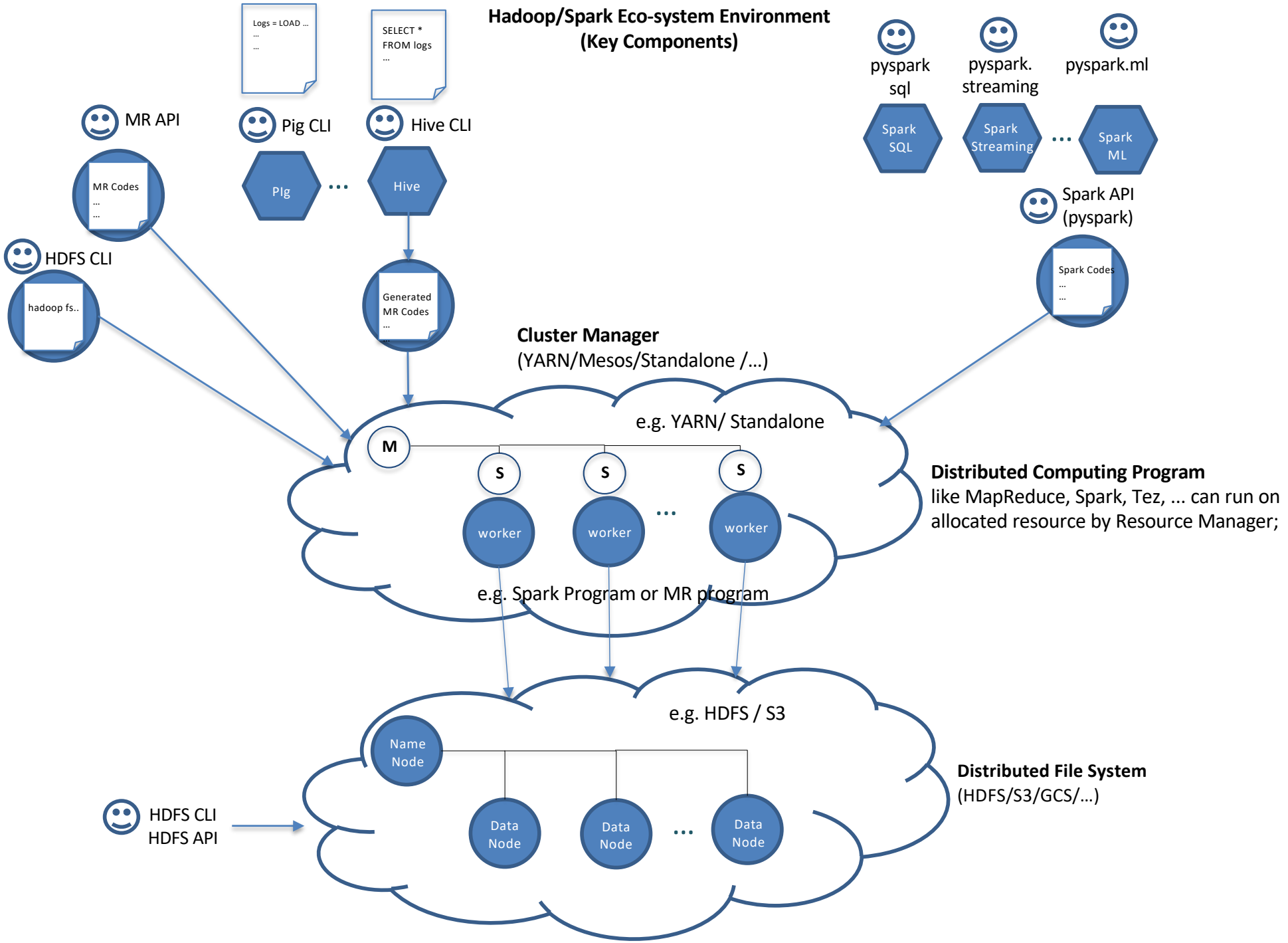
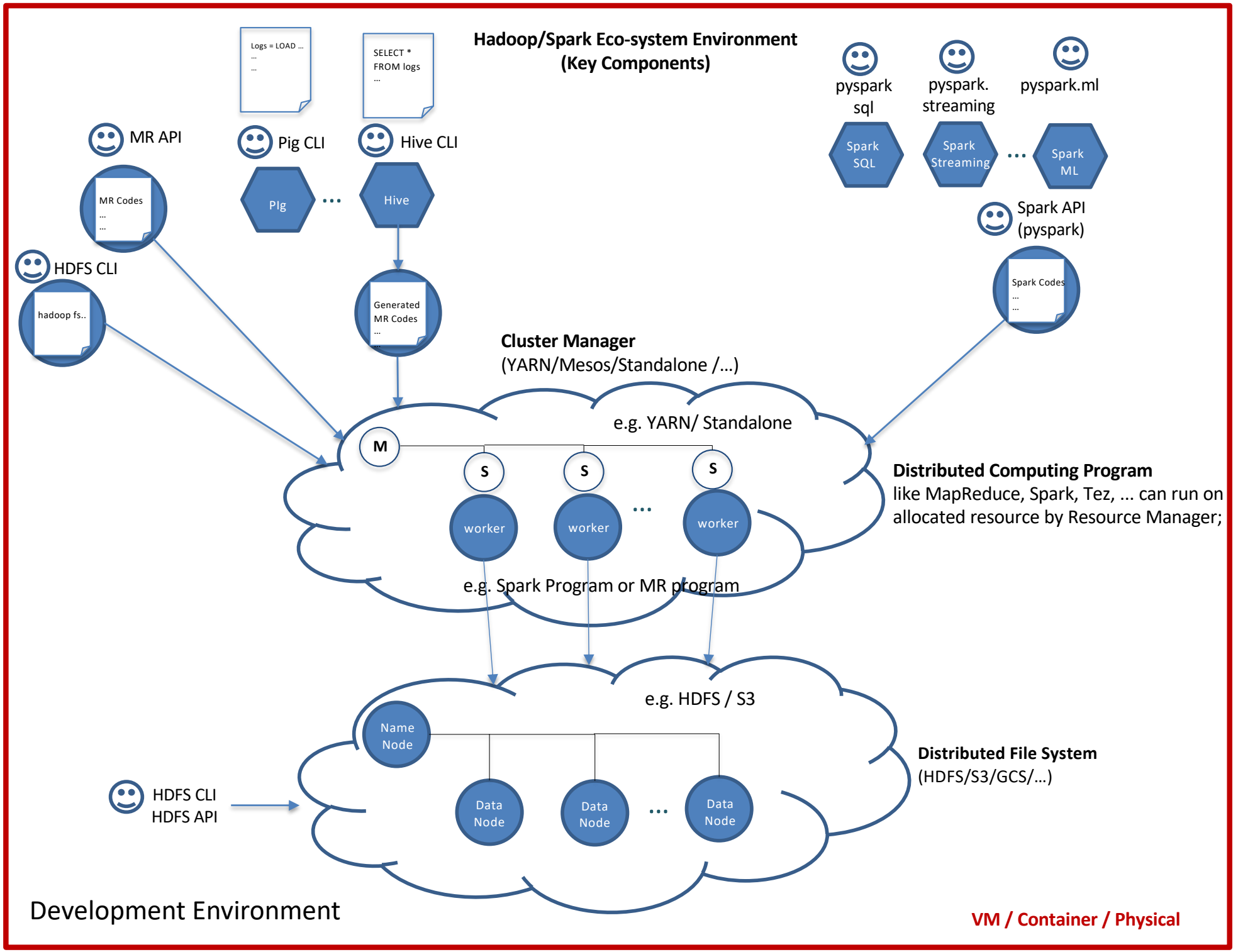
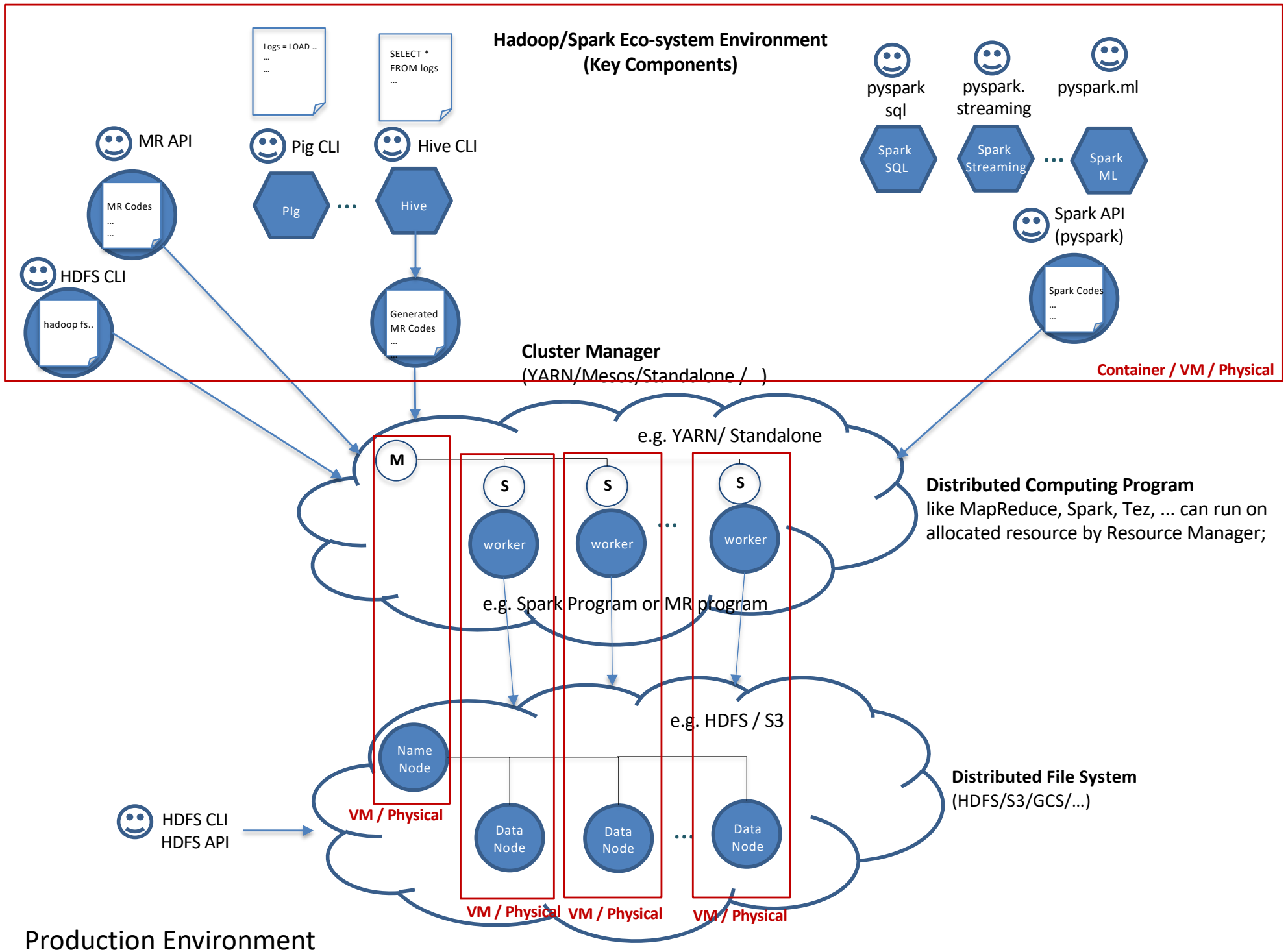
Hadoop/Spark Eco-system Overview


