Competence Network

Hadoop Fundamentals

Daniel Müller | 19.09.2019

About us

Seamless Analytics GmbH

- Founded in 2018
    - Marvin Follmann, M. Sc.
    - Daniel Müller, M. Sc.

- Consultation and Service provider
    - Project planning and implementation
    - Apache Hadoop ecosystem
    - Data analysis
    - Machine Learning and Deep Learning

- The Idea originated through research projects and scientific works
    - Hochschule Offenburg

https://seamless-analytics.de

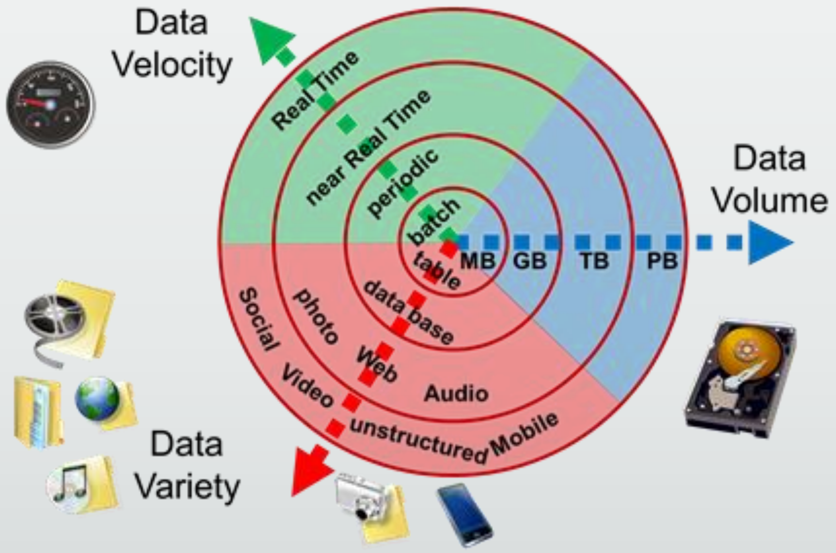
Why Hadoop in a ML Workshop?

	HDFS (Data Storage)	MapReduce / Spark (Data Processing)
Distribution	☺	☺
Reliability	☺	☺
Scalability	☺	☺

- Machine Learning usually requires a large volume of data
    - ideally containing both historical and current data
    - platform needed, that can handle this size of data

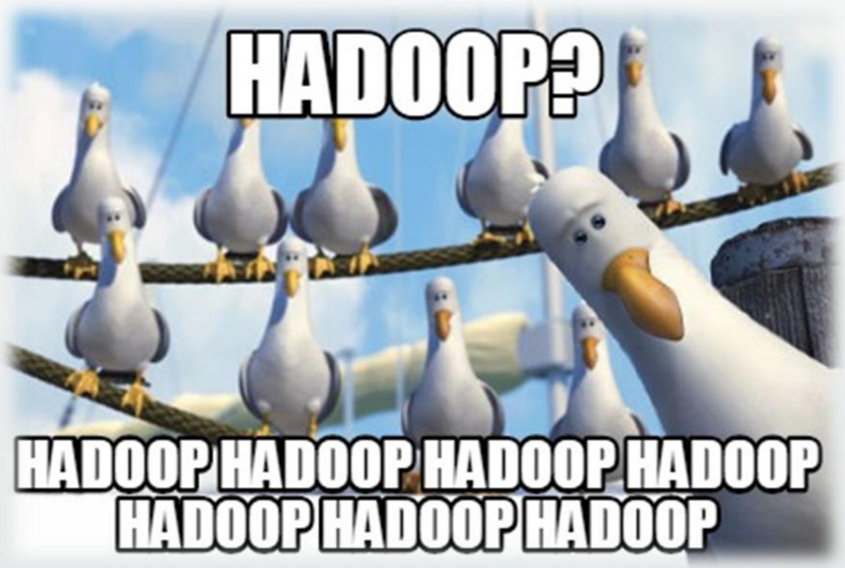
- Apache Hadoop is one of the best-known Big Data platforms
    - offers possibilities for storing and analyzing immense amounts of data
    - good solution for providing a data lake e.g. in company

⇒ Hadoop is an ideal foundation for ML and DL



Agenda of Today's Workshop

- Hadoop in General
- Ambari
- HDFS
- MapReduce
- Hive
- Spark
- YARN
- Zeppelin



Apache Hadoop

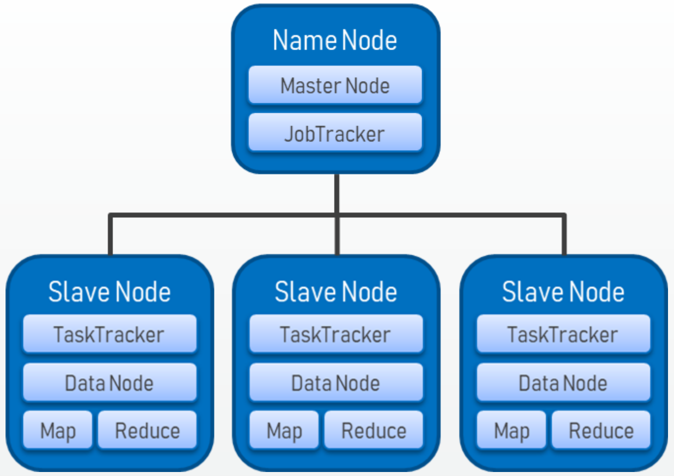
The Big Data Platform in General

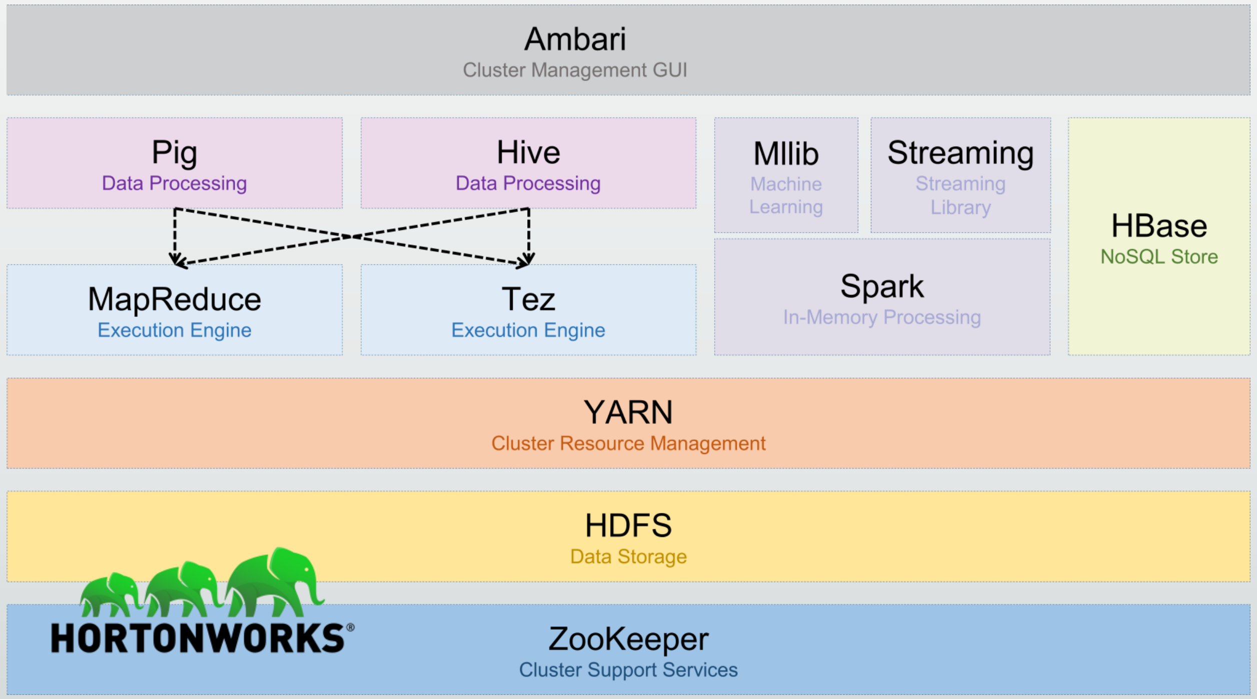
The Apache Hadoop Framework

- Framework for distributed data storing and analysis
  - Collection of tools for various tasks
    - **HDFS**: Distributed Data Storage in a cluster
    - **Hive**: SQL interface for structured data
    - **HBase**: Distributed NoSQL database
    - **Spark**: In-memory execution engine

- Organization of the platform as a cluster
  - Master-Slave architecture
  - Data replication via data nodes
  - Distributed computing

- Hadoop distributions
  - Hortonworks Data Platform (HDP)
  - Cloudera, MapR

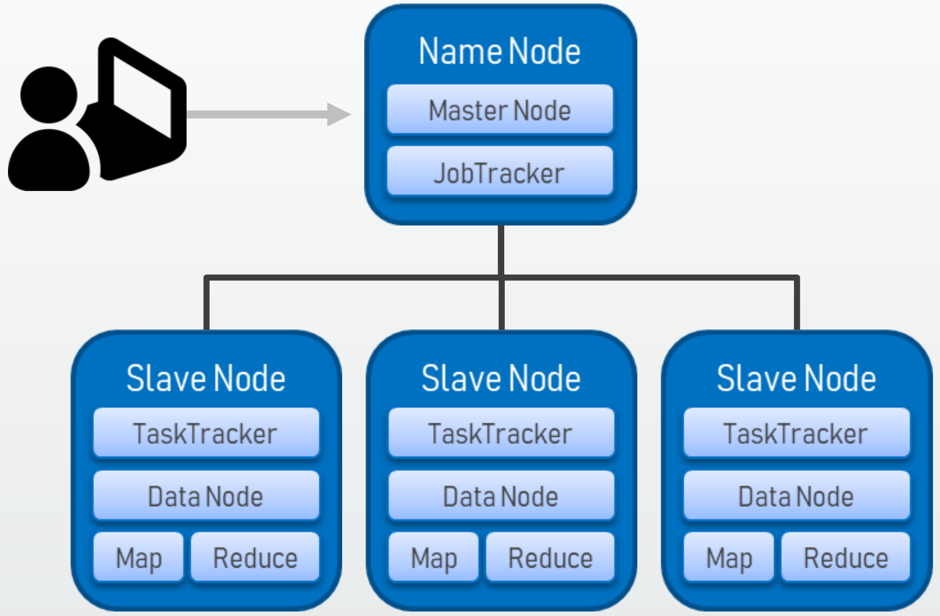




Advantages of Hadoop

- **Open Source**
  - Free solution with optional support
- **Horizontal scaling**
  - Cluster enlarging by adding new nodes
- **„Program runs at data‘s location“**
  - New paradigms for processing large amounts of data
- **Large developer community**
  - Active further development and many sources of information
- **Numerous interfaces**
  - Good connection to and expansion of existing applications

