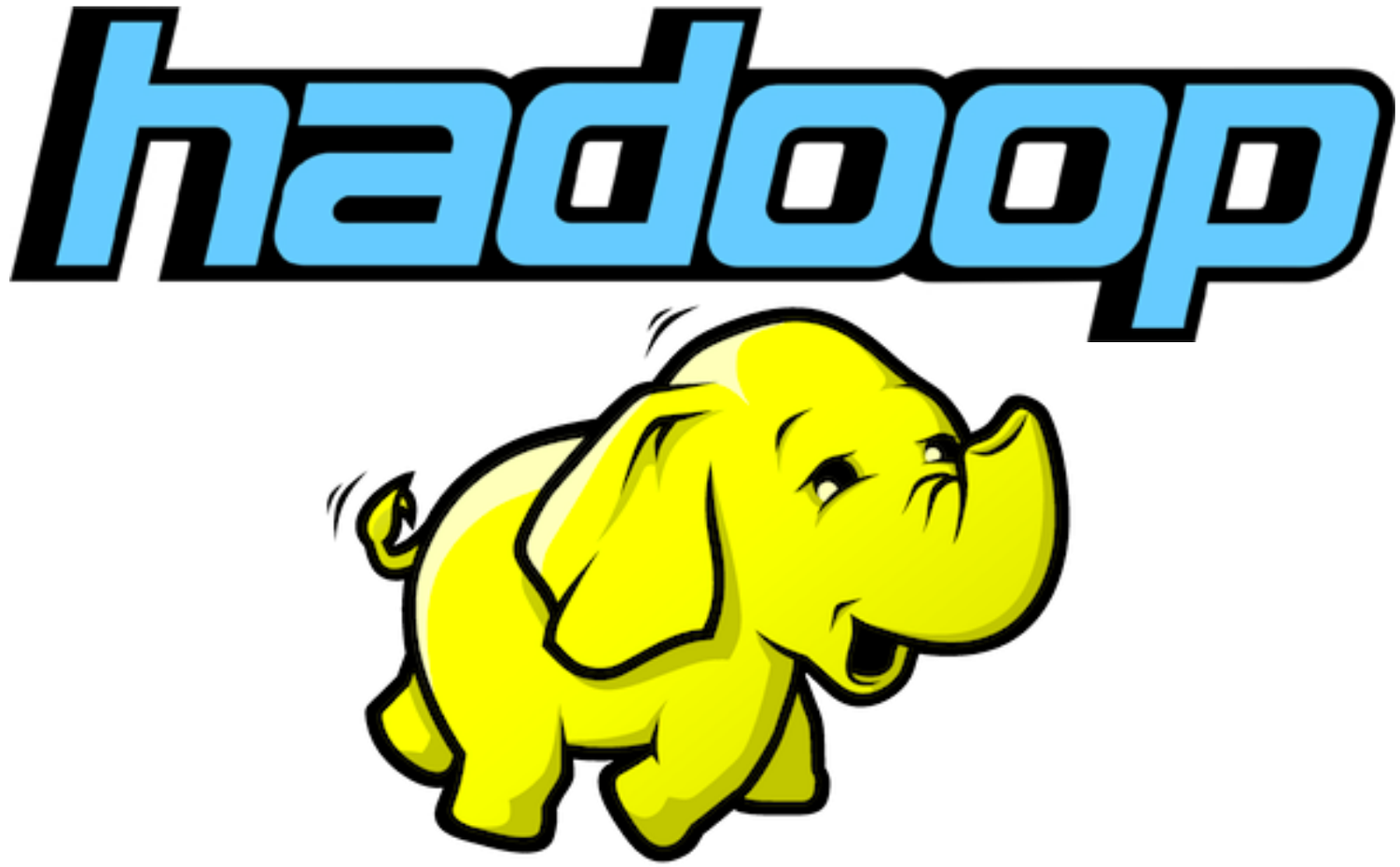


Tutorial

Christopher M. Judd

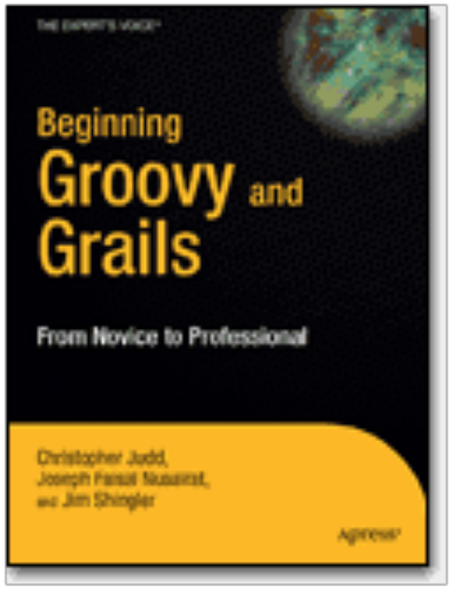
Manifest Solutions

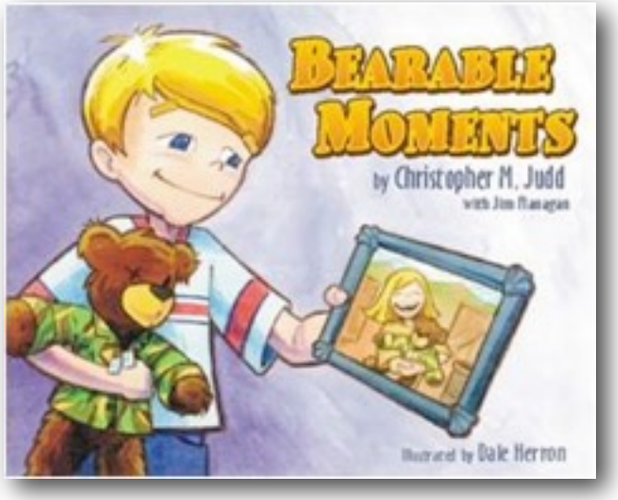
Christopher M. Judd

CTO and Partner at Manifest Solutions

Central Ohio Java Users Group leader

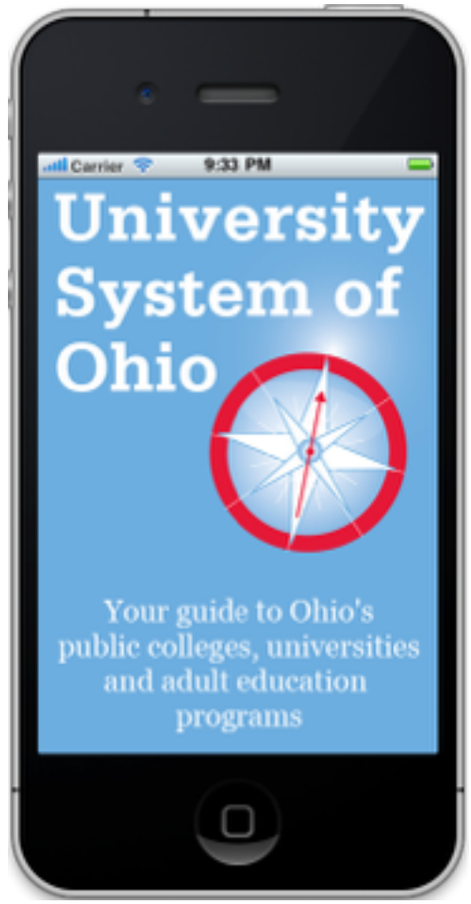
Columbus iPhone Developer User Group (CIDUG)





Carrier
9:32 PM
Remarkable
Ohio
Ohio | eTech Ohio
SITE OF OHIO'S FIRST STATEHOUSE

More
THE TABERNACLE
Pillar of Fire
"For the cloud of the LORD [was] upon the tabernacle by day, and fire was on it by night, in the sight of all the house of Israel, throughout all their journeys."
(Exodus 40:38)
Next


Carrier
9:33 PM
University System of Ohio
Your guide to Ohio's public colleges, universities and adult education programs

Carrier
12:20 PM
GERMAN VILLAGE SOCIETY
Welcome to historic German Village!
Shop
Dine
Stroll
Stay
Events
Donate

