CASSANDRA

Apache Cassandra es una base de datos columnar diseñada para ser ejecutada como un cluster muy escalable y con una gran disponibilidad.

Instalación

Al igual que con REDIS, podemos hacer una instalación desde una imagen docker con todo incluido, o bien, podemos hacer una instalación desde un sistema base nosotros mismos.

En primer lugar veremos cómo hacer la instalación desde cero, ya sea que estemos trabajando en una imagen docker compuesta por varios módulos o bien que queramos la instalación en un equipo sin docker.

Instalación

Pre-requisitos:

```
$ sudo apt-get install python openjdk-8-jdk
```

Para obtener los binarios se puede correr el comando:

```
$ wget
https://www-us.apache.org/dist/cassandra/3.11.10/apache-
cassandra-3.11.10-bin.tar.gz
```

Y para descomprimir:

```
$ tar -xzf apache-cassandra-3.11.10-bin.tar.gz
```

Instalación

Desde la nueva carpeta en *bin* se puede correr cassandra, pero seguramente prefieren agregarlo al *path*.

```
$ echo 'export PATH=$PATH:~/apache-cassandra-3.11.10/bin'
>> ~/.bashrc
```

En el caso de docker es mas sencillo agregarlo en el *Dockerfile* como variable de entorno.

```
$ ENV PATH="/apache-cassandra-3.11.10/bin:$PATH"
```

Instalación

En el caso que se desee usar una imagen Docker pre-instalada y lista para usarse, se puede utilizar la instrucción:

```
$ docker pull spotify/cassandra
```

Una vez que CASSANDRA esté instalada se puede iniciar con:

```
$ cassandra -f
```

Si estan en docker, por defecto tienen superusuario, asi que necesitan una bandera extra.

```
$ cassandra -Rf
```

Para verificar que el servidor este corriendo:

```
$ nodetool status
```

Y para iniciar nuestra conexión por línea de comandos CQL:

```
$ cqlsh
```

CQL

Lo primero es crear un KEYSPACE:

```
cqlsh> CREATE KEYSPACE bdnosql WITH replication = {'class'
: 'SimpleStrategy', 'replication_factor' : 1};
```

Y para usarlo:

```
cqlsh> USE bdnosql;
```

CQL

Se crean tablas como en SQL

cqlsh> CREATE TABLE estaciones (id int PRIMARY KEY, nombre text);

En CQL el *primary key* es obligatorio.

cqlsh> DESCRIBE tables;

Se tienen tipos de datos *boolean*, *blob*, *ascii*, *bigint*, *Counter*, *decimal*, *double*, *float*, *inet*, *int*, *text*, *varchar*, *timestamp*, *variant*

CQL

Se agregan valores igual que en SQL.

```
cqlsh> INSERT INTO estaciones (id,nombre) VALUES
(1,'Primavera');
cqlsh> INSERT INTO estaciones (id,nombre) VALUES
(2,'Verano');
cqlsh> INSERT INTO estaciones JSON
'{"id":3,"nombre":"Otoño"}';
cqlsh> INSERT INTO estaciones JSON
'{"id":4,"nombre":"Invierno"}';
```

CQL

Y también se consultan igual.

```
cqlsh> SELECT * FROM estaciones;

cqlsh> SELECT nombre FROM estaciones WHERE id = 2;

cqlsh> SELECT JSON * FROM estaciones WHERE id = 3;
```

CQL

Como es de esperar, también se pueden modificar y borrar. Se necesita especificar perfectamente las entradas, por eso es necesario el *id*.

```
cqlsh> UPDATE estaciones SET nombre = 'Frio' WHERE id = 4;
```

```
cqlsh> DELETE [columna] FROM estaciones
    WHERE id IN (1,4);
```

CQL

Aquí también podemos agrupar instrucciones para minimizar las rondas que se hacen con el servidor. A diferencia de REDIS, Cassandra si distribuye, pero solo ofrece aislamiento en lo que le toca a cada nodo.

```
cqlsh> BEGIN BATCH
           INSERT INTO estaciones (id,nombre)
               VALUES (1,'Primavera');
           INSERT INTO estaciones (id,nombre)
               VALUES (4,'Invierno');
       APPLY BATCH ;
```

Dentro del BATCH se pueden usar INSERT, UPDATE y DELETE.

CQL

Veamos otro ejemplo.

```
cqlsh> CREATE TABLE chat (
        usuario text,
        fecha timestamp,
        amigo text,
        mensaje text,
        ultima_actividad timestamp STATIC,
        PRIMARY KEY (usuario,fecha)
    );
```

CQL

```
cqlsh> INSERT INTO chat
  (usuario,fecha,amigo,mensaje,ultima_actividad)
  VALUES (
    'usuario1',
    toTimestamp(now()),
    'usuario2',
    'Hola mundo',
    toTimestamp(now())
  );
```

CQL

A pesar de ser tan similar a SQL, hay un par de cosas que se tienen que tener en cuenta.

Partition key Esta es la PRIMARY KEY (si es simple). Toda la información que se agrupe bajo un mismo renglón será guardada en un mismo nodo del cluster.

Cluster key Si la PRIMARY KEY es compuesta, desde la segunda parte en adelante, se tomará para formar las familias de columnas, cada valor se agrupará bajo esos grupos, los cuales se mantendrán ordenados en el almacenamiento.

