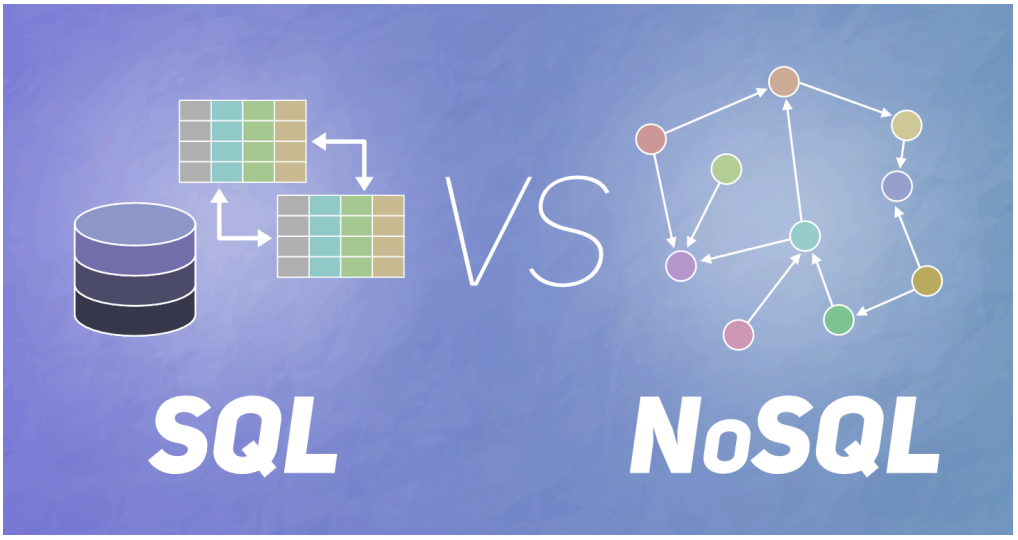
Joshua Sangareau Quesada **1ºDAM**
Rafael García Cambrón

Bases de Datos



Estudio sobre bases de datos NoSQL.

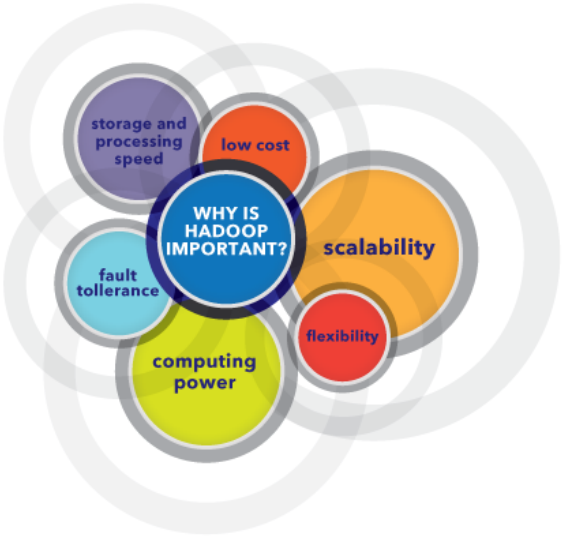
Introducción

Antes de comenzar a hablar de HBase debemos explicar Hadoop para poder entender mejor HBase.

¿Qué es Hadoop?

Apache Hadoop es un marco de código abierto que se utiliza para almacenar y procesar de manera eficiente conjuntos de datos grandes cuyo tamaño varía entre los gigabytes y los petabytes de datos. En lugar de utilizar una sola computadora grande para procesar y almacenar los datos, Hadoop facilita la creación de clústeres de varias computadoras para analizar conjuntos de datos masivos en paralelo y con mayor rapidez.

¿Por qué es importante Hadoop?



Capacidad de almacenar y procesar enormes cantidades de cualquier tipo de datos, al instante. Con el incremento constante de los volúmenes y variedades de datos, en especial provenientes de medios sociales y la Internet de las Cosas (IoT), ésa es una consideración importante.

Poder de cómputo. El modelo de cómputo distribuido de Hadoop procesa big data a gran velocidad. Cuantos más nodos de cómputo utiliza usted, mayor poder de procesamiento tiene.

Tolerancia a fallos. El procesamiento de datos y aplicaciones está protegido contra fallos del hardware. Si falla un nodo, los trabajos son redirigidos automáticamente a otros modos para asegurarse de que no falle el procesamiento distribuido. Se almacenan múltiples copias de todos los datos de manera automática.

Flexibilidad. A diferencia de las bases de datos relacionales, no tiene que procesar previamente los datos antes de almacenarlos. Puede almacenar tantos datos como desee y decidir cómo utilizarlos más tarde. Eso incluye datos no estructurados como texto, imágenes y videos.

Bajo costo. La estructura de código abierto es gratuita y emplea hardware comercial para almacenar grandes cantidades de datos.

Escalabilidad. Puede hacer crecer fácilmente su sistema para que procese más datos con sólo agregar nodos. Se requiere poca administración.

¿Cuáles son los cuatro módulos principales de Hadoop?

Hadoop consta de cuatro módulos principales:

Sistema de archivos distribuido de Hadoop (HDFS): sistema de archivos distribuido que se ejecuta en hardware estándar o de gama baja. HDFS proporciona un mejor rendimiento de datos que los sistemas de archivos tradicionales, además de una alta tolerancia a errores y compatibilidad nativa con conjuntos de datos de gran tamaño.

Yet Another Resource Negotiator (YARN): administra y supervisa los nodos del clúster y el uso de recursos. Programa trabajos y tareas.

MapReduce: un marco que ayuda a los programas a realizar el cálculo paralelo de los datos. La tarea de Map toma los datos de entrada y los convierte en un conjunto de datos que se puede calcular en pares de valores clave. La salida de la tarea de Map la consumen las tareas de Reduce para agregar la salida y proporcionar el resultado deseado.

Hadoop Common: proporciona bibliotecas Java comunes que se pueden usar en todos los módulos.

¿Qué es HBase?

HBase es una base de datos NoSQL distribuida y escalable, desarrollada como parte del proyecto Hadoop Distributed File System (HDFS). Está diseñada para manejar grandes cantidades de datos y proporcionar un acceso rápido y eficiente a los mismos.

Apache HBase, maneja grandes cantidades de datos estructurados y no estructurados, lo que significa que puede almacenar una amplia variedad de datos. Es una base de datos de clave-valor, en la que los datos se almacenan en una tabla hash. La clave es un identificador único que se utiliza para acceder a los datos y el valor es la información que se quiere almacenar. HBase utiliza la tecnología de indexación de Apache Hadoop, lo que permite un acceso rápido y eficiente a los datos.

