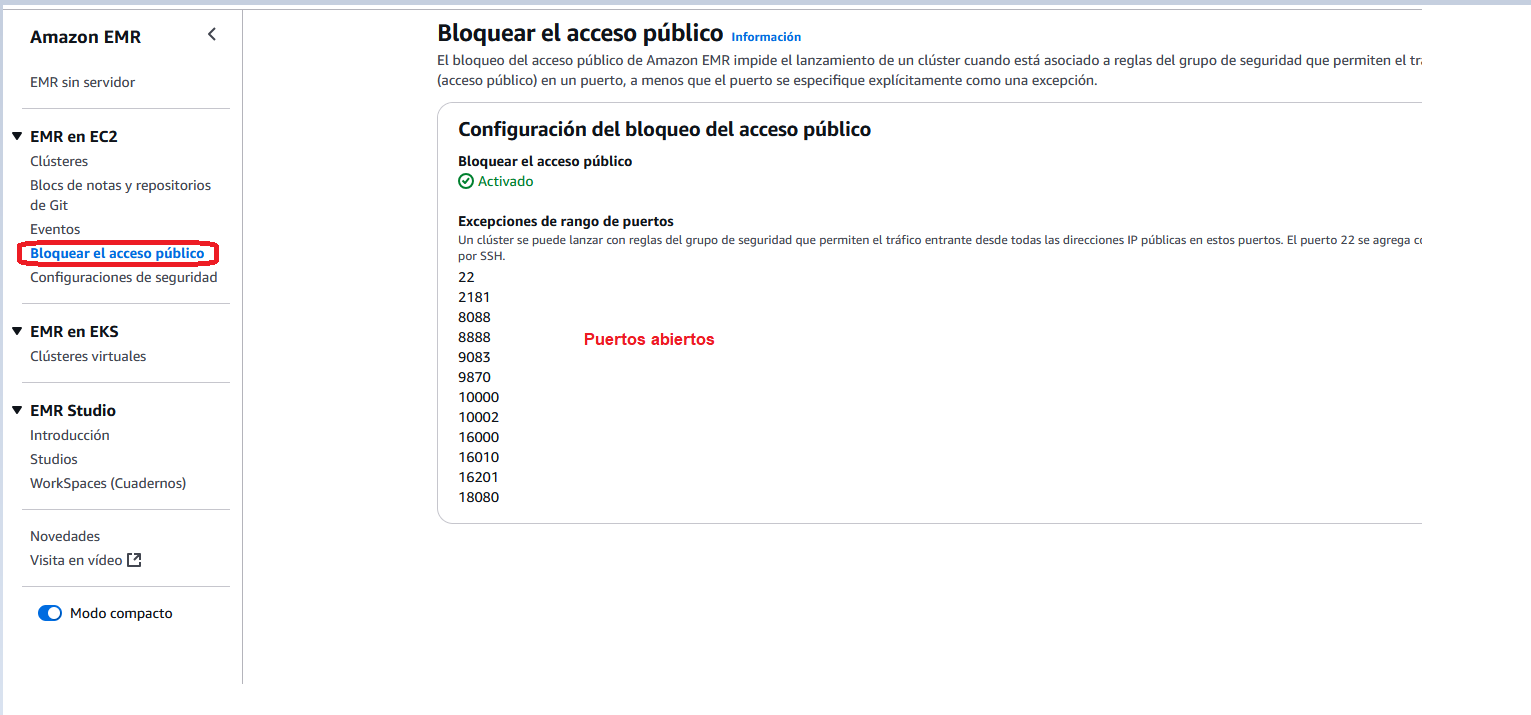
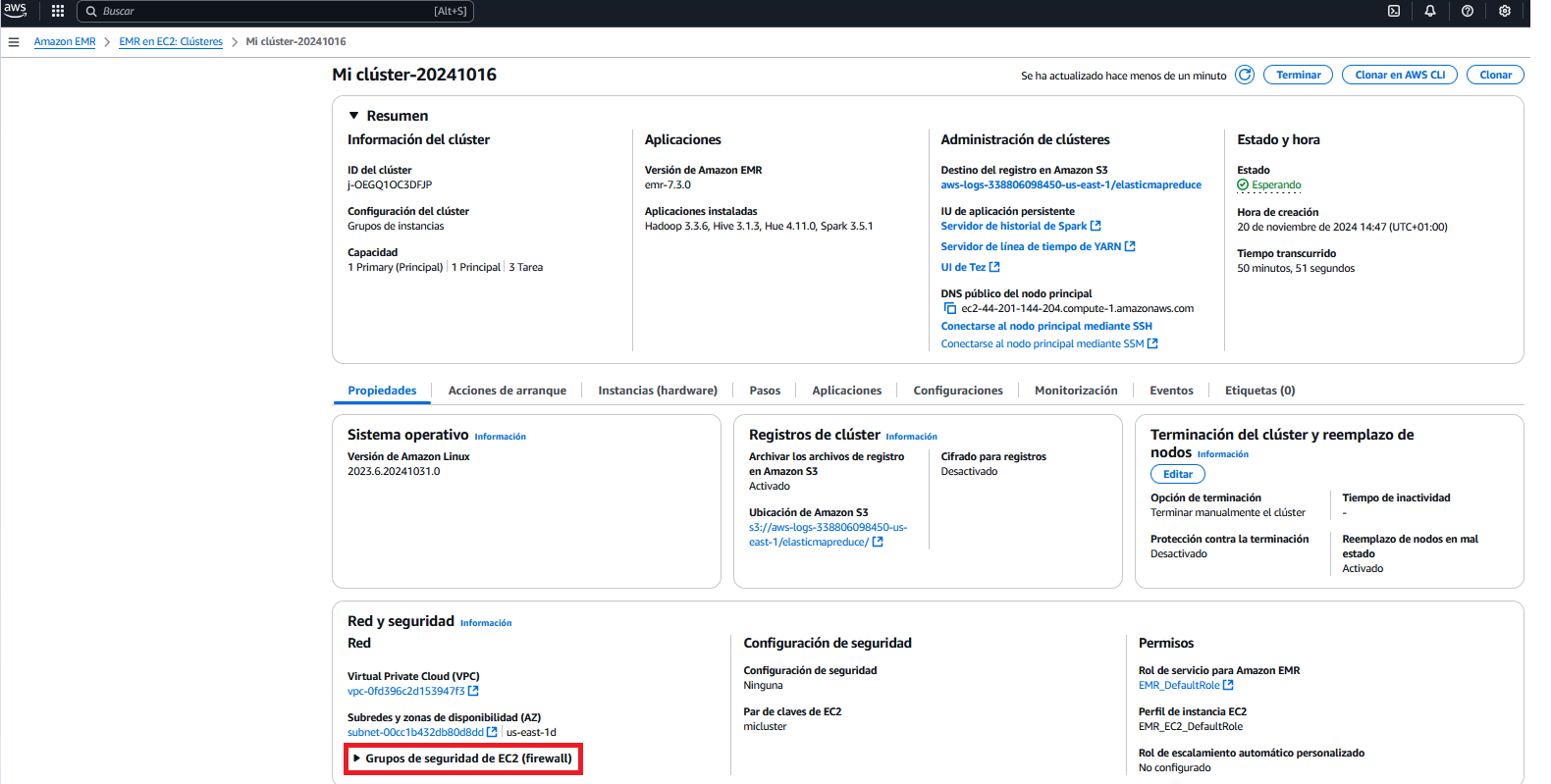
### para los ejercicios en aws

