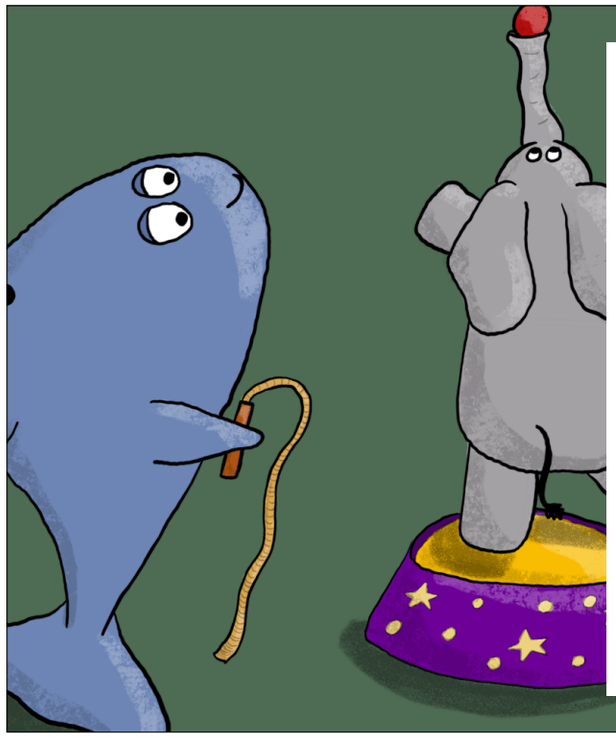
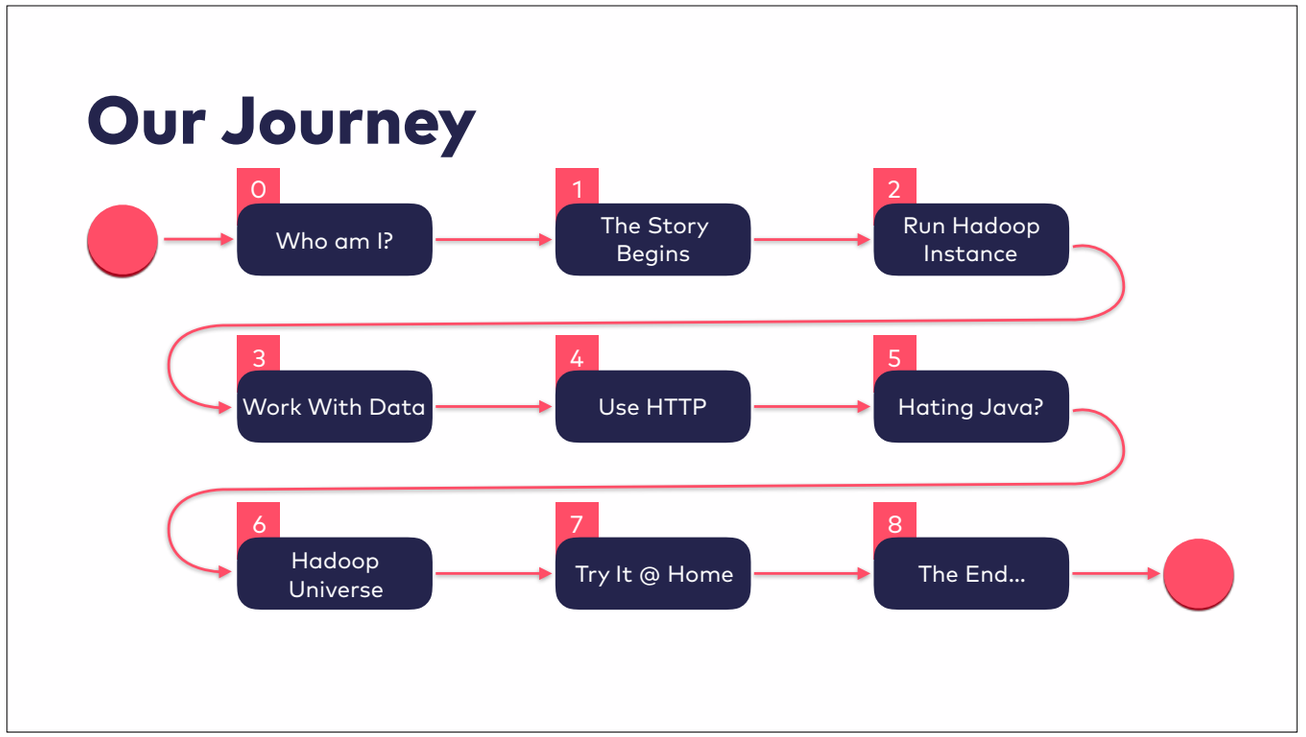


2019-01-14

Data Engineering Meetup, Munich

Hadoop - Taming the Elephant (With a Whale)

INNOQ



The Agenda aka Our Journey:

0. Introduction with some words about myself
1. The explanation on how the story begins, where everything started
2. The first experience: Running a Hadoop instance
3. Go on and work with some data
4. Usage of the HTTP API of Hadoop
5. I've heard of people not liking Java, do they have any other options to use Hadoop, besides using HTTP?
6. A small summary of the Hadoop universe, to have a better overview
7. Some recommendations for trying it at home going deeper
8. Conclusion and some final words

I will use 📖 for marking theory-parts and 🛠️ for marking practical-parts within this presentation.

0

Who am I?

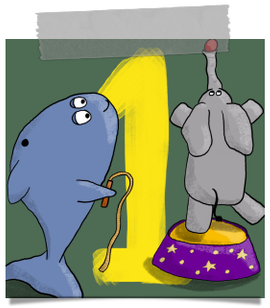
Lisa Maria Moritz
lisa.moritz@innoq.com

Teapot4181

Consultant since
September 2018

INNOQ
www.innoq.com

My name is Lisa Moritz and I’m working as an consultant for INNOQ since September 2018. I have a Twitter account and an e-mail address and of course a phone number.
I really like the HTTP-status-code 418, as you may have noticed.

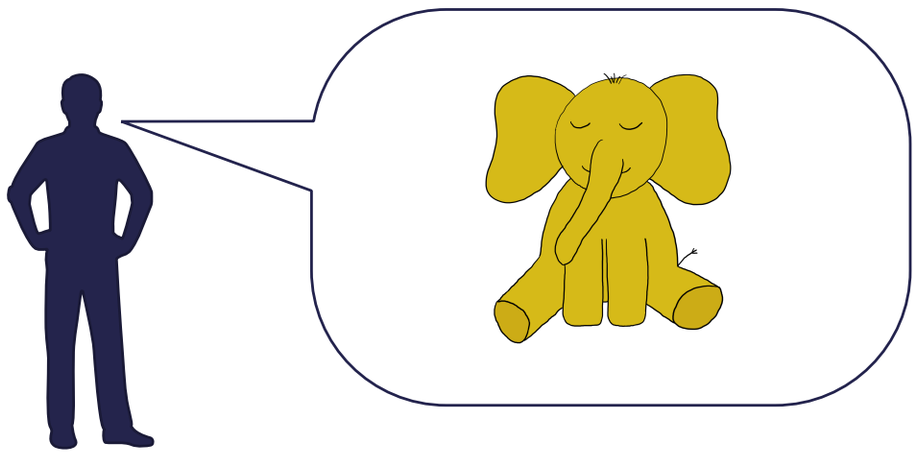


The Story Begins

Part 1: The Story Begins
Yes, I am going to tell you my story :)

1

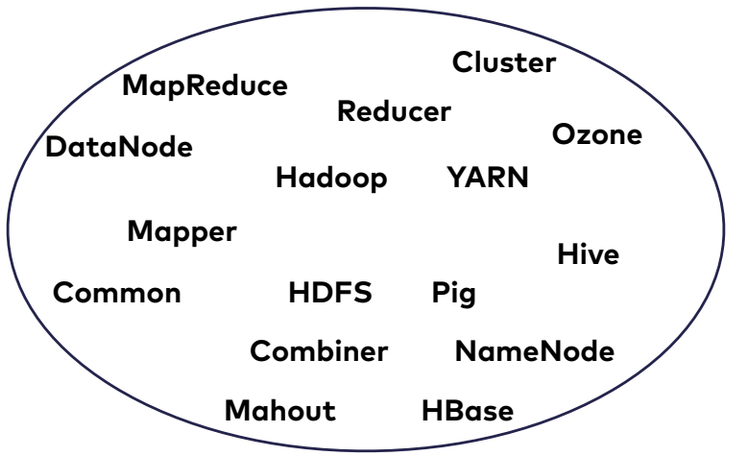
How the Story Began...



In September, when I started working at INNOQ, a colleague of mine came over and told me about a really interesting project. Containing a lot of interesting buzz-words like Kubernetes, Docker, Java, Spring, REST, … I've already worked with. But he also mentioned, that Hadoop knowledge, at least a basic understanding, is required to take part in this project, so I had to start figuring out, what Hadoop is and how to use it; that I may be able to take part in this pretty interesting project.
This is where my Hadoop journey starts…

1

More Than One Elephant

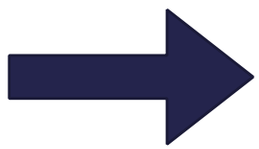


When I started to search for Hadoop at Google, I immediately recognised, that it is a big deal and a big universe. (Yeah, I know, it handles big data, of course it is some kind of ‘big’). Hadoop has a really big ecosystem and even Hadoop itself consists of multiple parts. You find a lot of information on how to set up your Hadoop cluster. But I have to admit, my goal was to get a quick and dirty introduction, which has a lot of hands on, instead of reading endless pages. I was not very happy with this. Also the cluster-stuff was not really interesting, for me it would be not necessary to set up a cluster. Of course, I did not know exactly, what the project requirements really were and how deep the knowledge of Hadoop should be, but all those things I found made it not easy to just discover things.
This is the reason, why I thought, that I should try out another way, which should work better for me.