AT&T
6:00 PM
Pick a Subject
General Facts
Psychology & Philosophy
Religion
Civilization
History
Geography
Social Sciences
Political Science
Law
Education
Music

AT&T
10:09 AM
Ohio
PocketBallot

Carrier
2:21 PM
imSafe
mobile security worldwide

Introduction

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

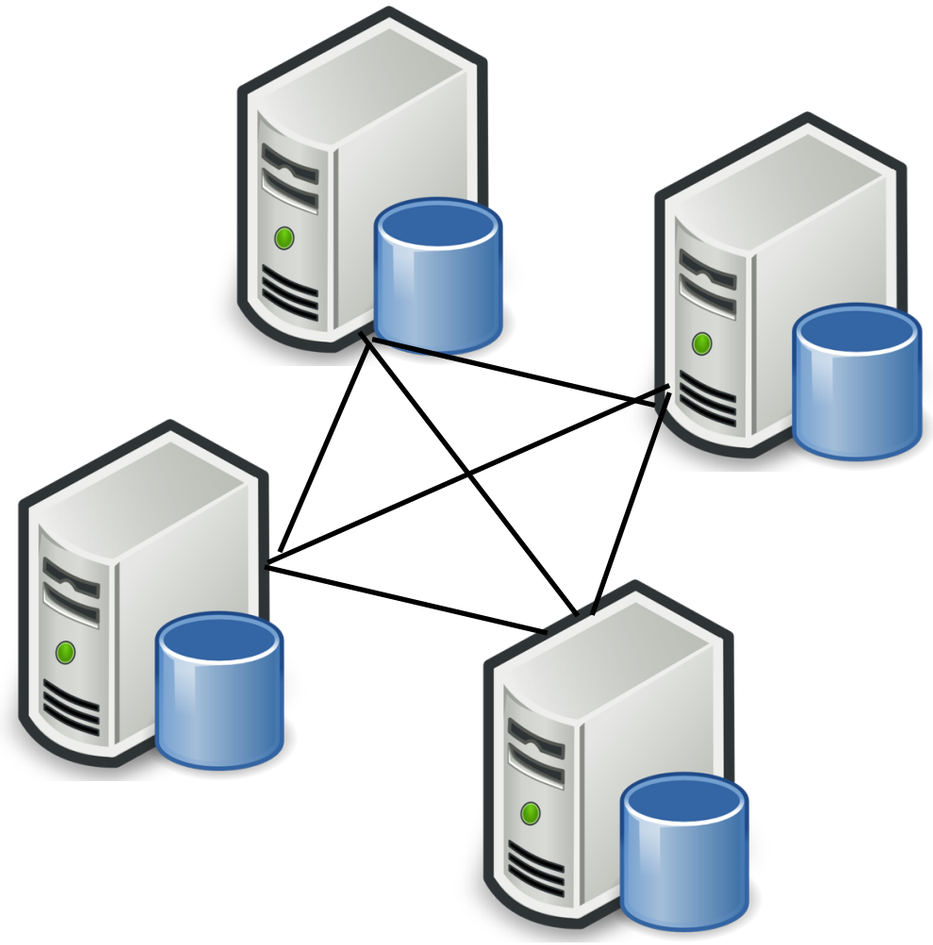
- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

http://hadoop.apache.org/

Scale up

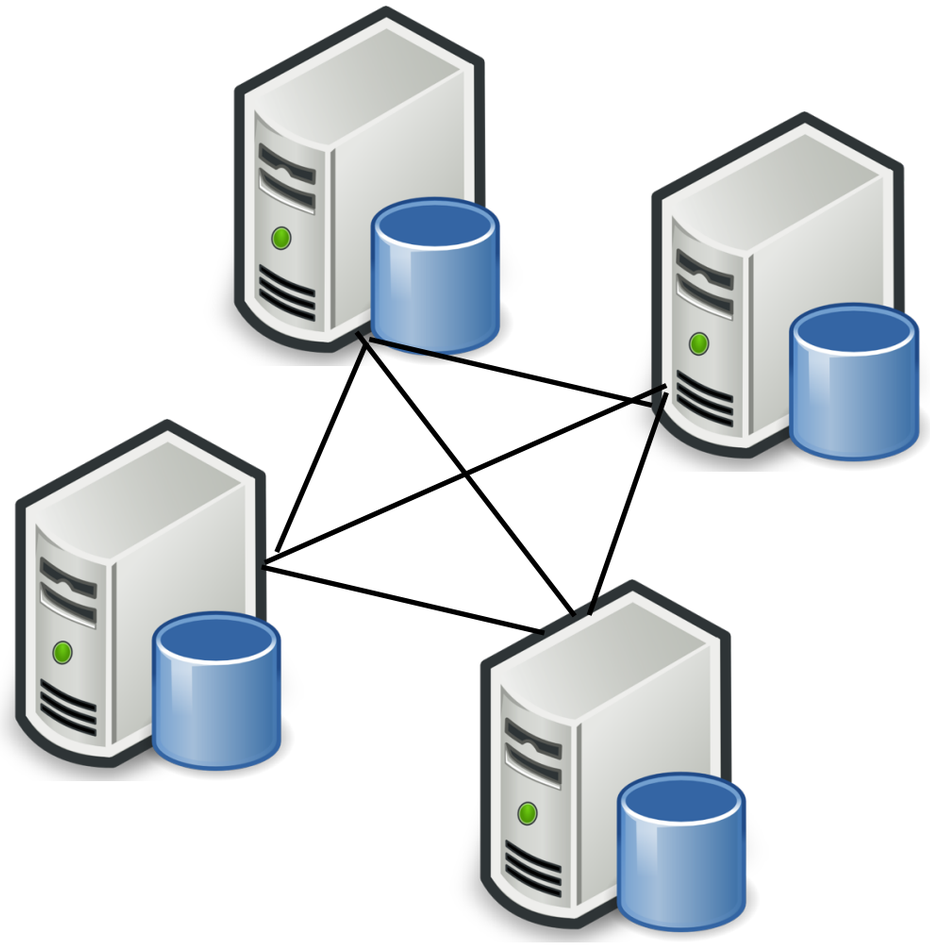
Scale up

Scale up



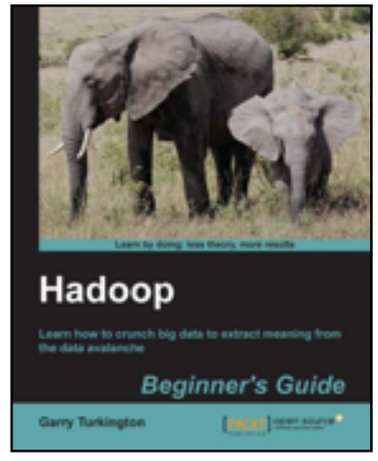
Scale up

Scale out



Hadoop Approach

- scale-out
- share nothing
- expect failure
- smart software, dumb hardware
- move processing, not data
- build applications, not infrastructure



What is Hadoop good for?