En el clúster hadoop aseguramos que tenemos acceso a los siguientes puertos de entrada:

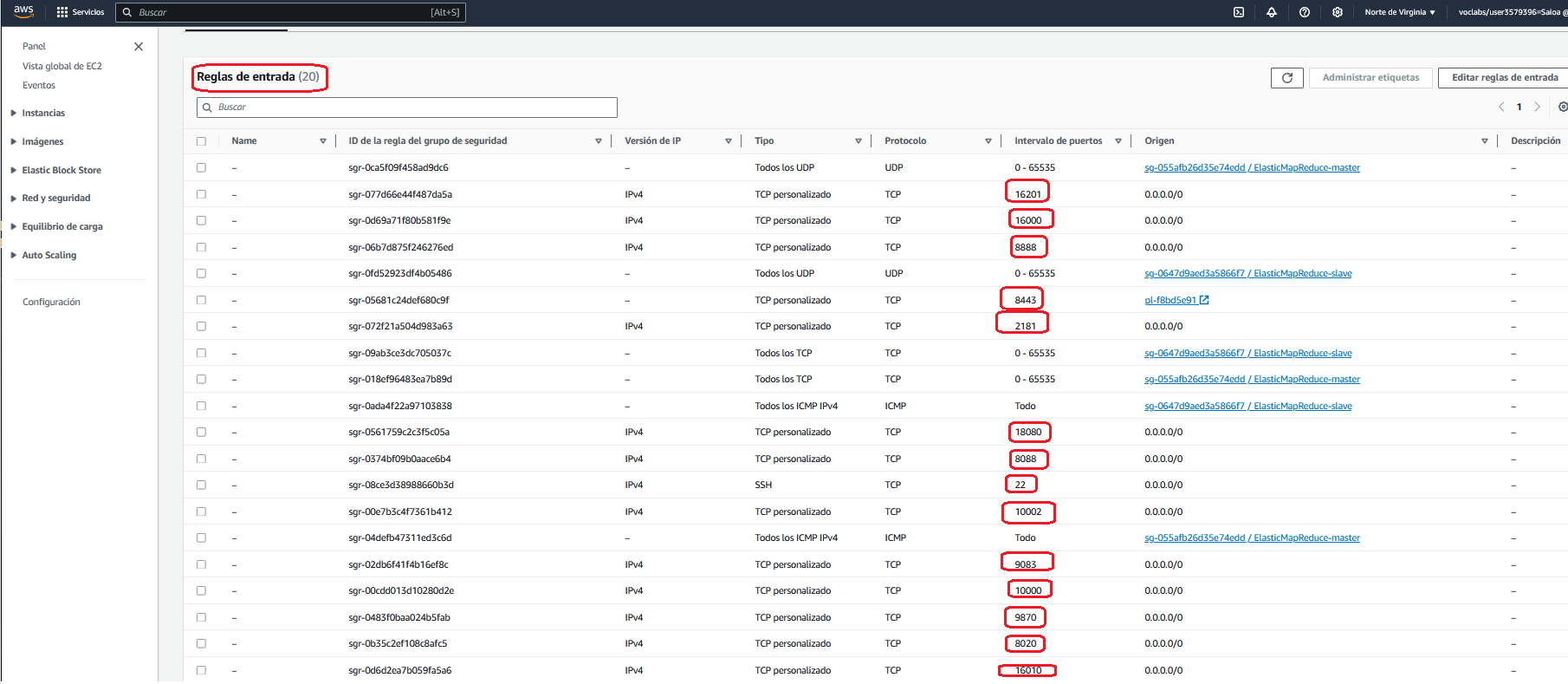


Aseguramos que los permisos de los grupos de seguridad:

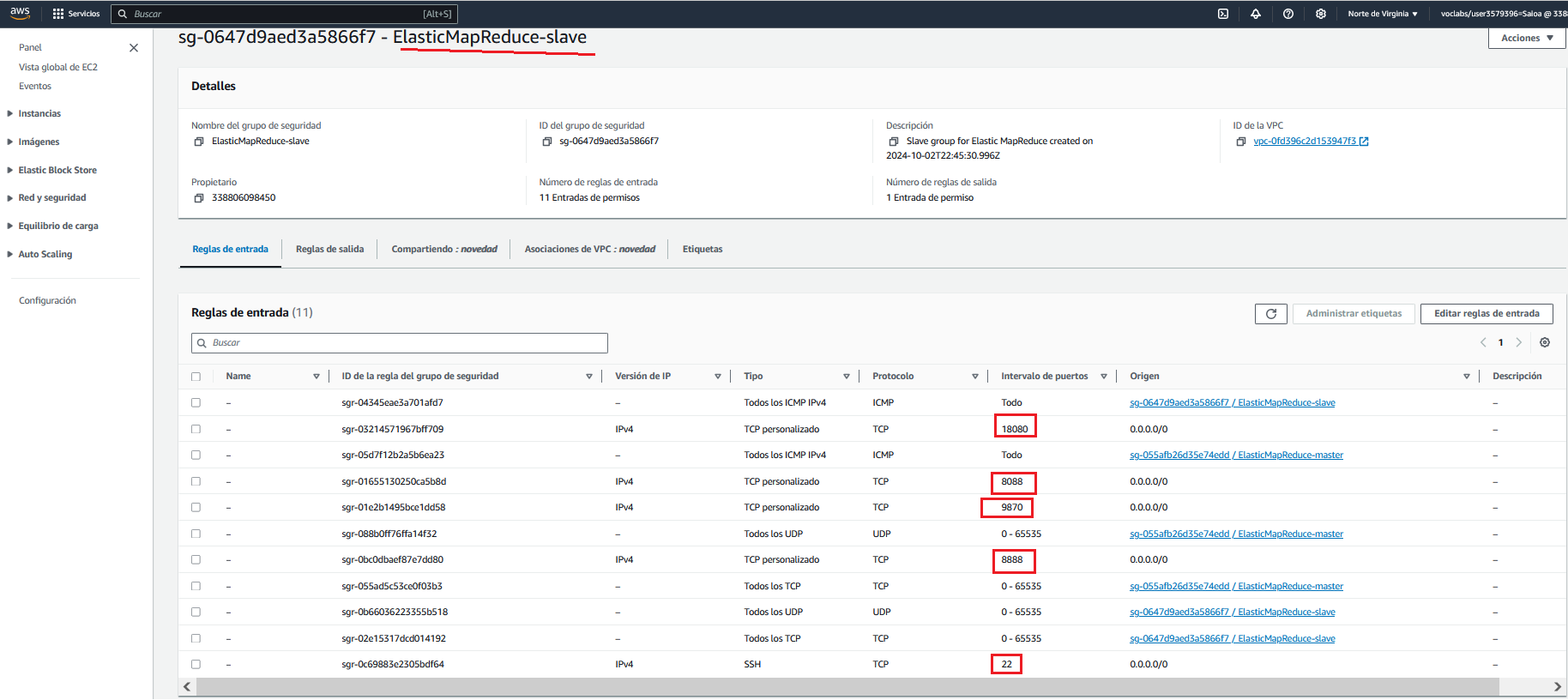




Aseguramos que tenemos definidas las siguientes reglas de entradas en el nodo principal:



En el nodo secundario aseguramos las siguientes reglas de entrada:



### EJERCICIO 1

**Hive** y **HUE** son herramientas que se utilizan junto a **Hadoop** para facilitar el manejo y análisis de grandes volúmenes de datos.

