

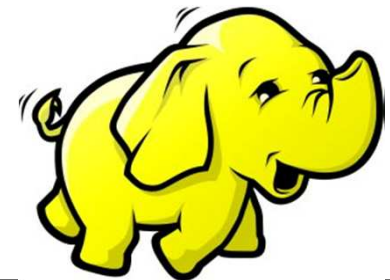
# **Processamento na nuvem**

Edward Ribeiro

# Hadoop

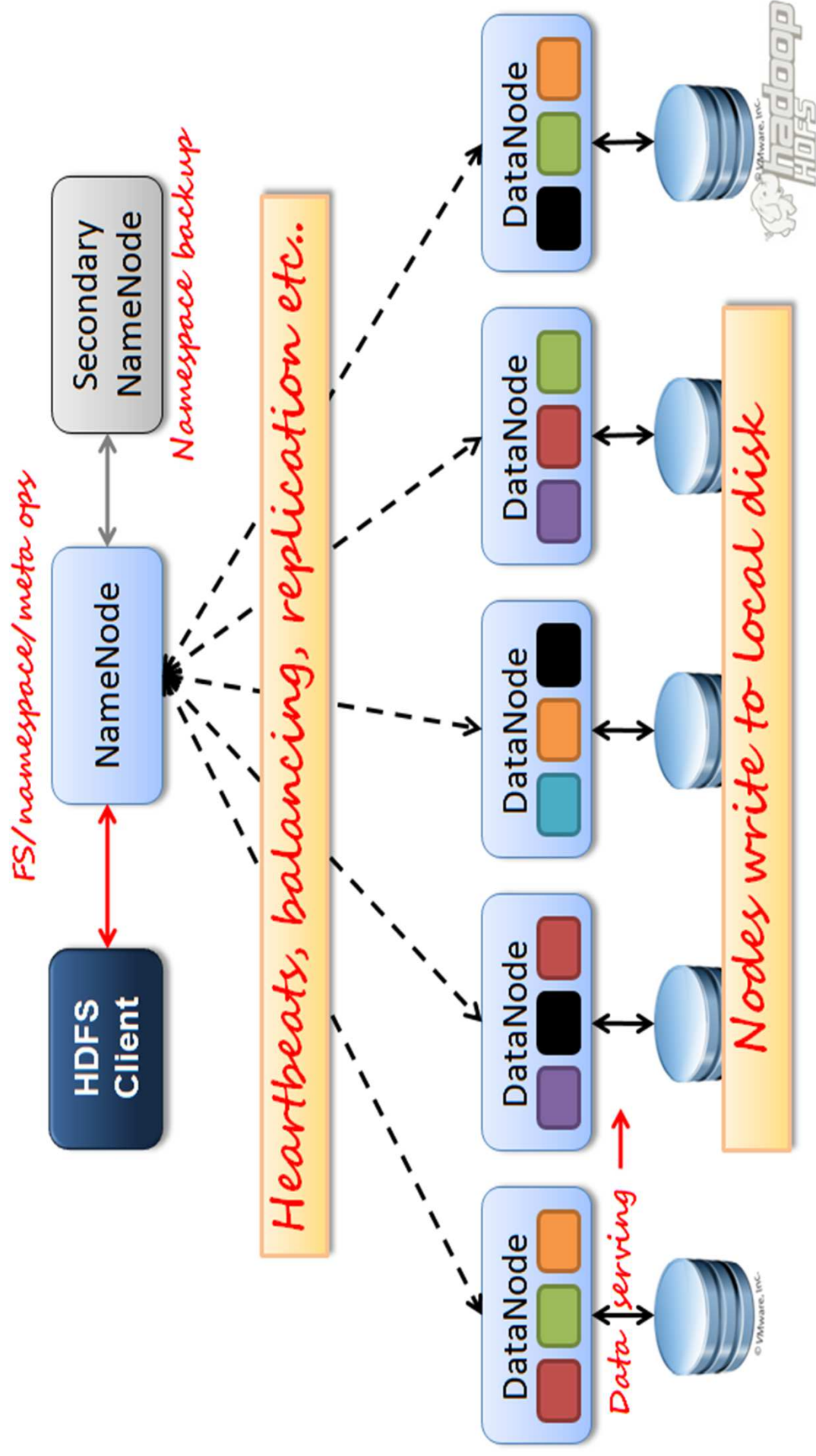
*Framework de código fonte aberto, escrito em Java, para aplicações distribuídas que usam dados intensivamente*

