

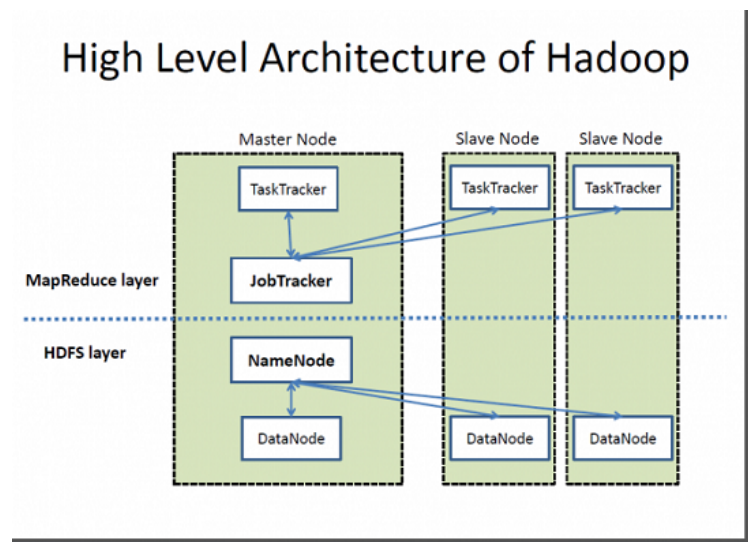
PTS Vaja 3

Za vajo 3 bomo uporabili naslednje tehnologije in orodja:

- Hadoop 3.2.1
- IntelliJ IDEA

1. Hadoop

Hadoop je odprtokodna platforma namenjena za obdelavo velikih množic podatkov v porazdeljenem okolju. Na trgu obstaja več ponudnikov Hadoop distribucij (npr. Apache, Cloudera, Hortonworks, ...). Najpogosteje se uporablja Apache Hadoop distribucija, ki predstavlja razvojni okvir znotraj Hadoop ekosistema različnih tehnologij in orodij za obdelavo podatkovnih tokov, datotek v datotečnem sistemu, podatkovnih baz, upravljanje gruč itd. Apache Hadoop je sestavljen iz štirih modulov: (1) HDFS, (2) Hadoop MapReduce, (3) YARN in (4) Hadoop Commons.



Slika 1: Osnovni prikaz Hadoop arhitekture ¹

Namestitev Hadoopa

Pred namestitvijo Hadoopa se je potrebno prepričati, da je v sistemu že nameščena **Java 8**, in sicer z izvedbo ukaza: *java- version*.

Konfiguracija Hadoop gruče

Obstajajo tri načina za izvedbo Hadoop gruče:

1. Samostojen (angl. standalone) - gruča se izvaja kot en Java proces,
2. **Psevdo-porazdeljen** (angl. pseudo-distributed) - gruča se izvaja na le enem fizičnem vozlišču, vendar se vsaki Hadoop daemon izvaja znotraj zasebnega Java procesa, in
3. Popolnoma porazdeljen (angl. distributed) - gruča se izvaja na različnih fizičnih strojih (vozliščih).

Preden zaženemo Hadoop gručo, moramo določiti osnovne lastnosti gručice in lokacijo za shrambo podatkov. Vsi nadaljnji koraki se izvajajo znotraj Hadoop korenske mape (`/opt/hadoop`). Za uspešno namestitev je izmed ostalih potrebno določiti lastnosti v naslednjih datotekah:

- `hadoop-env.sh` - pot do JAVA korenske mape;
- `core-site.xml` - lokacija map HDFS imenskega in podatkovnega vozlišč (angl. NameNode in DataNode) ter URL pot do HDFS-a;
- `hdfs-site.xml` - replikacijski faktor HDFS-a (privzeto 3, pri psevdo-porazdeljenem načinu bo 1), sistemske mape za shrambo podatkov na imenskem in podatkovnem vozlišču ter začasne shrambe podatkov v HDFS-u;
- `mapred-site.xml` - razvojni okvir YARN za izvedbo poslov v Hadoop gručici, standardna pot do razredov za MapReduce posle;
- `yarn-site.xml` - sistemske informacije za uspešno izvajanje poslov (seznam sistemskih spremenljivk).

