HADOOP

EXTRACIÓN, ALMACENAMIENTO Y ANÁLISIS DE TWEETS

Juan Casado Ballesteros

GitHub: https://github.com/JuanCasado/ADVANCED_DATABASES/tree/master/P3

ÍNDICE

INTRODUCCIÓN

Se desea construir una arquitectura flexible capaz de analizar grandes cantidades de datos. La característica principal de estos datos es que son producidos de forma constante y en cantidades masivas por lo que una arquitectura tradicional de almacenamiento no sería capaz de manejarlos.

ARQUITECTURA LAMBDA

Esto nos obliga a tener que adoptar una arquitectura propia del Big Data, la arquitectura Lambda que será implementada por medio de componentes del entorno Hadoop.

CAPA BATCH

Se utilizará Twitter como fuente de los datos. No obstante, tal y como se explicará esta fuente podría ser cambiada por otra de forma sencilla. Se utilizará Flume para realizar la extracción de los datos los cuales serán almacenados temporalmente sobre HDFS.

Las características de la capa batch es que los datos son procesados por lotes. Al utilizar HDFS como sistema de almacenamiento obtendremos ventajas adicionales como escalado, robustez y tolerancia a fallos.

CAPA DE SERVICIO

Los datos recolectados en bloques en la capa batch deben ser almacenados también en lotes en una base de datos que permita realizar consultas de forma rápida y eficaz. Las lecturas deben poder hacerse de forma aleatoria sobre esta base de datos, a diferencia de las escrituras que son en lotes.

Se utilizará Hbase, una base de datos columnar que se monta encima de HDFS. Esta base de datos hereda el escalado y la robustez de HDFS, permite lecturas aleatorias y escrituras en lotes y es sufrientemente flexible como para soportar variaciones en la estructura interna de los datos almacenados en ella.

Para transferir los datos de la capa batch a la capa de servicio se utilizará Pig. Esta herramienta utiliza la arquitectura de procesamiento mapreduce que se integra con HDFS por medio de Yarn para leer los datos almacenados en HDFS, procesarlos aplicando filtros sobre ellos y almacenándolos en Hbase.

Una vez que los datos han sido leídos de HDFS y almacenados en Hbase pueden ser borrados de HDFS.

Para realizar la práctica se ha utilizado Pig, pero otras arquitecturas de filtrado y procesamiento del entorno Hadoop podrían haberse utilizado en su lugar como Casacade también basada en mapreduce o Spark que propone su propia arquitectura de procesamiento.

CAPA EN TIEMPO REAL

El objetivo de la capa en tiempo real dentro de la arquitectura Lambda es proporcionar a la capa de acceso los datos que se están almacenando o procesando en las capas batch o de servicio. Mientras que el procesamiento en lotes sucede los datos dentro del lote todavía no están disponibles de modo que es necesario una capa incremental que los proporcione durante el periodo en el que el batch se está procesando.

Dentro del ecosistema Hadoop hay múltiples opciones para implementar esta capa, la más habitual es utilizar Kafka, un sistema distribuido de paso de mensajes que admite subscripción. Si esta capa se utilizara debería haber un sink de Flume que la populara de datos incrementalmente.

CAPA DE ACCESO

Acceder a los datos almacenados en Hbase no es la mejor solución dependiendo del uso que se vaya a dar a los datos. Para un acceso aleatorio a los datos como el que haría una aplicación Hbase puede ser una buena opción. No obstante, para analizar los datos, existen otras opciones mejores como Hive que proporcionan una interfaz similar a SQL integrada con mapreduce y Hbase para hacerlo.

TRABAJO REALIZADO

Se ha implementado una versión de la arquitectura Lambda en la que se ha eliminado la capa en tiempo real. Debido al uso que se dará a los tweets y a la forma en la que serán analizados esta no era necesaria. Si se hubiera requerido de proporcionar análisis en tiempo real sobre los tweets capturados esta capa hubiera tenido que ser implementada. No obstante, debido a que lo que se pretende es primero capturar los tweets y luego analizarlos no existe esta necesidad.

Los retos que abordar son extraer la gran cantidad de tweets que se generan de forma constante, filtrarlos y procesarlos de modo que nos quedemos solo con aquellos que sean de utilidad, almacenarlos con una estructura que nos permita analizarlos convenientemente y proporcionar un método rápido de acceso a toda esa información recolectada.

La instalación de los distintos componentes de Hadoop que forman la arquitectura Lambda mencionada se ha realizado sobre contenedores. Hacer esto permite un despliegue automatizado y escalable. Otras ventajas de este despliegue es la reducida dependencia entre los componentes utilizados. Cada uno de los bloques que forma parte de la arquitectura depende de los otros tan solo en las interfaces web que expone y no es la estructura de los directorios, de la instalación realizada o de las versiones de los componentes instalados.

La versión de todos los componentes utilizados es la última versión estable disponible: Hadoop-3.2.1, Hbase-2.2.4, Pig-0.17.0, Hive-3.1.2, Flume-1.7.0 y Zookeeper-3.4.10. Algunos de estos componentes no son plenamente compatibles con el resto debido a incompatibilidades en las librerías que utilizan o diferencias de configuración. No obstante, debido a que comparten una interfaz de comunicación estable y estandarizada se les ha podido hacer convivir gracias a su despliegue en contenedores.

HADOOP

La cantidad de datos que se producen diariamente crece de forma constante, los recursos necesarios para poderlos analizar son cada vez mayores y las arquitecturas de almacenamiento y procesamiento tradicionales no son capaces seguir este ritmo.

Son muchas las soluciones propuestas para solventar estos nuevos problemas. Algunas de estas soluciones se centran en resolver problemas concretos como almacenamiento distribuido (Big table, Cassandra, Ceph), acceso uniforme a los datos (Big query, Presto) o la computación distribuida y escalable (Kubernetes, Disco). Cada una de estas soluciones se adapta a resolver problemas concretos y lo hacen desde filosofías muy distintas, pero todas tien tres principios en común.

- **Escalado horizontal**: el escalado horizontal consiste en utilizar comodity hardware, es decir, equipos tradicionales que no tienen por qué pertenecer al mainframe, para aumentar la capacidad de cómputo. En lugar de utilizar un equipo más grande, caro y potente, se logra mayor potencia de cálculo por medio de la agregación de más equipos tradicionales.
- **Resistencia ante fallos**: cuantos más equipos se estén utilizando mayores serán las posibilidades de que uno falle. Si un equipo falla se debe evitar a toda costa perder información utilizando técnicas de replicación y de aseguramiento de la coherencia de los datos a lo largo de los equipos involucrados en almacenarla y procesarla.
- **Alto rendimiento**: un problema habitual en las arquitecturas tradicionales es que la capacidad de almacenamiento y la potencia de cómputo no crece linealmente con la cantidad de equipos pertenecientes al sistema. En las arquitecturas Big Data se pone especial cuidado en que añadir un equipo adicional al sistema suponga un aumento lineal en el rendimiento. De otro modo se entrarían ante un cuello de botella que limitaría su capacidad de procesar o almacenar datos alcanzada cierta cantidad de estos.

Hadoop es una solución más a todos estos problemas. Es un ecosistema rico y variado formado por múltiples aplicaciones que se integran a distintos niveles. Algunas de las características que permiten distinguir las aplicaciones que pertenecen o no al entorno Hadoop es que son Open Source y mantenidas por la plataforma Apache.

No obstante, y debido al gran crecimiento de Hadoop en los últimos años la línea entre lo que es Hadoop y lo que no es cada vez más difusa. Son más y más las aplicaciones que se integran en el ecosistema, algunas aplicaciones que originalmente formaron parte de él están empezando a quedar desactualizadas. Mientras que otras nunca quedaron claro si formaban parte o no como es el caso de Cassandra que a pesar de ser desarrollado por Apache según la fuente es considerado o no parte de Hadoop, pues no se basa en HDFS.

DISTRIBUCIONES

Hadoop puede instalarse de forma nativa sobre Windows o Linux, en una máquina virtual o sobre contenedores. En todos esos casos se corre el gran riesgo de tener problemas con la instalación o problemas de compatibilidad entre sus componentes.

Esto se debe a que en ocasiones los productos Open Source pueden divergir en las interfaces por las que se comunican, en las tecnologías que utilizan o simplemente no ser actualizados de forma homogénea.

