

# Comparación de Despliegue de un Gestor de Base de Datos NoSQL Mediante Docker

MARKO ANTONIO RIVAS RIOS

JORGE LUIS MAMANI MAQUERA

ORLANDO ANTONIO ACOSTA ORTIZ

YOFER NAIN CATARI CABRERA

ORESTES RAMIREZ TICONA

ROBERTO ZEGARRA REYES

Universidad Privada de Tacna

Junio 22, 2019

## Abstract

### Resumen

*Docker es un proyecto open source creado en 2013 y que ha supuesto una revolución para el desarrollo y despliegue de operaciones. Docker abstrae el hardware y el sistema operativo del host ejecutando las aplicaciones en contenedores, compartimentos aislados que contienen todos los recursos para una aplicación o servicio. En este trabajo veremos cómo usar Docker para el desarrollo de aplicaciones sencillas, aprendiendo a desplegar una base de datos NOSQL con Docker.*

### Abstract

*Docker is an open source project created in 2013 and which has been a revolution for the development and deployment of operations. Docker abstracts the host's hardware and operating system by running the applications in containers, isolated compartments that contain all the resources for an application or service. In this work we will see how to use Docker for the development of simple applications, learning how to deploy a NOSQL database with Docker.*

## I. INTRODUCCIÓN

EL potente concepto de Microservicios está cambiando poco a poco la industria. Grandes servicios monolíticos están dando paso lentamente al enjambre microservicios pequeños y autónomos que trabajan en conjunto. El proceso va acompañado de otra tendencia del mercado: la contenerización. Juntos, ayudan a construir sistemas sin precedentes. La contenerización cambia no sólo la arquitectura de los servicios, sino también la estructura de ambientes utilizados para crearlos.

Ahora, cuando el software se distribuye en contenedores, los desarrolladores tienen plena libertad para decidir qué aplicaciones necesitan. Como resultado, incluso los entornos complejos, como los servidores de grandes bases de datos e infraestructura de análisis complejos pueden crear instancias en cuestión de segundos. El desarrollo de software se hace más fácil y más eficaz.

## II. MATERIALES Y MÉTODOS

### i. Materiales

- Virtualización activada en el BIOS
- Docker Desktop
- Windows 10 64bit: Pro, Enterprise o Education, con al menos 4GB de RAM.

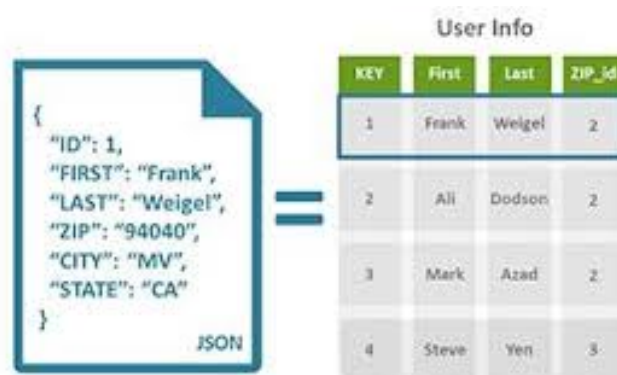
### ii. Métodos

- Se utilizó como material artículos y libros relacionados a la base de datos NoSQL y sus tipos, así como páginas web.

## III. MARCO TEÓRICO

### i. Creación de una base de datos NoSQL en docker

### ii. Inserción de datos y Consulta de datos (en una base de datos NOSQL)



Existen varias diferencias con respecto a cómo los distintos tipos de bases de datos permiten a los usuarios / aplicaciones realizar consultas. Desde las consultas más básicas por clave primaria, como por ejemplo, los almacenes clave – valor, pasando por otros que ofrecen un acceso a la información algo más complejo. En este terreno se encontrarían las bases de datos documentales.

SQL	MongoDB
CREATE DATABASE empresa;	USE empresa
CREATE TABLE dpto ( dep NUMBER(3) PRIMARY KEY, nom VARCHAR(20) NOT NULL);	db.createCollection("dpto")
CREATE TABLE empleado ( id NUMBER(4) PRIMARY KEY, nombre VARCHAR(20) NOT NULL, edad VARCHAR(20) NOT NULL, grado NUMBER(3) NOT NULL, dep NUMBER(3) REFERENCES dpto);	db.createCollection("empleado")
CREATE INDEX indiceDep ON empleado (dep);	db.empleado.ensureIndex({dep: 1}) #Suponiendo que empleado tiene una clave dep.
DROP TABLE empleado;	db.empleado.drop()

En general, la flexibilidad y la riqueza de las queries no son demasiado elevadas, puesto que lo que se prima por encima de las consultas es el rendimiento y la escalabilidad, por lo que es habitual que se delegue a la aplicación el implementar opciones más avanzadas en este terreno.