Don't use Hadoop - your data isn't that big

<100 mb - Excel
100 mb > 10 gb - Add memory and use Pandas
100 gb > 1 TB - Buy big hard drive and use Postgres
> 5 TB - life sucks consider Hadoop

http://www.chrisstucchio.com/blog/2013/hadoop_hatred.html

if your data fits in RAM

it is not Big Data

Hadoop is an evolving project

Hadoop is an evolving project

old api

org.apache.hadoop.mapred

new api

org.apache.hadoop.mapreduce

Hadoop is an evolving project

MapReduce 1

Classic MapReduce

MapReduce 2

YARN

Hadoop is an evolving project

Hadoop v1

MapReduce
(batch parallel computing)

HDFS
(distributed storage)

Hadoop v2

MapReduce V2
(batch parallel computing)

Spark
(in-memory processing)

Hoya
(HBase)

…

YARN
(resource management)

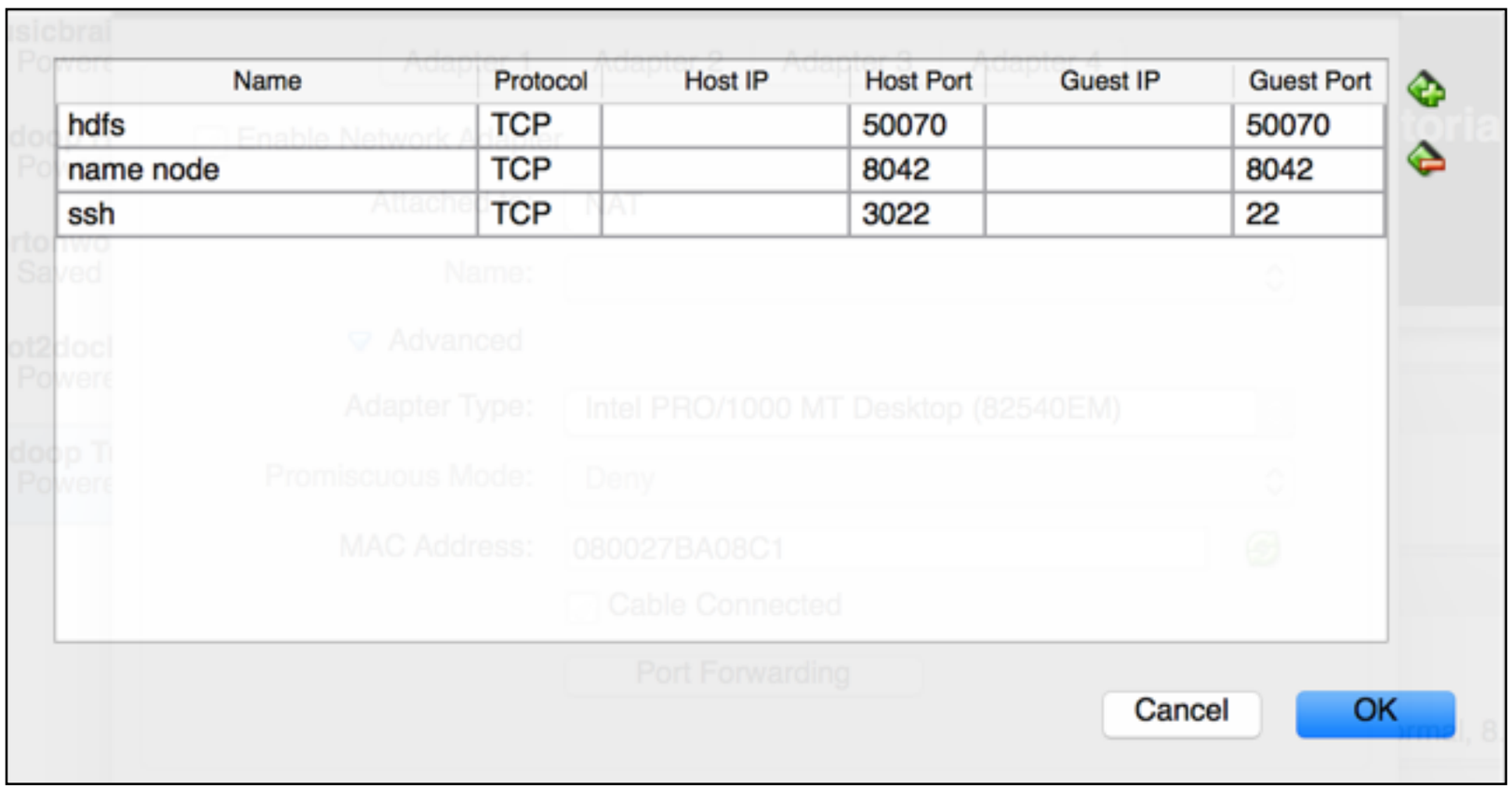
HDFS
(distributed storage)

Setup

Hadoop Tutorial

user/fun4all
hduser/hduser
/opt/data

Configure SSH



add port forwarding rule

SSH'ing

```
$ ssh -p 3022 hduser@localhost
user@127.0.0.1's password:
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.8.0-29-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Wed Jun 25 22:53:10 2014 from 10.0.2.2
user@user-VirtualBox:~$
```

HADOOP

http://hadoop.apache.org/

http://www.cloudera.com/

http://hortonworks.com/

http://www.mapr.com/

Amazon
Elastic
MapReduce

Windows Azure

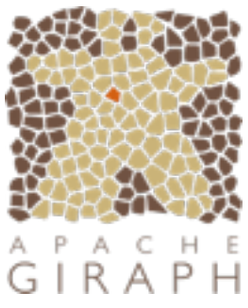
accumulo™

APACHE HBASE

KNOX

Spark

Thrift



HCatalog

Tez

Zebra

Owl

Add Hadoop User and Group

```
$ sudo addgroup hadoop
$ sudo adduser --ingroup hadoop hduser
$ sudo adduser hduser sudo

$ su hduser
$ cd ~
```

Install Hadoop

```
$ sudo mkdir -p /opt/hadoop
$ sudo tar vxzf /opt/data/hadoop-2.2.0.tar.gz -C /opt/hadoop
$ sudo chown -R hduser:hadoop /opt/hadoop/hadoop-2.2.0
$ vim .bashrc
```

```
# other stuff
# java variables
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

# hadoop variables
export HADOOP_HOME=/opt/hadoop/hadoop-2.2.0
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
```

```
$ source .bashrc
$ hadoop version
```

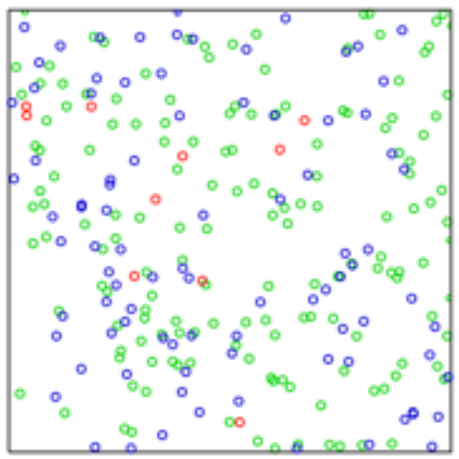
Run Hadoop Job

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 4 1000
```

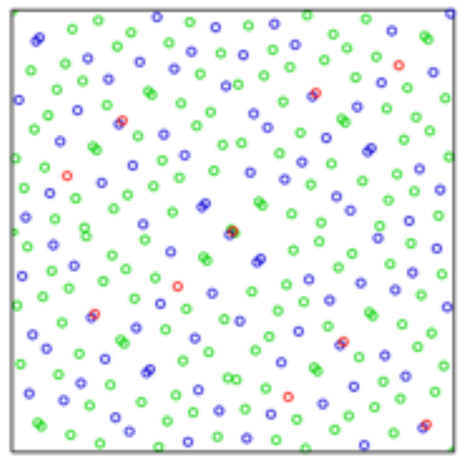
Quasi-Monte Carlo method

From Wikipedia, the free encyclopedia

In numerical analysis, **quasi-Monte Carlo method** is a method for numerical integration and solving some other problems using low-discrepancy sequences (also called quasi-random sequences or sub-random sequences). This is in contrast to the regular Monte Carlo method or Monte Carlo integration, which are based on sequences of pseudorandom numbers.



[Pseudorandom sequence]



[Low-discrepancy sequence (Sobol sequence)]

256 points from a pseudorandom number source, Halton sequence, and Sobol sequence (red=1,..,10, blue=11,..,100, green=101,..,256). Points from Sobol sequence are more evenly distributed.

Monte Carlo and quasi-Monte Carlo methods are stated in a similar way. The problem is to approximate the integral of a function f as the average of the function evaluated at a set of points $x_1, \ldots, x_N$:

$$\int_{[0,1]^s} f(u)\,\mathrm{d}u \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i).$$

Since we are integrating over the s-dimensional unit cube, each x_i is a vector of s elements. The difference between quasi-Monte Carlo and Monte Carlo is the way the x_i are chosen. Quasi-Monte Carlo uses a low-discrepancy sequence such as the Halton sequence, the Sobol sequence, or the Faure sequence, whereas Monte Carlo uses a pseudorandom sequence. The advantage of using low-discrepancy sequences is a faster rate of convergence. Quasi-Monte Carlo has a rate of convergence close to $O(1/N)$, whereas the rate for the Monte Carlo method is $O(N^{-0.5})$.[1]

The Quasi-Monte Carlo method recently became popular in the area of mathematical finance or computational finance.[1] In these areas, high-dimensional numerical integrals, where the integral should be evaluated within a threshold ε, occur frequently. Hence, the Monte Carlo method and the quasi-Monte Carlo method are beneficial in these situations.

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
```

aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
dbcount: An example job that count the pageview counts from a database.
distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
grep: A map/reduce program that counts the matches of a regex in the input.
join: A job that effects a join over sorted, equally partitioned datasets
multifilewc: A job that counts words from several files.
pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
randomwriter: A map/reduce program that writes 10GB of random data per node.
secondarysort: An example defining a secondary sort to the reduce.
sort: A map/reduce program that sorts the data written by the random writer.
sudoku: A sudoku solver.
teragen: Generate data for the terasort
terasort: Run the terasort
teravalidate: Checking results of terasort
wordcount: A map/reduce program that counts the words in the input files.
wordmean: A map/reduce program that counts the average length of the words in the input files.
wordmedian: A map/reduce program that counts the median length of the words in the input files.
wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi
Usage: org.apache.hadoop.examples.QuasiMonteCarlo <nMaps> <nSamples>
```

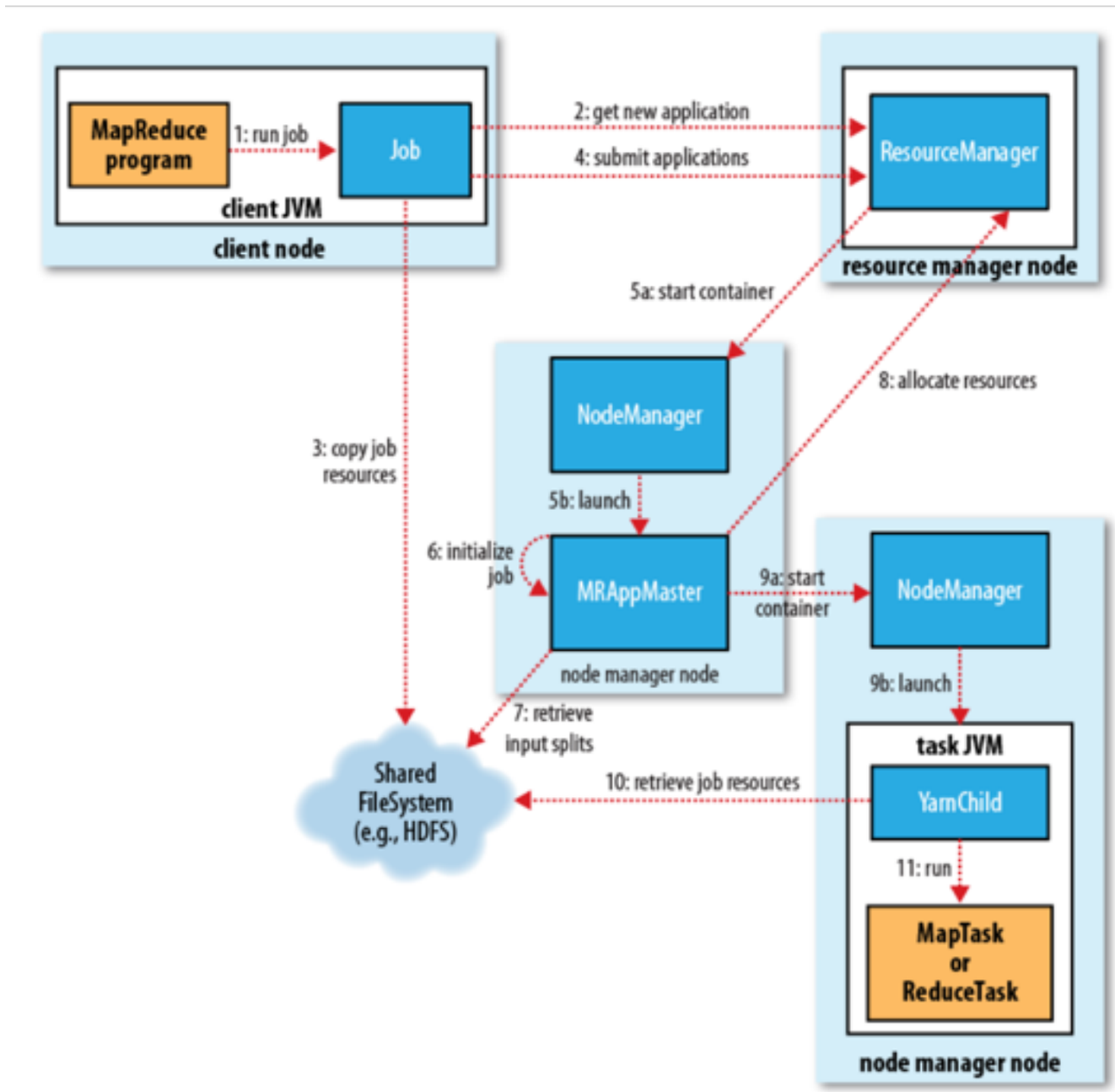
Lab 1

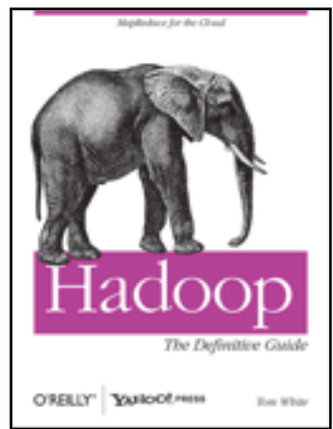
1. ~~Create Hadoop user and group~~
2. ~~Install Hadoop~~
3. Login as hduser/hduser
4. Run example Hadoop job such as pi

- Local Standalone mode
- Pseudo-distributed mode
- Fully distributed mode

HADOOP

PSEUDO-DISTRIBUTED


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit applications
ResourceManager
resource manager node
5a: start container
NodeManager
3: copy job resources
5b: launch
8: allocate resources
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
7: retrieve input splits
Shared FileSystem (e.g., HDFS)
10: retrieve job resources
node manager node


Hadoop
The Definitive Guide
O'REILLY
YAHOO! PRESS

Configure YARN

```
$ sudo vim etc/hadoop/yarn-site.xml
```

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>

</configuration>
```

Configure Map Reduce

```
$ sudo mv etc/hadoop/mapred-site.xml.template etc/hadoop/mapred-site.xml
$ sudo vim etc/hadoop/mapred-site.xml
```

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

Configure passwordless login

```
$ ssh-keygen -t rsa -P ''
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ ssh localhost
$ exit
```

Configure JAVA_HOME

```
$ vim etc/hadoop/hadoop-env.sh
```

```
# other stuff

export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

# more stuff
```