Para suplir estas carencias múltiples empresas como Cloudera y Hortonworks, recientemente fusionadas o mapR sirven distribuciones de Hadoop con diversos componentes ya instalados y listos para usarse con garantías de que funcionarán adecuadamente. Las ventajas de utilizar una de estas distribuciones se ve ampliada con soporte técnico, cursos y certificaciones para empleados, tutoriales, ejemplos y blogs donde poder preguntar dudas.

Las distribuciones de Hadoop también proporcionan componentes adicionales que enriquecen el entorno como entornos gráficos desde los que poder configurar y extender la instalación, visualizar su estado, así como componentes para extenderla como la base de datos Impala perteneciente al entorno de Cloudera.

ARQUITECTURA

Se elijan los componentes que se elijan dentro del entorno de Hadoop existe una arquitectura base sobre la que estos puede ser instalados. Esta arquitectura tiene en su base el sistema de almacenamiento distribuido HDFS. Este sistema de almacenamiento proporciona la robustez y la tolerancia a fallos de Hadoop a lo largo de cualquier cantidad de nodos.

Las aplicaciones pueden instalarse directamente sobre HDFS sin necesitar componentes adicionales como es el caso de Hbase o Flume.

Por encima de HDFS se encuentra Yarn que es un gestor de recursos. Su misión es proporcionar una interfaz del sistema de almacenamiento suficientemente general como para instalar entornos de cómputo distribuido. Algunos de estos entornos son Spark, Mapreduce o Tez.

Mapreduce es el entorno de cómputo original de Hadoop, de echo en un origen este se montaba directamente sobre HDFS sin posibilidad de utilizar otros entornos de cómputo. En la actualidad Mapreduce está perdiendo popularidad frente a Tez que ya es recomendado frente a Mapreduce en múltiples aplicaciones como Hive o Pig, por el contrario, Spark proporciona su propio mini entorno por encima de Hadoop no sin ello perder la compatibilidad con esas mismas aplicaciones gracias a la interfaz de Yarn desde la que se pueden comunicar.

Al margen del sistema de almacenamiento y del entorno de cómputo dentro de Hadoop han surgido múltiples aplicaciones en las que concentrar servicios como es el caso de Zookeeper el cual es un gestor distribuido para almacenar nombres y configuraciones con la intención de sincronizar servicios distribuídos. Zookeeper

por ejemplo implementa el servicio de sincronización entre el master activo de Hbase y sus regiones. Otro ejemplo bajo esta misma filosofía es Kafka que proporciona un context broker basado en streams distribuídos y en paso de mensajes que admite subscripciones con notificación en tiempo real.

Estas aplicaciones como Kafka o Zookeeper son también utilizadas en entornos fuera del ecosistema de Hadoop.

NUEVOS RETOS

Hadoop aunque es una gran solución dentro del entorno del Big Data presenta dos grandes debilidades o carencias, ambas notablemente experimentadas a lo largo de la realización de esta práctica.

HADOOP SAS

Muchas bases de datos como Firebase, Big Table o MongoDB ofrecen servicios SAS para acceder a ellas. Con esto se logran evitar problemas de instalación y de mantenimiento. La mayoría de los componentes de Hadoop como Hbase, Hive o Kafka podrían seguir modelos similares. Resulta extraño ver como otros productos similares, son comercializados de esta forma mientras que Hadoop no lo es.

ESCALADO AUTOMÁTICO

Algunos de los componentes de Hadoop admiten escalado dinámico como es el caso de los nodos de almacenamiento y los nodos de procesamiento en HDFS. Otras partes de Hadoop por el contrario solo admiten escalado estático como es el caso de las regiones de Habse o el quorum de Zookeeper.

Actualmente servicios como las rolling updates o el escalado automático son una parte esencial de los sistemas de orquestación como Kubernetes o Mesos. Estos servicios son la base de los centros de computación cloud pues permiten a las bases de datos y a los nodos de procesamiento adaptarse de forma dinámica y en tiempo real a las cargas de trabajo a las que sean cometidos.

FLUME

Flume es una herramienta de ingestión de datos. Su misión es tomar datos de una fuente concreta y dejarlos en otra. Flume es compatible con gran cantidad de fuentes de datos como las APIs que las redes sociales ponen a disposición de los usuarios, bases de datos o archivos almacenados en HDFS u otros sistemas de almacenamiento.

En nuestro caso utilizaremos como fuente de datos la API de Twiter pero modificando la configuración de flume se podrían obtener datos de cualquiera de las otras posibles fuentes mencionadas. Posteriormente esos datos serán depositados en HDFS. Podrían depositarse en otros destinos como Kafka o directamente sobre Hbase.

INSTALACIÓN

Para extraer los datos de Twitter se utilizará un programa escrito en Java cuya misión es permitir aplicar filtros sobre la fuente de Flume. Estos filtros son opcionales y pueden restringir la ingestión de datos dentro de un área geográfica o haciendo que contengan solo ciertas palabras. Para que Flume pueda utilizar esta fuente de datos se ha compilado el código junto con la librería Twitter4j en un .jar que se ha dejado junto al resto de librerías con las que Flume ya viene equipado.

Como salida de los datos se ha utilizado HDFS y como canal se ha utilizado MemChanel. Con esto quedan cubiertos los tres componentes de los que se compone un agente de Flume: fuente, canal y salida.

Esta configuración del agente se escribe en un archivo .conf cuya única peculiaridad es que ya que el namenode de HDFS es remoto se debe indicar su puerto junto a la dirección de HDFS, en nuestro caso:

hdfs://namenode:9000/ruta/donde/dejar/los/archivos.

Las dependencias internas de Flume consisten en hadoop-client que debe ser el mismo que el que esté instalado en el namenode y de Zookeeper-3.5.2. La versión de Zookeeper 3.4.10 que es la expuesta externamente es compatible con el uso que hace Flume con la 3.5.2.

Con respecto al uso de red Flume no necesita ser accesible desde el exterior, no obstante, requiere que el namenode en sus dos interfaces, el nodemanager y al menos un Zookeeper y un datanode estén disponibles.

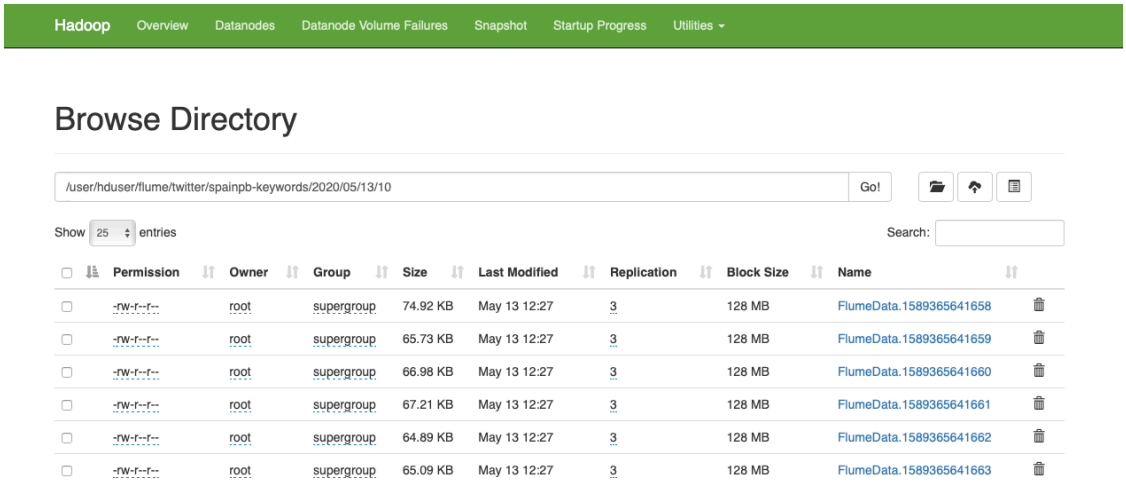
EJEMPLO

Finalmente, el agente de Flume puede ser lanzado con:

flume-ng agent –conf conf -f /archivo.conf -n NombreDelAgente

Algunas limitaciones de utilizar Flume para esta tarea es que la API de Twitter en su modo de uso gratuito impone limitaciones en la cantidad de agentes disponibles con una misma acreditación y en el tiempo en el que el agente puede estar recolectando los datos. De modo que solo podrá haber un agente activo extrayendo datos de la API y este sufrirá un error cuando lleve cierto tiempo en ejecución.

