

MTP03. ESCENARIO CON SISTEMAS NOSQL

1. ¿QUÉ ES UN SISTEMA NOSQL?

Es un sistemas de gestión de bases de datos que difieren del modelo clásico de **SGBDR** (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes, siendo el más destacado que no usan **SQL** como lenguaje principal de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones **JOIN**, ni garantizan completamente **ACID** (atomicidad, consistencia, aislamiento y durabilidad) y habitualmente escalan bien horizontalmente. Los sistemas **NoSQL** se denominan a veces "no solo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo **SQL**.

En este trabajo voy a exponer un poco sobre el software Apache Cassandra, como se instalaría y como se desplegaría.

2. ¿QUE ÉS APACHE CASSANDRA?

Apache Cassandra se trata de un software **NoSQL distribuido** y basado en un modelo de almacenamiento de «clave-valor», de código abierto que está escrita en **Java**. Permite grandes volúmenes de datos en forma distribuida. Por ejemplo, lo usa Twitter para su plataforma. Su objetivo principal es la **escalabilidad** lineal y la disponibilidad. La arquitectura distribuida de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo **P2P** con lo que la redundancia es máxima. Está desarrollada por Apache Software Foundation.



Apache Cassandra tiene pros y contras a la hora de usarlo:

- Ventajas:
 - Alta disponibilidad
 - Tolerancia a particiones y excalados
 - Cantidad de recursos disponibles
- Desventajas:
 - Conexión de nuevos nodos algo compleja
 - Debemos tener premeditado las queries que vayamos hacer.

3. INSTALACIÓN DE APACHE CASSANDRA

El software que utiliza Apache Cassandra es **Datastax** y hay distintas maneras de instalarlo. Una de las maneras es añadiendo los repositorios y descargándolo desde ahí. Una vez instalado se usaría la consola **cqlsh** propia de Cassandra, desde donde se puede crear los esquemas y hacer las respectivas **queries** y consultas que necesitamos.

Otra manera es mediante dockers o la instalación del **server** y **studio** para poder hacerlo todo de manera gráfica. Este es el método por el que he optado. Al principio lo intenté mediante dockers que se pueden obtener de **docker hub** con los siguientes pulls:

```
alunsiito@alunsiito: ~  
alunsiito@alunsiito:~$ sudo docker pull datastax/dse-studio  
Using default tag: latest  
latest: Pulling from datastax/dse-studio  
6ec8c9369e08: Pull complete  
3aa4e9b77806: Pull complete  
1a9b6b4d80d1: Pull complete  
845c172f556e: Downloading 23.1MB/105.1MB  
68ab8b21369a: Downloading 14.56MB/19.81MB  
a136fbb7d9e1: Download complete  
9801dc9ca21d: Waiting  
7326c8cd217b: Waiting  
2daf628ee62b: Waiting  
0fa1da0b5dd5: Waiting
```

```
alunsiito@alunsiito:~$ sudo docker pull makingsense/dse-server:6.8.0-apt  
6.8.0-apt: Pulling from makingsense/dse-server  
fc7181108d40: Pull complete  
73f08ce352c8: Pull complete  
eac271a34b40: Pull complete  
774220066612: Downloading 24.19MB/104.3MB  
54d82567f12a: Downloading 6.727MB/7.374MB  
e06e78a22c9c: Download complete  
c56f0fdf1a7a: Download complete  
592492bcaa23: Waiting  
adc487d58273: Waiting  
dc7790faf255: Waiting  
e60f1de511ab: Waiting  
aff58ad109fe: Waiting  
010dcc3c5f62: Waiting
```

Arrancaríamos el docker del server con las siguientes opciones:

```
alunsiito@alunsiito:~$ sudo docker run -p 9042:9042 -e DS_LICENSE=accept --name corostudio-server -d makingsense/dse-server:6.8.0-apt -g -s -k
```