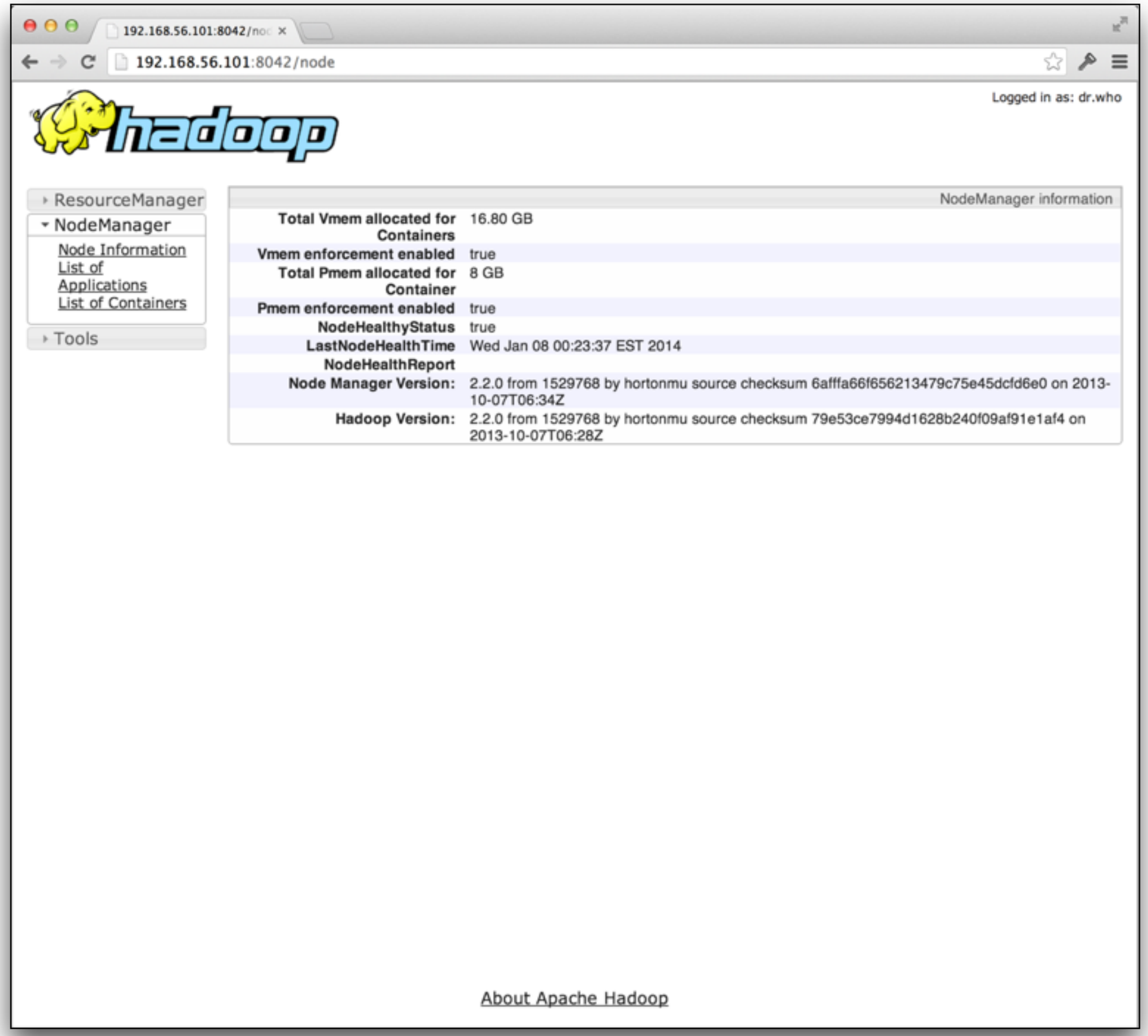
Start YARN

```
$ start-yarn.sh
```

```
$ jps
8355 Jps
8318 NodeManager
8090 ResourceManager
```

