XKCD-Comics

ETL Workflow

In Airflow wurden folgende Tasks definiert:

1. **create_local_import_dir_xkcd**: Erstellt lokal ein Ordner „xkcd“.
2. **create_local_import_dir_raw**: Erstellt lokal im Ordner “xkcd” den Ordner „raw“.
3. **create_local_init_file**: Erstellt im „raw“ Ordner eine Datei. Dieser Schritt wird benötigt, weil der clear Befehl fehlschlägt, wenn am Anfang keine Datei vorhanden ist.
4. **clear_local_import_dir**: Löscht alle Dateien im „raw“ Ordner. Es wurde ein Bash Operator verwendet, weil er sonst nicht rekursiv alle Ordner löscht.
5. **download_xkcd**: Lädt alle XKCD-Comics herunter.
6. **hdfs_create_xkcd_dir**: Erstellt in HDFS den Ordner „xkcd“.
7. **hdfs_create_xkcd_raw_dir**: Erstellt in HDFS den Ordner „raw“ in „xkcd“.
8. **create_hdfs_init_file**: Erstellt eine Datei, damit der clear Befehl nicht fehlschlägt, wenn am Anfang keine Datei vorhanden ist.
9. **hdfs_clear_xkcd**: Löscht alle Dateien in HDFS im Ordner „xkcd“.
10. **hdfs_push_xkcd**: Kopiert alle heruntergeladenen Dateien in HDFS.
11. **pyspark_xkcd_raw_to_final**: Speichert die optimierten Daten in die Datenbank und HDFS.

Download XKCD-Comics

Der Download holt sich zuerst den letzten veröffentlichten XKCD-Comic und speichert die ID. Anschließend wird mit einem GET Request alle XKCD-Comics heruntergeladen. Dabei wird geprüft, in welchem Jahr der Comic veröffentlicht wurden ist und erstellt dynamisch ein Ordner mit dem Jahr. Für jeden Comic wird eine Datei erstellt und als JSON abgespeichert. Die GET Requests wurden mit Threading in Python umgesetzt. Somit benötigt der Download nur noch ca. 9 Sekunden anstatt 10 Minuten.

Da die Zeit knapp wurde, konnte folgende Verbesserung nicht umgesetzt werden: Pro Comic wird eine JSON-Datei erstellt und in HDFS hochgeladen. Das dauert sehr lange und ist im Workflow ein Bottleneck. Als Verbesserung könnte pro Jahr eine Datei erstellt werden, indem alle Comics von dem Jahr enthalten sind. Mit der Verbesserung des Downloads, könnte die Durchlaufzeit des Workflows stark reduziert werden.

PySpark

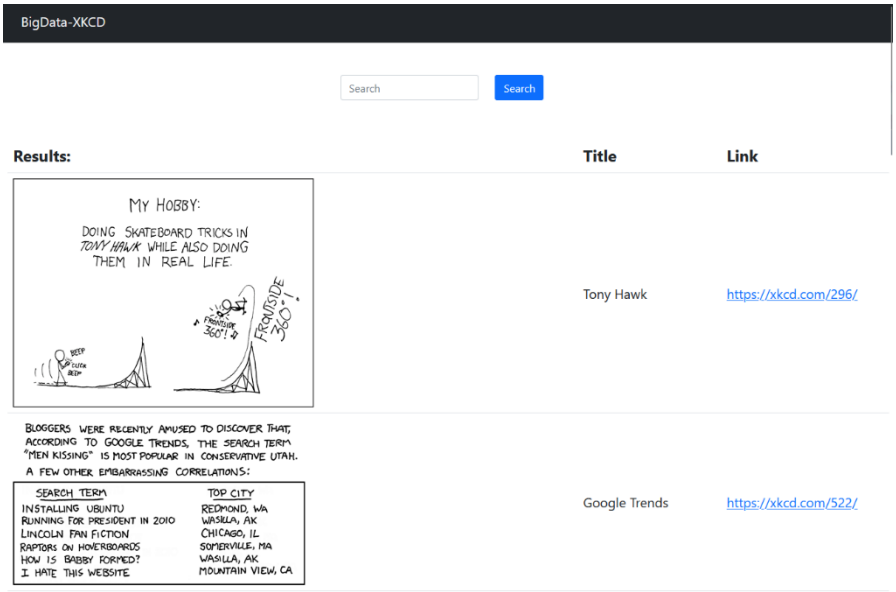
In Pyspark werden die Daten rekursiv von HDFS in ein DataFrame eingelesen. Anschließend werden auf folgenden Attributen selektiert:

1. **num**
2. **img**
3. **alt**
4. **safe_title**
5. **title**

Der Rest wird verworfen. Duplikate gibt es nicht und werden ignoriert. Anschließend wird der DataFrame in MySQL abgespeichert. Dabei wird die Datenbanktabelle dynamisch anhand dem DataFrame erstellt. Zum Schluss werden die Daten partitioniert und als CSV in HDFS abgespeichert.

XKCD-Webseite

Für die Webseite wurde PHP und für den Webserver Nginx verwendet. Dabei laufen PHP, Nginx und MySQL in einem eigenen Docker Container. Zusätzlich wurde Bootstrap für das Design der Webseite eingesetzt. Die Seite ist über den Port 80 aufrufbar.



In der Suchleiste kann mit einem Suchbegriff nach den XKCD-Comics gefiltert werden. Der Suchbegriff wird in den Attributen alt, safe_title und title durchsucht.

Zusätzliche Installationen

Damit das Projekt lauffähig ist, sind ein paar zusätzliche Installationen nötig. Vorausgesetzt werden die zwei Container Airflow und Hadoop von der Vorlesung.

1. Installation von Docker Compose
   ➔ sudo apt install docker-compose

2. Im Home Verzeichnis werden folgende Dateien und Ordner benötigt. Dabei ist die Dateistruktur Wichtig, da die Container darauf zugreifen. Jedes Mal, wenn eine Datei erstellt wird, muss von derselben Datei im Abgabeordner der Inhalt rüber kopiert werden:

```
Dockerfile  default.conf  docker-compose.yml  src
inf20125@big-data-exam-xkcd-presentation:~$
```

   ➔ mkdir src
   - Dort befinden sich die Dateien für die Webseite. Auf den Ordner greift der Webserver Nginx zu.

   ➔ nano docker-compose.yml

- In der docker-compose Datei werden zusätzliche Docker Container konfiguriert und gestartet. Darin wurden Container definiert für MySql, Nginx und PHP-fpm. PHP-fpm wird für Nginx benötigt, damit PHP ausgeführt werden kann.

➔ nano Dockerfile
- Im Dockerfile wird der PHP-fpm Container mit einem RUN-Befehl erweitert. Der Befehl installiert MySqli und ermöglicht den Webserver sich auf die MySql Datenbank zu verbinden.

➔ default.conf
- Das ist die Konfigurationsdatei für den Nginx Webserver. Dort wird z.B. PHP und das root Verzeichnis konfiguriert.

Anschließend in den /src Ordner wechseln und folgende Datei erstellen.

➔ nano index.php
- Ist die PHP-Datei von der Webseite.

Als nächstes muss in den Airflow Container gewechselt werden. Dort muss für MySql der JDBC-Driver heruntergeladen und im spar/jars Ordner kopiert werden.

➔ wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-j-8.0.31.tar.gz

➔ tar -xvzf mysql-connector-j-8.0.31.tar.gz
- Datei entpacken

➔ cd mysql-connector-j-8.0.31

➔ mv mysql-connector-j-8.0.31.jar ../
Die jar Datei eine ebene nach oben kopieren

➔ mv mysql-connector-j-8.0.31.jar spark/jars
Die jar Datei in spark/jars Ordner hinterlegen. Anschließend kann MySql verwendet werden.

3. Wenn alles installiert und aufgesetzt ist, kann zu dem Home Verzeichnis der VM gewechselt werden. Dort kann die docker-compose.yml Datei mit folgendem Befehl gestartet werden:

➔ docker-compose up -d (evtl. docker-compose build vorher)

Danach sind die Docker Container lauffähig und der DAG kann im Workflow gestartet werden.

Hinweis: Falls ein Error auftritt mit „… user specified IP adress ...“, dann muss die komplette Virtuelle Maschine neugestartet werden.