- Map-Reduce
- HDFS
- HBase
- Zookeeper
- Avro
- etc



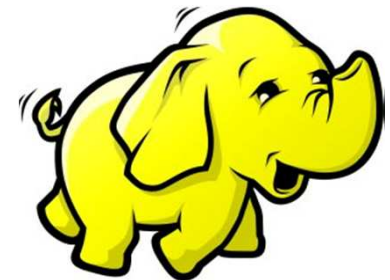
# HDFS

Sistema de arquivos distribuído, rodando no espaço do usuário, otimizado para grandes quantidades de dados.



# Map-Reduce

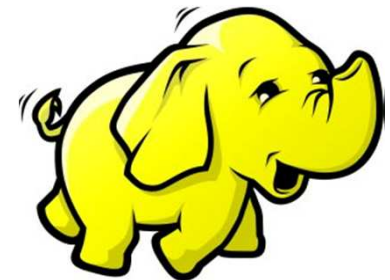
**Modelo de computação distribuída e framework distribuído** para processamento de grandes quantidades de dados de forma paralela em um cluster computacional.



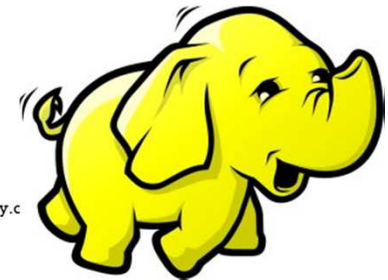
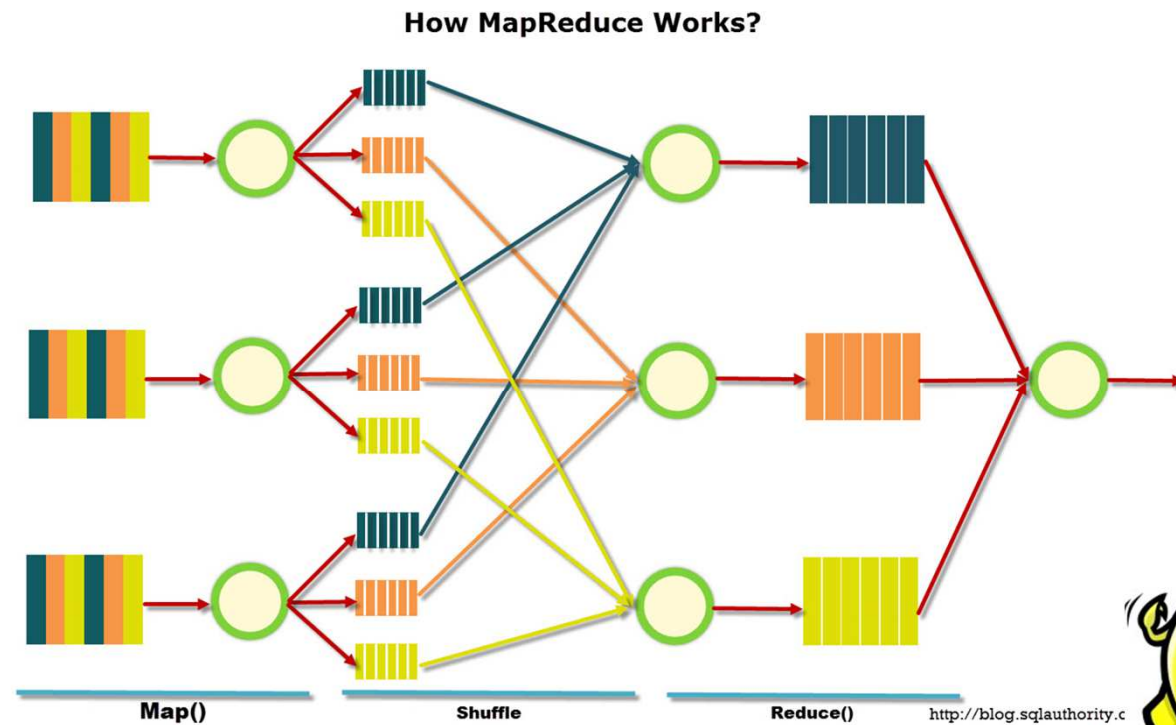
# Map-Reduce