Run Hadoop Job

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 4 1000
```

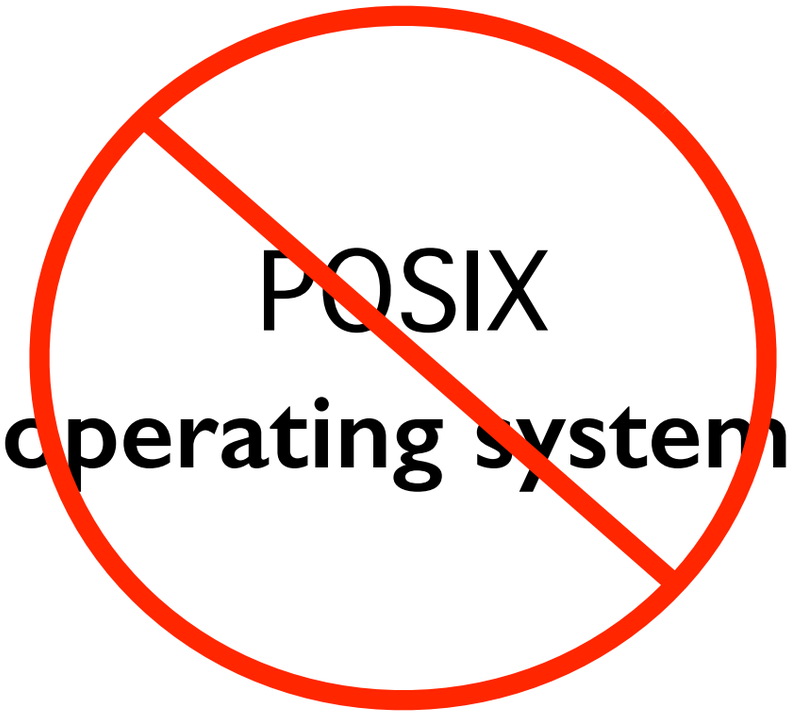


http://localhost:8042/node

Lab 2

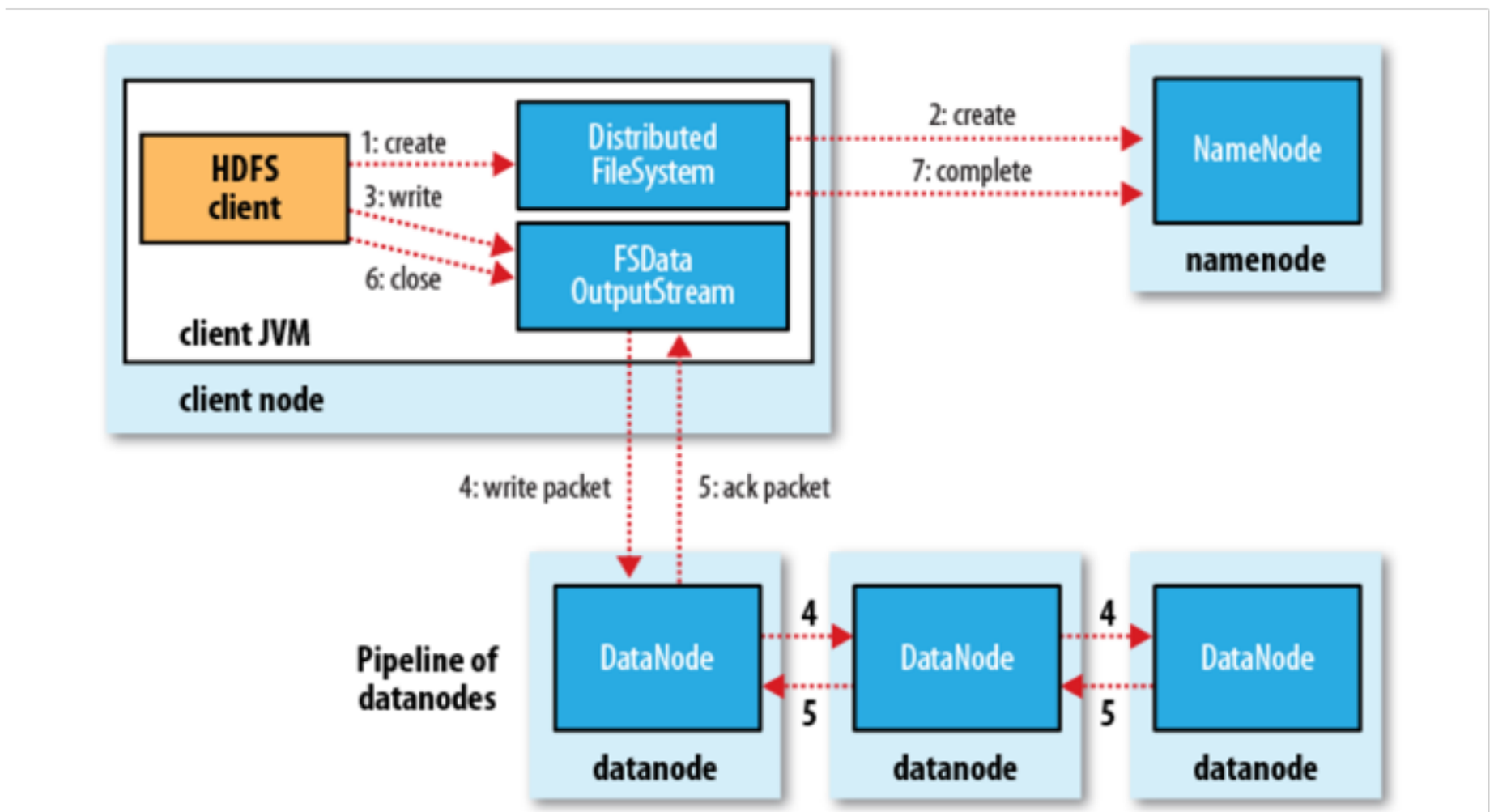
1. ~~Configure YARN~~
2. Configure Map Reduce
3. ~~Configure passwordless login~~
4. ~~Configure JAVA_HOME~~
5. Start YARN
6. Run pi job

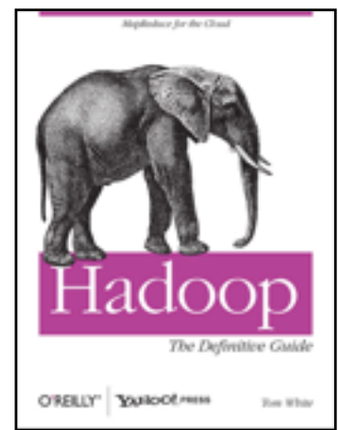
HDFS



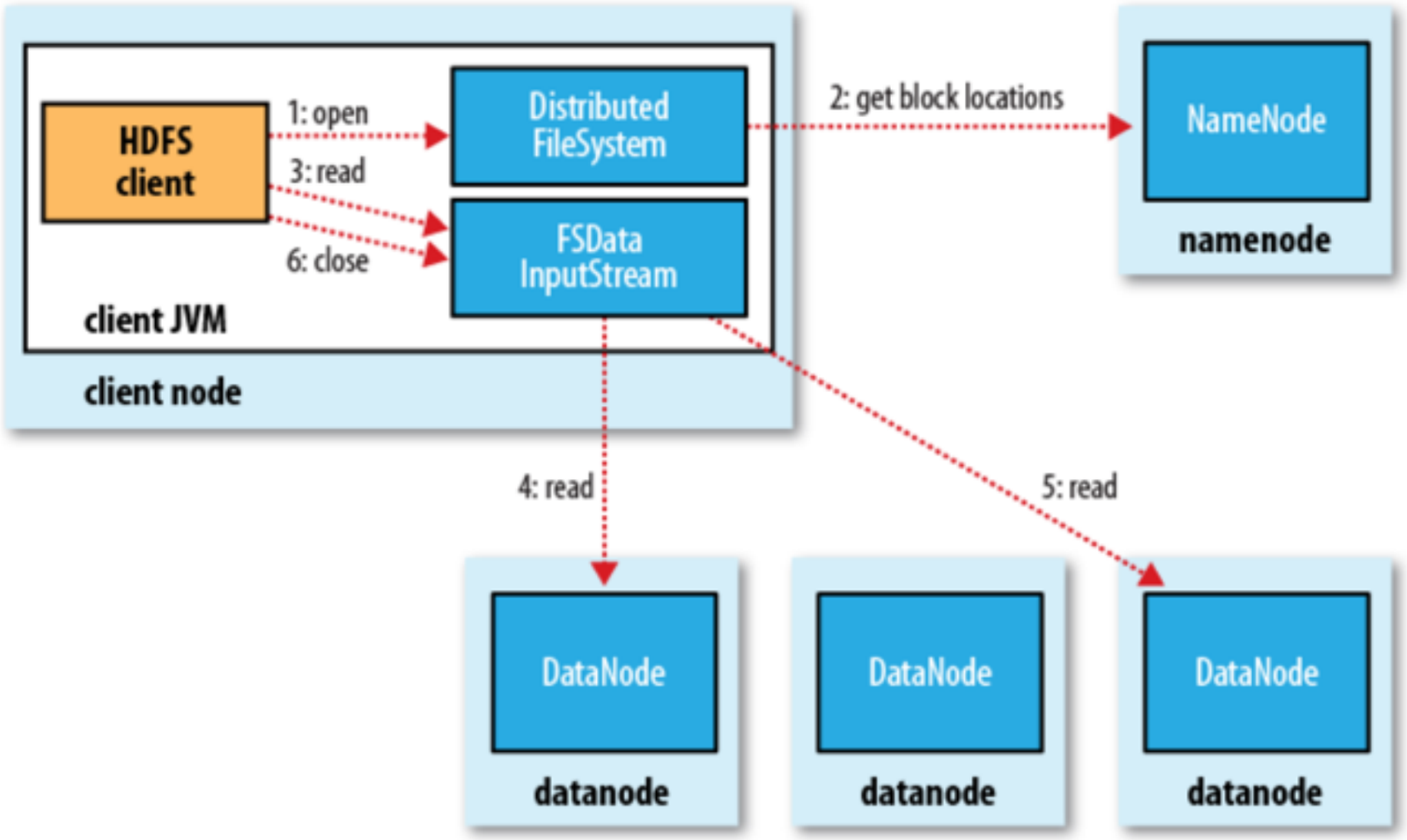
portable operating system interface

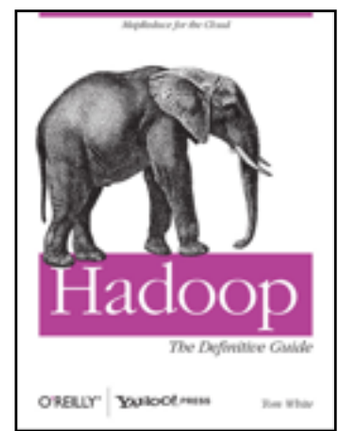
Writing Data





Reading Data





Configure HDFS

```
$ sudo mkdir -p /opt/hdfs/namenode
$ sudo mkdir -p /opt/hdfs/datanode
$ sudo chmod -R 777 /opt/hdfs
$ sudo chown -R hduser:hadoop /opt/hdfs
$ cd /opt/hadoop/hadoop-2.2.0
$ sudo vim etc/hadoop/hdfs-site.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/opt/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/opt/hdfs/datanode</value>
  </property>
</configuration>
```

Format HDFS

```
$ hdfs namenode -format
```

Configure Core

```
$ sudo vim etc/hadoop/core-site.xml
```

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Start HDFS