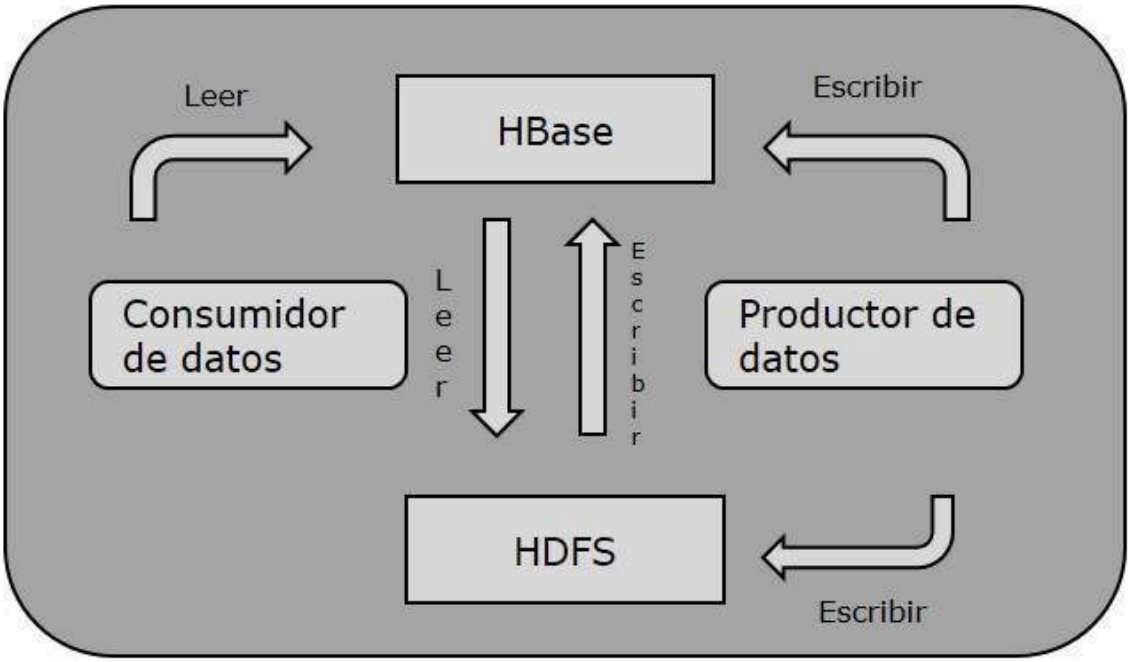
Una de las principales características de HBase es su capacidad de escalar horizontalmente. Esto significa que se pueden agregar nuevos nodos al cluster para aumentar la capacidad de almacenamiento y procesamiento de datos. HBase también proporciona una alta disponibilidad de datos, lo que significa que si un nodo falla, los datos se replican en otros nodos del cluster y se pueden acceder sin interrupción.

HBase utiliza el modelo de consistencia eventual, lo que significa que los cambios en la base de datos pueden tardar un tiempo en propagarse a través del cluster. Sin embargo, esto permite una alta disponibilidad y escalabilidad, lo que la convierte en

una opción popular para aplicaciones en las que la disponibilidad es más importante que la consistencia inmediata de los datos.
Otra característica importante de HBase es su capacidad de realizar consultas en tiempo real. HBase utiliza la tecnología de indexación de Apache Hadoop, que permite consultas rápidas y eficientes en grandes cantidades de datos. Además, HBase es compatible con una amplia variedad de lenguajes de programación, como Java, Python, Ruby y PHP.
En resumen, HBase es una base de datos NoSQL distribuida y escalable, diseñada para manejar grandes cantidades de datos y proporcionar acceso rápido y eficiente a los mismos. Su capacidad de escalar horizontalmente, alta disponibilidad, capacidad de realizar consultas en tiempo real y compatibilidad con una amplia variedad de lenguajes de programación la convierten en una opción popular para aquellos que necesitan una base de datos NoSQL confiable y escalable para sus aplicaciones.
HBase se basa en ZooKeeper para una coordinación de alto rendimiento. ZooKeeper está integrado en HBase, pero si está ejecutando un clúster de producción, se sugiere que tenga un clúster ZooKeeper dedicado que esté integrado con su clúster HBase.
HBase funciona bien con Hive, un motor de consulta para el procesamiento por lotes de big data, para habilitar aplicaciones de big data tolerantes a fallos.

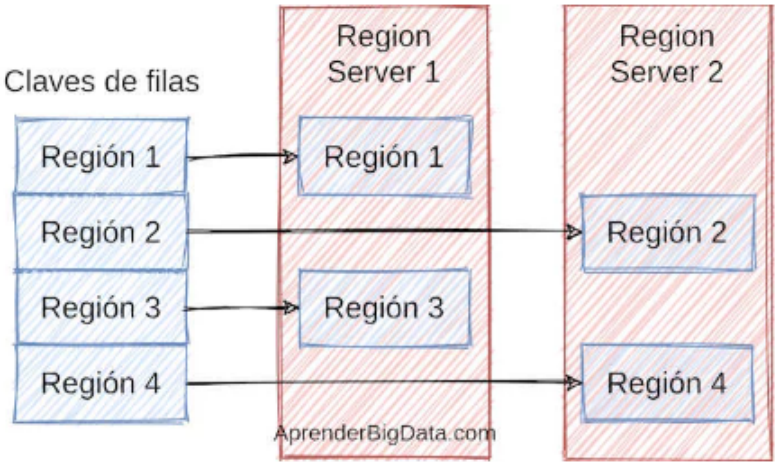
Cómo se relacionan HBase y HDFS

HDFS	HBase
HDFS es un sistema de ficheros distribuido adecuado para almacenar archivos de gran tamaño.	HBase es una base de datos creada en la parte superior de la HDFS.
HDFS no admite búsquedas rápidas de registro individual.	HBase proporciona búsquedas rápidas tablas más grandes
Proporciona una alta latencia de procesamiento, usa un concepto de procesamiento por lotes.	Proporciona acceso de baja latencia a filas de miles de millones de registros (acceso aleatorio).
Sólo proporciona acceso secuencial de los datos.	HBase internamente usa tablas Hash y proporciona acceso aleatorio, y que almacena los datos en archivos indexados HDFS búsquedas más rápido.



Modelo de datos

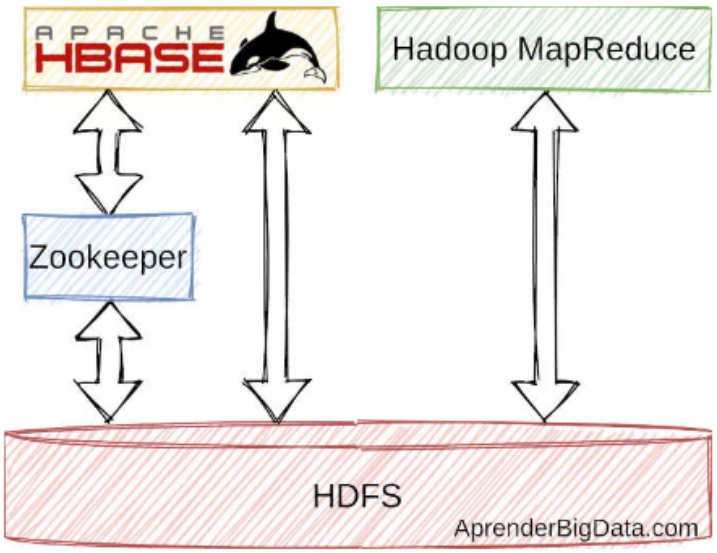
El modelo de datos está basado en Google Big Table, por lo que es muy similar. Está diseñado para proporcionar acceso aleatorio a una gran cantidad de datos estructurados.
En HBase los datos se dividen en tablas. Estas tablas, tienen filas que representan los datos. Cada fila tiene una clave que actúa de identificador único y que divide las tablas en regiones. Esta clave de fila es un array de bytes.



Como en una base de datos relacional o SQL, las columnas representan los atributos. Las columnas se agrupan en familias de columnas (con valores semánticos relacionados), que se almacenan en los ficheros de HDFS denominados HFiles. Cuando no tiene un valor, una columna no se almacena ni se reserva ese espacio.
Cada familia de columnas puede tener millones de columnas. Los usuarios pueden crear más en cualquier momento.

Arquitectura y componentes

HBase tiene una arquitectura de tipo maestro-esclavo (master-slave) y se puede desplegar en decenas y cientos de nodos.

