Práctica I – Cloud Storage
Buscador de Imágenes

Objetivo:

La práctica tiene como objetivo poner en práctica el uso de KVS, adicionalmente la práctica permitirá poner en práctica el uso del SDK de *Amazon Web Services* tanto en Java como en JavaScript.

Requisitos

Para este ejercicio es necesario contar con una cuenta en AWS

Desarrollo de la Práctica.

La práctica se encuentra dividida en dos etapas a fin de facilitar su desarrollo. Es importante señalar que debe funcionar la primera etapa para poder continuar con la segunda etapa. De igual manera cada uno de las etapas estas definida como una secuencia de pasos, es necesario terminar totalmente cada paso antes de avanzar al siguiente.

Fase 1 – Interfaz CLI utilizando SQLite

Dentro de esta fase, implementaremos un programa para consola que nos permita cargar y consultar la información de DBPedia/Wikipedia.

Es necesario descargar los siguientes elementos:

- Esqueleto del Proyecto
  - https://github.com/aparres/ITESO_CC_LP1MS1

- *Dataset* de Imágenes:
  - http://downloads.dbpedia.org/2016-10/core-i18n/en/images_en.ttl.bz2
- *Dataset* de Etiquetas
  - http://downloads.dbpedia.org/2016-10/core-i18n/en/labels_en.ttl.bz2

El archivo *images_en.ttl.bz2* asocia cada artículo de Wikipedia con su imagen principal y su miniatura, mientras que el archivo *labels_en.ttl.bz2* asocia cada artículo de Wikipedia con etiquetas. Por ejemplo, la URL de la imagen de una nube está relacionado con la categoría de Wikipedia NUBE la cual tiene como etiqueta de búsqueda la palabra nube entre otras. Nosotros utilizaremos estos dos archivos para crear un buscador de imágenes. Los dos archivos en cada renglón contienen tres datos <A> <B> <C> que describen varios aspectos de las categorías de

Wikipedia, no solo imágenes y etiquetas. Se puede leer la relación entre los tres elementos como <A> y <C> están relacionados y esta relación la describe <B>.

Para este ejercicio solo nos interesa un tipo de relación, para el caso de las imágenes nos interesa la relación de tipo *http://xmlns.com/foaf/0.1/depiction*(en este caso <A> es la categoría y <C> es la URL de la imagen) y para el caso de las etiquetas nos interesa el tipo: *http://www.w3.org/2000/01/rdf-schema#label*(donde <A> es la categoría y <C> es la etiqueta).

El objetivo de esta fase es crear un mapeo de etiquetas con imágenes. Es decir que la palabra clave *"cloud"* nos deberá regresar imágenes de nubes. Para esto primero generaremos una colección de datos, que denominaremos ***images*** con cada una de los artículos de Wikipedia y su imagen principal. Y otra colección llamada ***terms*** donde mapearemos palabras clave con artículos de Wikipedia.

Adicional a lo anterior en esta fase utilizaremos SQLite como KVS con el fin de poder probar nuestro algoritmo de manera local, sin utilizar recursos AWS, en la siguiente fase modificaremos nuestro código con el objetivo de utilizar AWS.

1.1 Loader

Lo primero que hay que hacer es programar el cargador de las imágenes, el cual se llamara **loadimages.py.** En la estructura del proyecto se encuentran la carpeta **keyvalue** que contiene diferentes clases que les permitirán de forma más simple construir este.

- La clase **ParseTriples** nos permite abrir los archivos ttl y obtener cada uno de los conjuntos de tres elementos. Cada conjunto se obtiene llamada al método **getNext()** el cual regreso un arreglo con los tres elementos (A,B,C)

Para programar el cargador se debe mandar llamar al *parser* sobre cada uno de los archivos e introducir la información en la colección apropiada. Algunas instrucciones/restricciones son:

- Se deben cargar las imágenes, en la colección ***images.*** Hay que recordar que no nos interesan todas las imágenes. Hay que programar alguna forma de limitar la cantidad de imágenes que subimos.
- La colección de ***terms*** deberá funcionar como un índice invertido, donde la etiqueta sea la llave y la categoría el valor. Es importante recordar que una misma etiqueta puede apuntar a diferentes categorías.
- Solo se deben almacenar las etiquetas de que correspondan a articulos de Wikipedia de los que tengamos alguna imagen.
- Si una etiqueta tiene múltiples palabras se debe separar y generar una entrada por cada palabra.

- Con el objetivo de poder procesar consultas con valores aproximados se deben regularizar todas las palabras de las etiquetas, para esto utilizaremos el algoritmo de *stemming* el cual normaliza cualquier palabra a su origen. Para esto se incluye el archivo ***stemmer.py*** con la función **stem(word)** que realiza este proceso.

Ejemplo:

ImagesStore
Key: http://dbpedia.org/resource/American_National_Standards_Institute
Value: http://en.wikipedia.org/wiki/Special:FilePath/ANSI_logo.svg

TermsStore
Key: american
Value: http://dbpedia.org/resource/American_National_Standards_Institute

Key: nate
Value: http://dbpedia.org/resource/American_National_Standards_Institute

Key: standard
Value: http://dbpedia.org/resource/American_National_Standards_Institute

Key: institut
Value: http://dbpedia.org/resource/American_National_Standards_Institute

Es importante mandar llamar al método *close()* al fin de cerrar la base de datos, sin perder información.