Se puede ver desde el explorador de ficheros web de Hadoop como Flume escribe los datos que recolecta en distintos ficheros de modo que la recolección sea lo más eficiente posible:



HDFS

HDFS es el Sistema de archivos distribuido de Hadoop. A pesar de ser un sistema de archivos distribuido pensado para ser ejecutado en múltiples nodos sobre múltiples máquinas también ofrece un modo de ejecución local no distribuido.

Las ventajas de utilizar este sistema de archivos es que proporciona seguridad ante fallos debido a redundancias en el almacenamiento, así como alta disponibilidad gracias a la posibilidad de duplicar recursos y procesos.

HDFS tiene cuatro componentes principales:

- **Nodo de nombres**: el nodo de nombres es el nodo maestro de HDFS. En él se registran la ubicación de los datos. Este nodo es el encargado de coordinar la replicación de la información. Solo podrá haber una instancia activa de este nodo, no obstante, podrá haber otras inactivas para sustituirlo en caso de fallo.
- **Nodos de datos**: este nodo es el encargado de almacenar la información en disco. Los datos almacenados en un nodo de datos deben registrarse sobre el nodo de nombre para que otras aplicaciones puedan encontrarlos y para que puedan ser replicados por otros nodos de datos para protegerlos en caso de fallo. Podrá haber múltiples instancias activas de los nodos de datos.
- **Nodos de recursos**: el nodo de recursos es el encargado de ejecutar YARN. Este es el que nos permitirá acceder a los datos para su explotación. También hará de interfaz entre los entornos de cómputo como Mapreduce que es ejecutado también por estos nodos u otros como Spark que serían ejecutados en nodos externos.
- **Nodo de histórico**: la misión de este nodo es registrar el estado actual y pasado de los accesos masivos a datos. Por ejemplo, cuando un proceso mapreduce sea iniciado deberá registrarse en este nodo y le deberá indicar de su proceso y estado en todo momento.

Dependiendo de la cantidad de nodos activos y de las características de la acción que se deba realizar el acceso a los nodos no será directo si no que deberá de ser coordinado por al menos un Zookeeper.

INSTALACIÓN

En nuestro caso cada tipo de nodo es aislado del resto en su propio contenedor y solo se les permite comunicarse entre ellos o con otras aplicaciones por medio de sus interfaces remotas. En cada contenedor se levantará el daemon correspondiente a cada nodo concreto. Esto tiene grandes ventajas como aislar los errores de ejecución o permitir a múltiples versiones de Hadoop coexistir.

La instalación de Hadoop se reduce a descargarlo, descargar Java-8, crear las variables de entorno JAVA_HOME, HADOOP_HOME y HADOOP_CONF_DIR. En la configuración de Hadoop deberemos tener especial cuidado en referenciar a los notos por su dirección y puerto del contenedor en el que se estén ejecutando.

EJEMPLO

Como hemos mostrado en el ejemplo de Flume los nodos exponen interfaces web para permitir a los usuarios y administradores ver su estado.

Nos conectamos a un nodo de Hadoop:

docker exec -it NODE bash

Visualizamos el contenido de un archivo con Tweets:

hadoop fs -cat /user/hduser/flume/twitter/ruta/al/archive


{"in_reply_to_status_id_str":"1260502576585486341","in_reply_to_status_id":1260502576585486341,"created_at":"Wed May 1
3 10:27:20 +0000 2020","in_reply_to_user_id_str":"1201541967425024000","source":"<a href=\"http://twitter.com/download
/android\" rel=\"nofollow\">Twitter for Android","retweet_count":0,"retweeted":false,"geo":null,"filter_level":"l
ow","in_reply_to_screen_name":"GandalfVago","is_quote_status":false,"id_str":"1260516972401512454","in_reply_to_user_i
d":1201541967425024000,"favorite_count":0,"id":1260516972401512454,"text":"@GandalfVago @Alvisepf @iescolar @IdiazAyus
o @PabloIglesias @PODEMOS @CajaIngenieros @policia Jarabe democr?tico. Lo inventaron ustedes.","place":null,"lang":"es
","quote_count":0,"favorited":false,"coordinates":null,"truncated":false,"timestamp_ms":"1589365640150","reply_count":
0,"entities":{"urls":[],"hashtags":[],"user_mentions":[{"indices":[0,12],"screen_name":"GandalfVago","id_str":"1201541
967425024000","name":"Gandalf El Vago","id":1201541967425024000},{"indices":[13,22],"screen_name":"Alvisepf","id_str":
"262822127","name":"Alvise P?rez","id":262822127},{"indices":[23,32],"screen_name":"iescolar","id_str":"14436317","nam
e":"Ignacio Escolar","id":14436317},{"indices":[33,44],"screen_name":"IdiazAyuso","id_str":"140034129","name":"Isabel
D?az Ayuso","id":140034129},{"indices":[45,59],"screen_name":"PabloIglesias","id_str":"158342368","name":"Pablo Iglesi
as ?","id":158342368},{"indices":[60,68],"screen_name":"PODEMOS","id_str":"2288138575","name":"PODEMOS","id":228813857
5},{"indices":[69,84],"screen_name":"CajaIngenieros","id_str":"404996892","name":"Caja de Ingenieros","id":404996892},
{"indices":[85,93],"screen_name":"policia","id_str":"23791197","name":"Polic?a Nacional","id":23791197}],"symbols":[]}
,"display_text_range":[94,136],"contributors":null,"user":{"utc_offset":null,"friends_count":3162,"profile_image_url_h
ttps":"https://pbs.twimg.com/profile_images/1213542651871612930/pD3RSG9P_normal.jpg","listed_count":2,"profile_backgro
und_image_url":"","default_profile_image":false,"favourites_count":21138,"description":"Doctor en Ciencias Biol?gicas.
 Luchar? incansable hasta acabar con la doctrina socialista generadora de miseria, paro y epidemias. ????????","create
d_at":"Sat Jan 04 08:04:12 +0000 2020","is_translator":false,"profile_background_image_url_https":"","protected":false
,"screen_name":"DoctoreGreen","id_str":"1213370485678858240","profile_link_color":"1DA1F2","translator_type":"none","i
d":1213370485678858240,"geo_enabled":false,"profile_background_color":"F5F8FA","lang":null,"profile_sidebar_border_col
or":"C0DEED","profile_text_color":"333333","verified":false,"profile_image_url":"http://pbs.twimg.com/profile_images/1
213542651871612930/pD3RSG9P_normal.jpg","time_zone":null,"url":null,"contributors_enabled":false,"profile_background_t
ile":false,"profile_banner_url":"https://pbs.twimg.com/profile_banners/1213370485678858240/1588273059","statuses_count
":7223,"follow_request_sent":null,"followers_count":1956,"profile_use_background_image":true,"default_profile":true,"f
ollowing":null,"name":"Doctor Green ????","location":null,"profile_sidebar_fill_color":"DDEEF6","notifications":null}}

Podemos ver los Tweets almacenados en formato JSON.

