Spark (Streaming) 🦵

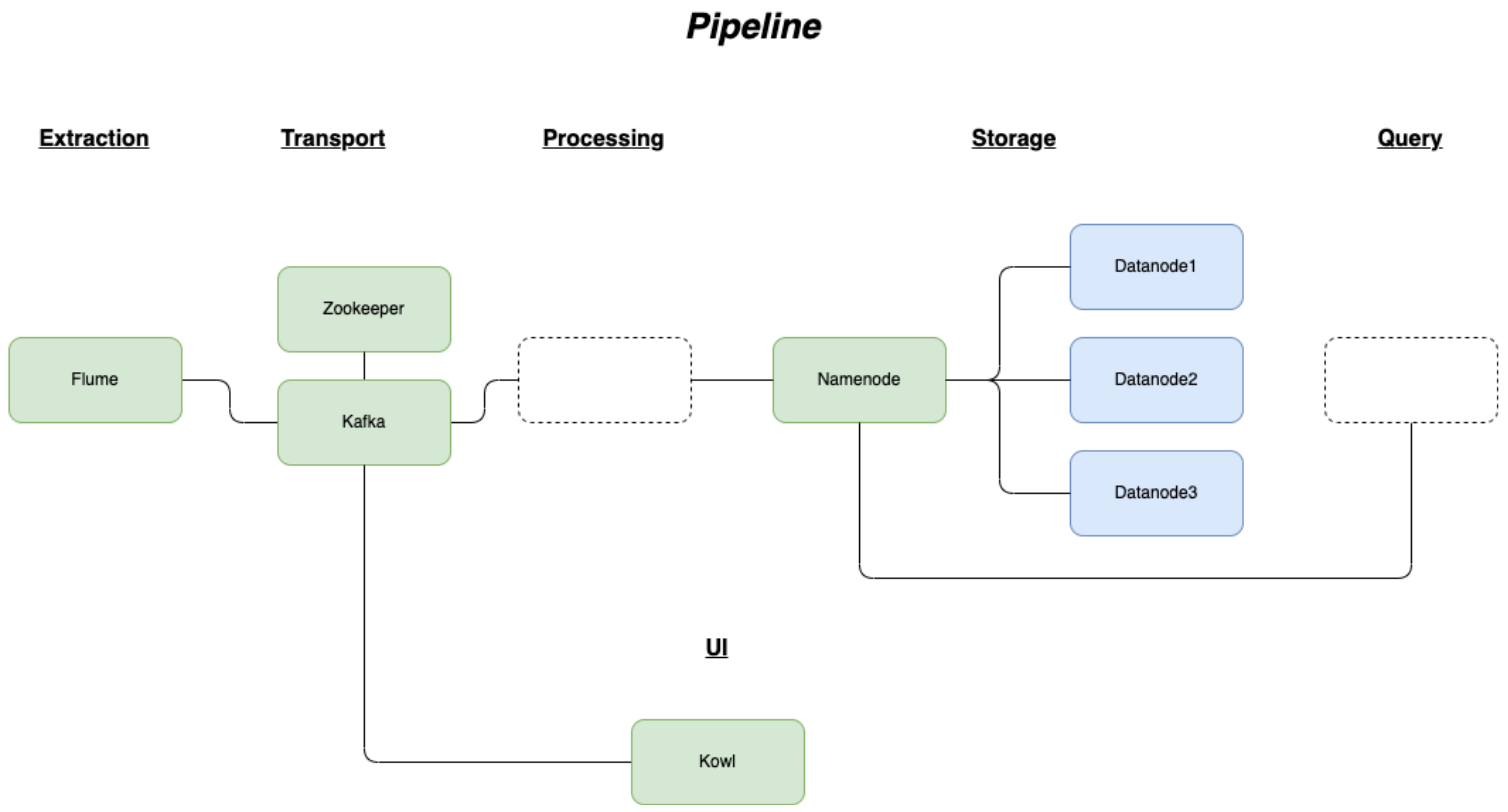
Big Data E22

Nikolai Emil Damm - nidam16@student.sdu.dk
Bende Siewertsen - besie18@student.sdu.dk

Context

What are we doing?

