Bases de datos a grande escala [P4181101] [2021/2022]

Práctica 4: Bases de datos NoSQL: Documentales

Marcar como feito

4.- Bases de datos NoSQL: Documentales

En esta práctica haremos una introducción al uso de la base de datos NoSQL de tipo documental MongoDB. En una primera parte probaremos la funcionalidad que proporciona para la creación de documentos, la lectura, modificación y borrado de los mismos y el uso de operadores de agregación de datos para su procesamiento. En la segunda parte veremos como configurar varias instancias de MongoDB para gestionar la replicación y el sharding.

4.1 Introducción a MongoDB

En esta primera parte veremos como instalar MongoDB (aunque las máquinas virtuales de la materia ya lo tienen instalado) y probaremos la funcionalidad de gestión de datos y de procesamiento para cálculo de agregados.

4.1.1 Instalación y ejecución de MongoDB Community Edition en Ubuntu

Para más detalles de los proporcionados aquí ver esta página del manual. Utilizaremos el paquete mongodb-org mantenido por MongoDB Inc., y no el paquete mongodb que proporciona la distribución del sistema operativo.del manual. Utilizaremos el paquete mantenido por MongoDB Inc., y no el paquetemongodb mantenido por MongoDB Inc., y no el paquetemongodb

Importar la clave pública utilizada por el gestor de paquetes. Se necesita tener instalado `gnupg`.

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
```

Crear el archivo .list para la versión de ubuntu 20.04 Crear el archivo .listpara la versión de 20.04 20.04

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-4.4.list
```

Actualizamos la base de datos de paquetes.

```
sudo apt-get update
```

Instalamos MongoDB.

```
sudo apt-get install -y mongodb-org
```

Una vez hemos terminado la instalación tenemos:

- Un usuario mongodb que se puede usar para ejecutar
- Una carpeta `/var/lib/mongodb` por defecto para almacenar los datospor defecto para almacenar los datos
- Una carpeta `/var/log/mongodb` por defecto para almacenar los logspor defecto para almacenar los logs
- Un archivo de configuración `/etc/mongod.conf`

Los siguientes comandos permiten gestionar el servicio `mongod`, utilizando la configuración del archivo `/etc/mongod.conf`

Iniciar el servicio:	`sudo systemctl start mongod`
Habilitarlo para inicio automático:	`sudo systemctl enable mongod`
Ver información de su estado:	`sudo systemctl status mongod`
Pararlo:	`sudo systemctl stop mongod`
Reiniciarlo:	`sudo systemctl restart mongod`

Verificar si está habilitado para inicio automático:	`sudo systemctl is-enabled mongod`
Verificar si está activo:	`sudo systemctl is-active mongod`

La shell de mongo se inicia con el siguiente comando.

```
mongo
```

Se puede especificar la IP o nombre de la máquina y el puerto de distintas formas

```
mongo "mongodb://mongodb0.example.com:28015"
```

```
mongo --host mongodb0.example.com:28015
```

```
mongo --host mongodb0.example.com --port 28015
```

La conexión con un Replica Set o con un cluster via `mongos` lo veremos más adelante cuando veamos la replicación y el sharding.lo veremos más adelante cuando veamos la replicación y el sharding.

Los siguientes comandos básicos nos permiten ver la base de datos actual, cambiar la base de datos actual, ver las bases de datos del servidor o borrar la base de datos actual.

Ver la base de datos actual:	`db`
Usar otra base de datos:	`use <base de datos>`
Listar las bases de datos del servidor:	`show dbs, show databases`

Listar las colecciones de una BD:	`show collections`
	`db.getCollectionNames()`
	`db.getCollectionInfos()`

Borrado de la base de datos actual:	`db.dropDatabase()`

Las bases de datos en general se crean al insertar datos en ellas por primera vez. Se puede cambiar a una base de datos no existente y al crear los datos se creará la base de datos. La shell permite insertar comandos de varias líneas, usando {, (, [, hasta que todos estén cerrados no se termina el comando. Se pude usar el tabulador para completar expresiones, y las flechas del teclado para acceder al historial.

4.1.2 Operaciones de gestión de datos (Inserción, Lectura, Modificación y Borrado)

En esta parte veremos algunos ejemplos de operaciones usando la shell. Más detalles en la página del manual.