Map → Função que mapeia os elementos de uma lista de elementos

Reduce → Operação de resumo em uma lista de elementos



# Map-Reduce



# Exemplo: Word Count

```
map(String input_key, String input_value):  
  // input_key: document name  
  // input_value: document contents  
  for each word w in input_value:  
    EmitIntermediate(w, "1");
```

```
<"Sam", "1">, <"Apple", "1">, <"Sam", "1">,  
<"Mom", "1">, <"Sam", "1">, <"Mom", "1">,
```

```
reduce(String output_key, Iterator  
intermediate_values):  
  // output_key: a word  
  // output_values: a list of counts  
  int result = 0;  
  for each v in intermediate_values:  
    result += ParseInt(v);  
  Emit(AsString(result));
```

```
<"Sam", ["1", "1", "1"]>, <"Apple", ["1"]>,  
<"Mom", ["1", "1"]>
```

```
"3"  
"1"  
"2"
```



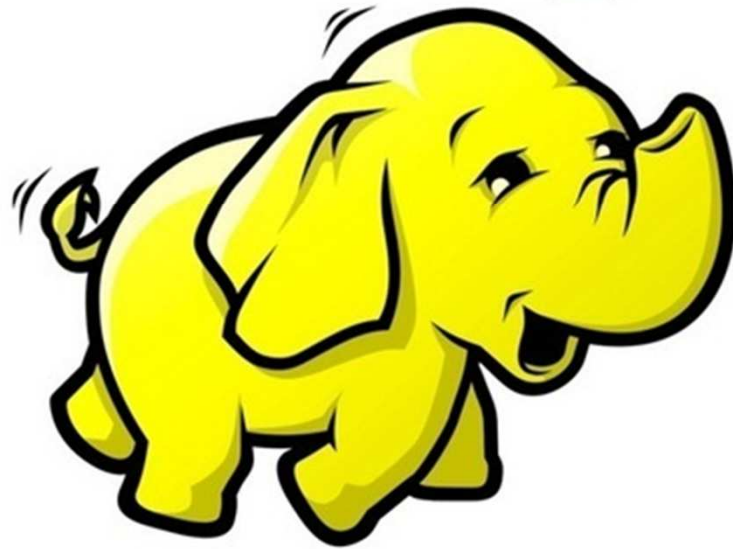
# Execução

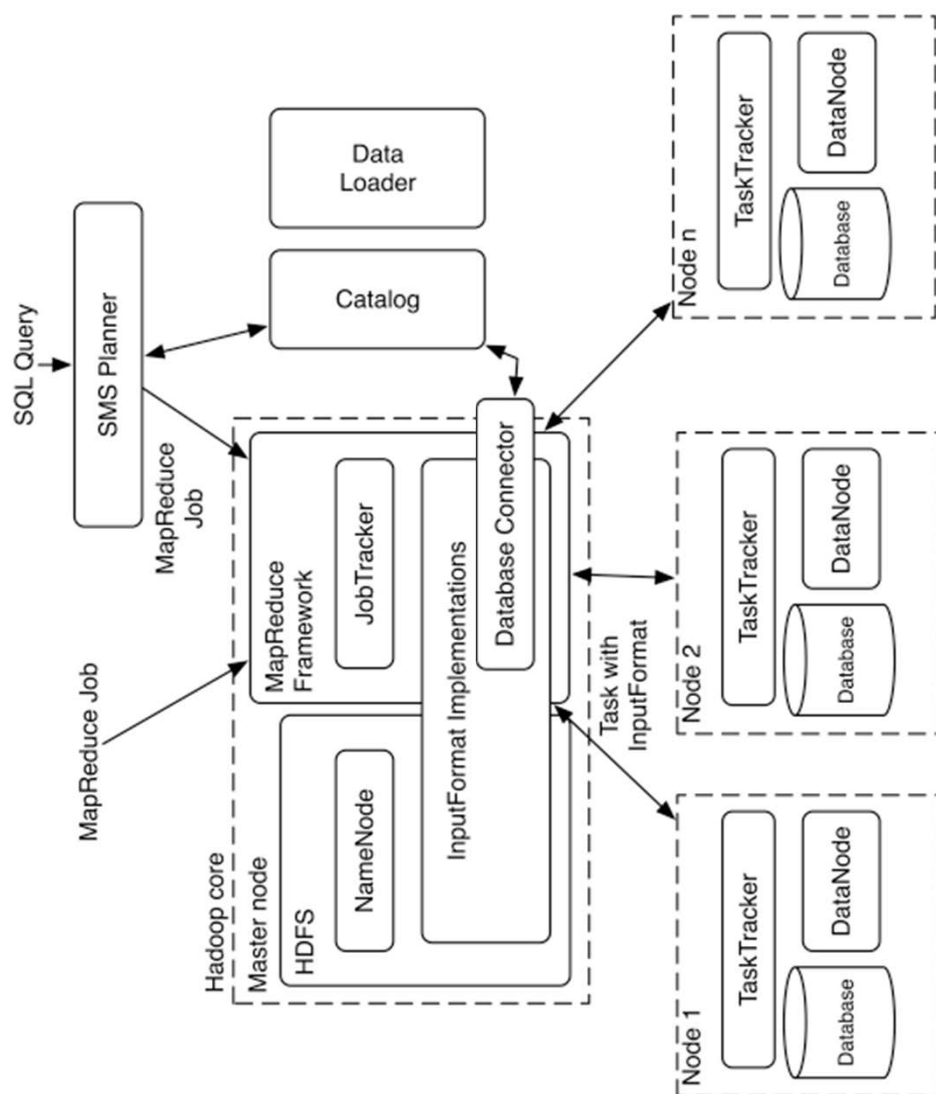
Usuário especifica (job):

M: número de tarefas de map

R: número de tarefas de reduce

SQL on *hadoop*





```

public class WordCount {

    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
            output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
            IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyValueClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

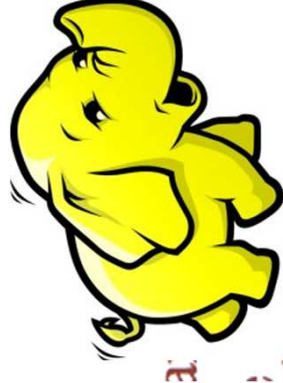
        JobClient.runJob(conf);
    }
}

