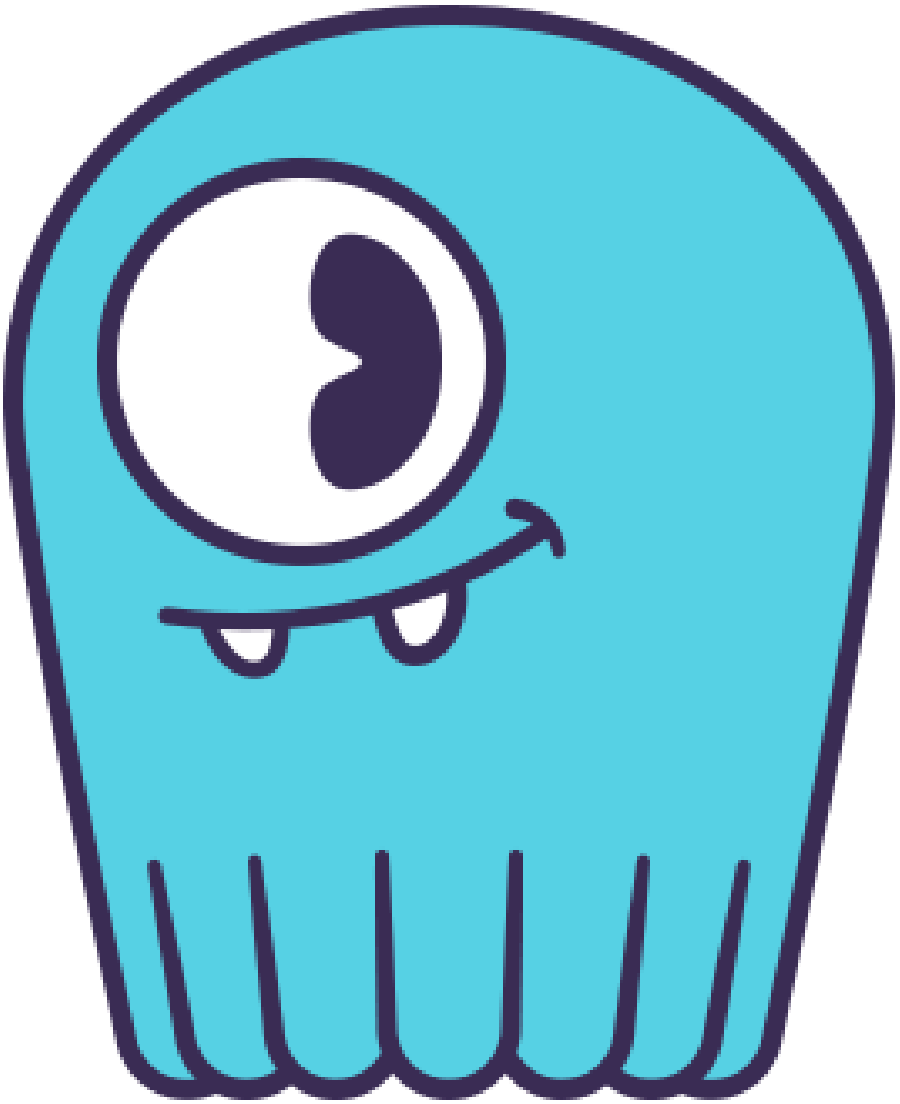
SGBD NoSQL: ScyllaDB



Grupo A1

Por Pedro Antonio Mayorgas Parejo y Adrián Acosa Sánchez

Índice

Descarga e instalación de Scylla NoSQL

Scylla es un sistema gestor de bases de datos NoSQL que se basa en un modelo orientado a columnas. Se diseñó para ser compatible con Cassandra pero con un mejor rendimiento en velocidad y latencia.

