

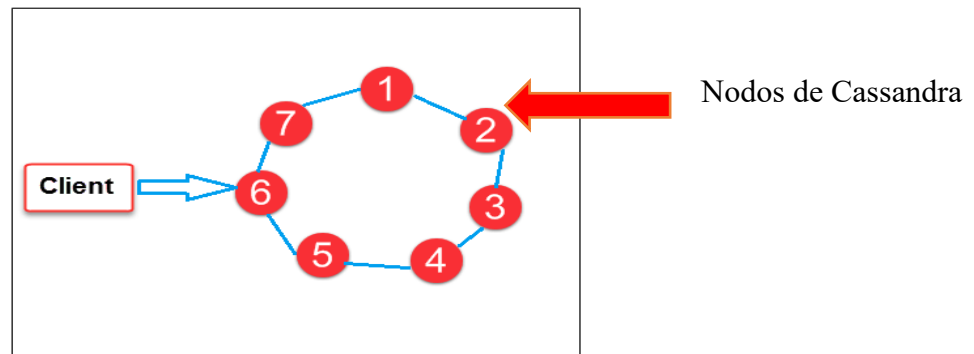
ALMACENAMIENTO Y RECUPERACIÓN DE LA INFORMACIÓN -ST1800
BITÁCORA CASSANDRA

Felipe Córtes Jaramillo
Luis Javier Palacio Mesa
María Camila García García
Alicia María Aguirre Lopera
Jamerson Stive Correa Correa

UNIVERSIDAD EAFIT
Edwin Montoya
Escuela de Ingeniería
Departamento de Informática y Sistemas
Maestría en Ciencias de los Datos y Analítica
2021-1

CASSANDRA

Cassandra es un sistema de administración de bases de datos, que se desarrolló por primera vez en Julio de 2008; para la búsqueda en bandejas de entrada para Facebook y ahora es un proyecto de alto nivel de Apache desde febrero de 2010. El cual, busca manejar un gran volumen de datos estructurados en servidores de productos básicos y colocarlos en diferentes máquinas con más de un factor de replicación que proporciona alta disponibilidad y ningún punto único de falla.



Además, según Guru 99 cumple con funciones como:

- **“Arquitectura masivamente escalable:** Cassandra tiene un diseño sin maestro en el que todos los nodos están en el mismo nivel, lo que proporciona simplicidad operativa y fácil escalabilidad .
- **Arquitectura sin maestro:** los datos se pueden escribir y leer en cualquier nodo.
- **Rendimiento de escala lineal:** a medida que se agregan más nodos, aumenta el rendimiento de Cassandra.
- **Sin punto único de falla:** Cassandra réplica datos en diferentes nodos que asegura que no haya un solo punto de falla.
- **Detección y recuperación de fallas:** los nodos fallados se pueden restaurar y recuperar fácilmente.
- **Modelo de datos flexible y dinámico:** admite tipos de datos con escrituras y lecturas rápidas.
- **Protección de datos:** los datos están protegidos con un diseño de registro de confirmación y seguridad incorporada, como mecanismos de copia de seguridad y restauración.
- **Coherencia de datos ajustables:** compatibilidad con una sólida coherencia de datos en toda la arquitectura distribuida.
- **Replicación de varios centros de datos:** Cassandra proporciona una función para replicar datos en varios centros de datos.
- **Compresión de datos:** Cassandra puede comprimir hasta un 80% de los datos sin gastos generales.
- **Lenguaje de consulta de Cassandra:** Cassandra proporciona un lenguaje de consulta similar al lenguaje SQL. Hace muy fácil para los desarrolladores de bases de datos relacionales pasar de una base de datos relacional a Cassandra” [1]

INSTALACIÓN

Para lograr una instalación exitosa de Cassandra en el equipo, es necesario cumplir algunos requisitos fundamentales como descargar Java versión JDK 1.8.0_251 para Windows o Mac y tener Python 2.7. Para el caso de Java, se hace a través de Oracle JDK 8 (Java Development Kit), así:

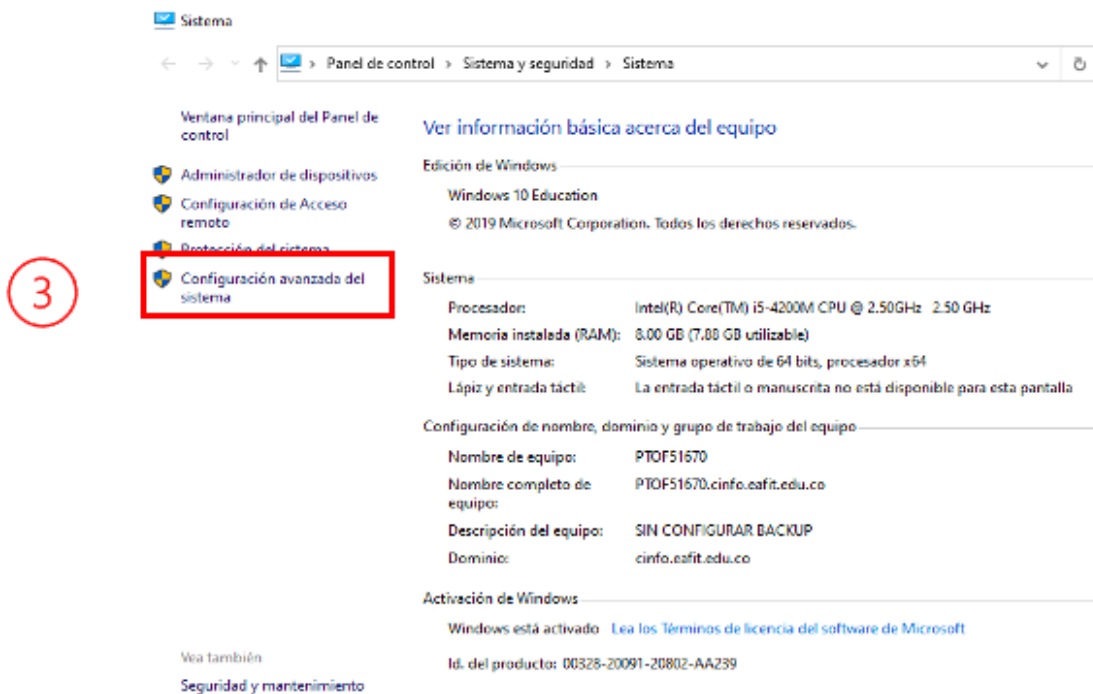
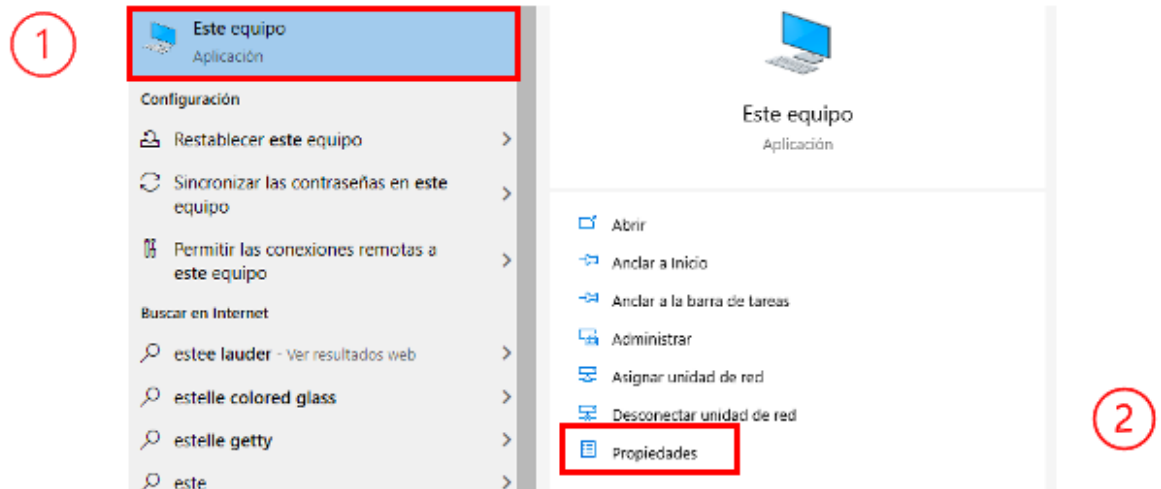
1. Descargar la versión de Java JDK 1.8.0_251

