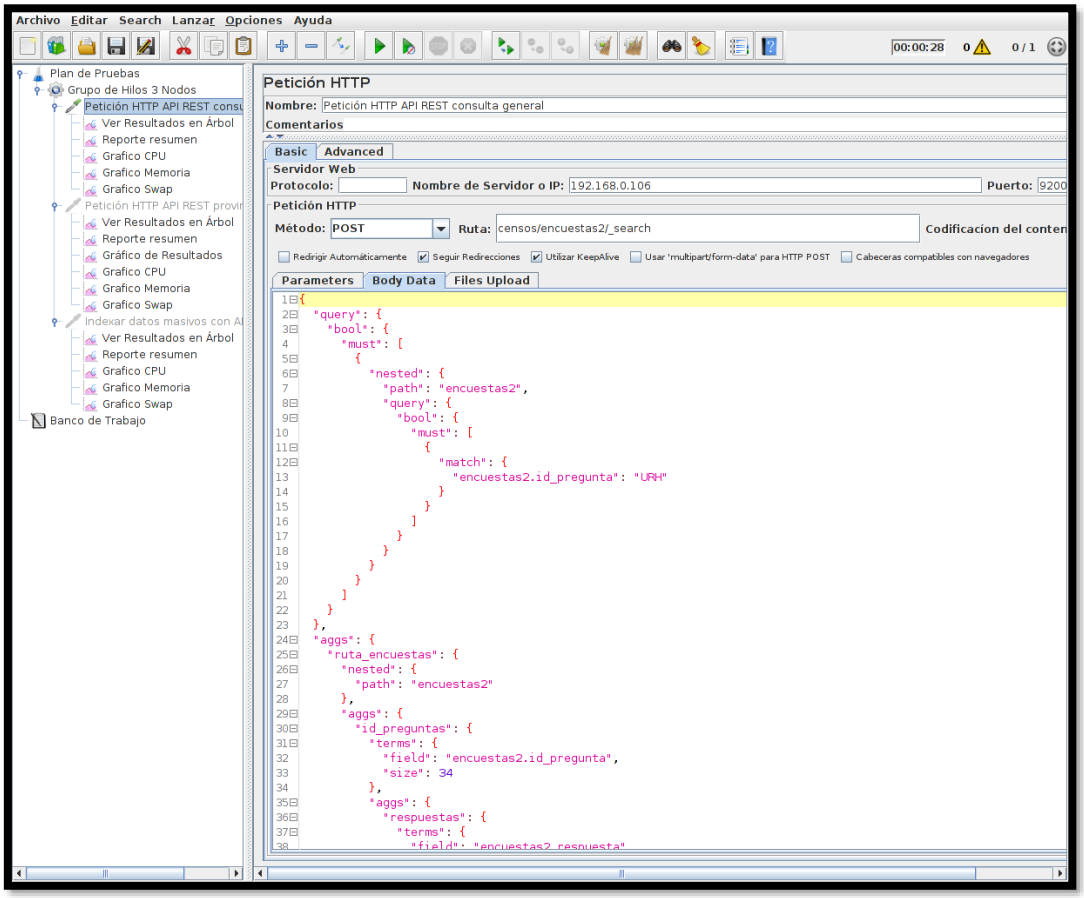


Figura 70. Petición HTTP a la ruta */censos/encuestas2/_search* de la API REST de Elasticsearch

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra la petición HTTP a la ruta *censos/encuestas2/_search* del API REST de Elasticsearch. Dicha ruta hace una solicitud al servidor por el método POST, la cual permite realizar una consulta que solicita todas las encuestas de la pregunta “URH” o “Área urbana o rural”.

Resultados en Árbol

Muestra #	Tiempo de co...	Nombre del hilo	Etiqueta	Tiempo de Mues...	Estado	Bytes	Sent Bytes	Latency	Co
1	15:36:22.182	Grupo de Hilos 3 No...	Petición HTTP API...	28215		19742	1156	28215	

Figura 71. Resultados en Árbol

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra los resultados de las pruebas en un receptor denominado “Resultado en Árbol”, el cual contiene datos como: tiempo de muestreo, estado, bytes, bytes enviados, latencia, entre otros.

Rendimiento de la CPU de Nodo1 y Nodo3, al realizar la consulta.

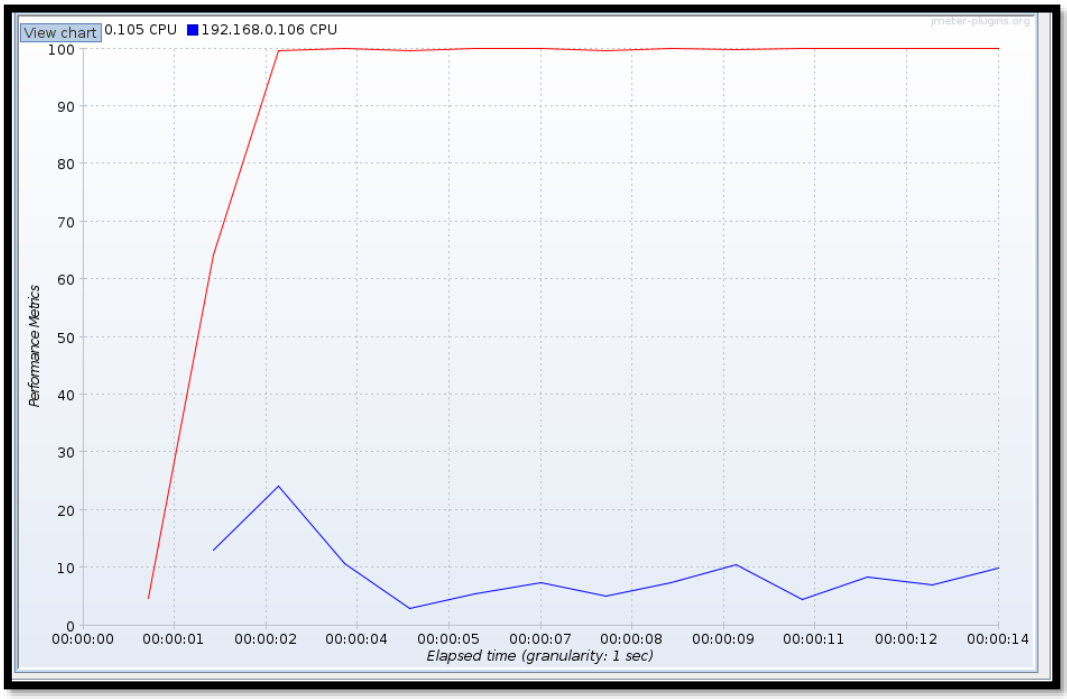


Figura 72. Rendimiento CPU, durante la consulta con Nodo1 y Nodo3.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la figura anterior, se muestra una comparación gráfica del uso de la CPU del Nodo1 (azul) y Nodo3 (rojo), que son los que están trabajando en la prueba. Como se puede observar el procesamiento es mayor (100%) para el Nodo3 debido ya que, es el único que se encarga de realizar la búsqueda.

Uso de la memoria RAM durante la consulta.

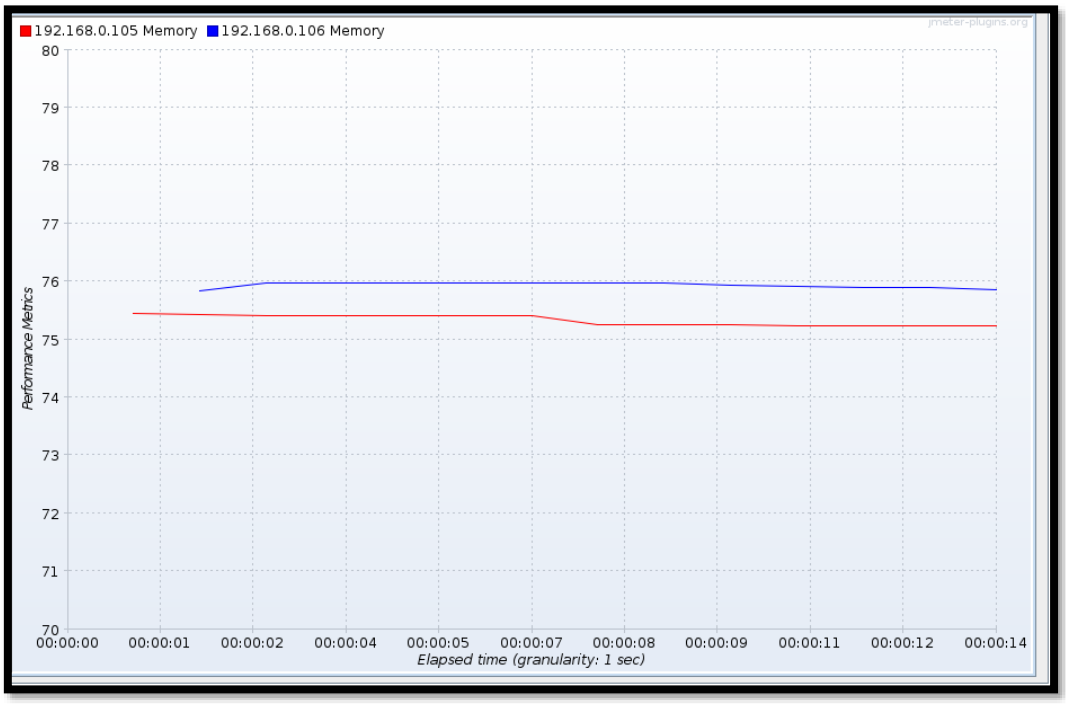


Figura 73. Uso de la memoria RAM, durante la consulta con Nodo1 y Nodo3.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la imagen anterior, se observa una comparación gráfica del uso de la memoria RAM por parte de Nodo1 (azul) y Nodo3 (rojo). En este caso se puede observar que, Nodo1 utilizó en mayor medida la memoria.

Uso de la Swap durante la consulta.