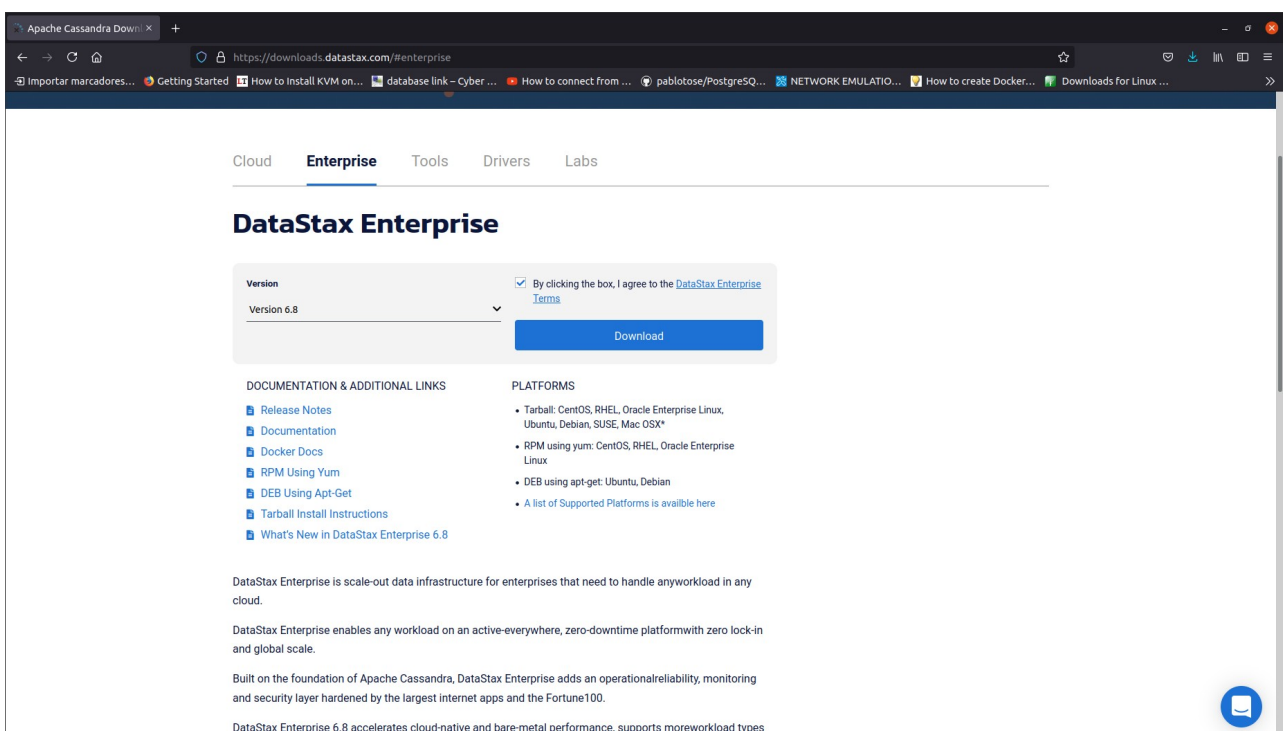
Y luego el studio con la siguiente línea:

```
alunsiito@alunsiito:~$ sudo docker run -e DS_LICENSE=accept --link corostudio-server -p 9091:9091 --name corostudio-studio -d datastax/dse-studio
```

Estas líneas lo que hacen es crear un contenedor llamado ‘**corostudio-server**’ y otro llamado ‘**corostudio-studio**’, estando este último enlazado al primero para que trabajen en el mismo puerto y, en base a los cambios que se apliquen en el **studio**, se hagan en el **server**. A mi no me funcionó porque no me arrancar el primer **docker** haga lo que haga, se me queda apagado y no hay manera de encenderlo. He probado a reinstalar todos los paquetes de **docker** y no puede enlazarse el **studio** con el **server** ya que este último no arranca. Por todo esto, opté a instalarlo de manera local en una máquina.