Hadoop/Spark Eco-system Environment
(Key Components)
Logs = LOAD ...
...
...
SELECT *
FROM logs
...
pyspark
sql
pyspark.
streaming
pyspark.ml
MR API
Pig CLI
Hive CLI
Spark
SQL
Spark
Streaming
Spark
ML
MR Codes
...
...
Pig
Hive
Spark API
(pyspark)
HDFS CLI
hadoop fs..
Generated
MR Codes
...
Spark Codes
...
...
Cluster Manager
(YARN/Mesos/Standalone /...)
e.g. YARN/ Standalone
M
S
S
S
worker
worker
worker
Distributed Computing Program
like MapReduce, Spark, Tez, ... can run on
allocated resource by Resource Manager;
e.g. Spark Program or MR program
e.g. HDFS / S3
Name
Node
Data
Node
Data
Node
Data
Node
Distributed File System
(HDFS/S3/GCS/...)
HDFS CLI
HDFS API

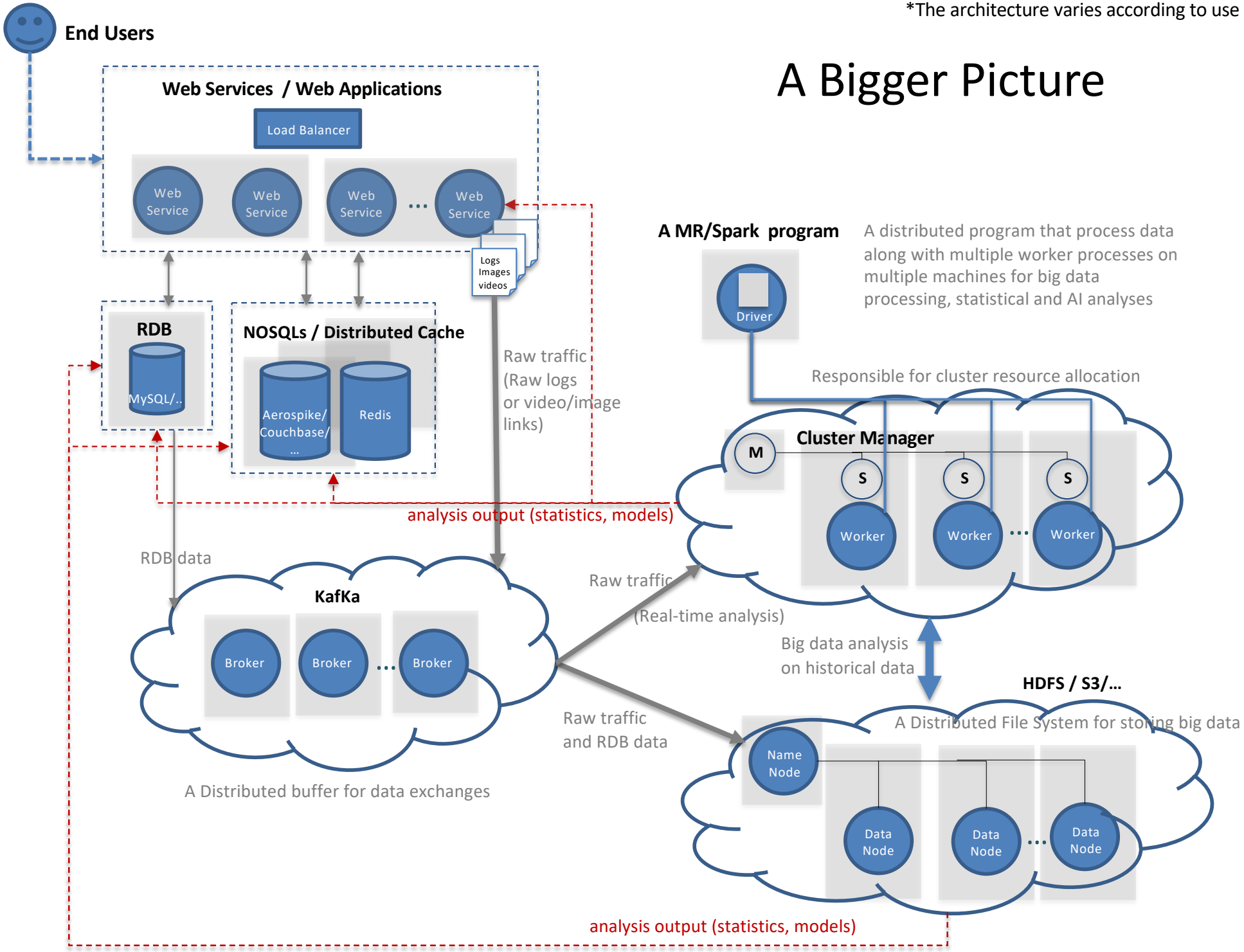

Hadoop/Spark Eco-system Environment
(Key Components)
Logs = LOAD ...
...
...
SELECT *
FROM logs
...
pyspark
sql
pyspark.
streaming
pyspark.ml
MR API
Pig CLI
Hive CLI
Spark
SQL
Spark
Streaming
Spark
ML
MR Codes
...
...
Pig
Hive
Spark API
(pyspark)
HDFS CLI
hadoop fs..
Generated
MR Codes
...
Spark Codes
...
...
Cluster Manager
(YARN/Mesos/Standalone /...)
e.g. YARN/ Standalone
M
S
S
S
worker
worker
worker
Distributed Computing Program
like MapReduce, Spark, Tez, ... can run on
allocated resource by Resource Manager;
e.g. Spark Program or MR program
e.g. HDFS / S3
Name
Node
Data
Node
Data
Node
Data
Node
Distributed File System
(HDFS/S3/GCS/...)
HDFS CLI
HDFS API
Development Environment
VM / Container / Physical



