|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| logo berri CIFP  CIFP Ciudad Jardín LHII | | | **Curso de Especialización:** Inteligencia artificial y Big Data | | | |
| **Modulo:**  Sistemas de Big Data  Big Data Aplicado | | | **Cód. Modulo:**  SBIG  BIGD |
| **Curso:** IABD | | | |
| **1ª Evaluación** | | | **Fecha**: 21/12/2022 | | | |
| **Cód. Prof:** PTI9  PTI3 | | **Profesor:** Andoni Inza  Ainara Montoya | |  | | |
|  | | | | | | |
| **Datos del Alumno** | | | | | | |
| **Nº** | **Nombre:** | | | | **Firma** | |
| **Apellido 1:** | | | | |
| **Apellido 2:** | | | | |

**Calificación:**

Para realizar este examen se aconseja utilizar los documentos de teoría y ejercicios realizados durante el trimestre actual, que están en Moodle. Está prohibido utilizar conexión a internet durante la prueba. Para aprobación del examen, habrá que puntuar mínimo en 2 de los ejercicios propuestos y que la puntuación total sea como mínimo 5.

Todos los pasos del examen tienen que estar documentados en este documento, todas las capturas de pantalla deberán enseñar el nombre del alumno, las modificaciones de archivos, el comando realizado y su salida.

## EJERCICIO 1

## HADOOP

### Creación de clúster Hadoop y yarn

Partiendo de la máquina virtual maestra proporcionada, monta un clúster de 2 nodos. Conexión NAT:

IP Maestro: 192.0.2.130

IP Nodo 1: 192.0.2.140

IP Nodo 2: 192.0.2.150

### Comprobacion DE FUNCIONAMIENTO clúster HADOOP

1. Pon en marcha el sistema hadoop, y verifica que están todos los nodos activos y en marcha.

Verifica los servidores web de HDFS y YARN.

1. Crea una carpeta y sube un archivo. (usar fichero “Subtitulos\_GameOfThrone.json”)
2. Crear un “Wordcount” del archivo subido y verifica el archivo de salida.

Comprueba el tiempo de ejecución del wordcount. Muestra los parámetros más relevantes.

1. Apaga uno de los nodos, y verifica que los archivos son accesibles, y si la ejecución del “Wordcount” es afectado.

**Configuración del Clúster Hadoop**

**Pasos para montar un clúster Hadoop con 2 nodos:**

1. **Configurar el clúster en cada máquina virtual:**
   * Configuración de la máquina maestra (192.0.2.130) como **NameNode**.
   * Configurar el Nodo 1 (192.0.2.140) y Nodo 2 (192.0.2.150) como **DataNodes**.
   * Utilizar la red NAT para comunicación.
2. **Configuración básica en cada nodo:**
   * Editar /etc/hosts para registrar las IP de los nodos:

192.0.2.130 master

192.0.2.140 node1

192.0.2.150 node2

* + Configurar variables de entorno en .bashrc o .profile:

export HADOOP\_HOME=/usr/local/hadoop

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

1. **Configuración de los archivos de Hadoop:**
   * **core-site.xml:** En el directorio $HADOOP\_HOME/etc/hadoop, configurar:

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://master:9000</value>

</property>

</configuration>

* + **hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

</configuration>

* + **yarn-site.xml:**

<configuration>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>master</value>

</property>

</configuration>

* + **slaves:** Lista de nodos esclavos (en todos los nodos):

node1

node2

1. **Inicialización del clúster:**
   * Formatear el sistema HDFS en el nodo maestro:

hdfs namenode -format

* + Iniciar los servicios:

start-dfs.sh

start-yarn.sh

**Verificación de Funcionamiento**

1. **Comprobación de nodos activos:**
   * Comprobar nodos del clúster:

hdfs dfsadmin -report

* + Comprobar el estado de los nodos YARN:

yarn node -list

1. **Verificar servidores web:**
   * HDFS: http://192.0.2.130:9870
   * YARN: http://192.0.2.130:8088

**Operaciones de Hadoop**

**1. Crear una carpeta y subir un archivo**

hdfs dfs -mkdir /gameofthrones

hdfs dfs -put Subtitulos\_GameOfThrone.json /gameofthrones/

**2. Ejecutar Wordcount**

* Preparar un archivo JAR para Wordcount:

hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples.jar wordcount /gameofthrones/Subtitulos\_GameOfThrone.json /gameofthrones/output

* Comprobar los resultados:

hdfs dfs -cat /gameofthrones/output/part-r-00000

**3. Tiempo de ejecución**

* Al ejecutar el comando anterior, el tiempo de ejecución aparece en los logs del proceso.  
  Utilizar:

time hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples.jar wordcount /path/to/input /path/to/output

**4. Apagar un nodo y verificar accesibilidad**

* Apagar Nodo 1:

ssh node1 "sudo shutdown now"

* Comprobar accesibilidad a los archivos:

hdfs dfs -ls /gameofthrones/

* Repetir el Wordcount y evaluar si se afecta el rendimiento.

