Lessons Learned

We heard a lot about Cassandra in our BDEA class and wanted to try it out, so it was our first choice (little did we know that it wasn't really suited for the task).

We also tried some other Databases more on that later in the file.

Cassandra

Lessons learned from installing Cassandra DB with Docker Compose to run a social network which loads data from Twitter.

Infrastructure

- Single Cassandra Docker container need a long time to startup and load the configuration (depending on the host system), to make sure it runs everywhere without crashing we implemented a health check in Docker Compose, this was not very easy as we had to determine the right parameters through trial and error. Additionally the containers have a restart always policy in case the crash.

```
healthcheck:
        test: ["CMD", "cqlsh", "-e", "describe keyspaces" ]
```

```
      interval: 10s
      timeout: 10s
      start_period: 50s
      retries: 10
```

- Cassandra dos not offer a web interface but luckily the open source community implemented one called cassandra-web. Cassandra web requires IPs to be set static as it tries to connect to them and does not work with Docker DNS.
    - cassandra-web requires an older Ruby version (a version > 3 causes problems).
- Cassandra requires a certain configuration to work, this configuration can be set using environment variables and configuration files. The following information is required and can be set as environment variables in Docker Compose:
    - CASSANDRA_SEEDS: "cass1,cass2"
    - CASSANDRA_CLUSTER_NAME: SolarSystem
    - CASSANDRA_DC: Mars
    - CASSANDRA_RACK: West
    - CASSANDRA_ENDPOINT_SNITCH: GossipingPropertyFileSnitch
    - CASSANDRA_NUM_TOKENS: 128
    - In our case we additionally set the following parameters to not overload the system:
        - MAX_HEAP_SIZE: 1024M
        - HEAP_NEWSIZE: 1024M
    - Advanced configuration can be set in the following required configuration files (stored in etc/cassandra):
        - cassandra-env.sh
        - cassandra-rackdc.properties
        - cassandra.yaml
        - commitlog_archiving.properties
        - jvm-clients.options
        - jvm-server.options
        - jvm8-clients.options
        - jvm8-server.options
        - jvm11-clients.options
        - jvm11-server.options

        - logback.xml
    - Parameter that we needed to set additionally in those files were: * `enable_materialized_views: true` to be able to create materialized views. * `enable_sasi_indexes: true` to be able to create index on twitter.tweets content. * `*_timeout: >default` to be able to read large csv-files into the cluster.
    - The Docker volume mapped to the Cassandra Container at `etc/cassandra` needs to have all the above files in order for cassandra to work, the values specified in the above environment variables will override the default values in the actual configuration files.

- For set the setting for cqlsh (a python script-shell to run cql-queries on the database) we add a csqhrc-file in `/.cassandra` , which allowed only one file in the directory with the setting for cql. There we change following settings:

```
          [ui]
          timezone = Etc/UTC` : Set the current timezone
          time_format = %d/%m/%Y %H:%M`: Change the date pattern
to import time_stamps
          [copy]
          ESCAPE = \: set the escape character
          QUOTE = ": set the quote character
```

- At the first try we experienced problems executing our startup script to run CQL commands: `Connection error: ('Unable to connect to any servers', {'172.20.0.6:9042': ConnectionRefusedError(111, "Tried connecting to [('172.20.0.6', 9042)]. Last error: Connection refused")})` . This was due to enabled authentication which we then ended using for the connection. The default user and password is *cassandra*.

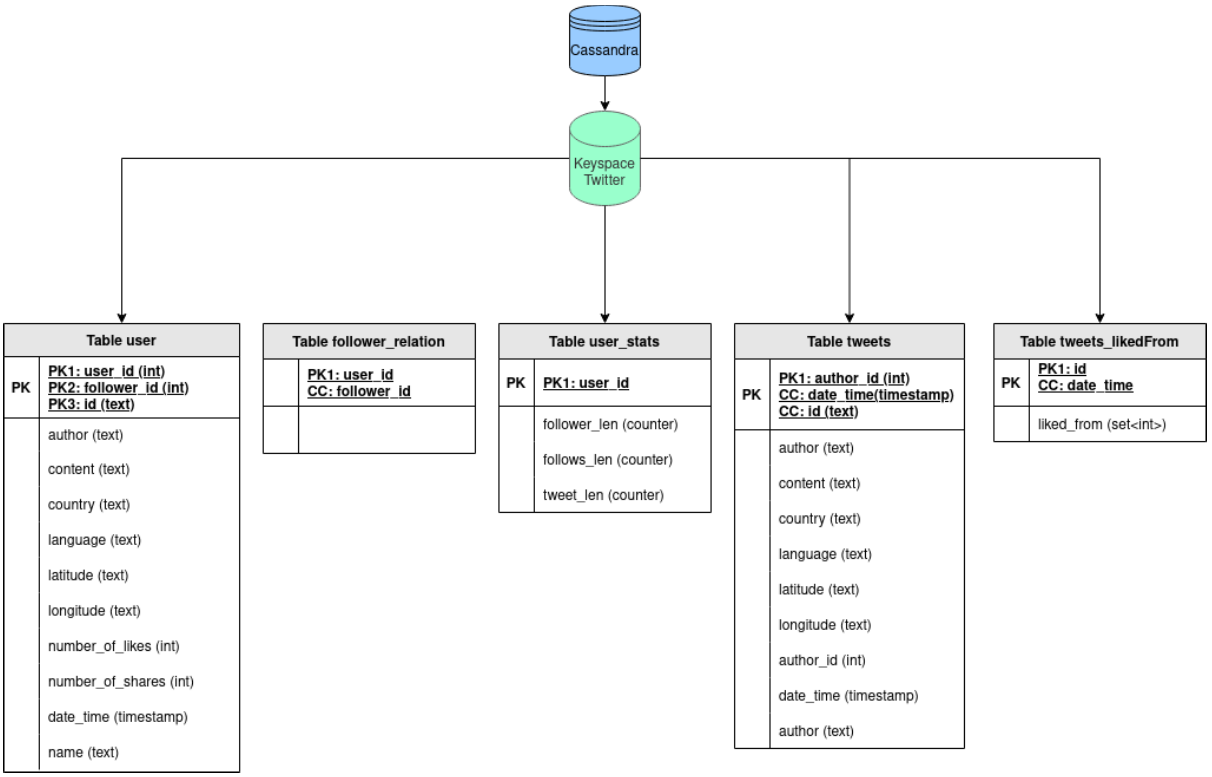
- When creating materialized views we received the following message `Warnings : Materialized views are experimental and are not recommended for production use.` Apparently materialized views are experimental and this is also why we had to explicitly enable them in the configuration. It is recommended to rather use duplicate tables as suggested in this stackoverflow thread.