 **Hive**: Es una herramienta de almacenamiento y procesamiento de datos sobre Hadoop que permite ejecutar consultas SQL (usando una versión llamada HiveQL) sobre datos almacenados en el sistema de archivos distribuido HDFS. Hive convierte estas consultas en trabajos MapReduce o Spark, facilitando así el análisis de datos en Hadoop para usuarios que están familiarizados con SQL.

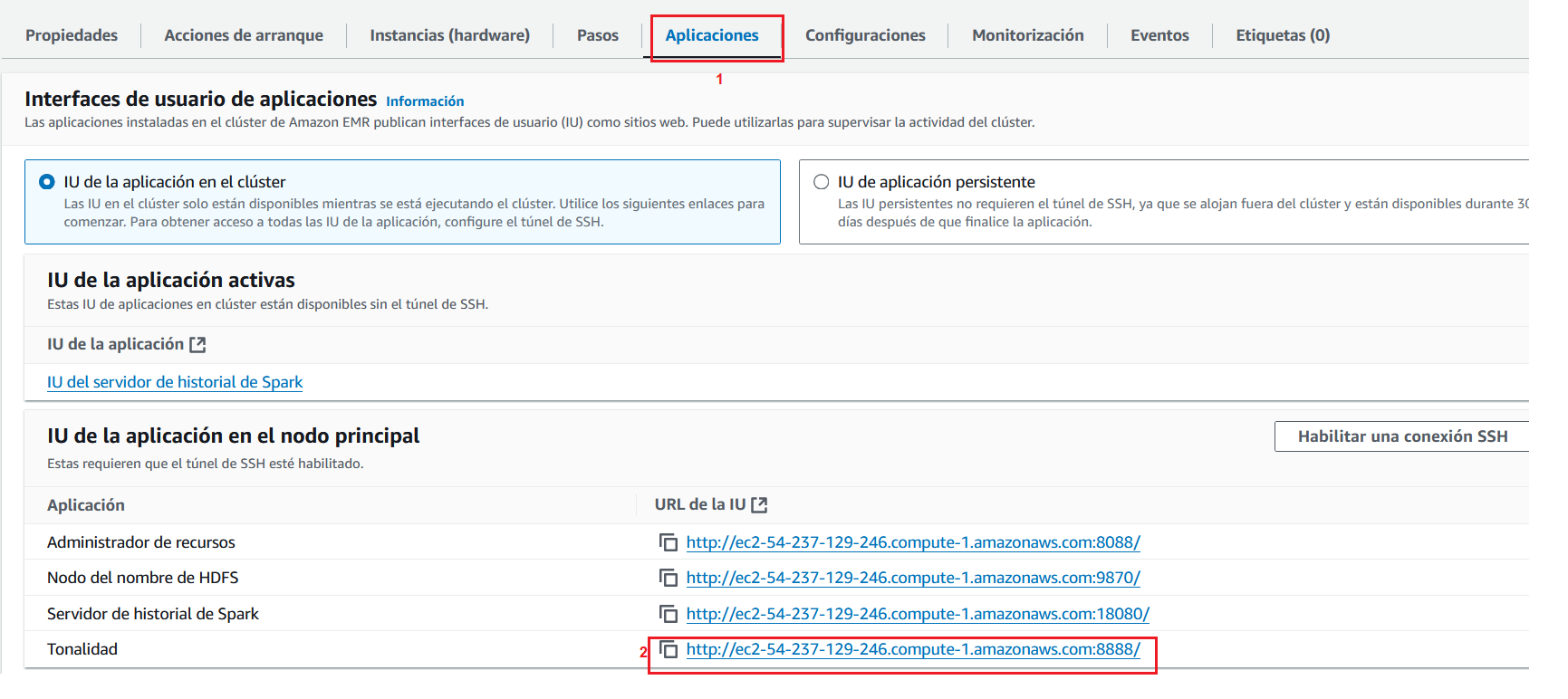
**HUE** (Hadoop User Experience): Es una interfaz gráfica de usuario para Hadoop que permite a los usuarios interactuar con componentes de Hadoop como Hive, Pig, HDFS, entre otros, de forma visual. HUE simplifica la administración y consulta de datos, permitiendo ejecutar comandos de HiveQL, cargar archivos y visualizar datos sin necesidad de usar la línea de comandos.

En este ejercicio vamos a trabajar con un clúster EMR de AWS en el que vamos a tener instaladas las siguientes aplicaciones: Hadoop, Hive y Hue y llevaremos a cabo el proceso de almacenar en una base de datos de HIVE un fichero de tipo .csv que tenemos almacenado en el sistema de archivos HDFS.

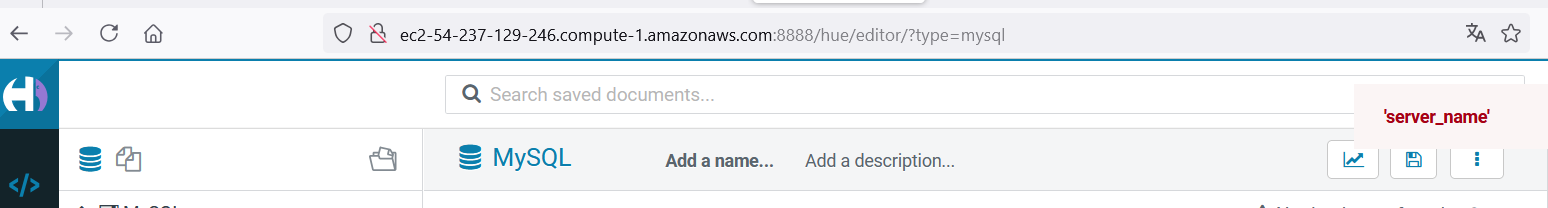
Trabajaremos con el archivo: customers-1000.csv facilitado en el enunciado del ejercicio.