**Parámetros Relevantes**

1. **Número de réplicas:** Configurado en hdfs-site.xml (dfs.replication).
2. **Tamaño de bloque:** Configurado en hdfs-site.xml (dfs.blocksize).
3. **Logs de ejecución:** Verificables en /usr/local/hadoop/logs.

Estos pasos aseguran la configuración y pruebas del clúster Hadoop.

4o

## EJERCICIO 2

## MONGODB

### Procesamiento de datos con MongoDB

**Creación de las colecciones y documentos de las Base de datos**

Considera el escenario de una aplicación de gestión de una biblioteca. Tu tarea es diseñar e implementar las colecciones necesarias en MongoDB para gestionar la información de libros, autores y préstamos. Utiliza las teorías de relaciones (1:1, 1:pocos, 1:muchos, muchos:muchos) para modelar de manera eficiente la base de datos.

Entidades Principales:

1. Cada libro tiene un ISBN único, título, año de publicación y género. Pueden tener uno o varios autores. 3 reseñas con calificación como máximo, y creadas por usuarios del apartado préstamo.
2. Autores: Numero único, nombre y nacionalidad;
3. Prestamos de los libros asociados a un libro y usuario, y puede tener varios prestamos asociados a lo largo del tiempo.
4. Un libro puede tener varios géneros.

Asegúrate de incluir las referencias adecuadas en cada colección para establecer las relaciones. Utiliza identificadores únicos para enlazar los documentos entre colecciones.

Diseña índices adecuados para optimizar las consultas relacionadas con las relaciones establecidas.

Crea ejemplos de documentos para cada colección que demuestren cómo se relacionan entre sí. Puedes incluir algunos datos ficticios para ilustrar el diseño de la base de datos.

Crea una base de datos e inserta varios documentos en las diferentes colecciones para los ejercicios.

**Colección: libros**

json

Copiar código

{

"\_id": "ISBN123456789",

"titulo": "El Nombre del Viento",

"anio\_publicacion": 2007,

"generos": ["Fantasía", "Aventura"],

"autores": ["AUTH001", "AUTH002"],

"reseñas": [

{

"usuario\_id": "USR001",

"calificacion": 5,

"comentario": "Excelente libro."

},

{

"usuario\_id": "USR002",

"calificacion": 4,

"comentario": "Muy entretenido."

}

]

}

**Colección: autores**

json

Copiar código

{

"\_id": "AUTH001",

"nombre": "Patrick Rothfuss",

"nacionalidad": "Estadounidense"

}

**Colección: usuarios**

json

Copiar código

{

"\_id": "USR001",

"nombre": "Juan Pérez",

"email": "juan.perez@gmail.com"

}

**Colección: prestamos**

json

Copiar código

{

"\_id": "PREST001",

"usuario\_id": "USR001",

"libro\_id": "ISBN123456789",

"fecha\_prestamo": ISODate("2024-01-10T10:00:00Z"),

"fecha\_devolucion": ISODate("2024-01-20T10:00:00Z")

}

**Ejercicios CRUD**

1. Inserta un nuevo libro en la colección "Libros". Asegúrate de proporcionar valores para todos los campos necesarios, como ISBN, título, año de publicación, géneros, autores, etc.

db.libros.insertOne({

\_id: "ISBN987654321",

titulo: "Juego de Tronos",

anio\_publicacion: 1996,

generos: ["Fantasía", "Drama"],

autores: ["AUTH003"],

reseñas: []

});

1. Recupera la información de un libro específico utilizando su ISBN. Asegúrate de visualizar solo detalles sobre el nombre del libro, los autores, géneros y reseñas asociadas.

db.libros.find(

{ \_id: "ISBN123456789" },

{ titulo: 1, autores: 1, generos: 1, reseñas: 1 }

);

1. Inserta 2 préstamos nuevos asociados a algún usuario existente

db.prestamos.insertMany([

{

\_id: "PREST002",

usuario\_id: "USR001",

libro\_id: "ISBN123456789",

fecha\_prestamo: ISODate("2024-01-15T10:00:00Z"),

fecha\_devolucion: null

},

{

\_id: "PREST003",

usuario\_id: "USR002",

libro\_id: "ISBN987654321",

fecha\_prestamo: ISODate("2024-01-16T10:00:00Z"),

fecha\_devolucion: null

}

]);