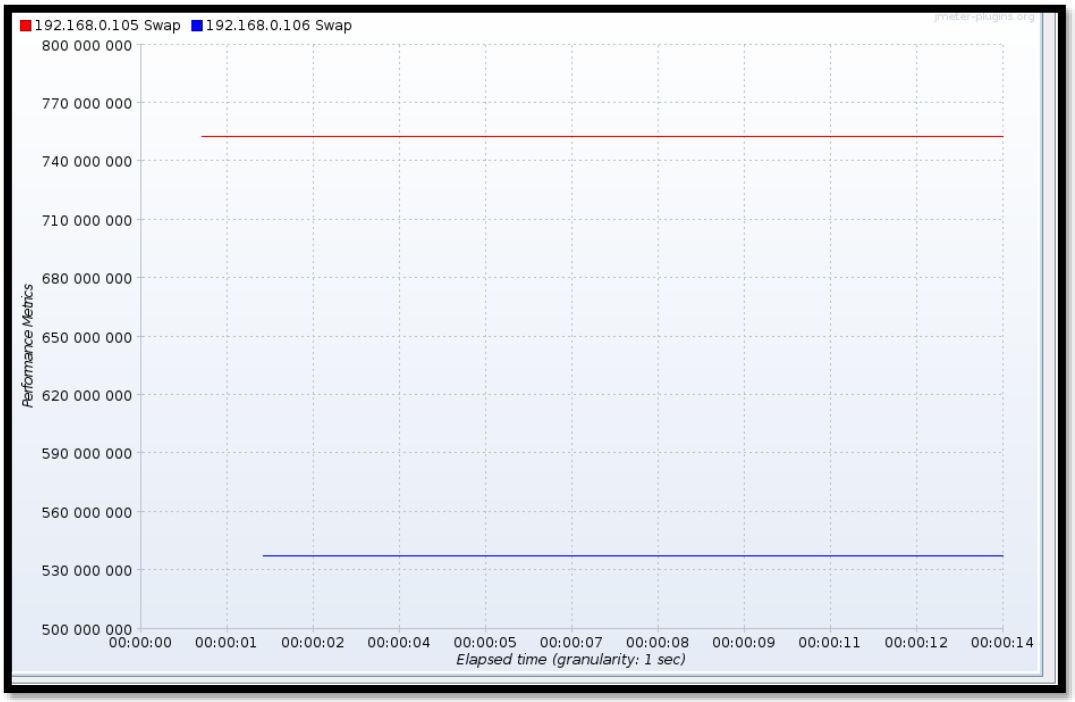


Figura 74. Uso de la Swap, durante consulta con Nodo1 y Nodo3.

Fuente: Borys Zuñiga

Elaboración: Borys Zuñiga

En la imagen anterior, se observa una comparación gráfica del uso de la swap por parte de Nodo1 (azul) y Nodo2 (rojo). En este caso se puede observar que, Nodo2 utilizó en mayor medida la swap.

4.6. Análisis del resultado de las pruebas

4.6.1. Resumen de los resultados de pruebas de indexación.

- Los resultados de las pruebas de PR1, arrojan lo siguiente: 94.188 milisegundos de duración a la solicitud de indexación, con un total de 1.688 bytes de datos a almacenar, con 30.000 datos de encuestas utilizando los 3 nodos. Además, se puede observar el rendimiento tanto de la CPU, memoria y swap (indicador del uso del entorno Java).

- Los resultados de las pruebas de PR2, arrojan lo siguiente: 121.175 milisegundos de duración a la solicitud de indexación, con un total de 1.688 bytes de datos a almacenar, con 50.000 datos de encuestas utilizando los 3 nodos. Además, se puede observar el rendimiento tanto de la CPU, memoria y swap.

- Los resultados de las pruebas de PR3, arrojan lo siguiente: 300.798 milisegundos de duración a la solicitud de indexación, con un total de 2.172 bytes de datos a almacenar, con 70.000 datos de encuestas utilizando los 3 nodos. Además, se puede observar el rendimiento tanto de la CPU, memoria y swap.

4.6.2. Interpretación general de los resultados de pruebas de indexación.

- Las solicitudes de indexación al servidor con 30.000 datos, los tiempos de respuesta oscilan aproximadamente en 1.35 minutos. En cuanto a la utilización de la CPU, se puede observar un uso constante entre el 97 y 100% del Nodo 2 de datos mientras duró la solicitud, respecto al Nodo3, usó el 100% del procesamiento únicamente al inicio de la solicitud, lo que indica un mejor rendimiento del Nodo3 para las solicitudes de indexación, particularmente el ¿por qué? de este comportamiento será analizado más adelante. En cuanto a la utilización de la memoria se puede observar un mayor uso de la misma (90%) por parte del nodo master respecto a ambos nodos de datos. Por otra parte, el uso de la swap se puede decir que se observa un mayor uso del Nodo2, lo que indica que el entorno Java está trabajando la mayor parte del tiempo por parte de este nodo, durante la solicitud de indexación.

- Las solicitudes de indexación al servidor con 50.000 datos, los tiempos de respuesta oscilan aproximadamente en 2.02 minutos, lo cual indica un aumento del 24.4% al agregar 20.000 datos más para indexar. En cuanto al utilización del CPU, se puede observar un uso prolongado del Nodo 2 de datos mientras duró la solicitud, respecto al Nodo3 que usa más procesamiento al inicio de la solicitud, lo que indica un mejor rendimiento del Nodo3 para las solicitudes de indexación. En cuando a la utilización de la memoria y swap se observa un comportamiento parecido al análisis anterior.

- Para las solicitudes de indexación al servidor con 70.000 datos, prácticamente tardó 225% más que con 30.000 y 150% más que con 50.000 datos, dando como resultado un “Error”, lo que indica que la capacidad máxima es de 50.000 datos (filas .csv) de procesamiento a solicitudes de indexación de datos con la estructura de una encuesta.

- Las pruebas realizadas comprenden la indexación de datos que particularmente en este proyecto tienen la estructura más compleja, como son las encuestas. El tiempo de

duración de las solicitudes de indexación variará en función de la estructura de los datos y el mapeo de los documentos.

¿Por qué el Nodo3 tiene mejor rendimiento que el Nodo2 para indexar datos?

Particularmente este comportamiento es esperado, debido a las características de los hosts en cuanto a hardware se refiere como se especificó en el punto 2.2.2. Al hacer una operación ya sea de indexación o eliminación, Elasticsearch crea subprocesos (1 + número de procesadores disponibles) para atender la demanda de dichas operaciones dependiendo del número de procesadores físicos y lógicos de la CPU, tal y como lo indica la documentación. En otras palabras, Nodo2 de datos tiene 3 subprocesos para atender las operaciones de indexación, y, por su parte, Nodo3, dispone de 5 hilos de procesamiento. Por tal motivo se justifica al observar en las pruebas realizadas que Nodo3 tenga un mejor rendimiento en la indexación de los datos.

4.6.3. Resumen de los resultados de las pruebas de una consulta.

- Los resultados de las pruebas de PR4, determinaron los siguiente: 14.458 milisegundos en atender la solicitud de consulta, 1.156 bytes enviados, utilizando los 3 nodos.
- Los resultados de las pruebas de PR5, determinaron los siguiente: 52.933 milisegundos en atender la solicitud de consulta, 1.156 bytes enviados, utilizando únicamente Nodo1 y Nodo2.
- Los resultados de las pruebas de PR6, determinaron los siguiente: 28.215 milisegundos en atender la solicitud de consulta, 1.156 bytes enviados, utilizando únicamente Nodo1 y Nodo3.

4.6.4. Interpretación general de los resultados de pruebas de una consulta.

Una vez analizados los resultados de las pruebas se puede concluir que:

- Casi 15 segundos tardó el servidor en devolver la respuesta exitosa, con los 3 nodos en funcionamiento. Así mismo, en la gráfica se puede observar que la CPU del Nodo2 está siendo usada entre el 98 y 100%, casi la totalidad del tiempo de duración de la consulta; mientras que, Nodo3 ocupa un máximo de 68% de la CPU para procesarla y en menos tiempo. El gráfico de la memoria indica que Nodo1 y Nodo2 la usó en un máximo de 74% y 75%, respectivamente, mientras que Nodo3 la usó un máximo del 68%, aproximadamente. La swap de Nodo1 fue usada la mayor cantidad y tiempo posible, respecto a ambos nodos de datos.

- Aproximadamente 53 segundos tardó el servidor en responder a la solicitud exitosamente, utilizando únicamente el Nodo1 y Nodo2 de datos. En la gráfica se puede observar una utilización que oscila entre el 98% y 100% de la CPU de Nodo2, durante casi todo el tiempo que duró la consulta. El Nodo1 usó más porcentaje en memoria y Swap.

- Aproximadamente 28 segundos tardó el servidor en responder a la solicitud exitosamente, utilizando únicamente el Nodo1 y Nodo3 de datos. En la gráfica se puede observar que Nodo3 utilizó el 100% de la CPU, durante casi todo el tiempo que duró la consulta. El Nodo1 usó más porcentaje en memoria y Nodo3 utilizó mayor porcentaje y tiempo la Swap.

- De esta manera se comprobó que, utilizando ambos nodos de datos para el procesamiento de las consultas se obtiene un mejor rendimiento del clúster.

- Nodo3 tiene mejor rendimiento al trabajar sólo junto al nodo master, en caso de que Nodo2 abandone el clúster por cualquier circunstancia.

- Los tiempos de respuesta a las solicitudes de consulta se duplican al trabajar únicamente un nodo de datos.

¿Por qué el Nodo3 tiene mejor rendimiento que el Nodo2 para consultar datos, también?