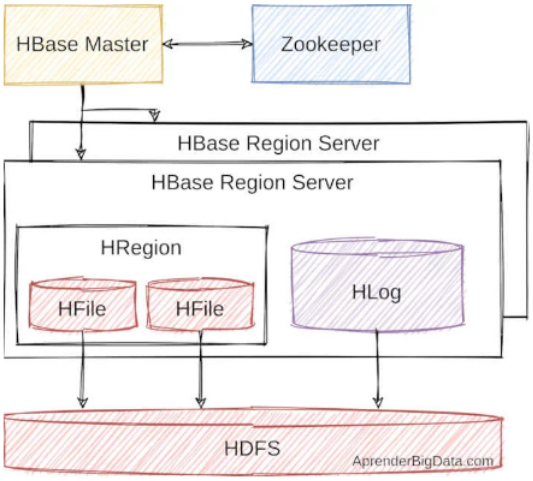


HBase Master: Es el componente responsable de asignar las regiones en las que se encuentran distribuidos los datos en los nodos del clúster. Para ello, particiona el espacio de claves de las tablas y asigna las regiones resultantes a los servidores de regiones (Region Servers). También, mantiene la carga balanceada asignando regiones a los nodos.
El Máster es el punto de entrada para las peticiones de los usuarios, las cuales redirige al Region Server correspondiente. También controla los fallos que se producen en el sistema. Es posible ejecutar varios Master para habilitar alta disponibilidad.

Region Server: Este componente se ejecuta en todos los nodos workers del cluster y aloja las regiones. Tiene una caché de bloques que contiene los datos que se acceden con más frecuencia para optimizar las consultas (BlockCache). También gestiona una caché de escritura en memoria (MemStore) que aloja los datos que aún no han persistido en el disco. Además, contiene el log de escritura o WAL (Write-ahead log). Cuando el MemStore acumula suficientes datos se persiste en un nuevo HFile con la operación de Flush.
Zookeeper: Apache Zookeeper es el coordinador del clúster de HBase. Contiene la configuración de los nodos y actúa también en caso de detectar un fallo mediante un demonio de monitorización. El HBase Master hace el seguimiento de los Region Servers con la información que mantiene Zookeeper.

Características de HBase

Escalabilidad Horizontal: HBase usa sharding automático para distribuir sus tablas de datos en regiones, que son las unidades de balanceo de carga y de distribución en los nodos del clúster. Las regiones también pueden combinarse o dividirse en las operaciones de balanceo.



Tolerancia a Fallos: HBase se puede recuperar automáticamente en caso de fallos en regiones. Cuando el HBase Master detecta un fallo en un Region Server, reasigna esas regiones a un nuevo Region Server. Los datos no persistidos se recuperan directamente del HLog. Por otro lado, los fallos en nodos de HDFS son transparentes, ya que HDFS se encarga de replicarlos.

Consistencia de datos: Proporciona consistencia fuerte de los datos, tanto en lecturas como en escrituras, por lo que siempre se devolverá un dato actualizado. Las operaciones de escritura no terminarán satisfactoriamente hasta que todas las réplicas del dato hayan sido actualizadas correctamente.

Apache Hbase: ventajas y desventajas.

Una vez que hemos visto todas las características de Apache HBase, también es importante ver cuáles son sus ventajas y desventajas en el trabajo dentro del análisis de datos. Empecemos con las ventajas de Hbase:

- HBase permite hacer búsquedas de versiones, así como de los históricos de los datos. De esta manera se puede encontrar fácilmente la versión de la información que necesitamos.
- Cuando existe una carga alta de información se puede escalar el sistema simplemente añadiendo nuevas máquinas al clúster de ordenadores.
- La integración con Hadoop garantiza la fiabilidad de sus datos para el análisis para Big Data.
- Los posibles fallos se pueden evitar ya que cada parte del clúster tiene copias de seguridad de todos los elementos que forman parte de la base de datos, por lo que, si uno falla, los otros pueden seguir trabajando.

No obstante, a pesar de todas sus ventajas, HBase también tiene algunas desventajas:

- Su implementación basada en Java y la arquitectura de Hadoop hace que su API sea más adecuada para proyectos desarrollados en este lenguaje de programación.
- El entorno de desarrollo de los nodos necesita distintas dependencias lo que hace que configurarlo sea problemático.
- Ocupa mucha memoria y al estar configurado y construido para HDFS para el análisis por lotes, el rendimiento de la lectura de datos no es especialmente alto.
- Es relativamente torpe en comparación con otras bases de datos no relacionales.

A pesar de sus desventajas, HBase forma parte de Hadoop y, con ello, resulta útil para empresas que buscan soluciones Big Data a costes reducidos para proyectos desarrollados con arquitectura Java.

Herramientas de Administración

HBase, es parte del ecosistema Apache Hadoop, por lo tanto cuenta con varias herramientas de administración que facilitan la gestión y supervisión de clústeres HBase. Aquí hay algunas de las herramientas más comunes:

1. **HBase Shell**: La HBase Shell es una interfaz de línea de comandos que permite a los usuarios interactuar con las tablas de HBase y ejecutar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Se puede utilizar para crear tablas, insertar datos, consultar datos y administrar el clúster.

2. **HBase Web UI**: La interfaz web de HBase proporciona una interfaz gráfica de usuario para supervisar y administrar el clúster de HBase. Permite monitorear el estado de las regiones, las tablas, las operaciones de lectura y escritura, y otras métricas del clúster.

3. **HBase REST**: HBase REST es un servicio web que permite interactuar con HBase a través de solicitudes HTTP RESTful. Esto proporciona una forma programática de interactuar con HBase, lo que puede ser útil para aplicaciones web y servicios que deseen acceder a datos almacenados en HBase.

4. **HBase Thrift**: HBase Thrift es otro servicio de interfaz que permite a los clientes acceder a HBase a través de un API de acceso a datos multi-lenguaje. Permite a los clientes utilizar una variedad de lenguajes de programación para interactuar con HBase, incluyendo Java, Python, Ruby, PHP, y otros.

5. **HBase Metrics**: HBase proporciona métricas detalladas sobre el rendimiento y el estado del clúster que pueden ser recopiladas y visualizadas utilizando herramientas de monitoreo y visualización como Apache Ambari, Grafana, o Prometheus.

6. **HBase Coprocessors**: Aunque no es una herramienta de administración en sí misma, los coprocesadores de HBase son extensiones que permiten a los desarrolladores ejecutar lógica personalizada en el servidor HBase para realizar tareas como la validación de datos, la agregación y la indexación en tiempo real.

1.1 HBase Shell Inicio

La shell de HBase permite a los usuarios realizar una variedad de tareas, como crear y eliminar tablas, insertar y recuperar datos, realizar escaneos en tablas y ejecutar operaciones administrativas, como habilitar y deshabilitar tablas, entre otras cosas. La shell de HBase generalmente se ejecuta en el nodo maestro del clúster HBase y se puede acceder a ella mediante la ejecución del script de shell proporcionado con la instalación de HBase.

2.1 HBase Web UI

La interfaz de usuario web de HBase Master es una herramienta simple pero útil para obtener una descripción general del estado actual del clúster. Desde su página, puede obtener la versión de HBase en ejecución, su configuración básica, incluida la ruta raíz HDFS y el quórum de ZooKeeper, el promedio de carga del clúster y una tabla, región y lista de servidores de regiones.
Además, puede dividir manualmente una región utilizando una clave de fila de límite particular. Esto es útil cuando desactiva la división automática de regiones de su clúster

3.1 HBase REST

Puede utilizar el servidor REST Apache HBase para interactuar con Apache HBase. Esta es una muy buena alternativa si no desea utilizar la API de Java. Las interacciones se producen mediante URL y la API REST. REST utiliza HTTP para realizar diversas acciones y esto facilita la interfaz con la base de datos operativa utilizando una amplia gama de lenguajes de programación.