1. Actualiza los datos de entrega de los prestamos anteriores

db.prestamos.updateMany(

{ \_id: { $in: ["PREST002", "PREST003"] } },

{ $set: { fecha\_devolucion: ISODate("2024-02-01T10:00:00Z") } }

);

**Ejercicios AGREGACIONES.**

1. Obtiene los primeros 5 préstamos de la colección "Prestamos" ordenados por fecha de préstamo. (más nuevo a más antiguo)

db.prestamos.aggregate([

{ $sort: { fecha\_prestamo: -1 } },

{ $limit: 5 }

]);

1. Agrupa los libros por género y calcula el número total de libros en cada género.

db.libros.aggregate([

{ $unwind: "$generos" },

{ $group: { \_id: "$generos", total\_libros: { $sum: 1 } } },

{ $sort: { total\_libros: -1 } }

]);

1. Obtiene el libro con la mejor calificación, y visualiza su nombre, autor/es, reseña, usuario.

db.libros.aggregate([

{ $unwind: "$reseñas" },

{ $group: {

\_id: "$\_id",

titulo: { $first: "$titulo" },

autores: { $first: "$autores" },

mejor\_calificacion: { $max: "$reseñas.calificacion" },

reseña: { $first: "$reseñas" }

}},

{ $sort: { mejor\_calificacion: -1 } },

{ $limit: 1 }

]);

1. Agrupa los préstamos por libro y cuenta cuántas veces cada libro ha sido prestado. Ordena los resultados por el número de préstamos en orden descendente.

db.prestamos.aggregate([

{ $group: { \_id: "$libro\_id", total\_prestamos: { $sum: 1 } } },

{ $sort: { total\_prestamos: -1 } }

]);

1. Lista de autores con la cantidad de libros escritos (filtrando aquellos con al menos 2 libros)

db.libros.aggregate([

{ $unwind: "$autores" },

{ $group: { \_id: "$autores", total\_libros: { $sum: 1 } } },

{ $match: { total\_libros: { $gte: 2 } } },

{ $lookup: {

from: "autores",

localField: "\_id",

foreignField: "\_id",

as: "autor\_info"

}},

{ $project: { \_id: 0, autor\_info: 1, total\_libros: 1 } }

]);

NOTA: Se podrá usar la terminal MongoSH, Python o la aplicación MongoCompass para realizar estos ejercicios. Se valorará usar diferentes vías de resolución.

### MONGODB REPLICAS

Crear dos conjuntos de réplicas de dos nodos cada uno, y a su vez, particionaremos los datos en dos y tres shaders respetivamente, usando docker-compose en una máquina virtual con Ubuntu 22.

Además, añadir un único router y un servidor de configuración.

Para el ejercicio, en la carpeta de recursos tenéis un fichero llamado images.json

1. Importar y particionar los datos por la id del image.json dataset.
2. Una vez cargado los datos, obtener el estado del sharding.
3. Si los datos no están particionados, forzar el split de los mismos, y obtener el estado del sharding.

**1. Configuración del docker-compose.yml**

**Estructura de Réplicas y Sharding:**

* **Config Server Replica Set**:
  + Un único servidor de configuración (configsvr1) que será el rs-config-server.
* **Shards**:
  + **Shard 1**: Réplica de dos nodos (shard1a, shard1b).
  + **Shard 2**: Réplica de dos nodos (shard2a, shard2b).
* **Router**:
  + Un router (router1) para interactuar con los shards y el servidor de configuración.

**Actualización del docker-compose.yml:**

version: '3.8'

services:

## Router

router1:

image: mongo

container\_name: router1

command: mongos --configdb rs-config-server/configsvr1:27017 --bind\_ip\_all

ports:

- 27017:27017

restart: always

depends\_on:

- configsvr1

- shard1a

- shard2a

networks:

- mongo-network-sharded

## Config Server

configsvr1:

image: mongo

container\_name: configsvr1

command: mongod --configsvr --replSet rs-config-server --bind\_ip\_all

volumes:

- mongodb\_cluster\_configsvr1:/data/db

networks:

- mongo-network-sharded

## Shard 1 Replica Set

shard1a:

image: mongo

container\_name: shard1a

command: mongod --shardsvr --replSet rshard1 --bind\_ip\_all

volumes:

- mongodb\_cluster\_shard1a:/data/db

networks:

- mongo-network-sharded

shard1b:

image: mongo

container\_name: shard1b

command: mongod --shardsvr --replSet rshard1 --bind\_ip\_all

volumes:

- mongodb\_cluster\_shard1b:/data/db

networks:

- mongo-network-sharded

## Shard 2 Replica Set

shard2a:

image: mongo

container\_name: shard2a

command: mongod --shardsvr --replSet rshard2 --bind\_ip\_all

volumes:

- mongodb\_cluster\_shard2a:/data/db

networks:

- mongo-network-sharded

shard2b:

image: mongo

container\_name: shard2b

command: mongod --shardsvr --replSet rshard2 --bind\_ip\_all

volumes:

- mongodb\_cluster\_shard2b:/data/db

networks:

- mongo-network-sharded

volumes:

mongodb\_cluster\_configsvr1:

mongodb\_cluster\_shard1a:

mongodb\_cluster\_shard1b:

mongodb\_cluster\_shard2a:

mongodb\_cluster\_shard2b:

networks:

mongo-network-sharded:

driver: bridge

**2. Inicialización de Réplicas y Sharding**

**Inicializar el Config Server Replica Set**

1. Conéctate al contenedor configsvr1:

docker exec -it configsvr1 mongosh

1. Configura el replica set:

rs.initiate({

\_id: "rs-config-server",

configsvr: true,

members: [

{ \_id: 0, host: "configsvr1:27017" }

]

});

**Inicializar los Shards**

1. **Shard 1 Replica Set**:
   * Conéctate al contenedor shard1a y ejecuta:

docker exec -it shard1a mongosh

* + Inicializa el replica set:

rs.initiate({

\_id: "rshard1",

members: [

{ \_id: 0, host: "shard1a:27017" },

{ \_id: 1, host: "shard1b:27017" }

]

});

1. **Shard 2 Replica Set**:
   * Conéctate al contenedor shard2a y ejecuta:

docker exec -it shard2a mongosh

* + Inicializa el replica set:

rs.initiate({

\_id: "rshard2",

members: [

{ \_id: 0, host: "shard2a:27017" },

{ \_id: 1, host: "shard2b:27017" }

]

});

**Configurar el Router**

1. Conéctate al router:

docker exec -it router1 mongosh

1. Añade los shards:

sh.addShard("rshard1/shard1a:27017,shard1b:27017");

sh.addShard("rshard2/shard2a:27017,shard2b:27017");

1. Habilita el sharding para la base de datos:

sh.enableSharding("imagesDB");

1. Configura el particionamiento:

sh.shardCollection("imagesDB.images", { id: "hashed" });

**3. Importar y Particionar Datos**

1. Copia images.json al contenedor del router:

docker cp images.json router1:/images.json

1. Importa los datos:

docker exec -it router1 mongosh

use imagesDB;

db.images.insertMany([

/\* Contenido del archivo JSON \*/

]);

1. Verifica el estado del sharding:

sh.status();

**4. Forzar Split si los Datos no Están Particionados**

Si los datos no se distribuyen automáticamente:

1. Divide manualmente el rango:

sh.splitAt("imagesDB.images", { id: 1000 });

1. Mueve un fragmento a otro shard:

sh.moveChunk("imagesDB.images", { id: 500 }, "rshard2");

**5. Comprobar Estado del Sharding**

Ejecuta el siguiente comando:

sh.status();

El estado debe mostrar los shards activos y el particionamiento de la colección images.

## CRITERIOS DE EVALUACIÓN

|  |  |
| --- | --- |
| EJERCICIO 1 | PUNTUACIÓN |
| Correcta instalación y configuración de Hadoop y verificación de conectividad entre master y nodos | 0..5 |
| Creación exitosa de carpetas y correcta subida de archivos en el sistema de archivos distribuido HDFS | 0.5 |
| Ejecución exitosa de “Wordcount”, y verificación del resultado correctos | 0.5 |
| Extra: Realizar el “Wordcount” en python | 0.5 |
| Verificación de tiempo de ejecución en el entorno web | 0.5 |
| Apagado controlado de un nodo, y verificación de disponibilidad de archivos | 0.5 |

|  |  |
| --- | --- |
| EJERCICIO 2 | PUNTUACIÓN |
| Diseño y creación de la base de datos. Claridad y coherencia de los datos. Consideración adecuada de relaciones entre colecciones. | 2 |
| Implementación correcta de las consultas CRUD | 0.6 |
| Uso adecuado de operadores de agregación para obtener resultados deseados. | 2 |
| Extras: |  |
| Uso de un servidor en docker | 0.1 |
| Uso de diferentes medios para consultas | 0.3 |

|  |  |
| --- | --- |
| EJERCICIO 3 | PUNTUACIÓN |
| Correcta configuración de los ficheros Docker-compose.yml e init.sh de todos los elementos. | 1 |
| Importación de archivos y verificación de estado del sharding | 1 |