02 Deploy a sharding

El primer paso para crear un clúster es iniciar todos los procesos necesarios, por tanto debe configurar los mongos y los fragmentos. También hay un tercer componente, los config servers, que son una pieza importante.

Los server configs son servidores mongod normales que almacenan la configuración del clúster: qué conjuntos de réplica alojan los fragmentos, qué colecciones se fragmentan y en qué fragmento se encuentra cada fragmento.

MongoDB 3.2 introdujo el uso de Replica Set como Config Server, reemplazando el mecanismo de sincronización original utilizado por los servidores de configuración. La capacidad de utilizar ese mecanismo se eliminó en MongoDB 3.4.

Config Servers

Los Config Servers son los cerebros de su clúster: contienen todos los metadatos sobre qué servidores contienen qué datos. Por lo tanto, deben configurarse primero, y los datos que contienen son extremadamente importantes: asegúrese de que se estén ejecutando con el journal habilitado y que sus datos estén almacenados en unidades no efímeras.

En las implementaciones de producción, el Replica Set del Config Server debe constar de al menos tres miembros. Cada Config Server debe estar en una máquina física separada, preferiblemente distribuida geográficamente.

Los Config Server deben iniciarse antes de cualquiera de los procesos mongos (servidores sharding), ya que mongos extrae su configuración de ellos.

Para comenzar, ejecutemos en tres máquinas separadas para iniciar sus servidores de configuración:

Creamos data/configServer1|2|3

y:

mongod --configsvr --replSet configServerGetafe --dbpath data/configServer1 --port 27100

mongod --configsvr --replSet configServerGetafe --dbpath data/configServer2 --port 27101

mongod --configsvr --replSet configServerGetafe --dbpath data/configServer3 --port 27102

Ahora nos conectamos a uno de los nodos:

mongo --port 27100

Y configuramos:

```
> rs.initiate({
_id: "configServerGetafe",
configsvr: true,
members: [
        {_id: 0, host: "localhost:27100"},
        {_id: 1, host: "localhost:27101"},
        {_id: 2, host: "localhost:27102"}
        ]
})

{
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1580833582, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1580833582, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1580833582, 1)
}
```

La opción --configsvr le indica al mongod que planea usarlo como un servidor de configuración. En un servidor que se ejecuta con esta opción, los clientes (es decir, otros componentes del clúster) no pueden escribir datos en ninguna base de datos que no sea config o admin.

La base de datos de administración contiene las colecciones relacionadas con la autenticación y autorización, así como las otras colecciones del system.* para uso interno.

La base de datos de configuración contiene las colecciones que contienen los metadatos del clúster fragmentado. MongoDB escribe datos en la base de datos de configuración cuando los metadatos cambian, como después de una migración de fragmentos o una división de fragmentos.

Al escribir en los servidores de configuración, MongoDB utiliza un nivel "majority" de writeConcern. De manera similar, cuando lee desde servidores de configuración, MongoDB usa un nivel readConcern "majority".

Esto garantiza que los metadatos de clúster fragmentados no se comprometerán con el conjunto de réplica del servidor de configuración hasta que no se pueda revertir con un Rollback.

También asegura que solo se leerán los metadatos que sobrevivirán a una falla de los servidores de configuración. Esto es necesario para garantizar que todos los enrutadores mongos tengan una visión coherente de cómo se organizan los datos en un clúster fragmentado.

En términos de aprovisionamiento, los servidores de configuración deben aprovisionarse adecuadamente en términos de redes y recursos de CPU. Solo contienen una tabla de contenido de los datos en el clúster, por lo que los recursos de almacenamiento necesarios son mínimos. Deben implementarse en hardware separado para evitar la contención de los recursos de la máquina.

ADVERTENCIA. Si se pierden todos sus servidores de configuración, debe buscar en los datos de sus fragmentos para averiguar qué datos están dónde. Esto es posible, pero lento y desagradable. Realice copias de seguridad frecuentes de los datos del servidor de configuración. Siempre realice una copia de seguridad de sus servidores de configuración antes de realizar cualquier mantenimiento de clúster.