„Data center as a computer“



Usage of the Cluster in Today‘s Workshop

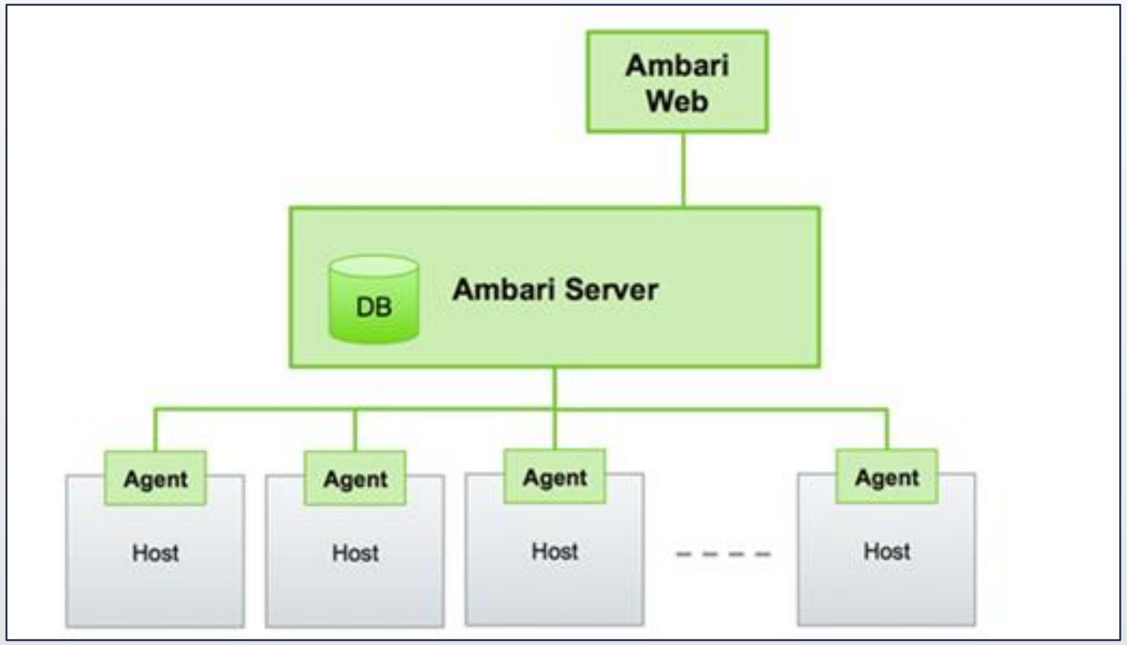
- **Ambari**
  - General overview
- **HDFS**
  - Upload files into HDFS using the terminal
- **MapReduce**
  - Letter / word count example (on slides)
- **Hive**
  - Create and query tables using Beeline (JDBC)
- **Spark**
  - Letter / word count on MovieLens dataset
- **Zeppelin**
  - Preparing MovieLens dataset for ML task

Apache Ambari

Cluster Installation and Management

Installation of the Hortonworks Data Platform

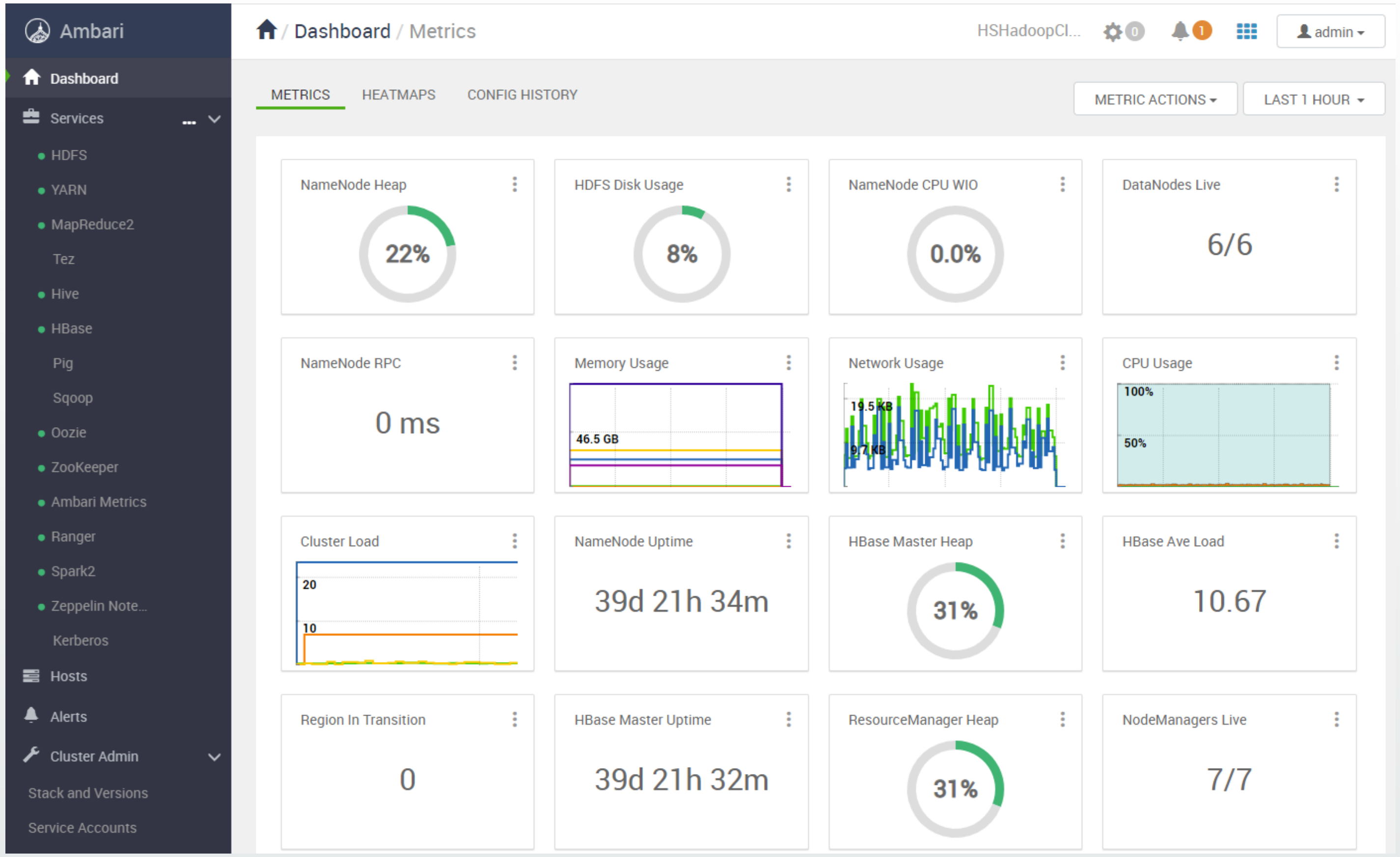
1. Setup Linux OS on the nodes, that will be part of the cluster
   - Mount hard disks / partitions used for HDFS data
2. Decide which node to use for Ambari Server
   - Setup password-less access to other nodes
3. Install Ambari Server
   - Download and run installer
4. Start Hadoop setup via Ambari UI
   - Provide the list of nodes during installation
   - Configure, which service to run on which node
5. Use Hadoop ☺
   - Ambari Server for cluster administration



Management of the Cluster

- **Apache Ambari - Server (HDP)**
  - Installation of a cluster
  - Global service configuration
  - Web interface
    - Administrators
    - Users (Ambari Views)

- **Ambari - Agents**
  - Communication with the server
  - Installed on every node of the cluster
    - Receiving the configurations
    - Monitoring of components



NameNode Heap 22%
HDFS Disk Usage 8%
NameNode CPU WIO 0.0%
DataNodes Live 6/6
NameNode RPC 0 ms
Memory Usage
Network Usage
CPU Usage
Cluster Load
NameNode Uptime 39d 21h 34m
HBase Master Heap 31%
HBase Ave Load 10.67
Region In Transition 0
HBase Master Uptime 39d 21h 32m
ResourceManager Heap 31%
NodeManagers Live 7/7

