Instrucciones

Una empresa líder en el sector de sensores ambientales desea utilizar MongoDB para almacenar los datos capturados por sus sensores. Los datos incluyen atributos como la fecha y hora de lectura, el identificador del sensor, la ubicación del sensor, la temperatura ambiental, porcentaje de humedad, nivel de ruido en decibelios y la intensidad luminosa en candelas.

La empresa solicita una consultoría para definir el formato del documento a almacenar en la base de datos. Además, se solicita la configuración del sistema para que cumpla los siguientes requisitos:

- Las consultas por identificador y fecha del sensor deben estar optimizadas.

  Para ello se tendrá que crear índices sobre aquellos datos que la empresa crea más adecuado para su utilización. MongoDB tiene el siguiente comando para lo que busca la empresa:

  "> db.sensor.ensureIndex({_id: 1})" Crea un metadato que guarde la información ordenada por índice en la RAM y así se tiene una mayor rapidez a la hora de acceder mediante ese indicador en concreto. También lo pide sobre el índice fecha. Así que se usará el mismo comando, pero en vez de poner "_id: 1", se pondrá "fecha: 1". El uno se pone para ordenar en forma ascendente.

- La información a almacenar es sumamente importante para la empresa, por lo que se pide que tenga al menos dos copias de la base de datos, actualizadas inmediatamente después de cada cambio.

  Recomendaría antes de empezar a trabajar con las bases de datos guardar periódicamente la información de manera manual en un disco duro externo para prevenir un conflicto en caso de que todo falle sin remedio, se podría hacer mediante Batch. Cada x tiempo se puede hacer una copia con el comando "> mongodump -sensores -o "dirección_donde_se_quierea_guardar""

  Una vez cerciorada una copia se creará una estructura que nos permita tener los datos guardado continuamente en varios servidores, con los objetivos de distribuir la información, mantenerla a salvo y que esté actualizada con cada cambio. Para ello se van a utilizar los *replicaSet* que proporciona MongoDB para tareas de esta envergadura.

  Lo primero es desconectar el servidor principal de *mongod* en el caso de que estuviera corriendo la máquina. Se inicia con todas sus características, pero esta vez con el comando

“>mongod --port 27XXX --dbpath “ruta_del_data” --replSet “nombre_del_data””. Donde port es el puerto que se usa. Dbpath es la ruta de los datos guardados y por último se dice que es un *replicaSet*

Se inicia MongoDB desde la consola de comando para inicializar los replicaSet. Para ello se usa el comando rs.initiate()

En este momento se puede ver como se ha configurado con . rs.conf ()

También vemos el estado del servidor y que aún se encuentra en solo conectado, pero ya llamado.

Se crea los nuevos servidores para añadir replicaSet, como creamos el primero con el comando “mongod --port 27XXX” --dbpath “ruta_del_data” --replSet nombre_del_replicaSet””

Ahora hay que añadir los nuevos servidores creados al servidor primario. Esto hace que tengamos tres servidores, en este caso concreto, dos de ellos sirven para almacenar la información en caso de fallo. Si hubiera un fallo, uno de ellos saltaría bajo unos criterios de elección por voto de cada servidor para convertirse en primario, y así, poder seguir trabajando.

Para añadir los replicatSet solo hay que usar los comando rs.add(“host:port”).

Y si comprueba otra vez el estado vemos que todo queda ya configurado como se quería.

Lo idóneo sería guardar todo esto en un archivo de configuración para que cuando el servidor caiga, todo vuelva a instalarse como debería estar.

- Se requiere que solo una persona tenga acceso a los datos, las credenciales son definidas por el consultor.

  Lo primero es asegurar una red privada y segura. Configurar los accesos de control mediante *blind* y luego preparar el firewall con las reglas especifica que se necesite.