Zdaj lahko zaženemo Hadoop gručo (**zaženemo HDFS in YARN skripte**). Znotraj `sbin` mape lahko samo izvedemo skripto `start-all.sh` ali posebej `start-dfs.sh` in `start-yarn.sh`. Če se je vse uspešno izvedlo, bi morali v brskalniku imeti dostop do HDFS (`localhost:9870`) ter YARN spletnih uporabniških vmesnikov (`localhost:8088`). Prebrskajte spletne strani, in poiščite, kje je vidna organizacija HDFS-a, tj. mape in dokumenti, ki so dostopni v HDFS-u.

Overview 'localhost:9000' (active)

Started:	Mon Nov 01 10:16:53 +0100 2021
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 17:56:00 +0200 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-87fd704c-f3dc-409b-a095-bfedb4ae3c4e
Block Pool ID:	BP-243984389-172.22.0.2-1635700693507

Summary

Security is off.
Safemode is off.


75 files and directories, 57 blocks (57 replicated blocks, 0 erasure coded block groups) = 132 total filesystem object(s).

Heap Memory used 108.44 MB of 174 MB Heap Memory. Max Heap Memory is 443 MB.

Non Heap Memory used 57.27 MB of 58.44 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	62.74 GB
Configured Remote Capacity:	0 B

Slika 2: Naslovna stran HDFS uporabniškega vmesnika.



Cluster

About Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total
0	0	0	0	0	0 B	0 B	0 B	0	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster App
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

Search

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	Reserved CPU V-Cores	Reserved Memory MB	% of Queue	% of Cluster	Progress
No data available in table																		

Showing 0 to 0 of 0 entries

Slika 3: Naslovna stran YARN uporabniškega vmesnika.

2. HDFS

HDFS (Hadoop Distributed File System) je datotečni sistem zasnovan za porazdeljeno shrambo podatkov, medtem pa omogoča visoko dostopnost sistema. Sestavljen je iz dveh komponent: podatkovnega vozlišča (DataNode) in imenskega vozlišča (NameNode). DataNode je odgovoren za upravljanje podatkovnih blokov (angl. data block). Privzeta replikacija posameznega bloka v različne DataNode komponente je 3, kar pomaga pri doseganju visoke dostopnosti. Pri replikaciji blokov, NameNode določa ID posameznega bloka in shranjuje metapodatek, medtem ko se pa sam blok posreduje DataNode-om za shrambo.

HDFS ukazi

Pomembno: Najprej dodamo *hadoop* ukaz v seznam sistemskih spremenljivk, kar lahko naredite z naslednjim ukazom (**potrebno izvesti ob vsakem zagonu Docker kontejnerja!**):

```
export PATH=$PATH:$HADOOP_HOME/bin
```

Vsi HDFS ukazi se začnejo z *hadoop fs* ali *hdfs dfs* (ta dva ukaza sta sinonima, če se kot porazdeljeni datotečni sistem uporablja HDFS). Ukazi so zelo podobni tistim, ki se uporabljajo v Unix operacijskih sistemih. Seznam osnovnih HDFS ukazov lahko najdete na <https://images.linuxide.com/hadoop-hdfs-commands-cheatsheet.pdf>.

Na začetku z uporabo *-ls* opcije lahko izpišemo seznam vseh map/datotek v sistemu. Izpiše se tudi faktor replikacije (angl. replication factor) za posamezni dokument (*hdfs dfs -ls*).

V HDFS bomo dodali datoteko *restaurant-orders.csv* s podatki o naročilih, ki jo lahko prevzamete z eŠtudija. Datoteko najprej moramo dodati iz gostiteljskega OS-a v Docker kontejner z Ubuntu OS-om. V terminalu se pozicioniramo v mapi, kjer se nahaja datoteka, in izvedemo ukaz:

```
docker cp restaurant-orders.csv pts-hbase:/etc
```

Zgornji ukaz bo premaknil datoteko v mapo */etc* znotraj Docker kontejnerja. Naslednji korak je še dodati tisto datoteko iz datotečnega sistema OS-a v HDFS. Za dodajanje dokumentov iz lokalnega sistema v HDFS se uporabljata opciji *-copyFromLocal* in *-put*, kjer je treba kot argument določiti pot do dokumenta na lokalnem sistemu in določene mape v HDFS-u.

Potem zaženete Hadoop gručo in dodate datoteko iz lokalne */etc* mape v korensko mapo v HDFS-u.

```
hdfs dfs -put /etc/restaurant-orders.csv /
```

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

/

Go!