```

## Map function

## Reduce function

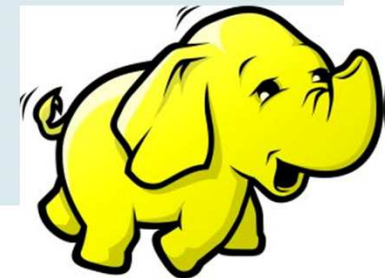
Run this program as a  
MapReduce job



# Apache Pig

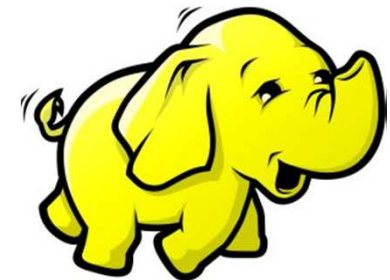
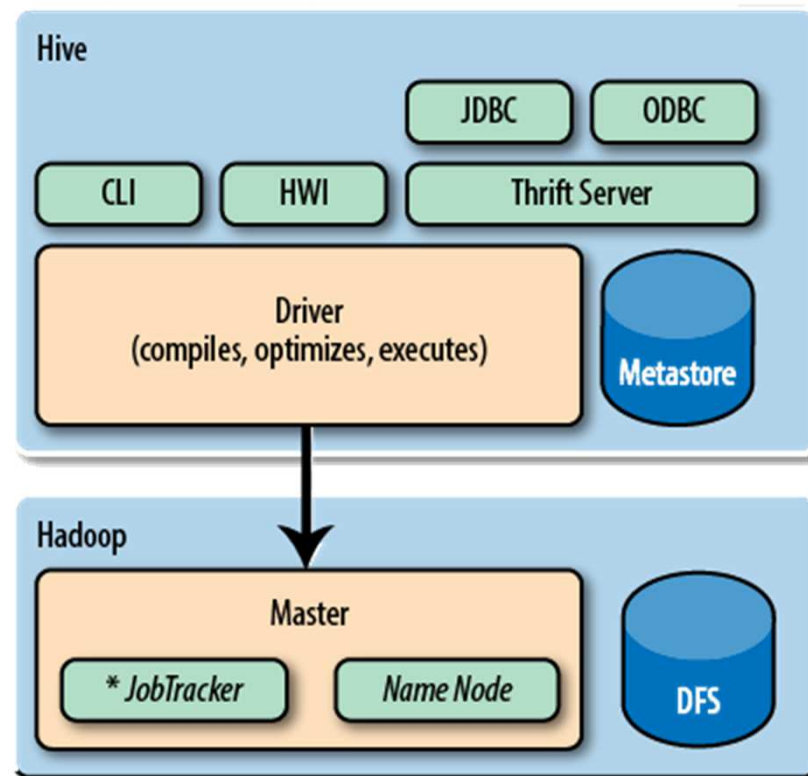
```
input_lines = LOAD '/tmp/books' AS (line:chararray);
-- Extract words from each line and put them into a pig bag
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces filtered_
words = FILTER words BY word MATCHES '\\w+';
-- create a group for each word
word_groups = GROUP filtered_words BY word;
-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words)
              AS count, group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words';
```





# Hive



## Pig Latin

```
countries = load '/user/gharriso/PIG_COUNTRIES' AS
(country_id, country_name , country_subregion , region);

customers= load '/user/gharriso/PIG_CUSTOMERS' AS
(cust_id,first_name, last_name, gender, yob, marital, postcode,city,country_id);

asianCountryys = filter countries by region matches 'Asia';

joined = join customers by country_id, asianCountryys by country_id;

grouped = group joined by country_name;

agged = foreach grouped generate group, COUNT(joined.customers::cust_id);

morethan500cust = filter agged by $1 > 500;

ordered =order morethan500cust by $1 desc;

dump ordered;
```

## SQL or Hive QL

```
SELECT country_name,COUNT(cust_id) AS cust_count
```

```
FROM countries co
```

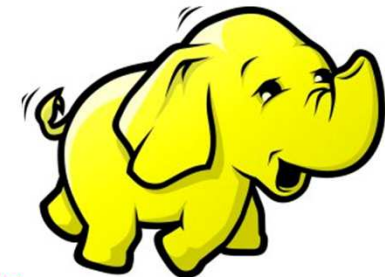
```
JOIN customers cu
ON (co.country_id=cu.country_id)
```

```
WHERE country_region='Asia'
```

```
GROUP BY country_name
```