4.1.2.1 Inserción de documentos

Las inserciones se hacen en una única colección. Las operaciones son atómicas a nivel de documento. MongoDB añade un campo `_id` en cada documento, si este no ha sido proporcionado por el usuario. Se puede especificar un compromiso de escritura concreto, pero esto lo veremos en una sección posterior.

```
use bdge

db.empleados.insertOne(
 {nombre: "Miguel Angel Pintor",
  direccion: {calle: "Via del Corso",
              numero: "14",
              cp: "23452",
              localidad: "Roma"},
  paga_mensual: [23000, 23500, 23800, 23800, 23500, 22900, 23200, 46000, 23100, 23700, 23500, 46100],
  cursos: [
        {nombre: "Administración MongoDB", nota:8.9},
        {nombre: "Programación JavaScript", nota:7.2},
        ]
 }
)
```

Esta función inserta un único documento, y el parámetro que se le pasa es un objeto JSON. Se crea automáticamente la colección empleados, y se inserta su primer documento. Para ver este documento podemos hacer una consulta.

```
db.empleados.find().pretty()
```

Al añadir la función "`pretty()`" el texto del resultado se muestra con el sangrado correcto para su mejor visualización. Puede observarse como se añade un nuevo campo _id con un identificador de objetos nuevo.

Si necesitamos insertar varios elementos podemos usar la función "`db.coleccion.insertMany()`", a la que se pasa como parámetro un array de objetos JSON.

```
db.empleados.insertMany([
{nombre: "Leonardo Ingenioso",
 direccion: {calle: "Leonardo Ingenioso",
             numero: "5",
                   cp: "24987",
                   localidad: "Roma"},
 paga_mensual: [12,24,18,12,143,56,34,34,64,456,34,756,34,14,23],
 cursos: [
       {nombre: "Administración MongoDB", nota:4.7}
       ]
},
{nombre: "Donato di Niccolo di Betto Bardi",
 direccion: {calle: "Via dei Coronari",
             numero: "28",
                   cp: "23452",
                   localidad: "Roma"},
 paga_mensual: [23000, 23500, 23800, 23800, 23500, 22900, 23200, 46000, 23100, 23700, 23500, 46100],
 cursos: [
       {nombre: "Administración MongoDB", nota:5.3},
       {nombre: "Administración PostgreSQL", nota:9.3},
       {nombre: "Administración Neo4J", nota:5.6}
       ]
},
{nombre: "Rafael Sanzio",
 direccion: {calle: "Via Margutta",
             numero: "34",
                   cp: "23456",
                   localidad: "Roma"},
 paga_mensual: [12,12,42,34,23,123,24,23,124,434,45,345],
 proyectos: [
       {titulo: "Procesamiento a gran escala de datos", presupuesto: 2343465.23},
       {titulo: "Construcción de soluciones eficientes de publicación", presupuesto: 323244.43}
       ]
 }
])
```

Recordamos que, en este caso, al atomicidad se mantiene a nivel de documento, por lo tanto, una operación de lectura podría leer el primer documento insertado antes de que el segundo se haya insertado.

Existen otras operaciones que permiten la inserción de documentos en MongoDB, así como métodos específicos para la importación y exportación, como veremos más abajo.

4.1.2.2 Modificación de documentos

Para modificar un único documento se utiliza la función `db.collection.updateOne()`. Con el siguiente código modificamos el código postal de la dirección del empleado "Leonardo Ingenioso".

```
db.empleados.find({nombre:"Leonardo Ingenioso"}).pretty()

db.empleados.updateOne(
        {nombre: "Leonardo Ingenioso"},
        {
                $set: {"direccion.cp": 24900}
        }
)

db.empleados.find({nombre:"Leonardo Ingenioso"}).pretty()
```

Este operador $set solo permite asignar un valor a un campo, no permite evaluar expresiones para conseguir ese valor, para esto necesitamos usar la etapa $set de un pipeline de agregación, tal y como se muestra abajo. Además del operador $set se pueden usar otros operadores para realizar la modificación. Por ejemplo, con el operador $push podemos añadir uno o varios elementos a un array.

```
db.empleados.find({nombre:"Leonardo Ingenioso"}).pretty()

db.empleados.updateOne(
        {nombre: "Leonardo Ingenioso"},
        {
                $push: {cursos: {nombre: "Administración HBase", nota: 4.2}}
        }
)

db.empleados.find({nombre:"Leonardo Ingenioso"}).pretty()
```

Subir un punto a todos los cursos del empleado "Miguel Angle Pintor" que contengan la palabra "Administración". En este caso necesitamos filtrar dentro de un array, y para ello tenemos que utilizar la opción arryFilters.

```
db.empleados.find({nombre:"Miguel Angel Pintor"}).pretty()

db.empleados.updateOne(
        {nombre: "Miguel Angel Pintor"},
        {$inc: {"cursos.$[curso].nota": 1 }},
        {arrayFilters: [{"curso.nombre": {$in: [/Administración/]}}]}
)

db.empleados.find({nombre:"Miguel Angel Pintor"}).pretty()
```

Además de la función anterior existen otras funciones para modificar el contenido de los documentos. Por ejemplo, `db.collection.updateMany()` permite modificar todos los documentos que cumplen la condición, y `db.collection.replaceOne()` reemplaza el primer documento que cumple la condición.