Solaris x64 (SVR4 package)	133.64 MB	jdk-8u251-solaris-x64.tar.Z
Solaris x64	919 MB	jdk-8u251-solaris-x64.tar.gz
Windows x86	201.77 MB	jdk-8u251-windows-i586.exe
Windows x64	211.54 MB	jdk-8u251-windows-x64.exe

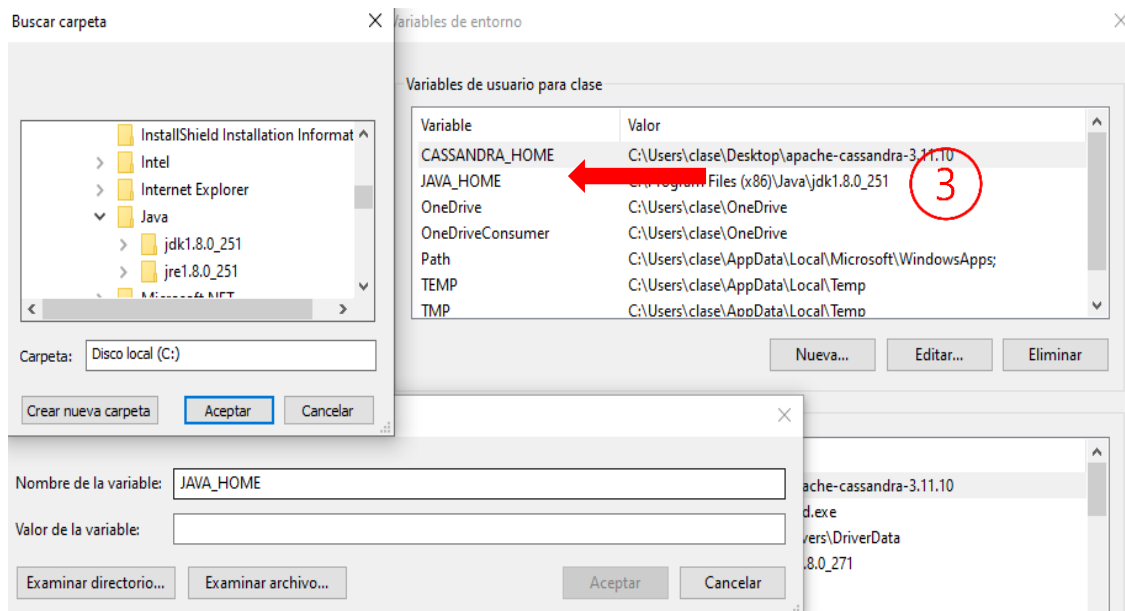
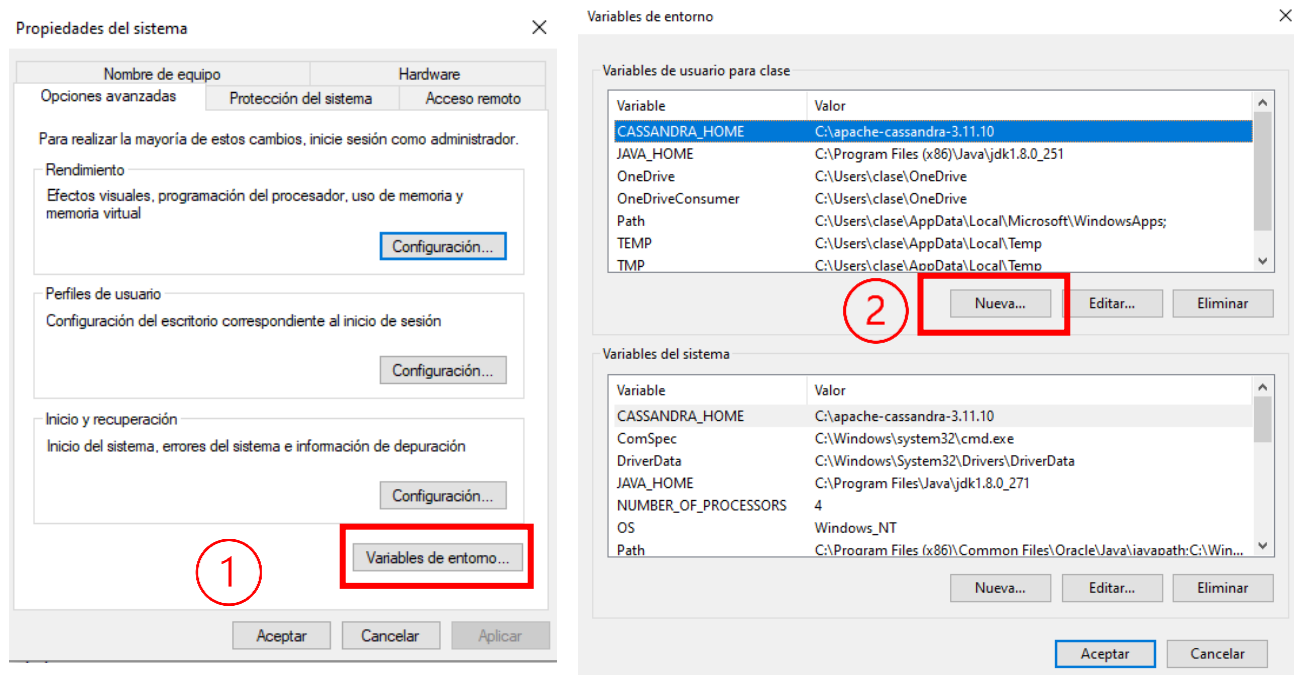
Cuando se realice la correcta instalación debe lucir de la siguiente manera:



2. Una vez descargado Java, iniciamos la configuración de las variables de ambiente para Java 8, a continuación se muestra el paso a paso a seguir:



3. Se inserta JAVA_HOME como nueva variable:



Python

Después, para instalar y configurar Python 2.7.18, se necesita ir a su página oficial y seleccionar la versión de Windows x64.


1. La página debe de aparecer de la siguiente manera y cuando se descarga exitosamente el programa, luce así:

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		38c84292658ed4456157195f1c9bcbe1	17539408	SIG
XZ compressed source tarball	Source release		fd6cc8ec0a78c44036f825e739f36e5a	12854736	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	ce98eeb7bdf806685adc265ec1444463	24889285	SIG
Windows debug information files	Windows		20b111ccfe8d06d2fe8c77679a86113d	25178278	SIG
Windows debug information files for 64-bit binaries	Windows		bb0897ea20fda343e5179d413d4a4a7c	26005670	SIG
Windows help file	Windows		b3b753dffe1c7930243c1c40ec3a72b1	6322188	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64	a425c758d38f8e28b56f4724b499239a	20598784	SIG
Windows x86 MSI installer	Windows		db6ad9195b3086c6b4cefb9493d738d2	19632128	SIG

Python 2.7.18 (64-bit) Setup

Select whether to install Python 2.7.18 (64-bit) for all users of this computer.

☒ Install for all users 

☐ Install just for me (not available on Windows Vista)

python for windows

Back Next > Cancel


Python 2.7.18 (64-bit) Setup

Customize Python 2.7.18 (64-bit)

Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.

Python

- Register Extensions
- Tcl/Tk
- Documentation
- Utility Scripts
- pip
- Test suite
- ☒ Add python.exe to Path

Python Interpreter and Libraries 

This feature requires 30MB on your hard drive. It has 6 of 7 subfeatures selected. The subfeatures require 32MB on your hard drive.

Disk Usage Advanced < Back Next > Cancel

Python 2.7.18 (64-bit) Setup

Complete the Python 2.7.18 (64-bit) Installer

Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

python for windows

Click the Finish button to exit the Installer.

< Back Finish Cancel

2. Finalmente, se deben editar las variables de entorno para Python:

1

2

3

4

Propiedades del sistema

Nombre de equipo Hardware

Opciones avanzadas Protección del sistema Acceso remoto

Para realizar la mayoría de estos cambios, inicie sesión como administrador.

Rendimiento
Efectos visuales, programación del procesador, uso de memoria y memoria virtual
Configuración...

Perfiles de usuario
Configuración del escritorio correspondiente al inicio de sesión
Configuración...

Inicio y recuperación
Inicio del sistema, errores del sistema e información de depuración
Configuración...

Variables de entorno...