Para instalarlo en la máquina hay que hacer lo siguiente:

- Nos dirigimos a la página principal de **Datastax**, donde si nos vamos a sus productos podremos descargar el **Datastax Enterprise** y el **Dat Enterprise astax Studio**. Nos pedirán nombre, correo y empresa para descargar el software:



Cloud Enterprise **Tools** Drivers Labs

DataStax Enterprise OpsCenter

DataStax Enterprise Kubernetes Operator

DataStax Apache Pulsar Connector

Stargate Open Source Data API Gateway

DataStax Apache Kafka Connector

DataStax Desktop

DataStax Studio

CQLSH

DataStax Bulk Loader

Graph Loader

DataStax DevCenter

DataStax Studio

Version

Version 6.8



☒ By clicking the box, I agree to the [DataStax Studio Terms](#)

Package

macOS/Linux



Download

DOCUMENTATION & ADDITIONAL LINKS

[Documentation](#)

[Install Docs](#)

PLATFORMS

• CentOS, RHEL, Oracle Enterprise Linux, Ubuntu, Debian, SUSE, Mac OS*, Windows

DataStax Studio is a visual developer tool that makes it easy for developers to write queries, slice and dice your

También podemos optar por descargar el **Enterprise** mediante repositorios, lo cual es sinceramente más rápido:

· Instalaríamos la librería **libaio1** con:

```
$ sudo apt-get install libaio1
```

· Añadimos a los **repositorios** la ruta para descargar Datastax Enterprise:

```
$ echo "deb https://debian.datastax.com/enterprise/ stable main" | sudo tee -a /etc/apt/sources.list.d/datastax.sources.list
```

· Añadimos la **key** del repositorio:

```
$ curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
```

· Hacemos un **update** y empezamos a **descargar** la versión 6.8 con:

```
$ sudo apt-get install dse=6.8.0-1 |
dse-full=6.8.0-1 |
dse-libcassandra=6.8.0-1 |
dse-libgraph=6.8.0-1 |
dse-libhadoop2-client-native=6.8.0-1 |
dse-libhadoop2-client=6.8.0-1 |
dse-liblog4j=6.8.0-1 |
dse-lisolsr=6.8.0-1 |
dse-libspark=6.8.0-1 |
dse-libtomcat=6.8.0-1
```

· Terminamos iniciando el **servicio** y comprobando el **status** del nodo:

```
$ sudo service dse start
```

```
$ nodetool status
```

```
Datacenter: Cassandra
```

```
=====
```

```
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
```

```
-- Address    Load    Tokens  Owns    Host ID                               Rack
UN 127.0.0.1  82.43 KB  128     ?       40725dc8-7843-43ae-9c98-7c532b1f517e rack1
```

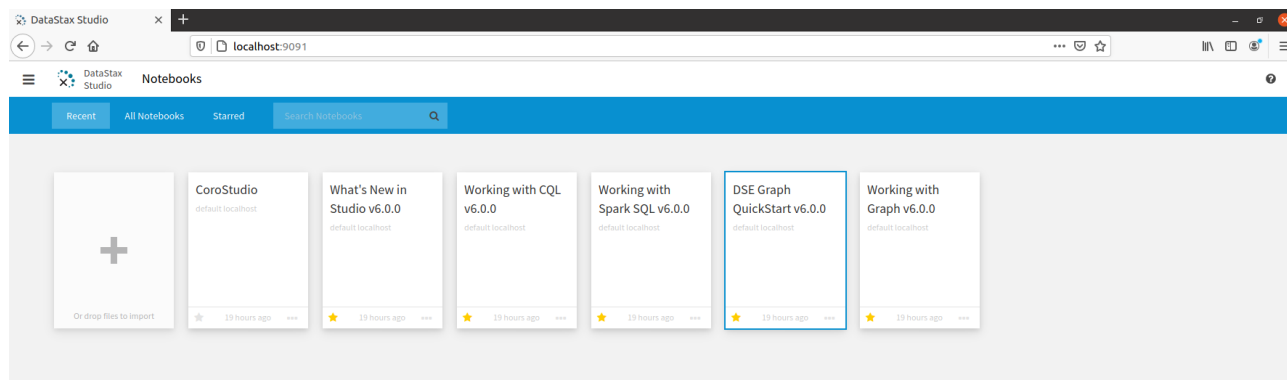
Para tener nuestro Studio funcionando, solamente habrá que extraer el archivo **tar** con:

```
$ tar xvfz [comprimido]
```