Shards

Aunque es recomendable que cada shard sea un replica set, podemos simplificar instancias standalone. Creamos de momento dos Shards. Por ejemplo:

Creamos sus directorios

data/shServer1

data/shServer1

Y levantamos:

mongod --shardsvr --dbpath data/shServer1 --port 27103
mongod --shardsvr --dbpath data/shServer2 --port 27104

Instancia mongos

Para levantar la instancia mongos que enrutará las operaciones no necesitamos directorio simplemente levantamos en la dirección y puerto (se puede utilizar --log ruta para guardar sus logs).

Por ejemplo:

mongos --configdb configServerGetafe/localhost:27100,localhost:27101,localhost:27102 --port 27099

Debe iniciar una pequeña cantidad de procesos mongos y ubicarlos lo más cerca posible de todos los shards. Esto mejora el rendimiento de las consultas que necesitan acceder a múltiples shard o que realizan operaciones scatter/gather.

La configuración mínima es de al menos dos procesos mongos para garantizar una alta disponibilidad. Es posible ejecutar decenas o cientos de procesos mongos, pero esto causa contención de recursos en los servidores de configuración. El enfoque recomendado es proporcionar un pequeño grupo de enrutadores.

Una vez que tenemos un mongos levantado podemos añadir los shard al cluster.

Conectamos una shell al mongos:

mongo --port 27099

Observamos como el prompt ahora es mongos.

Y usamos el método sh.addShard() para añadir los shard al cluster:

```
mongos>  sh.addShard("localhost:27103")
{
        "shardAdded" : "shard0000",
        "ok" : 1,
        "operationTime" : Timestamp(1580835650, 4),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1580835650, 4),
```

```
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.addShard("localhost:27104")
{
        "shardAdded" : "shard0001",
        "ok" : 1,
        "operationTime" : Timestamp(1580835660, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1580835660, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Podemos comprobar ahora con sh.status()

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5e399b3bc5d6aec941fe63bc")
  }
  shards:
      {  "_id" : "shard0000",  "host" : "localhost:27103",  "state" : 1 }
      {  "_id" : "shard0001",  "host" : "localhost:27104",  "state" : 1 }
  active mongoses:
      "4.2.3" : 1
  autosplit:
      Currently enabled: yes
  balancer:
      Currently enabled:  yes
      Currently running:  no
      Failed balancer rounds in last 5 attempts:  0
      Migration Results for the last 24 hours:
            No recent migrations
  databases:
      {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
```

Sharding Data

MongoDB no distribuirá sus datos automáticamente hasta que le indique cómo hacerlo. Debe indicar explícitamente tanto a la base de datos como a la colección que desea que se distribuyan.

Las siguientes operaciones se realizarán desde la mongos, no desde cada instancia particionada.

En primer lugar debemos indicar a la base de datos que permita el sharding de sus colecciones. Por ejemplo:

```
use config
db.settings.save( { _id:"chunksize", value: 16 } )
```

```
mongos> sh.enableSharding("shop")
{
        "ok" : 1,
        "operationTime" : Timestamp(1580836023, 6),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1580836023, 6),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Una vez que haya habilitado el fragmentación en el nivel de la base de datos, puede fragmentar una colección ejecutando sh.shardCollection ():

```
mongos> sh.shardCollection("shop.clientes", { edad : 1} )
{
        "collectionsharded" : "shop.clientes",
        "collectionUUID" : UUID("ab53fb3d-48cc-4c24-ba73-41e4222681df"),
        "ok" : 1,
        "operationTime" : Timestamp(1580838899, 9),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1580838899, 9),
                "signature" : {
```

```
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Ahora la colección de clientes estará fragmentada por el campo “edad”. Si está fragmentando una colección existente, debe haber un índice en el campo "edad"; de lo contrario, la llamada shardCollection devolverá un error. Si obtiene un error, cree el índice (mongos devolverá el índice que sugiere como parte del mensaje de error) y vuelva a intentar el comando shardCollection.

Si la colección que está fragmentando aún no existe, mongos creará automáticamente el índice de clave de fragmentación para usted.