Se pueden utilizar pipelines de agregación para realizar modificaciones. Un pipeline está formado por etapas, donde cada etapa realiza una operación en los documentos antes de pasárselos a la etapa siguiente. Las etapas que pueden usarse en operaciones de modificación son seis. Veremos como se procesan datos con estos pipelines en una sección posterior. Ahora solo algunos ejemplos de como se utiliza esto para hacer modificaciones.

En el siguiente ejemplo, para cada empleado, se calcula su paga mensual media, su paga anual, y un nuevo campo "tipo_empleado" que clasifica a los empleados según su paga_mensual_media.

```
db.empleados.updateMany(
{},
[{
        $set: {
            paga_mensual_media: {
                $trunc: [{
                        $avg: "$paga_mensual"
                    }, 0]
            },
            paga_anual: {
                $trunc: [{
                        $sum: "$paga_mensual"
                    }, 0]
            }
        }
    }, {
        $set: {
            tipo_empleado: {
                $trunc: [{
                        $divide: ["$paga_mensual_media", 10000]
                    }, 0]
            }
        }
    }
]
)
```

4.1.2.3 Borrado de documentos

Para borrar varios documentos se utiliza `db.coleccion.deleteMany()`, y para borrar un único documento de los que cumplen la condición usamos `db.coleccion.deleteOne()`.

Vamos a borrar el empleado con nombre "Rafael Sanzio".

```
db.empleados.deleteOne({nombre: "Rafael Sanzio"})
```

Borramos ahora a todos los empleados que viven en el código postal "23452"

```
db.empleados.deleteMany({"direccion.cp": "23452"})
```

Para borrar toda la colección usamos el método drop().

```
db.empleados.drop()
```

4.1.2.4 Consulta de documentos

Para poder realizar consultas, primero vamos a insertar una buena colección de documentos, que vamos a extraer de PostgreSQL y a importar en MongoDB. Para generar el archivo json con todas las películas ejecutamos el siguiente código en el esquema "agregados" de la base de datos "bdge".

```
copy
(select json_build_object(
        'id', id,
        'titulo', titulo,
        'presupuesto', presupuesto,
        'generos', generos,
        'idioma', idioma_original,
        'titulo_original', titulo_original,
        'sinopsis', sinopsis,
        'fecha_emision', fecha_emision,
        'ingresos', ingresos,
        'duracion', duracion,
        'reparto', reparto,
        'personal', personal
        )
from peliculasjson)
to '/home/alumnogreibd/public/peliculas.json'
with csv quote E'\r'
```

Ahora ya podemos importar las películas en MongoDB usando el comando "mongoimport".

```
mongoimport --db=bdge --collection peliculas --file=/home/alumnogreibd/public/peliculas.json
```

Obtenemos los datos de la película "John Carter".

```
db.peliculas.find({titulo: "John Carter"}).pretty()
```

Contamos el número de películas que tienen un presupuesto mayor que 275 millones.

```
db.peliculas.find({presupuesto:{$gt:275000000}}).count()
```

La lista de operadores que pueden usarse en las consultas se describe en el manual. Si se colocan varias condiciones, se combinarán usando el operador lógico AND.

```
db.peliculas.find({presupuesto:{$gt:275000000},
                   ingresos:{$lt:1000000000}}).pretty()
```

Si necesitamos realizar un cálculo durante la consulta, o comparar dos campos del documento, entonces necesitamos utilizar el operador $expr. Con la siguiente consulta obtenemos el número de películas posteriores a una determinada fecha (necesitamos obtener la fecha del string que tenemos almacenado).

```
db.peliculas.find({
    $expr: {
        $gt: [{
                $toDate: "$fecha_emision"
            }, ISODate("2012-01-01")]
    }
}).count()
```

Se pueden referenciar directamente campos de documentos anidados usando la notación "`campo1.campo2`", y también se puede consultar dentro de arrays y arrays de documentos anidados. Con esta consulta contamos las películas que tienen un género con nombre "Western".

```
db.peliculas.find({"generos.nombre": "Western"}).count()
```

Obtener las películas en las que "Johnyh Depp" ha interpretado el personaje "Tonto".

```
db.peliculas.find({"reparto.persona.nombre": "Johnny Depp", "reparto.personaje": "Tonto"}).pretty()
```