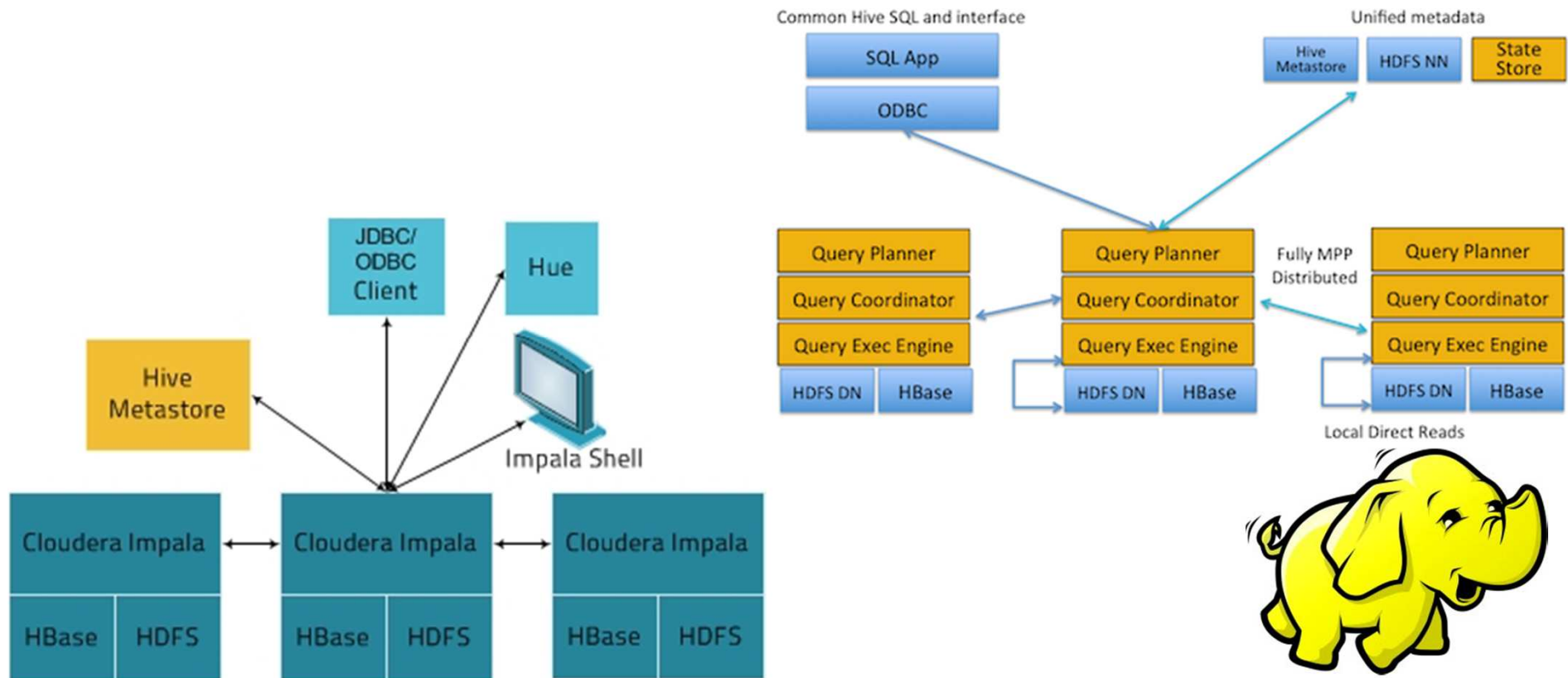
```
HAVING COUNT(cust_id)>500
```

```
ORDER BY cust_count DESC
```