```
$ start-dfs.sh
```

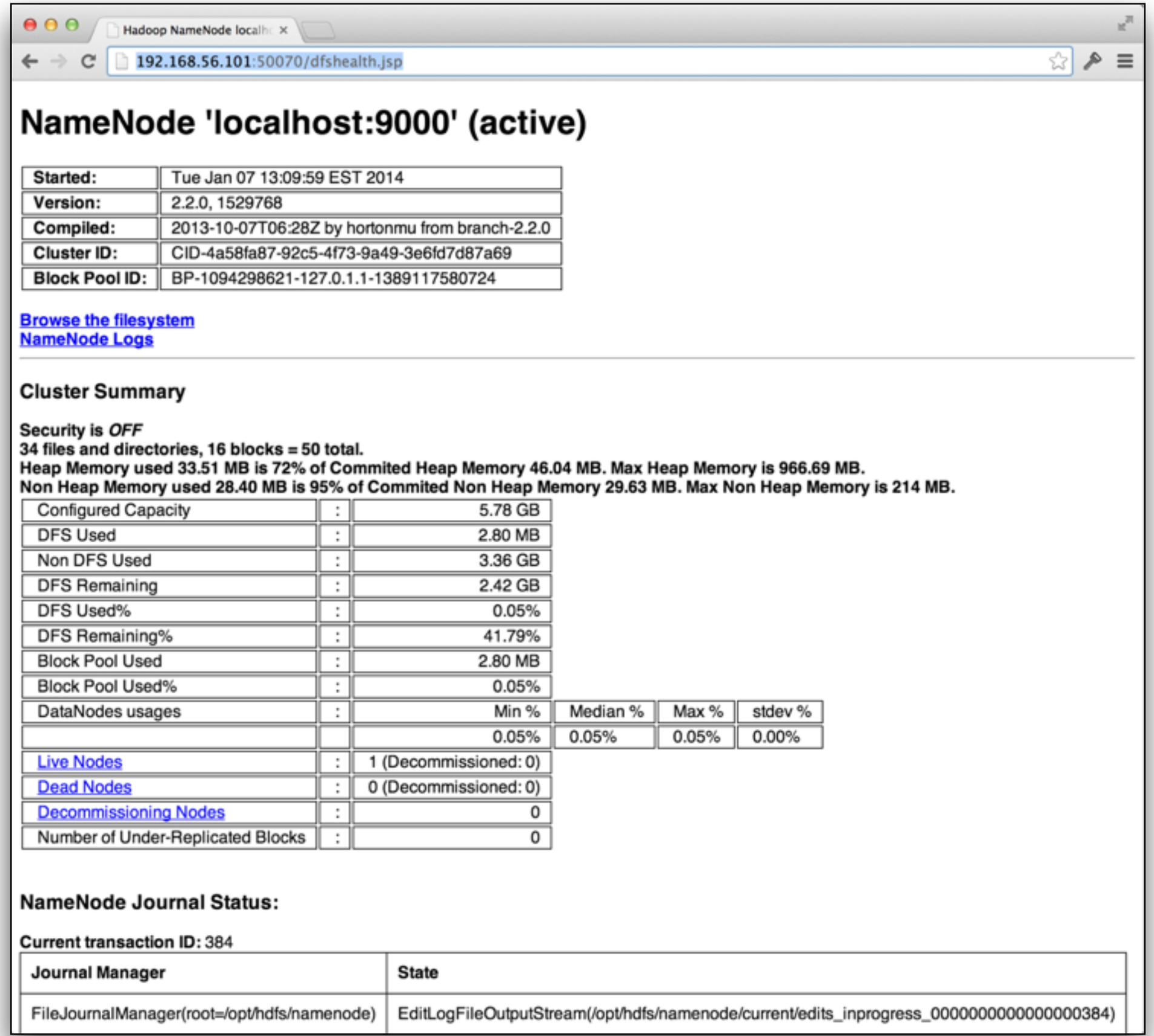
```
$ jps
6433 DataNode
6844 Jps
6206 NameNode
6714 SecondaryNameNode
```

Use HDFS commands

```
$ hdfs dfs -ls /
$ hdfs dfs -mkdir /books
$ hdfs dfs -ls /
$ hdfs dfs -ls /books
$ hdfs dfs -copyFromLocal /opt/data/moby_dick.txt /books
$ hdfs dfs -cat /books/moby_dick.txt
```

- appendToFile
- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- count
- cp
- du
- get
- ls
- lsr
- mkdir
- moveFromLocal
- moveToLocal
- mv
- put
- rm
- rmr
- stat
- tail
- test
- text
- touchz

http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-common/FileSystemShell.html



http://localhost:50070/dfshealth.jsp

Lab 3

1. ~~Configure HDFS~~
2. ~~Format HDFS~~
3. Configure Core
4. Start HDFS
5. Experiment HDFS commands

   (ls, mkdir, copyFromLocal, cat)

Combine

Hadoop & HDFS

Run Hadoop Job

```
$ hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar wordcount /books out
$ hdfs dfs -ls out
$ hdfs dfs -cat out/_SUCCESS
$ hdfs dfs -cat out/part-r-00000
```

```
young-armed  1
young;   2
younger  2
youngest 1
youngish 1
your 251
your@login   1
yours    5
yours?   1
yourselbs    1
yourself 14
yourself,    5
yourself,"   1
yourself.'   1
yourself;    4
yourself?    1
yourselves   1
yourselves!  3
yourselves,  1
yourselves," 1
yourselves;  1
youth    5
youth,   2
youth.   1
youth;   1
youthful 1
```

Lab 4

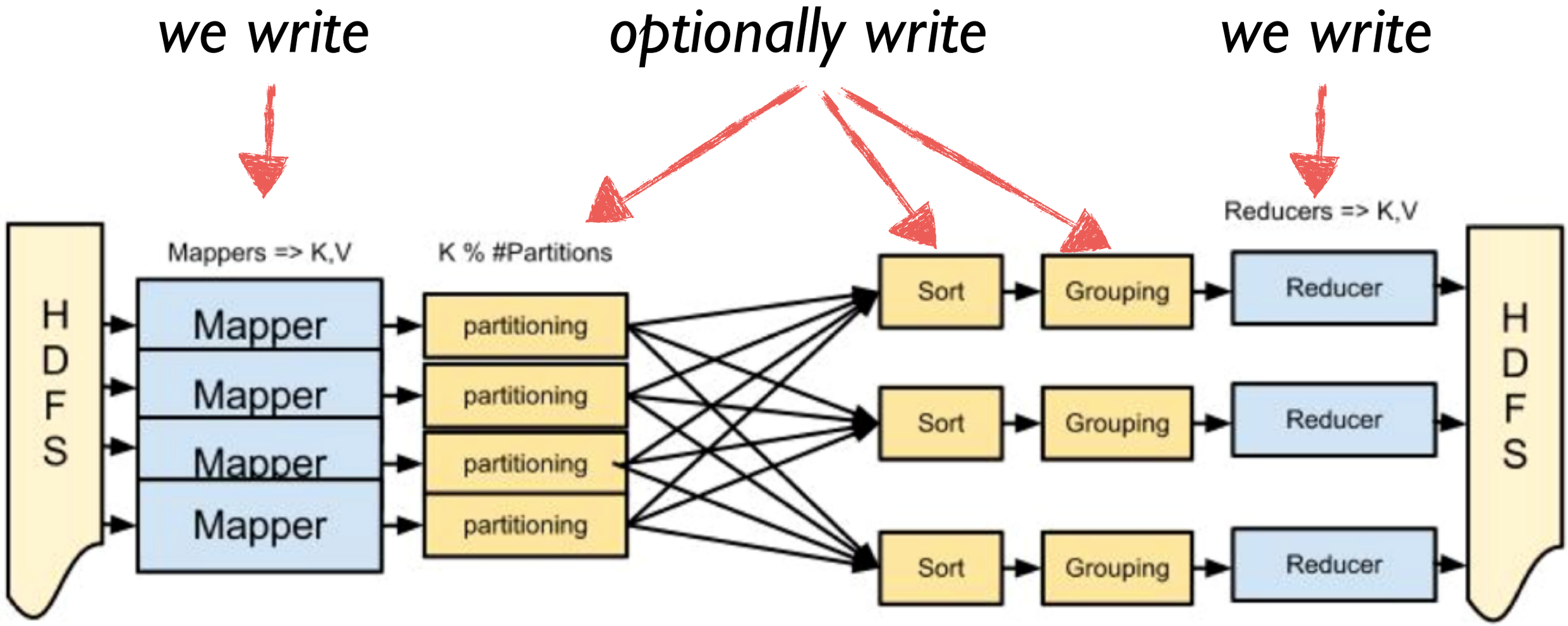
1. Run wordcount job
2. Review output
3. Run wordcount job again with same parameters

Writing Map Reduce Jobs

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

{K1,V1}



The MapReduce Pipeline

A mapper receives (Key, Value) & outputs (Key, Value)
A reducer receives (Key, Iterable[Value]) and outputs (Key, Value)
Partitioning / Sorting / Grouping provides the Iterable[Value] & Scaling

{K1,V1} ⟶ {K2, List<V2>} ⟶ {K3,V3}

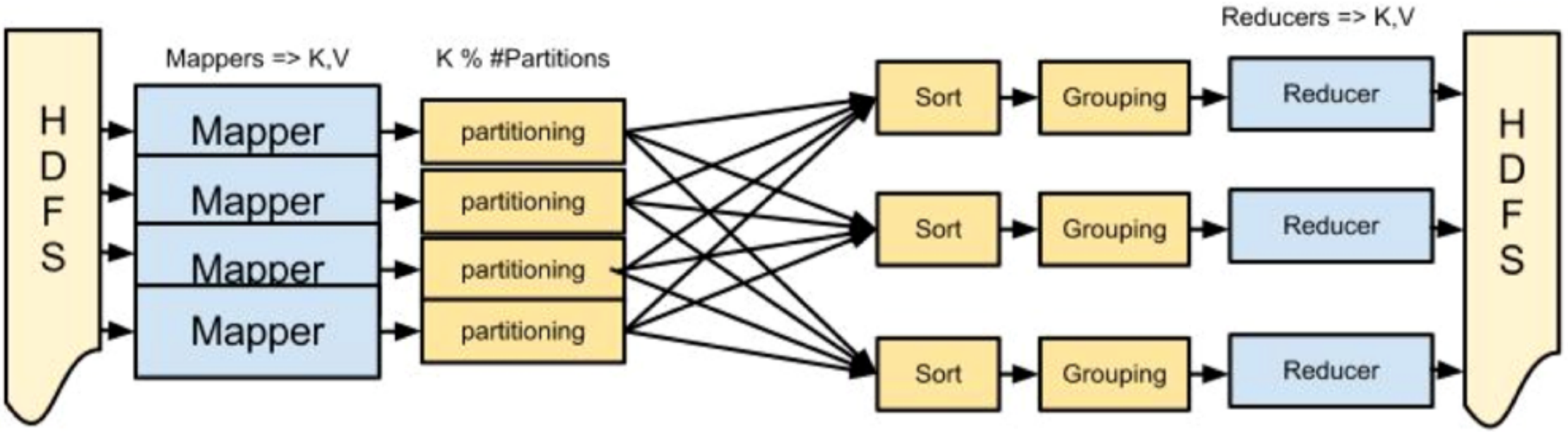
MOBY DICK; OR THE WHALE

By Herman Melville

CHAPTER 1. Loomings.

Call me Ishmael. Some years ago--never mind how long precisely--having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off--then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

There now is your insular city of the Manhattoes, belted round by wharves as Indian isles by coral reefs--commerce surrounds it with her surf. Right and left, the streets take you waterward. Its extreme downtown is the battery, where that noble mole is washed by waves, and cooled by breezes, which a few hours previous were out of sight of land. Look at the crowds of water-gazers there.



K V