Para obtener los documentos en los que algún elemento del array cumple un conjunto de condiciones, sin tener en cuenta el orden, también se puede utilizar el operador "`$elemMatch`"

```
db.peliculas.find({
    reparto: {
        $elemMatch: {
            personaje: "Tonto",
            "persona.nombre": "Johnny Depp"
        }
    }
}).pretty()
```

Se puede proyectar el resultado para obtener solo algunos de los campos. En la siguiente consulta nos quedamos solo con los campos título y presupuesto de las películas que tienen un presupuesto mayor de 275 millones.

```
db.peliculas.find({
    presupuesto: {
        $gt: 275000000
    }
}, {
    titulo: 1,
    presupuesto: 1
}).pretty()
```

El número 1 indica que se incluye el campo. También se pueden excluir campos, usando el número 0. No se puede mezclar la inclusión y la exclusión de campos, salvo para excluir el campo `_id` en una proyección de inclusión.

```
db.peliculas.find({
    presupuesto: {
        $gt: 275000000
    }
}, {
    titulo: 1,
    presupuesto: 1,
    _id:0
}).pretty()
```

```
db.peliculas.find({
    presupuesto: {
        $gt: 275000000
    }
}, {
    reparto: 0,
    generos: 0,
    personal: 0,
        _id:0
}).pretty()
```

Se puede proyectar también dentro de documentos anidados en arrays.

```
db.peliculas.find({
    presupuesto: {
        $gt: 275000000
    }
}, {
    titulo: 1,
    presupuesto: 1,
    reparto: {
        persona: {nombre: 1}
    }
}).pretty()
```

4.1.3 Operaciones de agregación

Las operaciones de agregación en MongoDB procesan los documentos y obtiene colecciones de resultado. Existen tres formas de realizar agregaciones en MongoDB, los pipelines de agregación, las funciones map-reduce y los métodos de agregación de propósito específico, que incluyen la cuenta de documentos `db.collection.estimatedDocumentCount()` y `db.collection.count()` y la obtención de valores distintos de un campo `db.collection.distinct()`.

Ya tenemos las películas cargadas en la base de datos. Ahora podemos usar los agregados de propósito específico para obtener algunos datos preliminares.

```
db.peliculas.estimatedDocumentCount()
db.peliculas.count()
db.peliculas.distinct("generos.nombre")
```

4.1.3.1 Pipelines de agregación

Un pipeline de agregación consta de una secuencia de etapas, cada una de ellas aplica una operación al resultado de la anterior. Las posibles etapas (operaciones) se describen en la parte de referencia del manual. También en la referencia se describen todos los operadores que se pueden usar dentro de las etapas.

Vamos a ver primero un ejemplo en el que se utilizan 4 etapas: la etapa `$match` para filtrar y después la etapa `$group` para agrupar y agregar, después la etapa `$sort` para ordenar y finalmente la etapa `$limit` para quedarse con los 10 primeros documentos.

```
db.peliculas.aggregate([{
            $match: {
                presupuesto: {
                    $gt: 1000000
                }
            }
        }, {
            $group: {
                _id: "$idioma.id",
                peliculas: {
                    $sum: 1
                },
                ingresos_medios: {
                    $avg: "$ingresos"
                }
            }
        },
                {$sort: {peliculas:-1}},
                {$limit: 10}
    ])
```

Con la siguiente agregación procesamos los documentos para obtener la 10 directores que más ingresos han generado con sus películas.

```
db.peliculas.aggregate([{
            $unwind: "$personal"
        }, {
            $replaceRoot: {
                newRoot: {
                    $mergeObjects: [{
                            ingresos: "$ingresos"
                        },
                        "$personal"
                    ]
                }
            }
        }, {
            $match: {
                trabajo: "Director"
            }
        }, {
            $group: {
                _id: "$persona.nombre",
                ingresos_totales: {
                    $sum: "$ingresos"
                }
            }
        }, {
            $sort: {
                ingresos_totales: -1
            }
        }, {
            $limit: 10
        }
    ])
```

Primero la etapa `$unwind` desanida el array de personal de cada película. Cada documento del resultado tiene los datos de la película y los datos de un miembro del personal, en un documento anidado. La segunda etapa `$replaceRoot`, define como nuevo root de cada documento el documento anidado con los datos del miembro del personal, al que se le une, con el operador `$mergeObjects` los ingresos de la película. En el resultado, ahora tenemos un documento para cada miembro del personal de cada película, con un campo que almacena los ingresos de la película. La siguiente etapa `$match`, filtra los documentos para quedarse solo con los que tienen en el campo "trabajo" el valor "Director". La etapa `$group`, ya la hemos visto arriba, agrupa por el nombre de cada director y suma los ingresos. Finalmente, `$sort` y `$limit` se quedan con los 10 ingresos más altos.