Aceptar Cancelar Aplicar

Variables de usuario para clase

Variable	Valor
CASSANDRA_HOME	C:\Users\clase\Desktop\apache-cassandra-3.11.10
JAVA_HOME	C:\Program Files (x86)\Java\jdk1.8.0_251
OneDrive	C:\Users\clase\OneDrive
OneDriveConsumer	C:\Users\clase\OneDrive
Path	C:\Users\clase\AppData\Local\Microsoft\WindowsApps;
TEMP	C:\Users\clase\AppData\Local\Temp
TMP	C:\Users\clase\AppData\Local\Temp

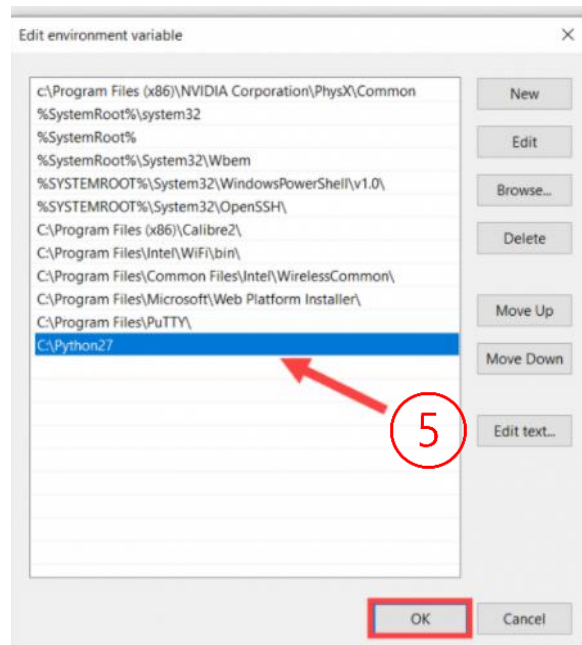
Nueva... Editar... Eliminar

Variables del sistema

Variable	Valor
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_271
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

Nueva... Editar... Eliminar

Aceptar Cancelar



Cassandra

Para descargar e instalar Cassandra, se necesita visitar la página oficial de Apache Cassandra y seleccionar la versión que se requiera, para nuestro ejercicio elegimos la versión Beta:

Apache CASSANDRA™ Home Download

Downloading Cassandra

Latest Beta Version
Download the latest Apache Cassandra 4.0 beta release: [4.0-beta4](#) (pgp, sha256 and sha512), released on 2020-12-30.

Latest Stable Version
Download the latest Apache Cassandra 3.11 release: [3.11.10](#) (pgp, sha256 and sha512), released on 2020-11-04.

1. Nos redirecciona a la página de Apache y damos enter para descargarla:

THE APACHE SOFTWARE FOUNDATION 20TH ANNIVERSARY

COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"

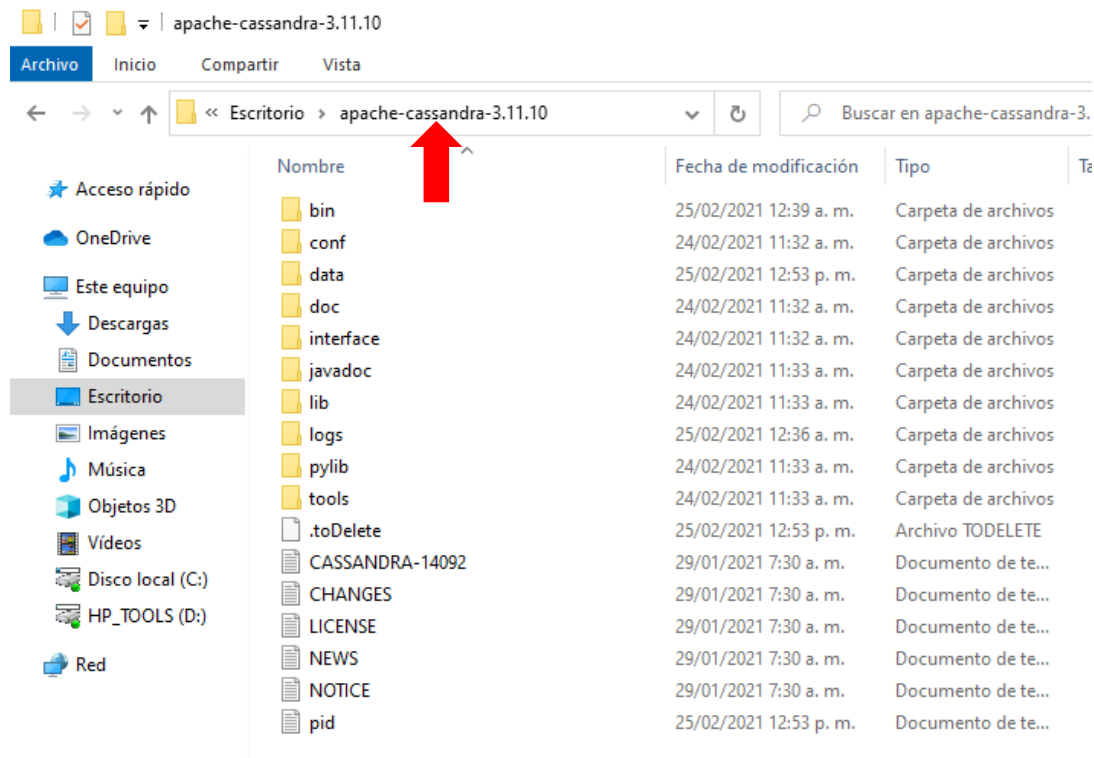
Projects ▾ People ▾ Community ▾ License ▾ Sponsors ▾

<https://downloads.apache.org/cassandra/3.11.10/apache-cassandra-3.11.10-bin.tar.gz>

Other mirror sites are suggested below.

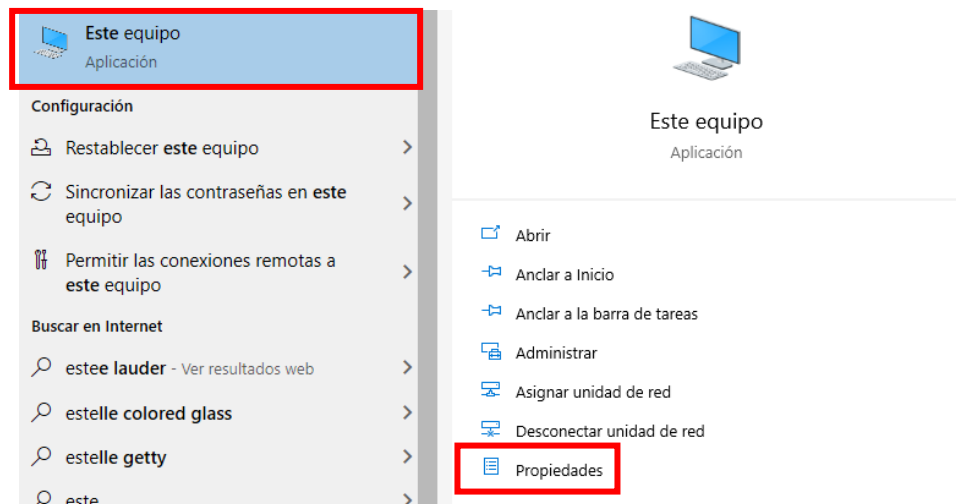
It is essential that you verify the integrity of the downloaded file using the PGP signature (.asc file) or a hash (.md5 or .sha* file). Please only use the backup mirrors to download KEYS, PGP signatures and hashes (SHA* etc) -- or if no other mirrors are working.

- Descomprimos la carpeta de Cassandra y nos deben de aparecer las siguientes carpetas:



- Finalmente, configuramos las variables de entorno para Cassandra y estaremos listos para programar:

1



Sistema

Panel de control > Sistema y seguridad > Sistema

Ventana principal del Panel de control

- Administrador de dispositivos
- Configuración de Acceso remoto
- Protección del sistema
- Configuración avanzada del sistema**

Ver información básica acerca del equipo

Edición de Windows

Windows 10 Education
© 2019 Microsoft Corporation. Todos los derechos reservados.