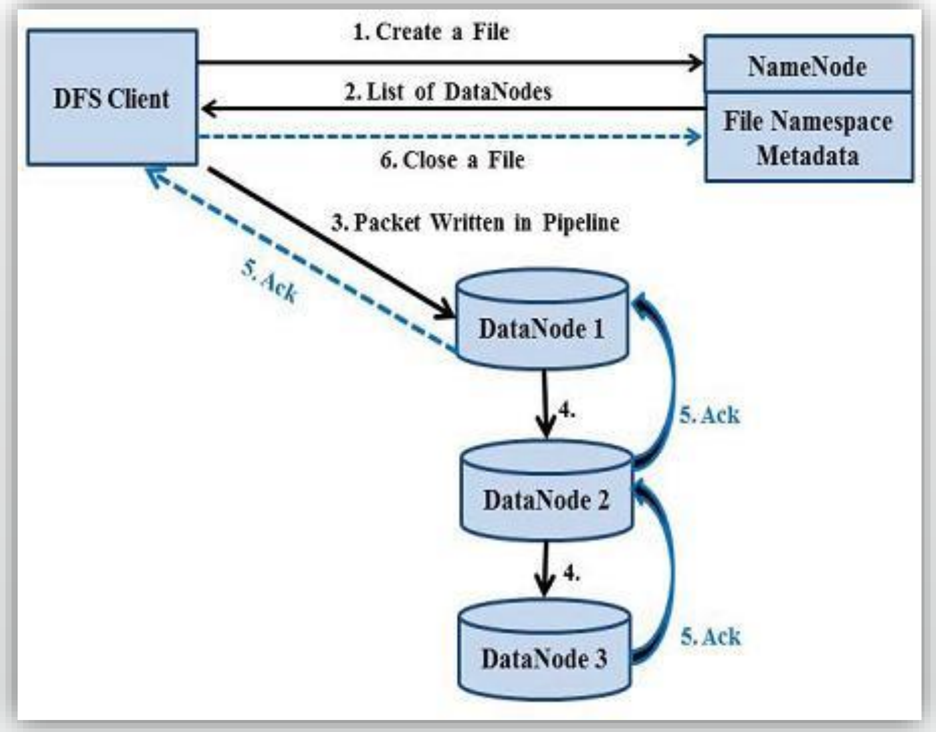
http://<ambari-server-host>:8080

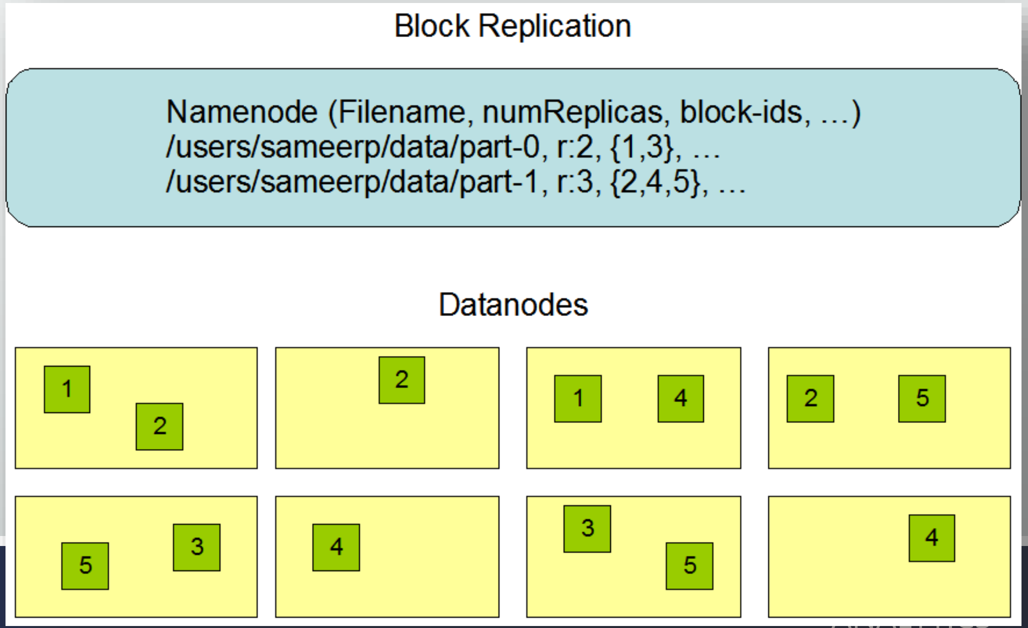
HDFS

Hadoop Distributed File System

HDFS - Hadoop Distributed File System

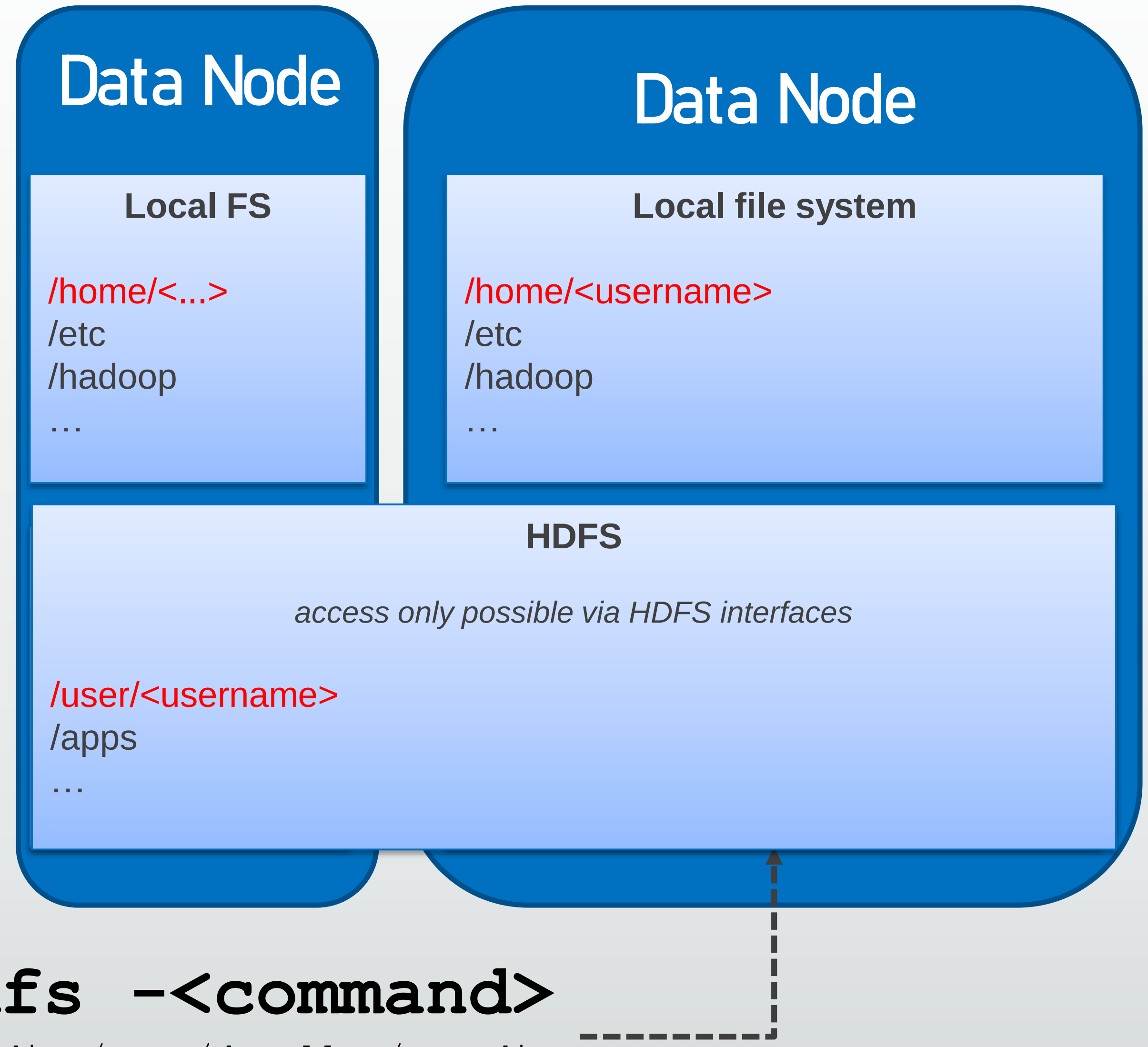
- Distributed data storage in the Hadoop cluster
  - Subdivision into NameNode and DataNodes
    - **NameNode**: Administration / Management
    - **DataNode**: Data holding, access and processing
- Redundant distribution over the DataNodes
  - Replication → Reliability and failure safety
  - No RAID or similar necessary
- Data blocks of fixed size (128 MB)
  - Optimized for fast reading of large files




Block Replication
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

HDFS Structure and Interaction

- **General file administration based on Linux file system**
  - Read, Write, Execute for Owner, Group, Others
  - More detailed permissions via Apache Ranger and ACLs
- **Fusion of local system user and Ambari/Hadoop user**
  - Access to HDFS and Ambari Server with the same username
- **User interfaces**
  - CLI, Web UI, Network Drive (NFS), API
- **Notes**
  - Risk of confusion: **local data storage ≠ HDFS**



```
hdfs dfs -<command>
hdfs dfs -mkdir /user/dmueller/testdir
hdfs dfs -chmod 700 /user/dmueller
```

Exercise: File Upload to HDFS

1. Upload local files into HDFS
   - MovieLens dataset
   - Server log file
2. Using HDFS commands to show / copy files inside HDFS
3. Using the Ambari Files View

Help: `hdfs dfs`

https://github.com/seamless-analytics/summer-school-2019/Day 1 - Hadoop/Task 1 - HDFS

The Hadoop Clusters in this Workshop

- Hadoop Cluster 1
  - Sponsored by **badenIT** GmbH (DE - Freiburg)
  - Running on cloud instances
  - NameNode: **213.164.81.211**
- Hadoop Cluster 2
  - Setup in the network of the university
  - Running in local network
  - NameNode: **bda-job.emi.hs-offenburg.de**
- **Allocation of participants**
  - User student1 … student30
  - Odd user id (1, 3, …) → use Cluster 1
  - Even user id (2, 4, …) → use Cluster 2