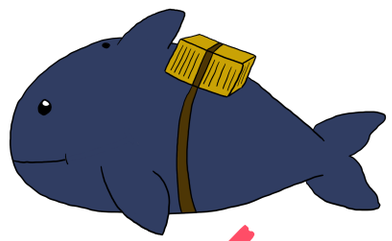
1

Break Into Smaller Pieces

Cluster



X



```
sequenceiq/hadoop-docker:2.7.0
```


I went to Docker Hub and searched for “Hadoop” stubbornly and it worked. There was an image, just a small Docker-image, not a cluster or anything.
This is great, the Docker-image will run on every machine, I can have real practical experience and can try out, how things work.
The Docker image, I’ve used is from sequenceiq and I will use it within this presentation, too.
We have something to work with, so let’s start by doing small steps and discovering the world of Hadoop with some practical things :)

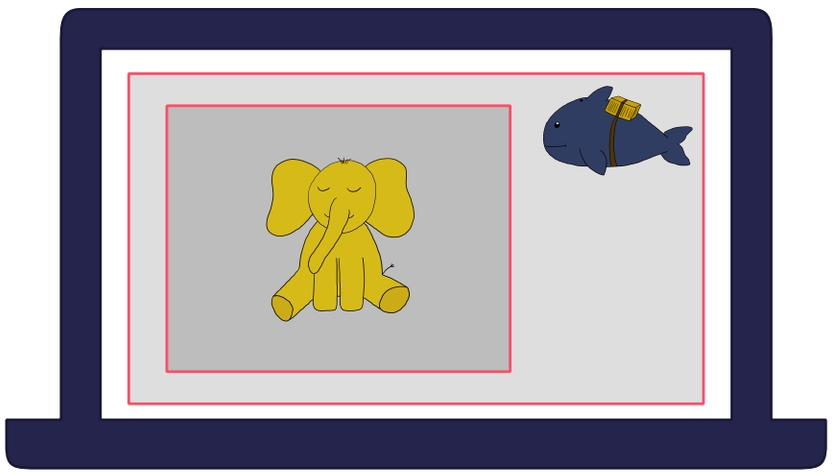
Run Hadoop

Part 2: Run Hadoop

Until now, I have only the knowledge, that a Docker-image is available. But it is not up and running right now. First things first: Let’s start the image.

2

📖 Goal - Part 2



The goal of this part, is to start Hadoop on our machine within a Docker container and test, if the installation is working.

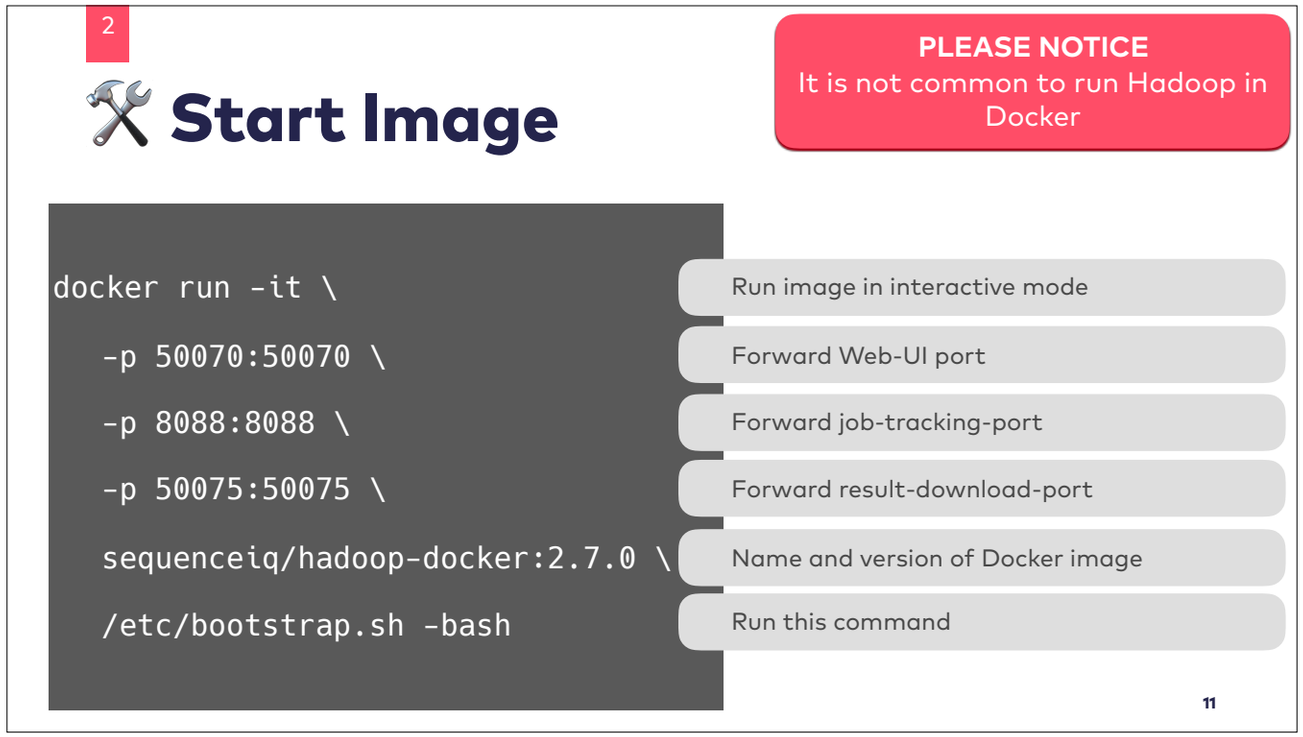
2

📖 Some Docker Basics

Show all running containers	`docker ps`
Get bash in running container	`docker exec -it <container-id> /bin/bash`
Stop running container	`docker container stop <container-id>`
Start stopped container	`docker container start <container-id>`

10

To start the image and in general to work with it, I think, it is useful to have some Docker-commands at hand. If you don’t know Docker, here is a summary of four important commands.
1.‘docker ps’ will show you a list with all your running Docker containers.
2. ‘docker exec -it <container-id> /bin/bash’ will open a bash inside a running container. The container-id can be identified by using the list of running containers. If you will ever run into an error, after typing this command, it is likely, that your image is based on Alpine-Linux, then you should try the command ‘docker exec -it <container-id> /bin/ash’)
3. ‘docker container stop <container-id>’ enables you to stop a running container, while
4. ‘docker container start <container-id>’ will restart a stopped container



Before we start the image locally, I want to keep you in mind, that Hadoop is not designed to run on only one machine, it acts best, when it runs within a cluster of multiple machines. I only wanted to have a quick and dirty introduction, instead of production-aware experience.
To get the container up and running, you only have to run the command ‘docker run -it -p 50070:50070 -p 8088:8088 -p 8020:8020 sequenceiq/hadoop-docker:2.7.0 /etc/bootstrap.sh -bash’.
The most interesting parts in this command are:

- ‘docker run -it’: which enables you to run an image in interactive mode (we can use the bash in the container directly, for example)
- ‘-p’ identifies a port-forward. The most important port-forward is ‘-p 50070:50070’ because this is the port-forward for Hadoop’s web UI.
- ‘sequenceiq/hadoop-docker:2.7.0’ is the image we are going to use. ‘sequenceiq’ is the company’s name, ‘hadoop-docker’ is the name of the image and ‘2.7.0’ is the version number of the image.
- The command ‘/etc/bootstrap.sh -bash’ will be run in the container, when the machine has started.

Small tip: if you add ‘--rm' to the Docker run command, the container will be deleted, right after it is stopped.

2

/etc/bootstrap.sh?

- Included script within sequenceiq Docker-image
- Important to know:
  - Starts services
  - Has two argument-options:
    - -d : Detached, Background-job
    - -bash: Open bash right after container start

The /etc/bootstrap.sh (The command we handed over to the container, when starting the image), is a script which is bundled inside the image. The script performs things to make Hadoop work inside the Docker container. It is important to know, that this script starts the services, which are important and it accepts to arguments:

1. -d : This will run the container in the background (detached)
2. -bash: This will bring you into a bash right after the container-start-up

2

🛠️ Run Example Job 1/2

Folder containing e.g. Examples

```
$HADOOP_PREFIX/share/hadoop/mapreduce
```

MapReduce Example Java App

```
hadoop-mapreduce-examples-2.7.0.jar
```

Running an Example MapReduce-Job

```
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.7.0.jar grep input output 'dfs[a-z.]+'
```


Now that we have started the image, we want to know, if it really runs properly. Of course, we can have a look at the list of running containers, but this will not ensure, that Hadoop is running well inside the container. To ensure this, we can run our first MapReduce-job.
I will explain later, what MapReduce exactly is, and how to create a MapReduce-job! No worries at this point.
Hadoop delivers some example MapReduce-jobs, that we can execute without any further knowledge. So we can run them and see if everything works.
To run the jobs, it is helpful to change to the $HADOOP_PREFIX-directory.
Then you can run the command ‘bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar grep input output ‘dfs[a-z.]+”, which will run one of the example-implementations on some testate, which is already inside.
Those MapReduce-examples are written in Java and you can check them out on Apache’s GitHub-repository. (https://github.com/apache/hadoop/tree/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples)
$HADOOP_PREFIX contains Hadoop’s path: ‘/usr/local/hadoop’.

2

🛠️ CLI - Check

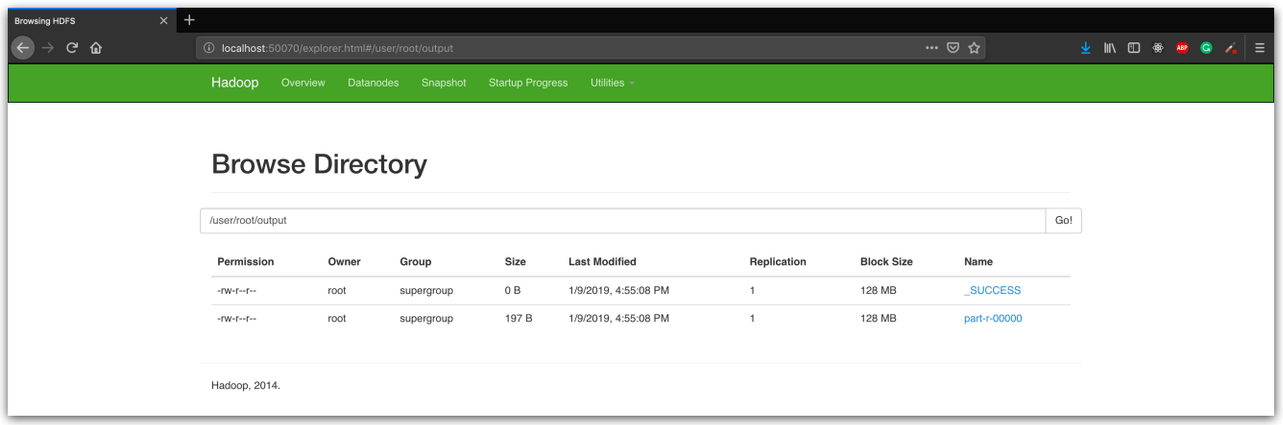
```
bin/hdfs dfs -cat output/*
```

```
bash-4.1# bin/hdfs dfs -cat output/*
6       dfs.audit.logger
4       dfs.class
3       dfs.server.namenode.
2       dfs.period
2       dfs.audit.log.maxfilesize
2       dfs.audit.log.maxbackupindex
1       dfsmetrics.log
1       dfsadmin
1       dfs.servers
1       dfs.replication
1       dfs.file
```


The MapReduce-job-example has performed some action on some data. If you may have noticed, it did a grep on names and searches for things starting with “dfs” in the input data. To see the results of the MapReduce-example, you have two options for checking:
The first possibility is using the command line interface, by running: ‘bin/hdfs dfs -cat output/*’. This will show you the above output.