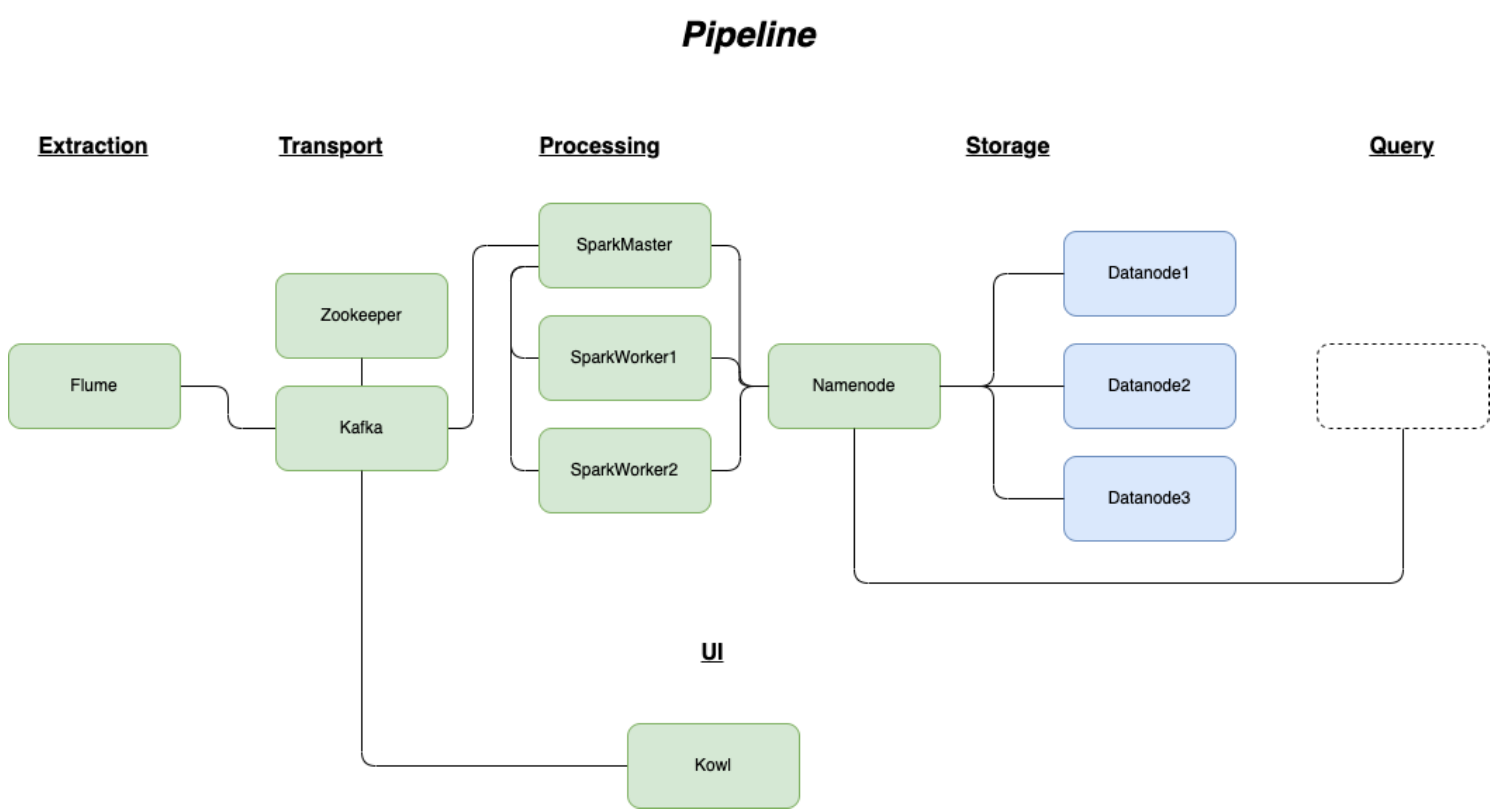
- A data-pipeline to ingest and store data.