Show

25

entries

Search:

<input type="checkbox"/>	<div><div></div></div> Permission	<div><div></div></div> Owner	<div><div></div></div> Group	<div><div></div></div> Size	<div><div></div></div> Last Modified	<div><div></div></div> Replication	<div><div></div></div> Block Size	<div><div></div></div> Name	<div><div></div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Oct 31 20:49	0	0 B	mysql-couriers	<div><div></div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Nov 01 10:21	0	0 B	mysql-offers	<div><div></div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Oct 31 20:48	0	0 B	mysql-users	<div><div></div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Oct 31 23:37	0	0 B	output	<div><div></div></div>
<input type="checkbox"/>	-rw-r--r--	root	supergroup	439.48 KB	Oct 31 22:25	1	128 MB	restaurant-orders.csv	<div><div></div></div>
<input type="checkbox"/>	drwx-----	root	supergroup	0 B	Oct 31 20:39	0	0 B	tmp	<div><div></div></div>

Showing 1 to 6 of 6 entries

Previous

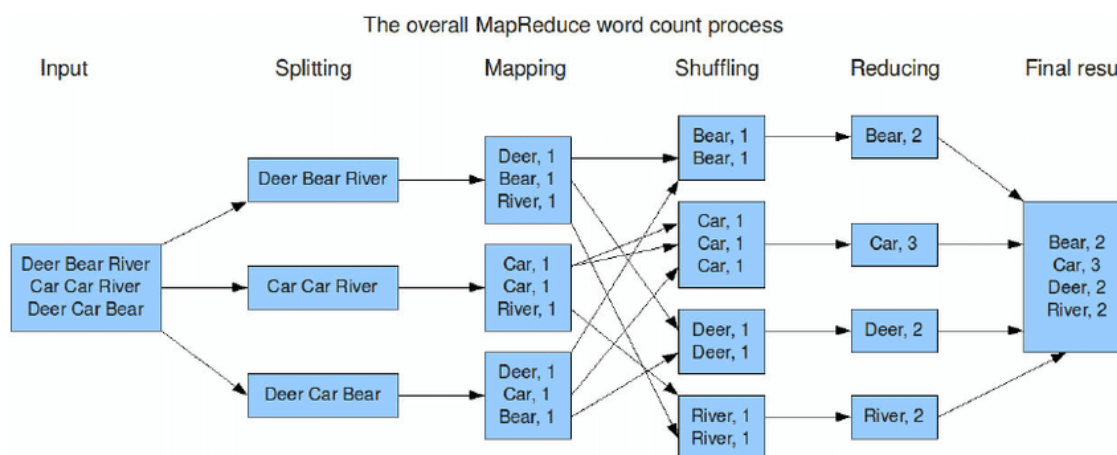
1

Next

Slika 4: Uspešno prenesena datoteka iz lokalnega sistema v HDFS.

3. MapReduce

MapReduce razvojno ogrodje je predstavljeno v verziji Hadoop 2. Omogoča vzporedno obdelavo velikih množic podatkov in izvajanje programov. Ogrodje je sestavljeno iz dveh funkcij (nalog): **Map** in **Reduce**. Map funkcija je odgovorna za analizo informacij iz podatkov, uporabo določenega algoritma, in generiranje rezultata v obliki parov ključ-vrednost. Dobljeni rezultat se potem posreduje v Reduce funkcijo, ki združi informacije na podlagi rezultata Map funkcije. Na sliki so prikazani procesi vključeni v izvajanje MapReduce poslova: Sledilec poslova (angl. Job Tracker) - master in Sledilec naloga (angl. Task Tracker) - slave.



Slika 5: Prikaz izvajanja WordCount primera kot MapReduce posla.

Na vajah bomo MapReduce (MR) posle pisali v Java programskem jeziku in IntelliJ Idea okolju. Java program bo vseboval tri dela: glavni del ter kodo Mapper in Reducer funkcij. Glavni del vsebuje glavno

metodo in parametre nujne za izvajanje MR posla. Funkcija Map vsebuje algoritem za obdelavo vhodne množice podatkov, medtem ko pa funkcija Reduce vsebuje algoritem za združitev rezultatov funkcije Map.

Cilj našega prvega MR posla je obdelati dokument "restaurant-orders.csv", ki vsebuje podatke o naročilih strank v restavraciji, in sicer na način da se na koncu za posamezne izdelke izpiše vsota njihove naročene količine.

Če pogledamo strukturo dokumenta "restaurant-orders.csv" (opomnik: ta dokument smo v prejšnjem koraku dodali v HDFS), lahko vidimo, da vsebuje naslednje stolpce (lastnosti): *Order Number*, *Order Date*, *Item Name*, *Quantity*, *Product Price in Total products* (opomba: datoteka nima glave). V rezultatu MR posla želimo izpisati le *Item Name* in vsoto naročene količine za tisti izdelek.