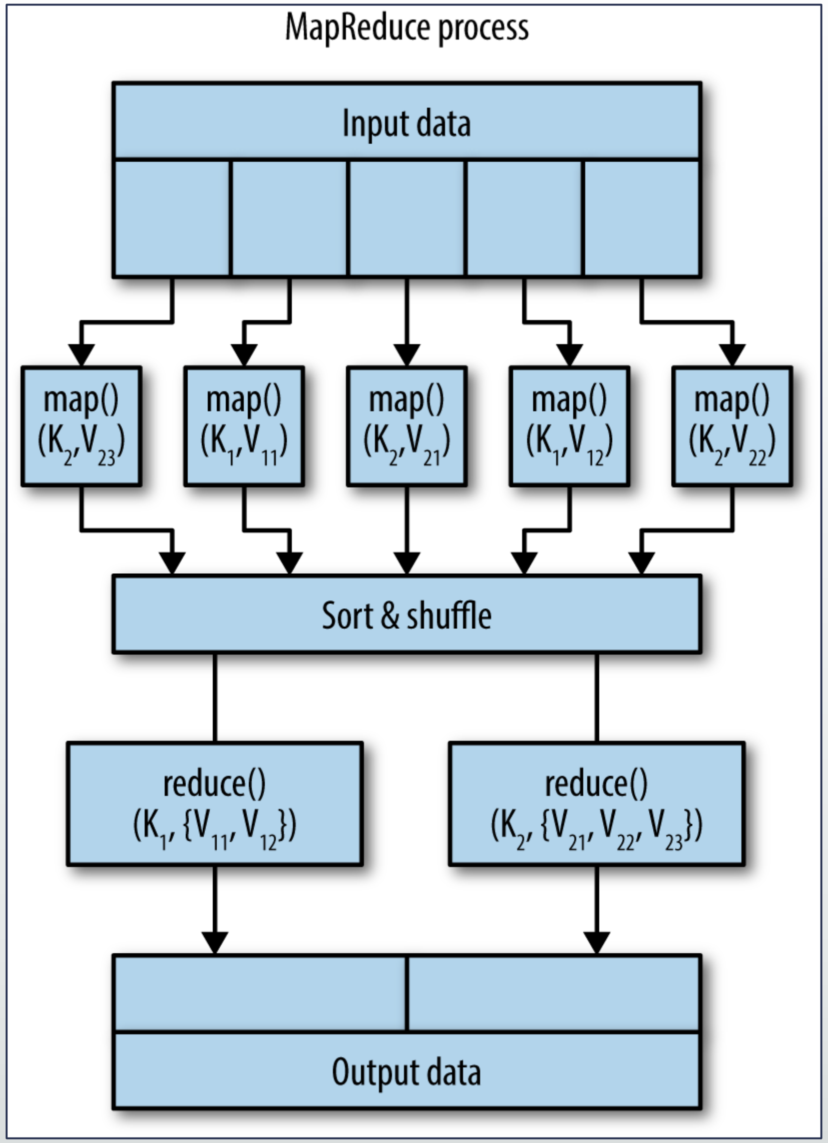
MapReduce

Paradigm / Algorithm for Parallel Data Processing

MapReduce

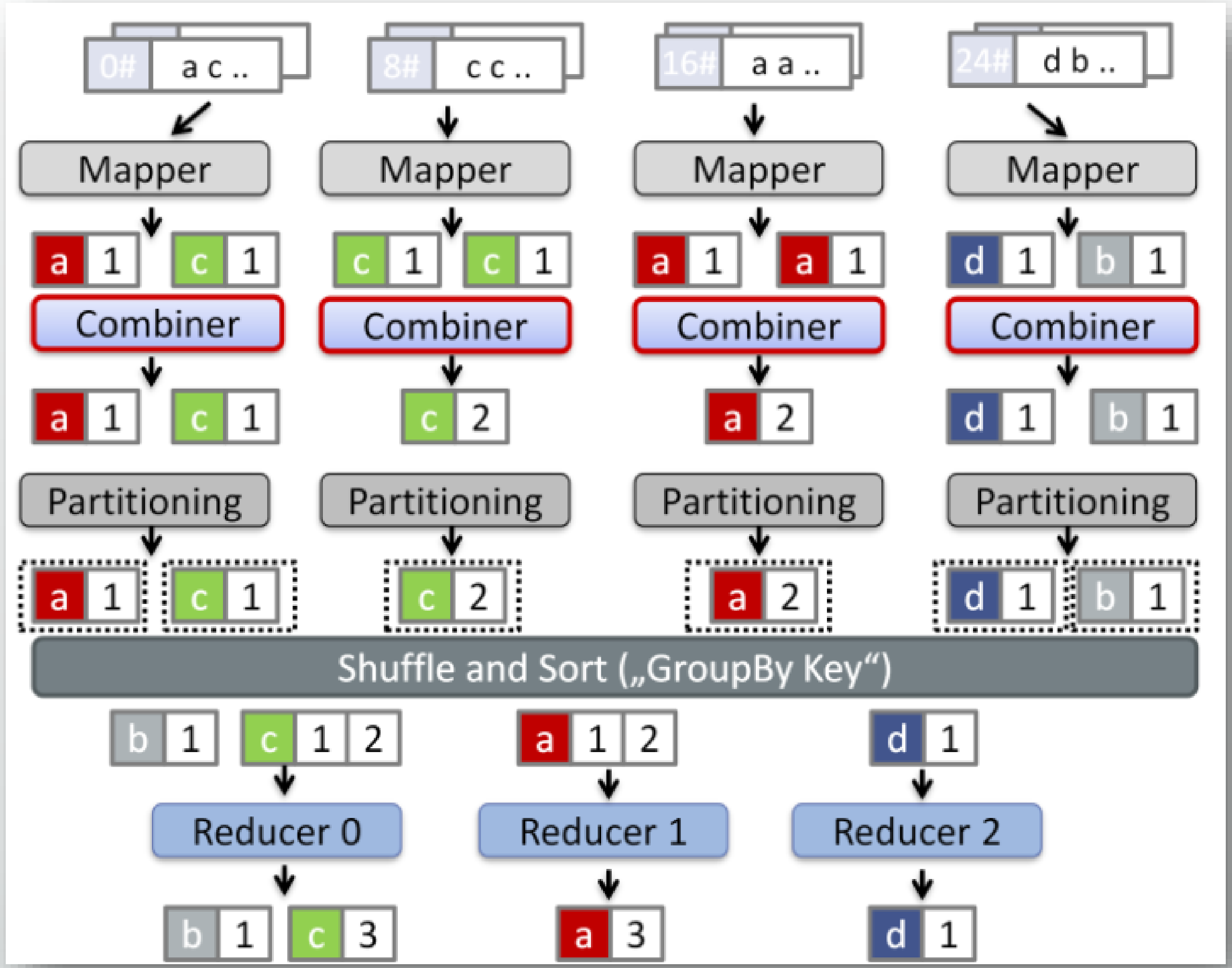
- **Basic programming model in Hadoop**
  - „Program runs at data's location"
  - Distributed calculations across multiple processing units
  - Calculate partial results (**map**)
    - merge them afterwards (**reduce**)

- **Prerequisites**
  - It must be possible to split data
  - Data is stored in a distributed manner inside the cluster (HDFS)
  - Independent calculations on the individual partitions possible



MapReduce - Optimizations

- **Combiner**
  - Merges key-value pairs within a node
    - Therefore reduces the network load
- **Partitioner**
  - Controls the distribution of intermediate results to the reducers
    - More even distribution of work

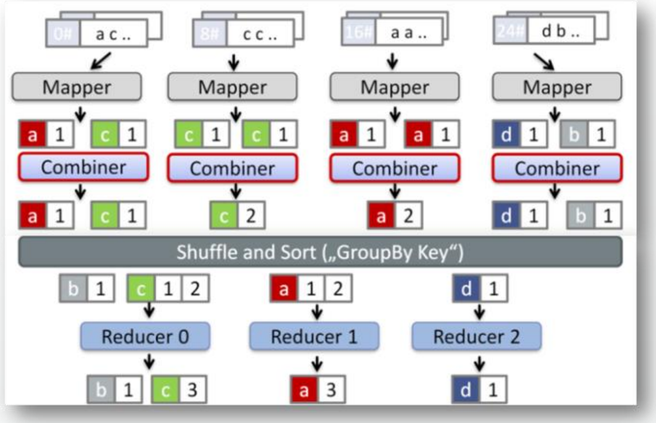


Exercise: Letter Count

- How many words start with a specific letter?
- Input text:

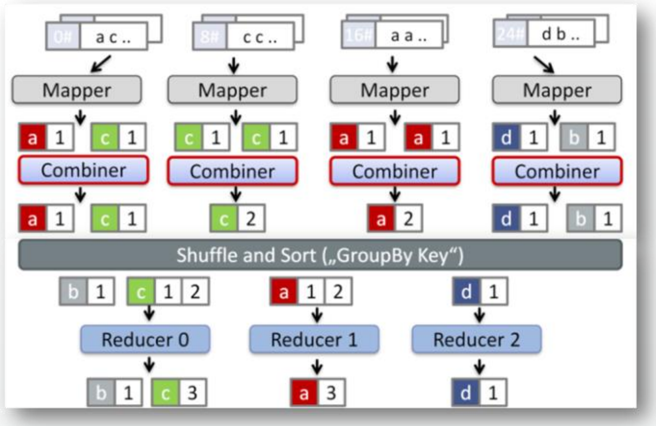
```
1 :: „The MapReduce technique processes the data in three phases (Map, Shuffle, Reduce),
2 :: two of which are specified by the user (Map and Reduce). This allows calculations to
3 :: be parallelized and distributed across multiple computers. For very large amounts of
4 :: data, parallelization may be necessary because the amount of data is too large for a
5 :: single process (and the executing computer system).“
```

Exercise: Letter Count



- Step 1: Each line of text is assigned to a mapper.
  - Example – Mapper 1 – Input:
    „The MapReduce technique processes the data in three phases (Map, Shuffle, Reduce),"
  - Preprocessing: Remove special characters and punctuation marks and convert the text to lower-case letters:
    „the mapreduce technique processes the data in three phases map shuffle reduce"
  - Split text into the individual words:
    { the, mapreduce, technique, processes, the, data, in, three, phases, map, shuffle, reduce }
  - Iterate over the word list and form pairs of the form [<Inital letter>, 1] → Mapper finished
    [t, 1], [m, 1], [t, 1], [p, 1], [t, 1], [d, 1], [i, 1], [t, 1], [p, 1], [m, 1], [s, 1], [r, 1]
- (Optional) Combiner: Merge the data at mapper level:
  [t, 4], [m, 2], [p, 2], [d, 1], [i, 1], [s, 1], [r, 1]

Exercise: Letter Count



- Step 2: Shuffling sorts and assigns the keys of the KV pairs to reducers
  - Input are the results of the mapper (or combiner)
    - Mapper 1: *[t, 4], [m, 2], [p, 2], [d, 1], [i, 1], [s, 1], [r, 1]*
    - Mapper 2: *[t, 4], [o, 1], [w, 1], [a, 3], [s, 1], [b, 1], [u, 1], [m, 1], [r, 1], [c, 1]*
    - Mapper 3: *[b, 1], [p, 1], [a, 3], [d, 1], [m, 1], [c, 1], [f, 1], [v, 1], [l, 1], [o, 1]*
    - …
  - Each reducer is responsible for processing a key
    - Input is the key and a list of the values calculated in each mapper → One entry per text line
    - Output for this example should be: [<Initial letter>, <Frequency in text>]

    - Example:
      Reducer **t**: Input: *{ t, [4, 4, 2, 1] }* → *[t, (4 + 4 + 2 + 1)]* → ***[t, 11]***
      Reducer **m**: Input: *{ m, [2, 1, 1, 1] }* → *[m, (2 + 1 + 1 + 1)]* → ***[m, 5]***
      …