Se comprobó que los tiempos de respuesta a las operaciones de lectura también varían de acuerdo a la cantidad de nodos disponibles. Como en el caso anterior, al momento de la indexación de los datos se pudo observar que Nodo3 tuvo un mejor rendimiento, también para las consultas resulta de la misma manera respecto al Nodo2 de datos. Se esperaba este comportamiento, debido a lo antes explicado en función de los hilos de indexación, pero con la diferencia de que Elasticsearch por defecto para este caso crea ***int ((número de procesadores disponibles * 3) / 2) + 1*** hilos de búsqueda. En otras palabras, Nodo2 dispone de cuatro y Nodo3 de siete, hilos de búsqueda. Se puede decir entonces que, a más procesadores, más hilos (búsqueda, indexación, eliminación, etc.) Por tal motivo, queda plenamente justificado el mejor rendimiento del clúster con Nodo3 que con Nodo2.

CONCLUSIONES

Una vez desarrollado el presente proyecto se puede concluir que:

- ✓ La documentación de Elasticsearch es tan extensa, por lo que el presente trabajo, aporta una guía donde se resume y detalla paso a paso la implementación, instalación y configuración de un clúster, beneficiando posteriores trabajos que requieran una instalación no sólo en entornos de pruebas sino también en producción.
- ✓ Una correcta optimización del servidor para almacenar un determinado data set, resulta un punto de inflexión en los tiempos de respuesta. Si los datos a almacenar en Elasticsearch son estructurados, el modelado de los mismos es de vital importancia, porque es el paso previo para definir posteriormente el mapeo de los índices y tipos. Un mal modelado de los datos, puede provocar dificultades a la hora de realizar operaciones de consulta, edición o eliminación.
- ✓ Es muy complejo determinar con exactitud los tiempos de carga de datos en un clúster Elasticsearch, porque estos tiempos están dados básicamente por la estructura de los mismos.
- ✓ Las aplicaciones web construidas con JavaScript como lenguaje de programación base del lado del servidor, tienen muchas ventajas, una de ellas es que, se puede usar el paradigma de programación orientada a eventos a través del entorno Node.js, lo cual es propicio para trabajar con volúmenes de datos. Además, el framework Express.js es muy útil a la hora de realizar aplicaciones web sencillas y de forma rápida, porque provee una estructura de aplicación muy bien definida y organizada.
- ✓ Elasticsearch es una herramienta *open_source* de mucho valor para quienes se dedican a trabajar con altas cantidades de datos, porque provee un análisis de texto completo con soporte para varios idiomas.
- ✓ Elasticsearch permite escalar horizontalmente de forma sencilla y rápida, además de garantizar la disponibilidad de los datos a la hora de integrar varios nodos dedicados al procesamiento de los mismos.
- ✓ Elasticsearch dispone de la API REST y la API denominada *elasticsearch.js* disponible para Node.js, con la cual se puede realizar las diferentes operaciones HTTP. El método Bulk de elasticsearch.js, es una herramienta muy poderosa a la hora de realizar una carga masiva de datos, porque permite enviar en una sola petición varias operaciones ya sean éstas de: indexación, eliminación, edición, etc.

- ✓ Elasticsearch permite heterogeneidad en los componentes de hardware, pero el rendimiento general del clúster depende básicamente de los procesadores físicos y lógicos además de, un correcto balance con la memoria RAM y disco duro, por lo que es aconsejable utilizar componentes de similares características.
- ✓ Apache JMeter es una herramienta muy valiosa que no sólo permite diseñar un plan de pruebas, sino también ejecutar el mismo, permitiendo evaluar los resultados de manera estadística y gráfica.
- ✓ Ansible es una poderosa herramienta de gestión y configuración, con la cual, si se usaría en sus capacidades completas, fácilmente se podría crear un *playbook* para instalar el software Elasticsearch y todas sus dependencias.
- ✓ A partir del presente trabajo, resultaría favorable continuar con otro proyecto donde se realice estimación de tiempos de carga y consultas para un cierto número de datos, así mismo, donde se pueda determinar los recursos (procesamiento real, procesadores físicos y lógicos, disco duro, memoria) que se necesitarían para las diferentes actividades a realizar con el servidor. La finalidad de este proyecto a futuro es la estimación real de costos de adquisición de los recursos necesarios y procesamiento real de un clúster Elasticsearch.
- ✓ Se puede utilizar el clúster Elasticsearch instalado para usar las capacidades de análisis y búsqueda de texto completo para realizar un buscador de información representativa y con significado real, como, por ejemplo: El contenido de las tesis digitales de la Universidad Técnica Particular de Loja. La finalidad de este proyecto a futuro no sólo es la búsqueda de palabras como tal, sino la búsqueda de información interpretando el pensamiento humano.

RECOMENDACIONES

Luego del desarrollo del proyecto se recomienda:

- ✓ La utilización de Elasticsearch en proyectos que requieren un alto análisis de texto completo sobre altas cantidades de datos, ya que provee muchas herramientas para realizar aquello.
- ✓ La utilización de Node.js para construir aplicaciones web en las cuales haya un alto promedio de interacciones asíncronas.
- ✓ La utilización de Ansible para administrar remotamente no sólo servidores Elasticsearch, sino cualquier tipo de servidor, de preferencia distribuidos.
- ✓ Continuar con proyectos basados en la utilización de Elasticsearch como servidor, en las que se aborde temas relacionados a la investigación de: carga masiva de datos para cualquier estructura de datos, con las que se pueda determinar o estimar los tiempos de carga de un conjunto de datos, las capacidades de procesamiento que se necesita, etc.
- ✓ Utilizar Elasticsearch para entornos de producción, con componentes homogéneos o al menos similares, para que el rendimiento del mismo no se vea afectado por la heterogeneidad.
- ✓ Los datos se deben mapear en función de “que preguntas se quieren responder”, y no en función de “que podemos hacer con los datos”.
- ✓ Tener mucha cautela en la utilización de Elasticsearch en proyectos que involucren datos de alta confidencialidad.

BIBLIOGRAFÍA

Akdoğan, H. (2015). *Elasticsearch Indexing* (First). Birmingham: Packt Publishing Ltd.

Apache JMeter. (2016). Apache JMeter. Retrieved February 7, 2018, from http://jmeter.apache.org/

Arkhipkin, D., Lauret, J., & Shanmuganathan, P. V. (2015). Modular and scalable RESTful API to sustain STAR collaboration's record keeping. *Journal of Physics: Conference Series*, *664*(5), 52021. https://doi.org/10.1088/1742-6596/664/5/052021

Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M., & Vallero, S. (2015). Towards Monitoring-as-a-service for Scientific Computing Cloud applications using the ElasticSearch ecosystem. *Journal of Physics: Conference Series*, *664*(2), 22040. https://doi.org/10.1088/1742-6596/664/2/022040

Barranco Fragoso, R. (2012). ¿Qué es Big Data? Retrieved June 28, 2016, from https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/

Blancarte, O. (2017). Introducción a NodeJS (JavaScript del lado del Servidor) - Oscar Blancarte Blog. Retrieved February 7, 2018, from https://www.oscarblancarteblog.com/2017/05/29/introduccion-a-nodejs-2/

Chen, X., Ji, Z., Fan, Y., & Zhan, Y. (2017). Restful API Architecture Based on Laravel Framework. *Journal of Physics: Conference Series*, *910*(1), 12016. https://doi.org/10.1088/1742-6596/910/1/012016

Cisco. (2016a). Cisco Visual Networking Index : Forecast and Methodology , 2015 – 2020. Retrieved June 29, 2016, from http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

Cisco. (2016b). VNI Complete Forecast Highlights Tool. Retrieved June 29, 2016, from http://www.cisco.com/c/m/en_us/solutions/service-provider/vni-forecast-highlights.html

Coulouris, G., Dollimore, J., & Kindberg, T. (2007). *Sistemas Distribuidos: Conceptos y Diseño*.

Crockford, D. (2008). JavaScript: The Good Parts. Online. *Journal of Information Processing and Management*, *44*(8), 584. https://doi.org/10.1241/johokanri.44.584

Daniel Berman. (2016). Elasticsearch Mapping: The Basics, Two Types, and a Few Examples.

Retrieved March 27, 2017, from https://logz.io/blog/elasticsearch-mapping/

Elasticsearch. (2016). Elasticsearch Search Search & Analyze in Real Time. Retrieved April 29, 2016, from https://www.elastic.co/downloads/elasticsearch

Express. (2014). Express - Infraestructura de aplicaciones web Node.js. Retrieved February 7, 2018, from http://expressjs.com/es/

Fundación Node.js. (2009). Node.js. Retrieved February 7, 2018, from https://nodejs.org/es/

Gheorghe, R., Hinman, M. L., & Russo, R. (2016). *Elasticsearch in Action* (First). Shelter Island: Manning Publications Co.

Gonzalez, S. (2017). Generalidades del protocolo HTTP - HTTP | MDN. Retrieved February 14, 2018, from https://developer.mozilla.org/es/docs/Web/HTTP/Overview

Google Developers. (2007). Uso de Google Charts | Gráficos | Desarrolladores de Google. Retrieved February 7, 2018, from https://developers.google.com/chart/interactive/docs/

Hall, D. (2013). *Ansible Configuration Management*.

He, Y., Yu, F. R., Zhao, N., Yin, H., Yao, H., & Qiu, R. C. (2016). Big Data Analytics in Mobile Cellular Networks. *IEEE Access*, *4*, 1985–1996. https://doi.org/10.1109/ACCESS.2016.2540520

Krishna R., A. (2016). Modelado de datos del documento para una base de datos NoSQL | Microsoft Docs. Retrieved December 29, 2016, from https://docs.microsoft.com/es-es/azure/cosmos-db/modeling-data

Kuc, R., & Rogozinski, M. (2014). *Elasticsearch Server: Elasticsearch server : A practical guide to building fast, scalable, and flexible search solutions with clear and easy-to-understand examples*.

M.L.Liu. (2004). *Computación Distribuida: Fundamentos y Aplicaciones*.

Mathe, Z., Haen, C., & Stagni, F. (2017). Monitoring performance of a highly distributed and complex computing infrastructure in LHCb. *Journal of Physics: Conference Series*, *898*(9), 92028. https://doi.org/10.1088/1742-6596/898/9/092028

Paro, A. (2015). *ElasticSearch Cookbook* (Second). Birmingham: Packt Publishing Ltd.