Puede utilizar el servidor REST para crear, eliminar tablas y realizar otras operaciones que tengan puntos finales REST. Puede configurar SSL(La ruta al archivo del almacén de claves TLS/SSL que contiene el certificado del servidor y la clave privada utilizados para TLS/SSL) para el cifrado entre el cliente y el servidor REST. Esto le ayuda a garantizar que sus operaciones sean seguras durante la transmisión de datos.
El uso del servidor REST le permite acceder a sus datos a través de diferentes límites de red. Por ejemplo, si tiene un clúster de centro de datos de base de datos operativa de Cloudera ejecutándose dentro de una red privada y no desea exponerlo a la red pública de su empresa, el servidor REST puede funcionar como puerta de enlace entre las redes públicas y privadas.

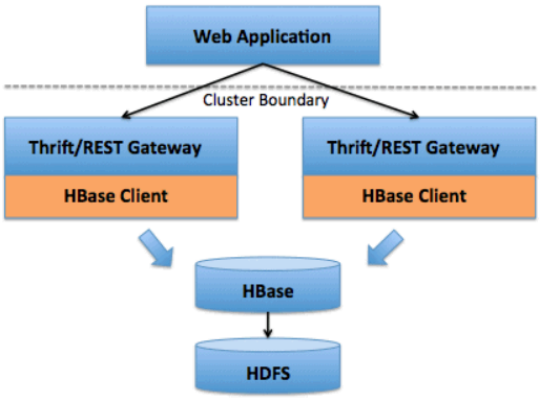
Instalación del servidor REST usando Cloudera Manager
Cloudera Manager es una aplicación líder para almacenar y procesar datos de forma distribuida, caracterizada por una plataforma centralizada con implementación automatizada bajo un ambiente empresarial.

Usando la API REST el servidor REST de HBase expone puntos finales que proporcionan operaciones CRUD (crear, leer, actualizar, eliminar) para cada proceso de HBase, así como tablas, regiones y espacios de nombres, también puede usarse para transmitir datos a HBase, desde otra aplicación o script de shell, o usando un cliente HTTP como wget o curl.

4.1 HBase Thrift

Thrift es un marco de software que permite crear enlaces entre idiomas. En el contexto de HBase, Java es el único ciudadano de primera clase. Sin embargo, la interfaz HBase Thrift permite que otros lenguajes accedan a HBase a través de Thrift conectándose a un servidor Thrift que interactúa con el cliente Java.

Para que Thrift y REST funcionen, es necesario ejecutar otro daemon HBase para manejar estas solicitudes. Estos daemon se pueden instalar con los paquetes hbase-thrift y hbase-rest. El siguiente diagrama muestra cómo se colocan Thrift y REST en el clúster.



Tenga en cuenta que los hosts de los clientes Thrift y REST generalmente no ejecutan ningún otro servicio (como DataNodes o RegionServers) para mantener la sobrecarga baja y la capacidad de respuesta alta para las interacciones REST o Thrift.

Asegúrese de instalar e iniciar estos daemon en nodos que tengan acceso tanto al clúster de Hadoop como a la aplicación que necesita acceso a HBase. La interfaz Thrift no tiene ningún equilibrio de carga incorporado, por lo que todo el equilibrio de carga deberá realizarse con herramientas externas como una operación por turnos de DNS, una dirección IP virtual o mediante código. Cloudera Manager también facilita la instalación y gestión de los servicios HBase REST y Thrift.

La desventaja de Thrift es que es más difícil de configurar que REST. Deberá compilar Thrift y generar los enlaces específicos del idioma. Estos enlaces son buenos porque le brindan código para el lenguaje en el que está trabajando; no es necesario analizar XML o JSON como en REST; más bien, la interfaz Thrift le brinda acceso directo a los datos de la fila. Otra característica interesante es que el protocolo Thrift tiene transporte binario nativo; No necesitará codificar y decodificar datos en base64.
Para comenzar a utilizar la interfaz Thrift, debe averiguar en qué puerto se está ejecutando. El puerto predeterminado para CDH es el puerto 9090.

5.1 HBase Metrics

Para recopilar las métricas expuestas por los distintos servicios de HBase se puede utilizar las API de Java Management Extensions (JMX). También puedes exportar las métricas en formato Prometheus.Puede utilizar varias métricas de HBase para comprender el estado de su clúster.

Puede acceder a las métricas de HBase a través de Java Management Extensions (JMX) a través de la interfaz web de HBase o accediendo directamente al agente remoto de JMX.

6.1 HBase Coprocessors

Coprocessor es un marco que proporciona una manera fácil de ejecutar su código personalizado en Region Server.

Cuando trabaja con cualquier almacén de datos (como RDBMS o HBase), obtiene los datos (en el caso de RDBMS puede utilizar la consulta y en el caso de HBase utiliza Obtener o Escanear). Para obtener solo datos relevantes, los filtra (para RDBMS coloca condiciones en la cláusula 'WHERE' y en HBase usa filtros.

Este escenario es casi ideal para "datos pequeños", como unos pocos miles de filas con un montón de columnas. Ahora imagine un escenario en el que hay miles de millones de filas y millones de columnas y desea realizar algún cálculo que requiera todos los datos, cómo calcular el promedio o la suma. Incluso si está interesado en

solo unas pocas columnas, aún debe recuperar todas las filas. Este enfoque tiene algunos inconvenientes, como se describe a continuación:

- En este enfoque, la transferencia de datos (desde el almacén de datos al lado del cliente) se convertirá en el cuello de botella, y el tiempo necesario para completar la operación está limitado por la velocidad a la que se realiza la transferencia de datos.

- El ancho de banda es uno de los recursos más preciados en cualquier centro de datos. Operaciones como esta afectarán gravemente el rendimiento de su clúster.

En un escenario como este, es mejor trasladar el cálculo a los datos mismos; al igual que el procedimiento almacenado (una mejor analogía es el modelo MapReduce). El coprocesador te ayuda a lograr esto. Para dar una idea de las capacidades del coprocesador, diferentes personas dan diferentes analogías. La analogía más famosa de coprocesador presentes en la industria son:
Triggers and Stored Procedure(Disparadores y procedimiento almacenado): esta es la analogía más común que encontrará para el coprocesador. (El documento oficial utiliza esta analogía). Coprocesador observador se compara con los activadores porque, al igual que los activadores, ejecutan su código personalizado cuando ocurre cierto evento (como Get o Put, etc.). De manera similar, el coprocesador de puntos finales se compara con los procedimientos almacenados y puede realizar cálculos personalizados en los datos directamente dentro del servidor de la región.

Lenguaje de Consultas en HBase Apache

HBase contiene una shell para que el usuario se comunique con HBase. Hadoop HBase utiliza el sistema de archivos para almacenar sus datos. Tendrá un servidor maestro y servidores de región. El almacenamiento de datos se hará en la forma de las regiones (tablas). Estas regiones se pueden dividir y se almacenan en servidores de región.
El servidor maestro administra estos servidores de región y todas estas tareas se realizan en HDFS. A continuación, se presentan algunos de los comandos admitidos por HBase Shell.