- Return of all reducer results by the driver process

Apache Hive

SQL Interface on Structured Data

Apache Hive

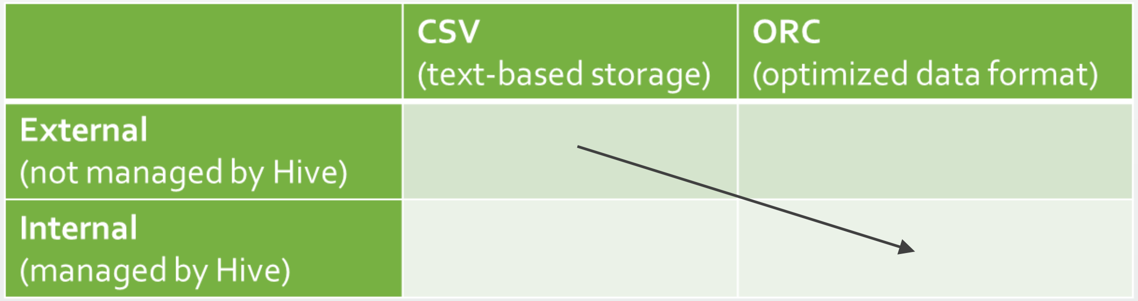
- **Data warehouse system for querying and analyzing structured datasets**
  - SQL engine for Hadoop data
  - Enables access to structured data in HDFS via SQL-like queries (HiveQL)
  - Execution of tasks via Tez jobs

- **Different interfaces**
  - JDBC, CLI (Beeline), SparkSQL
    - *jdbc:hive2://<hiveserver-url>:10000/<database>*

- **Notice**
  - Hive is not a full-featured database system
    - Duplicates possible
    - No row keys (nor foreign keys)
  - Build for fast reads of huge datasets
    - Partitioning, bucketing, etc.

Apache Hive

- **Hive can work with different types of data**
  - Text-based (CSV)
  - Avro
  - Parquet
  - **ORC**
    - Column-based storage format
    - Segmentation into „Stripes"
    - Compression (ZLIB, Snappy)
    - Predicate Pushdown
    - Partitioning



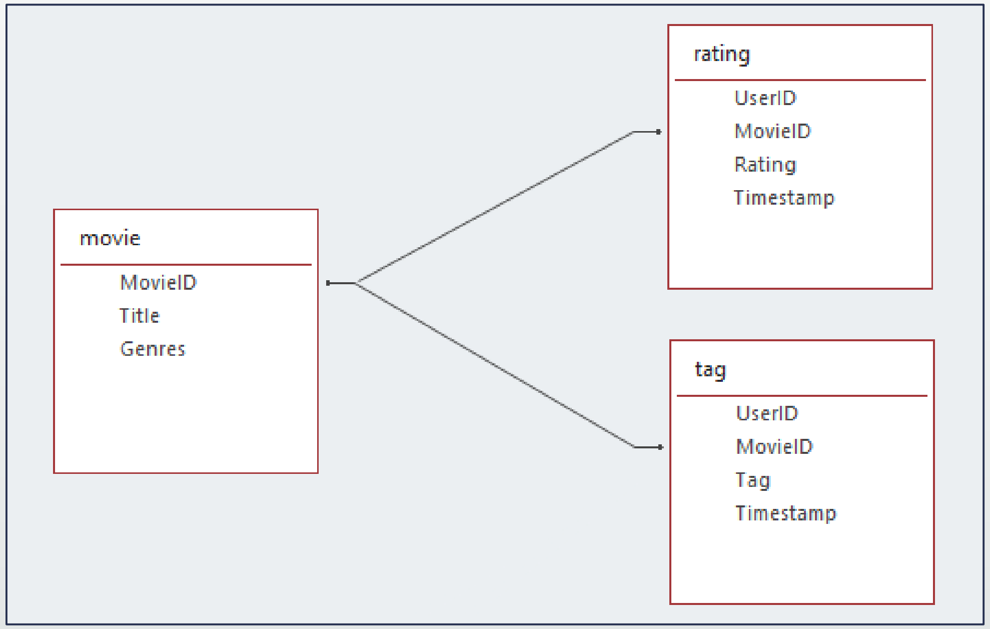
	CSV (text-based storage)	**ORC** (optimized data format)
External (not managed by Hive)		
Internal (managed by Hive)		

- **Storage of data as internal or external tables**
  - Intern: Managed by Hive
    - Storage in Hive warehouse directory in HDFS (hdfs:/warehouse/tablespace/managed/hive)
    - Deleting the table also deletes the data
  - Extern: Hive refers only to the data
    - Storage in any HDFS directory
    - Deleting the table only deletes the reference

Exercise: The MovieLens Dataset

The dataset contains ratings and free-text tagging activities from MovieLens, a movie recommendation service.

- ~27,000,000 ratings
- ~1,100,000 tag applications
- ~58,000 movies
- created by ~280,000 users
  - Users were selected at random for inclusion



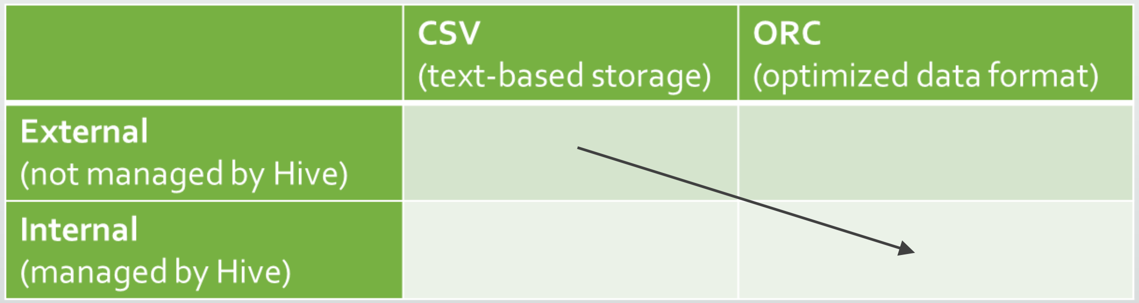
- Source: https://www.kaggle.com/grouplens/movielens-20m-dataset

Exercise: MovieLens

Goal: Use Apache Hive to query the MovieLens dataset

1. Create an internal Hive table in ORC format, matching the MovieLens CSV structure
2. Create an external Hive table that refers to the corresponding CSV file in HDFS
3. Transfer data from the external to the internal table

   → This converts the text-based format into ORC format
4. Fill the *movies* table by using Spark (spark-shell)
5. Query the tables using Beeline (JDBC)



	CSV (text-based storage)	**ORC** (optimized data format)
External (not managed by Hive)		
Internal (managed by Hive)		

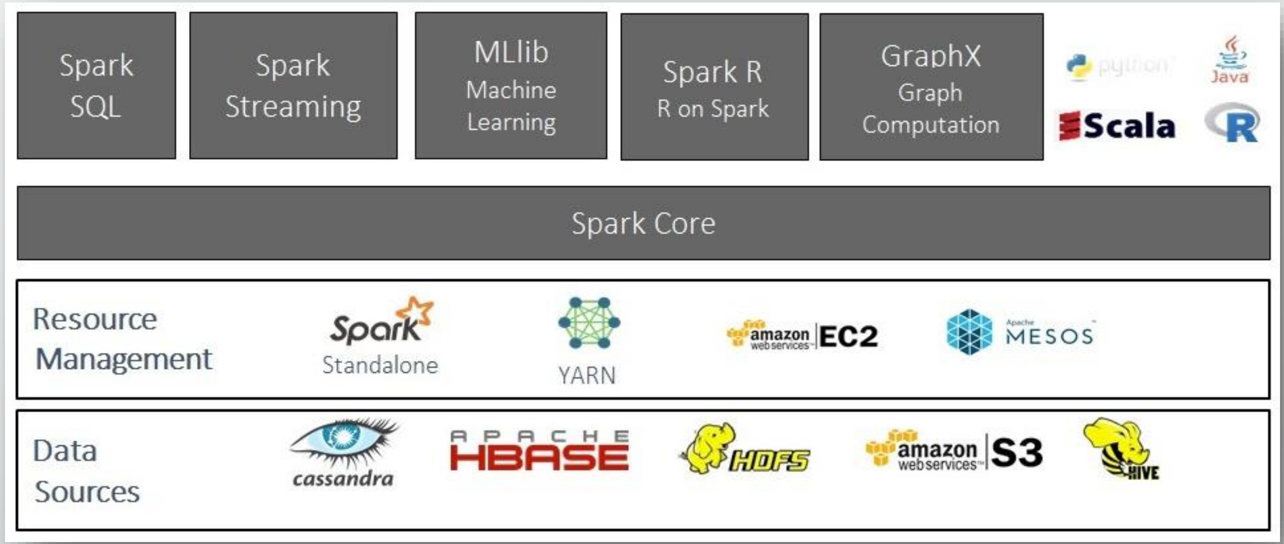
https://github.com/seamless-analytics/summer-school-2019/Day 1 - Hadoop/Task 2 - Apache Hive

Apache Spark

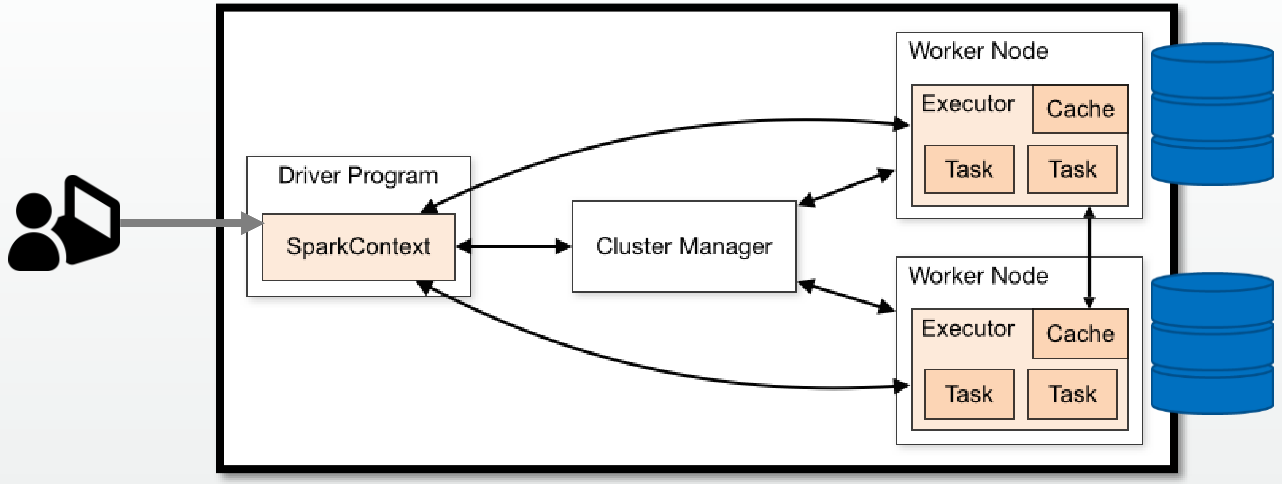
Execution Engine in Hadoop Ecosystem

Apache Spark in General

- Unified and general platform for distributed data processing
  - Use of **in-memory computing** for caching data and sub-results
- Abstraction via so-called **Resilient Distributed Datasets** (RDD) or **DataFrame**
  - „Lifelines" of the data partitions
  - Transformations (changing the data)
  - Actions (output / return of results)
- Extensive **API** (Java, Scala, Python, R)
  - Access to data abstractions RDD and DataFrame
  - Performing transformations and actions
- Organized as master/slave system
  - Driver & Executors