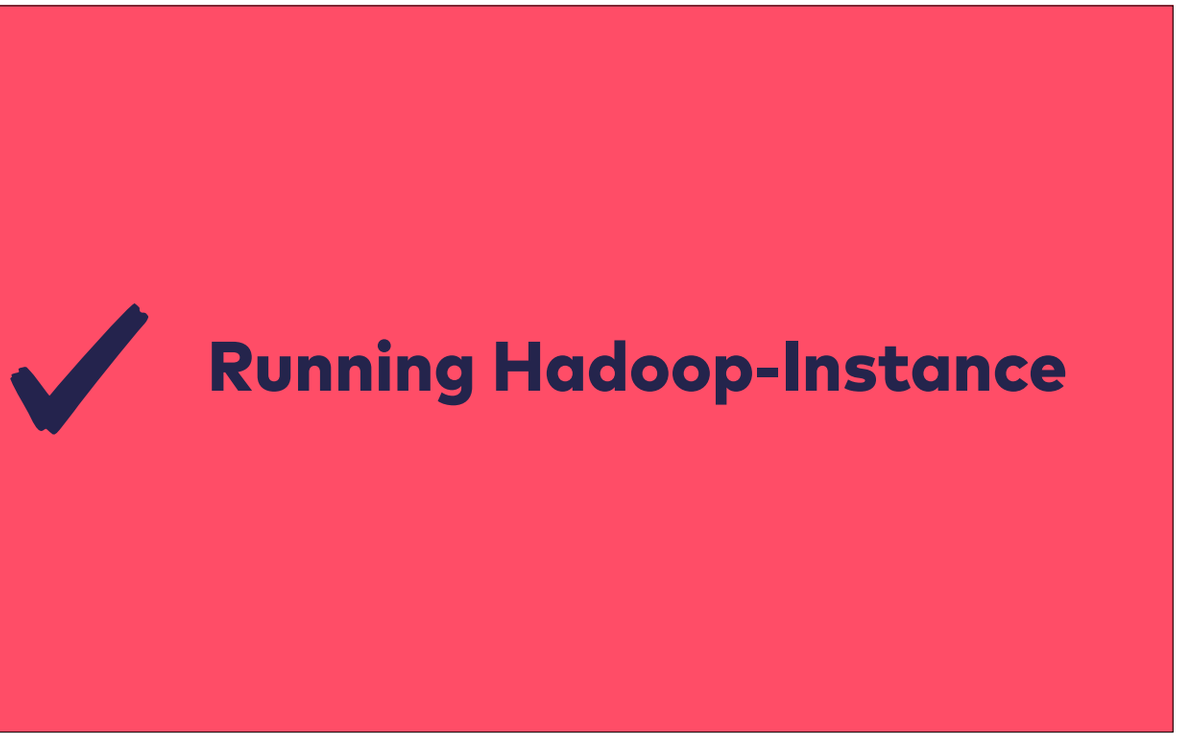
2

🛠 Web UI - Check



Besides the command line interface, you can also use the web user interface, which is provided by Hadoop. Due to the Docker-image start-up with port-forwarding, you can easily enter the web-ui by typing ‘localhost:50070’ in the browser of your choice.
By using the web UI, you also have the possibility to browse through the HDFS (more on what this means later on).

If you choose “Utilities” and then “Browse the file-system”, you can start browsing. And if you navigate to “/user/root/output”, you can see the results, generated by the MapReduce-job, we have run before.



Great, the first step is taken!
We have a running Hadoop-instance on our machine, and we also did prove, that it’s working by running an example provided by Apache.

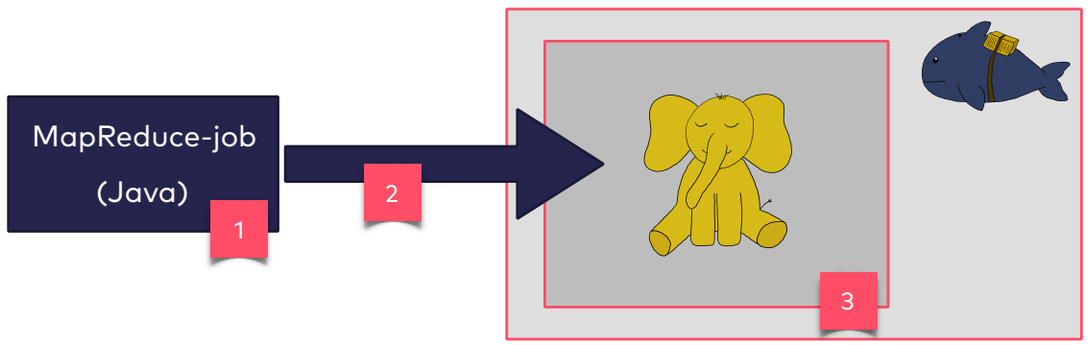
Working with Data

Part 3: Working With Data

We have a working instance, but in fact, until now we did not do very much of our own. Of course, we can be proud of making it run, but I think, we should go a step further and try to work with our own data and right our own MapReduce-job…

3

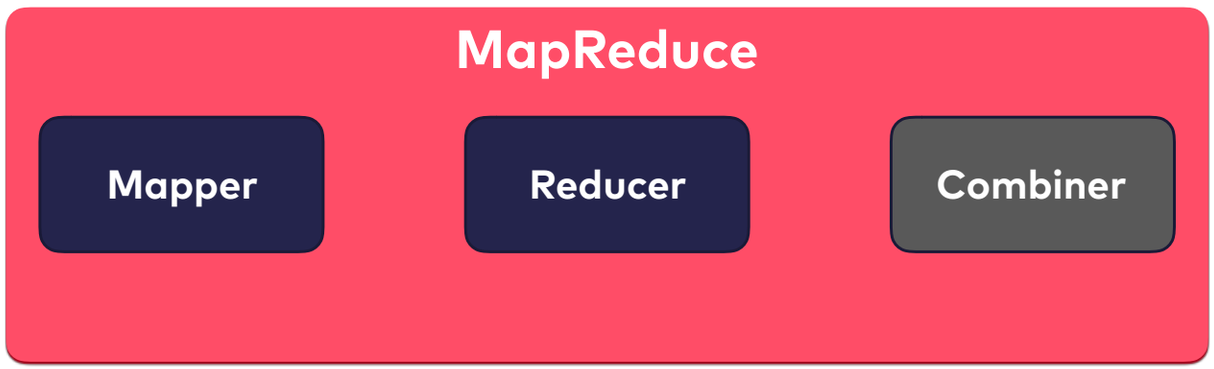
📖 Goal - Part 3



The goal of part 3 is to create our own MapReduce-job with Java (1), transfer it into the Hadoop-container (2) and run it on Hadoop (3). And of course check, if everything has worked well in the end.

3

📖 MapReduce?



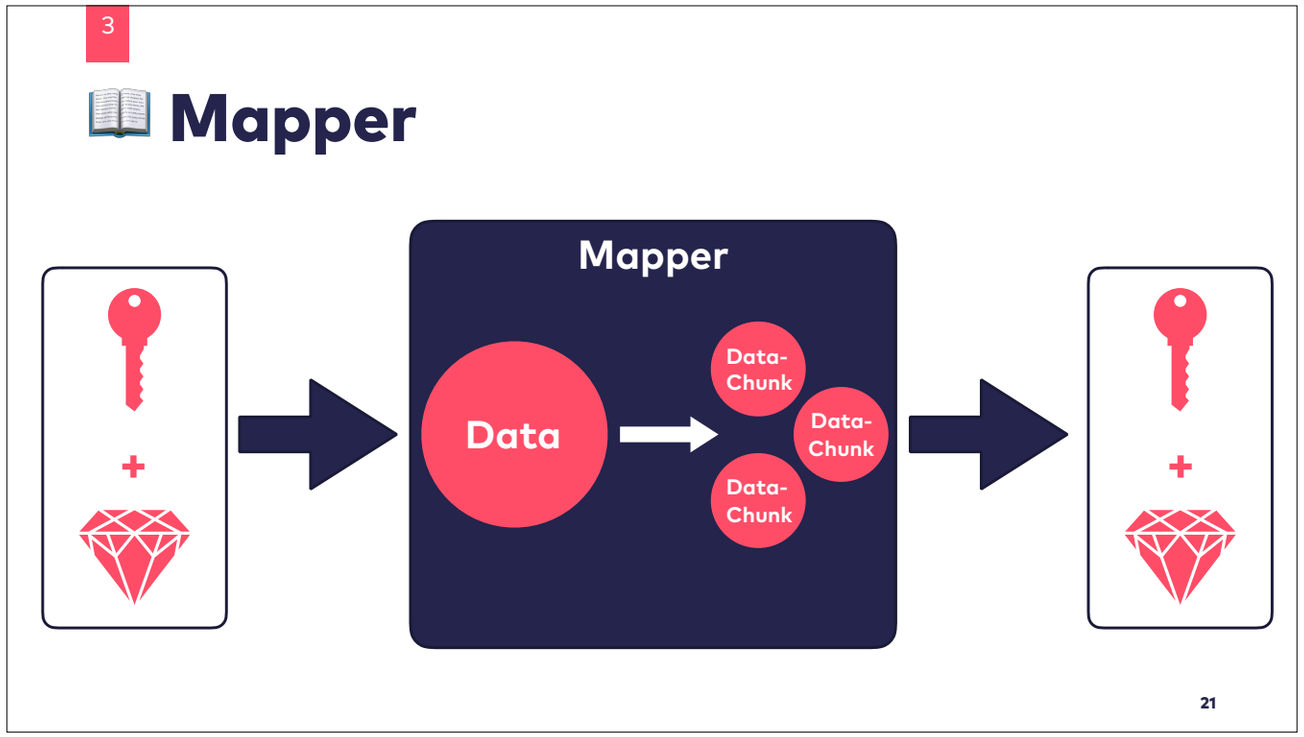
To do so, the first thing, we have to clarify is: What is MapReduce?
MapReduce is a software framework, which is part of Hadoop’s core. It enables you to write jobs, which are able to process a big set of data.
MapReduce consists of a Mapper and a Reducer, these are the necessary parts for your MapReduce-jobs. Some MapReduce-jobs also include a Combiner.
What are those parts and what are they good for?

3

📖 MapReduce: Example

ID	Animal	Name
1	Dog	Berta
2	Cat	Miezi
3	Cat	Fluffy
4	Dog	Baxter
...		

To make it easier to understand, what MapReduce-jobs do, I will show you a small example. Which will be used to explain the complete process.
Let’s say, we have a dataset like the one shown in the table above. The dataset contains date about pets. For every entry we can identify an ID a name and the animal-type.
With the line-key 1 you identify Berta, the dog.
In our example, we want our MapReduce-job to create a list containing the animal-types and how many are there of those kind. The shown excerpt would result into two dogs and two cats.



The Mapper accepts key-value-pairs as input and will produce key-value pairs as output. The function of the Mapper is to split the big set of data into multiple separate chunks of independent data.