Copiamos un archivo del sistema local de archivos a una carpeta nueva en hdfs:

hadoop fs -mkdir /input

hadoop fs -copyFromLocal /student.txt /input

Lanzamos uno de los programas de ejemplo de Hadoop que cuenta las veces que se repiten palabras en los archivos:

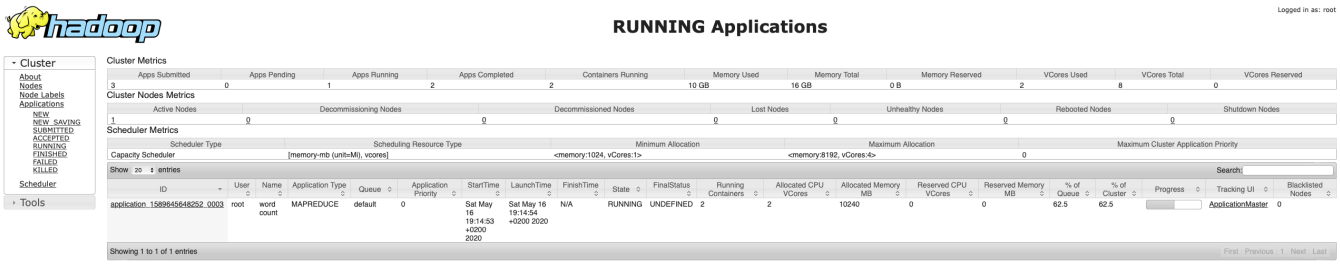
docker-compose -f docker-compose-data.yml run example

hadoop fs -cat /output/file

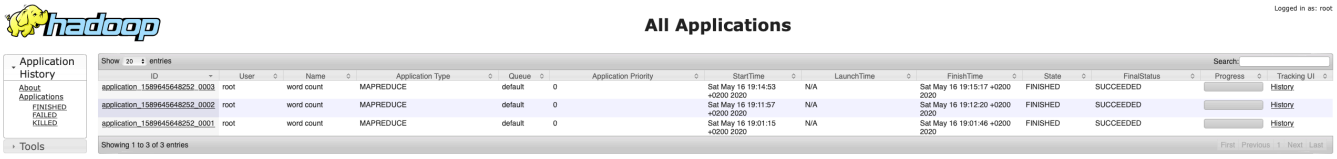
Como resultado obtendremos un archivo que contendrá el recuento de las apariciones de todas las palabras incluidas en el archivo.

```
goverenment     455
hoover  12
huntsville      15
hutsville       1
louisiana       12
madison 348
oregon  16
prattville      14
private 13
qalification    1
randomName      1
score   1
sector  1
state   1
tuscaloosa      10
vermont 14
```

Mientras que el proceso mapreduce está en proceso de ejecución podremos ver su estado y progreso en la interfaz web del nodo de recursos.



También podremos ver el histórico de procesos yarn desde el nodo de histórico.



HBASE

Hbase es la base de datos de referencia en el entorno Hadoop. Es una base de datos columnar noSQL. Las bases de datos columnares se centran en aumentar el rendimiento de accesos a bloques semi-homogéneos de datos. Es decir, alcanzan su máximo rendimiento en lectura cuando se accede columna entera de datos. Por el contrario, las escrituras son más eficientes cuando se realizan en bloques, es decir, se rellenan múltiples filas de todas las columnas de una tabla a la vez.

Hbase es además una base de datos distribuida basada en regiones. Existirá un nodo maestro con múltiples nodos esclavos o regiones conectados a él. La integridad de las lecturas no está garantizada por la propia base de datos si no por el quorum. El quorum consiste en un sistema de coordinación proporcionado por medio de un Zookeeper para cada región de Hbase que permite que estas acuerden un valor fijo para cada elemento almacenado de forma global con independencia del que tengan almacenado ellas.

Adicionalmente esta base de datos permite una configuración altamente granular y de muy bajo nivel sobre la forma en la que los datos se repartirán entre las distintas regiones. Esto mismo sucede con la configuración de las cachés o con los filtros de Bloom que permiten agilizar las consultas evitando accesos a disco o a regiones en las que no se encuentren los datos que se buscan.

La integración de Hbase con HDFS es muy elevada. Este es el sistema de archivos en el que almacena la información que contiene y gracias a él obtiene su robustez y tolerancia a fallos. Adicionalmente se integra con YARN y en especial con mapreduce para permitir accesos a sus datos.

Una característica a tener en cuenta sobre Hbase es que no admite borrado. Los datos cuando se eliminan son marcados como invisibles sin llegar a desaparecer del disco de modo que no son recomendables. Lo mismo sucede con otras bases de datos similares como Cassanda.

INSTALACIÓN

La instalación de Hbase al igual que la de Hadoop es muy simple, se reduce a descargarla, configurar las variables de entorno JAVA_HOME, HBASE_HOME y HBASE_CONF_DIR. En la última ruta se debe crear el archivo hbase-site.xml donde deberemos poner especial atención a la correcta configuración del quorum y de las regiones.

Una peculiaridad de esta configuración es que es estática y no puede modificarse sin apagar el servicio. Esto limita en gran medida el potencial de Hbase para escalar de forma automática o recibir Rolling updates.

Una vez configurado todo podremos iniciar Hbase como mestro:

hbase-daemon.sh master start

O como región:

hbase-daemon.sh regionserver start

O proporcionar acceso a sus otras interfaces:

hbase-daemon.sh start thirf

hbase-daemon.sh start rest

EJEMPLO

En total podremos acceder a la información almacenada en Hbase de tres formas distintas. Por medio de la interfaz Thirf en el puerto 9090, por medio de la interfaz HTTP en el puerto 8081 o por medio de la interfaz grpc en el puerto 16010.

Las aplicaciones propias de Hadoop suelen utilizar la interfaz grpc. En caso de querer utilizar múltiples versiones de Hadoop con un mismo Hbase debemos garantizar que su interfaz grpc se idéntica lo cual en gran parte depende de la versión de la librería de Guava que estemos utilizando.

El resto de las aplicaciones como happybase suelen preferir la interfaz Thirf aunque si solo deseamos realizar un número reducido de acciones podremos hacerlo por la interfaz HTTP como hace la librería de hbase para NodeJS.

Un problema de estas interfaces y por ende de estas librerías es que no son capaces de conectarse con Hbase por medio del quorum de modo que no podrán ser utilizadas cuando las regiones estén activadas. Es por ello por lo que el ejemplo lo haremos utilizando la interfaz de consola de Hbase.

Nos conectamos a Hbase

docker exec -it hbase-master hbase shell

Creamos una tabla nueva con tres familias de columnas

create 'student', 'personal', 'scholar', 'residence'

```
Created table student
Took 4.1011 seconds
=> Hbase::Table - student
```

Añadimos manualmente una fila de datos indicando la tabla, la fila, la familia de columnas, una columna concreta dentro de la familia y una marca de tiempo.

put 'student', 'student1', 'personal:StudentName', 'Pepe', 10

put 'student', 'student1', 'personal:sector', 'student', 10

put 'student', 'student1', 'personal:DOB', '10-05-1998', 10

put 'student', 'student1', 'scholar:qualification', '10', 10

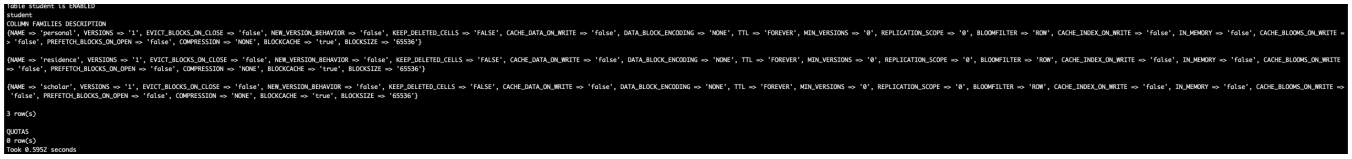
put 'student', 'student1', 'scholar:score', 'UAH', 10

put 'student', 'student1', 'residence:state', 'Spain', 10

put 'student', 'student1', 'residence:randomName', 'Madrid', 10

Podemos ver la descripción de la tabla.

describe 'student'



Podemos ver todos los datos insertados en la tabla, todos pertenecen a la misma fila, pero están en columnas distintas.

scan 'student'

```
ROW                                                    COLUMN+CELL
 student1                                              column=personal:StudentName, timestamp=10, value=Pepe
 student1                                              column=personal:sector, timestamp=10, value=student
 student1                                              column=residence:randomName, timestamp=10, value=Madrid
 student1                                              column=residence:state, timestamp=10, value=Spain
 student1                                              column=scholar:qualification, timestamp=10, value=10
 student1                                              column=scholar:score, timestamp=10, value=UAH
1 row(s)
Took 0.0910 seconds
```

Podemos extraer los datos de una fila concreta obteniendo todos sus valores por columnas.

get 'student', 'student1'

```
COLUMN                                                 CELL
 personal:StudentName                                  timestamp=10, value=Pepe
 personal:sector                                       timestamp=10, value=student
 residence:randomName                                  timestamp=10, value=Madrid
 residence:state                                       timestamp=10, value=Spain
 scholar:qualification                                 timestamp=10, value=10
 scholar:score                                         timestamp=10, value=UAH
1 row(s)
Took 0.0840 seconds
```

También se pueden borrar y modificar datos siendo preferable esto último. Como vemos las operaciones posibles sobre Hbase son muy reducidas no existiendo Joins o Uniones u otras operaciones propias de las bases de datos SQL.