El comando shardCollection divide la colección en chunks, que son las unidades que MongoDB usa para mover los datos. Una vez que el comando regrese con éxito, MongoDB comenzará a equilibrar la colección entre los fragmentos de su clúster. Este proceso no es instantáneo.

Para colecciones grandes, puede llevar horas terminar este equilibrio inicial. Este tiempo se puede reducir con la división previa, donde se crean fragmentos en los fragmentos antes de cargar los datos. Los datos cargados después de este punto se insertarán directamente en el fragmento actual sin requerir un equilibrio adicional.

Comprobamos ahora el sharding con una colección grande (importando un archivo) desde mongos.

```
mongos> load("clientes.js")
true
```

Para mongos el sharding de la colección es transparente

```
mongos> db.clientes.find()
{ "_id" : 0, "nombre" : "Fernando", "apellido1" : "López", "apellido2" : "González", "edad" : 76, "dni" : "22310244B" }
{ "_id" : 1, "nombre" : "María", "apellido1" : "Fernández", "apellido2" : "Fernández", "edad" : 16, "dni" : "25299855X" }
{ "_id" : 2, "nombre" : "Juan", "apellido1" : "López", "apellido2" : "López", "edad" : 51, "dni" : "95280457X" }
...
```

Podemos comprobar una vez que finalicen las migraciones con:

```
mongos> sh.status({verbose: true})
--- Sharding Status ---
  sharding version: {
          "_id" : 1,
          "minCompatibleVersion" : 5,
          "currentVersion" : 6,
          "clusterId" : ObjectId("5e399b3bc5d6aec941fe63bc")
  }
  shards:
      {  "_id" : "shard0000",  "host" : "localhost:27103",  "state" : 1 }
      {  "_id" : "shard0001",  "host" : "localhost:27104",  "state" : 1 }
  active mongoses:
      {  "_id" : "MacBook-Pro-de-Pedro.local:27099",  "advisoryHostFQDNs" : [
"macbook-pro-de-pedro.local" ],  "mongoVersion" : "4.2.3",  "ping" :
ISODate("2020-02-04T20:42:38.936Z"),  "up" : NumberLong(14674),  "waiting" : true
}
  autosplit:
      Currently enabled: yes
  balancer:
      Currently enabled:  yes
      Currently running:  no
      Failed balancer rounds in last 5 attempts:  0
      Migration Results for the last 24 hours:
          14 : Success
  databases:
      {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
          config.system.sessions
              shard key: { "_id" : 1 }
              unique: false
              balancing: true
              chunks:
                  shard0000       1
              { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : shard0000
Timestamp(1, 0)
      {  "_id" : "shop",  "primary" : "shard0001",  "partitioned" : true,  "version" : {  "uuid"
: UUID("b6cfa4d1-2028-4275-8b1f-bed9d9eb2344"),  "lastMod" : 1 } }
          shop.clientes
              shard key: { "edad" : 1 }
              unique: false
              balancing: true
```

```
                chunks:
                        shard0000       14
                        shard0001       15
                { "edad" : { "$minKey" : 1 } } -->> { "edad" : 0 } on : shard0000
Timestamp(3, 0)
                { "edad" : 0 } -->> { "edad" : 34 } on : shard0000 Timestamp(5, 0)
                { "edad" : 34 } -->> { "edad" : 36 } on : shard0000 Timestamp(10, 0)
```

...

También podemos obtener las estadísticas de la colección con:

```
mongos> use shop
switched to db shop
mongos> db.clientes.getShardDistribution()

Shard shard0000 at localhost:27103
 data : 703.64MiB docs : 6298749 chunks : 14
 estimated data per chunk : 50.25MiB
 estimated docs per chunk : 449910

Shard shard0001 at localhost:27104
 data : 1.09GiB docs : 10000000 chunks : 15
 estimated data per chunk : 74.47MiB
 estimated docs per chunk : 666666

Totals
 data : 1.77GiB docs : 16298749 chunks : 29
 Shard shard0000 contains 38.64% data, 38.64% docs in cluster, avg obj size on
shard : 117B
 Shard shard0001 contains 61.35% data, 61.35% docs in cluster, avg obj size on
shard : 117B
```