CQL

```
cqlsh> SELECT * FROM chat;
cqlsh> SELECT usuario FROM chat;
cqlsh> SELECT amigo FROM chat;
cqlsh> SELECT amigo FROM chat WHERE usuario = 'usuario1';

cqlsh> SELECT amigo FROM chat WHERE
    fecha < toTimestamp(now());
cqlsh> SELECT amigo FROM chat WHERE
    usuario = 'usuario1' AND fecha < toTimestamp(now());

cqlsh> SELECT DISTINCT amigo FROM chat WHERE
    usuario = 'usuario1';
cqlsh> SELECT mensaje FROM chat WHERE
    usuario = 'usuario1'ORDER BY fecha;
```

CQL

```
cqlsh> CREATE TABLE t (
    a int,
    b int,
    c int,
    d int
    PRIMARY KEY ((a,b),c,d)
);
```

Otro cambio importante relacionado con los PRIMARY KEY es el GROUP BY.
En esencia: Solo se puede agrupar según las columnas del PRIMARY KEY y en el orden en el mismo orden.

Índices secundarios

Cassandra no es capaz de hacer consultas que coincida con columnas que no sean parte del PRIMARY KEY. Para eso son útiles los índices secundarios, pero se debe tener en cuenta que aún así solo funcionan en conjunto con las *partition keys*.

cqlsh> CREATE INDEX ON chat(amigo);

No siempre es bueno crear índices, en general las recomendaciones son:

- Que no sea un campo con demasiados valores diferentes ni muy pocos tampoco.
- Que no sean campos que se modifiquen muy seguido.

Materialized Views

Como ya vimos, el orden de las columnas es muy importante para la eficiencia de las consultas. Una herramienta muy poderosa para consultas específicas (a costa de mayor uso de almacenamiento) son las Materialized Views:

```
cqlsh> CREATE MATERIALIZED VIEW chat_poramigo AS
SELECT amigo,usuario,fecha FROM chat WHERE amigo IS NOT
NULL AND usuario IS NOT NULL AND fecha IS NOT NULL
PRIMARY KEY (amigo,usuario,fecha);
```

Materialized Views

Las Materialized Views:

- No se pueden modificar directamente, solo reflejan.
- Debe incluir TODAS las PRIMARY KEY de la tabla base y también deben ser PRIMARY KEY en la nueva, lo que puede cambiar es el orden.
- Además de las PRIMARY KEY originales, solo se puede usar UNA columna extra en las PRIMARY KEYS nuevas.
- No se pueden incluir columnas STATIC dentro de una MV.
- Todas las columnas que se agreguen a la nueva PRIMARY KEY tienen que tener las restricción NOT NULL.

Configuración

Por defecto, la ruta de configuración de CASSANDRA se encuentra en:

`~/apache-cassandra-3.11.5/conf/cassandra.yaml`

Hay unas pocas entradas de configuración que son obligatorias y que es bueno tener en cuenta:

cluster_name: 'MiPrimerCluster'
listen_address: localhost
rpc_address: localhost

Configuración

Rutas

Cassandra también permite configurar las rutas que se usarán para almacenar la información.

```
data_file_directories:
    - /var/lib/cassandra/data

commitlog_directory: /var/lib/cassandra/commitlog

saved_caches_directory: /var/lib/cassandra/saved_caches
```

Esta configuración es principalmente útil cuando se tienen muchos discos para mejorar el desempeño general del sistema.

Configuración

Cluster

Podemos obtener un cluster local con ayuda de VirtualBox. Podemos usar tres máquinas virtuales, cada una necesita su propia configuración.

```
cluster_name: 'VBox'
listen_address: 192.168.56.4
rpc_address: 192.168.56.4

seed_provider:
   -class_name: org.apache.cassandra.locator.SimpleSeedProvider
      -seeds: "192.168.56.3,192.168.56.4,192.168.56.5"
```

Hay que recordar agregar a la configuración de las máquinas un adaptador de red de solo-anfitrión para que se puedan comunicar entre ellas.

Configuración

Cluster

Si alguno de los nodos de su cluster fue utilizado antes con configuración individual, o si fue clonado de una maquina virtual probablemente necesitaran limpiar Cassandra:

```
$ rm -rf ~/apache-cassandra-3.11.5/data/*
```

```
$ rm -rf ~/apache-cassandra-3.11.5/logs/*
```

CQL

Nos conectamos al cluster.

```
$ cqlsh 192.168.56.4
```

El *keyspace* ahora va a cambiar.

```
cqlsh> CREATE KEYSPACE datosprueba WITH replication =
{'class' : 'SimpleStrategy', 'replication_factor' : 3};
```

Recordemos que la replicación debe ser 1 si no es un cluster. También puede ser *NetworkTopologyStrategy*. Esa opción es para cuando se quieren especificar replicaciones diferentes según los centros de datos en donde se encuentren los nodos.

cqlsh> USE datosprueba;

Ahora en el cluster podemos utilizar las opciones de consistencia. Cada que queramos cambiar la consistencia para las siguientes instrucciones.

cqlsh> CONSISTENCY QUORUM;

En los drivers la consistencia se suele manejar en cada una de las escrituras o lecturas.

Actividades

- Desarrollar una plataforma de libros.
- Debe incluir producto, cliente y calificación.
- Cada cliente debe tener información asociada (nombre, país, membresía, etc.)
- Además, cada cliente debe poder elegir una categoría para cada uno de sus libros (fantasía, misterio, etc.).
- Todos deben participar como clientes.

Como administradores, deben poder buscar cosas como:

- La categoría preferida de un cliente dado.
- Obtener los clientes que más disfrutaron un libro dado.
- Los mejores libros de una categoría dada.