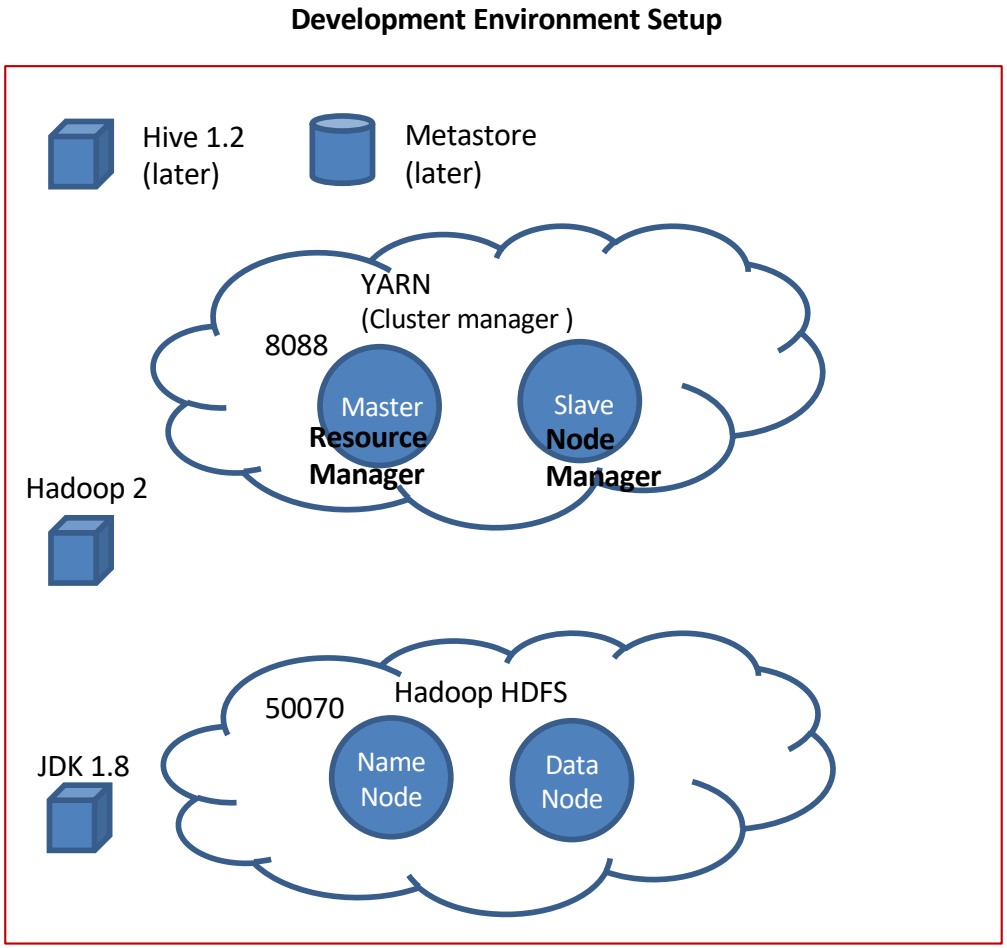
Production Environment

*The architecture varies according to use cases

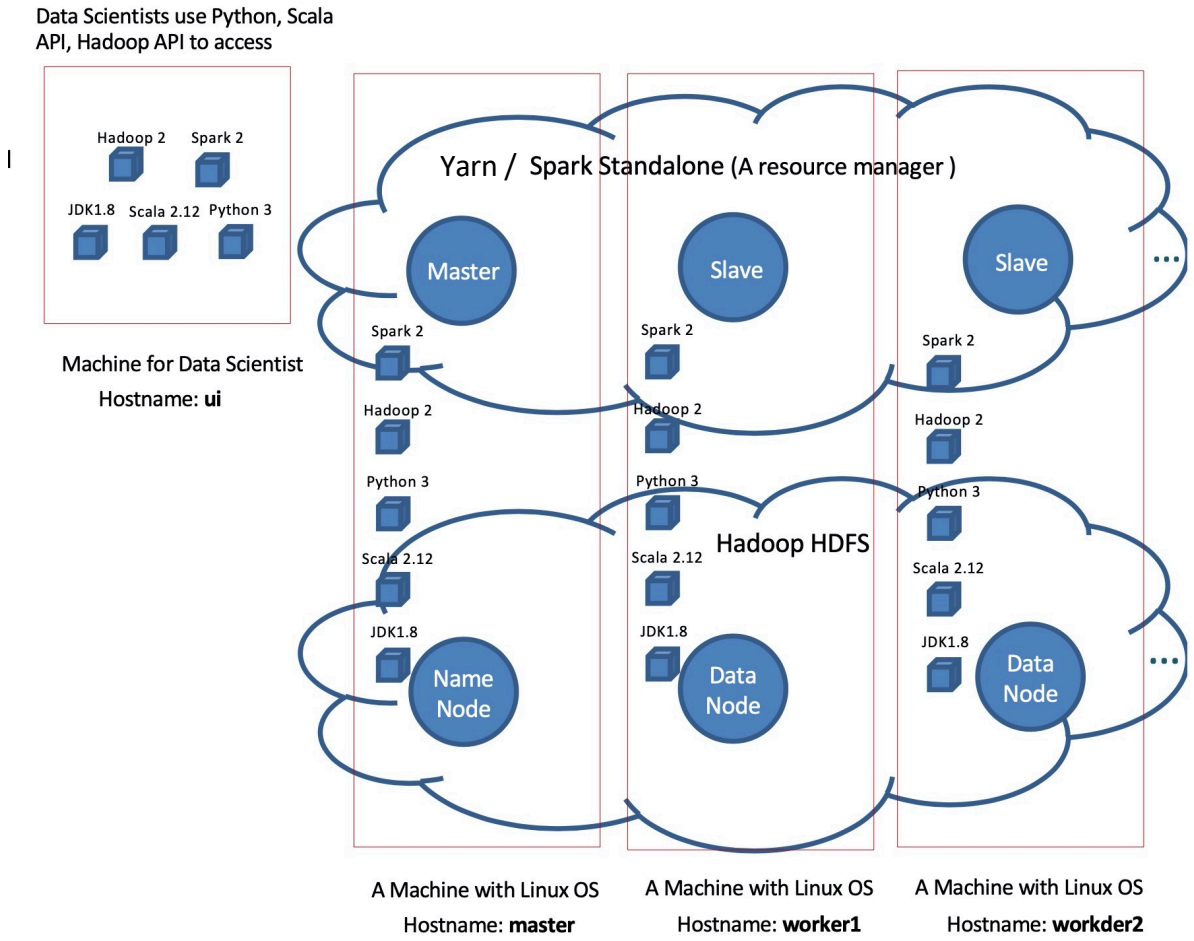
A Bigger Picture



Demo
Development Environment Setup



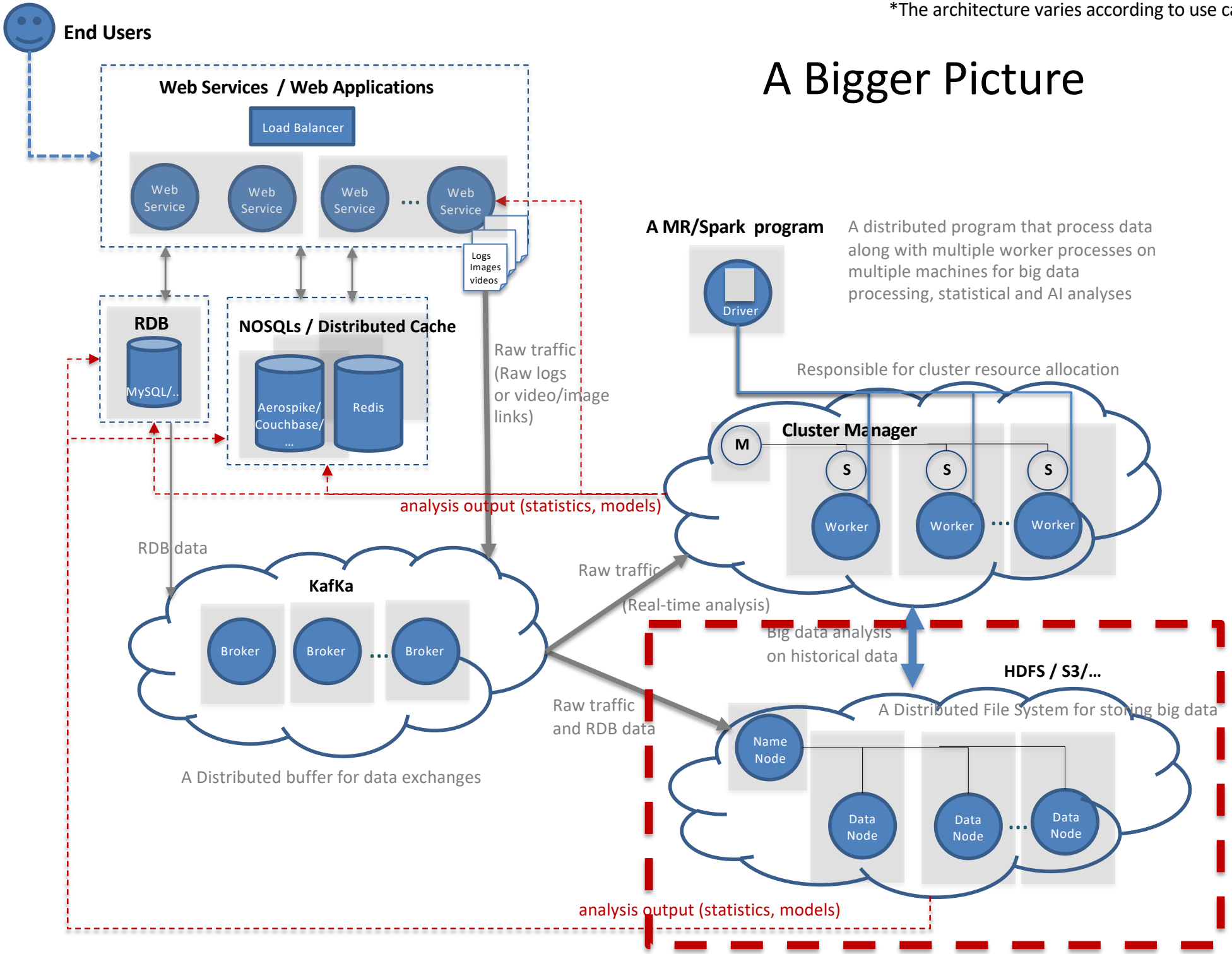
Demo (later on)
Production Environment Setup