Vamos a comenzar con la instalación siguiendo el tutorial de la documentación oficial. En primer lugar tenemos que instalar todas sus dependencias. Lo primero que nos pide el tutorial es que instalemos el paquete epel-release:

```
[usuario@localhost ~]$ sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
Oracle Linux 8 BaseOS Latest (x86_64)                                                                 24 kB/s | 3.6 kB     00:00
Oracle Linux 8 BaseOS Latest (x86_64)                                                                5.2 MB/s |  41 MB     00:07
Oracle Linux 8 Application Stream (x86_64)                                                            29 kB/s | 3.9 kB     00:00
Oracle Linux 8 Application Stream (x86_64)                                                           7.9 MB/s |  31 MB     00:03
Latest Unbreakable Enterprise Kernel Release 6 for Oracle Linux 8 (x86_64)                            31 kB/s | 3.0 kB     00:00
Latest Unbreakable Enterprise Kernel Release 6 for Oracle Linux 8 (x86_64)                           7.7 MB/s |  31 MB     00:03
Visual Studio Code                                                                                   8.1 kB/s | 3.0 kB     00:00
Visual Studio Code                                                                                   5.8 MB/s |  21 MB     00:03
Last metadata expiration check: 0:00:12 ago on Tue 28 Dec 2021 17:55:26 CET.
epel-release-latest-7.noarch.rpm                                                                      35 kB/s |  15 kB     00:00
Dependencies resolved.
================================================================================================================================
 Package                          Architecture                 Version                     Repository                        Size
================================================================================================================================
Installing:
 epel-release                     noarch                       7-14                        @commandline                      15 k
```

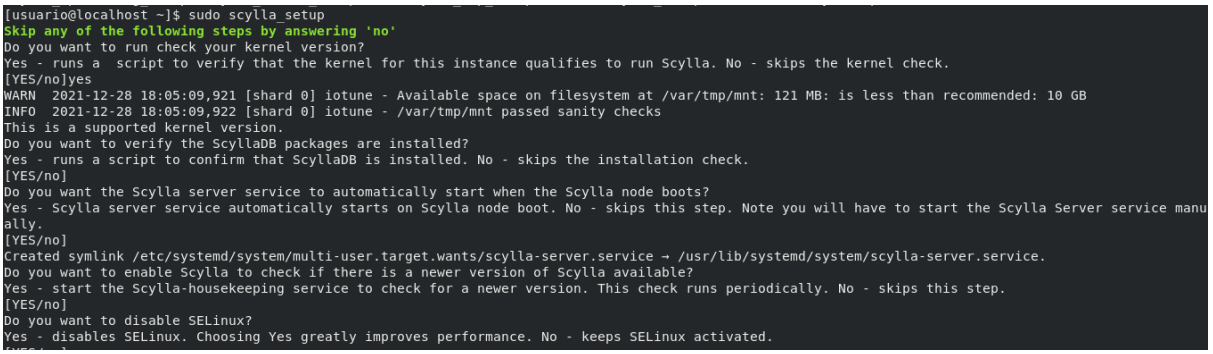
Tras esto, tenemos que añadir el repositorio de Scylla a nuestra lista de repositorios de yum:

```
[usuario@localhost ~]$ sudo curl -o /etc/yum.repos.d/scylla.repo -L http://downloads.scylladb.com/rpm/centos/scylla-4.5.repo
[sudo] password for usuario:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   312  100   312    0     0    866      0 --:--:-- --:--:-- --:--:--   866
[usuario@localhost ~]$
```

Una vez ya tenemos añadido el repositorio de Scylla, podemos proceder a instalar el paquete de yum de scylla:

```
[usuario@localhost ~]$ sudo dnf install scylla
Extra Packages for Enterprise Linux 7 - x86_64                                                       6.4 MB/s |  17 MB     00:02
Scylla for Centos - x86_64                                                                           123 kB/s | 124 kB     00:01
Scylla for centos                                                                                     23 kB/s |  11 kB     00:00
Last metadata expiration check: 0:00:01 ago on Tue 28 Dec 2021 18:00:10 CET.
Dependencies resolved.
================================================================================================================================
 Package                          Architecture       Version                                    Repository                  Size
================================================================================================================================
Installing:
 scylla                           x86_64             4.5.2-0.20211114.26aca7b9f                 scylla-4.5                 6.9 k
Installing group/module packages:
 scylla-kernel-conf               x86_64             4.5.2-0.20211114.26aca7b9f                 scylla-4.5                 8.6 k
     replacing  tuned.noarch 2.15.0-2.0.1.el8
Installing dependencies:
 python2                          x86_64             2.7.18-7.0.1.module+el8.5.0+20361+8a9d3d27 ol8_appstream              110 k
 python2-libs                     x86_64             2.7.18-7.0.1.module+el8.5.0+20361+8a9d3d27 ol8_appstream              6.0 M
 python2-pip-wheel                noarch             9.0.3-18.module+el8.3.0+7833+4aaf98ce      ol8_appstream              1.0 M
 python2-pyyaml                   x86_64             3.12-16.module+el8.3.0+7833+4aaf98ce       ol8_appstream              191 k
```