Apache Spark – Driver & Executors



- Driver Process
  - Organization of execution
  - Management of worker threads
  - Resource requests to YARN
  - References the SparkContext / SparkSession
  - Gathering of the partial results to an overall result

- Multiple Executor / Worker processes
  - Optimally, they run directly on the nodes that contain the data
    - **Program runs at data's location, not vice versa**
  - Calculate partial results based on the **data partitions** assigned to them

Apache Spark – Programming (Java)

Bsp: Maven Projekt - pom.xml

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.3.2</version>
    <scope>provided</scope>
</dependency>
```

```
spark-core
spark-sql
spark-mllib
spark-streaming
```

Initialization
→ *Generating a Spark session and/or contexts*

Transformations
→ *Create and manipulate RDDs and datasets*

Actions
→ *Getting the results and use them on the Driver process*
→ *start the processing*

```
SparkSession session = SparkSession.builder
                                     .master("yarn-client")
                                     .appName("Word Count")
                                     .config("spark.some.config.option", "some-value")
                                     .enableHiveSupport()
                                     .getOrCreate();

SQLContext sqlContext = session.sqlContext();
SparkContext sc = session.sparkContext();

Dataset<Row> ds = sqlContext.sql("SELECT * FROM db.table").filter(...);
Dataset<MyObject> myDs = ds.map(...);
...

RDD<String> rdd = sc.textFile("hdfs:/user/test/myfile.txt").filter(...);
RDD<MyObject> myRdd = rdd.map(...);
...

System.out.println(myDs.count());
System.out.println(myRdd.count());
```

Spark API JavaDoc (Core)

https://spark.apache.org/docs/2.3.2/api/java/index.html?org/apache/spark/api/java/JavaSparkContext.html

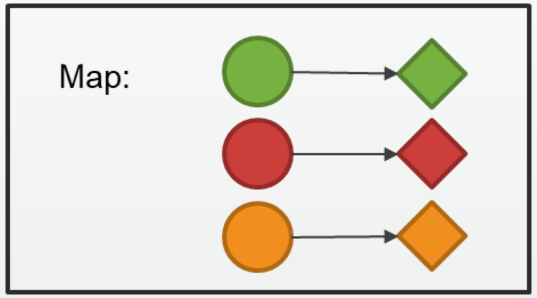
```
org.apache.spark.api.java.JavaSparkContext

...
textFile(String path):                 JavaRDD<String>
```

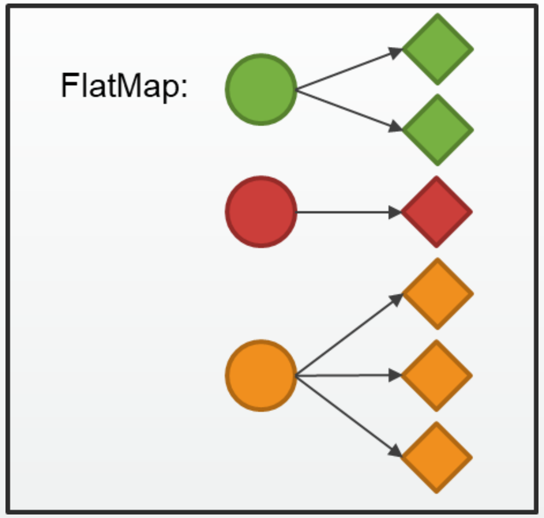
https://spark.apache.org/docs/2.3.2/api/java/index.html?org/apache/spark/api/java/JavaRDD.html

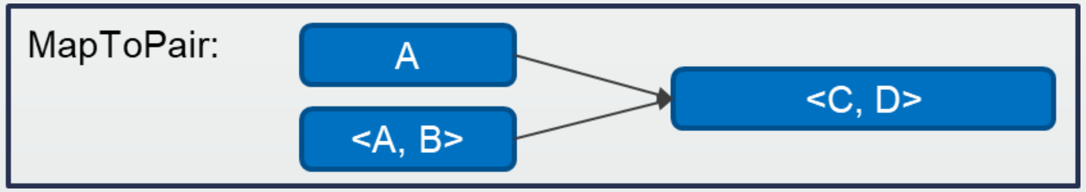
```
org.apache.spark.api.java.JavaRDD<T>

...
flatMap(FlatMapFunction<T,U> f):       JavaRDD<U>
map(Function<T,R> f):                  JavaRDD<R>
mapToPair(PairFunction<T,K,V> f):      JavaPairRDD<K,V>
collect():                             List<T>
take(int num):                         List<T>
...
```

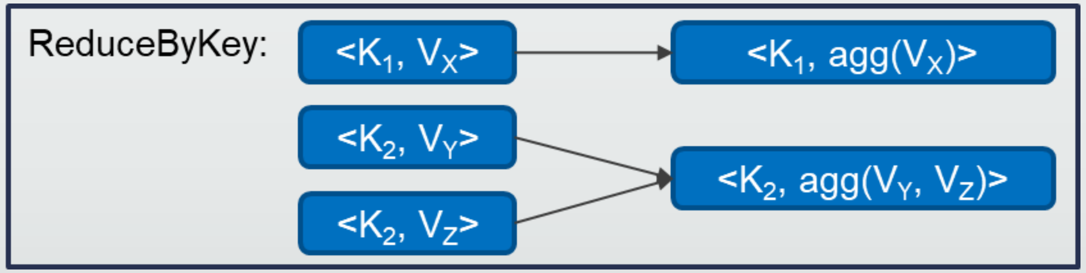
https://spark.apache.org/docs/2.3.2/api/java/index.html?org/apache/spark/api/java/JavaPairRDD.html

```
org.apache.spark.api.java.JavaPairRDD<K,V>

...
reduceByKey(Function2<V,V,V> func):    JavaPairRDD<K,V>
sortByKey(boolen ascending):           JavaPairRDD<K,V>
collect():                             List<T>
take(int num):                         List<T>
...
```

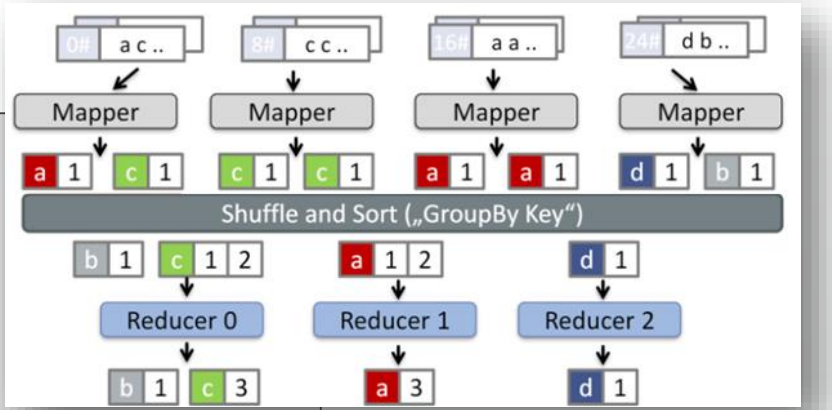






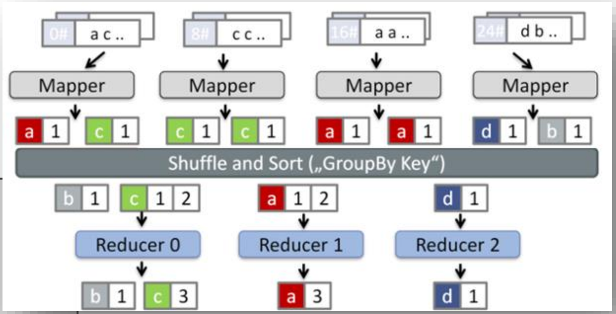


Exercise: Letter Count



```
public class LetterAndWordCount {

  public static void main(String[] args) {
    // 1. Getting a new Spark session
    SparkSession session = SparkSession.builder()
                                       .appName("Spark Letter and Word count")
                                       .getOrCreate();

    // 2. New Spark context -> Entry point for programming
    JavaSparkContext jsc = new JavaSparkContext(session.sparkContext());

    // 3. Read an example file from HDFS -> One entry inside the RDD per row
    JavaRDD<String> inputRDD = jsc.textFile(args[0]);

    // 4. Split each row into the different words -> One entry inside the RDD per word
    JavaRDD<String> wordsRDD = inputRDD.flatMap(new FlatMapFunction<String, String>() {
      public Iterator<String> call(String theRow) throws Exception {
        String[] wordsInRow = theRow.split("\\W+");
        List<String> wordsList = Arrays.asList(wordsInRow);
        return wordsList.iterator();
      }
    });

    // 5. Read the first character of each word and change it to lower-case letter
    JavaRDD<String> firstCharacterAndLowercaseRDD = wordsRDD.map(new Function<String, String>() {
      public String call(String theNormalWord) throws Exception {
        return String.valueOf(theNormalWord.charAt(0)).toLowerCase();
      }
    });
...
```

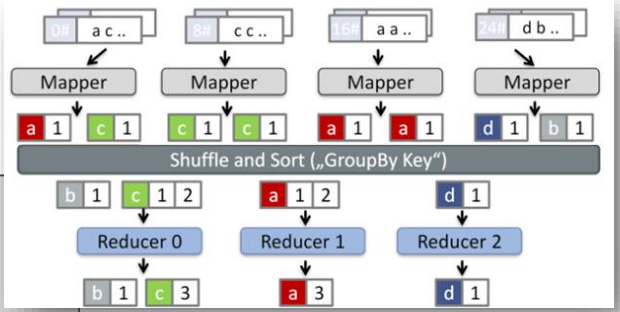
Exercise: Letter Count