Nos extraerá una carpeta en la cual entraremos y **ejecutamos** el server con el comando:

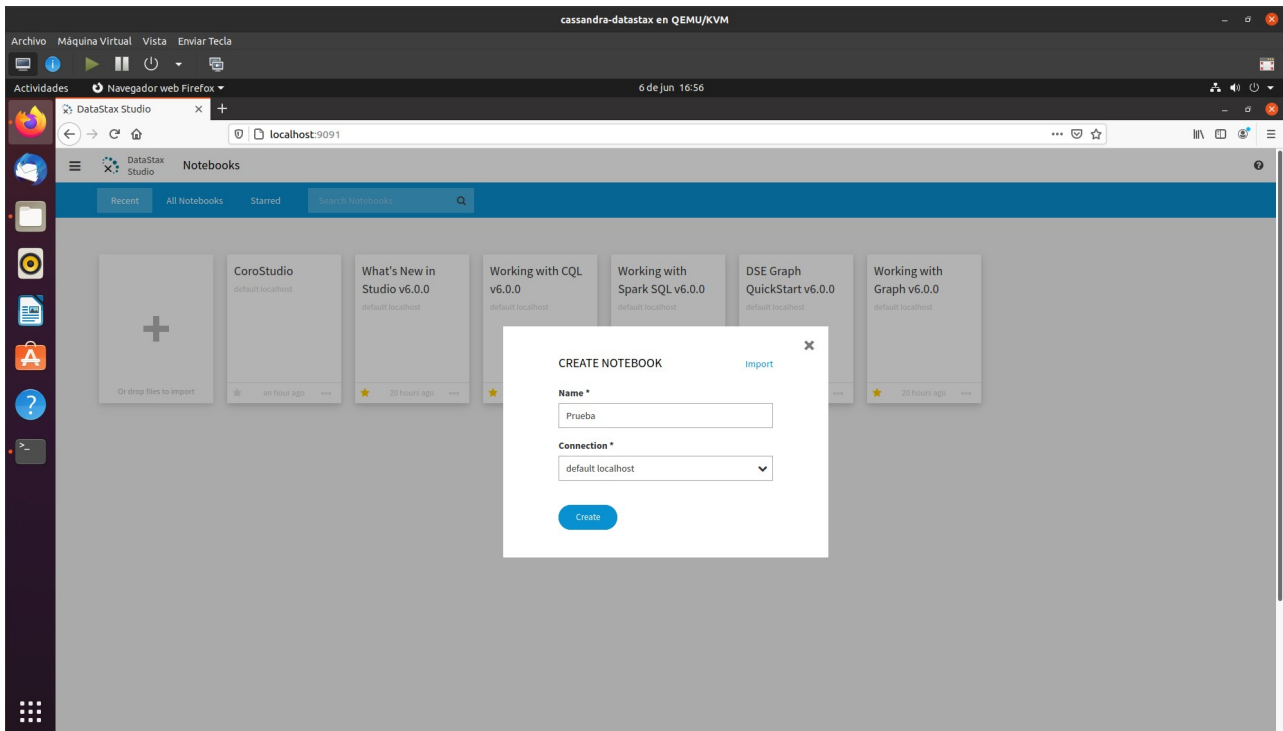
```
$ bin/server.sh
```

Y, si accedemos a localhost:9091 tendremos nuestro studio listo...



4. COMO USAR APACHE CASSANDRA CON DATASTAX STUDIO

Primero para empezar a hacer nuestras queries debemos crear nuestro notebook donde vamos a hacerlas.



Le damos un nombre y elegimos en este caso la conexión **default de localhost**. Una vez nos metemos en él podemos añadir todas las instrucciones **CQL** que queramos para crear nuestro **keyspace** y completarla con toda la información que queramos. Primero, debemos añadir un **keyspace**, el lo que se podría ver como la capa exterior que sirve como **contenedor** de toda la información almacenada. Toda la información en Cassandra debe de ir dentro de un **keyspace**. Puede ser vista como una **base de datos** en un sistema relacional, donde la base de datos tiene una colección de **tablas**.