4.1.3.2 Agregación con funciones map-reduce

Se recomienda el uso de pipelines de agregación en lugar de map-reduce por su mejor rendimiento, en general. Solo vamos a ver un ejemplo muy simple ya que no asumiremos conocimientos previos de JavaScript.

```
db.peliculas.mapReduce(
        function(){
                for (let i=0;i<this.generos.length;i++){
                        emit(this.generos[i].nombre, 1);
                }
        },
        function(key, values) {return Array.sum(values)},
        {
         query: {generos:{$ne:null}, "idioma.id":"es"},
         out:"generos"
        }
)

db.generos.find().pretty()
```

La primera función JavaScript (map) genera pares (clave,valor). La segunda función (reduce) agrega los valores para cada clave. Con la opción query se filtran los documentos antes de pasárselos a las funciones. La opción out proporciona el nombre de la colección de salida.

Como puede verse, con map-reduce, se puede utilizar cualquier función JavaScript para realizar la agregación.

4.2 Ejemplos de replicación y sharding con MongoDB

En esta parte de la práctica vamos a ver como montar un cluster de varias máquinas con MongoDB, para dar soporte a la replicación y sharding de las colecciones de documentos. Para llevar a cabo esta práctica vamos a clonar la máquina "GreiBDSeverBase" en la que también tenemos ya una instalación de MongoDB. Las instrucciones para clonar la máquina ya las hemos visto en el tutorial de citus.

Recordamos los pasos:

1.- Clonar la máquina en virtual box, con cuidado de generar nuevas MAC para las tarjetas.

2.- Cambiar el nombre y regenerar los identificadores (recordar anotar la IP para poder hacer después una conexión SSH)

```
sudo hostnamectl set-hostname citus2
sudo hostnamectl

sudo rm /etc/machine-id
sudo dbus-uuidgen --ensure=/etc/machine-id
dbus-uuidgen --ensure

sudo shutdown -r now
```

Para simplificar el resto de la práctica iniciamos una sesión SSH con un terminal en el que podamos copiar y pegar. En Windows podemos instalar PuTTY. Se puede descargar un instalador o la versión standalone para ejecutar directamente.

4.2.1 Ejemplo de creación de un cluster con replicación en MongoDB

En el tutorial del manual de MongoDB hay secciones independientes para desplegar un cluster de pruebas, como el que vamos a ver aquí, y un cluster de producción.

Aquí nos centraremos en un cluster de pruebas o desarrollo, para facilitar este ejemplo, y vamos a utilizar solo una máquina en la que levantaremos tres instancias distintas (procesos) de MongoDB, que simularán tres nodos del cluster.

Como MongoDB ya está instalado, lo único que tenemos que hacer es asegurarnos de que no está iniciado como servicio, usando el comando `sudo systemctl status mongod`.

Creamos una carpeta distinta en el home para los los datos de cada uno de los tres nodos.

```
mkdir mongo1
mkdir mongo2
mkdir mongo3
```

Iniciamos ahora las tres réplicas, cada una en un puerto distinto, y referenciando la carpeta de datos que acabamos de crear. Para todas ellas, utilizamos el mismo identificador de `replicaSet`, en este caso `rs0`. Para que los procesos se mantengan iniciados, incluso si se cierra la terminal, usaremos el comando `hohup` para lanzarlos. La salida la redirecionamos a `/dev/null` para que no nos moleste en la terminal. Terminamos el comando con el símbolo `&` para que quede funcionando en background.

```
nohup mongod --replSet rs0 --port 27017 --bind_ip localhost --dbpath /home/alumnogreibd/mongo1 > /dev/null &
```

```
nohup mongod --replSet rs0 --port 27018 --bind_ip localhost --dbpath /home/alumnogreibd/mongo2 > /dev/null &
```

```
nohup mongod --replSet rs0 --port 27019 --bind_ip localhost --dbpath /home/alumnogreibd/mongo3 > /dev/null &
```

Para ver los procesos usamos el comando `ps -x`. Si necesitamos terminar alguno, usamos su identificador pid en el comando `kill -9 pid`. La opción --bind_ip define la ip que se usará para lanzar peticiones al nodo. Debe de ser la ip del propio nodo por lo tanto. Si las tres instancias estuvieran en nodos distintos, en lugar de usar localhost, deberíamos usar "localhost,192.168.56.xxx", para admitir conexiones tanto desde localhost si estamos en local y queremos hacerlo, como desde cualquier otro nodo usando la ip de este.