El tutorial nos indica que ejecutemos un script de configuración inicial para activar los servicios principales de Scylla. Este script invoca a otros script que configuran varios ajustes del sistema. También ejecuta un pequeño benchmark en nuestro almacenamiento y genera el archivo de configuración /etc/scylla.d/io.conf. Cuando se haya terminado de generar este archivo, ya podremos iniciar Scylla. Hay que tener en cuenta que Scylla no puede ejecutarse sin un sistema de archivos XFS o el archivo io.conf. En la siguiente imagen se ve la ejecución del script de configuración.



Measuring sequential write bandwidth: 486 MB/s (deviation 94%)
Measuring sequential read bandwidth: 744 MB/s (deviation 37%)
Measuring random write IOPS: 406 IOPS (deviation 27%)
Measuring random read IOPS: 65 IOPS (deviation 48%)
Writing result to /etc/scylla.d/io_properties.yaml
Writing result to /etc/scylla.d/io.conf
Created symlink /etc/systemd/system/multi-user.target.wants/scylla-node-exporter.service → /usr/lib/systemd/system/scylla-node-exporter.service.
Do you want to set the CPU scaling governor to Performance level on boot?
Yes - sets the CPU scaling governor to performance level. No - skip this step.
[YES/no]

scylla_setup accepts command line arguments as well! For easily provisioning in a similar environment than this, type:

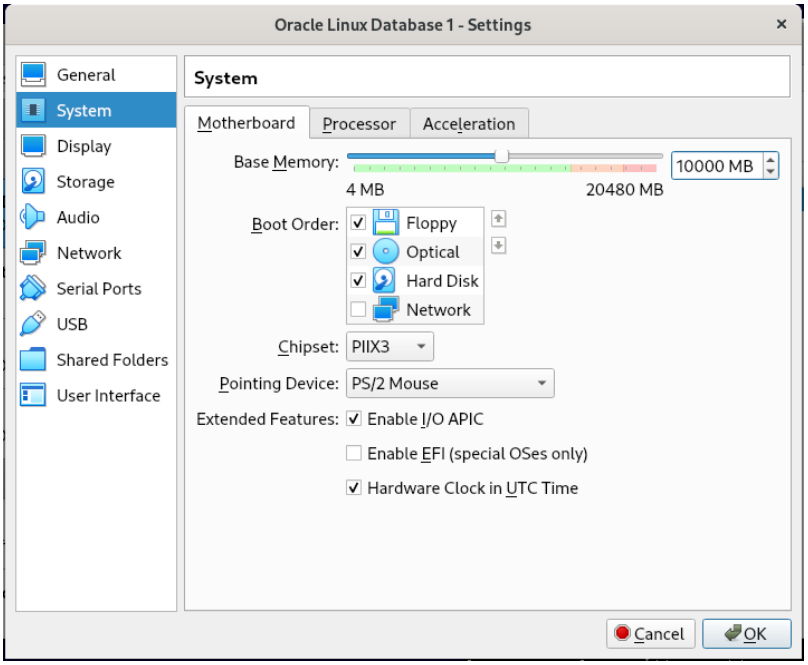
```
scylla_setup --no-raid-setup --nic enp0s3 --io-setup 1
```

Problemas que pueden aparecer en la instalación.

Durante la instalación puede aparecer el problema siguiente:

Only 992 MiB per shard; this is below the recommended minimum of 1 GiB/shard scylla.