Empecemos…

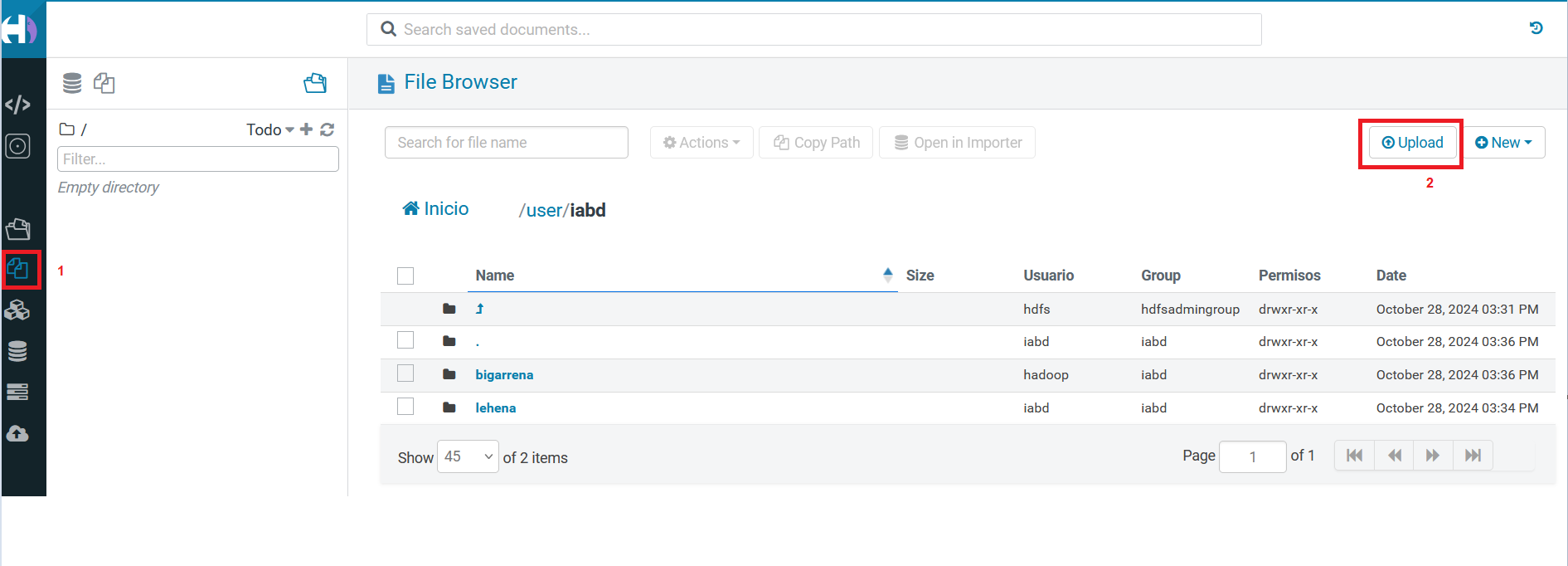
Una vez montado y arrancado el clúster nos conectamos su interfaz HUE:

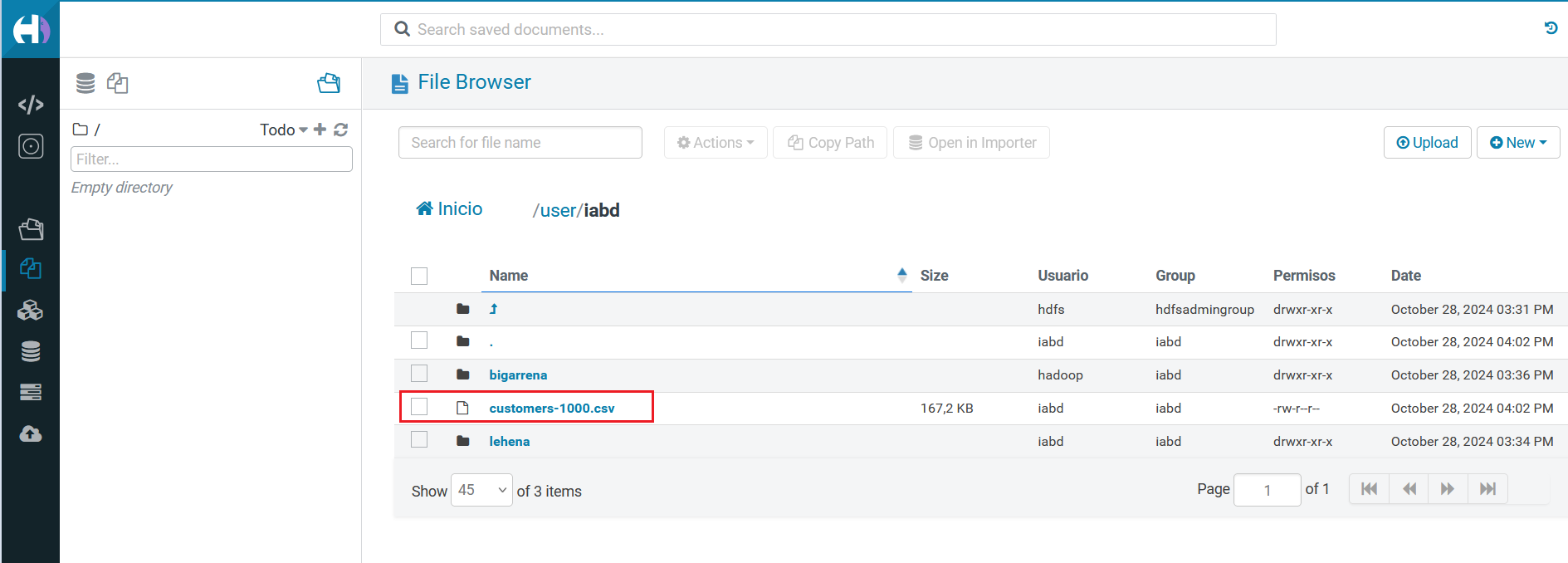


La interfaz HUE es la siguiente

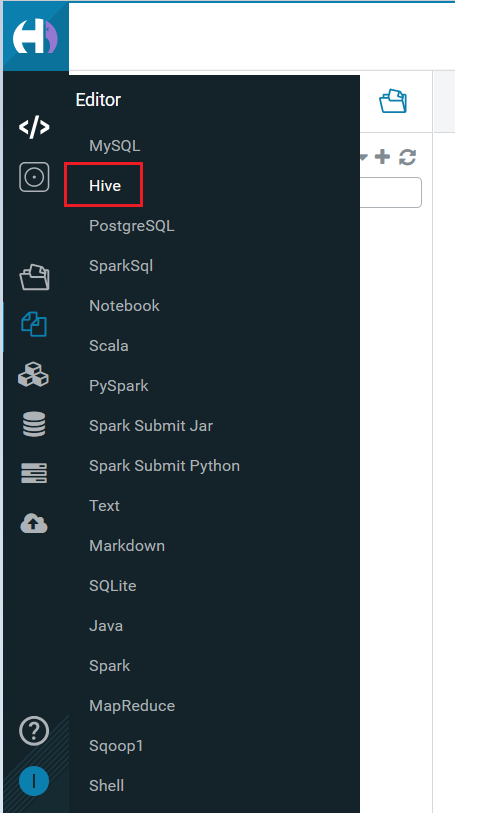


Subimos el archivo customers-1000.csv al clúster



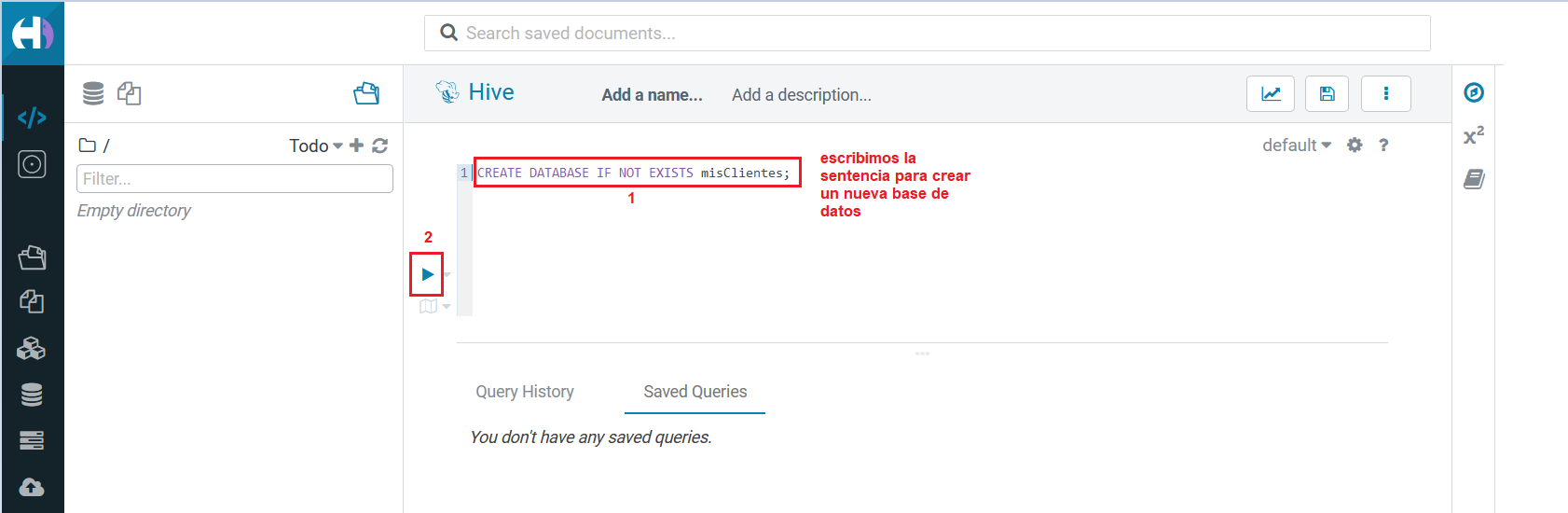


En HUE abrimos el editor de Hive Query



Ejecutamos el siguiente comando para crear una base de datos.

CREATE DATABASE IF NOT EXISTS misClientes;



Seleccionamos la base de datos para las siguientes consultas:

USE misClientes;

Creamos la tabla que se ajusta a los campos del archivo: customers-1000.csv

CREATE TABLE IF NOT EXISTS personas (

Index INT,

CustomerId STRING,

FirstName STRING,

LastName STRING,

Company STRING,

City STRING,

Country STRING,

Phone1 STRING,

Phone2 STRING,

Email STRING,

SubscriptionDate STRING,

Website STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

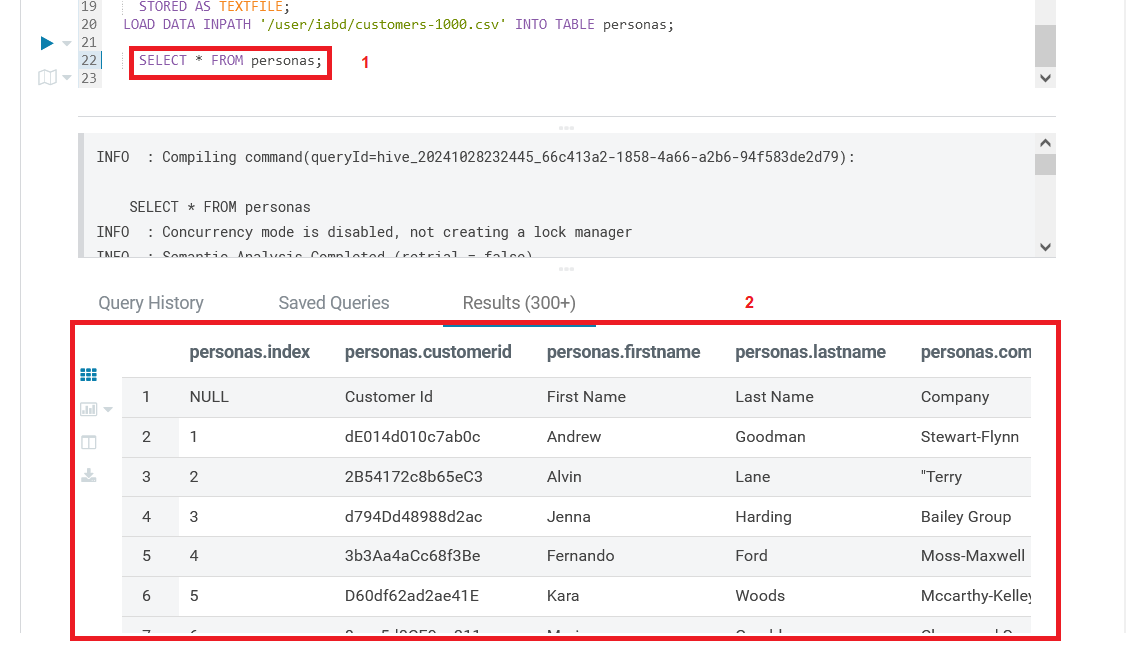
STORED AS TEXTFILE;

Cargamos los datos del CSV en la Tabla

LOAD DATA INPATH '/user/iabd/customers-1000.csv' INTO TABLE personas;

Leemos los datos de la tabla

SELECT \* FROM personas;



### EJERCICIO 2

Repetiremos los pasos del ejercicio 10 pero cargaremos los datos del archivo .csv en una base de datos HBase.

HBase es una base de datos NoSQL y orientada a columnas diseñada para almacenar grandes cantidades de datos distribuidos. Funciona sobre HDFS (el sistema de archivos de Hadoop) y permite el acceso rápido y aleatorio a datos estructurados. HBase es útil en aplicaciones que requieren lectura y escritura de grandes datos en tiempo real, como en análisis de redes sociales o sistemas de recomendación.