Realizamos ahora la conexión con la shell.

```
mongo --port 27017
```

Ahora desde la shell iniciamos el replicaSet.

```
rs.initiate(
        {
          _id: "rs0",
          members: [
                {
                 _id: 0,
                 host: "localhost:27017"
                },
                {
                 _id: 1,
                 host: "localhost:27018"
                },
                {
                 _id: 2,
                 host: "localhost:27019"
                }
           ]
        }
)
```

Si las instancias estuvieran en máquinas distintas, tendríamos aquí que usar la IP de cada máquina. Una vez que hemos iniciado el replicaSet desde uno de los nodos, ya podemos abandonar la shell de ese nodo para poder realizar la conexión con el replicaSet.

```
mongo "mongodb://localhost:27017,localhost:27018,localhost:27019/?replicaSet=rs0"
```

Si las instancias de las tres réplicas estuvieran en máquinas distintas, aquí tendríamos que poner las ips de las tres máquinas. Vemos la configuración del replicaSet

```
rs.conf()
```

Ahora vamos a crear una colección de pruebas en una base de datos nueva.

```
use xine

db.coleccionPrueba.insertOne({clave:1, valor:5})

for (var i=2; i<1000; i++) {db.coleccionPrueba.insertOne({"clave":i,"valor":i+5})}

db.coleccionPrueba.find().count()
```

Ahora eliminamos uno de los nodos y probamos si todavía podemos acceder a todos los datos. Para eliminar un nodo, vemos su id con `ps -x` y lo matamos con `kill -9 pid`. Las consultas siguen funcionando.

```
db.coleccionPrueba.find().count()
```

Si eliminamos dos podemos comprobar que ya no podemos leer los datos. Podremos leer los datos si cambiamos la preferencia de lectura para leer, con preferencia del primario, pero si no puede permitimos leer de cualquier otro.

```
db.getMongo().setReadPref("primaryPreferred")
```

Ahora ya funciona la lectura de nuevo.

```
db.coleccionPrueba.find().count()
```

Si levantamos cualquiera de los demás nodos, ya podemos leer de nuevo con la opción por defecto "primary" para la preferencia de lectura.

```
db.getMongo().setReadPref("primary")
```

Una vez terminado el ejemplo, podemos salir de la shell de mongo con `exit`. Terminamos todos los procesos lanzados. Al terminar borramos las tres carpetas de datos, `mongo1`, `mongo2`, `mongo3`.

4.2.2 Ejemplo de creación de un cluster con sharding en MongoDB

En el manual tenemos un tutorial para desplegar un cluster con sharding, tanto usando directamente la línea de comando para iniciar los nodos, como usando archivos de configuración. Para simplificar, aquí vamos a usar una sola máquina y no vamos a usar replicación en los componentes.

Para desplegar nuestro cluster, y dado que no vamos a replicar ningún componente, necesitaremos iniciar los siguientes procesos:

- Un proceso mongod para cada uno de los dos shards que vamos a desplegar.
- Un proceso mongod para el servidor de configuración del cluster
- Un proceso mongos para el broker que da acceso al cluster

Recordar que cada uno de los procesos de arriba, en un cluster de producción, debería de estar replicado como mínimo 3 veces, es decir, necesitaríamos 12 procesos.

En este ejemplo, vamos a iniciar los cuatro procesos del cluter mínimo en una sola máquina, usando de nuevo terminales ssh distintos.

4.2.2.1 Despliegue del servidor de configuración (Config Server)

Iniciamos el servidor de configuración en una terminal, pero antes necesitamos crear una carpeta para sus datos. Ya no vamos a explicar como sería en caso de usar distintas máquinas, y los cambios que habría que hacer en las opciones --bind_ip.

```
mkdir servconf

nohup mongod --configsvr --replSet rsConfServer --port 27017 --dbpath /home/alumnogreibd/servconf --bind_ip localhost > /dev/null &
```

Una vez iniciado el proceso, necesitamos otra terminal para iniciar el replicaSet correspondiente. Primero entramos en la shell.

```
mongo --host localhost --port 27017
```

Dentro iniciamos el replicaSet con la función `rs.inititate`. Tal y como hemos hecho ya en el ejemplo de replicación, solo que ahora el replicaSet tiene solo un miembro.

```
rs.initiate({
  _id: "rsConfServer",
  configsvr: true,
  members: [{_id: 0, host: "localhost:27017"}]
})
```

Dado que el replicaSet tiene solo un miembro que coincide con el nodo con el que hemos conectado la shell, podríamos haber usado el comando `rs.inititate()` sin parámetros. Una vez iniciado, podemos ver la configuración del mismo con `rs.conf()`.