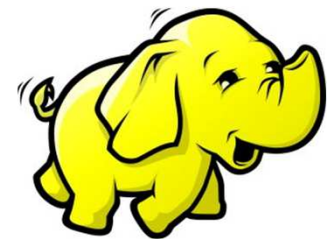
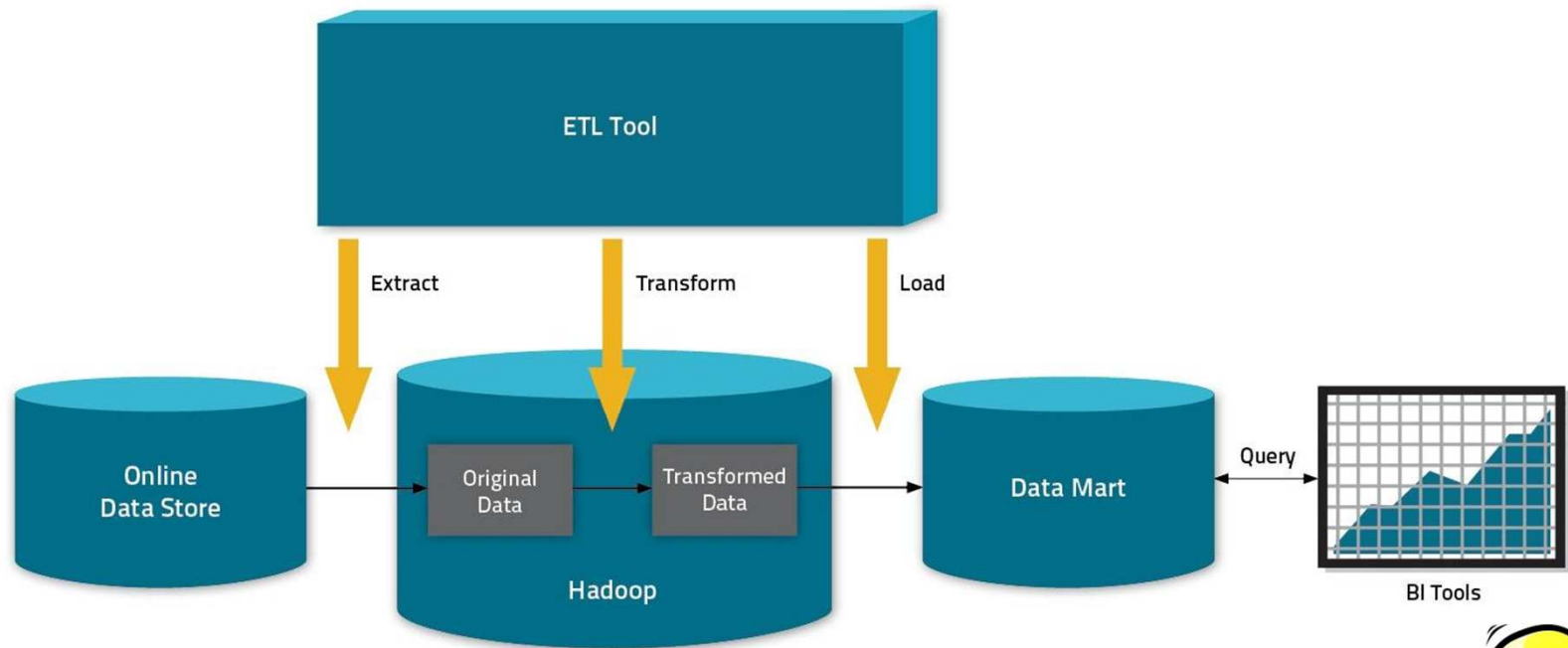