- Last time we introduced distributed event streaming (Kafka)

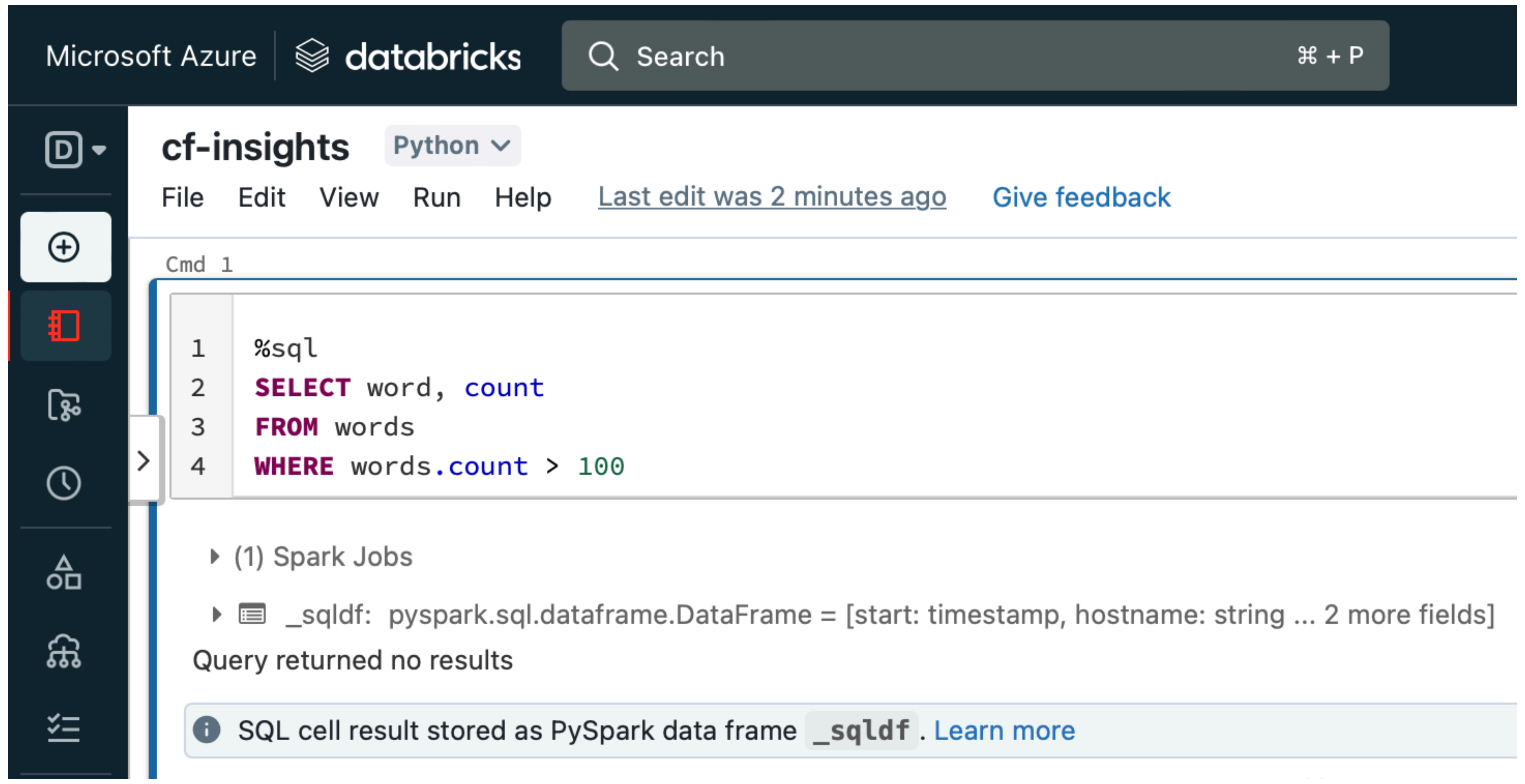


Context

What are we doing today?

- Setting up a Spark cluster for Distributed Data Processing
  - 1 Spark Master
  - 2 Spark Workers
- Why do we use Spark?
  - Sequential vs Paralellel processing.
  - Remember to use Spark!
- We will not use the Spark SQL language but we will use the API provided by Pyspark
  - Feel free to use it yourself if you want to. Pyspark supports sending SQL queries.





```
%sql
SELECT word, count
FROM words
WHERE words.count > 100
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [start: timestamp, hostname: string ... 2 more fields]

Query returned no results

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Hadoop

Interacting with Hadoop

- Quick demonstration of using Hadoop’s web UI.
    - http://localhost:9870
- Lets upload alice-in-wonderland.txt through the UI!

Spark

Interacting with Spark

- Quick demonstration of using Spark’s web UI.
  - Master: http://localhost:8080
  - Worker 1: http://localhost:8081
  - Worker 2: http://localhost:8082
- Visualizes running Spark Jobs, resource usage, and the state of jobs

Exercise 01

Composing a Spark Cluster

- *All docker-compose files have been updated to compose all services needed for the days exercise. For example will exercise 3's docker compose file compose Hadoop, Kafka and Spark.*

- To work with Spark we need to setup a Spark cluster with a Spark Master and Spark Workers.

- To do this you need to do the following.