PIG

Pig es una aplicación del entorno Hadoop cuya misión es permitir interactuar con mapreduce u otras plataformas de ejecución de una forma sencilla. Permite escribir programas en el lenguaje Pig latin el cual está pensado para trabajar con datos de estructura flexible.

Los programas escritos para Pig son ejecutados de forma relajada. Esto quiere decir que las operaciones no se llegan a ejecutar hasta que no es necesario. Por ejemplo, si indicamos a Pig que queremos leer datos de una fuente no hará nada, si posteriormente le indicamos que aplique un filtro sobre ellos tampoco hará nada, pero si finalmente le decimos que los muestre por pantalla será entonces y solo entonces cuando los lea, filtre y muestre.

La razón de la forma en la que Pig tiene este tipo de ejecución es poder crear un grafo de los pasos que debe realizar para poder optimizarlo. Esto le permite crear trabajos mapreduce altamente eficientes.

Otra gran ventaja de usar esta herramienta es la facilidad para leer y manipular datos de múltiples fuentes, así como de inyectarle código y funciones adicionales desde JAVA como hemos hecho en esta práctica.

INSTALACIÓN

Quizá el principal problema de Pig es que tiene un mantenimiento y ritmo de actualizaciones menor que otras aplicaciones del entorno Hadoop. Esto ha causado problemas a la hora de instalarlo.

Para poder completar la instalación ha sido necesario buscar una versión de Hadoop que fuera compatible con Pig-0.17 que es la última versión disponible y a la vez con una versión de Hbase que fuera compatible con este Pig. Por otro lado, ambas versiones deberían de ser compatibles con las interfaces de los Hadoop instalados fuera del contenedor. Finalmente, las versiones elegidas han sido

Hadoop-2.6.0 y Hbase-1.6.0 de las cuales se utilizan sus librerías solo dentro de este contenedor.

Adicionalmente se ha instalado la librería elephant-bird-4.17 que junto al código proporcionado para manipular dos lados de Twitter se deberán añadir compiladas al CLASSPATH.

Una vez hecho esto tendremos un Pig listo para interactuar con el resto de la arquitectura y en especial para interactuar con Hbase.

EJEMPLO

Como ejemplo se cargarán datos de un archivo y se introducirán en la base de datos Hbase. Este ejemplo es una versión simplificada de lo que se hará posteriormente con los Tweets, pero permite comprobar que la instalación de Pig es adecuada y que este puede comunicarse con Hbase adecuadamente.

Habiendo modificado el archivo docker-compose-data.yml para entrar en el contenedor sobre la terminal bash

docker-compose -f docker-compose-data.yml run pig

Copiamos el archivo local a hdfs para que luego sea accesible desde pig

hadoop fs -copyFromLocal /scripts/student.txt /pig-demo/student.txt

Lanzamos la interfaz interactive de pig

pig -x local

Cargamos los datos del archivo, la instrucción se lee, pero no se ejecuta.

students = LOAD 'hdfs://namenode:9000/pig-demo/student.txt' USING PigStorage(',') AS (StudentName:chararray, sector:chararray, DOB:chararray, qualifications:chararray, score:int, state:chararray, randomName:chararray);

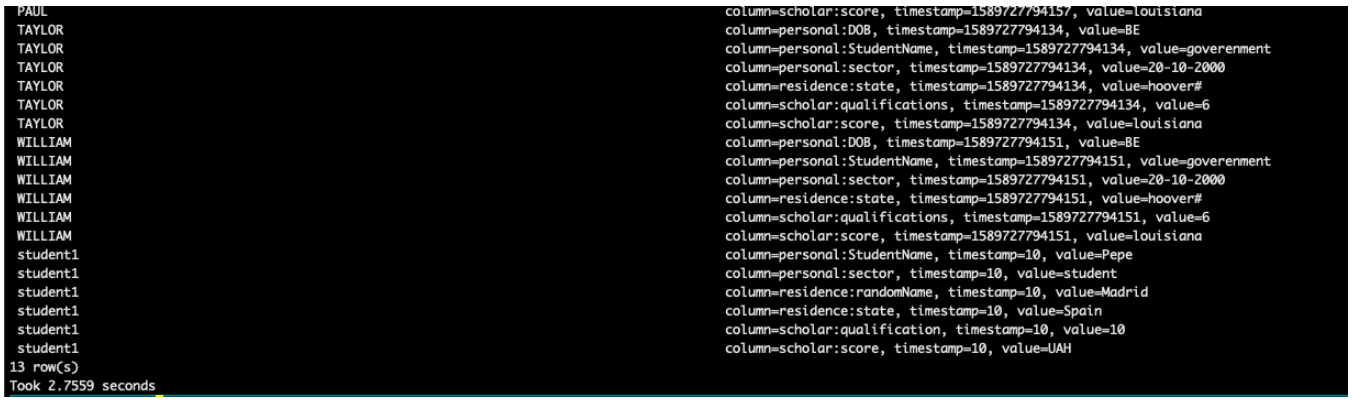
Filtramos los datos para quedarnos solo con los estudiantes de lousiana que como vimos en la demo de HDFS son 12. Esta sentencia instrucción se ejecuta.

students = filter students by state matches ‘louisiana’;

Guardamos los datos de los estudiantes en Hbase. Solo al ejecutar esta instrucción se ejecutarán todas las anteriores.

store students into 'hbase://student' using org.apache.pig.backend.hadoop.hbase.HBaseStorage('personal:StudentName, personal:sector, personal:DOB, scholar:qualifications, scholar:score, residence:state, residence:randomName');

Desde Hbase podemos ver que los datos han sido cargados correctamente.



En total tenemos 13 estudiantes como esperábamos, el que cargamos en la demo de Hbase y los 12 nuevos.

HIVE

Hive es una base de datos de explotación o data warehouse. Su misión es proporcionar una forma de acceso uniforme a una o varias bases de datos distribuidas por medio de una sintaxis similar a SQL.

Dependiendo del tipo de consulta realizada esta podrá ser resuelta por medio de accesos directos a la base de datos correspondiente o por medio del entorno de ejecución elegido, por defecto mapreduce.

Hive tiene dos características que la hacen una base de datos ideal para la explotación. Permite crear una interfaz rígida encima de datos cuya estructura interna sea flexible. Los datos almacenados en bases de datos noSQL cuya estructura puede ser más laxa se podrá convertir en una estructura uniforme cuando sean importados en Hive. Adicionalmente Hive almacena en un metastore o base de datos auxiliar información relativa a la estructura de los datos en las bases de datos externas a las que accede. Esto le permite aprender sobre dichos datos y poder generar consultas más optimizadas a la hora de explotarlos.

En nuestro caso utilizaremos Hive para manejar las tablas creadas en Hbase. Cuando creemos las tablas de explotación en Hive solo tendremos que indicar que estas se corresponden con tablas ya existentes en otra base de datos.

INSTALACIÓN

Para la instalación de Hive necesitaremos configurar Hadoop, Java y las variables de entorno del mismo modo que para el resto de las intalaciones.

Lo único excepcional de esta instalación es que la versión de grpc incluida por defecto en la última versión de Hive no es compatible con la interfaz expuesta por la última versión de Hadoop, por suerte esto es fácilmente solucionable eliminando la versión 19 de guava e instalando la que esté incluida en ese Hadoop, en este caso la 27. Adicionalmente deberemos actualizar la versión de Zookeeper y añadir

las librerías de cliente de Hbase para que la comunicación con el quorum pueda producirse y podamos replicar tablas de Hbase.

Finalmente debemos iniciar las instancias de Hive, el master:

hive –service hiveserver2

y el metastore:

hive –service metastore

el cual además deberemos conectar con una base de datos SQL en la que deberemos de crear las tablas que Hive necesita para almacenar sus metadatos.

EJEMPLO

Como ejemplo crearemos una tabla externa que haga referencia a la tabla de estudiantes de Hbase.