### EJERCICIO 3 (OPCIONAL)

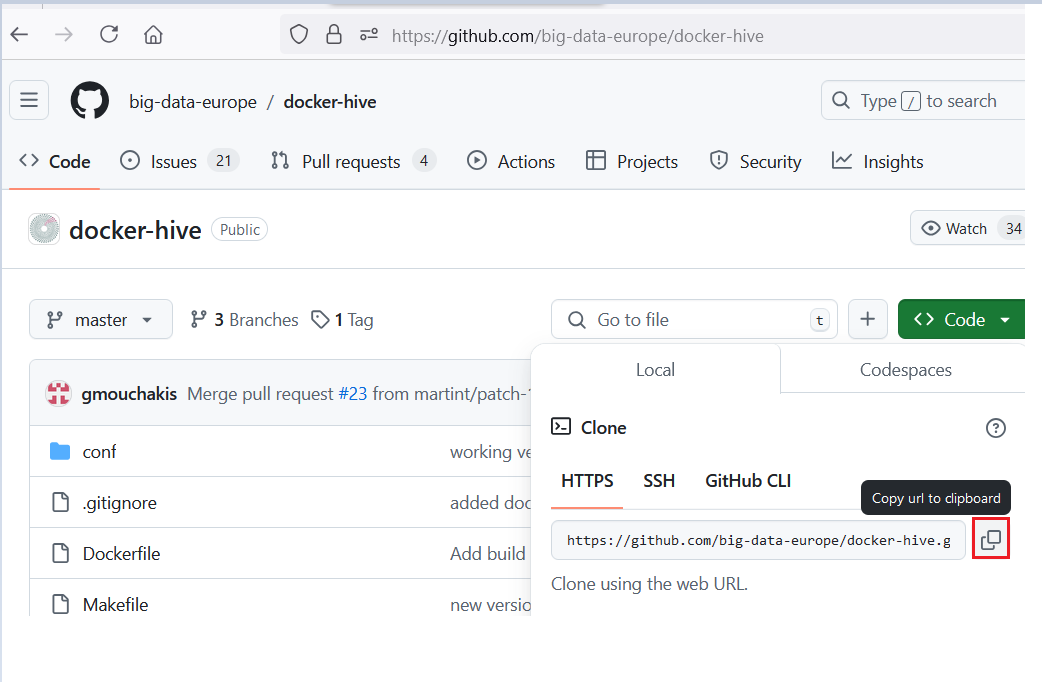
Vamos a cargar el contenido de una archivo .csv en una tabla generada por HIVE. En este caso las aplicaciones Hadoop, Hive y alguna otra aplicación más estan en un contenedor docker-compose.

Trabajaremos con el archivo: customers-1000.csv facilitado en el enunciado del ejercicio.

Empecemos…

Vamos a la web de repositorios de docker de Git Hub y obtenemos el enlace para clonar el docker haciendo click en el botón de la derecha

<https://github.com/big-data-europe/docker-hive>



Arrancamos un terminal y en nuestro equipo crearemos una carpeta donde clonaremos la imagen del docker

*mkdir /home/iabd/docker-hive*

Nos movemos a la carpeta creada

*cd docker-hive*

Clonamos la imagen del docker: (Esta acción puede llevar un tiempo)

*git clone* [*https://github.com/big-data-europe/docker-hive.git*](https://github.com/big-data-europe/docker-hive.git) *.*

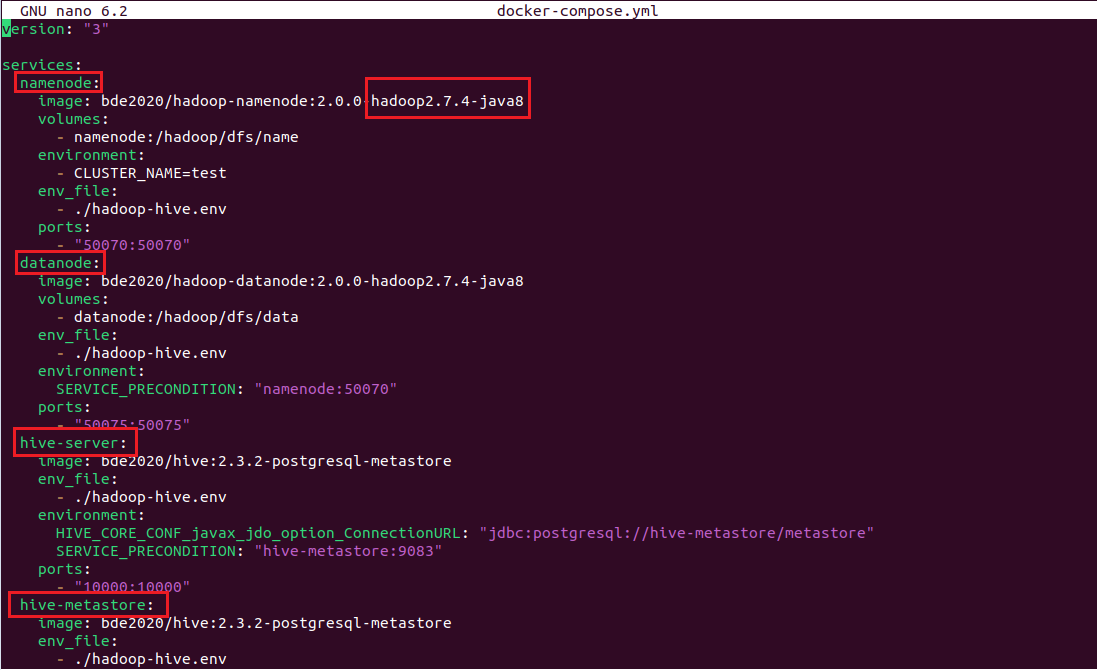
Comprobamos que se han creado los archivos

*ls*

Comprobamos los servicios implementas en el docker

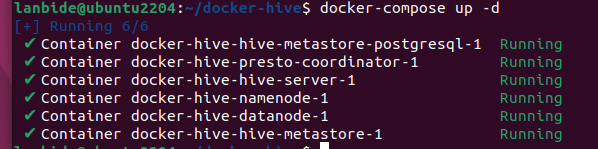
*nano docker-compose.yml*

y veremos algo así:

**

Arrancamos el docker (el parámetro -d indica que se arranque el docker en segundo plano)

*docker-compose up -d*

Y veremos los contenedores del docker levantados

Vamos a copiar el archivo desde nuestro equipo al contender:

*docker cp ~/Descargas/customers-1000.csv docker-hive\_compilado-hive-server-1:/opt/hive/examples*

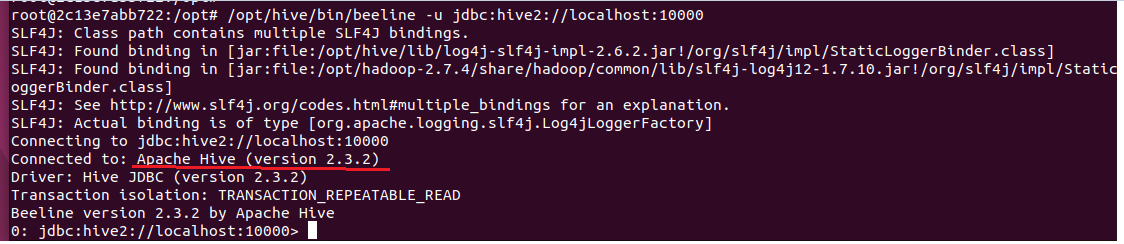
Nos conectamos al servicio HIVE (estamos conectándones al terminal del contenedor):