Red Hat, I. (2016). About Ansible. Retrieved June 27, 2016, from

http://docs.ansible.com/ansible/index.html

Soares, S. (2012). Not Your Type? Big Data Matchmaker On Five Data Types You Need To Explore Today. Retrieved June 29, 2016, from http://www.dataversity.net/not-your-type-big-data-matchmaker-on-five-data-types-you-need-to-explore-today/?cm_mc_uid=10501353963114671705266&cm_mc_sid_50200000=1467210690

Sphinx, J. (2016). Acerca de JavaScript - JavaScript | MDN. Retrieved February 7, 2018, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

Tutorials Point. (2015). Node.js Quick Guide. Retrieved February 7, 2018, from https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm

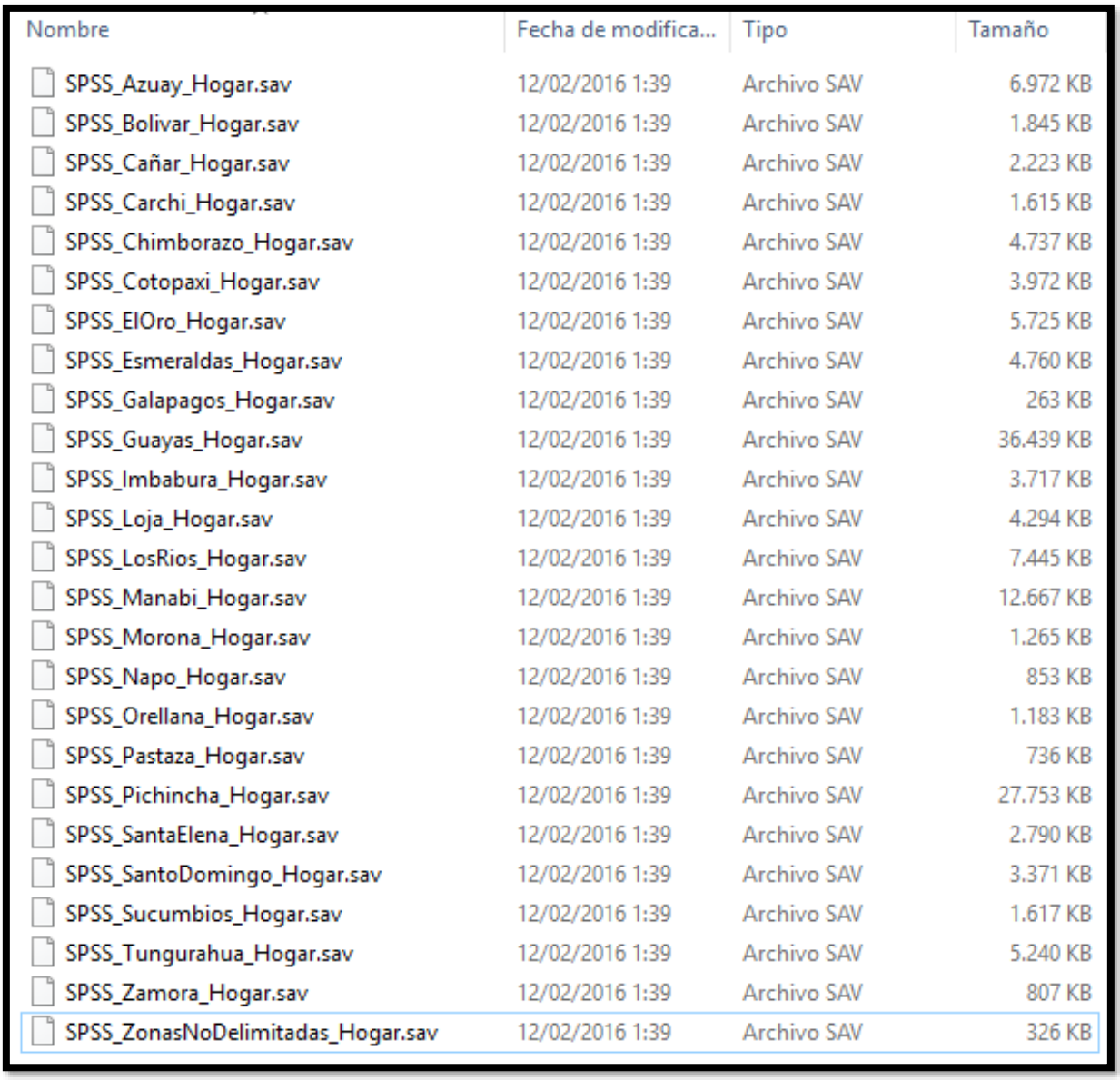
ANEXOS

ANEXO A.

Tabla 17. Detalle de los archivos descargados

Id.	Provincia	Archivo	Número de Datos
1	Azuay	SPSS_Azuay_Hogar.sav	188.765
2	Bolívar	SPSS_Bolivar_Hogar.sav	47.836
3	Cañar	SPSS_Cañar_Hogar.sav	58.712
4	Carchi	SPSS_Carchi_Hogar.sav	44.239
5	Cotopaxi	SPSS_Cotopaxi_Hogar.sav	103.259
6	Chimborazo	SPSS_Chimborazo_Hogar.sav	125.552
7	El Oro	SPSS_ElOro_Hogar.sav	163.527
8	Esmeraldas	SPSS_Esmeraldas_Hogar.sav	129.630
9	Guayas	SPSS_Guayas_Hogar.sav	959.719
10	Imbabura	SPSS_Imbabura_Hogar.sav	103.159
11	Loja	SPSS_Loja_Hogar.sav	117.163
12	Los Ríos	SPSS_LosRios_Hogar.sav	202.125
13	Manabí	SPSS_Manabi_Hogar.sav	343.581
14	Morona Santiago	SPSS_Morona_Hogar.sav	33.475
15	Napo	SPSS_Napo_Hogar.sav	22.585
16	Pastaza	SPSS_Pastaza_Hogar.sav	19.897
17	Pichincha	SPSS_Pichincha_Hogar.sav	728.336
18	Tungurahua	SPSS_Tungurahua_Hogar.sav	140.754
19	Zamora Chinchipe	SPSS_Zamora_Hogar.sav	21.436
20	Galápagos	SPSS_Galapagos_Hogar.sav	7.376
21	Sucumbíos	SPSS_Sucumbios_Hogar.sav	43.262
22	Orellana	SPSS_Orellana_Hogar.sav	31.635
23	Santo Domingo de los Tsáchilas	SPSS_SantoDomingo_Hogar.sav	95.273
24	Santa Elena	SPSS_SantaElena_Hogar.sav	76.335
90	Zonas no delimitadas	SPSS_ZonasNoDelimitadas_Hogar.sav	7.897
		Total	3.815.528

Elaboración: Borys Zuñiga



Nombre	Fecha de modifica...	Tipo	Tamaño
SPSS_Azuay_Hogar.sav	12/02/2016 1:39	Archivo SAV	6.972 KB
SPSS_Bolivar_Hogar.sav	12/02/2016 1:39	Archivo SAV	1.845 KB
SPSS_Cañar_Hogar.sav	12/02/2016 1:39	Archivo SAV	2.223 KB
SPSS_Carchi_Hogar.sav	12/02/2016 1:39	Archivo SAV	1.615 KB
SPSS_Chimborazo_Hogar.sav	12/02/2016 1:39	Archivo SAV	4.737 KB
SPSS_Cotopaxi_Hogar.sav	12/02/2016 1:39	Archivo SAV	3.972 KB
SPSS_ElOro_Hogar.sav	12/02/2016 1:39	Archivo SAV	5.725 KB
SPSS_Esmeraldas_Hogar.sav	12/02/2016 1:39	Archivo SAV	4.760 KB
SPSS_Galapagos_Hogar.sav	12/02/2016 1:39	Archivo SAV	263 KB
SPSS_Guayas_Hogar.sav	12/02/2016 1:39	Archivo SAV	36.439 KB
SPSS_Imbabura_Hogar.sav	12/02/2016 1:39	Archivo SAV	3.717 KB
SPSS_Loja_Hogar.sav	12/02/2016 1:39	Archivo SAV	4.294 KB
SPSS_LosRios_Hogar.sav	12/02/2016 1:39	Archivo SAV	7.445 KB
SPSS_Manabi_Hogar.sav	12/02/2016 1:39	Archivo SAV	12.667 KB
SPSS_Morona_Hogar.sav	12/02/2016 1:39	Archivo SAV	1.265 KB
SPSS_Napo_Hogar.sav	12/02/2016 1:39	Archivo SAV	853 KB
SPSS_Orellana_Hogar.sav	12/02/2016 1:39	Archivo SAV	1.183 KB
SPSS_Pastaza_Hogar.sav	12/02/2016 1:39	Archivo SAV	736 KB
SPSS_Pichincha_Hogar.sav	12/02/2016 1:39	Archivo SAV	27.753 KB
SPSS_SantaElena_Hogar.sav	12/02/2016 1:39	Archivo SAV	2.790 KB
SPSS_SantoDomingo_Hogar.sav	12/02/2016 1:39	Archivo SAV	3.371 KB
SPSS_Sucumbios_Hogar.sav	12/02/2016 1:39	Archivo SAV	1.617 KB
SPSS_Tungurahua_Hogar.sav	12/02/2016 1:39	Archivo SAV	5.240 KB
SPSS_Zamora_Hogar.sav	12/02/2016 1:39	Archivo SAV	807 KB
SPSS_ZonasNoDelimitadas_Hogar.sav	12/02/2016 1:39	Archivo SAV	326 KB

Figura 75. Captura de pantalla que muestra los archivos descargados con sus respectivas propiedades.

Elaboración: Borys Zuñiga

ANEXO B.