# Impala

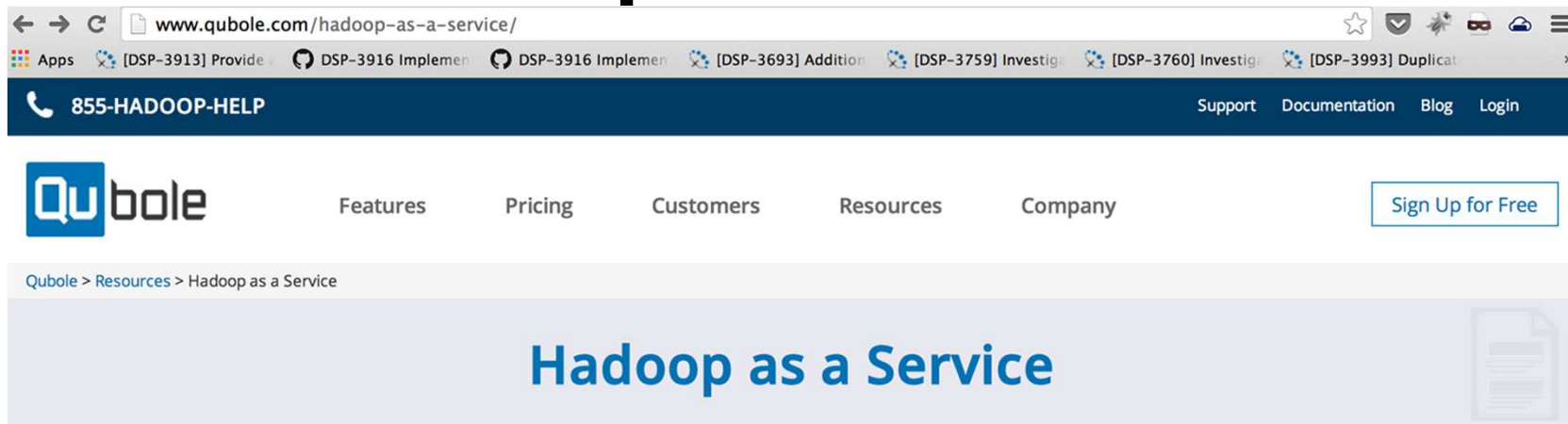




# DataWarehouse Moderno



# Hadoop as a Service



The screenshot shows a web browser window with the URL [www.qubole.com/hadoop-as-a-service/](http://www.qubole.com/hadoop-as-a-service/). The browser's address bar and tabs are visible at the top. The website's header is dark blue with a phone icon and the text "855-HADOOP-HELP" on the left, and links for "Support", "Documentation", "Blog", and "Login" on the right. Below the header is a navigation bar with the Qubole logo, links for "Features", "Pricing", "Customers", "Resources", and "Company", and a "Sign Up for Free" button. The main content area has a breadcrumb trail: "Qubole > Resources > Hadoop as a Service". The title "Hadoop as a Service" is displayed in large blue text. A faint document icon is visible on the right side of the main content area.




Hadoop as a Service, as offered by Qubole Data Service (QDS) or Amazon Web Services' Elastic MapReduce (EMR), describes a cloud computing solution that abstracts away the operational challenges of running Hadoop to make medium and large scale data processing accessible, easy, fast, and inexpensive. This article introduces Hadoop and some of its most useful tools, and how they can be used as a cloud computing service.

Related Blog: [Forbes: Qubole Data Service Road to Hadoop](#)

## Hadoop

Hadoop is a platform for processing large scale data sets in a distributed fashion. In recent years, it has established itself as the de facto big data processing platform in many companies. Hadoop can be used for small clusters of a few nodes up to several thousand computing nodes. While it is an extremely powerful platform, it poses a technology hurdle

## Contact Qubole

-  [Request free trial](#)
-  [Ask a Question](#)
-  [855-HADOOP-HELP](#)



Lightning-Fast Cluster Computing

# Spark

Alternativa ao Map-Reduce

Código fonte aberto (em Scala)

Alta performance (100x que MR)

Grandes massas de dados

Sistema em cluster de baixa latência

# Spark

Usa processamento em memória

APIs em Scala, Python, e Java

<http://spark.apache.org/>