Nos conectamos a Hive

docker exec -it hive-server hive

Creamos la tabla externa

create external table student (key string, studentName string, sector string, DOB string, qualifications string, score string, state string, randomName string) stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with serdeproperties('hbase.columns.mapping'= ':key, personal:StudentName, personal:sector, personal:DOB, scholar:qualifications, scholar:score, residence:state, residence:randomName') tblproperties('hbase.table.name'='student');

Ahora podremos realizer consultas utilizando la sintaxis HQL que es muy similar a SQL sobre los datos que teníamos antes.

select * from student

```
BENJAMIN        goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
BLOOM   goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
BURD    goverenment     20-10-2000      ME      6       louisiana       hoover# NULL
CHAMBERS        goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
GEORGE  goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
HEFFLEY goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
HUMBERT goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
NISBET  goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
OLINGER goverenment     01-01-2003      BE      6       louisiana       hoover# NULL
PAUL    goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
TAYLOR  goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
WILLIAM goverenment     20-10-2000      BE      6       louisiana       hoover# NULL
student1        Pepe    student NULL    NULL    UAH     Spain   Madrid
Time taken: 0.475 seconds, Fetched: 13 row(s)
```

Podemos ver que mientras que a Hbase no le importaba que el formato de los datos fuera distinto al cargar estos sobre Hive obtendremos valores NULL sonde la estructura de la información de Hbase que es muy flexible no coincida con la de Hive que es mucho más rigida.

select StudentName from student where score!='louisiana';

```
Query ID = root_20200517152452_6691311e-c8cb-42ff-a710-ad53bc23507d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2020-05-17 15:24:55,027 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_local1175846064_0003
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Pepe
Time taken: 2.53 seconds, Fetched: 1 row(s)
```

DESPLIEGUE

Se proporcionan dos despliegues distintos ambos basados en contenedores. Ambos despliegues se basan en los mismos principios con ligeras diferencias a la hora de llevarlos a cabo.

Los objetivos de un buen despliegue que utilice contenedores son ser declarativo y aislar los servicios. Un despliegue declarativo es aquel en el que se indica lo que se desea obtener sin necesidad de indicar como debe obtenerse. Esto es apreciable en el uso de Docker en el sentido de que se trabaja con máquinas completas como si fueran servicios. Las dependencias son conocidas, el código está instalado y solo debemos indicar qué deseamos obtener sin tener que decir como.

Podemos realizar acciones como:

docker-compose -f docker-compose-data.yml run pig

echo"create'mentions','f'"|docker exec -i hbase-master hbase shell

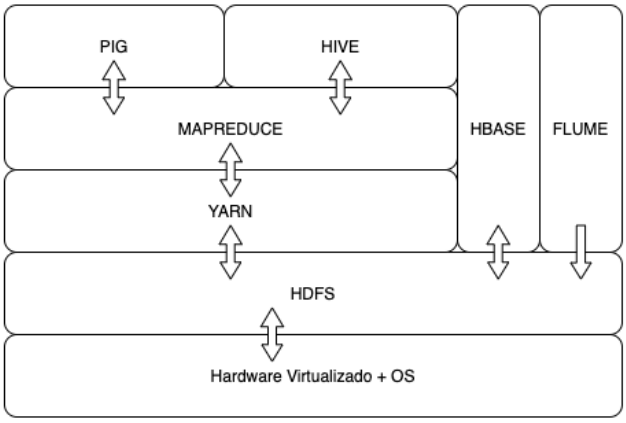
En las que indicamos qué deseamos que suceda ejecutar pig o crear una tabla sin indicar expresamente todo lo que ello conlleva, conexiones de red, descargas d archivos, configuraciones de instalación pues estas o ya han sido realizadas o se realizarán de forma automáticamente en base a otras declaraciones previas.

Por otro lado, el aislamiento de servicios hace referencia a que cada máquina en ejecución o contenedor haga una y solo una cosa. Esto permite separar cada deamon de HDFS en su propia burbuja sin que el resto de los contenedores sepan o necesiten saber cómo funciona. Lo único que una a los contenedores son los puertos de red que exponen.

Esto permite hacer cosas como utilizar distintas versiones del mismo componente lo cual en esta instalación ha sido necesario en múltiples ocasiones para poder utilizar versiones en principio no compatibles de algunos de ellos.

ARQUITECTURA

La arquitectura base del despliegue realizado puede visualizarse de la siguinte forma:



Tendremos un HW y un OS virtualizados con una instancia de la JVM instalada sobre ellos. Encima de esto se situará el sistema de archivos HDFS que es el úcleo de Hadoop. Directamente conectado al sistema de archivos se encuentra Hbase que puede comunicarse con él de forma bidireccional y Flume que lo hace solo escribiendo en él.

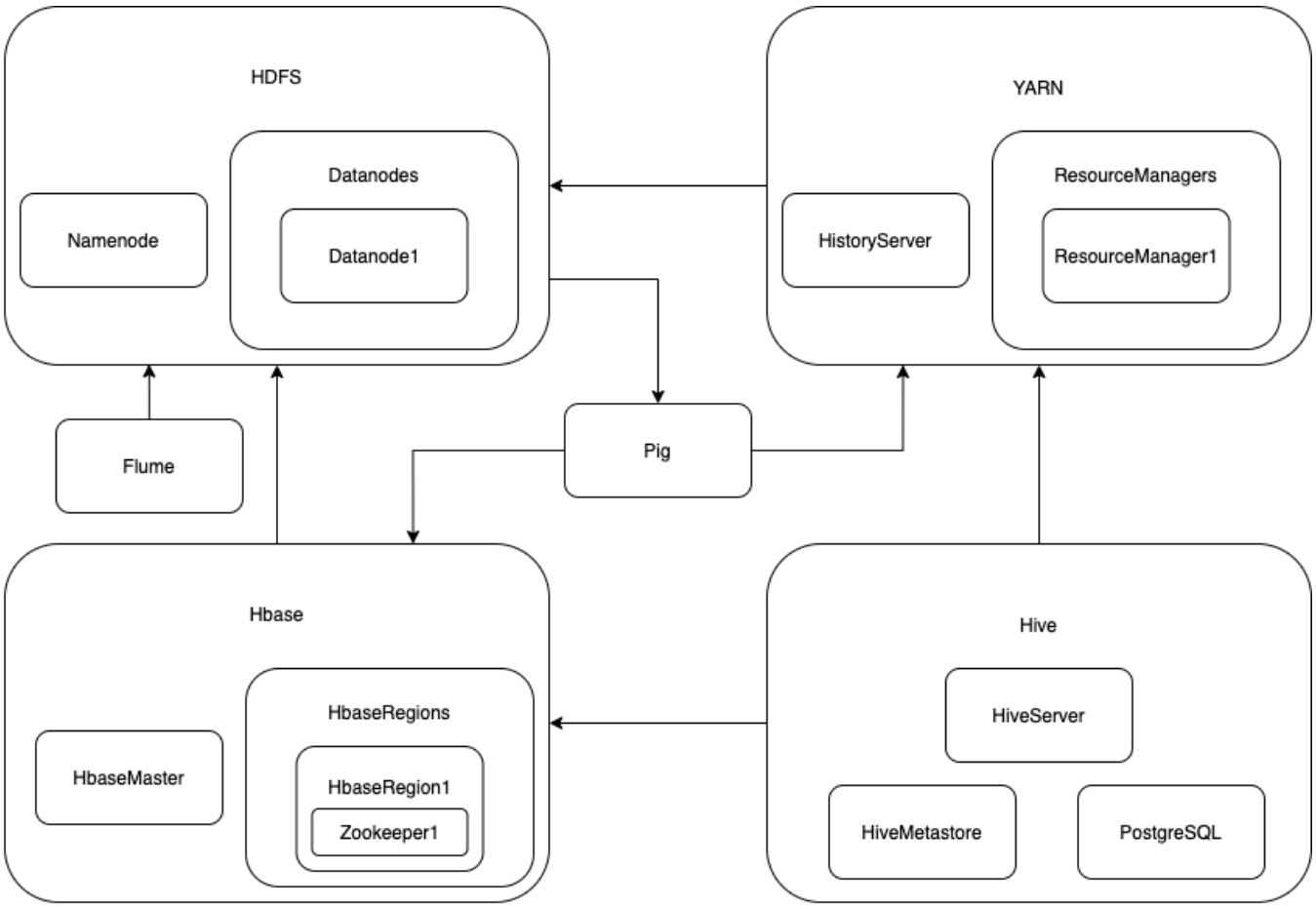
Por encima del sistema de archivos se encuentra YARN que es el gestor de recursos. En este caso solo hay instalado un entorno de cómputo, mapreduce, per si hubiera otros como Spark se comunicarían directamente con YARN. Sobre el entorno de cómputo se sitúan las aplicaciones de explatación, en este caso Pig y Hive que se podrán comunicar con los datos por medio de Mapreduce.