Para crear nuestro keyspace debemos escribir '**create keyspace**' con unas opciones como la siguiente:

```
create keyspace prueba with replication = {'class': 'SimpleStrategy',  
'replication_factor': 1};
```

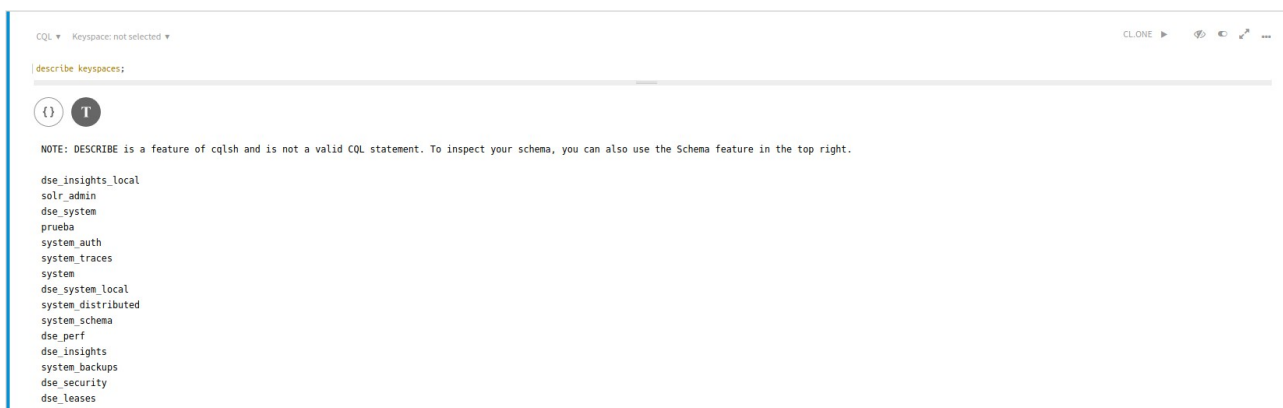
Donde **SimpleStrategy** especifica una replicación simple para el clúster.

Esto nos dará la instrucción como exitosa y, si escribimos '**describe keyspaces**' nos saldrán todas las keyspaces que tendremos instaladas:

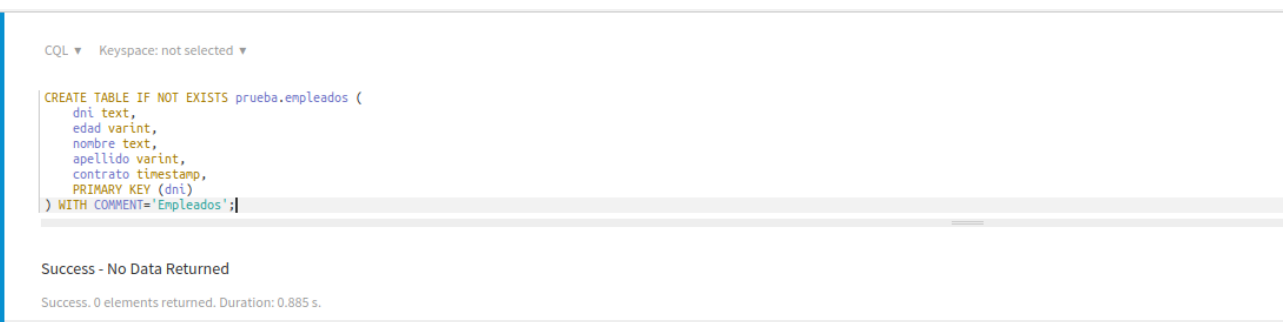
- Keyspace creada con éxito:



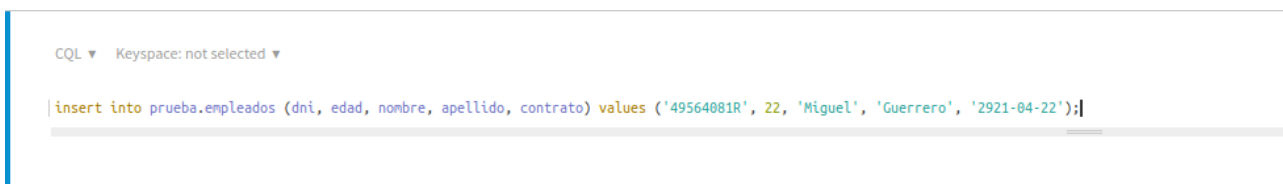
- Salida de describe keyspaces:



Una vez tenemos nuestro keyspace podemos añadirle las tablas que queramos. Vamos a añadir una tabla de prueba:



Añadimos nuevas filas a la tabla:



Y comprobamos que está en el keyspace haciendo un select:

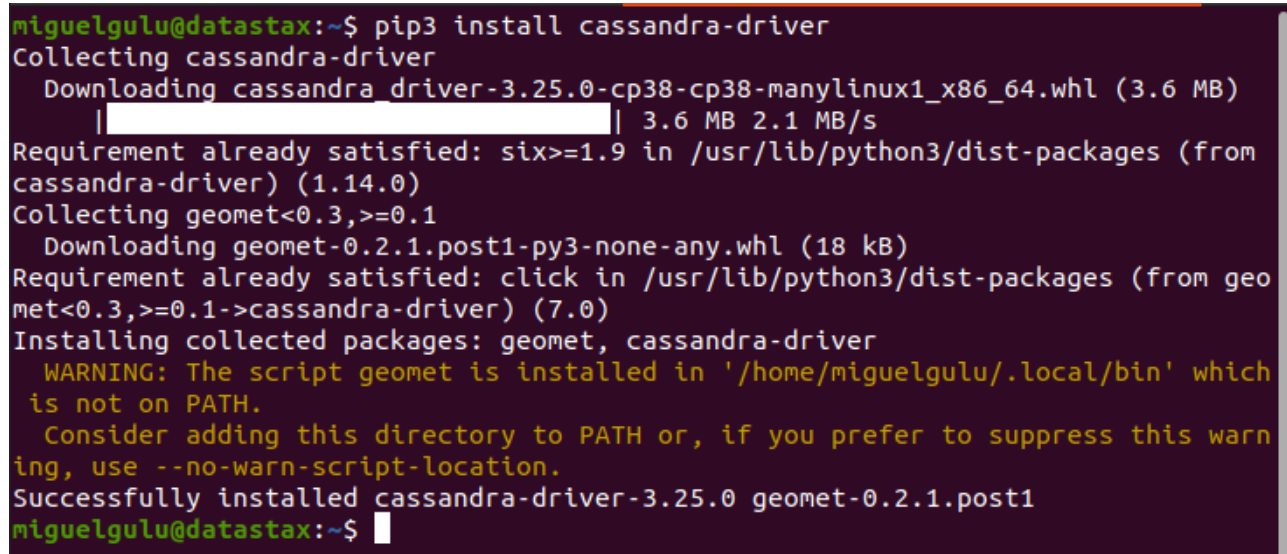


CQL Keyspace: not selected

```
|select * from prueba.empleados|
```

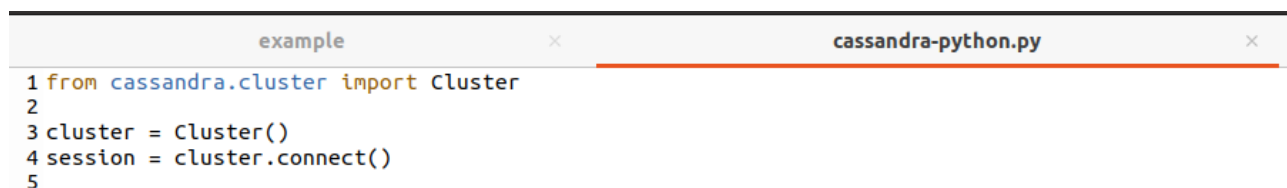
index ↑	dni	apellido	contrato	edad	nombre
0	49564081R	Guerrero	2921-04-21T22:00:00.000+0000	22	Miguel