Sistema

Procesador: Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz 2.50 GHz
Memoria instalada (RAM): 8.00 GB (7.88 GB utilizable)
Tipo de sistema: Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil: La entrada táctil o manuscrita no está disponible para esta pantalla

Configuración de nombre, dominio y grupo de trabajo del equipo

Nombre de equipo: PTOF51670
Nombre completo de equipo: PTOF51670.cinfo.eafit.edu.co
Descripción del equipo: SIN CONFIGURAR BACKUP
Dominio: cinfo.eafit.edu.co

Activación de Windows

Windows está activado [Lea los Términos de licencia del software de Microsoft](#)

Id. del producto: 00328-20091-20802-AA239

Vea también
Seguridad y mantenimiento

Variables de entorno

Variables de usuario para clase

Variable	Valor
CASSANDRA_HOME	C:\Users\clase\Desktop\apache-cassandra-3.11.10
JAVA_HOME	C:\Program Files (x86)\Java\jdk1.8.0_251
OneDrive	C:\Users\clase\OneDrive
OneDriveConsumer	C:\Users\clase\OneDrive
Path	C:\Users\clase\AppData\Local\Microsoft\WindowsApps;
TEMP	C:\Users\clase\AppData\Local\Temp
TMP	C:\Users\clase\AppData\Local\Temp

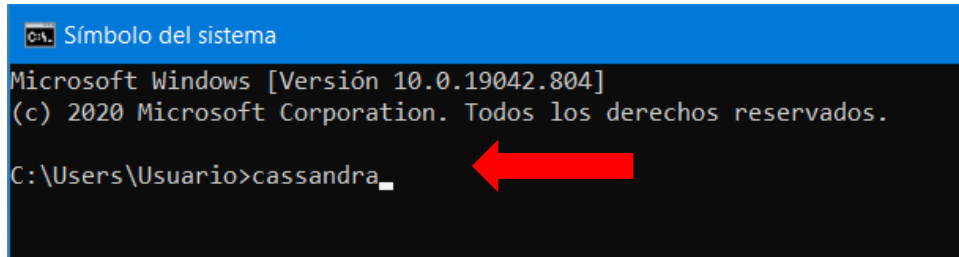
Nueva... Editar... Eliminar

Variables del sistema

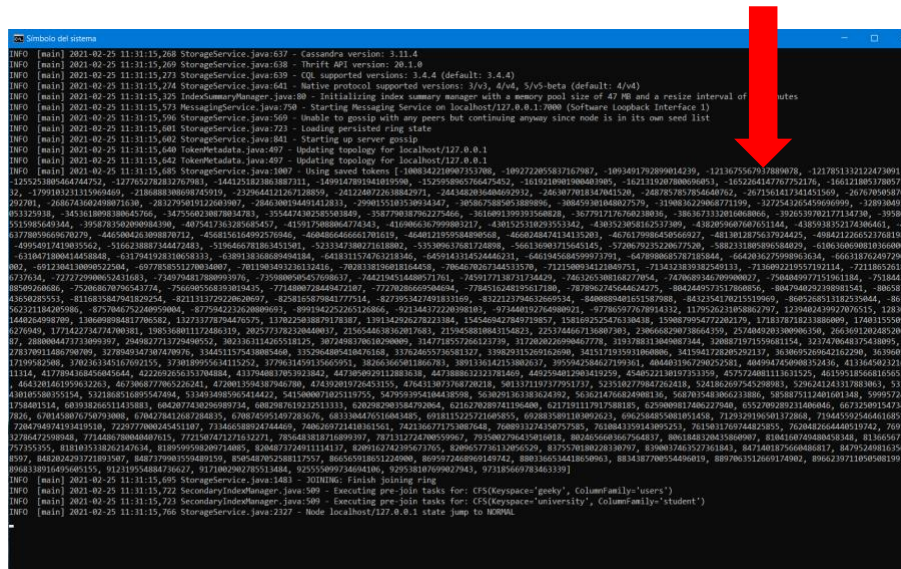
CONFIRMACIÓN

Después de tener listos los programas de Python, Java y Cassandra, se debe de confirmar que estén corriendo de forma correcta. Esto, se hace a través de 2 consolas de cmd abiertas al tiempo, una que es Cassandra como el (servicio) y otra que es CQLSH como el (cliente):

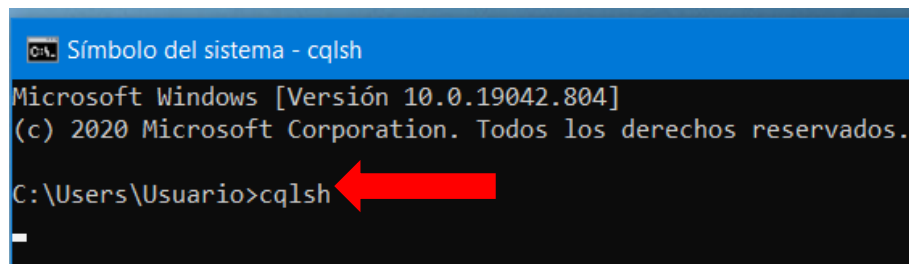
1. Se debe de llamar al lenguaje, digitando Cassandra en el cmd así:



- Una vez se digite Cassandra en la consola cmd y se hace enter, el programa debe de aparecer de esta forma indicando que no tenemos ningún error de instalación:



3. Asimismo, se debe de abrir otro cmd digitando “cqlsh” que es el cliente al cual opera Cassandra:



4. Una vez se digite “cqlsh” en la segunda consola de cmd y se haga enter, debe de aparecer así, indicando que todo está procesando de forma correcta:

```
Símbolo del sistema - cqlsh
Microsoft Windows [Versión 10.0.19042.804]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

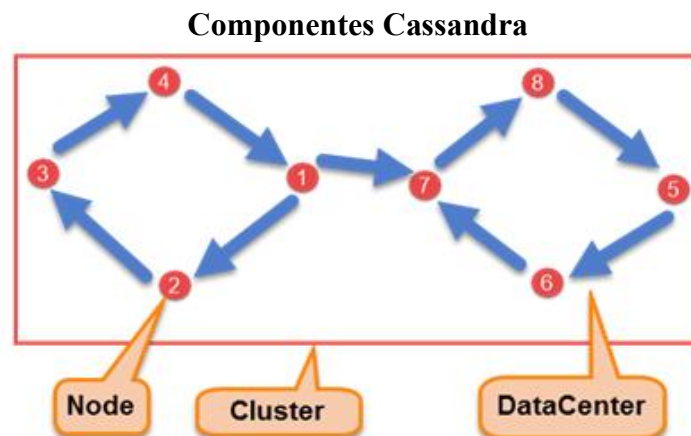
C:\Users\Usuario>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
```

ARQUITECTURA DE CASANDRA Y ESTRATEGIA DE FACTOR DE REPLICACIÓN

La característica principal de Cassandra es guardar data en múltiples nodos sin punto singular de fallo. Esto es así porque un error en el hardware puede ocurrir en cualquier momento.



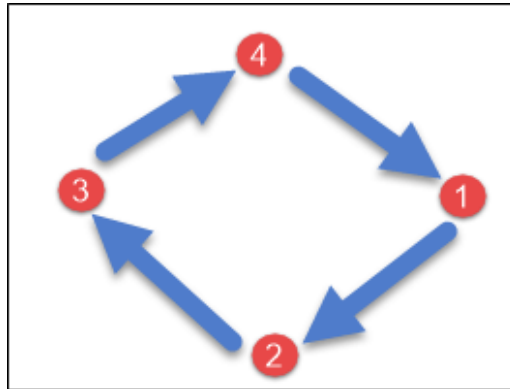
- ✓ **Nodo:** Lugar donde la data se guarda. Componente básico de Cassandra
- ✓ **Data Center:** Colección de nodos
- ✓ **Cluster:** Colección de varios Data Center
- ✓ **Commit Log:** Cualquier operación escrita se escribe para Commit Log, y se usa para la recuperación de data en caso de accidentes
- ✓ **Mem-table:** Luego de que la data se escribe en Commit Log, se escribe temporalmente en Mem-table
- ✓ **SSTable:** Cuando de Mem-table alcanza cierto umbral, la data se va a un archivo SSTable

Replicación de la data

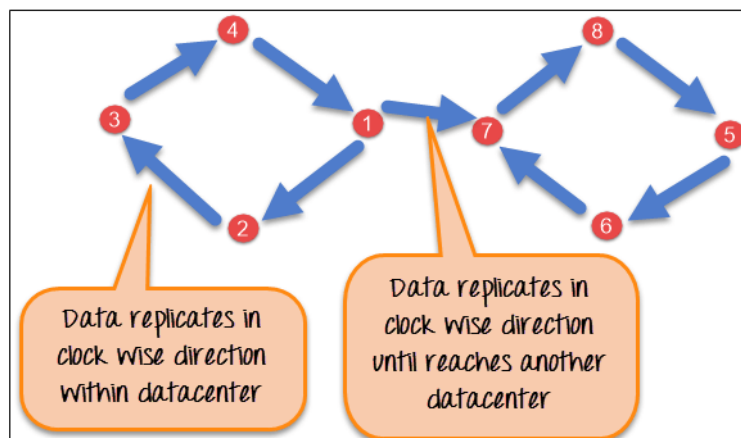
Se da porque en algún momento puede suceder algún problema. Replication factor significa que hay una copia de la data, en Cassandra se usan 3 replications factor para asegurar que la data no se pierda, por lo cual habría 3 copias de la data en 3 diferentes nodos.