Un método de consulta llamado Companion SQL database consiste en tener una base de datos auxiliar (que puede ser una base de datos SQL o una TextDB) de forma que se utilice esta secundaria para almacenar ciertos metadatos importantes para realizar la búsqueda, y se empleen para facilitar la búsqueda posterior en el contenedor NoSQL.

SQL	MongoDB	Resultado en MongoDB
SELECT * FROM empleado WHERE edad < 30 AND grado = 8;	db.empleado.find({edad: {\$lt: 30}, grado: 8})	{ "_id": 1, "nombre": "Juan", "edad": 23, "grado": 8, "dep": 10 }
SELECT * FROM empleado WHERE nombre IN ( 'Pablo', 'Ana' );	db.empleado.find( { nombre: { \$in: [ "Pablo", "Ana" ] } } )	{ "_id": 2, "nombre": "Pablo", "edad": 25, "grado": 10, "dep": 10 } { "_id": 3, "nombre": "Ana", "edad": 11, "grado": 6, "dep": 20 }
SELECT * FROM empleado WHERE nombre LIKE "%blo";	db.empleado.find({ nombre: /blo/ })	{ "_id": 2, "nombre": "Pablo", "edad": 25, "grado": 10, "dep": 10 }
SELECT SUM(edad) AS total FROM empleado;	db.empleado.aggregate([ { \$group: { _id: null, total: { \$sum: "\$edad" } } ] )	[ { "_id": null, "total": 69 } ]
SELECT dep, MAX(grado) AS mg FROM empleado GROUP BY dep;	db.empleado.aggregate([ { \$group: { _id: "\$dep", mg: { \$max: "\$grado" } } ] )	[ { "_id": 10, "max": 10 }, { "_id": 20, "mg": 6 } ]

Búsqueda local dispersa. Otra forma de realizar consultas consiste en, puesto que se tiene el conjunto de datos repartido entre los distintos servidores, repartir de igual modo la consulta, de forma que cada servidor ejecute localmente cada consulta y reenvíe los resultados a un nodo maestro, que sería el encargado de juntar todos los resultados y presentárselos a la aplicación.

Arboles B+ Distribuidos.

Una forma eficiente de acelerar las búsquedas consiste en mantener un árbol B+ que forme un índice de entradas a la base de datos NoSQL (Aquilera, Golab and Shah 2008).

El procedimiento consistiría en sacar los valores hash de los atributos que nos interese indexar, y construir con ellos el árbol B+. Cuando se quiera realizar una consulta, se comenzará desde la raíz y se irá descendiendo en orden hasta llegar a la hoja correspondiente, que nos dará la entrada concreta donde se encuentra el registro que se está buscando.

Operación	SQL	MongoDB
Creación base de datos	CREATE DATABASE nombreBD;	USE nombreBD
Creación tabla (en SQL) / Colección (en MongoDB)	CREATE TABLE nombreTabla ( atributo1 tipo_de_dato restricciones <, atributo2...>);	db.createCollection("nombreColección")
Creación índice	CREATE INDEX nombreIndice ON nombreTabla (atributo1 <, atributo2... >);	db.nombreColeccion.ensureIndex( { "atributo": 1 })
Destrucción tabla (en SQL) / Colección (en MongoDB)	DROP TABLE nombreTabla;	db.nombreColeccion.drop()





Al tratarse de un árbol B+, se debe tener en cuenta las particularidades de este tipo de estructura de datos a la hora de realizar los mantenimientos necesarios, las inserciones y borrados que puedan hacer redimensiones en el árbol, etc.

### iii. Comparacion de distintos tipos de base de datos NoSQL

Dependiendo de la forma en la que almacenen la información, nos podemos encontrar varios tipos distintos de bases de datos NoSQL. Veamos los tipos más utilizados.

Bases de datos clave – valor

- Bases de datos documentales
- Bases de datos en grafo
- Bases de datos orientadas a objetos

<b>Documentales</b>	Datos semi-estructurados en documentos (XML, YAML, JSON y BSON)	
<b>Grafo</b>	Datos estructurados como nodos relacionados entre si	
<b>Clave / valor</b>	Datos estructurados como clave / valor	
<b>Familia de columnas</b>	Datos estructurados en columnas donde cada fila puede tener una configuración diferente	

#### CouchDB:

CouchDB es catalogado muchas veces como una base de datos NoSQL, un término que se hizo cada vez más popular a finales de 2009 y principios de 2010. Si bien este término es una caracterización más bien genérica de una base de datos, o almacén de datos, sí define claramente un descanso de SQL tradicional bases de datos. Una base de datos CouchDB carece de un esquema o estructuras de datos pre-definidos rígidos tales como tablas.