Introduction to HDFS

*The architecture varies according to use cases

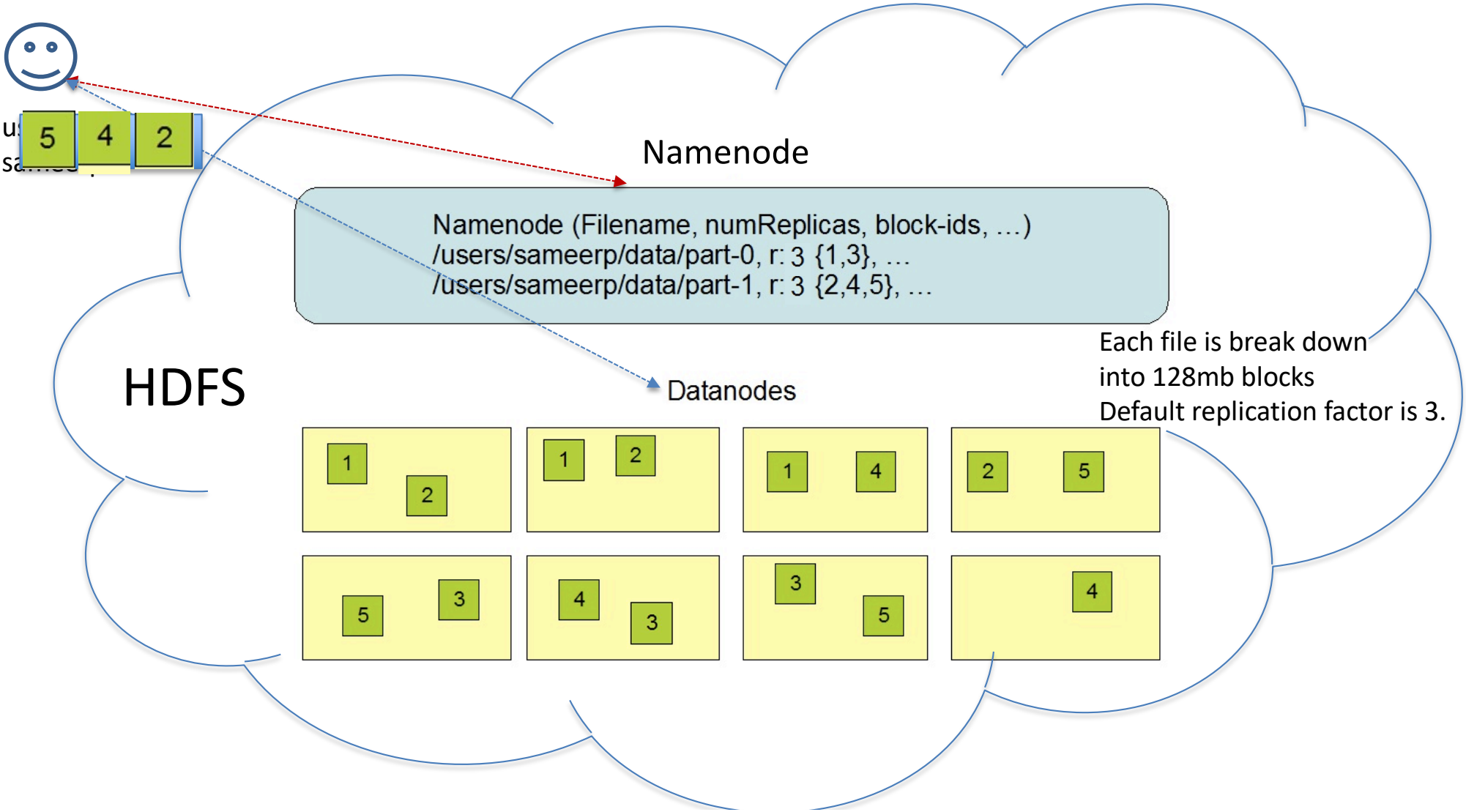
A Bigger Picture



HDFS/S3/GCS Architecture

```
hadoop fs -mkdir /user/sameerp/data
hadoop fs –put part-0 /user/sameerp/data
hadoop fs –put part-1 /user/sameerp/data
```

