MDS miniHive
Winter Term 2024/25
Lab 1 (last updated 20-Oct-2024)

As a course project, you will develop "miniHive", a minimalist version of the SQL-on-Hadoop system Hive. This worksheet introduces the miniHive Docker container, our standardized development environment.

1 Prerequisites

We expect you to have the following skills:

- **Basic programming skills in Python.**
- **Familiarity with Docker:** Understanding its basic functionality.
- **Understanding Docker Images:** Basic knowledge of what a Docker image is and how to use it.
- **Understanding Docker Containers:** Basic knowledge of what a Docker container is and how to use it.
- **Distinction Between Docker Image and Container:** Understanding the difference between these two concepts.
- **Basic Bash Command Line Skills in Linux**, since the miniHive Docker container does not have a graphical user interface.
- **Familiarity with Git:** Understanding the very basics of version control using Git.
- **Basic SSH Usage:** Knowledge of how to connect to a remote machine via SSH.

For a refresher, we recommend the following free courses and manuals:

- Python: `https://learn.udacity.com/courses/ud1110`
- Docker: `https://docs.docker.com/get-started/overview/`
- Bash: `https://www.udacity.com/course/linux-command-line-basics--ud595`
- Git: `https://www.udacity.com/course/version-control-with-git--ud123`
- SSH: `https://www.openssh.com/manual.html`

2 Hardware Resources

To run the miniHive Docker container effectively, have a minimum of 4 GiB of RAM available. For optimal performance, have at least 8 GiB of RAM on your machine. This will ensure smooth operation and allow you to manage larger datasets without performance issues.

3 Installing Docker

Install the Docker environment on your machine according to `https://docs.docker.com/get-docker/`. Ensure a correctly running Docker environment.

4 The miniHive Docker Container

We will use the following notation for command-line tasks. Do *not* copy the prefixes to the command line, they are just an indicator of *where* to execute the command.

- `user@host$`: Execute the command on your *local* machine.
- `minihive@container$`: Execute the command *inside the miniHive Docker container*.

4.1 Pulling the miniHive Docker Image

Have Docker up and running. Pull the pre-built Docker image from `ghcr.io` to your local machine:

```
user@host$ docker pull ghcr.io/sdbs-uni-p/minihive-docker:v1.1.2
```

If successful, the command

```
user@host$ docker image ls
```

will list the miniHive image among the available images.

4.2 Tagging the Image

Once the Docker image has been pulled, create a tag for more convenient use:

```
user@host$ docker tag ccaf5278d567 minihive-docker:latest
```

If successful, the command

```
user@host$ docker image ls
```

will list the miniHive image with its new tag.

4.3 Running the miniHive Docker Container

The following command will create and run a container and name it `minihive`. This command will also redirect the connections on your local port 2222 to the container's port 22:

```
user@host$ docker run -d --name minihive -p 2222:22 minihive-docker
```

If successful, the system shows the unique container ID assigned to the created container.

At this point, the docker container is running in the background. You can also verify that the container is running with the following command:

```
user@host$ docker ps -a
```

Now, `minihive-docker` should be listed with a status like `Up 10 seconds`.

The container comes with various software packages installed (an SSH server, as well as Hadoop, HDFS, Hive, and Spark). Thus, it may take a few seconds to start up.

4.4 The miniHive User Account

The user name and password for the miniHive Docker container are:

- username: minihive
- password: minihive

The `minihive` user has `sudo` rights.

4.5 Accessing the Container via SSH

When connecting to the container via

```
user@host$ ssh -p 2222 minihive@localhost
```

and after entering the password, you should see a welcome message:

```
Welcome to Ubuntu 21.04 (GNU/Linux 5.10.0-6-amd64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
```

```
           _       _ _   _ _
 _ __ ___ (_)_ __ (_) | | (_)_   _____
| '_ ` _ \| | '_ \| | |_| | \ \ / / _ \
| | | | | | | | | | |  _  | |\ V /  __/
|_| |_| |_|_|_| |_|_|_| |_|_|_| \_/ \___|

miniHive Docker
```

5 HDFS Basics

Let us get hands-on with the Hadoop Distributed File System (HDFS). You have been assigned to help the Trick-or-Treat Candy Warehouse 👻 (TTCW) manage their collection of Halloween candy.

Using HDFS, you will store, organize and fetch candy data. The HDFS commands are syntactically similar to the commands you are familiar with from the Linux command line. This makes them easier to use and remember. However, while they appear similar, these commands operate very differently under the hood. In HDFS, the commands interact with a distributed file system, managing data across multiple machines.

As you work through this exercise, it is important to understand the difference between your local file system on the host machine, in the miniHive Docker container, and HDFS running inside the miniHive Docker container.

Thus, each of the commands "`user@host$ ls`", "`minihive@container$ ls`", and further "`minihive@container$ hdfs dfs -ls`" targets a different location.