Los datos almacenados en CouchDB es un documento (s) JSON. La estructura de los datos, o documento(s), puede cambiar dinámicamente para adaptarse a las necesidades cambiantes. CouchDB es una base de datos que abarca por completo la web. Almacene sus datos con documentos JSON. Tenga acceso a sus documentos y consultar sus índices con su navegador web, a través de HTTP.Índice, combinar y transformar sus documentos con JavaScript. CouchDB funciona bien con la web moderna y aplicaciones móviles. Usted puede incluso servir aplicaciones web directamente de CouchDB. Y usted puede distribuir sus datos o

sus aplicaciones, de manera eficiente mediante la replicación incremental de los CouchDB. CouchDB soporta configuraciones maestro-maestro con detección automática de conflictos. CouchDB viene con una serie de características, como la transformación de documentos sobre la marcha y notificaciones de cambio en tiempo real, que hace que el desarrollo de aplicaciones web una brisa. Incluso viene con un fácil utilizar la consola de administración web.

**Las principales características son las siguientes:**

- **Almacenamiento de documentos:**

Almacena los datos como documentos esto es, uno o más pares campo/valor expresados en JSON. Los valores de los campos pueden ser datos simples como cadenas de caracteres, números o fechas. Pero también se pueden usar listas ordenadas y vectores asociativos. Todos los documentos en una base de datos CouchDB tienen un identificador único y no requieren un esquema determinado.

- **Vistas e índices Map/Reduce:**

Los datos almacenados se estructuran por medio de vistas. En CouchDB, cada vista se construye por medio de una función JavaScript que actúa como la mitad Map de una operación map/reduce. La función recibe un documento y lo transforma en un único valor, retornándolo. CouchDB puede indexar vistas y mantener actualizados esos índices a medida que se agregan, eliminan o actualizan documentos.

- **Arquitectura distribuida con replicación:**

Se diseñó con teniendo en mente la replicación bidireccional (o sincronización) y la operación off-line. Eso significa que múltiples réplicas pueden tener cada una sus propias copias de los mismos datos, modificarlas y luego sincronizar esos cambios en un momento posterior.

- **Interfaz REST:**

Todos los ítems tienen una URI única que queda expuesta vía HTTP. REST usa los métodos HTTP POST, GET, PUT y DELETE para las cuatro operaciones básicas CRUD (Create, Read, Update, Delete) con todos los recursos.

- **Consistencia Eventual:**

Garantiza consistencia eventual para poder ofrecer tanto disponibilidad como tolerancia a las particiones.

- **Hecha para operar offline:**

Puede replicar datos a dispositivos (como smartphones) que pueden quedar offline y manejar automáticamente la sincronización de los datos cuando el dispositivo vuelve a estar en línea.

### **Neo4j:**

Es una base de datos orientada a grafos escrita en Java, es decir la información se almacena de forma relacionada formando un grafo dirigido entre los nodos y las relaciones entre ellos. Se integra perfectamente con múltiples lenguajes como Java, PHP, Ruby, .Net, Python, Node, Scala, etc. La base de datos está embebida en un servidor Jetty. Está especialmente indicada

para modelar redes sociales y sistemas de recomendación.

Se distribuye en dos versiones: la community edition (open source) y la Enterprise edition. Para hacer pruebas de concepto nos basta con la community edition pero si quieres sacarle todo el partido a Neo4j la opción enterprise es la más recomendable ya que permite ponerla en cluster, monitorización, backups en caliente y un sistema de cache de alto rendimiento, además de soporte de sus creadores.

Otra de las ventajas que tiene Neo4j es que se pueden efectuar las consultas directamente a través de un API Rest lo que hace especialmente interesante su integración con aplicaciones web.

**Principales características de neo4j:**

- Alto desempeño y alta disponibilidad (Escalamiento de lectura) Soporte sólido y real para transacciones ACID.
- Escalable: 32 miles de millones de Nodos, 32 miles de millones de Relaciones, 64 miles de millones de Propiedades.
- Servidor con una API REST o usable como una biblioteca Java.

## IV. RESULTADOS

### Comparaciones de 2 Bases de Datos NoSQL

i. Grafos

ii. Tabular (Column-Store)

iii. Documental

iv. Clave-Valor

## V. DISCUSIÓN

## VI. CONCLUSIONES

### REFERENCES

1. <http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/23/21>
2. <https://programarfacil.com/blog/que-es-un-orm/>
3. <https://www.beeva.com/beeva-view/tecnologia/mas-alla-de-la-virtualizacion-contenedores/>
4. <https://searchdatacenter.techtarget.com/es/definicion/virtualizacion-basada-en-contenedores-virtualizacion-a-nivel-de-sistema-operativo>
5. <https://www.incibe-cert.es/blog/asegurando-virtualizacion-tus-sistemas-control>
6. <http://www.datakeeper.es/?p=716>
7. <https://sigmodrecord.org/publications/sigmodRecord/1012/pdfs/04.surveys.cattell.pdf>
8. <https://ieeexplore.ieee.org/abstract/document/6625441>
9. <http://nosql-database.org/>