HDFS CLI

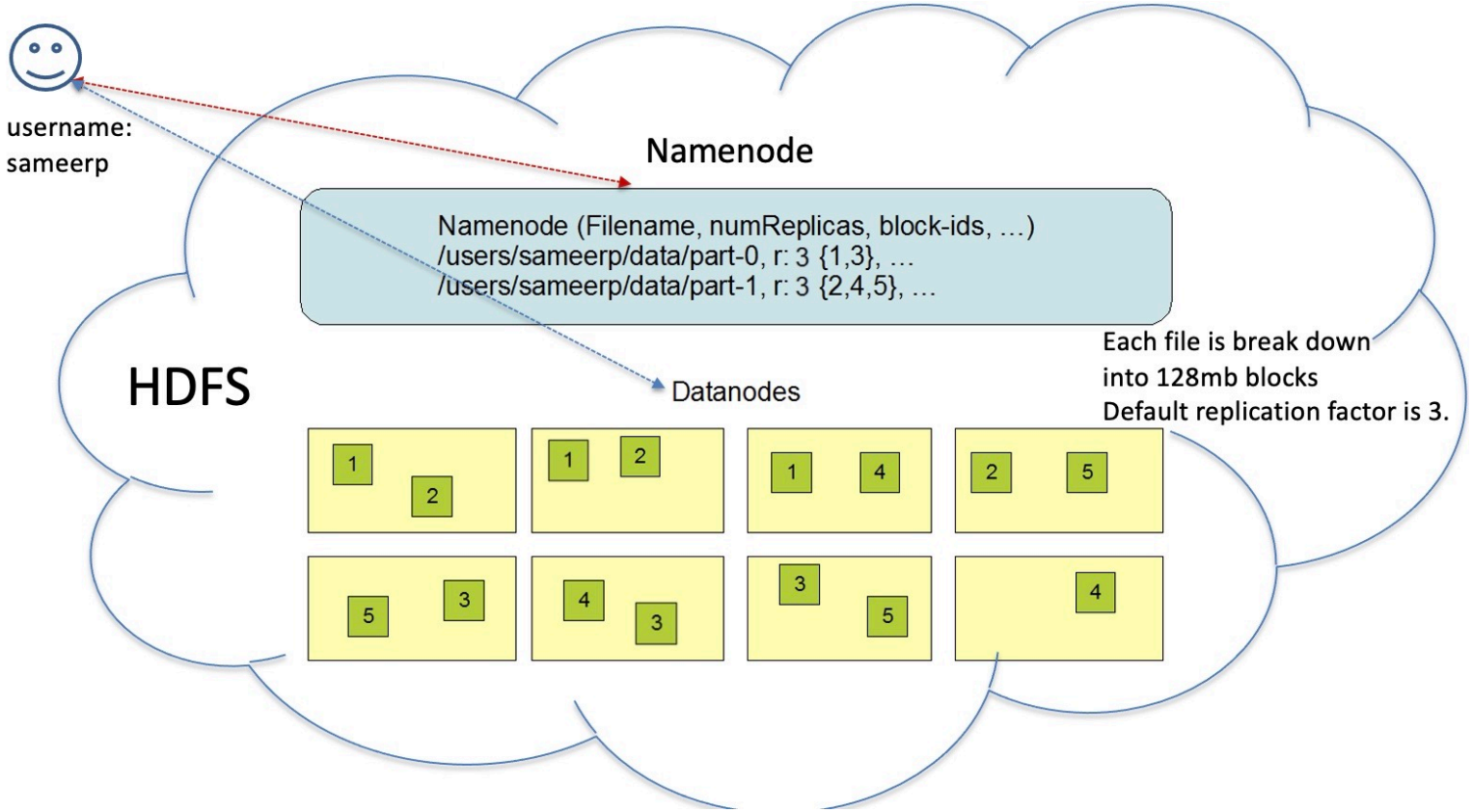


HDFS Architecture

- Present **a single file system** for the end user though it consists of multiple machines under the hood (s3, gcs,..)
- Used to **store small to very large files**
- Files are broken up into blocks stored on Data Nodes
  - Typically, **128 MB** each block
  - Each block has **3 copies** by default
  - **Configurable by clients**
- **Commodity hardware**
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them

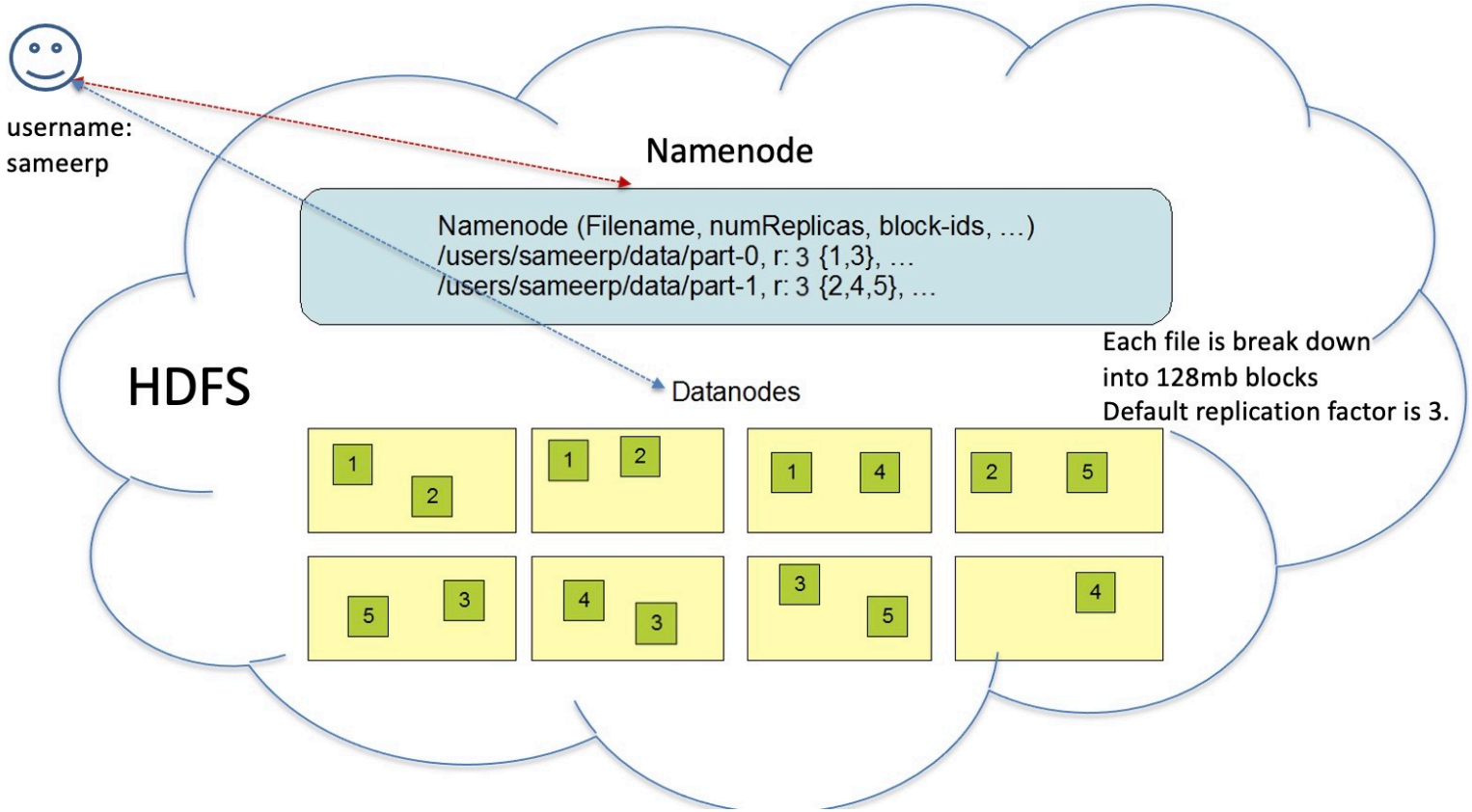
NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides



Name Node Metadata

- Meta-data in Memory
  - The entire metadata is in main memory

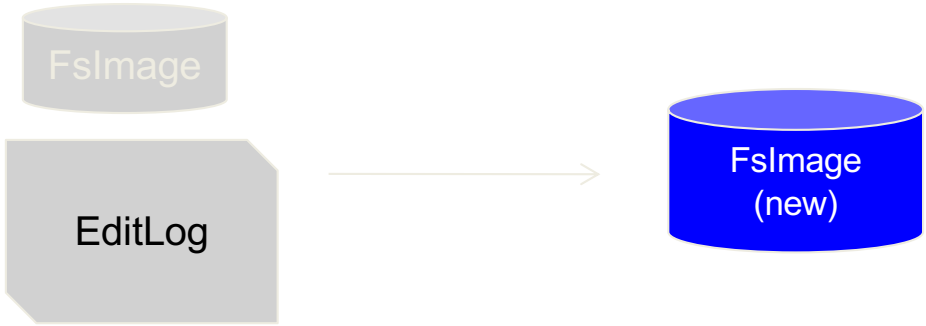


NameNode Metadata (cont.)