Existen otras formas de comunicación transversales no expuestas como el quorum de Zookeeper para comunicarse con las regiones de Hbase o la comunicación directa que existe entre Hbase y Pig y entre Hbase y Hive la cual busca sincronizar la transferencia de información que siempre se produce mediante mapreduce.

Adicionalmente buchos de estas aplicaciones exponen otras interfaces HTTP o Thirf para que los usuarios puedan conectarse a ellas.

TRADUCIÓN A CONTENEDORES

Esta arquitectura llevada a contenedores se manifiesta de la siguiente forma:



Tendremos un contenedor para el namenode, uno por cada datanode sucediendo lo mismo para el node de histórico y los manejadores de recursos. Esta misma estructura se encuentra en Hbase donde está el master en un contenedor y cada región en otro con la salvedad de que cada región deberá tener su propio contendedor con un Zookeeper dentro de él. Por otro lado, tendremos los tres contenedores que forman Hive, el master, el metastore y un Postgres para almacenar en él los metadatos.

Finalmente tendremos el contenedor de Flume que inicia el ciclo batch de extracción de datos y el contenedor de Pig que realiza el ciclo batch de inserción de los datos extraídos en Hbase.

DOCKER

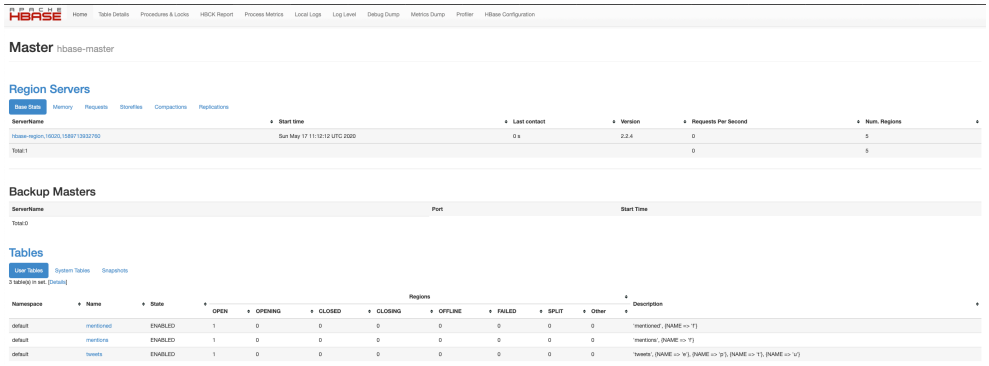
El despliegue de Docker es un despliegue estático y prácticamente manual. Si necesitamos múltiples datanodes o múltiples manejadores de recursos o múltiples regiones de Hbase deberemos crearlas manualmente.

El problema de esta creación manual es que no podremos exponer estos servicios sobre el mismo puerto. Estos problemas se solucionan al utilizar Swarm aunque no del todo pues parte de la replicación deberá de seguir siendo manual.

Podemos ver una captura de todos los contenedores levantados:

```
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS                    PORTS                                                                                NAMES
1d0edd0a45c4   hbase-master:latest                  "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes              0.0.0.0:8081->8081/tcp, 0.0.0.0:9090->9090/tcp, 0.0.0.0:16010->16010/tcp, 16000/tcp   hbase-master
37d2fc6535dc   nodemanager:latest                   "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes (healthy)    0.0.0.0:8042->8042/tcp                                                               nodemanager
e54986192501   hive-server:latest                   "entrypoint.sh /opt/…"   3 minutes ago    Up 3 minutes              0.0.0.0:9083->9083/tcp, 10000/tcp                                                    hive-metastore
f555fc1db291   resourcemanager:latest               "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes (healthy)    0.0.0.0:8088->8088/tcp                                                               resourcemanager
801feca0b03e   hive-metastore-postgresql:latest     "docker-entrypoint.s…"   3 minutes ago    Up 3 minutes              0.0.0.0:5432->5432/tcp                                                               hive-metastore-postgresql
49fb13eeb7d4   datanode:latest                      "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes (healthy)    0.0.0.0:9864->9864/tcp                                                               datanode
dcea74ff3eaf   historyserver:latest                 "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes (healthy)    0.0.0.0:8188->8188/tcp, 10020/tcp                                                    historyserver
87c2e2fcfda3   namenode:latest                      "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes (healthy)    0.0.0.0:8020->8020/tcp, 0.0.0.0:9000->9000/tcp, 0.0.0.0:9870->9870/tcp               namenode
59082e614184   hbase-regionserver:latest            "/entrypoint.sh /run…"   3 minutes ago    Up 3 minutes              16020/tcp, 0.0.0.0:16030->16030/tcp                                                  hbase-regionserver
6d79bf2af9f6   hive-server:latest                   "entrypoint.sh /opt/…"   3 minutes ago    Up 3 minutes              0.0.0.0:10000->10000/tcp, 9083/tcp, 0.0.0.0:10002->10002/tcp                         hive-server
653386952dae   zookeeper:3.4.10                     "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes              2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp                                           zoo
```

En este caso solo estamos utilizando una región de Habse:



Hive supuestamente debería de proporcionar también una interfaz web, no obstante, ese servicio no se ha logrado hacer funcionar, en su lugar se muestra un select de la tabla de menciones:

```
FCCDC080B927099104010CEB0B1CB4291031DA43245D988E5211BA842195C50B        15096659        dorrego 984542121385431041      PresuntoPodcast
FCD2F9863CE55BF0BBD48254B14500ED1CEC4D92E42B3DCB3294A64C3AA424D1        96244690        liliamm_27      1256606950114107396     cmdtemiliciab
FD39DE8EC11926F2B0FC28AF53198C3696AB7E37A146705BF26A1536A8823FC8        259441783       galletita2415   7996082 el_pais
FD39DE8EC11926F2B0FC28AF53198C369A487E44ADD50F5790B1BD5534F33BFD        259441783       galletita2415   112515706       aguilarcamin
FD78D9F066BFCFBF51F3FD98AC01F17B0886938302DF385C5B632567789A77D8        163835757       Parruita        491222521       AllSoy
FD78D9F066BFCFBF51F3FD98AC01F17BDF35C5EE26F2C81256482009C6633BB7        163835757       Parruita        402719918       fuencarralpardo
FD84D3FB01C9815CECF713912B5518D56690AC9CCDB1BAAE1480ACFC68FAD12D        74604187504024781O      AngelBB7777777  20823773        30SECONDSTOMARS
FE214D09334DB5C375AA8ED1D2EA8F5A281FD9A8C6C3E4656872D8DB01F46744        2928126825      NfmMartin       72484779        BertaRojas
FE214D09334DB5C375AA8ED1D2EA8F5AC1E9112BEF85219B4FB706C0B7263EBE        2928126825      NfmMartin       59225421        kchiporros
FE7EA3C2579A474495C0A0ABEE96D92C83FF838075AA2488F822FC857693C611        3082223424      mineto41        35805725        marcofeliciano
FE7EA3C2579A474495C0A0ABEE96D92CEC00EC6D4743688369FADB4326320E9F        3082223424      mineto41        19667016        veramagalhaes
FED72B9BCA79E5D7A956C1A5C30968AAE765F2A465BFCA52143C3E4EEB70CED2        3154365418      PCdoBpe 1252924187586560002     ctbnosintepe2
FFDA2ED636EAFDCA859D792979B9E20D8B5A8840805B3C1543BA4AED880AAA4A        886941373039075328      Misabelix       1144531022526197763     FeminismesSom
Time taken: 8.831 seconds, Fetched: 974 row(s)
```

SWARM

El despliegue en Swarm automatiza gran parte de despliegue en contenedores cuando deseamos tener múltiples instancias de un servicio pues este se expone oculto tras un balanceador de carga.