*docker exec -it docker-hive-hive-server-1 /bin/bash*

**

Utilizamos beeline que es el CLI para operar con HIVE:

*/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000*

**

Ya estamos en condiciones de trabajar con HIVE a través de BEELINE. Recuerda que para los nombres de las bases de datos y tablas es caseSensitive pero para escribir los comandos no es case sensitive. Por tanto, los siguiente comandos funcionarán de igual manera:

*show databases;*

*SHOW databases;*

Listamos las bases de datos existentes:

*show databases;*

Creamos la base de datos:

*create database organizaciones;*

Indicamos que queremos trabajar con la base: organizaciones

*use organizaciones;*

Creamos la tabla: personas

CREATE TABLE IF NOT EXISTS personas (

Index INT,

CustomerId STRING,

FirstName STRING,

LastName STRING,

Company STRING,

City STRING,

Country STRING,

Phone1 STRING,

Phone2 STRING,

Email STRING,

SubscriptionDate STRING,

Website STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

*Cargamos los datos a la tabla desde el archivo .csv*

LOAD DATA LOCAL INPATH '/opt/hive/examples/organizations-10000.csv' OVERWRITE INTO TABLE personas;

Leemos los datos de la tabla:

select \* from personas;

Nota: Una vez que hemos terminado de trabajar con los contenedores es buena práctica terminarlos.

Comando para eliminar todos los contenedores y volúmenes del archivo-docker-compose.yml:

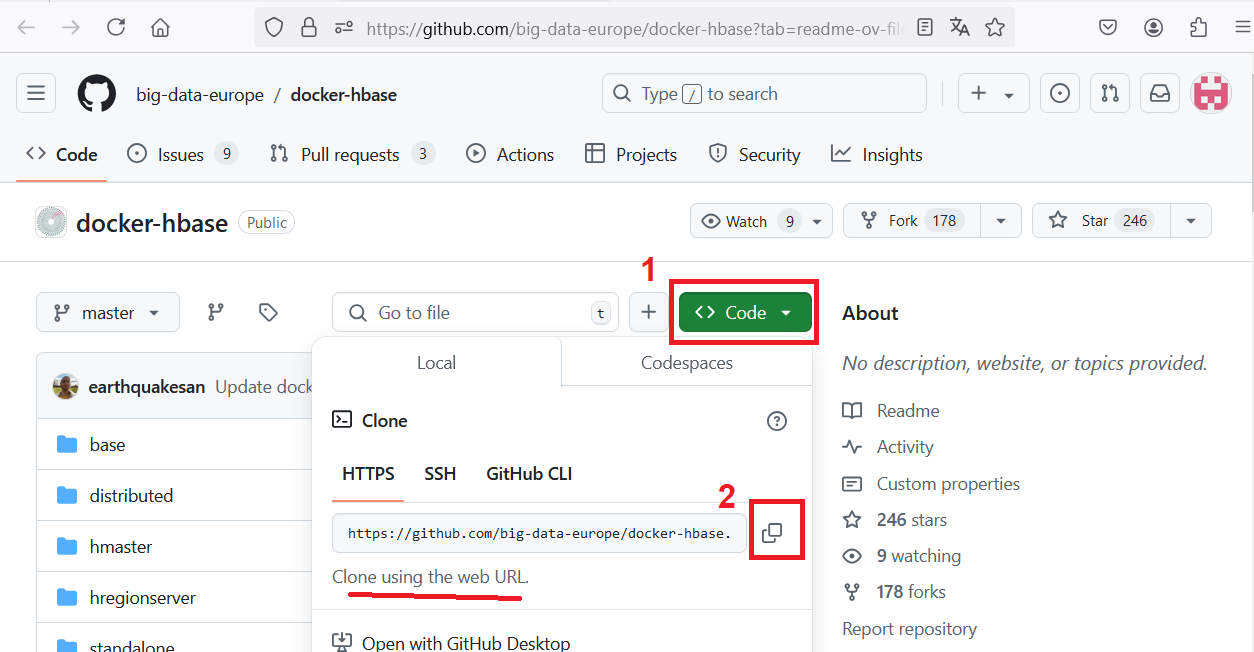
*sudo docker-compose -f archivo-docker-compose.yml down -v*

### EJERCICIO 4 (OPCIONAL)

En este ejercicio vamos a empezar a trabajar con hbase. Trabajaremos con hbase en un contenedor docker.

Vamos a la web de repositorios de docker de Git Hub y obtenemos el enlace para clonar el docker haciendo click en el botón de la derecha

<https://github.com/big-data-europe/docker-hbase?tab=readme-ov-file>



Arrancamos un terminal y en nuestro equipo crearemos una carpeta donde clonaremos la imagen del docker

*mkdir /home/iabd/docker\_hbase\_compilado*

Nos movemos a la carpeta creada

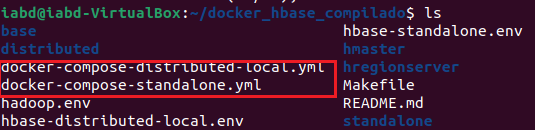
*cd docker\_hbase\_compilado/*

Clonamos la imagen del docker: (Esta acción puede llevar un tiempo)

*git clone https://github.com/big-data-europe/docker-hbase.git .*

Comprobamos que se han creado los archivos

*ls*

**

Tenemos dos docker-compose:

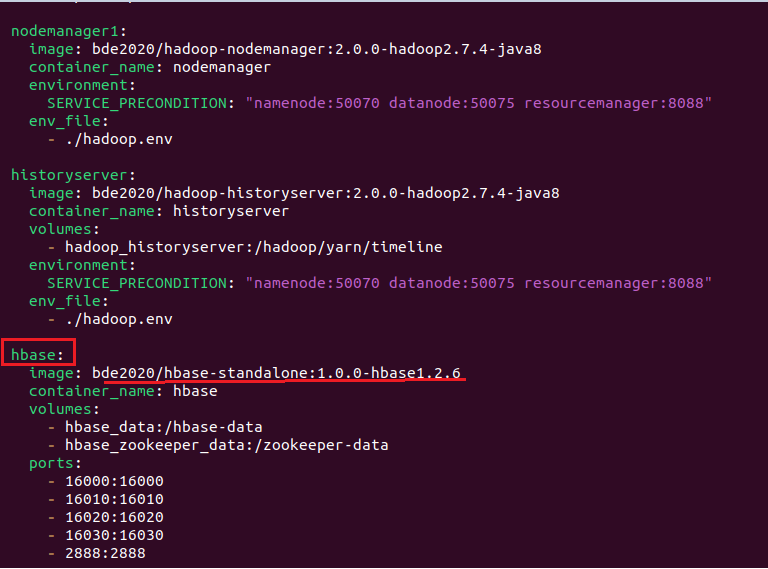
docker-compose-distributed-local.yml: HBase en un entorno distribuido local simula un clúster de múltiples nodos en una sola máquina.

docker-compose-standalone.yml: Permite ejecutar HBase en una sola máquina, sin la necesidad de un clúster completo de Hadoop.