Comandos generales:

- **Status**: indica el estado de HBase, por ejemplo, el número de servidores.
- **Version**: ofrece la versión de HBase que se utiliza.
- **Table_help**: proporciona ayuda en la tabla de referencia.
- **Whoami**: proporciona información sobre el usuario, es decir devuelve los detalles del usuario de HBase.

Lenguaje de definición de datos:

Estos son algunos comandos que operan en las tablas de HBase.

- **create(Crear)**: Crea una tabla.
- **list(lista)**: Te dice todas las tablas creadas.
- **describe <table_name>**:Describe la tabla indicada
- **disable(desactivar)**: Desactiva una tabla.
- **disable <table_name>:**Deshabilita la tabla nombrada (proceso necesario previamente a eliminar la tabla)
- **disable_all<expresion>:**Este comando deshabilitará todas las tablas que coincidan con la expresión regular dada.
- **enable <nombre_tabla>**: activa una tabla, deshabilitada previamente, la devuelve al estado anterior a deshabilitarla.
- **drop <table name>:** Borra la tabla a la que se haga referencia (Para eliminar la tabla presente en HBase, primero tenemos que deshabilitarla**)**
- **drop_all<expresion>:**Este comando eliminará todas las tablas que coincidan con la expresión regular dada
- **is_disabled 'nombre_tabla'**: Verifica si una tabla está habilitada o deshabilitada.
- **alter <tablename>, NAME=><column familyname>, VERSIONS=>5:** Altera una tabla Para cambiar o agregar la familia de columnas 'familyname' en la tabla 'table_name' desde el valor actual para mantener un máximo de 5 VERSIONES de celda,

Lenguaje de manipulación de datos:

- **count <'nombre_tabla'>**:El comando devuelve el recuento de filas de la tabla nombre_tabla. El intervalo de recuento se puede especificar opcionalmente.

  ejemplo: `count <'tablename'>, CACHE =>1000` **:** limita a 1000 el valor a devolver que en este caso es el número de filas de la tabla referenciada

- **put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>:** Este comando indica que estás realizando una operación de inserción de datos en una tabla,fila y columna específica de HBase.

  ejemplo:
  `put<'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>10`

  rowname es la fila donde se desea insertar datos
  columnvalue es la columna donde se desea insertar datos
  value es para decir que el valor va a ser 10

- **delete <'tablename'>,<'nombre_fila'>,<'nombre_columna'>:** Borra datos de una tabla, fila y columna determinada, la expresión seria

- **deleteall <'nombre_tabla'>, <'nombre_fila'>** : Este comando eliminará todas las celdas de una fila determinada.

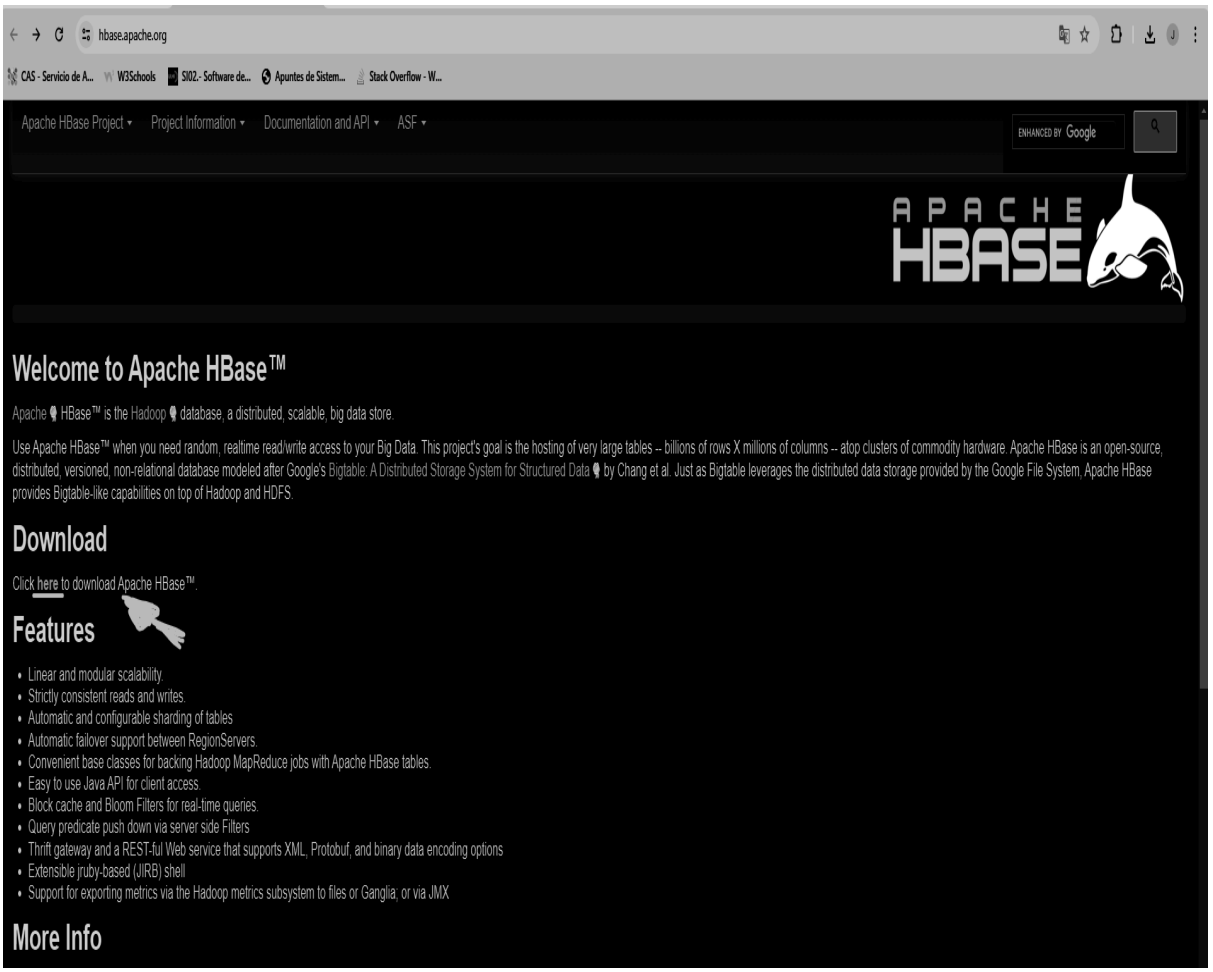
- **scan <'tablename'>:** Este comando escanea toda la tabla y muestra el contenido de la tabla.