4.2.2.2 Creación de los replicaSet de los dos shards

```
mkdir shard1
mkdir shard2

nohup mongod --shardsvr --replSet rsShard1 --port 27018 --dbpath /home/alumnogreibd/shard1 --bind_ip localhost > /dev/null &
```

De nuevo, necesitamos otra terminal para iniciar el replicaSet usando la función `rs.initiate()`. Podemos ver la configuración con `rs.conf()`.

```
mongo --host localhost --port 27018

rs.initiate()

rs.conf()
```

Hacemos lo mismo para el segundo shard

```
nohup mongod --shardsvr --replSet rsShard2 --port 27019 --dbpath /home/alumnogreibd/shard2 --bind_ip localhost > /dev/null &
```

Y de nuevo otro terminal para iniciar el replicaSet.

```
mongo --host localhost --port 27019

rs.initiate()

rs.conf()
```

4.2.2.3 Depliegue del broker de consultas mongos y configuración del cluster

Primero iniciamos el proceso mongos para que actúe como punto de entrada al cluster.

```
nohup mongos --port 27020 --configdb rsConfServer/localhost:27017 --bind_ip localhost > /dev/null &
```

De usar varias instancias para cada replicaSet, la opción `--configdb` tendría la forma "`rsConfServer/ip1:port1,ip2:port2,ip3:port3`". Ahora ya podemos abrir la última terminal para realizar una conexión con el cluster y terminar su configuración.

```
mongo --host localhost --port 27020
```

Para añadir los shards al cluster se usa la función `rs.addShard`.

```
sh.addShard("rsShard1/localhost:27018")
sh.addShard("rsShard2/localhost:27019")
```

De usar varias instancias en cada replicaSet, el formato de cada añadir cada shard sería: "`rsShardid/ip1:port1,ip2:port2,ip3:port3`". Habilitamos el sharding en una base de datos de prueba.

```
use prueba
sh.enableSharding("prueba")
```

Antes de crear una colección de prueba y de particionarla, vamos a cambiar la configuración del cluster para reducir el tamaño del chunk, así no necesitaremos introducir muchos datos para que se repartan entre los dos shards. Para ver el tamaño actual del chunk podemos ver la colección "`settings`".

```
use config
db.settings.find().pretty()
```

Por defecto, no hay un valor especificado para el tamaño de chunk, y el sistema usa el valor de 64MB. Para cambiar el tamaño de chunck a 1MB utilizamos la siguiente llamada, tal y como se detalla en el manual.

```
db.settings.save({ _id:"chunksize", value: 1 })
```

Ahora podemos crear una colección nueva, indexarla (utilizaremos hashing) y particionarla. Primero creamos la colección y la indexamos.

```
use prueba

db.coleccionPrueba.insertOne({clave:1, valor:5})
db.coleccionPrueba.createIndex({clave:"hashed"})
```

Ahora la particionamos

```
sh.shardCollection("prueba.coleccionPrueba", {clave:"hashed"})
```

De momento, todos los datos estarán en un único shard, ya que todos camben en un único chunk de 1MB

```
sh.status()
```

Añadimos ahora suficientes datos para que se generen más chuncks. Puede tardar unos minutos en insertar todos los documentos.

```
for (var i=2; i<100000; i++) {db.coleccionPrueba.insertOne({"clave":i, "valor":i+5})}
```

Comprobamos finalmente como los chunks se han repartido entre los shards

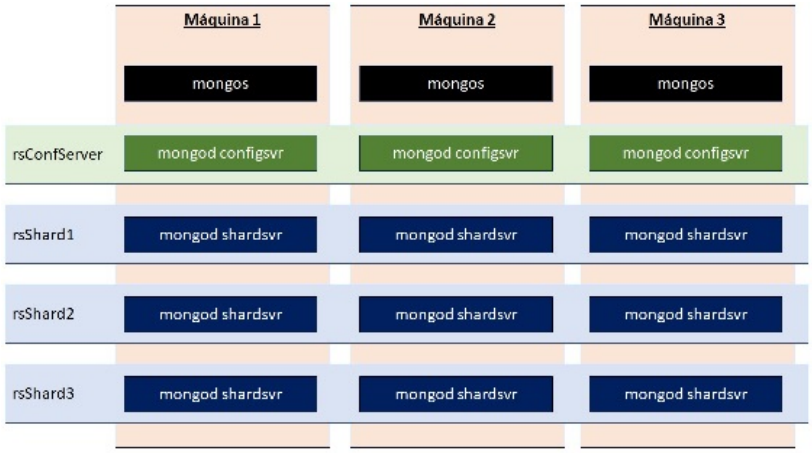
```
sh.status()
```

Debería de haber unos 6 chunks, repartidos entre los dos shards.