- A Transaction Log (called *EditLog*)
  - Records file creations, file deletions. Etc
- *FsImage*
  - The entire namespace, mapping of blocks to files and file system properties are stored in a file called *FsImage*.
  - NameNode can be configured to maintain multiple copies of *FsImage* and *EditLog*.

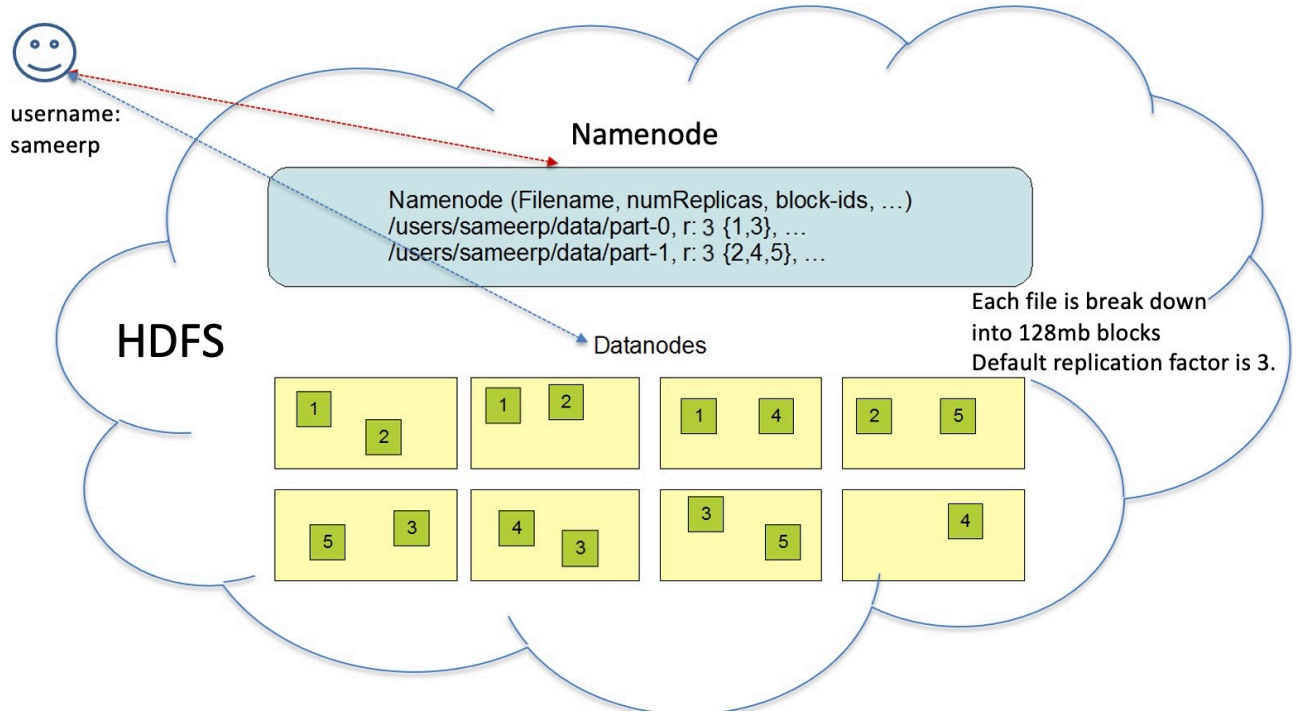
Secondary NameNode

- Copies *FsImage* and *EditLog* from NameNode to a temporary directory
- Merges FSImage and *EditLog* into a new *FSImage* in temporary directory.
- Uploads new *FSImage* to the NameNode
  - Transaction Log on NameNode is purged



DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Serves data and meta-data to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode



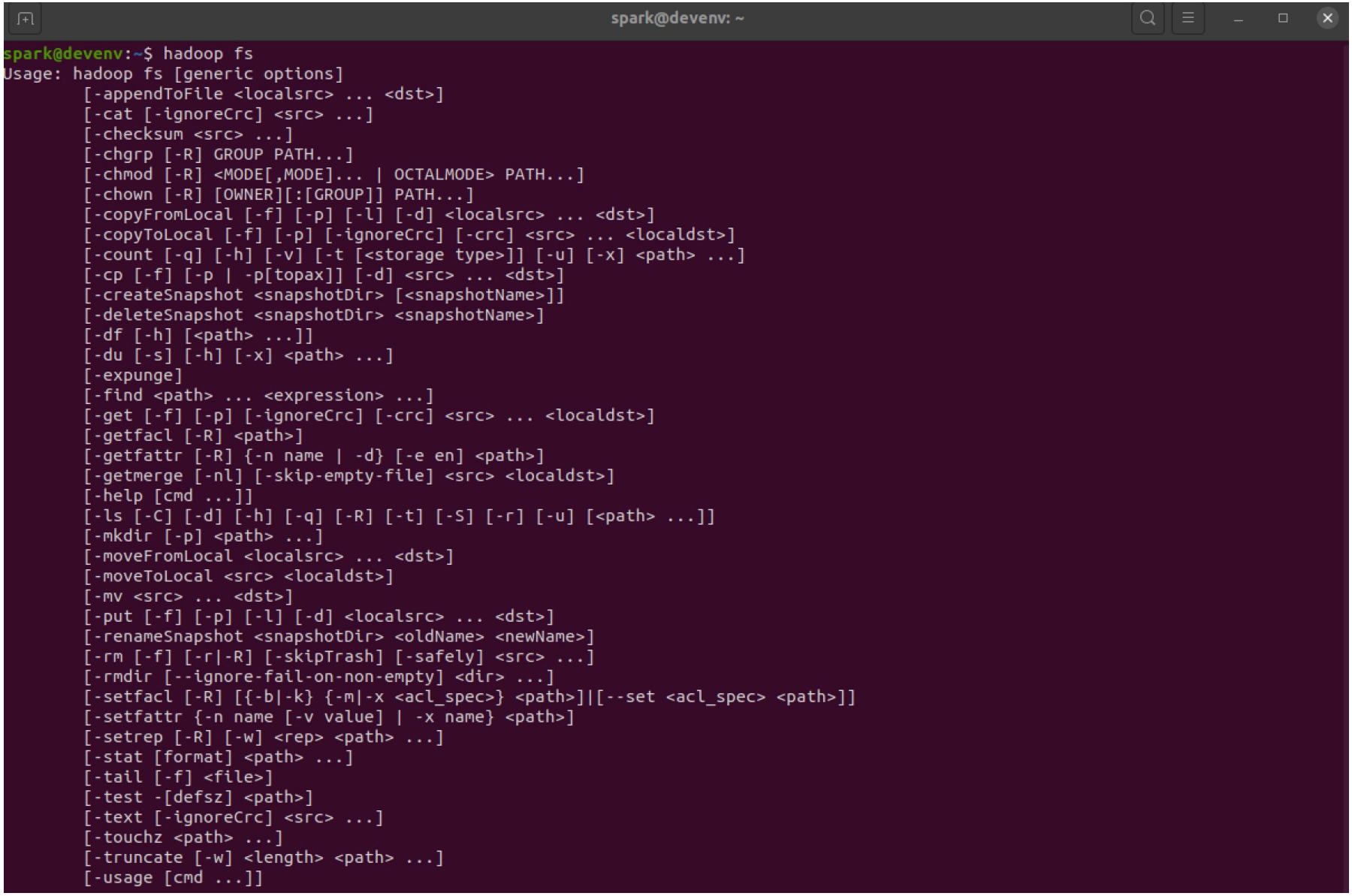
HDFS User Interface

- The command line interface
  - hadoop fs -mkdir /foodir
  - hadoop fs -cat /foodir/myfile.txt
  - hadoop fs -rm /foodir myfile.txt

  hdfs dfs –mkdir /foodir

- APIs in JAVA, Python, etc.