También podemos conectar con el cluster de cassandra con python:



```
miguelgulu@datastax:~$ pip3 install cassandra-driver
Collecting cassandra-driver
  Downloading cassandra_driver-3.25.0-cp38-cp38-manylinux1_x86_64.whl (3.6 MB)
    | 3.6 MB 2.1 MB/s
Requirement already satisfied: six>=1.9 in /usr/lib/python3/dist-packages (from cassandra-driver) (1.14.0)
Collecting geomet<0.3,>=0.1
  Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from geomet<0.3,>=0.1->cassandra-driver) (7.0)
Installing collected packages: geomet, cassandra-driver
  WARNING: The script geomet is installed in '/home/miguelgulu/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed cassandra-driver-3.25.0 geomet-0.2.1.post1
miguelgulu@datastax:~$
```

Instalaremos el cassandra driver con pip3, el cual se incorporará como módulo en nuestro python. El módulo se utilizaría de la siguiente manera:



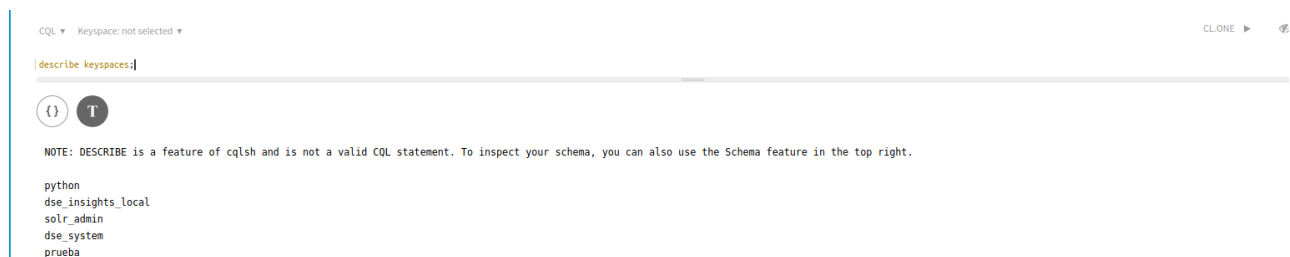
```
example cassandra-python.py
1 from cassandra.cluster import Cluster
2
3 cluster = Cluster()
4 session = cluster.connect()
5
```

Así se nos importa el módulo Cluster. La variable **cluster** que hemos creado a partir de 'Cluster()' lo que hace es apuntar al cluster de cassandra que tenemos creado en local y con el que interactuaremos. Crearemos otra variable que será **session** la cual establece la conexión con el cluster. A partir de aquí podemos hacer lo que queramos, como por ejemplo crear otro keyspace:



```
6 session.execute("""CREATE KEYSPACE python WITH replication = {'class':'SimpleStrategy',
'replication_factor':1}""")
```


Si ejecutamos con python3 el archivo que contiene todo esto y nos vamos a nuestro **notebook** y hacemos **'describe keyspaces'**...



Podemos ver como se crea el keyspace **python**!

A partir de aquí podemos hacer **connect** con:

session = cluster.connect('python')

Y crear las tablas con toda la información que queramos!

5. ENFOQUE REAL DE CASSANDRA

Este **NoSQL** tendría en mente usarlo para estudiar la evolución del coronavirus en los distintos países del mundo. He conseguido un archivo **JSON** el cual contiene registros de casos de coronavirus de distintas fechas a lo largo del tiempo:

```
{
  "records" : [
    {
      "dateRep" : "14/12/2020",
      "day" : "14",
      "month" : "12",
      "year" : "2020",
      "cases" : 746,
      "deaths" : 6,
      "countriesAndTerritories" : "Afghanistan",
      "geoId" : "AF",
      "countryterritoryCode" : "AFG",
      "popData2019" : 38041757,
      "continentExp" : "Asia",
      "Cumulative_number_for_14_days_of_COVID-19_cases_per_100000" : "9.01377925"
    },
    {
      "dateRep" : "13/12/2020",
      "day" : "13",
```

Este archivo json se puede **parsear** fácilmente gracias a **Python** y obtener los valores correspondiente a cada una de las columnas.

Y gracias a la conexión que hemos establecido gracias a los drivers de cassandra podemos insertar mediante instrucciones los distintos valores que hemos extraído del archivo JSON.