```java
  // 6. Map: Prepare for letter counting
  JavaPairRDD<String, Long> mappedWordsRDD = firstCharacterAndLowercaseRDD.mapToPair(new PairFunction<String, String, Long>() {
    public Tuple2<String, Long> call(String firstCharacterLowercase) throws Exception {
      return new Tuple2<String, Long>(firstCharacterLowercase, 1L);
    }
  });

  // 7. Reduce: Summarize the same keys -> Add the occurences inside the text
  JavaPairRDD<String, Long> characterCountRDD = mappedWordsRDD.reduceByKey(new Function2<Long, Long, Long>() {
    public Long call(Long countA, Long countB) throws Exception {
      return countA + countB;
    }
  });

  // 8. Exchange Key and Value -> Preparation for sorting later
  JavaPairRDD<Long, String> turnedRDD = characterCountRDD.mapToPair(new PairFunction<Tuple2<String,Long>, Long, String>() {
    public Tuple2<Long, String> call(Tuple2<String, Long> keyValue) throws Exception {
      String character = keyValue._1;
      Long count = keyValue._2;
      return new Tuple2<Long, String>(count, character);
    }
  });

  // 9. Order by letter frequency
  JavaPairRDD<Long, String> sortedCharacterCountRDD = turnedRDD.sortByKey();

  // 10. Action: Return the content of the RDD -> List of KeyValue (Tuple2) entries
  List<Tuple2<Long, String>> collectedListOnDriver = sortedCharacterCountRDD.collect();

  // 11. Iterate that list and print result text to console
  for(Tuple2<Long, String> oneEntry : collectedListOnDriver) {
    System.out.println(oneEntry._1 + " words start with the letter " + oneEntry._2);
  }
...
```

Exercise: Word Count

```
    // 12. Find the most used words and print the top-10 -> combine transformations
    List<Tuple2<Long, String>> topWords = wordsRDD
                                             .map(oneWord -> oneWord.toLowerCase())
                                             .mapToPair(oneWord -> new Tuple2<String, Long>(oneWord, 1L))
                                             .reduceByKey((countA, countB) -> countA + countB)
                                             .mapToPair(keyValue -> new Tuple2<Long, String>(keyValue._2, keyValue._1))
                                             .sortByKey(false)
                                             .take(10);

    // 13. Iterate that list and print results text to console
    for(Tuple2<Long, String> topWord : topWords) {
      System.out.println("The word \"" + topWord._2 + "\" occures " + topWord._1 + " times.");
    }
  }
}
```

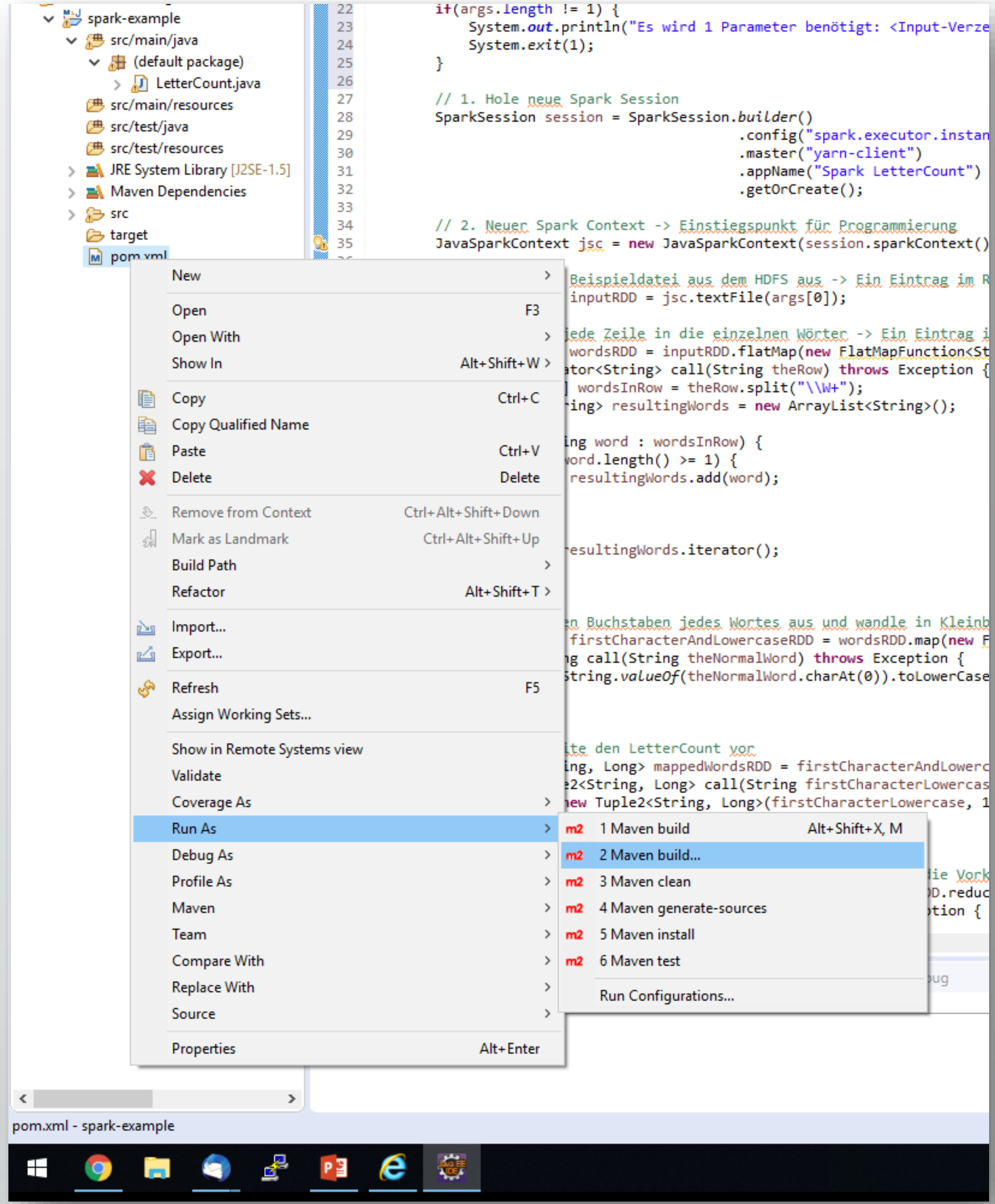


Exercise: Letter and Word Count

- Deployment
  1. Compile the Java project and export it as JAR file
     - E.g. Maven Project → Goal „clean package"
  2. Upload the JAR file to a node of the cluster
     - FTP client, e.g. WinSCP, FileZilla
     - E.g. /home/studentX/letterAndWordCountSpark.jar
  3. Start the application using the spark-submit command

     -
       ```
       spark-submit
       --master yarn
       --deploy-mode client
       --class main.LetterAndWordCount
       /home/studentX/letterAndWordCountSpark.jar
       data/movielens/movies/movies.csv
       ```
     - Alternative: Use Apache Livy (REST interface for Spark)
  4. Spark History UI: http://<server-ip>:18081/

https://github.com/seamless-analytics/summer-school-2019/
Day 1 - Hadoop/Task 3 - Apache Spark



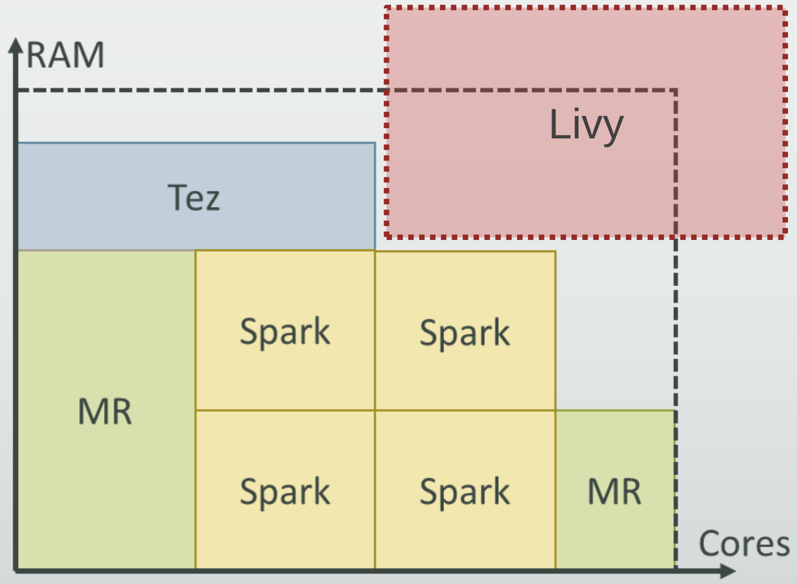
```
if(args.length != 1) {
    System.out.println("Es wird 1 Parameter benötigt: <Input-Verze
    System.exit(1);
}

// 1. Hole neue Spark Session
SparkSession session = SparkSession.builder()
                                   .config("spark.executor.instan
                                   .master("yarn-client")
                                   .appName("Spark LetterCount")
                                   .getOrCreate();

// 2. Neuer Spark Context -> Einstiegspunkt für Programmierung
JavaSparkContext jsc = new JavaSparkContext(session.sparkContext()