Usos de HBase en empresas reales:

1. Facebook: Facebook utiliza HBase para almacenar su Graph API, que es una API que permite a los desarrolladores acceder a los datos sociales de Facebook. HBase también se utiliza para almacenar el chat de Facebook y la información del usuario.
2. Twitter: Twitter utiliza HBase para almacenar sus tweets, menciones, seguidores y otra información relacionada con la plataforma. HBase permite a Twitter escalar horizontalmente y manejar grandes cantidades de datos en tiempo real.
3. Yahoo!: Yahoo! utiliza HBase para almacenar y procesar grandes cantidades de datos. HBase se utiliza en Yahoo! Mail, Yahoo! Sports y en otros productos de Yahoo! que requieren una gran cantidad de datos.
4. Netflix: Netflix utiliza HBase para almacenar y procesar datos de sus usuarios, como las películas y programas de televisión que han visto y las calificaciones que han dado. HBase permite a Netflix escalar horizontalmente y proporcionar recomendaciones personalizadas a sus usuarios.
5. Adobe: Adobe utiliza HBase como base de datos escalable para almacenar y procesar grandes cantidades de datos para su plataforma de marketing en la nube. HBase se utiliza para almacenar información del cliente, como datos de transacciones y comportamiento en línea.
6. Alibaba: Alibaba, el gigante chino del comercio electrónico, utiliza HBase para almacenar y procesar los datos de sus aplicaciones en tiempo real, como la gestión de pedidos y el seguimiento de envíos.
7. Pinterest: Pinterest utiliza HBase para almacenar y procesar los datos de su plataforma, incluyendo la información de los usuarios, los pines y tableros.
8. Spotify: Spotify utiliza HBase para almacenar y procesar datos relacionados con la música, como las canciones que han sido reproducidas por los usuarios, las listas de reproducción y las recomendaciones personalizadas.
9. Microsoft: Microsoft utiliza HBase como base de datos para almacenar y procesar grandes cantidades de datos en su plataforma de nube, Azure. HBase se utiliza en aplicaciones como Bing, Office y Skype.
10. Airbnb: Airbnb utiliza HBase para almacenar y procesar datos relacionados con las reservas, las evaluaciones de los usuarios y la información de los anfitriones.

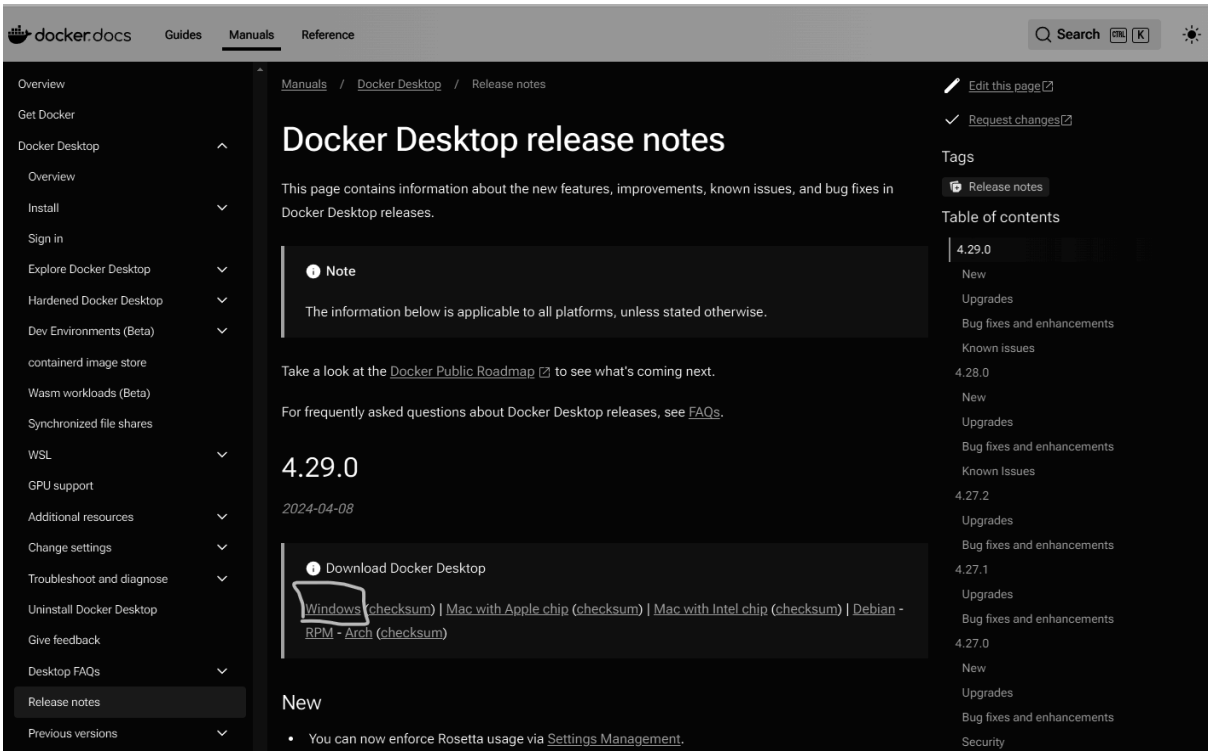
Tutorial de descarga e instalación:

Entrar en página principal de HBase y clickear en ‘here’ en la sección de download.



El paso anterior es si se quiere instalar en nuestro PC. En nuestro caso trabajaremos en un repositorio de Docker.

El primer paso es tener docker instalado en el ordenador descargando el ejecutable desde:
https://docs.docker.com/desktop/release-notes/



Se instala en el ordenador, se ejecuta el cmd como administrador y se usa el comando: wsl --update para tener actualizado el virtualizador de Linux sobre Windows, este paso es fundamental para el correcto funcionamiento.(En nuestro caso ya lo tenemos realizado y actualizado):

```
C:\Windows\system32>wsl --update
Comprobando actualizaciones.
La versión más reciente de Subsistema de Windows para Linux ya está instalada.

C:\Windows\system32>
```

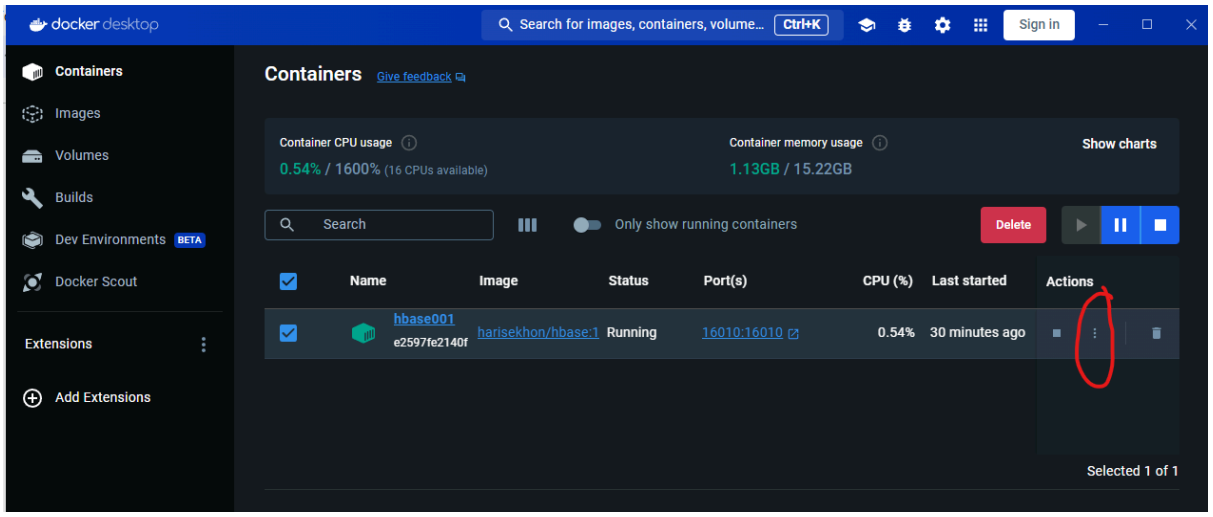
Tras este paso se reinicia el ordenador y se ejecuta Docker.
A partir de aquí seguiremos los pasos de la siguiente página para proceder a la descarga de la imagen de la base de datos para poder usarla en el repositorio.
https://www.programmersought.com/article/6344708049/