HDFS User Interface



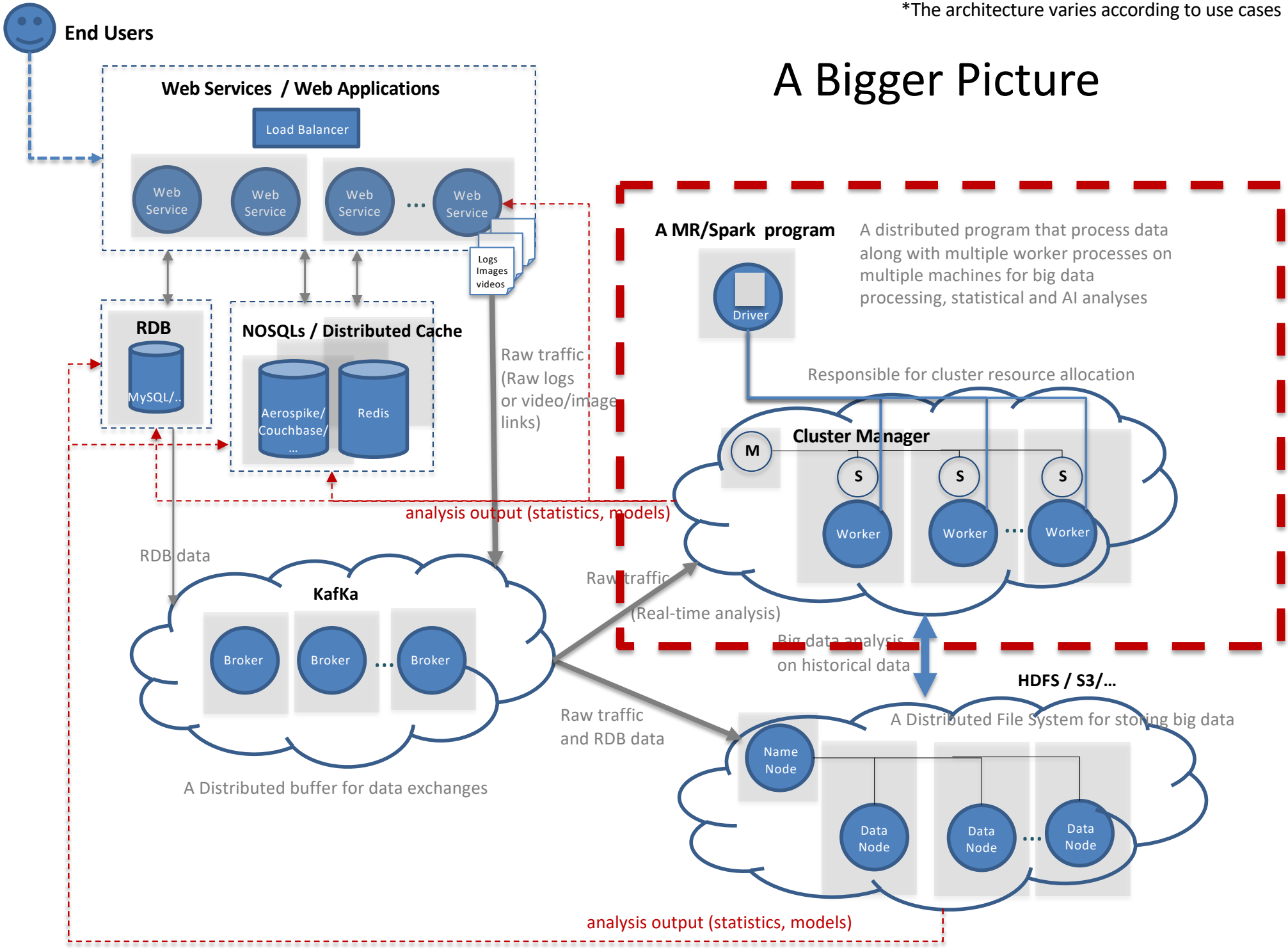
Demo

- Use HDFS CLI to handle files and folders

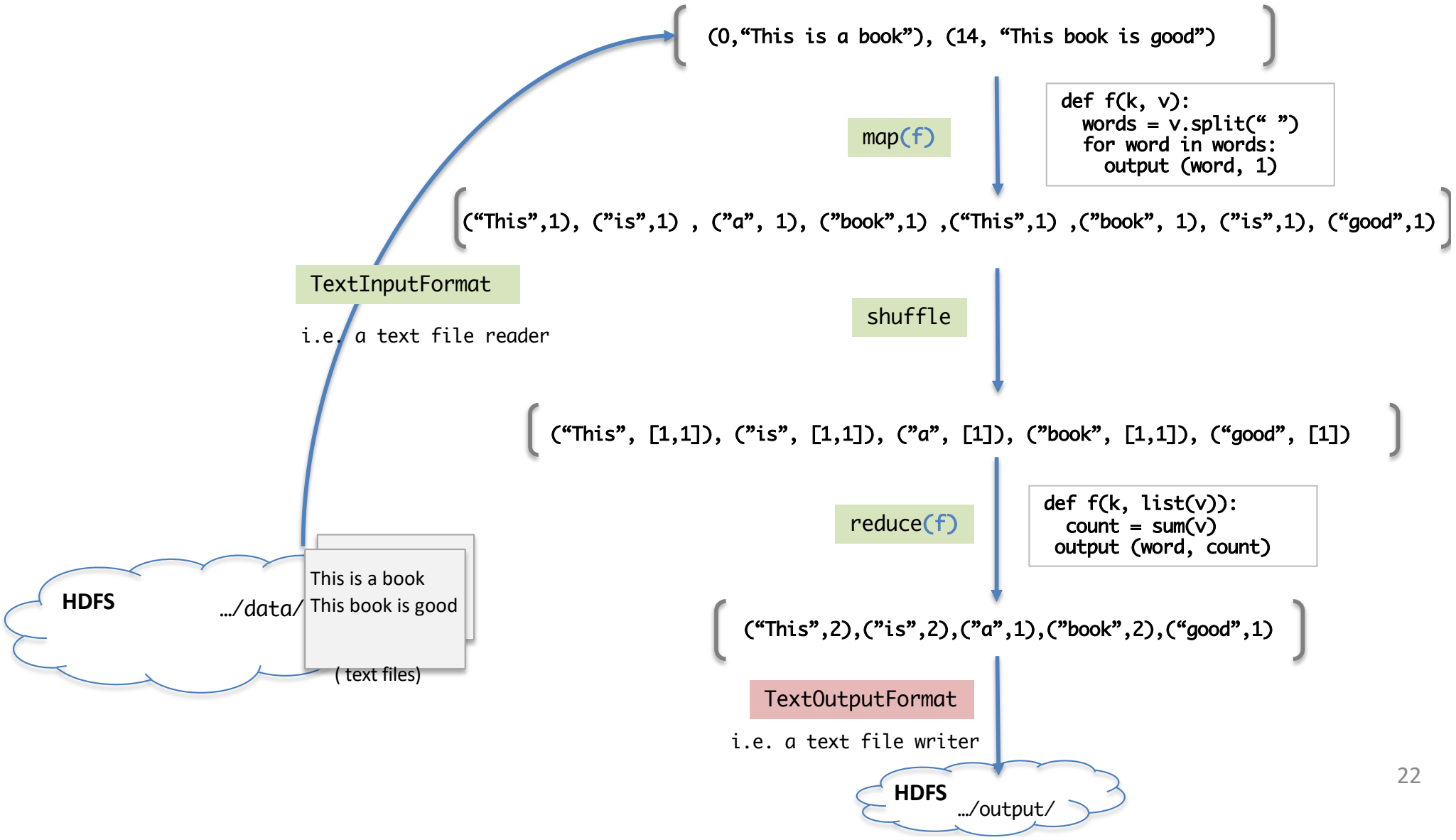
Introduction to MapReduce

*The architecture varies according to use cases

A Bigger Picture



MapReduce Programming Model (Concept)