Beispieldatei aus dem HDFS aus -> Ein Eintrag im R
inputRDD = jsc.textFile(args[0]);

jede Zeile in die einzelnen Wörter -> Ein Eintrag i
wordsRDD = inputRDD.flatMap(new FlatMapFunction<St
tor<String> call(String theRow) throws Exception {
] wordsInRow = theRow.split("\\W+");
ing> resultingWords = new ArrayList<String>();

ing word : wordsInRow) {
ord.length() >= 1) {
resultingWords.add(word);

esultingWords.iterator();

n Buchstaben jedes Wortes aus und wandle in Kleinb
firstCharacterAndLowercaseRDD = wordsRDD.map(new F
g call(String theNormalWord) throws Exception {
tring.valueOf(theNormalWord.charAt(0)).toLowerCase

ite den LetterCount vor
ing, Long> mappedWordsRDD = firstCharacterAndLowerc
2<String, Long> call(String firstCharacterLowercas
ew Tuple2<String, Long>(firstCharacterLowercase, 1

ie Vork
D.reduc
tion {
```

Apache YARN

Ressource Manager of the Hadoop Cluster

Apache YARN – **Y**et **A**nother **R**esource **N**egotiator

- Responsible for managing available resources in the cluster
  - YARN allocates a portion of the resources of each node (default 80%) for cluster computations
- Assigns hardware resources (CPU, RAM, GPU) to programs in the form of containers
  - Container = fixed hardware assignments that are made available to a (sub-) program
- Spark processes run e.g. in such containers
  - Separate container for each Driver and Executor process
- Scheduling of incoming requests
  - First Come First Serve
  - Default: Wait at cluster utilization
- YARN Resource Manager UI: http:<server-host>:8088/cluster

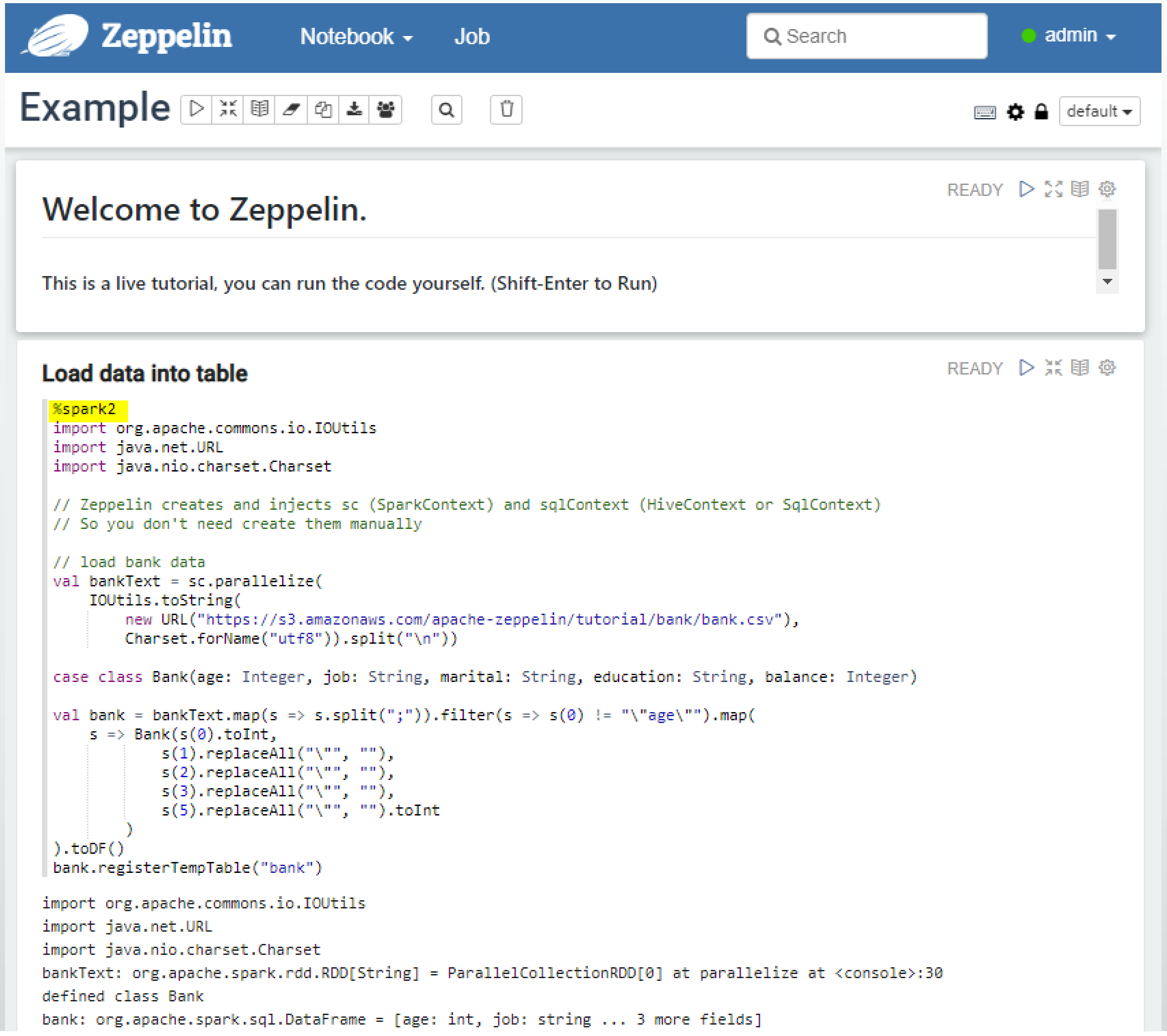


Apache Zeppelin

Web-Based Notebook for Hadoop

Apache Zeppelin

- Web-based notebook
- Interactive data analysis with Spark (Core) and SQL
  - Scala, Python, R
- Interpreters as interfaces to different components
  - Hive (via JDBC): **%jdbc(hive)**
  - Spark: **%spark**
  - Markdown: **%md**
  - (optional): Local shell commands: **%sh**
- Well suited for prototype construction
  - Segregation into so-called paragraphs
  - Access to Spark session, Spark context and SQL context
- Zeppelin Web UI: http://<server-url>:9995



Welcome to Zeppelin.

This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)

Load data into table

```
%spark2
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually

// load bank data
val bankText = sc.parallelize(
    IOUtils.toString(
        new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv"),
        Charset.forName("utf8")).split("\n"))

case class Bank(age: Integer, job: String, marital: String, education: String, balance: Integer)

val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
    s => Bank(s(0).toInt,
            s(1).replaceAll("\"", ""),
            s(2).replaceAll("\"", ""),
            s(3).replaceAll("\"", ""),
            s(5).replaceAll("\"", "").toInt
        )
).toDF()
bank.registerTempTable("bank")
```

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:30
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string ... 3 more fields]
```

Exercise: MovieLens Data Preparation

- For ML analysis later, we need to combine the movies, tags and ratings data to following structure:

```
+--------+-----------------------+----------------------------------+---------------------------------+--------+
|movieId |title                  |genres                            |tags                             |rating  |
+--------+-----------------------+----------------------------------+---------------------------------+--------+
|1       |Toy Story (1995)       |Adventure|Animation|Children|C...  |friendship|toys come to life ... |3.88664 |
|2       |Jumanji (1995)         |Adventure|Children|Fantasy        |adapted from book|filmed in a... |3.24658 |
|3       |Grumpier Old Men (1995)|Comedy|Romance                    |old|comedinha de velhinhos es... |3.17398 |
+--------+-----------------------+----------------------------------+---------------------------------+--------+
```

- Use Spark SQL with Dataset API to access the MovieLens files in HDFS
  - **%spark** Interpreter in Zeppelin notebook → access to Spark's SQLContext via **sqlContext** variable
- Transform them in SQL-like way (groupBy, join, etc.)
- Save the final dataset back to HDFS

https://github.com/seamless-analytics/summer-school-2019/Day 1 - Hadoop/Task 4 - Apache Zeppelin

Spark API JavaDoc (Spark SQL)

https://spark.apache.org/docs/2.3.2/api/java/org/apache/spark/sql/SQLContext.html

```
org.apache.spark.sql.SQLContext

...
sql(String sqlText):                                        Dataset<Row>
read():                                                     DataFrameReader
```

https://spark.apache.org/docs/2.3.2/api/java/org/apache/spark/sql/Dataset.html

```
org.apache.spark.sql.Dataset<T>

...
groupBy(Column... cols):                                    RelationalGroupedDataset
drop(String colName):                                       Dataset<Row>
agg(Column expr, Col... exprs):                             Dataset<Row>
withColumn(String colName, Column col):                     Dataset<Row>
withColumnRenamed(String eName, String nName):              Dataset<Row>
join(Dataset<?> right, Column joinExpr, String joinType):   Dataset<Row>
show(int numRows, boolean truncate):                        void
write():                                                    DataFrameWriter
```

https://spark.apache.org/docs/2.3.2/api/java/org/apache/spark/sql/functions.html

```
org.apache.spark.sql.functions

...
collect_set(Column e):                                      Column
lower(Column e):                                            Column
concat_ws(String separator, Column... exprs):               Column
avg(Column e):                                              Column
```

OK – One Last Exercise: Web Log Analysis

- Using Zeppelin to analyze a web log file: Find the top-10 IP addresses, that generated most entries
  - Structure of one entry: `10.223.157.186 - - [15/Jul/2009:14:58:59 -0700] "GET / HTTP/1.1" 403 202`
  - Get the IP by using `row.split(" - ")[0]`

- Use Spark Core with RDD API to read the log files in HDFS
  - **%spark** Interpreter in Zeppelin notebook → access to SparkContext via **sc** variable
- Transform them in MapReduce like way (map(ToPair), reduce(ByKey), sort(ByKey))
- Save the final dataset back to HDFS

https://github.com/seamless-analytics/summer-school-2019/Day 1 - Hadoop/Task 5 - Apache Zeppelin

That‘s it for Today ;-)

Public GitHub repository
https://github.com/seamless-analytics/summer-school-2019

Our website
https://seamless-analytics.de

Competence Network

Daniel Müller
daniel.mueller@seamless-analytics.de

Marvin Follmann
marvin.follmann@seamless-analytics.de

Image Sources

Slide	URL
3	https://gi.de/fileadmin/GI/Hauptseite/Service/Lexikon/bigdata-abb1.png
10	https://i2.wp.com/www.vamsitalkstech.com/wp-content/uploads/2015/06/AmbariArch.png
13	https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/images/hdfsdatanodes.png
	https://www.researchgate.net/profile/Mohamed_Khafagy4/publication/299588043/figure/fig1/AS:346796439162886@1459694122981/WRITING-A-FILE-ON-HDFS-USING-PIPELINE-REPLICATION-APPROACH.png
17	https://www.oreilly.com/library/view/data-algorithms/9781491906170/assets/daal_0001.png
18ff.	https://content.edupristine.com/images/blogs/HIDDEN_PATTERN%27s_image1.jpg
23	https://mapr.com/products/apache-hive/assets/hive-logo.png
28	https://spark.apache.org/
29	https://spark.apache.org/docs/latest/img/cluster-overview.png