A continuación, se describe las variables de los archivos *(.sav).*

Tabla 18. Descripción de las variables de los archivos SPSS

Variable	Preguntas	Valores	Significado
I01	Provincia	1, 2, 3…24, 90.	Representa una provincia.
I02	Cantón	1, 2, 3…	Representa un cantón perteneciente una provincia.
I03	Parroquia	50,51,52…	Representa una parroquia perteneciente a un cantón.
URH	Área urbana o rural	1, 2	1 = Área Urbana 2 = Área Rural
H01	Del total de cuartos de este hogar, Cuantos son exclusivos para dormir	0, 1, 2, 3, 4, 5, 6	0 = 0 Dormitorios 1 = 1 Dormitorio 2 = 2 Dormitorios 3 = 3 Dormitorios 4 = 4 Dormitorios 5 = 5 Dormitorios 6 = 6 Dormitorios
H02	Tiene este hogar cuarto o espacio exclusivo para cocinar	1, 2	1 = Si 2 = No
H03	El servicio higiénico o escusado que dispone el hogar es	1, 2, 3	1 = De uso exclusivo 2 = Compartido con varios hogares 3 = No tiene
H04	Dispone este hogar de espacio con instalaciones y/o ducha para bañarse	1, 2, 3	1 = De uso exclusivo del hogar 2 = Compartido con varios hogares 3 = No tiene

H05	Cuál es el principal combustible o energía que utiliza este hogar para cocinar	1, 2, 3, 4, 5, 6, 7	1 = Gas (tanque o cilindro) 2 = Gas centralizado 3 = Electricidad 4 = Leña, carbón 5 = Residuos vegetales y/o de animales 6 = Otro (Ej. Gasolina, kérex o diésel, etc) 7 = No cocina
H06	Principalmente, el agua que toman los miembros del hogar	1, 2, 3, 4, 5	1 = La beben tal como llega al hogar 2 = La hierven 3 = Le ponen cloro 4 = La filtran 5 = Compran agua purificada
H07	Dispone este hogar de servicio de teléfono convencional	1, 2	1 = Si 2 = No
H09	Dispone este hogar de servicio de internet	1, 2	1 = Si 2 = No
H10	Dispone este hogar de computadora	1, 2	1 = Si 2 = No
H11	Dispone este hogar de servicio de televisión por cable	1, 2	1 = Si 2 = No
H13A	Algún miembro de este hogar se traslada fuera de esta ciudad o parroquia rural para trabajar	1, 2	1 = Si 2 = No
H14A	Algún miembro de este hogar se traslada fuera de esta ciudad o parroquia rural para estudiar	1, 2	1 = Si 2 = No
H15	La vivienda que ocupa este hogar es	1, 2, 3, 4, 5, 6, 7	1 = Propia y totalmente pagada

			2 = Propia y la está pagando 3 = Propia (regalada, donada, heredada o por posesión) 4 = Prestada o cedida (no pagada) 5 = Por servicios 6 = Arrendada 7 = Anticresis
M1	Durante el año 2010, Alguna persona de este hogar recibió dinero por parte de familiares o amigos que viven en el exterior	1, 2	1 = Si 2 = No
M2A	A partir del último censo de población y vivienda (Noviembre 2001) una o más personas que vivían en este hogar viajaron a	1, 2	1 = Si 2 = No

Elaboración: Borys Zuñiga

Tabla 19. Descripción de los datos de las preguntas de censo.

Identificador	Código Pregunta(Variable)	Etiqueta
1	I01	Provincia
2	I02	Cantón
3	I03	Parroquia
4	URH	Área urbana o rural
5	H01	Del total de cuartos de este hogar, Cuantos son exclusivos para dormir
6	H02	Tiene este hogar cuarto o espacio exclusivo para cocinar

7	H03	El servicio higiénico o escusado que dispone el hogar es
8	H04	Dispone este hogar de espacio con instalaciones y/o ducha para bañarse
9	H05	Cuál es el principal combustible o energía que utiliza este hogar para cocinar
10	H06	Principalmente, el agua que toman los miembros del hogar
11	H07	Dispone este hogar de servicio de teléfono convencional
12	H09	Dispone este hogar de servicio de internet
13	H10	Dispone este hogar de computadora
14	H11	Dispone este hogar de servicio de televisión por cable
15	H13A	Algún miembro de este hogar se traslada fuera de esta ciudad o parroquia rural para trabajar
16	H14A	Algún miembro de este hogar se traslada fuera de esta ciudad o parroquia rural para estudiar
17	H15	La vivienda que ocupa este hogar es
18	M1	Durante el año 2010, Alguna persona de este hogar recibió dinero por parte de familiares o amigos que viven en el exterior
19	M2A	A partir del último censo de población y vivienda (Noviembre 2001) una o más personas que vivían en este hogar viajaron a

Elaboración: Borys Zuñiga

Tabla 20. Descripción de los datos de las provincias, cantones y parroquias.

Archivo	Variable	Descripción
provincias.csv	id_provincia	Almacena un identificador para cada provincia.
	nombre_provincia	Almacena un nombre de una provincia
cantones.csv	codigo_pais	Almacena el código de país, ejemplo: 1 = Ecuador.
	codigo_provincia	Almacena el código de la provincia a la que pertenece el cantón, ejemplo: 1 = Azuay.
	codigo_canton	Almacena el código del cantón al cual se le ha sido asignado, ejemplo: 1 = Cuenca.
	nombre_canton	Almacena el nombre del cantón, ejemplo: Cuenca.
parroquias.csv	codigo_provincia	Almacena el código de la provincia a la que pertenece el cantón, ejemplo: 1 = Azuay.
	codigo_canton	Almacena el código del cantón al cual se le ha sido asignado, ejemplo: 1 = Cuenca.
	codigo_parroquia	Almacena el código de la parroquia a la cual se le ha sido asignada, ejemplo: 51 = Baños.
	nombre_parroquia	Almacena el nombre de la parroquia, ejemplo: Baños.

Elaboración: Borys Zuñiga

ANEXO C.

Modelado de los datos expresados en formato JSON:

Tabla 21. Documento del tipo “país”


```
{
    "id_pais": "1",
    "nombre_pais": "Ecuador"
}
```


Elaboración: Borys Zuñiga

Tabla 22. Documentos del tipo “provincias”


```
{
  "id_provincia": "1",
  "nombre_provincia": "Azuay",
  "id_pais": 1
},
{
  "id_provincia": "2",
  "nombre_provincia": "Bolívar",
  "id_pais": 1
},
.
.
.
{
  "id_provincia": "90",
  "nombre_provincia": "Zonas No Delimitadas",
  "id_pais": 1
}
```


Elaboración: Borys Zuñiga

Tabla 23. Documentos del tipo “cantones”


```
{
  "codigo_pais": "1",
  "codigo_provincia": "1",
  "codigo_canton": "1",
  "nombre_canton": "Cuenca"
},
{
  "codigo_pais": "1",
  "codigo_provincia": "2",
  "codigo_canton": "1",
  "nombre_canton": "Guaranda"
},
.
.
.
{
  "codigo_pais": "1",
  "codigo_provincia": "90",
  "codigo_canton": "4",
  "nombre_canton": "El Piedrero"
}
```


Elaboración: Borys Zuñiga

Tabla 24. Documentos del tipo “parroquias”


```
{
  "codigo_provincia": "1",
  "codigo_canton": "1",
  "codigo_parroquia": "50",
  "nombre_parroquia": "Cuenca - Cabecera Cantonal y Capital Provincial"
},
{
  "codigo_provincia": "2",
  "codigo_canton": "1",
  "codigo_parroquia": "51",
  "nombre_parroquia": "Baños"
},
.
.
.
{
  "codigo_provincia": "24",
  "codigo_canton": "3",
  "codigo_parroquia": "52",
  "nombre_parroquia": "Jose Luis Tamayo (Muey)"
}
```


Elaboración: Borys Zuñiga

Tabla 25. Documentos del tipo “preguntas”


```
{
  "id_pregunta": "1",
  "codigo_pregunta": "I01",
  "etiqueta_pregunta": "Provincia"
},
{
  "id_pregunta": "2",
  "codigo_pregunta": "I02",
  "etiqueta_pregunta": "Canton"
},
.
.
.
{
  "id_pregunta": "19",
  "codigo_pregunta": "H09",
  "etiqueta_pregunta": "Dispone este hogar de servicio de internet"
}
```


Elaboración: Borys Zuñiga

Tabla 26. Documento del tipo “encuestas”


```
{
  "id_encuesta": 1,
  "encuestas": [
    {
      "id_pregunta": "I01",
      "respuesta": 1
    },
    {
      "id_pregunta": "I02",
      "respuesta": 1
    },
    .
    .
    .
    {
      "id_pregunta": "H09",
      "respuesta": 2
    }
  ]
}
```


Elaboración: Borys Zuñiga

ANEXO D.

La siguiente función Javascript crea el índice “censos” y mapea los tipos “país”, “provincias”, “cantones”, “parroquias” y “preguntas”, con sus respectivos campos, detallados en el punto 3.8.