Estrategias de réplica Cassandra

Estrategia simple: Sólo se tiene 1 data center. Se pone la primera réplica en el nodo seleccionado y las réplicas restantes se colocan en los nodos restantes en el sentido de las manecillas del reloj.



Estrategia Network Topology: Se usa cuando se tiene más de 2 data centers. Las réplicas van para cada data center por separado. Las réplicas se ubican en la dirección de las manecillas del reloj hasta que llegue al primer nodo del otro data center.



OPERACIÓN DE ESCRITURA

Se envía una solicitud de escritura a las réplicas, y si estas están activas, recibirán dicha solicitud. El nivel de consistencia determina cuántos nodos responden exitosamente a esta petición.

Por ejemplo, en un solo centro de datos con un factor de replicación igual a tres, tres réplicas recibirán una solicitud de escritura. Si el nivel de coherencia es uno, solo una réplica responderá con el reconocimiento de éxito y las dos restantes permanecerán inactivas. A continuación, se explican los pasos detallados del proceso de escritura:

- Cuando la solicitud de escritura llega al nodo, se registra en el commit log.
- Luego, Cassandra escribe los datos en la mem-table. Los datos escritos en la mem-table en cada solicitud de escritura también se escriben en el registro de confirmación por separado. Mem-table es un dato almacenado temporalmente en la memoria, mientras que Commit log registra los registros de transacciones con fines de respaldo.
- Cuando el mem-table esté lleno, los datos se vacían en el archivo de datos de SSTable.

LENGUAJE DE PROGRAMACIÓN

KeySpaces

Los keyspaces en casandra son objetos que contienen familias de columnas, y tipos de datos definidos por el usuario. Para hacer una analogía, sería los databases si estuviéramos hablando de un PostgreSQL.

De la siguiente forma creamos un keyspace en Cassandra:

```
Create keyspace KeyspaceName with replication={'class':strategy name,
                                             'replication_factor': No of replications on different nodes};
```

En este sentido vamos a abordar cada uno de los parámetros para crear un keyspace.


Strategy: En el parámetro class, definimos la estrategia con la cual se va a crear el keyspace. Puede ser Simple Strategy, el cual se maneja solo un data center, en este sentido se coloca la primera partición para colocar los datos y el resto se colocan siguiendo las manecillas del reloj.

La otra estrategia es Network Topology Strategy, la cual se usa cuando se poseen más de un data center, en esta estrategia se deben proveer estrategias de replicación para cada data center.

Replication Factor: En este parámetro se colocan el número de réplicas que se van a colocar en los diversos nodos. Para que haya resiliencia, se recomiendan 3 replications. Todo para prevenir que, si se cae un nodo, los datos no se pierdan.


Este es un ejemplo aplicado en una de las instancias EC2 creadas para este laboratorio:

```
cqlsh> Create keyspace University with replication={'class':'SimpleStrategy','replication_factor':3};  
cqlsh> █
```



Aquí se pueden observar varias cosas. Primero se creó un keyspace llamado universidad, el cual contiene la estrategia simple para la replicación, es decir, las particiones se van a realizar en sentidos de las manecillas del reloj. Además, se replicará 3 veces, por ende este será su factor de replicación.

```
cqlsh> DESCRIBE keyspaces; █
```



university	system	system_traces
system_auth	system_distributed	system_schema

```
cqlsh> █
```

Con el comando Describe, podemos observar que se creó el keyspace exitosamente. Adicionalmente, se pueden alterar estos keyspaces, se pueden modificar las estrategias y factores de replicación. Con el propósito de capacitar a los keyspaces creados a las necesidades del usuario.


Esta es la estructura del Taller:

```
Alter Keyspace KeyspaceName with replication={'class':'StrategyName',  
        'replication_factor': no of replications on different nodes}  
with DURABLE_WRITES=true/false
```

En este, el nombre no puede ser modificado. Sin embargo, los parámetros de la replicación sí pueden serlo como se mencionó anteriormente. Además, se puede modificar la variable Durable Writes, la cual se encarga de permitir la escritura de los cambios en el keyspace en el log.

Este es un ejemplo de ejecución en un EC2:

```
cqlsh> Alter keyspace University with replication={'class':'NetworkTopologyStrategy','DataCenter1':1};  
cqlsh> █
```



En este ejemplo, modificamos la estrategia de replicación para el keyspace. Cómo se modificó para un tipo de estrategia topológica de Red, es necesario especificar el número de

replicaciones por data center. De esta manera, se define 1 para la replicación en el primer data center y se confirma que existe de la siguiente manera:

```
AlreadyExists: Keyspace 'keyspacename' already exists
cqlsh> DESC KEYSPACES;

geeky          system_auth  system_distributed  keyspacename
system_schema  system        system_traces       university

cqlsh> USE univesity;
```


Por último, para eliminar un Keyspace, se puede realizar de la siguiente manera:

```
Drop keyspace KeyspaceName
```

Este comando se encargará de eliminar el keyspace y todos los datos relacionados a este. Si el keyspace no existe, saldrá un error al respecto.

Ejemplo de borrado:

```
cqlsh> Drop keyspace University;
cqlsh> Use University;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Keyspace 'university' does not exist"
cqlsh> █
```



Aquí podemos observar que se borra el keyspace y cuando intentamos usarlo, nos muestra un error. Este error nos dice que el keyspace no existe.

Tablas

El proceso de creación de una tabla en Cassandra es similar a una RDBMS, las columnas son utilizadas para almacenar datos. El comando para crear una tabla es el siguiente:

```
Create table KeyspaceName.TableName
(
  ColumnName DataType,
  ColumnName DataType,
  ColumnName DataType
  .
  .
  .
  Primary key(ColumnName)
) with PropertyName=PropertyValue;
```


En este comando se puede observar que se crea una tabla, bajo un keyspace generado, luego van las columnas y el tipo de datos de cada una. Esto, se utiliza para establecer en memoria cuánto espacio va a utilizar cada columna y en este caso optimizar los recursos utilizados.

Por último, tenemos la clave primaria, que se encarga de hacer la partición y describir cual es el valor que no se puede repetir en cada tupla de datos. Puede ser simple o compuesta, una llave simple, es solo el nombre de una columna, mientras que la llave compuesta, es una mezcla de 2 o más columnas, que particionan los datos en los nodos. En esta segunda forma, la estructura es la siguiente:

```
Primary key(Column1,Column2 . . .)
```

En esta estructura, la columna 1 es la columna de partición y la segunda es la columna de clustering. Es decir, la primera realiza la partición y la segunda define qué datos va a qué clústeres disponibles.

El siguiente es un ejemplo de la forma para crear una tabla en Cassandra, dentro del keyspace anteriormente creado:

```
cqlsh:university> CREATE TABLE University.Student(  
    ... RollNo int,  
    ... Name text,  
    ... dept text,  
    ... primary Key (RollNo));  
AlreadyExists: Table 'university.student' already exists
```

Con este comando, creamos una tabla en el keyspace university. Esta tabla se llama Student y contiene información importante acerca de un estudiante. Entre estos esta, el numero del estudiante, el nombre, el departamento de estudio y por último, se define la llave primaria Rollo, de esta manera un estudiante tiene una llave única para ser identificado en la tabla.

```
cqlsh> use University;  
cqlsh:university> describe tables;  
  
student  
  
cqlsh:university> █
```

Aquí podemos observar que, si utilizamos el Keyspace University y observamos sus tablas, encontramos la tabla anteriormente creada.

```
cqlsh:university> Describe student;

CREATE TABLE university.student (
  rollno int PRIMARY KEY,
  dept text,
  name text
) WITH bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
  AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND crc_check_chance = 1.0
  AND dclocal_read_repair_chance = 0.1
  AND default_time_to_live = 0
  AND gc_grace_seconds = 864000
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair_chance = 0.0
  AND speculative_retry = '99PERCENTILE';
```

Y aquí encontramos los datos específicos de la tabla creada.