5.1 Store Candy Data in HDFS

The TTCW has received candy from various towns. We need to store the records in HDFS, where each file represents a shipment from a given town:

File `candy_town1.csv`**:**

```
Candy Corn
Snickers
Reese's
Candy Corn
Snickers
```

File `candy_town2.csv`**:**

```
M&Ms
Butterfinger
Milky Way
M&Ms
Butterfinger
```

File `candy_town3.csv`**:**

```
Reese's
Almond Joy
Butterfinger
KitKat
Butterfinger
```

Create the files inside the `data` directory inside your miniHive Docker container. For convenience, you may install a command-line editor (such as `nano`, `emacs`, or `vi`).

5.1.1 Creating a directory in HDFS

```
minihive@container$ hdfs dfs -mkdir /candy_warehouse
```

Use `hdfs dfs -ls /` to confirm that the directory has been created.

5.1.2 Upload candy data for each town into HDFS

Upload the files `candy_town1.csv`, `candy_town2.csv`, and `candy_town3.csv` into HDFS, e.g. for the first file:

```
minihive@container$ hdfs dfs -put data/candy_town1.csv /candy_warehouse/
```

5.2 Check the Inventory

Now that the candy data has been stored in HDFS, inspect the contents.

```
minihive@container$ hdfs dfs -ls /candy_warehouse
minihive@container$ hdfs dfs -cat /candy_warehouse/*
```

5.3 Clean Up the Warehouse

Let's remove the inventory:

```
minihive@container$ hdfs dfs -rm -r /candy_warehouse/
```

Verify that the directory has been removed:

```
minihive@container$ hdfs dfs -ls /
```

6 Running MapReduce Jobs on Hadoop

A famous MapReduce example is the WordCount problem. The goal is to count the occurrences of each word in a given dataset. In the context of Hadoop MapReduce running on HDFS, the input data is split across multiple nodes, and the map function processes each chunk of text by breaking it into individual words. The reduce function then aggregates the results by adding the occurrences of each word, providing a final count for each unique word across the entire dataset.

This problem serves as a classic demonstration of the ability of the MapReduce framework to handle large-scale data processing tasks in parallel.

6.1 Counting City Names

Inside the miniHive Docker container, enter the `hadoop` directory

```
minihive@container$ cd hadoop
```

and run `ls` to list all files.

`WordCount.java` is a Java-implementation of the WordCount problem. The folder `data` contains a file `cities.csv` with a sequence of city names. These names are unsorted and contain duplicates.

6.1.1 Compiling the Java Code

Compile the Java code:

```
minihive@container:~/hadoop$ javac -classpath $(hadoop classpath) \
de/uni_passau/minihive/WordCount.java
```

You may have to enter this command as one line, without the "\".

6.1.2 Creating a JAR

Then, create a JAR (Java ARchive) file from the compiled Java classes:

```
minihive@container:~/hadoop$ jar cf wordcount.jar -C . de
```

Use `ls -l de/uni_passau/minihive` to confirm that `.class`-files have been generated and `ls -l` to confirm that one `.jar`-file has been generated.

6.1.3 Uploading the Input Data

Upload the file `cities.csv` into HDFS, as shown below.

```
minihive@container:~/hadoop$ hdfs dfs -mkdir /user
minihive@container:~/hadoop$ hdfs dfs -mkdir /user/minihive
minihive@container:~/hadoop$ hdfs dfs -put data/cities.csv
minihive@container:~/hadoop$ hdfs dfs -ls
```

The last command should list the file `cities.csv`, now uploaded to HDFS.

6.1.4 Running the MapReduce Job

We are now ready to run the MapReduce job.

```
minihive@container:~/hadoop$ hadoop jar wordcount.jar \
de.uni_passau.minihive.WordCount cities.csv count
```

Execution may take time, even for small or toy inputs. This is because Hadoop is designed to handle large-scale, distributed data processing across multiple nodes. As a result, even simple tasks can incur overhead from initializing the distributed environment, resource allocation, and other system operations.

If successful, `hdfs dfs -ls` lists a directory `count`, which contains the result. Inspect its contents:

```
minihive@container:~/hadoop$ hdfs dfs -cat count/part*
```

When processing large inputs, HDFS will store the results in separate chunks, each prefixed `part`. Given our toy input, we expect the result to fit into a single chunk.

6.1.5 Re-Running the MapReduce Job

If a MapReduce Job fails to run, it is often due to the output directory `count` having been generated during an earlier run. In Hadoop, the output directory must not already exist, as it cannot be overwritten by default. To resolve this, you will need to delete the `count` directory before running the command again.

The following command can be used to delete the directory from HDFS:

```
minihive@container:~/hadoop$ hdfs dfs -rm -r count
```

This will remove the `count` directory, allowing you to re-run the process without errors.

6.1.6 Improving the Java Code

Inspecting the output, observe that the Java program can be improved by properly tokenizing city names. Currently, names such as `'Ayn al 'Arab` have obviously been processed incorrectly.

Improve and re-run the Java code, reiterating the steps above, for practice.

6.2 Counting the Halloween Candy Stash

Next, run the WordCount implementation against the Halloween Candy Data. This will require you to reiterate and adapt the previous steps.