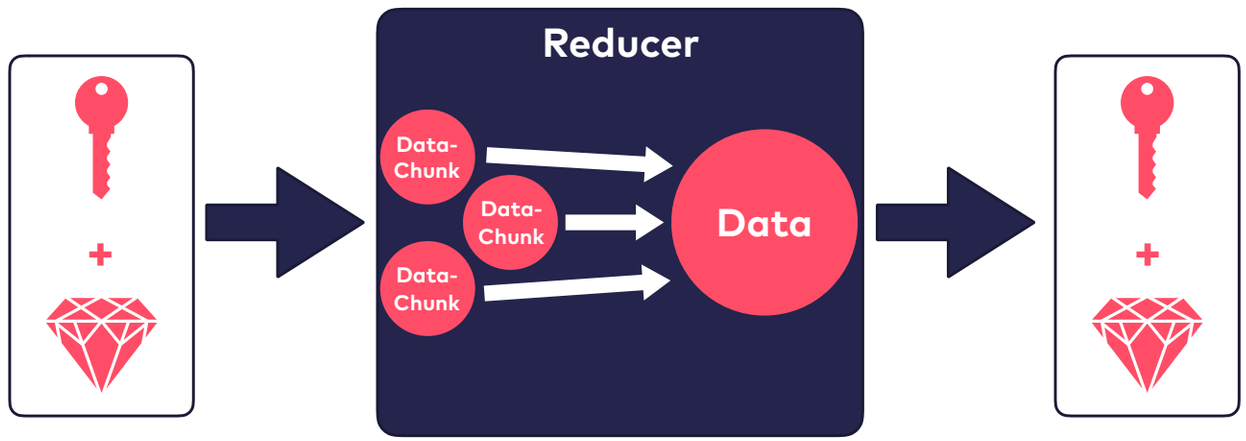
3

📖 Mapper: Example

The Mapper processes the data line by line and adds a unique key to each line. As you can see, the input-key-value-pair of the Mapper is the unique line key as key and the line-data as value.
To count, how many animals of which kind are in the dataset, the most important information is the animal-kind, in this kind ‘Dog’.
So we want our Mapper to extract the animal-kind as the line’s key and the unique line-key as the line’s value.

3

📖 Reducer



The Reducer accepts key-value-pairs as input and produces key-value-pairs as output.
The Mapper will hand over the available keys and a list of their corresponding values to the Reducer. The Reducer itself will combine the split data.

3

📖 Reducer: Example

The Reducer gets a list of key-value-pairs from the Mapper, which is again processed line by line.
The keys of those pairs are the animal-kinds, while the value is a list of unique line-keys.
The Reducer is the last step in our example, it will count, how many line-keys are in each key's list.
The result will be a list containing animal-types as keys and their quantity as value.

3

📖 Combiner

PLEASE NOTICE
Combiners don't replace Reducers.

- Optional
- Minimize network traffic
  - Combine Mapper's output and send chunks of combined data to Reducer
- Reducer can often be used as Combiner

A Combiner is not necessarily part of a MapReduce-job. For our discoveries it is not necessary to use a Combiner, but it is nice to know, what a Combiner does.
To minimize network-traffic, the Combiner combines the content of the Mapper's output and sends chunks of already combined data to the Reducer. This can fasten up your MapReduce-job.
Combiners do usually perform the same steps as Reducers, so it is often possible to use your Reducer also as Combiner.
Please notice, that Combiners does not replace Reducers.

3

🛠 We Need Data

Columns

- A Vendor
- A Category
- A Item
- A Item Description
- A Price
- A Origin
- A Destination
- A Rating
- A Remarks

- **Kaggle:**
  Dark Net Marketplace Data
  (Agora 2014-2015)
- **30.98 MB**
- **csv-file**

Original Dataset: https://www.kaggle.com/philipjames11/dark-net-marketplace-drug-data-agora-20142015
Slightly adapted version: https://github.com/Teapot-418/hadoop-taming-the-elephant/blob/master/darknet-data.csv

After all these theoretical stuff, we should start with something more practical.
We have already worked with some data, but in fact, we do not know, which data it is, and how it got there. And of course our own data is much more interesting than some random data provided by Apache.
My way of getting something “Big Data”-alike was to search Kaggle for some fitting datasets. I’ve started my experiences with a set of data about reviews on women’s clothing. I can tell you, that this is nothing, which thrills everyone, you tell about. Therefore I’ve searched another dataset at Kaggle and found one about the darknet.
—> https://www.kaggle.com/philipjames11/dark-net-marketplace-drug-data-agora-20142015
It shows vendors, categories, prices and a lot of other information. (Of course darknet is more thrilling than women’s clothing)
I’ve modified the set slightly to make it easier to work with. You can find the modified dataset here: https://github.com/Teapot-418/hadoop-taming-the-elephant/blob/master/darknet-data.csv
If you would like to use your own dataset: Don’t hesitate, you will have to adapt the upcoming things slightly, but you will manage to do it :)

HDFS

We will run into this abbreviation more often from now on. This is the abbreviation for Hadoop Distributed File System, that’s where Hadoop stores it’s data in, it is also a part of Hadoop’s core, like MapReduce. More on this topic later on.

3

🛠️ Get Data Into Hadoop

Copy dataset into container

```
docker cp darknet-data.csv \
  ab0978def5ae:/tmp/
```

Put dataset into Hadoop

```
$HADOOP_PREFIX/bin/hdfs dfs \
  -put /tmp/darknet-data.csv \
  /user/root/input/darknet
```


We have a dataset in place and we have a running Hadoop instance in place.
The first thing to make our MapReduce-job happen is to copy the dataset-csv-file into our Hadoop Docker container. To do so, we have to use the ‘docker cp’-command.
‘docker cp <path-on-your-system> <container-id>:<file-path>’
Now the dataset is within the Docker container, but this does not mean, that it is inside “Hadoop”, so to say. We have to add it to the HDFS inside our Docker container.
To do so, perform the following command: ‘/usr/local/hadoop/bin/hdfs dfs -put <file-path> /user/root/input/<name-in-hdfs>’
Of course, we have to adapt <file-path> with the file-path inside the Docker-container and <name-in-hdfs> with the datasets name in HDFS like “darknet”.

3

🛠️ CLI - Check

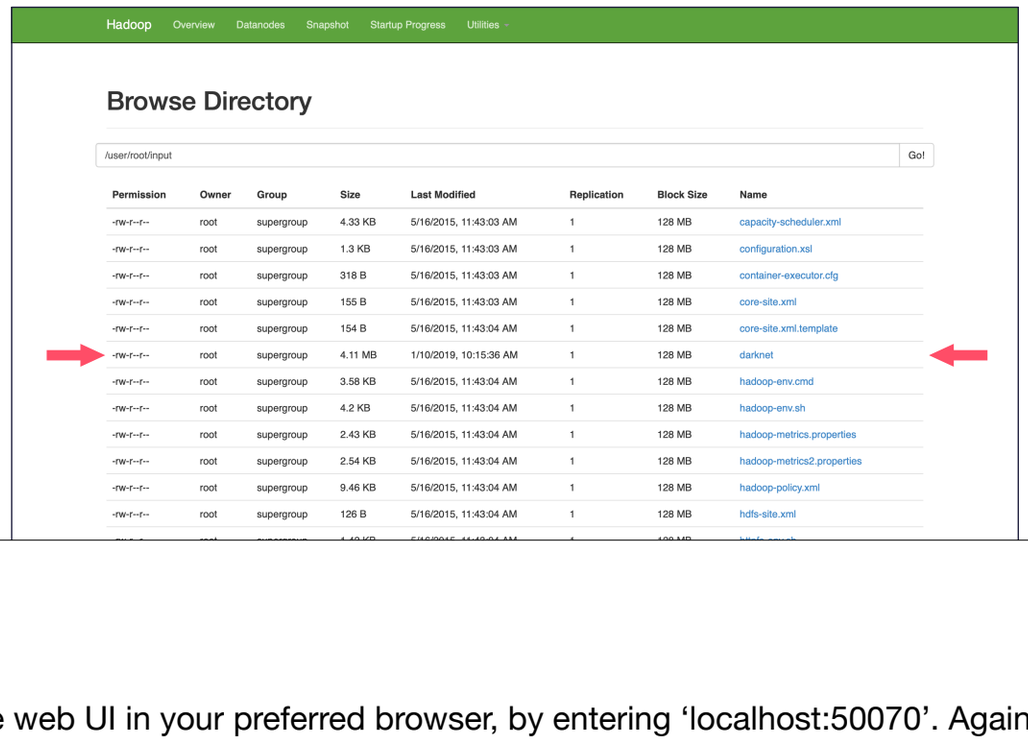
```
/usr/local/hadoop/bin/hdfs dfs -ls /user/root/input
```

```
bash-4.1# /usr/local/hadoop/bin/hdfs dfs -ls /user/root/input
Found 32 items
-rw-r--r--   1 root supergroup       4436 2015-05-16 05:43 /user/root/input/capacity-scheduler.xml
-rw-r--r--   1 root supergroup       1335 2015-05-16 05:43 /user/root/input/configuration.xsl
-rw-r--r--   1 root supergroup        318 2015-05-16 05:43 /user/root/input/container-executor.cfg
-rw-r--r--   1 root supergroup        155 2015-05-16 05:43 /user/root/input/core-site.xml
-rw-r--r--   1 root supergroup        154 2015-05-16 05:43 /user/root/input/core-site.xml.template
-rw-r--r--   1 root supergroup    4311104 2019-01-10 04:15 /user/root/input/darknet
-rw-r--r--   1 root supergroup       3670 2015-05-16 05:43 /user/root/input/hadoop-env.cmd
-rw-r--r--   1 root supergroup       4302 2015-05-16 05:43 /user/root/input/hadoop-env.sh
-rw-r--r--   1 root supergroup       2490 2015-05-16 05:43 /user/root/input/hadoop-metrics.properties
-rw-r--r--   1 root supergroup       2598 2015-05-16 05:43 /user/root/input/hadoop-metrics2.properties
-rw-r--r--   1 root supergroup       9683 2015-05-16 05:43 /user/root/input/hadoop-policy.xml
-rw-r--r--   1 root supergroup        126 2015-05-16 05:43 /user/root/input/hdfs-site.xml
-rw-r--r--   1 root supergroup       1449 2015-05-16 05:43 /user/root/input/httpfs-env.sh
-rw-r--r--   1 root supergroup       1657 2015-05-16 05:43 /user/root/input/httpfs-log4j.properties
-rw-r--r--   1 root supergroup         21 2015-05-16 05:43 /user/root/input/httpfs-signature.secret
-rw-r--r--   1 root supergroup        620 2015-05-16 05:43 /user/root/input/httpfs-site.xml
-rw-r--r--   1 root supergroup       3518 2015-05-16 05:43 /user/root/input/kms-acls.xml
-rw-r--r--   1 root supergroup       1527 2015-05-16 05:43 /user/root/input/kms-env.sh
-rw-r--r--   1 root supergroup       1631 2015-05-16 05:43 /user/root/input/kms-log4j.properties
```


Again there are two ways to check, if it worked. Some people like eye-candy, some people don’t, therefore I stay with my way of showing both.
The first way is to check it via the command line interface. Run the command ‘/usr/local/hadoop/bin/hdfs dfs -ls /user/root/input’ and check if the name, you given your data is listed in the output. If it’s listed, everything worked fine.