Nota: En este caso como solo se tiene un solo nodo, solo habrá una partición que contiene los datos de la tabla Student.

El comando Alter es utilizado para agregar, quitar o modificar el nombre de alguna columna dentro de una tabla creada.

```
Alter table KeyspaceName.TableName +
Alter ColumnName TYPE ColumnDatatype |
Add ColumnName ColumnDataType |
Drop ColumnName |
Rename ColumnName To NewColumnName |
With propertyName=PropertyValue
```

Como se observa en la estructura mostrada, es necesario definir la tabla y luego el comando a utilizar en las columnas de esta.

```
cqlsh> Alter table University.Student
... Add Semester int;
cqlsh> desc University.Student;

CREATE TABLE university.student (
  rollno int PRIMARY KEY,
  dept text,
  name text,
  semester int
) WITH bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
  AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND crc_check_chance = 1.0
  AND dclocal_read_repair_chance = 0.1
  AND default_time_to_live = 0
  AND gc_grace_seconds = 864000
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair_chance = 0.0
  AND speculative_retry = '99PERCENTILE';

cqlsh>
```

En este comando agregamos una columna a la tabla y luego con las propiedades de esta, observamos la columna nueva.

Luego existen dos comandos específicos, Drop y Truncate. El primero se encarga de borrar la tabla completamente, mientras que la segunda se encarga de borrar todos los datos de la misma, pero deja la estructura de la tabla intacta. En este caso correremos el Drop, ya que la tabla no contiene datos todavía.

```
Drop Table KeyspaceName.TableName
```

Este es el comando ejecutado:

```
cqlsh> Drop table University.Student;
cqlsh> use University;
cqlsh:university> desc tables;

<empty>

cqlsh:university> █
```

Donde se puede observar que tras el comando al inspeccionar las tablas en keyspace, no se encuentra ninguna.

Manejo de los datos en Cassandra (Insert, Update, Delete) Ahora hablaremos de los comandos para poder manipular los datos dentro y fuera de la tabla. Primero está la estructura del comando para insertar datos en la tabla:

```
Insert into KeyspaceName.TableName(ColumnName1, ColumnName2, ColumnName3 . . . .)
values (Column1Value, Column2Value, Column3Value . . . .)
```

Este comando se encarga de recibir el nombre de la tabla y sus respectivas columnas, para luego colocar en cada una de ellas los valores respectivos. Este es un ejemplo en ejecución:

```
cqlsh> Create table University.Student ( RollNo int, Name text, Dept text, primary key(RollNo) );
cqlsh> Insert into University.Student(RollNo,Name,Dept) values (2,'Felipe','CS');
cqlsh> Select * from University.Student;

 rollno | dept | name
-----+-----+-----
      2 |   CS | Felipe
(1 rows)
```

Aquí podemos observar la creación de una nueva tupla dentro de la tabla de estudiantes, al revisar luego con un query simple, podemos observar que el dato resultante está en la tabla.

Para actualizar los datos se utiliza el comando de update, este comando se encarga de recibir los siguientes parámetros:

```
Update KeyspaceName.TableName
Set ColumnName1=new Column1Value,
    ColumnName2=new Column2Value,
    ColumnName3=new Column3Value,
    .
    .
    .
Where ColumnName=ColumnValue
```

Como se puede observar, además de recibir la tabla, se define las columnas a utilizar y el nuevo valor a poner. Este es el comando en ejecución.

```
cqlsh> Select * from University.Student;

rollno | dept | name
-----+-----+-----
      2 |   CS | Felipe

(1 rows)
cqlsh> Update University.Student
... Set name='Lucho'
... where rollno=2;
cqlsh> Select * from University.Student;

rollno | dept | name
-----+-----+-----
      2 |   CS | Lucho

(1 rows)
```

Aquí se puede observar el snapshot antiguo de la tabla y al actualizar el nombre del registro donde se selecciona la llave primaria, se observa el cambio en la segunda consulta.

Por último, tenemos el comando para borrar los datos, la estructura del comando es la siguiente:

```
Delete from KeyspaceName.TableName
Where ColumnName1=ColumnValue
```

De esta manera, la ejecución del mismo sobre el ejercicio anteriormente hecho es el siguiente:

```
cqlsh> select * from University.Student;

rollno | dept | name
-----+-----+-----
      2 |   CS | Lucho

(1 rows)
cqlsh> Delete from University.Student where rollno=2;
cqlsh> select * from University.Student;

rollno | dept | name
-----+-----+-----

(0 rows)
```

En este comando se selecciona el único dato que se tiene por medio de su llave primaria y a partir de esto se elimina. Al ejecutar nuevamente la tabla, se puede ver que ya no posee ningún dato.

Consulta de datos en Cassandra (Queries)

CQL o el lenguaje de queries en Cassandra, no soporta los siguientes datos:

1. Agregación como max, min, avg
2. Group By
3. Joins
4. Or
5. Uniones o intersecciones
6. Filtros sin la creación de índices
7. Más grande o menor que (<,>)

De esta manera, la forma para consultar en Cassandra se concentra en condiciones de mayormente. Esta es la taxonomía de los queries:

```
Select ColumnNames from KeyspaceName.TableName Where ColumnName1=Column1Value AND  
ColumnName2=Column2Value AND  
.  
.  
.
```

Con esto en mente se pueden filtrar datos específicos por medio de estas condiciones, este es un ejemplo de lo enunciado:

```
cqlsh> Select * from University.Student where rollno=2;  
  
rollno | dept | name  
-----+-----+-----  
2 | CS | Felipe  
  
(1 rows)  
cqlsh> Select name from University.Student where rollno=2;  
  
name  
-----  
Felipe  
  
(1 rows)  
cqlsh> █
```

En el primer ejercicio se busca todo lo relacionado a la tupla con el indicador 2. Y en el segundo se busca solo el campo nombre, donde se hace una relación con su indicador para que sea 2 igualmente.

Índices en Cassandra (Index)

Los índices crean en la columna especificada por el usuario una especie de cursos para poder iterar los elementos de la columna. Al momento de crear datos en una tabla, Cassandra crea un índice para iterar en cada columna, luego de correr el comando de create index.

El índice es necesario para filtrar los datos en la tabla, que no sea por medio de la llave primaria como se mostró anteriormente.

El comando para crear el índice es el siguiente:

```
Create index IndexName on KeyspaceName.TableName(ColumnName);
```

Donde se observa la columna de la tabla para crear el índice.

```

cqlsh> Select * from University.Student where dept='CS';
InvalidRequest: Error from server: code=2200 [Invalid query] :
redictability, use ALLOW FILTERING"
cqlsh> Create index DeptIndex on university.student (dept);
cqlsh> Select * from University.Student where dept='CS';

rollno | dept | name
-----+-----+-----
      2 |   CS | Felipe

(1 rows)

```

En este ejemplo se puede observar que al intentar buscar la información de una tabla en un campo que no es primario, salta un error. Luego se define el índice y ahora sí se puede filtrar la información.

También se pueden eliminar los índices en caso de no necesitarlos, se realiza de la siguiente forma:

```
Drop index IF EXISTS KeyspaceName.IndexName
```

Y esta es el caso práctico:

```

cqlsh> Select * from University.Student where dept='CS';

rollno | dept | name
-----+-----+-----
      2 |   CS | Felipe

(1 rows)
cqlsh> drop index IF EXISTS university.deptindex ;
cqlsh> Select * from University.Student where dept='CS';
InvalidRequest: Error from server: code=2200 [Invalid quer
redictability, use ALLOW FILTERING"

```

Finalmente, se observa que, al eliminar el índice, ya no se puede filtrar la tabla bajo ese campo. El parámetro IF EXISTS se coloca, para en caso de una falla, no retornar error.

Tipos de datos en Cassandra y la expiración de los datos usando TTL

Estos son los diversos tipos de datos usados en Cassandra:

CQL Type	Constants	Description
ascii	Strings	US-Ascii character string
Bigint	Integers	64-bit signed long
Blob	Blobs	Arbitrary bytes in hexadecimal
Boolean	Booleans	True or false
Counter	Integers	Distributed counter values 64 bit
Decimal	Integers, floats	Variable precision decimal
Double	Integers, floats	64-bit floating point
Float	Integers, floats	32-bit floating point
Frozen	Tuples, collections, user defined types	Stores cassandra types
Inet	Strings	IP address in IPV4 or IPV6 format
Int	Integers	32 bit signed integer
List		Collection of elements
Map		Json style collection of elements
Set		Collection of elements
Text	Strings	UTF-8 encoded strings
Timestamp	Integers, strings	Id generated with date plus time
Timeuuid	Uuids	Type 1 uuid
Tuple		A group of 2,3 fields
Uuid	Uuids	Standard uuid
Varchar	Strings	UTF-8 encoded string
Varint	Integers	Arbitrary precision integer

Esta lista es útil al momento de saber qué dato es el más adecuado al momento de representar la información que necesitamos.