¿Qué indica este problema? Indica que en una máquina con 4 cores y 6 GB de Ram originalmente, a cada core le intenta asignar 1 GiB de RAM como caché intermedia entre el disco y la CPU. El sistema requiere su propia RAM que ya está previamente asignada a sí mismo unos 1.5 GiB aproximadamente. Así que hemos tenido que ampliar la RAM de la máquina virtual para poder ejecutar correctamente el servidor.



DML y DDL utilizado por Scylla

Scylla usa **CQL (Cassandra Query Language).** Este lenguaje guarda los datos en tablas, cuyo esquema define el diseño de dichos datos en la tabla, y esas tablas se agrupan en "keyspaces" o espacios de claves. Un espacio de claves define una serie de opciones que se aplican a todas las tablas que contiene, siendo la más destacada la estrategia de replicación utilizada por el espacio de claves. Una aplicación solo puede tener un espacio de claves.

Para crear un espacio de claves, se hace con la sentencia:

```
cqlsh> CREATE KEYSPACE TEMA_4 WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
cqlsh> USE TEMA_4;
cqlsh:tema_4>
```

Hay una sentencia que permite cambiar el espacio de claves actual. Esta sentencia es la sentencia **USE**, como aparece en la imagen superior.

Al igual que con las tablas en SQL, también podemos modificar los datos de un keyspace con la sentencia **ALTER KEYSPACE:**

ALTER KEYSPACE `keyspace_name` WITH `options`

Y para borrarlos se hace con la sentencia **DROP KEYSPACE:**

DROP KEYSPACE [IF EXISTS] `keyspace_name`

La creación de tablas tiene la siguiente sintaxis:

```
cqlsh:tema_4> CREATE TABLE TEMARIO (NOMBRE_TEMA text PRIMARY KEY, DESCRIPCION text, N_PAGINAS int);
```

Si queremos borrar todos los datos de una tabla pero no la propia tabla se usa la sentencia **TRUNCATE**, cuyo uso es:

TRUNCATE [TABLE] table_name

En cuanto al DML, todas las sentencias se hacen de forma muy parecida a SQL y hacen lo mismo. CQL ofrece un modelo similar a SQL donde los datos se almacenan en tablas que contienen filas de columnas. Por esa razón todas las sentencias se realizan de igual manera que en SQL. Son las siguientes:

- Selección:

```
cqlsh:tema_4> INSERT INTO TEMARIO (NOMBRE_TEMA, DESCRIPCION) VALUES ('Tema 3', 'SGBD NoSQL');
cqlsh:tema_4> SELECT * FROM TEMARIO;

 nombre_tema | descripcion | n_paginas
-------------+-------------+-----------
      Tema 2 |  SGBD NoSQL |       100
      Tema 3 |  SGBD NoSQL |      null
      Tema 4 |  SGBD NoSQL |       100
      Tema 1 |  SGBD NoSQL |       100

(4 rows)
```

- Inserción:

```
cqlsh:tema_4> INSERT INTO TEMARIO (NOMBRE_TEMA, DESCRIPCION, N_PAGINAS) VALUES ('Tema 4', 'SGBD NoSQL', 100);
```

- Actualización:

```
cqlsh:tema_4> UPDATE TEMARIO SET n_paginas = 400 WHERE nombre_tema = 'Tema 3';
cqlsh:tema_4> SELECT * FROM TEMARIO;

 nombre_tema | descripcion | n_paginas
-------------+-------------+-----------
      Tema 2 |  SGBD NoSQL |       100
      Tema 3 |  SGBD NoSQL |       400
      Tema 4 |  SGBD NoSQL |       100
      Tema 1 |  SGBD NoSQL |       100

(4 rows)
```

- Borrado de elementos:

```
cqlsh:tema_4> DELETE FROM TEMARIO WHERE nombre_tema = 'Tema 2';
cqlsh:tema_4> SELECT * FROM TEMARIO;

 nombre_tema | descripcion | n_paginas
-------------+-------------+-----------
      Tema 3 |  SGBD NoSQL |       400
      Tema 4 |  SGBD NoSQL |       100
      Tema 1 |  SGBD NoSQL |       100

(3 rows)
```

Conexión desde aplicaciones

Hay drivers como los de SGBD SQL normales, la única diferencia está en que la conexión se hace hacia clusters. Pero nosotros no tenemos un cluster, requiere mucha memoria y trabajo levantar un cluster, es válido para un servidor simple NoSQL, pero tiene la ventaja de que se puede escalar y replicar automáticamente.