3

🛠 Web UI - Check



Checking the web UI requires you to open the web UI in your preferred browser, by entering ‘localhost:50070’. Again choose “Utilities” and “Browse the file-system” and navigate to the root-users directory, but this time choose the input folder. Is the <name-in-hdfs> listed? If yes: Everything worked well.

3

🛠️ Maven Dependencies

Our first experiences with Hadoop and MapReduce will be made with Java. We will use Maven to integrate Hadoop Common and Hadoop MapReduce Client Core into our Java application. This will enable us to easily write Mapper and Reducer.

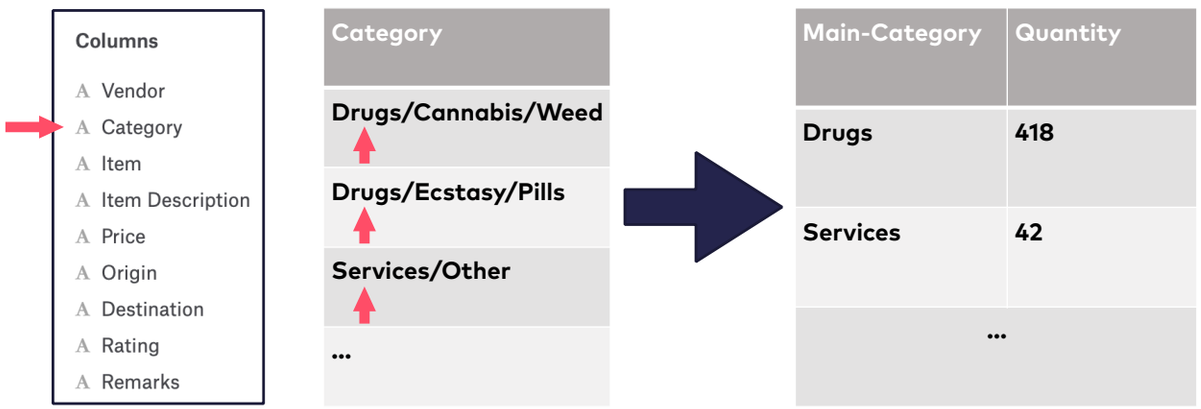
I used these dependencies:

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>3.1.1</version>
</dependency>

<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>3.1.1</version>
</dependency>
```

3

🛠️ What We Want



The idea is to extract the main-categories of the dataset and identify the quantity of each main-category.

Categories in this dataset are split by slashes, the first word is the main-category and every other word identifies a more specific category.
Example: “Drugs/Cannabis/Weed” and “Drugs/Ecstasy/Pills” are both part of the main category “Drugs”

3

🛠️ Create Mapper

Extend Hadoop's Mapper

```
Mapper< [input-key], [input-value],
   [output-key], [output-value] >
```

Override "map"-function

```
public void map(
   [input-key] key,
   [input-value] value,
   Context context
)
```


First things first: Everything starts with the Mapper, therefore we will start with implementing the Mapper.

If you want to create a Mapper the class you create has to extend the Mapper-class. This one needs some specifications about what kind of data is used as key and value for input and output.
The next step to create your own Mapper is to override the map function. Which gets the input's key and value, as well as a context as parameter. This context will be used to write your output to.

3

🛠️ Create Mapper

PLEASE NOTICE
Hadoop uses special value-types.

```
public class MainCategoryMapper extends
    Mapper<LongWritable, Text, Text, LongWritable> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws […] {
        String line = value.toString();
        String[] lineData = line.split(",");
        String[] categories = lineData[1].split("/");
        if(categories.length > 0) {
            context.write(new Text(categories[0]), key);
        }
    }
}
```

Create output

Like I’ve told you before, the Mapper processes the data line by line, and get’s the line-data itself as a value and a line-identifier as key for each line.
The only interesting part of the dataset for our example is the column containing the information about categories. To identify the main-category of a line, we have to split up the category at the Slashes and afterwards identify the first part as main-category. This will be the output’s key.
We will only count, how many times a main-category appears in the dataset, so we can keep it simple and reuse the line-identifier as the output’s value. Like we did in the example with the pets before.
Please notice, that Hadoop is using special value types like “LongWritable” instead of “Long”.

3

🛠️ Create Reducer

Extend Hadoop's Reducer

```
Reducer< [input-key], [input-value],
   [output-key], [output-value] >
```

Override "reduce"-function

```
public void reduce(
   [input-key] key,
   Iterable<[input-value]> values,
   Context context
)
```


Now, that we have the Mapper in place, we can go on with creating the reducer.
As the Mapper had to extend the Mapper-class and override the "map"-function, the Reducer has to extend the Reducer-class and override the "reduce"-function.
The "reduce"-function gets the input's key and Iterable of the value's type, as well as a context as parameter. This context will be used to write your output to.

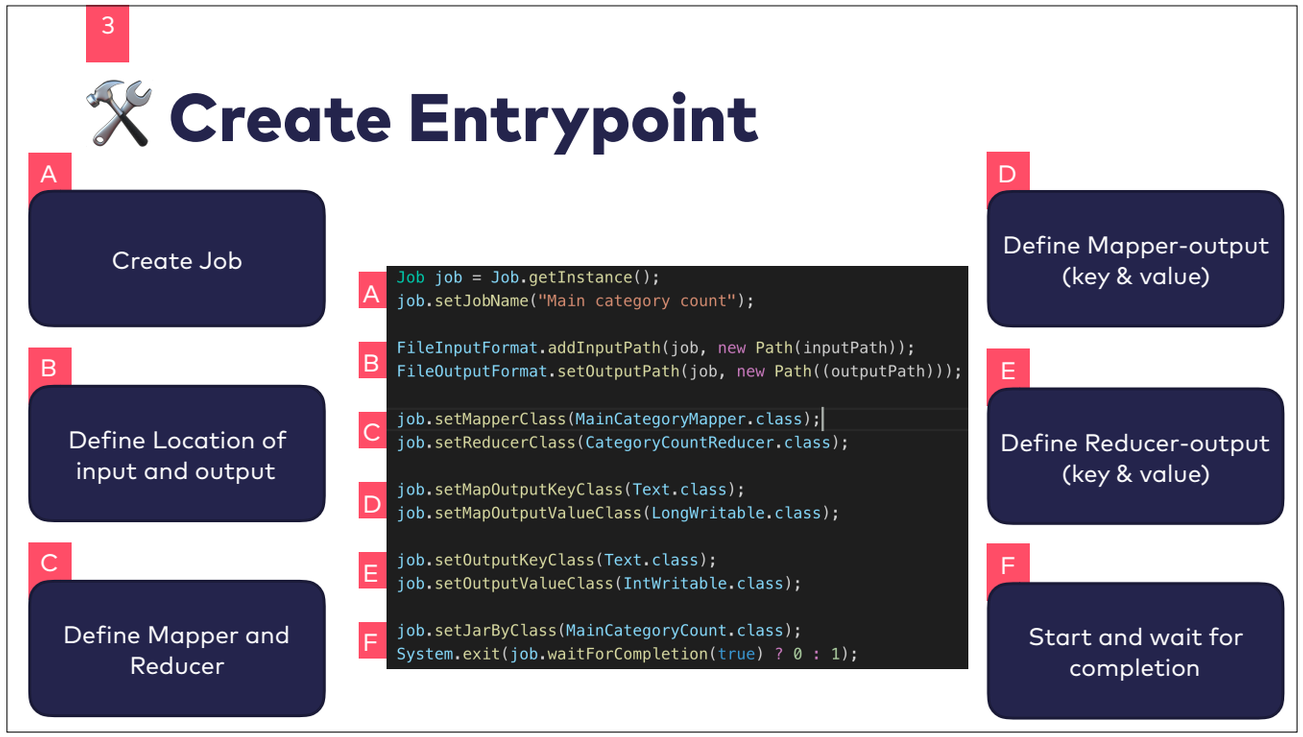
3

🛠️ Create Reducer

```
public class CategoryCountReducer extends
     Reducer<Text, LongWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<LongWritable> values, Context context)
           throws […] {
        int count = 0;
        for (LongWritable value : values) {
            count++;
        }
        context.write(key, new IntWritable(count));
    }
}
```

Create output

As you can see, the iterator allows us to iterate over the list of values for a single key. Therefore the handling is very easy here.



To run our MapReduce-job, we have to provide an entry point.
This is integrated within the Maven pom-file:

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>
              <mainClass>Entrypoint</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
```

The Entrypoint contains the main-method. And enables the execution.

The Entrypoint consists of six parts:

1. Creation of the job itself
2. Definition of the locations of input and output
3. Definition of Mapper and Reducer

3

🛠️ Run it

Build executable jar

```
mvn clean package
```

Find jar into /target

Copy jar into container

```
docker cp darknet-mapreduce.jar\
ab0978def5ae:/tmp/darknet-mapreduce.jar
```

Execute

```
$HADOOP_PREFIX/bin/hadoop \
  jar /tmp/darknet-mapreduce.jar
```


Now that we have a Mapper and a Reducer and an Entrypoint, which knows about Mapper, Reducer, input-path and output-path, we can run our first MapReduce-job.
To run it, we have to create an executable jar-file for our MapReduce job, which simply can be achieved by running the command ‘mvn clean package’, this will put the jar into the ‘target’-folder.
Afterwards the jar has to be copied into the Docker container, where our Hadoop instance is running.
Now that it’s in, we can execute the MapReduce job with the command on the slide.

3

🛠️ Run it

```
[…] INFO mapreduce.Job: The url to track the job: http://ab0978def5ae:8088/
proxy/application_1547046677403_0012/
[…] INFO mapreduce.Job: Running job: job_1547046677403_0012
[…] INFO mapreduce.Job: Job job_1547046677403_0012 running in uber mode : false
[…] INFO mapreduce.Job:  map 0% reduce 0%
[…] INFO mapreduce.Job:  map 100% reduce 0%
[…] INFO mapreduce.Job:  map 100% reduce 100%
[…] INFO mapreduce.Job: Job job_1547046677403_0012 completed successfully
```


This is the main output, while the MapReduce-job is running. Hadoop offers an url to track the job's state as you may have noticed reading the first line.
The job gets a unique identifier and than it shows the progress on mapping and reducing the data.
In the end it hopefully tells you, that everything worked successfully.