Cassandra contiene una funcionalidad llamada TTL, la cual establece un tiempo de longevidad del dato. Después de que el tiempo pasa, el dato es eliminado.

La estructura para ingresar un dato TTL, es el siguiente:

```
Insert into KeyspaceName.TableName(ColumnNames) values(ColumnValues)
using ttl TimeInseconds;
```

Tiene la misma estructura de un insert normal, pero se define el tiempo en segundos de vida. Este es el escenario efectuado:

```
cqlsh> insert into university.student (rollno,dept,name) values (4,'ME','Maria') using ttl 30;
```

Con este comando se creó una tupla nueva donde después de 30 segundos será eliminada.

Colecciones en Cassandra

Las colecciones en Cassandra son utilizadas para el manejo de tareas, sirven para almacenar múltiples objetos en una sola entidad.

Las colecciones no pueden tener un espacio mayor a 64KB, entre menor es la colección, es mucho más fácil su manipulación y el rendimiento de las consultas en general. Hay tres tipos de colecciones en Cassandra

- **Set:** El set almacena un conjunto de elementos, que se retornan ordenados al buscarse. La sintaxis es la siguiente:

```
Create table University.Teacher
(
  id int,
  Name text,
  Email set<text>,
  Primary key(id)
);
```

Aquí se puede observar que se almacena un campo llamado Email, el cual es una colección de Emails, con este ejemplo se puede observar la columna:

```
cqlsh> Create table University.Teacher
... (
... id int,
... Name text,
... Email set<text>,
... Primary key(id)
... );
cqlsh> insert into university.teacher (id,Name,Email) values(1,'Edwin',{'abc@gmail.com','xyz@gmail.com'});
cqlsh> select * from university.teacher;

 id | email | name
---+-----+-----
  1 | {'abc@gmail.com', 'xyz@gmail.com'} | Edwin
(1 rows)
```

En este ejemplo se puede observar que el set se comporta como una colección de elementos, en este caso textos que representan los emails del profesor. Es importante decir que todos los elementos del set deben tener la misma estructura, es por esta razón que se define su tipo al principio.

Lista: Cuando importa el orden en los elementos, se implementa la lista. Aquí vamos a utilizar el ejemplo anterior y le vamos a añadir una lista.

```
cqlsh> alter table university.teacher add coursename list<text>;
cqlsh> INSERT INTO university.teacher(id,Name,Email,coursename) values(2,'Johnes', {'johnes@gmail.com'}, ['Data Science']);
cqlsh> select * from university.teacher where id=2;
```

id	coursename	email	name
2	['Data Science']	['johnes@gmail.com']	Johnes

En este ejemplo, creamos una columna nueva que será una lista. Esta lista cambia la estructura del set, ya que utiliza corchetes y su orden es inmutable, a diferencia del otro.

Mapas: En últimas tenemos el mapa, este crea una estructura de llave-valor, este es muy utilizado cuando se quieren guardar referencias de datos por lo que se crean llaves y valores que referencian a su pareja adecuada. En el ejemplo se crearía de la siguiente forma:

```
cqlsh> Create table University.Course
... (id int,
... prereq map<text,text>,
... primary key(id)
... );
cqlsh> insert into university.course(id,prereq) values(1,{'DataScience':'Database', 'Neural Network':'Artificial Intelligence'});
```

Estos últimos comandos crearían una nueva tabla que contiene un mapa. Como se ve, en este caso las llaves son los cursos y los valores sus respectivos prerrequisitos, de esta manera. Cuando busquemos una llave, nos mostrará su respectivo valor.

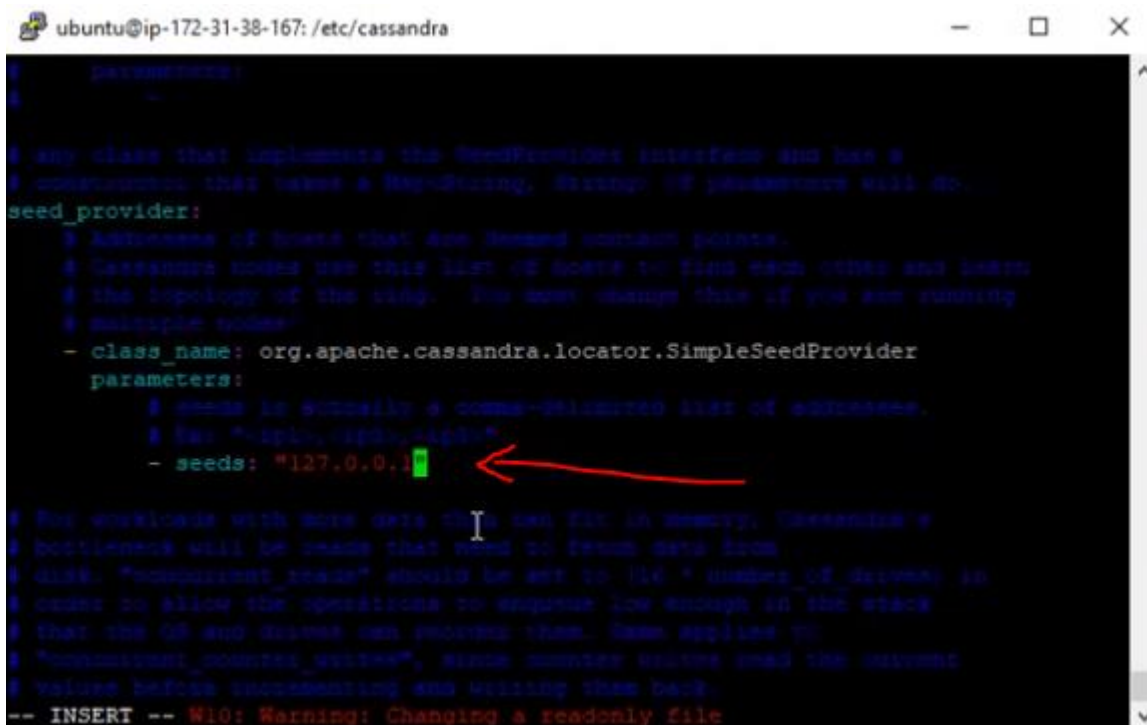
Clusterizar las aplicaciones de Cassandra

Por último, observaremos la forma de como crear una agrupación de nodos, donde se replicarán los datos y se permitirán replicar los datos en Cassandra. Esta funcionalidad es muy importante, debido a que la principal ventaja en el uso de esta tecnología, es la posibilidad de distribuir el almacenamiento de los datos y así, mitigar un único punto de falla. Por otro lado, esta replicación permite paralelizar la información, para que sea más segura al usuario en caso de modificar un nodo o se presente algún problema con ellos.

Para este ejercicio utilizamos instancias EC2 de Amazon, debido a su alta portabilidad y la facilidad de utilizar un sistema operativo Linux. Estas instancias se consumen a demanda por lo que se crearon dos para mostrar el proceso de replicación y así, no utilizar muchos recursos por parte del equipo implementador.

Empezaremos por el supuesto de tener dos instancias EC2 corriendo con Cassandra instalado. El proceso de instalación es similar al explicado al inicio del documento, por lo que en este caso se obviara. Una vez instalado Cassandra es necesario habilitar una red en la cual los nodos (instancias EC2) puedan escucharse, por lo que se procede a modificar el sistema de configuraciones ubicado en : /etc/cassandra/cassandra.yaml. En este archivo hay que modificar tres aspectos: el canal de broadcast, las semillas y el canal local.