Dado que no hemos usado replicación, podemos comprobar que con para cualquiera de los tres componentes, el servidor de configuración o cualquiera de los dos shards, ya no podremos acceder a la colección de prueba que acabamos de crear.

Ejercicios

Crea un cluster de MongoDB con la siguiente arquitectura, carga datos de la base de datos de películas en el y realiza algunas consultas y agregados. Genera un documento (tipo blog), en donde describas los pasos que vas realizando, proporcionas los comandos y código que vas ejecutando y explicas los resultados que vas obteniendo. Entrega el documento a través del campus virtual. Utiliza preferiblemente un formato de texto que facilite la visualización del código.



E1.- Crea las tres máquinas clonando la máquina MongoBase proporcionada. Recuerda que MongoDB ya está instalado en esa máquina, pero es necesario generar nuevas direcciones MAC para los adaptadores de red, cambiar el nombre del host en el sistema operativo y cambiar el identificador de la máquina en el sistema operativo.

E2.- Inicia el replicaSet "rsConfServer" para utilizar como Servidor de Configuración del cluster. Este replicaSet debe tener tres miembros, uno en cada máquina. Recuerda que ahora será necesario indicar la IP de la máquina correspondiente en la opción --bind_ip cuando se lanza cada proceso. Recuerda que además de lanzar el proceso es necesario configurar el replicaSet usando la función rs.initiate(), que en este caso no podrá usar los parámetros por defecto, al tener el replicaSet más de un miembro.

E3.- Inicia un replicaSet para cada uno de los tres shards (rsShard1, rsShard2, rsShard3). Cada uno de los tres replicaSets tendrá tres miembros, uno en cada una de las tres máquinas, tal y como se indica en la arquitectura. Usa rs.inititate() de forma apropiada para configurar cada uno de los tres replicaSet.

E4.- Inicia una instancia de mongos en cada una de las tres máquinas. Conecta después con una de las instancias y añade los tres shards al cluster usando la función rs.addShard. Recuerda que ahora, al tener tres miembros cada replicaSet de cada shard, la URL de cada shard debe tener las IPs y puertos de todos los miembros.

E5.- Carga los datos completos de la base de datos de películas, usando el mismo archivo json generado en la práctica. Para poder hacer esto debes de mover el archivo json generado en la máquina "GreiBD" a esta nueva máquina. Puedes usar una conexión SFTP.

E6.- Indexa y particiona la colección utilizando hashing sobre el atributo id.

E7.- Realiza una consulta en la que cuentes el número de películas que tienen un presupuesto mayor que medio millón de dólares.

E8.- Apaga una de las tres máquinas, y comprueba que todavía puedes resolver la consulta anterior.

E9.- Apaga otra de las tres máquinas, y comprueba que ya no puedes resolver la consulta anterior.

E10.- Vuelve a iniciar las dos máquinas apagadas y vuelve a lanzar los procesos mongod y mongos necesarios en cada una de ellas: un proceso mongos, un proceso mongod para el configsvr y tres procesos mongod, uno para cada uno los tres shardsvr. Comprueba ahora que la consulta anterior ya funciona de nuevo.

E11.- Modifica la columna "fecha_emision" para que en lugar de ser de tipo string sea de tipo Date. Utiliza para ello el operador "$dateFromString" dentro de una llamada "updateMany".

E12.- Realiza una consulta en la que obtengas todos los títulos y presupuestos de las películas del género "Science Fiction", que tengan unos ingresos de más de 800000000 dolares.

E13.- Realiza una consulta que obtenga el título y el título original de las películas protagonizadas por "Penélope Cruz" del género de acción (Action). Ordena el resultado por fecha de emisión.

E14.- Realiza una agregación en la que obtengas para cada uno de los géneros disponibles en la base de datos, el número de películas que tiene, el total del presupuesto invertido, el total de ingresos generados y la diferencia entre los ingresos y el presupuesto. Ordena el resultado por esta diferencia.

E15- Realiza una agregación para obtener los 10 directores que tiene la duración media de sus películas mas alta, de entre sus películas con ingresos (ingresos>0). Para cada director, saca el número total de películas, la duración media de las películas y la suma de los ingresos de las mismas.

Última modificación: Luns, 13 de Setembro de 2021, 16:36

◄ Práctica 4: Bases de datos NoSQL: Documentales (Presentación MongoDB)

Ir a

Práctica 5: Base de datos NoSQL: Column Family (Presentación HBase) ►