3

🛠️ CLI - Check

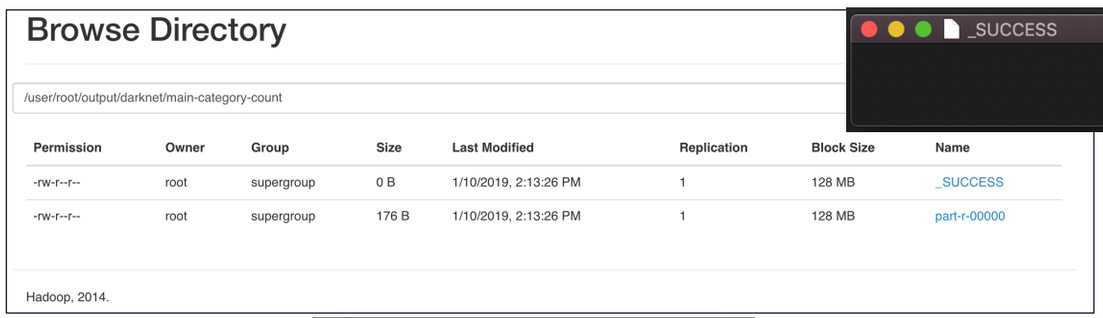
```
$HADOOP_PREFIX/bin/hdfs dfs -cat output/darknet/main-category-count/*
```

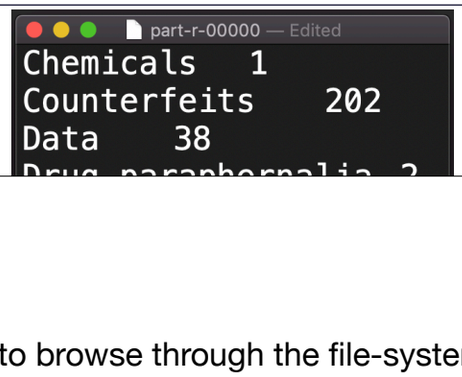
```
Chemicals       1
Counterfeits    202
Data    38
Drug paraphernalia      2
Drugs   11834
Electronics     41
Forgeries       101
Info    15
Information     13
Jewelry 23
Other   97
Services        179
Tobacco 6
Weapons 83
```

Main-Category	Quantity
Chemicals	1
Counterfeits	202
Drug paraphernalia	2
Drugs	11834
...	

3

🛠️ Web UI - Check





If you want to check out the results via Hadoop's Web UI, you have to browse through the file-system again inside the web UI. Navigate to your output-folder.
You will see, that there is not just one file, containing the results, but there are (in our case) two:

- A "_SUCCESS"-descriptor, which is an empty file, showing, that everything went great
- And a "part-r-xxxxx"-file, which contains the results. The results does not have to be stored in one file. There can also be multiple files.

3

🛠️ Remove Results

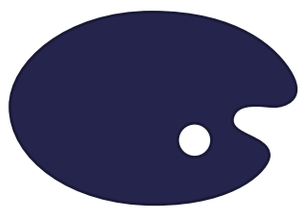
```
$HADOOP_PREFIX/bin/hdfs dfs -rm -r <folder-name>
```

Of course, sometimes stuff goes wrong or you want to rerun things. And of course, you don’t want to give the (in fact) same results different names, only because you’re not aware, that results (and everything in HDFS) are removable.
It’s good to know how to clean up!

3

Be Creative

- Work with this dataset and try to do other things like:
  - Counting detailed categories
  - List the maximum price per main-category
  - ...



There are many possibilities with this one dataset, and I want to motivate you to try it out at home. I would like to hear from your results and how everything is going on. Here are some ideas how to go on, but there are plenty more possibilities. Don’t hesitate to make up ideas on your own.

Great!
Instead of stubbornly running something what is already present on data which is already present; we included our own dataset and we've written our first MapReduce-job and of course used it with our own data. And had a short glance at the results.

Displaying Results (Http)

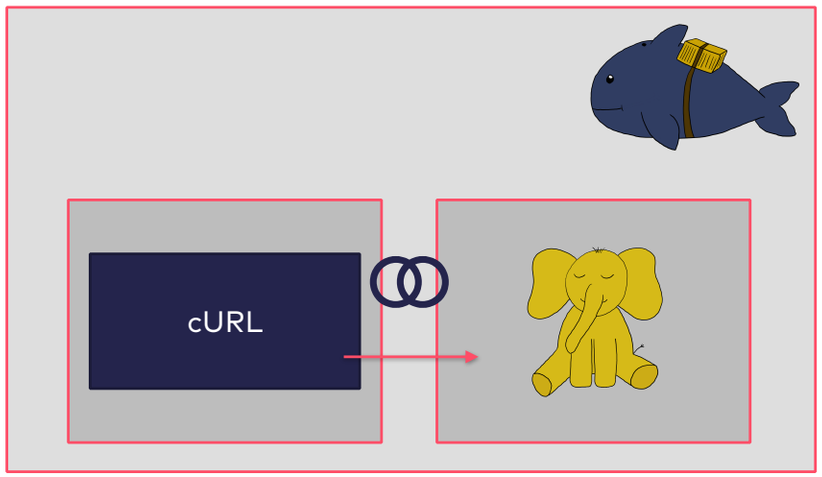
Part 4: HTTP

Now, that we've build our first MapReduce job, you may wonder if there is any way to get the processed data. Maybe you want to build a super fancy web frontend to show your findings.

To do so, this part will show you, how to use the build-in HTTP API.

4

📖 Goal - Part 4



Now, that we have created some results in part 3, we will see, how we could fetch those results from another application via HTTP, in this case we use cURL to do so, but you can adapt those knowledge afterwards and create your own application, using the MapReduce-results. We will link the Docker containers and then fetch information via HTTP.

4

📖 HttpFS?

Hadoop HDFS over HTTP

Hadoop HDFS over HTTP, short HttpFS, provides a REST HTTP gateway supporting all filesystem operations, that can be performed in HDFS.

4

🛠️ Linking Docker-Containers

```
docker run \
   -it \
   --link 7680676ced68:hadoop \
   –name curl_container \
   ubuntu:latest
```

Start in interactive mode directly

Link to container and give internal name

```
/ # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      hadoop 7680676ced68 compassionate_robinson
```

To perform the following http-requests from one Docker container to another, we have to link the two containers to make it work.
For our second container, which we will use as a linked container to perform the http-requests, we will use the basic Ubuntu image.
We link this Ubuntu-container to our running Hadoop instance.
When we have a look at the hosts-file, of the newly created container, we can see, that the linking worked.

4

🛠️ Navigate through HDFS

The Base URL

```
http://hadoop:50070/webhdfs/v1/user
```

Browsing Directories

```
curl -i -L \
“http://hadoop:50070/webhdfs/v1/user/
root/output/?op=LISTSTATUS"
```

Open Files

```
curl -i -L \
“http://hadoop:50070/webhdfs/v1/user/
root/output/darknet/category-count/
part-00000?op=OPEN”
```


Now, that we have cURL in place, we can have a closer look at HttpFS. The base url, to navigate through HDFS is ‘http://hadoop:50070/webhdfs/v1/user' while ‘hadoop’ identifies the internal name of the linked Docker-container (which is mentioned in /etc/hosts.
From there you can navigate through HDFS via HttpFS.
We can browse through directories and open files.

4

🛠️ Navigate through HDFS

Browsing Directories


```
{
    "FileStatuses": {
        "FileStatus": [{
            "accessTime": 0,
            "blockSize": 0,
            "childrenNum": 1,
            "fileId": 16435,
            "group": "supergroup",
            "length": 0,
            "modificationTime": 1547137495047,
            "owner": "root",
            "pathSuffix": "darknet",
            "permission": "755",
            "replication": 0,
            "storagePolicy": 0,
            "type": "DIRECTORY"
        }]
    }
}
```


Open Files


```
Pragma: no-cache
Content-Type: application/octet-stream
Location: http://7680676ced68:50075/webh
dress=7680676ced68:9000&offset=0
Content-Length: 0
Server: Jetty(6.1.26)

HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET
Access-Control-Allow-Origin: *
Content-Type: application/octet-stream
Connection: close
Content-Length: 1905

Counterfeits/Accessories        1
Counterfeits/Clothing   8
Counterfeits/Electronics        10
Counterfeits/Money      5
```


When browsing directories, the response contains “type”, which identifies, if the next file is again a DIRECTORY or a FILE. “pathSuffix” identifies the name of the directory/file, which is described.

When you open a file, the data is send as “application/octet-stream” as you can see in the http-response-header. The data is shown in the body.

The experiences we’ve made with cURL can now be adapted to create a client, collecting data via HttpFS. We know, how to browse directories and are also aware of how to open files to receive the results of our MapReduce-jobs.
To create your client, you can use any programming language, which is aware of using HTTP. Maybe you would like to create a JavaScript-based web application which displays your data?

Now we have also an idea on how to fetch the results via Http :)

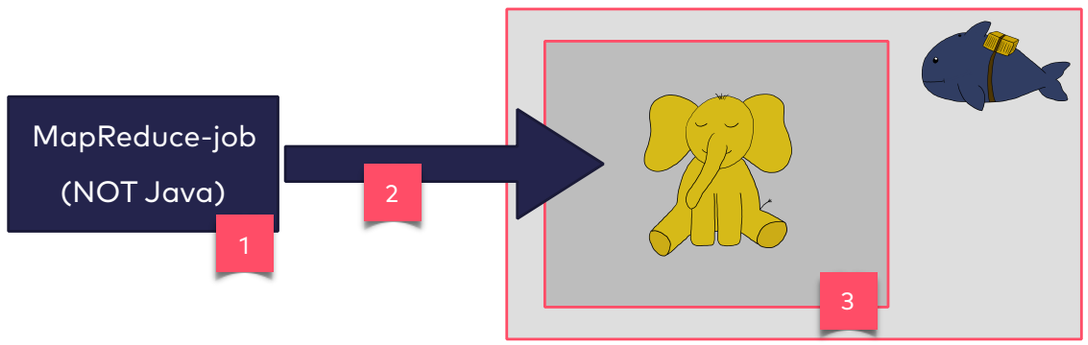
I don’t like Java 😕

Part 5: Hating Java?

I’ve heard of people using other languages than Java, maybe it’s just a tale, that they are outside. But if they exist, we should definitively show them, how they can build MapReduce-jobs without writing Java-code!