  Para ello se tiene que crear un usuario de administrador. Desde la consola de MongoDB se usa el comando: use admin. Luego creamos el usuario, el administrador, para ello usamos: db.createUser({ user: "siteUserAdmin", pwd: "password",roles: [{ role: "userAdminAnyDatabase", db:"admin"}]}). También le asignamos los privilegios en role.

  El cliente puede crear su propia seguridad en su base de datos. Para restringirlo en su base de datos hay que usar *sensores*. Como ejemplo: db.createUser({user: "recordsUserAdmin",pwd: "password",roles: [{ role: "userAdmin", db:"sensores" }]})

  En userAdmin puede poner

  read – solo lectura en la base de datos.

  readWrite – permite lectura y escritura.

  dbAdmin – para las tareas administrativa.

  Para especificar que se necesita autorización, necesitamos decirle a mongod antes de ejecutarlo que vamos a requerir autentificación, mediante la línea de comando --auth

Para poner la contraseña primero tendremos que entrar en la db de admin y luego especificar db.auth(usuario, contraseña). En el caso del cliente tendrá que entra en su base de dato: use sensores. Y lo mismo para autentificarse.

- Uno de los servidores donde habrá una instancia de MongoDB tiene una interfaz de red conectada a internet. Se solicita que esta interfaz no tenga acceso a la base de datos.

  Para ello solo hay que usar el comando --bind_ip con dirección exclusiva local, en nuestro caso 127.0.0.1, cuando iniciamos mongod.

- Se estima que en un futuro se manejará una gran cantidad de sensores alrededor del mundo, por lo que se requiere una arquitectura escalable.

  Para ello se tiene dos opciones o bien mediante el escalar vertical, que implicaría la adquisición de máquinas más potentes, con cpu Xeon de Intel o bien la famila Opteron de AMD. También hay que tener especial cuidado de elegir una RAM apropiada para la carga de trabajo. Como memoria RAM DDR4 de 3000 para un gran rendimiento.

  También hay que pensar en una estructura escalar horizontal, que depende más del Administrador de datos. En este caso lo que habría que hacer es aprovechar las ventajas de MongoDB para mejorar en este sentido. Para ello lo más conveniente sería empezar en una primera instancia creado tres servidores con MongoDB. Crearemos clúster de fragmentación para hacer balancear la carga de trabajo y así tener un reparto apropiado para todos los servidores.

  Lo primero que vamos a hacer es configurar los servidores, para ello dentro del directorio data de cada servidor crearemos una carpeta llamada configdb. A la hora de hacer la llamada a mongod damos como parámetro --configsvr (para configurar el servidor y a continuación damos la ruta y el puerto que le vamos a asignar) --dbpath "C:\data\configdb" --port 27019 hacer esto con nuestros tres servidores(puertos y rutas pueden variar).

  Ahora es la hora de crear nuestro router para ello le damos el nombre del host esta vez a la instancia mongos y las ips bajo el parámetro configdb. Es decir quedaría > mongos --configdb nombre_host:27019, nombre_host:XXXXX, nombre_host:XXXXX --port 27030

  Ahora toca la parte de crear los Shard, para ello es más recomendable crearlo en los replicaSet

  Para ello creamos carpetas de shard en cada data. Dentro de la instancia mongod damos como parámetro --shardsvr --dbpath "ruta\shard" --port27040

Lo que queda es agregar los Shard a los clúster. Vamos a entrar dentro del router mediante su puerto >mongo --puerto 27030(en nuestro caso)

Con la variable reservada sh.addSard(“rep1/nombre_host:XXXXX”) donde las equis es el número del puerto que vamos a ingresar. Y rep1 es el nombre de nuestro replicaSet

Por último vamos a activarlo sobre nuestros db mediante el reservado sh.enableSharding(“db_nombre”), recordemos que tiene que está optimizado el criterio de consulta en este caso las _id y fecha. Así que a sh.shardCollection(“sensores.sensor”, {“_id”: 1, “fecha”: 1})