  1. Teardown previous clusters e.g. Hadoop and Kafka. You can use `docker system prune` after stopping the stacks

  2. cd `./lecture04-exercises/`

  3. Examine the `docker-compose.yml` file

  4. Run `docker compose up -d`. Remove the argument `-d` if you need to see what happens.

Exercise 02

Upload Alice In Wonderland to Hadoop

1. As we are starting with a fresh cluster, we need to reupload Alice In Wonderland to Hadoop.
2. We can do this in two ways:
    1. With the CLI as described in Lecture 02 Exercise 03
    2. With the Hadoop UI as demonstrated earlier.
        1. This does not work with our current configuration, but you might be able to find a solution if you really need this.
3. You should upload the file to /stream-in/ and it should be called alice-in-wonderland.txt

Exercise 03

Running a Spark Job 🤩

- Start a clustered spark job
  - You can also run it locally, but then its not distributed 😉
  - There is a local example in lecture04-exercises/localPyspark for those interested
- Examine the python file, what does it do?
- TODO:
  - Navigate to ./lecture04-exercises/clusterPyspark
  - Start the spark job by running bash run.cmd
- What happened?

Exercise 04

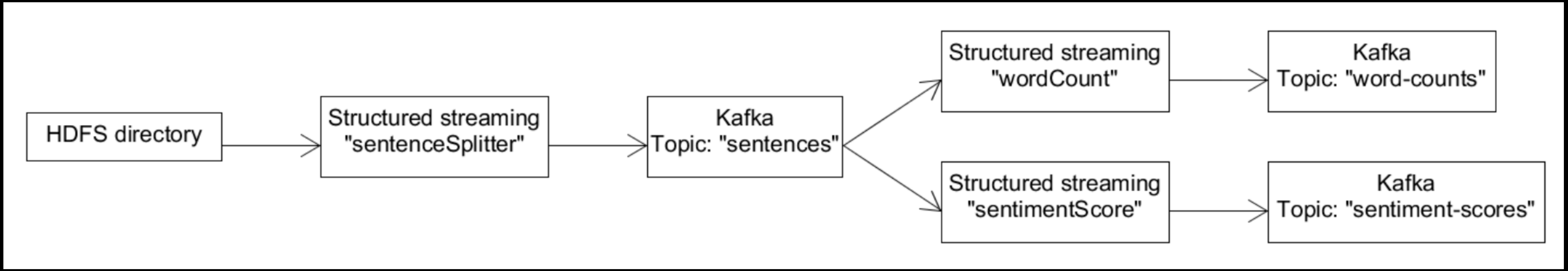
Sentiment exercise - Is Alice Mad or Happy?

Now lets set up a spark job that can analyse the alice-in-wonderland.txt to check whether a sentence is positive or negative.

1. cd into ./lecture04-exercises/sentimentExercise
2. Examine the Dockerfile, the example.py, the requirements.txt, and the run.cmd files.
   - The example.py streams data from files in /stream-in/ with Spark, and finds and prints the top words in the files.
3. Start the spark job by running bash run.cmd.
4. Now you must extend the solution to:
   1. To calculate the overall sentiment score of streamed files.
   2. Your solution should make use of a function that:
      1. Returns 1 if a word matches a positive word
      2. Returns -1 if a word matches a negative word
      3. Returns 0 if a word does not match a positive or negative word.
   3. Use this function on each word and accumulate the results to get the total sentiment score.
   4. There are hints in the solution on what you need to change if you get stuck!

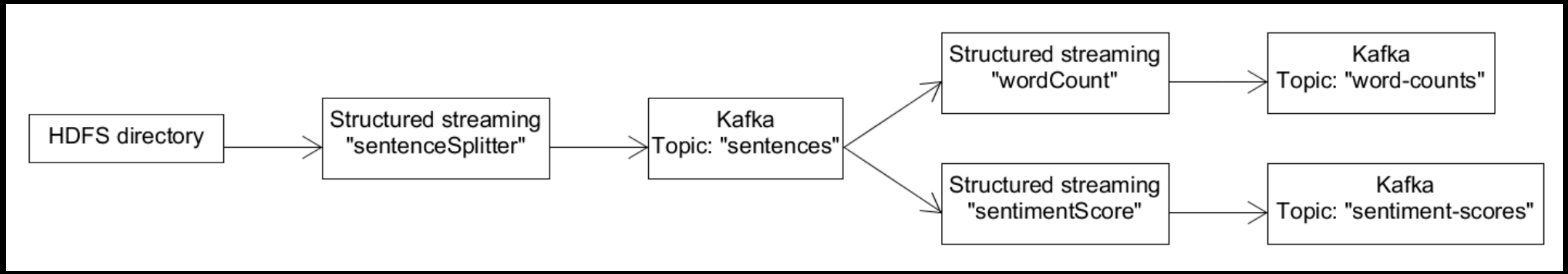
Exercise 05

Sentiment exercise extended - Is Alice Really Mad or Happy?


HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

Exercise 05

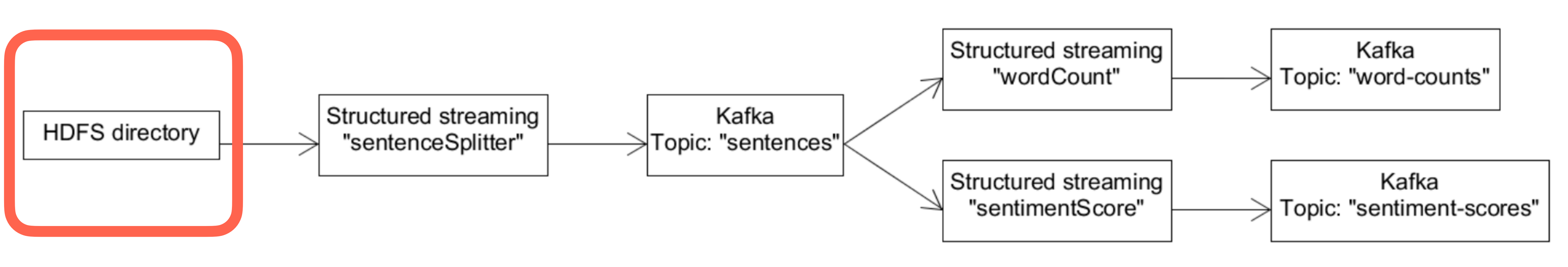
Overview


HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

1. Create a HDFS directory
2. Set up a Spark job (Structured Streaming) responsible for:
    - Reading files from a HDFS directory.
    - Splitting content of files into sentences.
    - Writing sentences to a Kafka topic.
3. Set up a Spark job (Structured Streaming) responsible for:
    - Reading data from a Kafka topic.
    - Counting words in sentences.
    - Writing word counts to a Kafka topic.
4. Create a another Spark Job (Structured Streaming) responsible for:
    - Reading data from a Kafka topic.
    - Calculating a sentiment score sentences.
    - Writing sentiment scores to a Kafka topic.

Exercise 05

Creating a HDFS directory

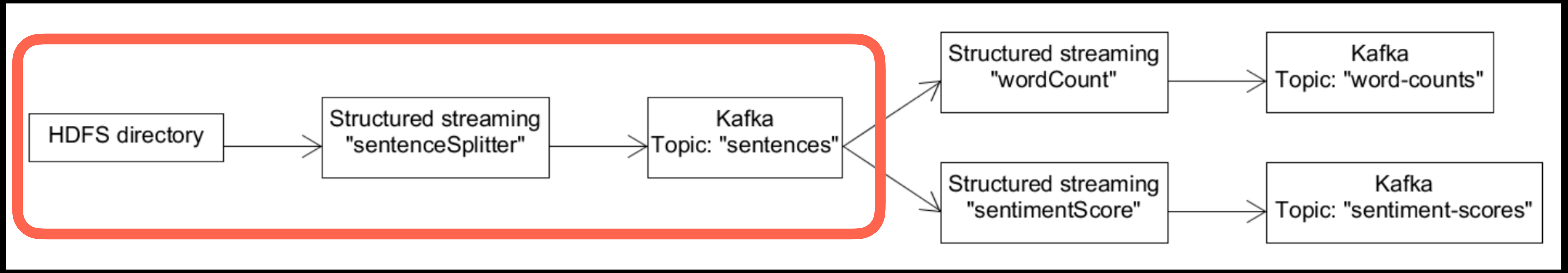

HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

Let's create a folder that can be used as a data ingestion source!

1. With the CLI:
   1. docker exec -ti namenode bash
   2. hdfs dfs -mkdir /stream-in - Creates a directory in HDFS called stream-in
2. With the Hadoop UI
   1. Go to http://localhost:9870
   2. Go to Utilities > Browse the file system > Create a directory

Exercise 05

Splitting Sentences! 🖖


HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

Now lets set up a spark job to continuously split sentences from files added to the newly created HDFS folder.