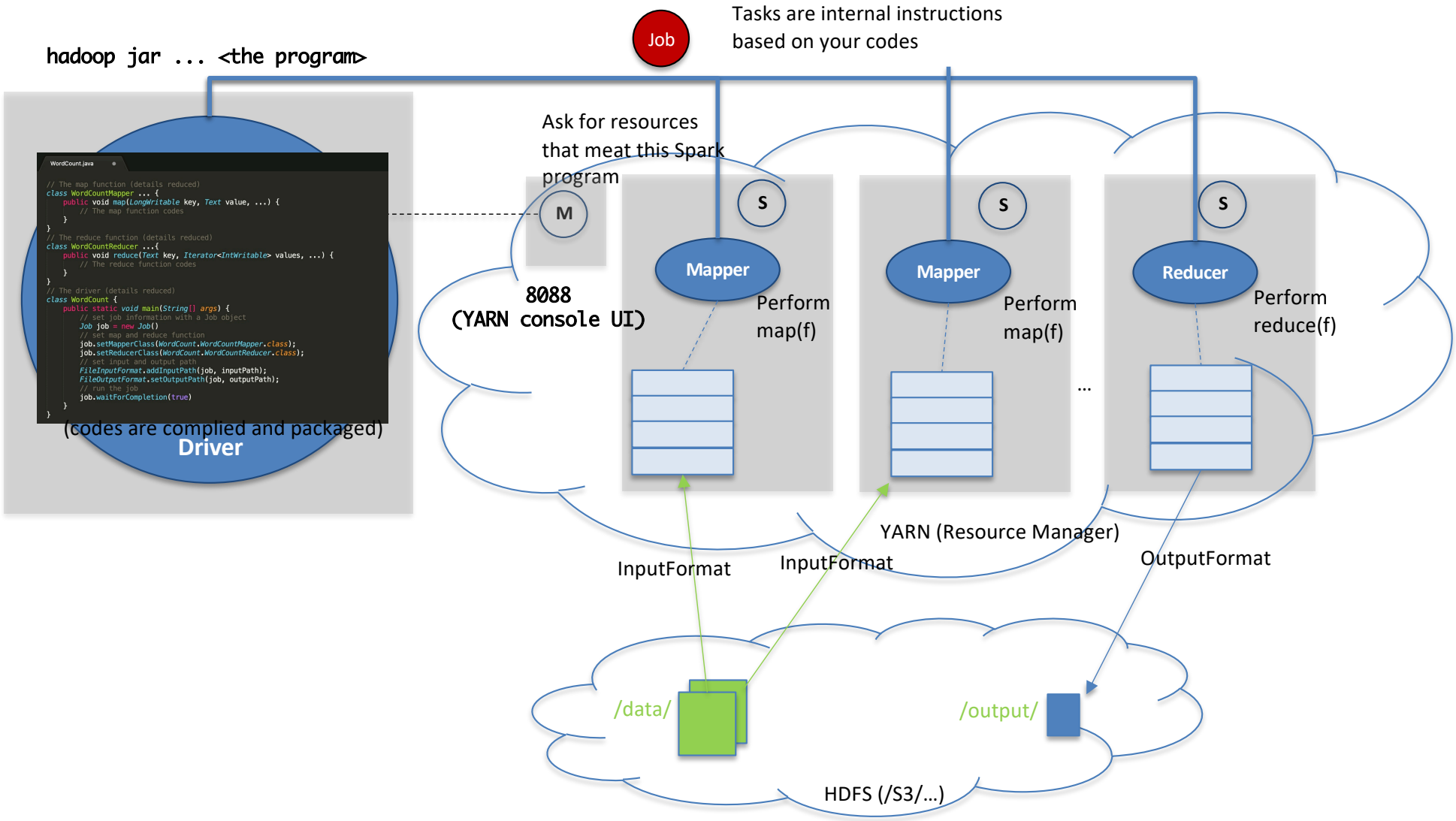
How a MR program is like ?

WordCount.java

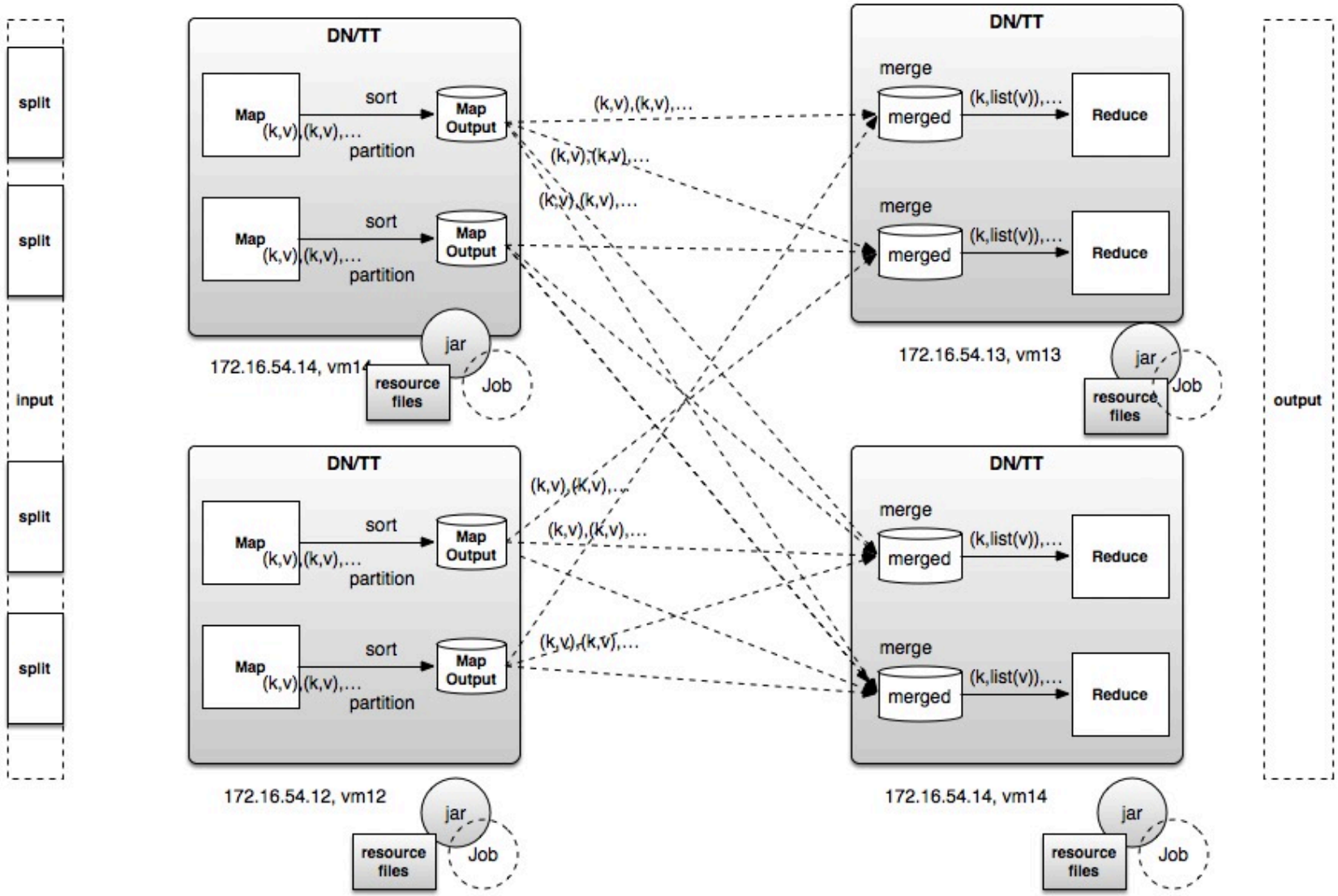
```
// The map function (details reduced)
class WordCountMapper ... {
    public void map(LongWritable key, Text value, ...) {
        // The map function codes
    }
}
// The reduce function (details reduced)
class WordCountReducer ...{
    public void reduce(Text key, Iterator<IntWritable> values, ...) {
        // The reduce function codes
    }
}
// The driver (details reduced)
class WordCount {
    public static void main(String[] args) {
        // set job information with a Job object
        Job job = new Job()
        // set map and reduce function
        job.setMapperClass(WordCount.WordCountMapper.class);
        job.setReducerClass(WordCount.WordCountReducer.class);
        // set input and output path
        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputPath);
        // run the job
        job.waitForCompletion(true)
    }
}
```

MR Program Run-time Excution Flow

hadoop jar mr101.jar com.iii.mr101.WordCount hdfs://devenv/user/spark/mr101/wordcount/data hdfs://devenv/user/spark/mr101/wordcount/output



MR Program Run-time Excution Flow (another view)



Demo

- Install Java IDE
- Calculate word count example in MR
- Calculate average temperature in MR

(Just to have an idea how the MR program are like and run. Don’t worry about JAVA ☺)