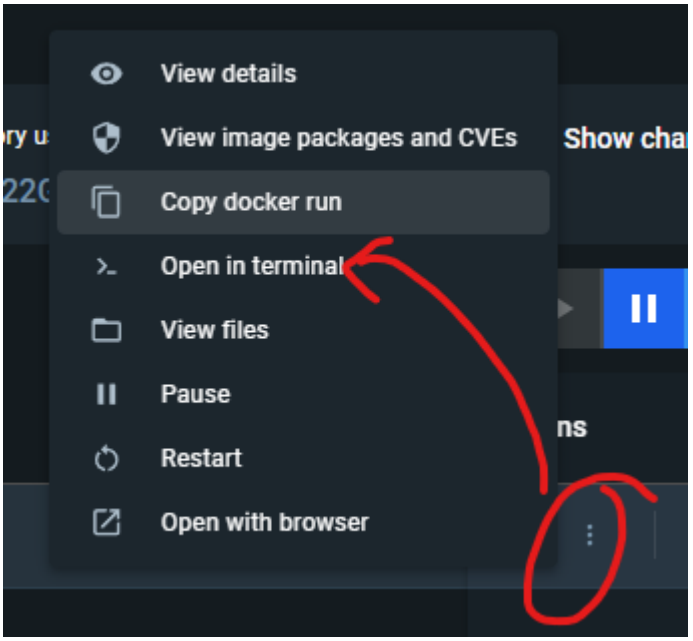
Lanzando desde cmd estos 2 comandos ya inicia la descarga de la imagen y la ejecuta desde docker

```
docker pull harisekhon/hbase:1.3
```

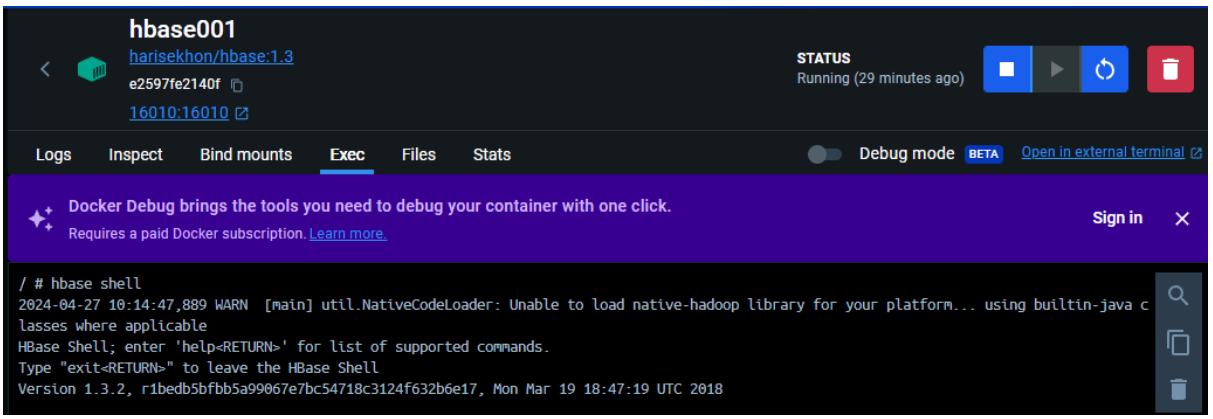
```
docker run -d --name hbase001 -P harisekhon/hbase:1.3
```

Tras esto ya tendremos instalada la imagen en el docker y podremos acceder a la base de datos desde el terminal de Docker para poder empezar a trabajar con ella.

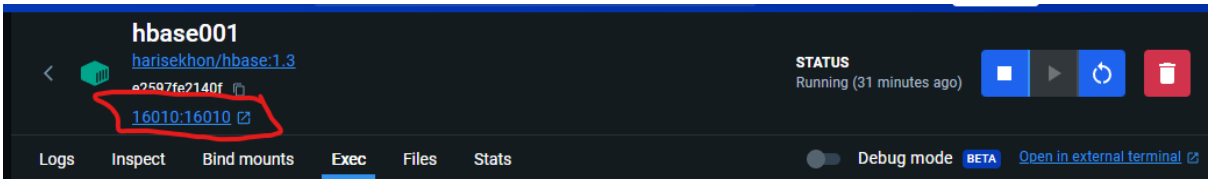


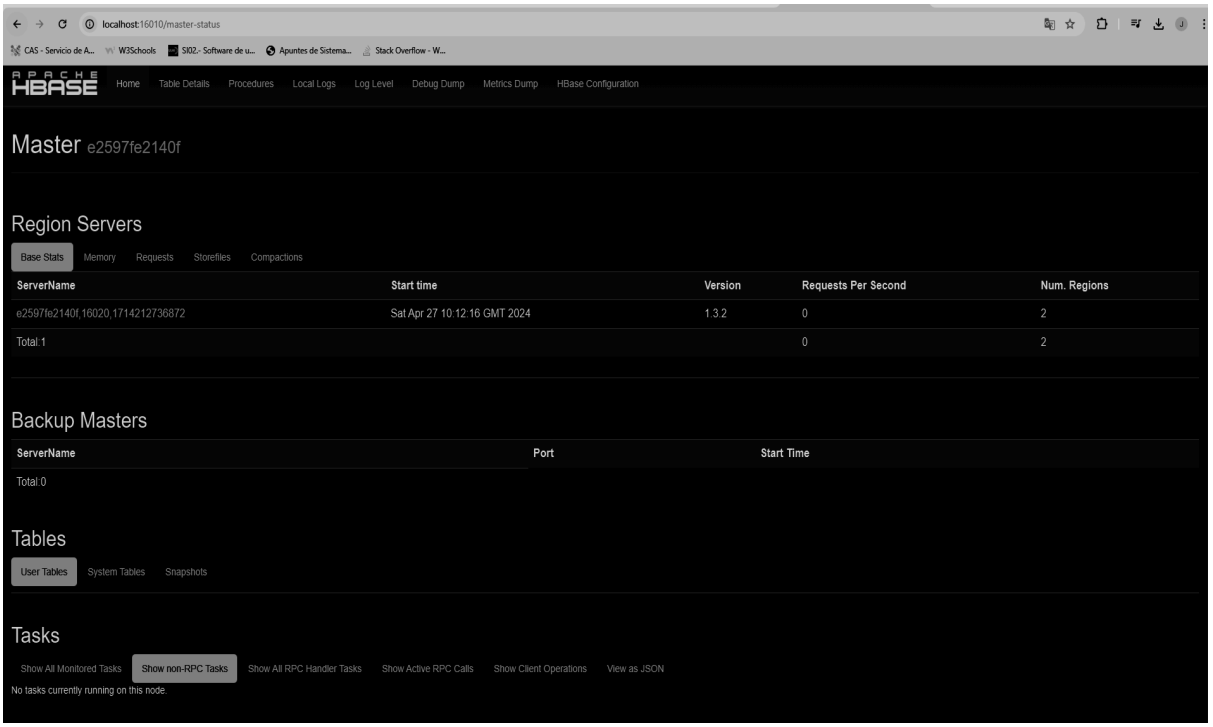


En la terminal se lanza el comando hbase shell para entrar en la terminal de la base de datos y poder trabajar con ella:



Utilizando el host indicado en el docker se abre una interfaz gráfica que nos da información sobre la base de datos:





Uso de comandos básicos de HBASE

1. Creamos una tabla con el comando create ‘TablaPrimera’, ‘columna1’, ‘columna2’
   Este comando crea una tabla con las columnas deseadas indicadas tras el nombre de la tabla.

```
hbase(main):001:0> create 'TablaPrimera', 'columna1', 'columna2'
0 row(s) in 1.3610 seconds
```