Comprobamos los servicios implementados en el docker

*nano* *docker-compose-standalone.yml*

y veremos algo así:



Arrancamos el docker, como tenemos varios archivos .yml en la misma carpeta para arrancar el docker que queremos le pasaremos por parámetro el archivo docker a arrancar:

(el parámetro -d indica que se arranque el docker en segundo plano)

*docker-compose -f docker-compose-standalone.yml up -d*

Nota: Si te da problemas de conflictos de algún contenedor, es porque tienes algún contenedor con el mismo nombre en ejecución. Para solucionar esta situación debes cerrar todos los contenedores (por lo menos los que tengan nombres iguales al contenedor que estás intentando arrancar).

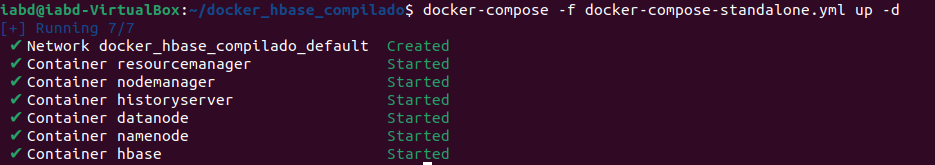
Comando para comprobar los contenedores en ejecución: *docker ps*

Comando para comprobar todos los contenderos (en ejecución y detenidos): *docker ps -a*

Comadno para eliminar todos los contenedores y volúmenes del archivo-docker-compose.yml:

*sudo docker-compose -f archivo-docker-compose.yml down -v*

Y veremos los contenedores del docker levantados



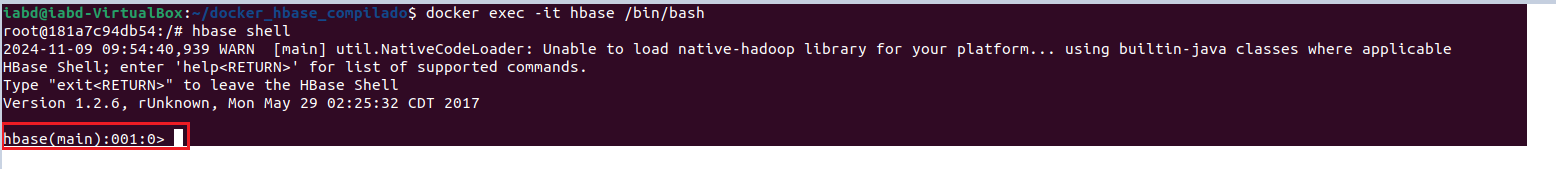
Nos conectamos al contenedor hbase (estamos conectándones al terminal del contenedor):

*docker exec -it hbase /bin/bash*

**

Arrancamos la consola de hbase:

*hbase shell*

**

Ya estamos en condiciones de trabajar con hbase a través de su consola.

En hbase se trabaja a nivel de tablas y no hay un concepto de bases de datos como MongoDB por ejemplo.

Veamos algunas instrucciones básicas para interactuar con hbase:

Listamos todas las tablas:

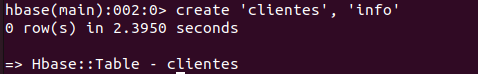
*list*

**Creamos una tabla**:

Para crear una tabla, se define el nombre de la tabla y las familias de columnas.

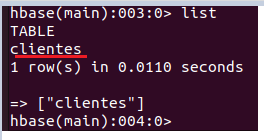
Creamos la tabla clientes con la familia de columnas info.

*create 'clientes', 'info'*

**

Para verificar si la tabla se ha creado:

*List*

**

Ya podemos realizar operaciones CRUD sobre la tabla: clientes

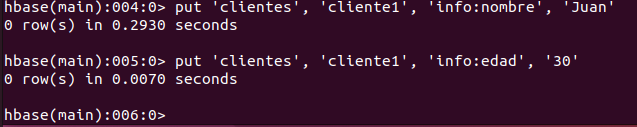
**Insertar datos (Put):**

put 'nombre\_tabla', 'clave\_fila', 'familia\_columna:columna', 'valor'

Vamos a insertar datos para un cliente:

*put 'clientes', 'cliente1', 'info:nombre', 'Juan'*

*put 'clientes', 'cliente1', 'info:edad', '30'*



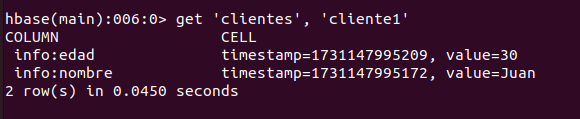
**Leer datos (Get)**

Para leer datos de una fila específica, usamos get.

get 'nombre\_tabla', 'clave\_fila'

Vamos a leer los datos de cliente1:

*get 'clientes', 'cliente1'*

**

**Escanear datos (Scan)**

Para ver todas las filas de una tabla, usamos scan.

scan 'nombre\_tabla'

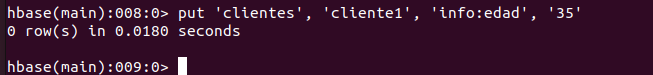
Vamos a leer los datos de la tabla que hemos creado: clientes

*scan 'clientes'*

**Actualizar datos (Put)**

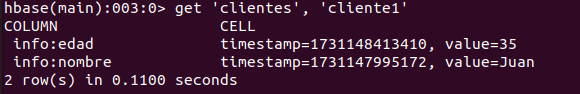
La misma instrucción put se usa para actualizar un valor, indicando la misma clave de fila y columna pero con un valor nuevo.

*put 'clientes', 'cliente1', 'info:edad', '35'*

**

Vamos a leer los datos y verificar el cambio:

*get 'clientes', 'cliente1'*



**Eliminar datos**

**Eliminar una columna específica:**

delete 'nombre\_tabla', 'clave\_fila', 'familia\_columna:columna'

Vamos a borrar la columna edad:

*delete 'clientes', 'cliente1', 'info:edad'*

**

**Eliminar una fila completa:**

deleteall 'nombre\_tabla', 'clave\_fila'

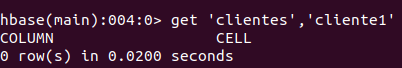
Vamos a borrar la fila cliente1

*deleteall 'clientes', 'cliente1'*

**

Comprobamos que se ha borrado la fila completa:

*get 'clientes','cliente1'*

**

Nota: Una vez que hemos terminado de trabajar con los contenedores es buena práctica terminarlos.

Comando para eliminar todos los contenedores y volúmenes del archivo-docker-compose.yml:

*sudo docker-compose -f archivo-docker-compose.yml down -v*