V IntelliJ-u ustvarimo novi projekt Maven, kjer bomo dodali kodo za tri razrede znotraj *um.si* paketa. Dodatno je potrebno pod *File > Project Structure > Project Settings > Artifacts > Jar > from modules with dependencies...* določiti, da bomo zgradili JAR datoteko, ko zaženemo projekt.

Mapper razred bo potem vseboval naslednje:

```
package um.si;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class OrderMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
        String row = value.toString();

        String itemName = row.split(",")[2];
        int quantity = Integer.valueOf(row.split(",")[3]);

        context.write(new Text(itemName), new IntWritable(quantity));
    }
}
```

Reducer razred pa vsebuje:

```
package um.si;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class OrderReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
```

```

protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {

    int sum = 0;
    for(IntWritable val: values){
        sum += val.get();
    }
    context.write(key, new IntWritable(sum));
}
}

```

Zdaj je potrebno registrirati ta dva razreda v glavnem delu (razredu):

```

package um.si;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class OrderProcessing {
    public static void main(String[] args) throws IOException, ClassNotFoundException,
        InterruptedException {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration, "CalculateOrderedItemQuantity");

        job.setJarByClass(OrderProcessing.class);
        job.setMapperClass(OrderMapper.class);
        job.setReducerClass(OrderReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

Ko "buildamo" projekt, bomo pod korensko mapo projekta v lokalnem sistemu v mapi /out dobili JAR datoteko *MapReduce.jar*, ki jo premestimo v Docker container u ukazom: *docker cp MapReduce.jar pts-hbase:/etc*.

Ko smo to uredili, lahko zaženemo naš MapReduce posel z naslednjim ukazom:

```
$HADOOP_HOME/bin/hadoop jar /etc/MapReduce.jar
```

```
/restaurant-orders.csv /output
```

Opomba: Mape "output" ni potrebno kreirati preden zaženemo MR posel, YARN jo ustvari samodejno - lahko pa določimo in uporabimo to mapo samo za en posel oz. samo enkrat zaženemo MR posel. Drugič bomo dobili napako, da ta mapa že obstaja v sistemu. Če dobimo to napako, lahko določimo drugi naziv mape, ali pa obstoječo mapo izbrišemo z ukazom:

```
hdfs dfs -rm -r /output
```

Možno je tudi, da bi dobili napako, da NameNode ne more pristopiti določenemu podatkovnemu bloku. V tem primeru to pomeni, da je množica podatkov večja kot prostorna velikost blokov na enem vozlišču. Napako razrešite tako, da zmanjšate število podatkov.

Če se vse uspešno izvede, lahko znotraj YARN uporabniškega vmesnika v seznamu vidite zagnani MR posel kot izvedeno aplikacijo s statusom "SUCCEEDED". Rezultate MR posla pa lahko v berljivi obliki izpišete z:

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

```
root@f02916d268b0:/# hdfs dfs -cat /output/*
2021-11-01 09:23:07,703 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2021-11-01 09:23:08,259 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
Aloo Chaat      5
Aloo Gobi       6
Baingan Hari Mirch      1
Bhindi Bhajee   2
Bhuna - Chicken 1
Bhuna - Chicken Tikka  1
Bhuna - Lamb    2
Bombay Aloo     13
Bottle Coke     1
Bottle Diet Coke      2
Butter Chicken  2
COBRA ( LARGE ) 3
Cauliflower Bhajee    2
Chana Masala     1
Chapati 33
Chicken Balti   3
Chicken Biryani 6
Chicken Chilli Garlic  2
Chicken Mysore  1
Chicken Pakora  3
Chicken Shashlick      2
Chicken Tikka   1
Chicken Tikka (Main)   5
Chicken Tikka Biryani  4
Chicken Tikka Chilli Masala    1
Chicken Tikka Jalfrezi  5
Chicken Tikka Karahi    2
Chicken Tikka Masala    13
Curry - Chicken 3
Curry - Lamb    1
Curry Sauce     1
Dhansak - Chicken      2
Dhansak - Lamb    1
Dhansak Sauce     1
```

Slika 6: Rezultat uspešno izvedenega Map Reduce posla.

Celotna koda MapReduce posla je dostopna na <https://github.com/msestak2/MapReduceExample/>.

Ko ste končali, ustavite vse komponente gruče, ki ste jih uporabljali (HDFS, YARN).