2. Lanzamos los comandos list y describe ‘TablaPrimera’. El primero de estos comando sirve para mostrar una lista con todas las tablas que existen en la base de datos y el segundo comando sirve para recibir información detallada de las columnas creadas en la tabla indicada:

```
=> Hbase::Table - TablaPrimera
hbase(main):002:0> list
TABLE
TablaPrimera
1 row(s) in 0.0140 seconds

=> ["TablaPrimera"]
hbase(main):003:0> describe 'TablaPrimera'
Table TablaPrimera is ENABLED
TablaPrimera
COLUMN FAMILIES DESCRIPTION
{NAME => 'columna1', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'N
ONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'columna2', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'N
ONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.0570 seconds
```

3. A continuación se añadirán valores a los campos de la tabla. Usamos el comando put ‘TablaPrimera’, ‘Id1’, ‘columna1:nombre’, ‘juan’. De esta forma estaremos añadiendo a la fila con el Id1 (Clave principal o identificador de línea) que añada en la primera columna (columna1) un campo llamado nombre con el valor juan. De esta forma estamos generando una rejilla dentro de columna1. Se repite el proceso para añadir otro campo más en columna1 y otros 2 campos más en columna2.

```
hbase(main):006:0> put 'TablaPrimera', 'Id1', 'columna1:nombre', 'juan'
0 row(s) in 0.0280 seconds

hbase(main):007:0> put 'TablaPrimera', 'Id1', 'columna1:apellidos', 'perez'
0 row(s) in 0.0030 seconds

hbase(main):008:0> put 'TablaPrimera', 'Id1', 'columna2:edad', '30'
0 row(s) in 0.0030 seconds

hbase(main):009:0> put 'TablaPrimera', 'Id1', 'columna2:direccion', 'calle desconocida'
0 row(s) in 0.0020 seconds
```

```
hbase> put 'ns1:t1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value'
hbase> put 't1', 'r1', 'c1', 'value', ts1
hbase> put 't1', 'r1', 'c1', 'value', {ATTRIBUTES=>{'mykey'=>'myvalue'}}
hbase> put 't1', 'r1', 'c1', 'value', ts1, {ATTRIBUTES=>{'mykey'=>'myvalue'}}
hbase> put 't1', 'r1', 'c1', 'value', ts1, {VISIBILITY=>'PRIVATE|SECRET'}
```

4. Tras esto mostraremos los valores almacenados en cada uno de los campos de la fila con Id1. Usamos el comando get ‘TablaPrimera’, ‘Id1’. De esta forma mostraremos todos los campos de las diferentes columnas de la fila. También se pueden añadir más argumentos enseñando solo la columna, campo o valor deseado:

```
hbase(main):013:0> get 'TablaPrimera', 'Id1'
COLUMN                          CELL
 columna1:apellidos             timestamp=1714215451791, value=perez
 columna1:nombre                timestamp=1714215431504, value=juan
 columna2:direccion             timestamp=1714215483802, value=calle desconocida
 columna2:edad                  timestamp=1714215468793, value=30
1 row(s) in 0.0140 seconds
```

```
hbase> t.get 'r1'
hbase> t.get 'r1', {TIMERANGE => [ts1, ts2]}
hbase> t.get 'r1', {COLUMN => 'c1'}
hbase> t.get 'r1', {COLUMN => ['c1', 'c2', 'c3']}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
hbase> t.get 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
hbase> t.get 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}
hbase> t.get 'r1', {FILTER => "ValueFilter(=, 'binary:abc')"}
hbase> t.get 'r1', 'c1'
hbase> t.get 'r1', 'c1', 'c2'
hbase> t.get 'r1', ['c1', 'c2']
hbase> t.get 'r1', {CONSISTENCY => 'TIMELINE'}
hbase> t.get 'r1', {CONSISTENCY => 'TIMELINE', REGION_REPLICA_ID => 1}
```

5. Se puede realizar una búsqueda por valor para que nos devuelva el la fila columna y campo en el que se encuentra. Se usa el comando scan 'TablaPrimera', { FILTER =>"ValueFilter(=, 'regexstring:juan')" } para hacer la búsqueda del valor juan devolviendonos la fila columna y campo donde se encuentra:

```
hbase(main):014:0> scan 'TablaPrimera', { FILTER =>"ValueFilter(=, 'regexstring:juan')" }
ROW                             COLUMN+CELL
 Id1                            column=columna1:nombre, timestamp=1714215431504, value=juan
1 row(s) in 0.0290 seconds
```

6. Realizaremos también un borrado de una celda en concreto. Usamos el comando
delete 'TablaPrimera', 'Id1', 'columna1:apellidos' para el borrado de esa celda y mostramos la información de la fila con el comando get:

```
hbase(main):003:0> delete 'TablaPrimera', 'Id1', 'columna1:apellidos'
0 row(s) in 0.0270 seconds

hbase(main):004:0> get 'TablaPrimera', 'Id1'
COLUMN                          CELL
 columna1:nombre                timestamp=1714215431504, value=juan
 columna2:direccion             timestamp=1714215483802, value=calle desconocida
 columna2:edad                  timestamp=1714215468793, value=30
1 row(s) in 0.0160 seconds
```

7. También se puede realizar un borrado de la fila completa. Usamos el comando
deleteall 'TablaPrimera', 'Id1' eliminando así toda la fila y mostramos la información de la fila mostrando que está vacía.

```
hbase(main):001:0> deleteall 'TablaPrimera', 'Id1'
0 row(s) in 0.1300 seconds
```

```
hbase(main):003:0> get 'TablaPrimera', 'Id1'
COLUMN                                CELL
0 row(s) in 0.0120 seconds
```

8. HBase también da una opción bastante útil que es deshabilitar la tabla sin tener que borrarla con el comando disable:

```
hbase(main):004:0> disable 'TablaPrimera'
0 row(s) in 2.2840 seconds
```

9. Para volver a activar o habilitar la tabla se usa el comando enable:

```
hbase(main):006:0> enable 'TablaPrimera'
0 row(s) in 1.2340 seconds
```

10. Y por último se puede borrar la tabla con el comando drop una vez deshabilitada la tabla, haciendo así más seguro el proceso de eliminado de la tabla evitando posibles eliminaciones por error

```
hbase(main):007:0> disable 'TablaPrimera'
0 row(s) in 2.2550 seconds

hbase(main):008:0> drop 'TablaPrimera'
0 row(s) in 1.2570 seconds

hbase(main):009:0> list
TABLE
0 row(s) in 0.0070 seconds

=> []
```

Bibliografía:

- https://www.ibm.com/es-es/topics/hbase
- https://hbase.apache.org/
- https://www.tutorialspoint.com/es/hbase/hbase_overview.htm
- https://www.youtube.com/watch?v=9YkurBN5Pt0
- https://www.tokioschool.com/noticias/hbase/
- https://aws.amazon.com/es/what-is/hadoop/
- https://www.josebernalte.com/tool/hbase/
- https://www.tutorialspoint.com/es/hbase/hbase_architecture.htm
- https://www.tutorialspoint.com/es/hbase/hbase_shell.htm
- https://docs.cloudera.com/runtime/7.2.17/accessing-hbase/topics/hbase-access-rest-api.html
- https://blog.cloudera.com/how-to-use-the-hbase-thrift-interface-part-1/
- https://docs.cloudera.com/cdp-private-cloud-base/7.1.9/managing-hbase/topics/hbase-metrics.html
- https://www.3pillarglobal.com/insights/blog-posts/hbase-coprocessors/