5

📖 Goal - Part 5



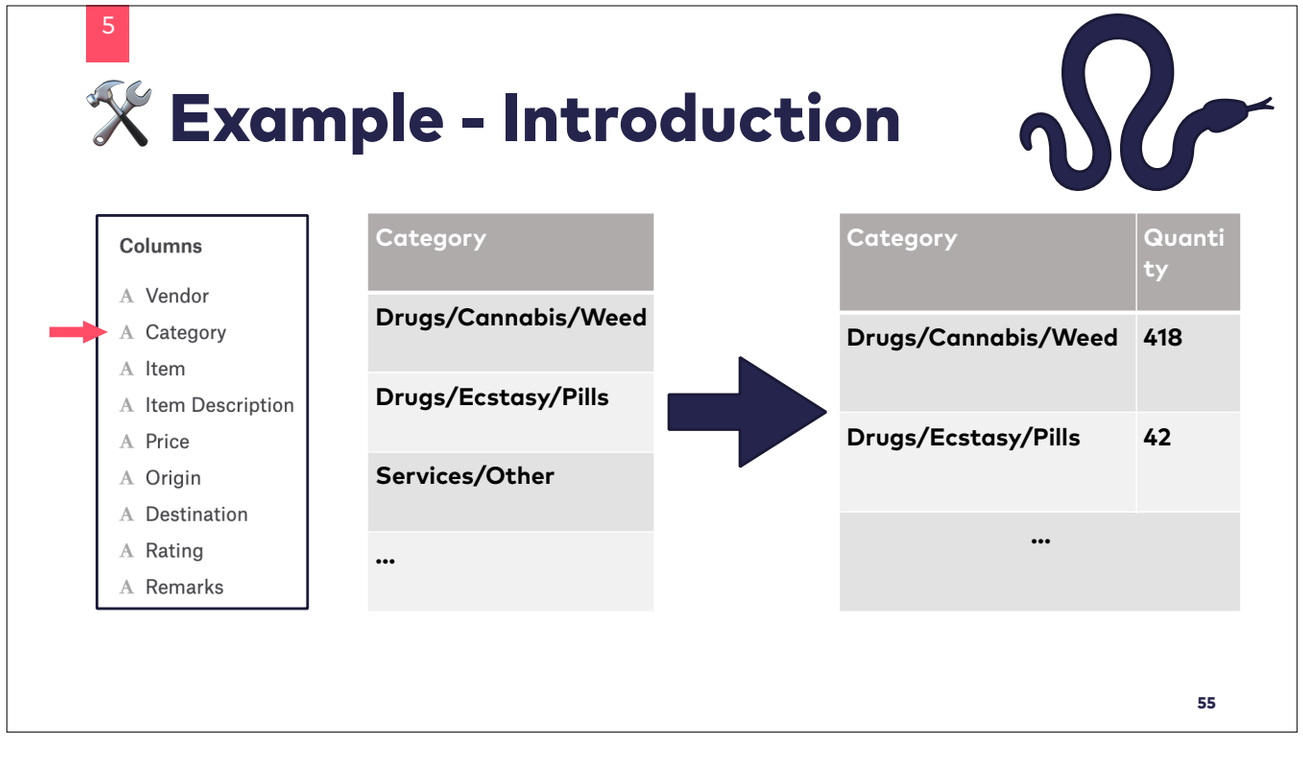
Part 5 will show how to create a MapReduce-job without using Java (1), put the job into the Docker container (2) and afterwards run it in Hadoop (3); of course we will check for the results again in the end.

5

📖 Hadoop Streaming API

Reducer receives keys in order

Of course Hadoop provides a way to use it, without writing Java code.
Hadoop offers the so called Hadoop Streaming API, which allows you to write your MapReduce jobs in any language, which supports Unix standard input and output. A handy feature is, that you can test your job within the command line by using Unix pipes.
While the Hadoop Java API provides an iterator over each key-group, the user of Hadoop Streaming API has to find the key-group boundaries by him-/herself. But that is not that bad, because the MapReduce framework ensures, that the keys are ordered.



The following example will be created in Python. It will only show the basics, so that you are able to adapt this to any language of your choice.
As I mentioned before, we will use Unix standard input and output, which implies the mapper and reducer will receive their data via Unix standard input and write it to Unix standard output.
The mapper will receive the data ordered like it appears in the input file; the reducer will receive the data ordered by key.
This time, we will count how often “complete” categories appear within the dataset.

5

🛠️ Mapper

```
#!/usr/bin/env python
import sys

id = 0

for line in sys.stdin:
  val = line.strip()
  data = val.split(',')
  category = data[1]
  print(category + '\t' + str(id))
  id = id + 1
```

Handle stdin line by line

Handover data to Reducer

The mapper has to handle the given data line by line. We are using a csv-file, which uses comma as a separator.
For Java a unique line-identifier was already handed in by the MapReduce API, this time we have to create one for ourselves, if we want to use it.
To hand the output-data over to the reducer, we only have to print the results into Unix standard output.

5

🛠️ Reducer

```
previous_key = None
count = 0

for line in sys.stdin:
    (key, val) = line.strip().split('\t')
    if previous_key is None:
        previous_key = key
    if key == previous_key:
        count = count + 1
    else:
        print(key + '\t' + str(count))
        count = 1
    previous_key = key
```

Get key and value from line

Print result for key, if last

Remember key

The mapper hands over the data to the reducer by using stdout. The MapReduce framework ensures, that the keys are ordered, so that we can process them easily. The reducer will count the values per key, this time we do not have a list of values for one key, like in the Java example before; this time keys appear multiple times. The reducer has to fetch the standard output of the mapper, has to process it line by line and has to ensure, that same keys are handled in the same way. This implies, that the line before the line which is currently processed, has to be remembered.
Attention: I had to leave out
#!/usr/bin/env python
import sys
At the beginning of this script.

5

🛠️ Copy into Container

Copy Mapper into container

```
docker cp mapper.py ab0978def5ae:/tmp/
```

Copy Reducer into container

```
docker cp reducer.py ab0978def5ae:/tmp/
```

Make files executable

```
chmod +x mapper.py
chmod +x reducer.py
```


Now, that we have written mapper and reducer, the final step is to run the MapReduce-job with the help of Hadoop's Streaming API.
Of course the first step of running it successfully is, to copy the python files into the running Docker container. Additionally both Python-scripts have to be made executable.

5

🛠️ Run it - Base Command

```
$HADOOP_PREFIX/bin/hadoop jar \
  $HADOOP_PREFIX/share/hadoop/tools/lib/hadoop-streaming-2.7.0.jar
```


We will start a build in jar, which contains Hadoop’s Streaming API. This slide shows the beginning of the run command. On the next slide, you can find the arguments, that have to be passed to the command.

5

🛠️ Run it - Arguments

```
-files /tmp/mapper.py,/tmp/reducer.py \
-input input/darknet \
-output output/darknet/category-count \
-mapper /tmp/mapper.py \
-reducer /tmp/reducer.py
```

- Ships the listed files to the cluster
- Input-file for MapReduce-job
- Output-path for MapReduce-job
- Set Mapper
- Set Reducer

Now, that we have written mapper and reducer, the final step is to run the MapReduce-job with the help of Hadoop’s Streaming API.
Of course the first step of running it successfully is, to copy the python files into the running Docker container.

5

🛠️ CLI - Check

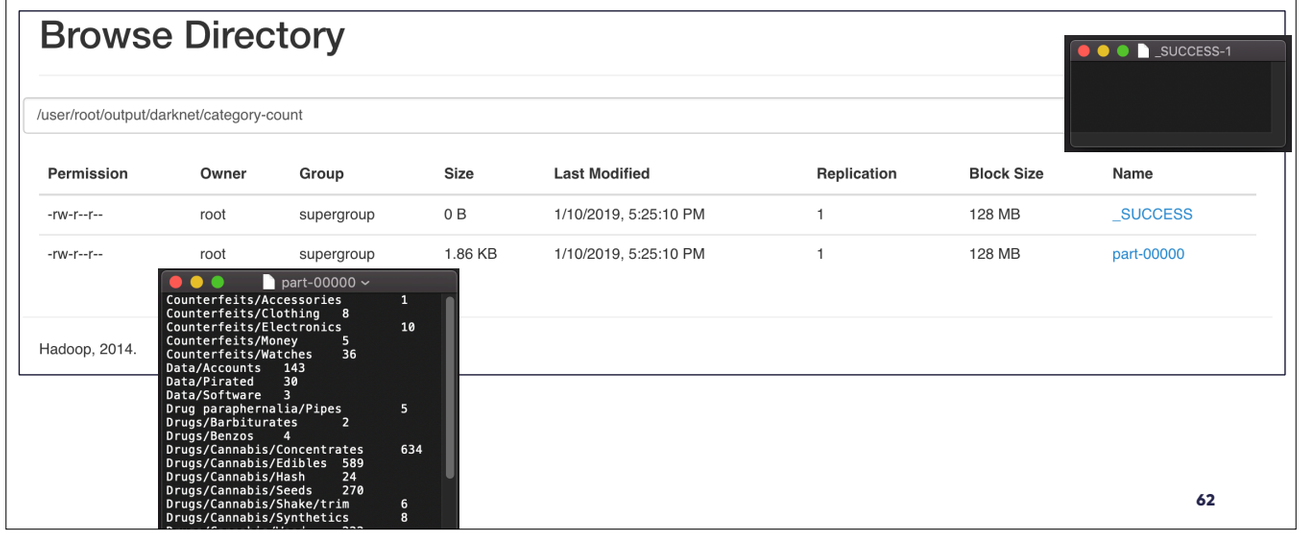
```
$HADOOP_PREFIX/bin/hdfs dfs -cat output/darknet/
category-count/*
```

```
Counterfeits/Accessories        1
Counterfeits/Clothing   8
Counterfeits/Electronics        10
Counterfeits/Money      5
Counterfeits/Watches    36
Data/Accounts   143
Data/Pirated    30
Data/Software   3
Drug paraphernalia/Pipes        5
Drugs/Barbiturates      2
Drugs/Benzos    4
Drugs/Cannabis/Concentrates     634
Drugs/Cannabis/Edibles  589
Drugs/Cannabis/Hash     24
```

Checking the command line interface shows the results of the MapReduce-job. As you may have already expected, there are plenty much more results, than before, when we counted the appearances of main-categories. Here is an excerpt of the complete list.

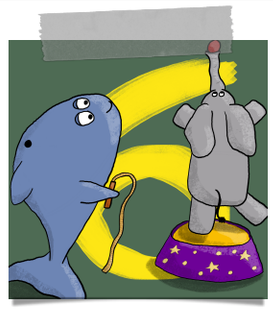
5

🛠️ Web UI - Check



When we check the results in Web UI, we can again identify the empty “_SUCCESS”-file and a file containing the results of our MapReduce-job.

Alright, now also the Java-Haters are happy because they know that they can write MapReduce-jobs, too.



What About the Rest?

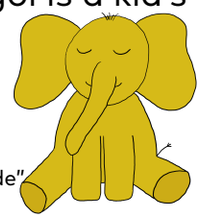
Part 6: Isn’t there more about Hadoop?
Of course there is plenty more information about Hadoop and the stuff belonging to the Hadoop ecosystem. But my focus is to show you an easy way to get your hands on Hadoop and do some practical stuff. But it would be a very strange presentation, if I would not include some more theory.
For all of you, who is not interested in some basics, just think about what we’ve achieved in the time until know, and just stare at the slides, until another one with pink background shows up, this will be the sign that the theory-part is over, and you can start listening again.