1.2 Querier

Completar el archivo **queryimages.py** el cual toma uno o más palabras clave desde la línea de comandos, es decir como parámetros al mandar llamar el programa e imprime en consola la lista de URLs de imágenes relacionadas. Algunos puntos importantes:

- A la hora de la búsqueda se debe aplicar de igual manera el método de *stemming* a los palabras clave buscadas.

Fase 2 – Cambiar Storage a DynamoDB

Dentro de esta fase, implementaremos la clase **dynamostorage.py** con el fin de modificar el código anterior y utilizar DynamoDB como storage. Algunos puntos importantes para esto son:

- Ayuda específica para desarrolladores y el trabajo con DynamoDB se puede encontrar en: https://aws.amazon.com/es/dynamodb/developer-resources/

- Las tablas deberán tener la siguiente definición:
  - Primary Key: *keyword* de tipo String.
  - Primary Sort Key: *inx* de tipo Number.
  - 5 unidad de lectura
  - 5 unidad de escritura.
- Los parámetros de conexión: accessKey, secretKey deben estar en un archivo de configuración del sistema. No en el código fuente.
- La Región se deben leer desde la clase Config.
- La clase debe validar si la tabla existe o no, en caso de no existir debe crear la tabla.
- Se debe pensar en reducir el número de GET/PUT que se realizan a Amazon.
- No es necesario implementar todos los métodos, solos necesarios para que el código funcione.

Para probar esta fase se recomienda limitar la cantidad de información que se sube

Fase 3 – Entorno WEB para consultas.

Dentro de esta fase, implementaremos una interfaz web para realizar las consultas, la cual estará conectada a DynamoDB.

Es necesario descargar el Esqueleto del Proyecto

- https://github.com/aparres/ITESO_CC_LP1MS2.git

El esqueleto del proyecto es la base para empezar a codificar el mismo, este usa Node.JS, Express, JADE, JQuery y BootStrap. Algunos puntos importantes son:

- Es importante ejecutar ***npm install*** para descargar todos los módulos necesarios.
- En el archivo ***config.json*** se deben capturar la región donde se trabajara.
- En el archivo ***views/index.jade***
  - Este archivo contiene el diseño completo de la página web que usaremos.
  - Al principio viene el cuadro de dialogo que se utilizara para presentar los resultados. Este cuadro de dialogo de inicio arranca oculto. Dentro de este cuadro de dialogo hay una tabla con cuatro celdas, numeradas de 1 al 4 donde se presentarán los resultados.
  - Casi al final existe una sección denominada *autor_name* donde deben cuidar de color su nombre y número de expediente.
- En el archivo **public/js/*app.js***
  - En este archivo se encuentra la lógica de la página que correrá en el lado del cliente.

- Podemos ver que al principio del archivo están las instrucciones necesarias para que cuando el cliente presione el botón de **Search** o de **Intro** aparezca el cuadro de dialogo con los resultados.
- Lo que podemos observar en este archivo es que al presionar **Search** se manda llamar la función *displayModal()* la cual hace que aparesca el cuadro de dialogo y modifica el código HTML algunas secciones del cuadro dialogo. Posteriormente hace una solicitud al servidor con el método *getJSON()* y por ultimo manda presentar los resultados mediante la función *renderQueryResults();*
- La función *renderQueryResults()* si la comunicación con el servidor fue exitosa y presenta o no los botón de **Next** y **Previous**
- **En este archivo falta codificar varias secciones como el presentar las fotografías, el hacer que los botones *next* y *pevious* solo aparezcan en caso de ser necesarios, y programas la funcionalidad de estos mismos botones.**

- En el archivo **keyvaluestore.js**
  - Programar la función **init** en la cual se debe validar que la tabla exista. Y en caso positivo llamar a la función de callback. Como se ve en el código.
  - Programar el método GET para recuperar valores de DynamoDB.
    - El código macro que se presenta contiene el uso de CACHE para optimizar el acceso a DynamoDB.
    - Es importante que una vez que se recuperan los valores insertarlos al CACHE.
    - Los valores se deben regresar como una colección de objetos.
  - Es importante revisar la lógica del servidor **app.js** a fin de ver que se espera de la función.

Fase 4 – Todo en la Nube

Utilizando una máquina de EC2 publicar el servicio en un servidor. Importante cuidar que sea en la misma zona.

Entregables de la Práctica.

- Cada uno de los Códigos Fuente desarrollados.
  - En caso de no usar maven
    - Incluir todas las librerías necesarias en la capeta de lib/
  - Archivo README con la siguiente información:
    - Explicación breve sobre lo que hace su código y que no hace.
  - No incluir los datasets.

- Video demostrativo de la funcionalidad completa directamente en la nube. (La carga de datos se puede hacer desde el host o desde la nube, como ustedes lo prefieran).
- Reporte Técnico de la práctica.
    - Indicar en una sección del reporte la lógica de **app.js** (express) y de **keyvaluestore.js** (acceso a AWS desde javascript)
- Se les recuerdo que todo el código y reporte no puede tener referencias a ustedes a fin de poder ser revisado.