1. cd into ./lecture04-exercises/sentenceSplitter

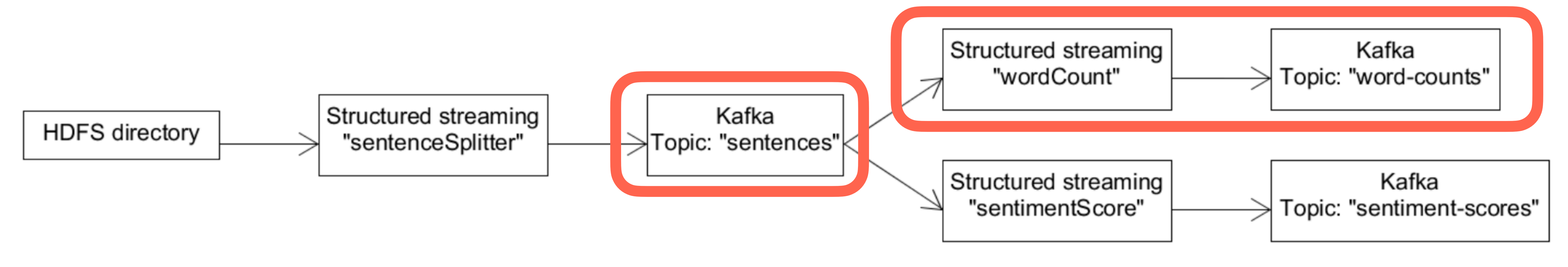
2. Examine the Dockerfile, the example.py, the requirements.txt, and the run.cmd files

3. Start the spark job by running bash run.cmd

4. To trigger the stream, upload the alice-in-wonderland.txt to the hdfs folder "stream-in" (Might have to do it again)

Exercise 05

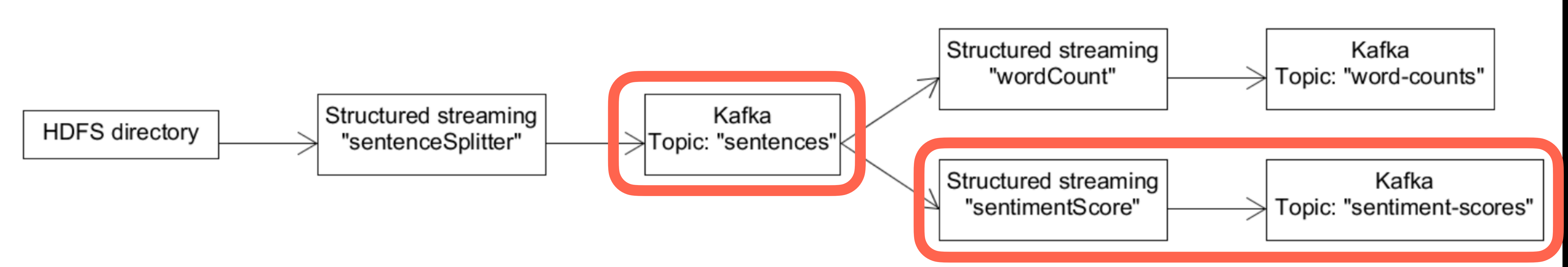
Count the words! 📈


HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

1. Just as you did before with simple spark, your job is to count the words, but this time as a Stream!
   1. Read the sentences you have pushed to kafka in the previous exercise
   2. Count the words
   3. Send it back to Kafka!
2. TODO:
   1. Remember to examine the python file, youll need it in the next exercise 😉
   2. Navigate to ./lecture04-exercises\wordCounter
   3. Start the spark job by running bash run.cmd
   4. To trigger the stream, upload the alice-in-wonderland.txt AGAIN to the hdfs folder "stream-in" you just created

Exercise 05

Your Turn!
Calculate the Sentiment Score 📈


HDFS directory
Structured streaming "sentenceSplitter"
Kafka Topic: "sentences"
Structured streaming "wordCount"
Kafka Topic: "word-counts"
Structured streaming "sentimentScore"
Kafka Topic: "sentiment-scores"

1. Just as you did before with simple spark, your job is to figure out if a word is happy 😊 or sad 😔 but this time as a Stream!
2. Read the sentences you have pushed to kafka in the previous exercise
3. Calculate the sentiment score for each word
4. Send it back to Kafka!
5. TODO:
    1. Upps, the solution was not uploaded, its your job to find a solution!
        1. Change the python file to calculate the sentiment score!
    2. Navigate to ./lecture04-exercises\SentimentExerciseExtended
    3. Start the spark job by running bash run.cm