Tabla 27. Función para crear el mapeo del índice “censo”.

```
function mapearIndex(index){
  elasticsearch.elasticCliente().indices.create({
    index: index,
    body: {
      mappings: {
        pais: {
          properties: {
            id_pais: {
              type: 'integer'
            },
            nombre_pais: {
              type: 'string',
              analyzer: 'standard',
              search_analyzer: 'standard'
            }
          }
        },
        provincias: {
          properties: {
            id_provincia: {
              type: 'integer'
            },
            nombre_provincia: {
              type: 'string',
              analyzer: 'standard',
              search_analyzer: 'standard'
            }
          },
          _parent: {
            type: 'pais'
          }
        },
        cantones: {
          properties: {
            id_canton: {
              type: 'integer'
            },
            codigo_pais: {
              type: 'integer'
            },
            codigo_provincia: {
              type: 'integer'
            },
            codigo_canton: {
              type: 'integer'
            },
            nombre_canton: {
              type: 'string',
```

```
            analyzer: 'standard',
            search_analyzer: 'standard'
          }
        },
        _parent: {
          type: 'provincias'
        }
      },
      parroquias: {
        properties: {
          id_parroquia: {
            type: 'integer'
          },
          codigo_provincia: {
            type: 'integer'
          },
          codigo_canton: {
            type: 'integer'
          },
          codigo_parroquia: {
            type: 'integer'
          },
          nombre_parroquia: {
            type: 'string',
            analyzer: 'standard',
            search_analyzer: 'standard'
          }
        }
      },
      preguntas: {
        properties: {
          id_pregunta: {
            type: 'integer'
          },
          codigo_pregunta: {
            type: 'string'
          },
          etiqueta_pregunta: {
            type: 'string',
            analyzer: 'standard',
            search_analyzer: 'standard'
          }
        }
      }
    }
  });
}
```

Elaboración: Borys Zuñiga

A continuación, se explica el código:

- El objeto ***elasticCliente(),*** crea la conexión hacia el servidor Elasticsearch.

```
 */
var elasticsearch = require('elasticsearch');
var elasticClient = new elasticsearch.Client({
    host: '192.168.0.106:9200',
    log: 'trace'
});

function elasticCliente(){
  return elasticClient;
}

exports.elasticCliente = elasticCliente

function hacerPing(){
  var servidor;
  elasticClient.ping({
    // undocumented params are appended to the query string
    hello: "elasticsearch"
  }, function (error) {
  if (error) {
    servidor = 0;
    console.error('El cluster no esta levantado !');
    return servidor;

  } else {
    servidor = 1;
    console.log('El servidor está bien !'+ servidor);
    return servidor;
  }
  });
}
```

Figura 76. Captura del código que realiza la conexión al servidor Elasticsearch.

Elaboración: Borys Zuñiga

- El método ***indices.create()***, se utiliza para crear un índice, en este caso “censos”, en el cual se envía como parámetro un objeto JSON, que deberá contener algunos parámetros para crear el índice, entre ellos: nombre del índice, cuerpo (*body*), etc.

  En la creación de un índice, por defecto Elasticsearch crea 5 fragmentos primarios y 1 fragmento de réplica por cada fragmento primario, sino se especifica estos parámetros. De esta manera, este proyecto dispone de un total de 10 fragmentos para el índice “censos”, los cuales son distribuidos de manera automática por Elasticsearch, que garantiza un balanceo correcto de los datos, así como también la disponibilidad de los mismo.

  El parámetro *number_of_shards*, donde se determina el número de fragmentos primarios, no es modificable una vez creado el índice, pero si *number_of_replicas.* El número de fragmentos varía en función de los datos a almacenar, para lo cual requiere un análisis

muy exhaustivo de los casos de uso, tal y como lo recomienda la documentación https://www.elastic.co/guide/en/elasticsearch/guide/current/replica-shards.html .

- En el campo ***index,*** se envía el nombre del índice a crear.
- En el campo ***body***, se envía un objeto JSON que debe contener todos los tipos con su respectivo mapeo. Tenga en cuenta que aquí se utilizan las cláusulas del API REST de Elasticsearch.
- La cláusula ***mapping,*** sirve para especificar que se va a hacer un mapeo de los diferentes tipos.
- ***país,*** es el nombre del tipo, al igual que "provincias", "cantones", "parroquias" y "preguntas".
- La cláusula ***properties,*** sirve para especificar los campos (*fields*) que contendrá el tipo.
- ***id_pais,*** es el nombre del campo.
- ***type,*** sirve para especificar el tipo de dato que almacenará el campo, por ejemplo: integer, string, boolean, date, object, nested, etc.
- ***analyzer,*** sirve para especificar el tipo de analizador que se utilizara para el campo al momento de almacenar, algunos analizadores de texto ya vienen incorporados, otros se deberán crear, dependiendo de las necesidades.
- ***search_analyzer,*** sirve para especificar el tipo de analizador que se utilizará para el campo al momento de realizar búsquedas.
- ***_parent,*** sirve para establecer una relación de padre-hijo entre documentos, por ejemplo, en el tipo "provincias" se especifica que es hijo del tipo "pais". Esto es fundamental para garantizar el enrutamiento del documento dentro del mismo fragmento.

  Por otro lado, en el tipo "cantones" también existe una relación de padre-hijo con el tipo "provincias", pero aquí las relaciones pueden extenderse a una generación más, los nietos, por ejemplo: existe una relación indirecta entre "cantones" y "país", si no se quiere perder dicha relación, o en su defecto, que los documentos terminen en fragmentos diferentes, es vital especificar el parámetro "*rounting*" a la hora de indexar los datos.

Función Javascript que permite mapear el tipo “encuestas2” con sus respectivos campos.

Tabla 28. Función Javascript para mapear el tipo “encuestas2”

```
function mapearTypeEncuesta(index){
elasticsearch.elasticCliente().indices.putMapping({
    index: index,
    type: 'encuestas2',
    body: {
      properties: {
        id_encuesta: {
          type: 'integer'
        },
        encuestas2: {
          type: 'nested',
          include_in_parent: true,
          properties: {
            id_pregunta: {
              type: 'string'
            },
            respuesta: {
              type: 'integer'
            }
          }
        }
      }
    }
  });
}
```

Elaboración: Borys Zuñiga

Estaba pendiente el mapeo del tipo “encuestas”, esto merece un punto a parte debido a la complejidad, que se pasa a explicar a continuación y solo se explicará las cláusulas que no se utilizaron en el punto anterior:

- El método ***indices.putMapping(),*** sirve para agregar un mapeo a un tipo en específico.
- **type,** que se encuentra antes del *body**,*** en este caso sirve para especificar el nombre del tipo.
- Pase directamente al segmento donde se encuentra ***type: 'nested'***. Dentro de Elasticsearch también admite que un campo sea de tipo *object*, si lo que se almacenará son documentos que incluyen una matriz plana, por ejemplo. [“Carlos”, “Juan”, “María”]. Por otro lado, también se admite que un campo sea de tipo “nested”, cuando lo que se va a almacenar es un arreglo de objetos, por ejemplo: [{nombre: “Carlos”, edad:21}, {nombre:

"Juan", edad:22}, {nombre: "María", edad:25}], la principal diferencia entre estos dos tipos, es que, *nested* permite consultar cada objeto de forma independiente.

- ***include_in_parent: true,*** sirve para especificar que los campos internos también serán almacenados como *nested*.

Una vez que se haya terminado de mapear el índice y todos sus tipos se hace una petición al API REST para verificar si todo el mapeo está correctamente creado.

$ http://localhost:9200/censos/_mapping?pretty

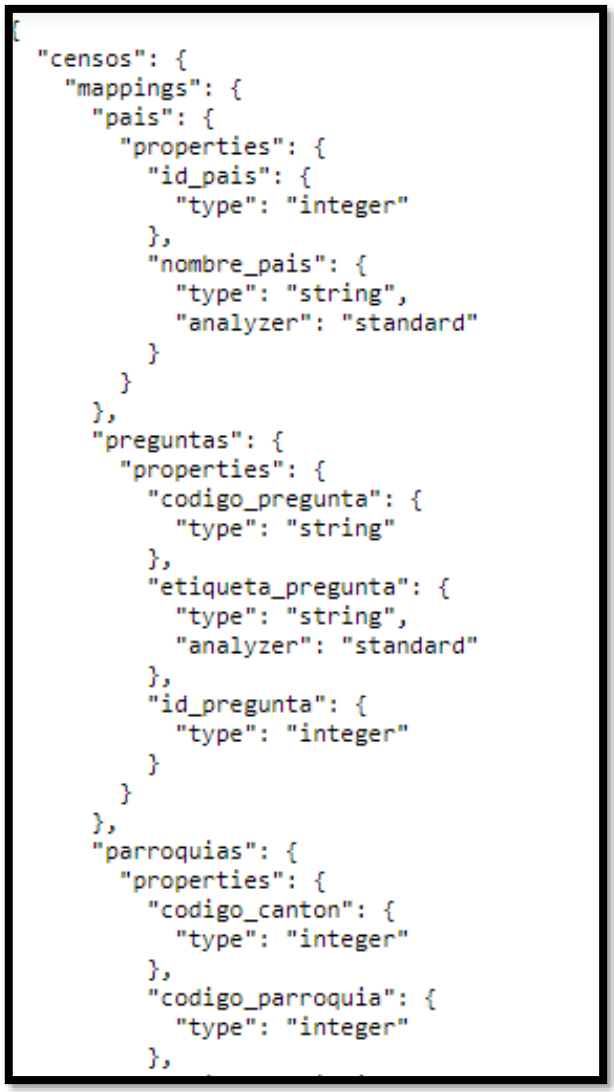


Figura 77. Captura que muestra parte del mapeo realizado al índice "censos"

Elaboración: Borys Zuñiga

ANEXO E.

Manual de Usuario de la Aplicación “Elastic Client”

Figura 78. Interfaz gráfica de Inicio de Aplicación

Elaboración: Borys Zuñiga

En la imagen anterior se muestra la interface gráfica de inicio de la aplicación. Compuesto en la parte superior por un menú que permite consultar los datos a nivel general, por provincias, por cantones y por parroquias, además, se encuentra la opción de carga masiva de datos.

En la sección principal “Preguntas Censo”, se listan todas las preguntas que conformaron el censo de Población y Vivienda del Ecuador en el año 2010, las cuales podrán ser consultadas de manera independiente.

- **Consultar los datos generales de una pregunta**

  Permite consultar el resumen de los datos generales de una pregunta en particular, para lo cual realice lo siguiente:

1. Seleccione una pregunta y haga clic en el ícono .
2. La aplicación mostrará: la pregunta seleccionada, una tabla con el resumen numérico de los datos y la representación de los mismos en un gráfico de pastel.

Nota: Si selecciona cualquiera de las 3 primeras preguntas y hace clic en , la aplicación re direccionará, para consultar ya sea por provincia, por cantón o por parroquia.