Creamos un objeto Cluster que contiene la siguiente información, dirección IP, puerto, certificado SSL que Cassandra / Scylla integran el cifrado.

```
from cassandra.cluster import Cluster
cluster = Cluster(['192.168.0.1', '192.168.0.2'], port=..., ssl_context=...)
```

Podemos indicarlo con localhost al 127.0.0.1.

Luego abrimos como en SQL sería un conn (connection) y un cursor a la vez. Que es una session.

```
session = cluster.connect()
```

También podemos indicar como en SQL, en concreto MariaDB y MySQL. El keyspace (Database) que se quiera utilizar directamente desde el connect.

Luego con el objeto de la sesión que hemos creado podemos ejecutar sentencias CQL que nos permita ejecutar DML, DDL,...

```
rows = session.execute('SELECT name, age, email FROM users')
for user_row in rows:
    print user_row.name, user_row.age, user_row.email
```

Implementación sobre el SI de la práctica

Sería adecuado ya que cada operación que realiza Scylla es atómica, es decir un INSERT es atómico en sí mismo, por lo que no hay datos parciales o incompletos o se muestra un dato en el estado anterior a la operación atómica.

No hay transacciones, existen BATCHs que es en esencia agrupaciones de transacciones atómicas pero no las necesitamos porque el objetivo de este tipo de NoSQL es evitar el modelo relacional donde las transacciones ocurren al utilizar tablas relacionales que tienen que modificarse secuencialmente.

Es decir en los ejemplos siguientes:

- https://cassandra.apache.org/doc/latest/cassandra/data_modeling/data_modeling_conceptual.html
- https://cassandra.apache.org/doc/latest/cassandra/data_modeling/data_modeling_rdbms.html
- https://cassandra.apache.org/doc/latest/cassandra/data_modeling/data_modeling_schema.html

Se ve un ejemplo con los hoteles, se tiene que romper el modelo relacional desde la 1ª Forma Normal. Y eliminar todas las relaciones posibles y unir en una tabla por ejemplo hotel_puntos_de_visita, toda la información necesaria que se agruparía en 2 o 3 tablas relacionales.

En nuestro trabajo sería que la tabla de Venta, agruparía más información que estaría incluida en la relación que ocurre entre cliente venta y stock.

Si en venta está la clave de cada uno ahora se incluiría:

Nombre_cliente, Nombre_producto, Cantidad, Domicilio, estado, Repartidor, DNI_ID_Repartidor,...

Esa sería la información clave que contiene la tabla que estaría preparada para miles de compras por segundo sin tener que contener la latencia en buscar el modelo relacional.

Pero la desventaja es que la integridad de los datos hay que asegurarla manualmente o utilizar BATCH o otras herramientas previstas previa investigación en el CQL (hemos visto que existen algunas pero no hemos nos hemos adentrado mucho en el tema ya que no es el objetivo de este trabajo).

Lo mismo para los usuarios donde la tabla más o menos sería la misma, la de repartidores y stock también sería el mismo tipo de tabla. Pero al no tener referencias o claves externas se hacen muy rápidas en escritura con Scylla.

Bibliografía

- Página web oficial de Scylla: https://www.scylladb.com/
- Documentación oficial de Scylla: https://docs.scylladb.com/getting-started/
- Página oficial de descarga de Scylla: https://www.scylladb.com/download/?platform=rhel&version=scylla-4.5#open-source
- Paquete epel-release: https://fedoraproject.org/wiki/EPEL/es
- Driver de Python para Scylla: https://docs.scylladb.com/using-scylla/drivers/cql-drivers/scylla-python-driver/