Es necesario modificar el canal de broadcast, debido a que este es el que permite transmitir la información a los otros nodos, en este caso es el canal de salida. Por lo que se coloca la dirección IP pública de la instancia. Luego, para las semillas deben colocarse los nodos externos al nodo modificado, ya que estos serán las direcciones válidas para la escritura y lectura dentro del cluster. Finalmente, el canal local permitirá establecer el canal por el cual el nodo nos transmitirá la información en pantalla.



```
ubuntu@ip-172-31-38-167: /etc/cassandra
# any class that implements the SeedProvider interface and has a
# constructor that takes a MapString, String[] of parameters will do.
seed_provider:
  # Addresses of hosts that are deemed contact points.
  # Cassandra nodes use this list of hosts to find each other and learn
  # the topology of the ring.  You may change this if you are running
  # multiple nodes.
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      # seeds is usually a comma-delimited list of addresses.
      # For "ip1,ip2,ip3"
      - seeds: "127.0.0.1"
```

-Modificación de semillas

```
ubuntu@ip-172-31-38-167: /etc/cassandra
ssl_storage_port: 7001

# Address or interface to bind to and tell other Cassandra nodes to connect to.
# You _must_ change this if you want multiple nodes to be able to communicate!
#
# Set listen_address OR listen_interface, not both.
#
# Leaving it blank leaves it up to InetAddress.getLocalHost(). This
# will always do the Right Thing _if_ the node is properly configured
# (hostname, name resolution, etc), and the Right Thing is to use the
# address associated with the hostname (it might not be!).
#
# Setting listen_address to 0.0.0.0 is always wrong.
#
listen_address: local

# Set listen_address OR listen_interface, not both. Interfaces must correspond
# to a single address, IP aliasing is not supported.
# listen_interface: eth0

# If you choose to specify the interface by name and the interface has an ipv4 a
# nd an ipv6 address
###
-- INSERT --
```

-Modificación del canal local o IP listener.

```
ubuntu@ip-172-31-38-167: /etc/cassandra

# Set listen_address OR listen_interface, not both. Interfaces must correspond
# to a single address, IP aliasing is not supported.
# listen_interface: eth0

# If you choose to specify the interface by name and the interface has an ipv4 a
# nd an ipv6 address
# you can specify which should be chosen using listen_interface_prefer_ipv6. If
# false the first ipv6
# address will be used. If true the first ipv4 address will be used. Defaults to
# false preferring
# ipv4. If there is only one address it will be selected regardless of ipv4/ipv6
#
listen_interface_prefer_ipv6: false

# Address to broadcast to other Cassandra nodes
# Leaving this blank will set it to the same value as listen_address
broadcast_address: 3.81.3.161

# When using multiple physical network interfaces, set this
# to true to listen on broadcast address in addition to
# the listen_address, allowing nodes to communicate in both
# directions.
:wg!
```

-Modificación del canal broadcast.

Una vez modificados estos datos, los nodos están conectados. Es importante reiniciar el proceso de cassandra en ambos nodos, con el propósito de reiniciar sus configuraciones, y

así, puedan escuchar a las otras instancias. El comando sería el siguiente: *sudo systemctl restart cassandra*

Esto permitirá conectar los nodos y con todos los datos almacenados, borrados o modificados en una instancia, se replicarán en la otra. A continuación se muestran unos ejemplos de dos instancias con IP diferentes que contienen los mismos datos:

```
ubuntu@ip-172-31-38-167:~$ cqlsh
Command 'cqlsh' not found, did you mean:
  command 'clash' from snap clash (1.2.5)
  command 'clush' from deb clusershell
See 'snap info <snapname>' for additional versions.
ubuntu@ip-172-31-38-167:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> DESC KEYSPACE;
Improper DESC command.
cqlsh> DESC KEYSPACE;
Not in any keyspace.
cqlsh> SHOW keyspace;
Improper SHOW command.
cqlsh> describe keyspaces;

system_schema system_auth system system_distributed test system_traces
cqlsh>
```

```
ubuntu@ip-172-31-91-49:~$ cqlsh
at org.apache.cassandra.tools.NodeTool.execute(NodeTool.java:185)
at org.apache.cassandra.tools.NodeTool.main(NodeTool.java:56)
ubuntu@ip-172-31-91-49:/etc/cassandra$ nodetool flush system
nodetool: Failed to connect to '127.0.0.1:7199' - ConnectException: 'Connection
refused (Connection refused)'.
ubuntu@ip-172-31-91-49:/etc/cassandra$ sudo systemctl restart cassandra.service
ubuntu@ip-172-31-91-49:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> create KEYSPACE test WITH replication = ('class': 'SimpleStrategy', 'repl
ication_factor': 2);
cqlsh> exit
ubuntu@ip-172-31-91-49:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> describe keyspaces;

system_schema system_auth system system_distributed test system_traces
cqlsh>
```

En este ejemplo se crea un keyspace en la instancia de la derecha y al buscar los keyspaces disponibles en la instancia de la izquierda, podemos observar el keyspace creado: **test**.

```
ubuntu@ip-172-31-38-167:~$ cqlsh
See 'snap info <snapname>' for additional versions.
ubuntu@ip-172-31-38-167:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> DESC KEYSPACE;
Improper DESC command.
cqlsh> DESC KEYSPACE;
Not in any keyspace.
cqlsh> SHOW keyspace;
Improper SHOW command.
cqlsh> describe keyspaces;

system_schema system_auth system system_distributed test system_traces
cqlsh> create TABLE test.User (id int, name text, cel int, primary key int);
SyntaxException: line 1:64 mismatched input 'int' expecting '(' (... cel int, pr
imary key (int)...)
cqlsh> create TABLE test.User (id int, name text, cel int, primary key id);
SyntaxException: line 1:64 mismatched input 'id' expecting '(' (... cel int, pri
mary key [id]...)
cqlsh> create TABLE test.User (id int, name text, cel int, primary key(id));
cqlsh>
```

```
ubuntu@ip-172-31-91-49:~$ cqlsh
ubuntu@ip-172-31-91-49:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> create KEYSPACE test WITH replication = ('class': 'SimpleStrategy', 'repl
ication_factor': 2);
cqlsh> exit
ubuntu@ip-172-31-91-49:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> describe keyspaces;

system_schema system_auth system system_distributed test system_traces
cqlsh> use test ;
cqlsh:test> show tables;
Improper show command.
cqlsh:test> describe tables;

user
cqlsh:test>
```

En este segundo ejemplo, se crea una tabla en la instancia de la izquierda y luego se busca en la otra instancia. Es importante recalcar que en este caso el proceso de creación lo realizó la instancia contraria en comparativa con el ejercicio anterior.

```
ubuntu@ip-172-31-38-167: ~  
Not in any keyspace.  
cqlsh> SHOW keyspace;  
Improper SHOW command.  
cqlsh> describe keyspaces;  
  
system_schema system_auth system system_distributed test system_traces  
  
cqlsh> create TABLE test.User (id int, name text, cel int, primary key int);  
SyntaxException: line 1:64 mismatched input 'int' expecting '(' (... cel int, pri  
mary key [int]...)  
cqlsh> create TABLE test.User (id int, name text, cel int, primary key id);  
SyntaxException: line 1:64 mismatched input 'id' expecting '(' (... cel int, pri  
mary key [id]...)  
cqlsh> create TABLE test.User (id int, name text, cel int, primary key(id));  
cqlsh> use test ;  
cqlsh:test> select * from test.user;  
  
id | cel | name  
-----  
1 | 123456 | alejo 2  
  
(1 rows)  
cqlsh:test> INSERT INTO test.user (id, name, cel) VALUES (2, 'Maria', 234134); 3  
cqlsh:test>
```

```
ubuntu@ip-172-31-91-49: ~  
cqlsh  
Connected to Test Cluster at 127.0.0.1:9042.  
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]  
Use HELP for help.  
cqlsh> describe keyspaces;  
  
system_schema system_auth system system_distributed test system_traces  
  
cqlsh> use test ;  
cqlsh:test> show tables;  
Improper show command.  
cqlsh:test> describe tables;  
  
user 1  
  
cqlsh:test> INSERT INTO test.user (id, name, cel) values (1, 'alejo', 123456);  
cqlsh:test> select name from test.user where id = 2;  
  
name 4  
-----  
Maria  
  
(1 rows)  
cqlsh:test>
```

Finalmente se realiza la creación y búsqueda de datos en ambas instancias y se puede observar la replicación en acción, debido a que los datos en una instancia son similares a la de su instancia pareja.

Referencias

- [1] Guru99 (2018), recuperado de: <https://www.guru99.com/cassandra-tutorial.html>
- [2] Oracle (2020), recuperado de: <https://www.oracle.com/java/technologies/javase-downloads.html>
- [3] Cassandra (2018), recuperado de: <https://cassandra.apache.org/download/>