- **Consultar los datos de una pregunta por provincia**

  Muestra en resumen los datos de una pregunta seleccionada para una provincia en particular.

Figura 79. Interfaz gráfica del menú “Provincias”

Elaboración: Borys Zuñiga

1. Seleccione una provincia y haga clic en el ícono .
2. Acto seguido, lo llevará a seleccionar una pregunta. Haga clic en el ícono .
3. La aplicación mostrará: la pregunta seleccionada, la provincia seleccionada, una tabla con el resumen numérico de los datos y la representación de los mismos en un gráfico de pastel.

Figura 80. Interfaz gráfica resultados para una provincia

Elaboración: Borys Zuñiga

- **Consultar los datos de una pregunta por cantón**

  Muestra en resumen los datos de una pregunta seleccionada para un cantón en particular.

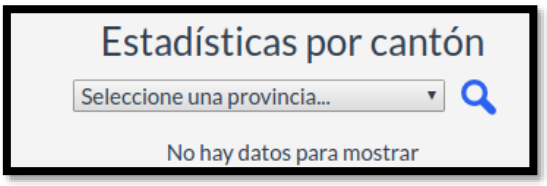


Figura 81. Interfaz gráfica del menú “Cantones”

Elaboración: Borys Zuñiga

4. Seleccione una provincia y haga clic en el ícono .
5. Acto seguido, la aplicación listará todos los cantones de la provincia seleccionada.

ID.	Cantones de la provincia de Azuay	Ver
1	Cuenca	
2	Girón	
3	Gualaceo	
4	Nabón	
5	Paute	
6	Pucará	

6. Seleccione un cantón y haga clic en el ícono .
7. Lo llevará a seleccionar una pregunta. Haga clic en el ícono .
8. La aplicación mostrará: la pregunta seleccionada, el cantón seleccionado, una tabla con el resumen numérico de los datos y la representación de los mismos en un gráfico de pastel.

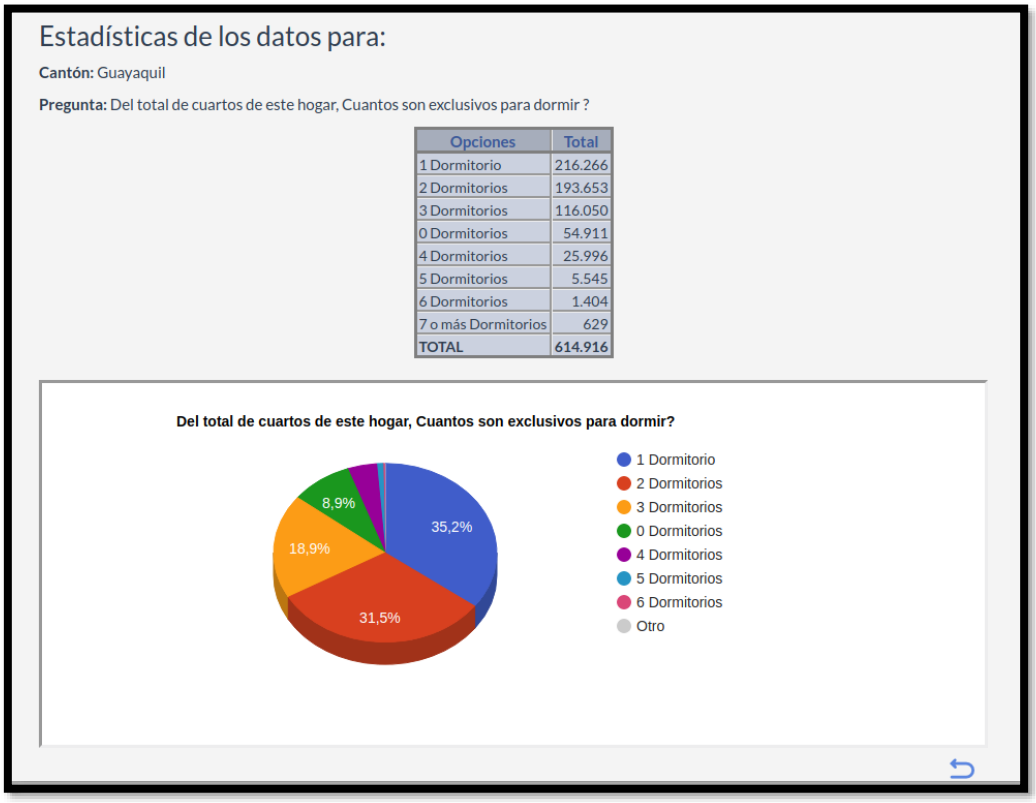


Figura 82. Interfaz gráfica que muestra los datos de un cantón

Elaboración: Borys Zuñiga

- **Consultar los datos de una pregunta por parroquia**

  Muestra en resumen los datos de una pregunta seleccionada para un cantón en particular.

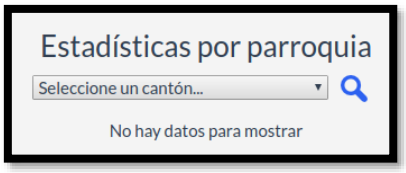


Figura 83. Interfaz gráfica del menú “Cantones”

Elaboración: Borys Zuñiga

1. Seleccione un cantón y haga clic en el ícono .
2. Acto seguido, la aplicación listará todas las parroquias del cantón seleccionado.

Figura 84. Interfaz gráfica que muestra las parroquias de un cantón

Elaboración: Borys Zuñiga

3. Seleccione una parroquia y haga clic en el ícono .
4. Lo llevará a seleccionar una pregunta. Haga clic en el ícono .
5. La aplicación mostrará: la pregunta seleccionada, la parroquia seleccionada, una tabla con el resumen numérico de los datos y la representación de los mismos en un gráfico de pastel.

Figura 85. Interfaz gráfica del menú “Cantones”

Elaboración: Borys Zuñiga

ANEXO F.

Consultas realizadas dentro de la Aplicación Web (Elastic Client)

- **Consultar las preguntas del censo.**

La consulta realizada devuelve: los documentos de las preguntas del censo, ordenados por id.

Archivo: routes/index.js

```
19        elasticsearch.elasticCliente().search({
20          index: 'censos',
21          requestTimeout: 60000,
22          body: {
23            sort : [
24                { id_pregunta: { order: 'asc' }}
25            ],
26              query:{
27                match: {
28                  _type: 'preguntas'
29                }
30            },
31            size: 1200
32          }
33      }).then(function (resp) {
```

Figura 86. Consulta para mostrar las preguntas del censo

Elaboración: Borys Zuñiga

Línea 19: se utiliza el método ***search()*** del API elasticsearch.js para realizar búsquedas, en donde, se envía como parámetro un objeto JSON.

Línea 20: ***index***, es el índice "*censos*" en donde se va a buscar.

Línea 21: ***requestTimeout***, es el tiempo máximo (milisegundos) que durará la petición de búsqueda.

Línea 22: ***body***, contiene el cuerpo de la consulta a realizar.

Línea 23: ***sort***, permite ordenar por uno o más campos específicos, en este caso ordena los documentos devueltos por "*id_pregunta*" de manera ascendente.

Línea 26: ***query***, contiene las cláusulas de la consulta.

Línea 27: ***match***, esta cláusula sirve para buscar texto completo en un campo. En este caso se la utiliza porque se desea buscar todos los documentos de tipo "*preguntas*".

Línea 28: ***_type***, es el campo por el que se está buscando, en este caso, se busca todos los documentos del tipo "*preguntas*".

Línea 31: ***size***, sirve para especificar el límite de documentos a devolver.

Nota importante: La misma estructura de la consulta se utiliza para buscar provincias, con la diferencia de que en lugar de enviar a buscar el tipo "encuestas", se envía "provincias".

- **Consultar los cantones de una provincia.**

La consulta realizada devuelve: los documentos de los cantones de una provincia, ordenados por identificador.

Archivo: routes/cantones.js

```
32  elasticsearch.elasticCliente().search({
33      index: 'censos',
34      body: {
35          sort : [
36              { id_canton: { order: 'asc' }}
37          ],
38            query:{
39              "has_parent": {
40                  "type": "provincias",
41                  "query": {
42                      "match": {
43                          "nombre_provincia": busqueda_provincia
44                      }
45                  }
46              }
47          },
48          size: 25
49      }
50  }).then(function (resp) {
```

Figura 87. Consulta para mostrar los cantones de una provincia

Elaboración: Borys Zuñiga

Línea 39: ***has_parent***, esta cláusula sirve para devolver documentos secundarios (hijos). En donde, se debe especificar un tipo "*provincias*" como documento principal.

Línea 43: es el nombre del campo a través del cual se está realizando la búsqueda, en este caso, *"nombre_provincia".*

Por ejemplo: si la provincia a buscar fuera Guayas, devolvería todos los hijos: Guayaquil, Durán, Naranjal, Milagro, etc.

- **Consultar las parroquias de un cantón.**

La consulta realizada devuelve: los documentos de las parroquias de un cantón, ordenados por identificador.

Archivo: routes/parroquias.js

```
47  elasticsearch.elasticCliente().search({
48      index: 'censos',
49      type: 'parroquias',
50      body: {
51          sort : [
52              { id_parroquia: { order: 'asc' }}
53          ],
54          query:{
55              "constant_score" : {
56                 "filter" : {
57                    "bool" : {
58                      "must" : [
59                         { "term" : {"codigo_provincia" : codigo_prov_enc}},
60                         { "term" : {"codigo_canton" : codigo_canton_enc}}
61                      ]
62                   }
63                }
64             }
65          },
66          size: 100
67      }
68  }).then(function (resp) {
```

Figura 88. Consulta para mostrar las parroquias de un cantón

Elaboración: Borys Zuñiga

Línea 55: ***constant_score***, esta cláusula sirve para devolver documentos con una puntuación constante, dependiendo de los filtros que se apliquen dentro de la misma.