No obstante, no todos los servicios de Hadoop admiten estar detrás de un balanceador de carga. Por desgracia en quorum de Hbase y Zookeeper debe tener visibilidad directa. Podría haberse hecho un script para ser lanzado de forma automática no obstante este sería el mismo que si escribiera para Docker, es decir, sería una automatización "manual".

El script debería modificar la configuración del quorum en los archivos de habse-site, y lanzar Zookeepers y regiones con visibilidad directa. Ya que esto no aporta nada nuevo no se ha reliazo.

Lo que si se ha realizado es la automatización de un despliegue de HDFS y YARN sobre tres máquinas utilizando un registro común para difundir las imágenes.

./swarm.sh up

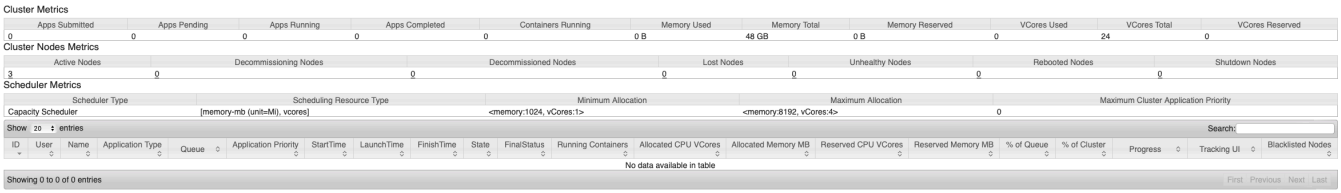
Podemos ver un despliegue realizado sobre tres máquinas aisladas salvo por una conexión de red. Se han levantado un contenedor con el registro de imágenes y otro que muestra la interfaz de la captura y la actualiza en tiempo real. Se ha levantado también un contenedor con el namenode y otro con el histórico que se asiganan a máquinas aleatorias y se ocultan tras un balanceador de carga que los redirecciona a la IP del master para facilitar el descubrimiento de servicios. Para los datanode y manejadores de recursos se crean uno por máquina y se les añade un balanceador de carga que hace que desde fuera se vena como uno solo.


master
manager
0.966G RAM
x86_64/linux
worker-1
worker
0.966G RAM
x86_64/linux
worker-2
worker
0.966G RAM
x86_64/linux
hadoop_datanode
image : datanode:latest@sha256:be
tag : latest@sha256:beaeddf493b8d
updated : 17/5 12:54
9a9ac48ff4c4cfed00eb30a7f4c9fcecf
state : running
viz
image : visualizer:latest@sha256:54c
tag : latest@sha256:54d65cbcbff52e
updated : 17/5 12:45
1ccfeb5648d0fcadd84030579a18597
state : running
hadoop_nodemanager
image : nodemanager:latest@sha25
tag : latest@sha256:490f8c08249f9b
updated : 17/5 12:54
7180861d9e073753adb5daa1bfdc14
state : running
registry
image : registry:latest@sha256:7d08
tag : latest@sha256:7d081088e4bfd
updated : 17/5 12:46
103a0ffb68bbad43fe61fb427041c66
state : running
hadoop_nodemanager
image : nodemanager:latest@sha25
tag : latest@sha256:490f8c08249f9b
updated : 17/5 12:54
78428716a8362c2e9b83cd517a776
state : running
hadoop_datanode
image : datanode:latest@sha256:be
tag : latest@sha256:beaeddf493b8d
updated : 17/5 12:54
e2271ca8768a3be10ebaa12db8772
state : running
hadoop_resourcemanager
image : resourcemanager:latest@sh
tag : latest@sha256:82c9005185b53
updated : 17/5 12:53
5041fd03e7658dedf3196b364be900
state : running
hadoop_namenode
image : namenode:latest@sha256:6
tag : latest@sha256:6cfdf9a2176012
updated : 17/5 12:53
5e98d20135b60da6773c8e5639703
state : running
hadoop_nodemanager
image : nodemanager:latest@sha25
tag : latest@sha256:490f8c08249f9b
updated : 17/5 12:54
ac07be376c9693f0c83eb56e01ca5d
state : running
hadoop_datanode
image : datanode:latest@sha256:be
tag : latest@sha256:beaeddf493b8d
updated : 17/5 12:54
daa60ab011950089a85091e435dc60
state : running
hadoop_historyserver
image : historyserver:latest@sha256
tag : latest@sha256:25b11980c0ff2c
updated : 17/5 12:54
2a6c588030dff5360859266b43c3f89
state : running

Podemos ver que Hadoop reconoce la existencia de los tres datanodes

Configured Capacity:	53.54 GB
Configured Remote Capacity:	0 B
DFS Used:	72.05 KB (0%)
Non DFS Used:	5.22 GB
DFS Remaining:	45.37 GB (84.74%)
Block Pool Used:	72.05 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	3 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)

Así como de los tres manejadores de recursos



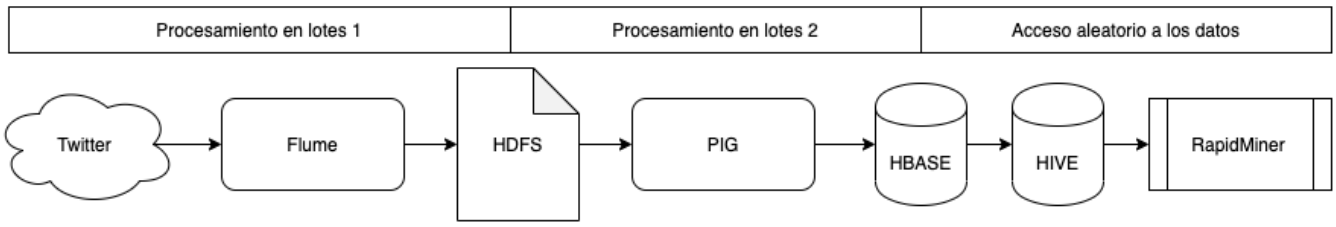
ANÁLISIS DE LOS DATOS

El análisis de los datos propiamente dicho se realizará en la siguiente práctica. No obstante, algunas partes preparatorias para dicho análisis han sido ya realizadas. El objetivo de la preparación de los datos es crear un flujo de información que los filtre de modo que en Hive, que será la base de datos de explotación queden solo aquellos datos que sean de interés.

Posteriormente con otras aplicaciones del entorno Hadoop como Spark o Mahout o con aplicaciones externas como RapidMiner o Pentaho podremos acceder a Hive contando con todo el poder de Mapreduce para extraer información sobre los Tweets almacenados.

FLUJO DE LA INFORMACIÓN

Debido a las limitaciones de la API de Twitter para permitirnos extraer datos de forma continua y debido a la baja potencia del equipo en el que se realizará la explotación esta se hará siguiendo estos pasos:



- **docker-compose up** para levantar HDFS, YARN, Hbase y Hive. Los datos de la sesión anterior se cargan de forma automática. Para inciar una sesión nueva se deben borrar los volúmenes antes de ejecutar este comando.
- **./exploit.sh** En este script se realizan los siguientes pasos:
    - Comprobar si las tablas en Hbase y Hive existen y si no existen se crean.
    - Lanzar Flume durante un tiempo concreto extrayendo datos de todas las fuentes indicadas, para cada fuente se estarán extrayendo datos durante el tiempo indicado, por ejemplo, un minuto.

- Lanzar Pig. Con flume parado se crea el proceso de mapreduce que carga los datos que Flume almacenó en HDFS sobre Hbase.
- Borrar los datos de HDFS. Una vez que los datos se cargan sobre Hbase ya no se necesita que estén en HDFS, además así se evita cargar datos duplicados.

En todo momento Hive estará activo y se podrán analizar dos datos almacenados según se estén cargando datos nuevos sobre Hbase. Cuando se introduzca un nuevo batch de datos este quedará disponible en Hive de inmediato.

FILTRADO DE LOS DATOS

A lo largo de este proceso se producen dos puntos de filtrado, el primero en Flume dejando pasar solo Tweets escritos en España, Madrid o Barcelona y que además tengan o no los tags que indiquemos.

El otro filtrado de datos se produce en Pig donde nos quedaremos solo con Tweets que contengan las palabras indicadas en el archivo de análisis.