Una vez hemos poblado el **keyspace**, podemos hacer una comparación con la evolución de los casos a lo largo del tiempo y, dependiendo de los casos que haya por cada 100, se podrían imponer un nivel de estado de alarma y unas medidas.

CQL Keyspace: not selected

select * from corostudio.casos;

index	idregis	cases	continentexp	countriesandterritories	countryterritorycode	cumulative_number_for_14_da...	daterep	day	deaths	geoid	month	popdata2019	year
0	2	298	Asia	Afghanistan	AFG	7	2020-12-12T23:00:00.000+0000	13	9	AF	12	38041757	2020
1	3	113	Asia	Afghanistan	AFG	6	2020-12-11T23:00:00.000+0000	12	12	AF	12	38041757	2020
2	1	746	Asia	Afghanistan	AFG	9	2020-12-13T23:00:00.000+0000	14	6	AF	12	38041757	2020

Esto es un ejemplo de las entradas que he añadido para comparar como ha evolucionado el **coronavirus** en **Afghanistan** en los 12, 13 y 14 de Diciembre del 2020.

También podemos conectar con el cluster e interactuar con el gracias a un driver de cassandra que podemos descargar con **pip**:

`$ pip3 install cassandra-driver`

Y con esto tenemos importamos el módulo de **Cluster** a Python con las siguientes líneas en el script:

```
1 from cassandra.cluster import Cluster
2 import sys
3 import json
4 cluster = Cluster()
5 session = cluster.connect()
6 session = cluster.connect('corostudio')
7 #session.execute("""CREATE KEYSPACE IF NOT EXISTS WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1}""")
```

Y podemos crear scripts como por ejemplo **parsear** un archivo JSON para insertar cantidades masivas de datos en nuestra base de datos:

```
8
9 with open('coronacases.json') as f:
10     data = json.load(f)
11
12 #print(data)
13
14 for regis in data['records']:
15     id = 1
16     for campo in regis:
17         idreg = id
18         if campo == 'dateRep':
19             fecha = regis[campo]
20             #print(fecha)
21         elif campo == 'day':
22             dia = regis[campo]
23         elif campo == 'month':
24             mes = regis[campo]
25         elif campo == 'year':
26             ano = regis[campo]
27         elif campo == 'cases':
28             casos = regis[campo]
29         elif campo == 'deaths':
30             muertes = regis[campo]
31         elif campo == 'countriesAndTerritories':
32             pais = regis[campo]
33         elif campo == 'geoId':
34             geoID = regis[campo]
35         elif campo == 'countryterritoryCode':
36             codigo = regis[campo]
37         elif campo == 'popdata2019':
38             poblacion2019 = regis[campo]
39         elif campo == 'continentExp':
40             continente = regis[campo]
41         elif campo == 'cumulative_number_for_14_days_of_COVID-19_cases_per_100000':
42             por100k = regis[campo]
43
44     print(idreg, fecha, dia, mes, ano, casos, muertes, pais, geoID, codigo, poblacion2019, continente, por100k)
45
46 statement = session.prepare("INSERT INTO corostudio.casos (idRegis, dateRep, day, month, year, cases, deaths, countriesAndTerritories, geoId, countryterritoryCode, popData2019, continentExp, Cumulative_number_for_14_days_of_COVID19_cases_per_100000) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) IF NOT EXISTS")
47 session.execute(statement, [idreg, fecha, dia, mes, ano, casos, muertes, pais, geoID, codigo, poblacion2019, continente, por100k])
48 f.close()
49
50
```

Python Anchura del tabulador: 8 Ln 46, Col 137 INS

Y comprobamos que al ejecutarlo se guardan con una **consulta**:

```
| select * from corostudio.casos;
```

index ↑	idregis	cases	continentexp	countriesandterritories	countryterritorycode	cumulative_number_for_14_da...	daterep	day	deaths	geoid	month	popdata2019	year
0	2	298	Asia	Afghanistan	AFG	7.05277624	13/12/2020	13	9	AF	12	38041757	2020
1	1	746	Asia	Afghanistan	AFG	9.01377925	14/12/2020	14	6	AF	12	38041757	2020