Línea 57: ***bool***, cláusula que sirve para devolver documentos que resultan de combinaciones booleanas de otras consultas, entre ellas: *must, filter, should y must_not*.

Línea 58: ***must***, esta cláusula indica que los documentos **deben** aparecer de acuerdo a los criterios de búsqueda, además se les asignará un *score*, que determina el porcentaje de coincidencia del documento.

Línea 59 y 60: ***term***, devuelve documentos que contienen el termino exacto almacenado en el índice invertido.

código_provincia y **código_canton,** son los campos del documento a buscar.

- **Consultar las encuestas del censo a nivel general (todas las provincias).**

La consulta realizada devuelve: los datos estadísticos de las encuestas de todas las provincias, de una pregunta en particular.

Archivo: routes/mostrarDatos.js

```
19        elasticsearch.elasticCliente().search({
20          index: 'censos',
21          type: 'encuestas2',
22          requestTimeout: 130000,
23          body: {
24             "query": {
25                    "bool": {
26                      "must": [
27                        {
28                          "nested": {
29                            "path": "encuestas2",
30                            "query": {
31                              "bool": {
32                                "must": [
33                                  {
34                                    "match": {
35                                      "encuestas2.id_pregunta": codigo_preg_busq
36                                    }
37                                  }
38                                ]
39                              }
40                            }
41                          }
42                        }
43                      ]
44                    }
45                },
46                "aggs": {
47                    "ruta_encuestas": {
48                      "nested": {
49                        "path": "encuestas2"
50                      },
51                      "aggs": {
52                        "id_preguntas": {
53                          "terms": {
54                            "field": "encuestas2.id_pregunta",
55                            "size": 34
56                          },
57                          "aggs": {
58                            "respuestas": {
59                              "terms": {
60                                "field": "encuestas2.respuesta",
61                                "size": 50
62                              }
63                            }
64                          }
65                        }
66                      }
67                    }
68                },
```

Figura 89. Consulta para mostrar los datos estadísticos generales de las encuestas

Elaboración: Borys Zuñiga

Línea 20 y 21: ***index, type***, sirven para especificar el índice y tipo, en este caso “censos” y “encuestas2”, respectivamente, que es en donde se va a realizar la búsqueda.

Línea 25: ***bool***, cláusula que sirve para devolver documentos que resultan de combinaciones booleanas de otras consultas, entre ellas: *must, filter, should y must_not*.

Línea 26: ***must***, esta cláusula indica que los documentos **deben** aparecer de acuerdo a los criterios de búsqueda, además se les asignara un *score*, que determina el porcentaje de coincidencia del documento.

Línea 28: ***nested***, permite consultar objetos/documentos anidados. En este caso se desea realizar una consulta en el tipo “encuestas2” que contiene el campo del mismo nombre mapeado como un objeto anidado.

Línea 29: ***path***, especifica la ruta del campo que esta mapeado como un objeto anidado. El nombre del campo es “encuestas2”. Luego en las siguientes líneas se envían los criterios de búsqueda para un campo en específico, por ejemplo, en la línea 35, ***encuestas2.id_pregunta,*** se está buscando la pregunta solicitada por el usuario por identificador o código.

Línea 46: ***aggs***, sirve para hacer una agregación *single-value* (solo valor), es decir, cuenta el número de valores que se extraen de los documentos agregados previamente en la consulta realizada. Los valores pueden extraerse de campos específicos de los documentos o generarse mediante un script proporcionado. Por ejemplo, en este caso en particular se utilizan agregaciones para obtener estadísticas de las diferentes preguntas y respuestas de las encuestas.

Línea 47: ***ruta_encuestas***, es el nombre de la agregación que contendrá la ruta raíz del objeto a devolver como resultado. Posteriormente, se especifica el campo mapeado como objeto anidado sobre el que se va a aplicar la agregación.

Línea 52: ***id_preguntas***, es el nombre de la agregación que contendrá las diferentes opciones. En este caso, contiene los diferentes identificadores de las preguntas del censo, por ejemplo: URH, H01, H02, etc.

Línea 53: ***terms***, esta cláusula sirve para especificar que es una agregación basada en el origen de múltiples valores. Por ejemplo: URH: 81.562, H01: 40.455, H02: 65.765, etc., en este caso el resultado de la agregación sería, los valores con el número de documentos totales.

Línea 54: ***field***, es el campo sobre el cual se aplicará la agregación. En este caso es el campo ***encuestas2.id_preguntas.***

- **Consultar las encuestas del censo por provincia.**

La consulta realizada devuelve: los datos estadísticos de las encuestas de una provincia, de una pregunta en particular.

Archivo: routes/mostrarDatosProvincia.js

```
23          body: {
24             "query": {
25                    "bool": {
26                      "must": [
27                        {
28                          "nested": {
29                            "path": "encuestas2",
30                            "query": {
31                              "bool": {
32                                "must": [
33                                  {
34                                    "match": {
35                                      "encuestas2.id_pregunta": "I01"
36                                    }
37                                  },
38                          {
39                            "match": {
40                              "encuestas2.respuesta": codigo_provincia
41                            }
42                          }
43                                ]
44                              }
45                            }
46                          }
47                        }
48                      ]
49                    }
50                },
51                "aggs": {
```

Figura 90. Consulta para mostrar los datos estadísticos de las encuestas por provincia

Elaboración: Borys Zuñiga

Todas las cláusulas que se utilizaron en esta consulta, se explicaron en las consultas anteriores, por lo que en este caso se centrara en la explicación de la lógica de la consulta para obtener los datos estadísticos por provincia.

Para poder explicar el código, enfoque su mirada entre las líneas 26 y 48 de la imagen anterior. Lo que se está haciendo es, aplicar una consulta booleana que contiene una subconsulta. en la que se solicitan todos los documentos que coincidan con el identificador de la pregunta “I01”, y a su vez; que coincidan con la respuesta que se está enviando a buscar. No se olvide que, “I01” es la etiqueta de la pregunta que almacena el identificador de la provincia. Para lograr una mejor comprensión repase el ANEXO B y ANEXO C.

Importante: ***código_provincia***, representan la variable dentro de la aplicación, la cual almacena el identificador de la provincia a buscar.

- **Consultar las encuestas del censo por cantón.**

La consulta realizada devuelve: los datos estadísticos de las encuestas de un cantón, de una pregunta en particular.

Archivo: routes/mostrarDatosCanton.js

```
            body: {
               "query": {
                     "bool": {
                       "must": [
                         {
                           "nested": {
                             "path": "encuestas2",
                             "query": {
                               "bool": {
                                 "must": [
                                   {
                                     "match": {
                                       "encuestas2.id_pregunta": "I01"
                                     }
                                   },
                                   {
                                     "match": {
                                       "encuestas2.respuesta": codigo_provincia
                                     }
                                   }
                                 ]
                               }
                             }
                           }
                         },
                         {
                           "nested": {
                             "path": "encuestas2",
                             "query": {
                               "bool": {
                                 "must": [
                                   {
                                     "match": {
                                       "encuestas2.id_pregunta": "I02"
                                     }
                                   },
                                       {
                                     "match": {
                                       "encuestas2.respuesta": codigo_canton
                                     }
                                   }
                                 ]
                               }
                             }
                           }
                         }
                       ]
                     }
                   },
                   "aggs": {
```

Figura 91. Consulta para mostrar los datos estadísticos de las encuestas por cantón

Elaboración: Borys Zuñiga

Todas las cláusulas que se utilizaron en esta consulta, se explicaron en las consultas anteriores, por lo que en este caso se centrara en la explicación de la lógica de la consulta para obtener los datos estadísticos por cantón.

Para poder explicar el código, enfoque su mirada entre las líneas 27 y 70 de la imagen anterior. Lo que se está haciendo es, aplicar una consulta booleana que contiene dos subconsultas. En la primera, se solicitan todos los documentos que coincidan con el identificador de la pregunta “I01”, y a su vez; que coincidan con la respuesta que se está enviando a buscar. No se olvide que, “I01” es la etiqueta de la pregunta que almacena el identificador de la provincia; **y**, en la segunda, se solicitan todos los documentos que coincidan con el identificador de la pregunta “I02”, y a su vez; que coincidan con la respuesta que se está enviando a buscar. No se olvide que, “I02” es la etiqueta de la pregunta que almacena el identificador del cantón. Dicho de otra manera, se aplica un doble filtro, sobre un mismo campo que contiene diferentes identificadores de las preguntas.

Para lograr una mejor comprensión repase el ANEXO B y ANEXO C.

Importante: ***código_provincia*** y ***código_canton***, representan las variables dentro de la aplicación, las cuales almacenan el identificador de la provincia y cantón a buscar, respectivamente.

- **Consultar las encuestas del censo por parroquia.**

La consulta realizada devuelve: los datos estadísticos de las encuestas de una parroquia, de una pregunta en particular.

Archivo: routes/mostrarDatosParroquia.js

Figura 92. Consulta para mostrar los datos estadísticos de las encuestas por parroquia

Elaboración: Borys Zuñiga

Todas las cláusulas que se utilizaron en esta consulta, se explicaron en las consultas anteriores, por lo que en este caso se centrará en la explicación de la lógica de la consulta para obtener los datos estadísticos por parroquia.

Al igual que la lógica de la consulta anterior por cantón, con la diferencia que aquí además de solicitar los documentos por provincia y por cantón, se agrega una subconsulta más, en la cual se solicitan todos los documentos que coincidan con el identificador de la pregunta “I03”, y a su

vez; que coincidan con la respuesta que se está enviando a buscar. No se olvide que, “I03” es la etiqueta de la pregunta que almacena el identificador del cantón. En otras palabras, se aplica un triple filtro sobre un mismo campo que contiene diferentes identificadores de las preguntas.

Para lograr una mejor comprensión repase el ANEXO B y ANEXO C.

Importante: ***código_parroquia***, representa la variable dentro de la aplicación, que almacena la parroquia a buscar.