- Views are also experimental and have to enabled in the cassandra.yml-file. However they are only usefull to rearange the order of the keys-column or to add one new "no key"-column as a new key value. Additionaly is not possible to add new fields like count(col_name) to a view. To archive this it is recommended to use a new data schema and load data with the cassandra-spark-connector as a new table.

- To import large CSV-files is recommended to use the sstableloader or a spark cluster. Because of the time restriction we only tried out to use ready to use cvs_to_sstable_convert and don't try do convert them manuly. But the data can not bet imported to the database and we getting correct fileupload with 0-files upload message.

- Since we are not able to the sstableloader to import large files, we had to minimize the liked_from_user list for each tweet. We reduce the data from the orignal likes (has around 7GB and new user_ids where generated) to tenth of the size (around 480 MB).

Data Model



- When first importing the data into Cassandra we ran into the follwing problem: `Failed to import 1 rows: ParseError - Failed to parse 5.34896E+17 : invalid literal for int() with base 10: '5.34896E+17', given up without retries 'builtin_function_or_method' object has no attribute 'error'` . In order to fix this we had to manipulate the data before the import manually.
- We had to change the data schema a lot of times and try out driffent combination to make the querries work. We end up to use the realation table with the realationship between user_id,follower_id and tweet_id which build the primary key. So the data will be saved multiple times for eacht id and get around 45 times larger thand the orginal data. Addtionly we also use a stats table with counter to get a faster querry for the length of follower or follows. Because of the "world-search"-querry (task 6) we also add the tweets in a separate table to be

able to run a index on the content col and filter with the LIKE-keyword.

- UDTs: We tried to load the tweet as UDTs to the above data schema. To perform the this we updated the structure of the combined csv-file after this-stackoverflow post but always get an error for columns missmatch. We assume the content in the UDTs was not quoted so `,` in the conent section cause these erros.
- The only materialized view we used is for the exercies 5 to get a fanout-like style, that updates after a new tweet is added.
- SASI-index are (like most of the thinga) experimental and are not recommened for production use. However we had to use it for the exercies 6 to enable the searching with the `LIKE` -statment. We assume that cassandra make a intern table with each word associated with the tweets row in which its appears, since the index take around 5 minutes to create.

Other

- To optimize for read over write (as this will be our case), we set compaction to `LeveledCompactionStrategy` as recommended by the Cassandra documentation for this kind of system: `[..] WITH compaction = {'class' : 'LeveledCompactionStrategy'};`
- Formula for replication factor as suggested by this blog post: [read-consistency-level] + [write-consistency-level] > [replication-factor]
- Pre-sort data can be achieved by: `CLUSTERING ORDER BY (number_of_likes ASC);`
- ..

Neo4j

Lessons learned from installing Neo4j with Docker Compose and feed it with Twitter data.

Because we had many problems with tha Queries in Cassandra we wanted to try another Database and choose Neo4j:

- We realized very quickly that the free version of Neo4j (community version) does not offer the possibility to use multiple nodes in a cluster. That's why we switched to the Enterprise Edition which normaly costs some money (we only used it for this small university project, so please don't sue us).
-

We wrote a Compose File to setup a Neo4j Enterprise Cluster with 3 CORE Nodes and 1 Replica Node. Replica Nodes are only used for read operations (no writes) so we thought it would be a good idea to have one for analysis tasks to take the load away from the CORE Nodes (they also can write into the Database)

- We followed this Tutorial but had some Problems the get the enviroment variables working in the compose file and the containers itself. After modifying our Makefile and adding the needed enviroment variabled there, it worked.
- We also encountered many Problems regarding User-Permission so the Neo4j instantly shut down after starting because it could Write or Read the Folders it created. After creating this script and running it befor starting the compose file, we fixed that.
- After these fixes the compose file worked and every cluster node started and connected to each other. But couldnt get the neo4j Browser connection to work. After some hours of trying we realised that it had something todo with our Gitpod enivormet and it only works on our local devices. Because we dont have much RAM on our devices we decided to use the neo4j Cyper-Shell on one of the CORE nodes in our gitpod eviorment.
- After creating the Keyspace and copying the csv file into the Import folder, we faced a problem with the CSV import. Following the LOAD command we got the following error: `Couldn't load the external resource at file:/twitter_combined.csv` . We read many forum articles etc. but after hours we still could not get it to work, so we stopped here and focused on cassandra again.

Other Databases (Scylla and Redis)

After some frustration about cassandra we Setup some other Database Clusters (Scylla and Redis). Both clusters setups did start and seem to work but due to time constrains we again focused back to cassandra.

- If we had more time left we would try our Cassandra Queries and Script on Scylla because it also uses CQL as a Query Language and is called a faster Cassandra alternative.