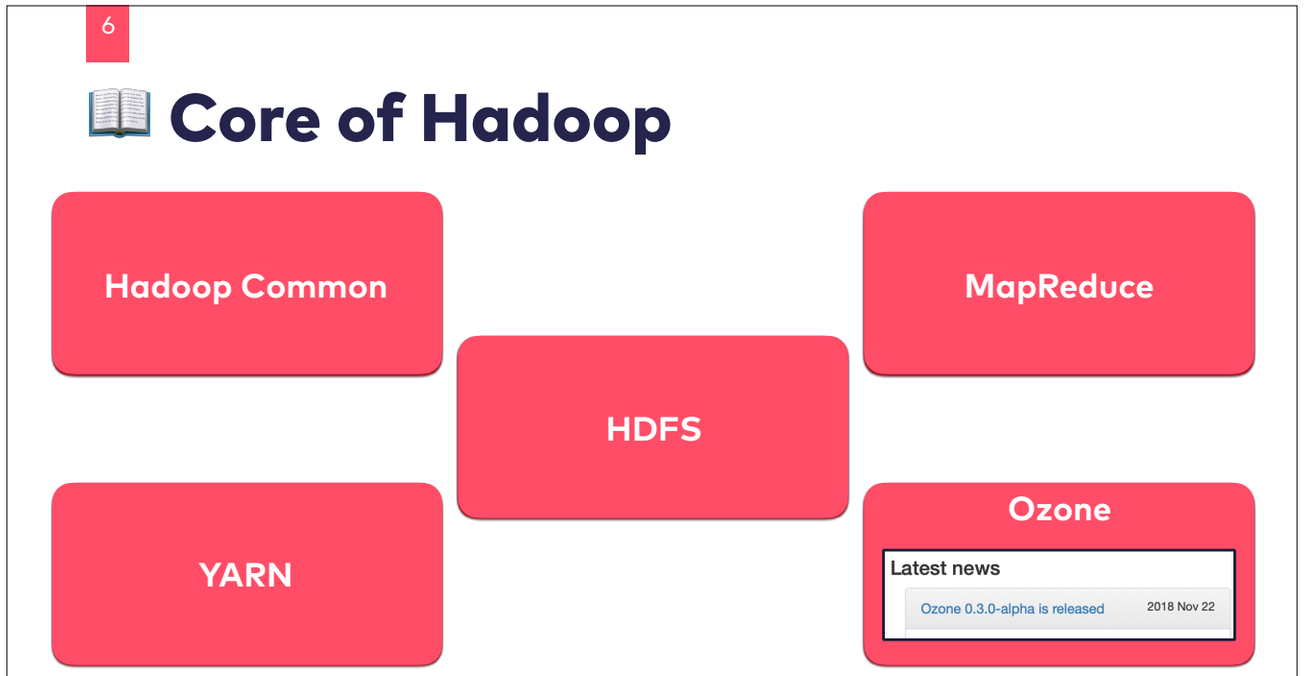
6

📖 Naming

"The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term."
(Doug Cutting)

Excerpt From: Tom White. "Hadoop: The Definitive Guide"



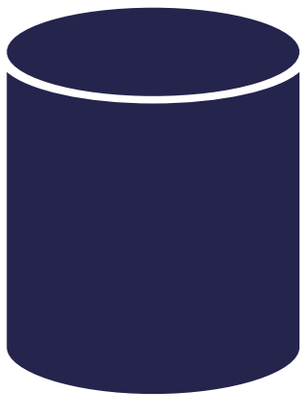


As I mentioned before, Hadoop's core consists of multiple parts, two of them where already mentioned.

- Hadoop Common contains the common application parts of Hadoop.
- HDFS is Hadoop's Distributed File System, which is used for file storage.
- YARN is a framework for job-scheduling and cluster resource management. The abbreviation YARN is short for "Yet Another Resource Negotiator"
- MapReduce allows the parallel processing of big data sets.
- Hadoop Ozone is a new part of Hadoop's core, it is used for object storage

6

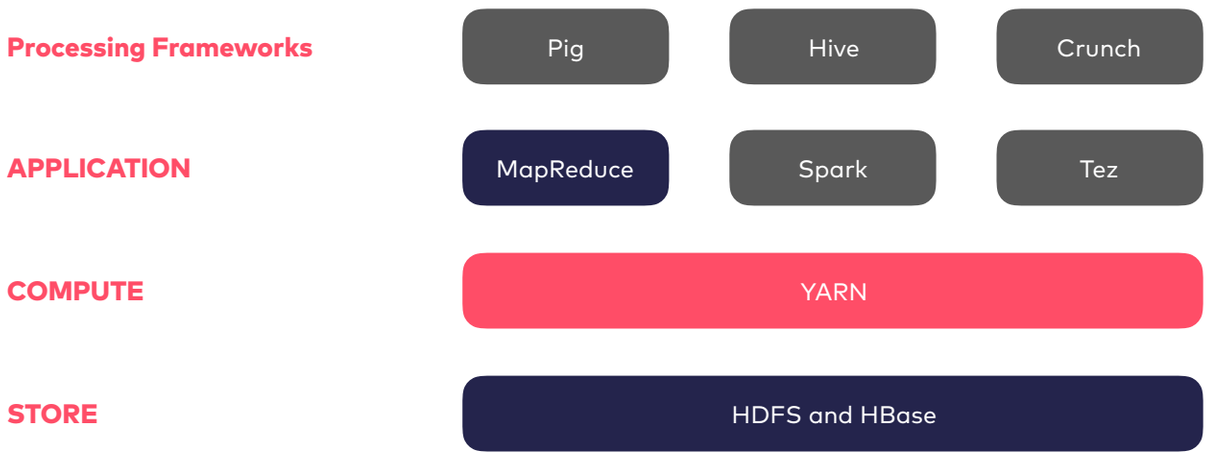
📖 HDFS



HDFS is Hadoop’s Distributed File System. As the name already tells, it is distributed, which we have not seen in our example.
HDFS is really good with very large files, but acts badly with a high amount of small files. Inside of HDFS two different node-kinds are working in a master-worker-pattern.
The “NameNode” keeps in memory metadata about the filesystem, while the “DataNodes” store the data itself. Files are stored in “Blocks” within the HDFS.

6

📖 YARN



Yet Another Resource Negotiator…

YARN, a cluster resource management system, was introduced in Hadoop 2 to improve the MapReduce implementation. MapReduce is an application, which directly interacts with YARN, other examples for applications, which interact with YARN are: Spark and Tez.

Besides those applications, also processing frameworks are existent, which do not interact directly with YARN, but with MapReduce, Spark or Tez, like: Pig, Hive and Crunch

Check, some theory was added to this presentation.

I Want To Try It!

Part 7: Try it!!!
Of course you are hyped and want to try everything at home, what I showed you in the presentation. This is one reason, why this presentation was created. I will provide you some links and book-tipps, that you can do something at home.

7

@ Home

- More Information About Hadoop
  - http://hadoop.apache.org
  - Tom White - “Hadoop - The Definitive Guide” (4th)
  - Alex Holmes - “Hadoop In Practice” (2nd)

- More Information Regarding This Talk
  - www.teapot418.de
  - https://github.com/Teapot-418/hadoop-taming-the-elephant

Here you find the ongoing information. Of course the blog-post-series on which this presentation is based. Two books, and yes, the Apache documentation, because it is useful, if you want to get deeper knowledge! You can also find my GitHub-repository for this presentation, which includes all code-examples and some additional stuff (like the manipulated dataset)

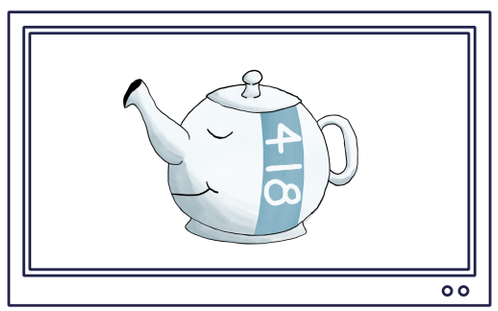
The End...

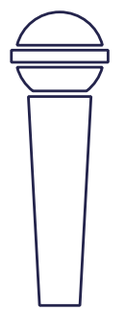
Part 8: The End…
Everything has an end. Except for sausages, they have to…

8

Share Discoveries

MY GOAL
Breaking the ice and giving a short introduction

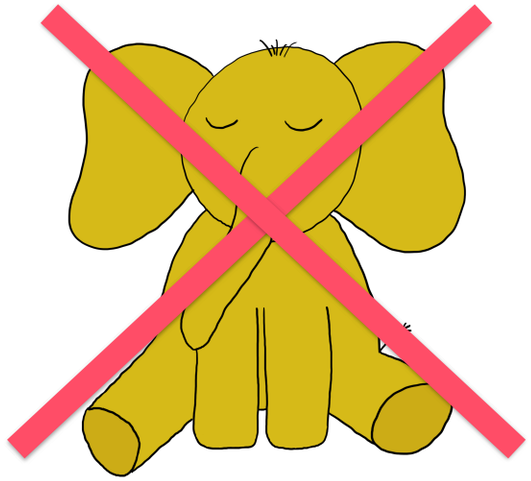




I wanted to share my findings with as many people as possible. Therefore I started to write a blog about my quick and dirty Hadoop-introduction.
This presentation has the same reason. I want to share my findings and I want you to play with Hadoop. I want to break the ice for people who are overwhelmed by this immense Hadoop universe. I want to motivate you, to play around with datasets at home.

8

What's Up Now?

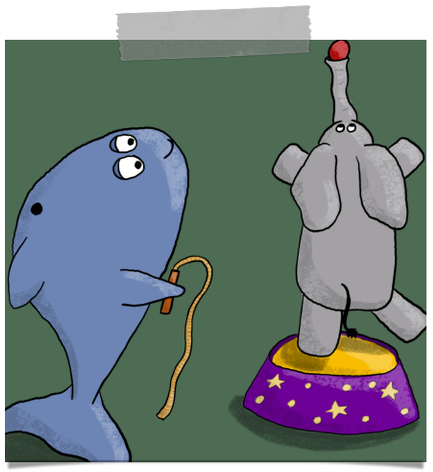


Some funny story at the end: Of course I ended up in a totally different project, which is also great. But instead of Big Data or Java and Spring, I work a lot with node.js and stuff like Pug and Postgres, so no Hadoop and Big Data for me at the moment.

Thank you!

Lisa Maria Moritz
lisa.moritz@innoq.com

Teapot4181



INNOQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany
+49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
+49 2173 3366-0

Kreuzstr. 16
80331 München
Germany
+49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 0116

I thank you very much for listening to this presentation. For this is the first time I ever did something like this, I would really appreciate some feedback via mail, Twitter, … .
THANK YOU!