```
1  Call me Ishmael. Some years ago--never mind how long precisely--having
2  little or no money in my purse, and nothing particular to interest me on
3  shore, I thought I would sail about a little and see the watery part of
4  the world. It is a way I have of driving off the spleen and regulating
5  the circulation.
```

{K1,V1} ⟶ {K2, List<V2>} ⟶ {K3,V3}

Mapper

```
package com.manifestcorp.hadoop.wc;

import java.io.IOException;

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

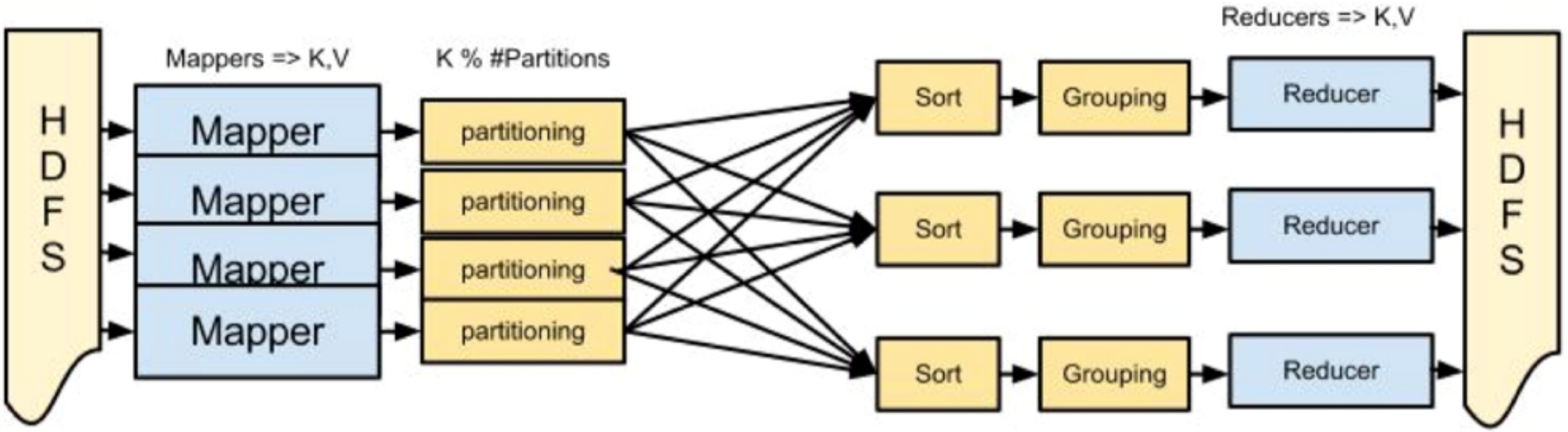
K1 V1 K2 V2

```
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private static final String SPACE = " ";
    private static final IntWritable ONE = new IntWritable(1);
    private Text word = new Text();
```

K1 V1

```
    public void map(LongWritable key, Text value, Context context)
                                                    throws IOException, InterruptedException {
        String[] words = value.toString().split(SPACE);

        for (String str: words) {
            word.set(str);
            context.write(word, ONE);
        }

    }
}
```

K2 V2



K *V*

```
1  Call me Ishmael. Some years ago--never mind how long precisely--having
2  little or no money in my purse, and nothing particular to interest me on
3  shore, I thought I would sail about a little and see the watery part of
4  the world. It is a way I have of driving off the spleen and regulating
5  the circulation.
```

map

K	*V*
Call	1
me	1
Ishmael.	1
Some	1
years	1
ago--never	1
mind	1
how	1
of	1
long	1
little	1
of	1
or	1
of	1

sort

K	*V*
ago--never	1
Call	1
how	1
Ishmael.	1
me	1
little	1
long	1
mind	1
of	1
of	1
of	1
or	1
Some	1
years	1

group

K	*V*
ago--never	1
Call	1
how	1
Ishmael.	1
me	1
little	1
long	1
mind	1
of	1,1,1
or	1
Some	1
years	1

{K1,V1} → {K2, List<V2>} → {K3,V3}

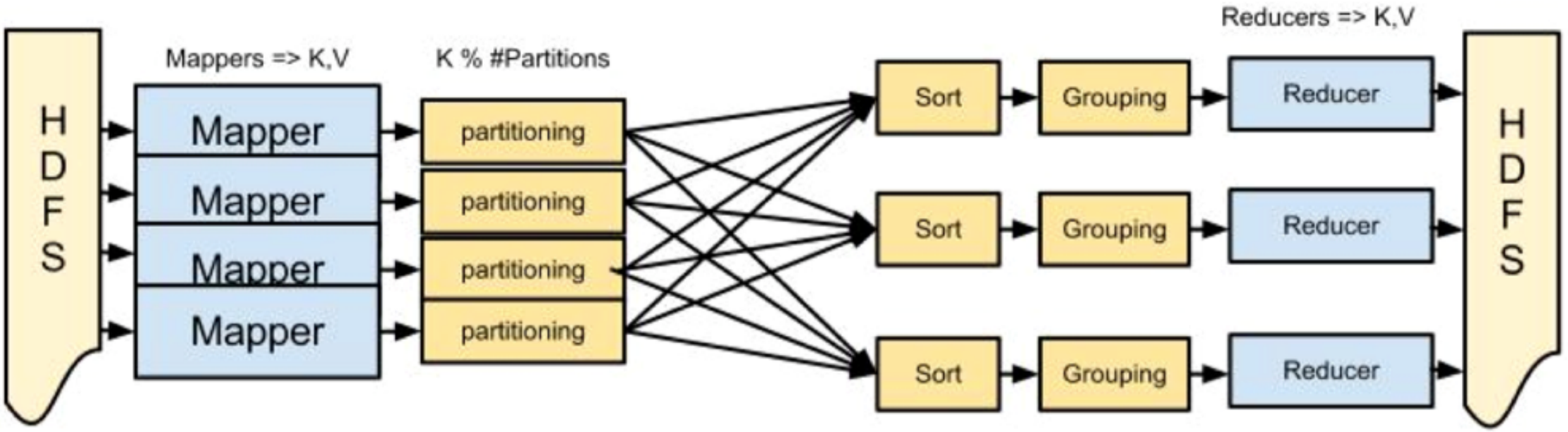
Reducer

```
package com.manifestcorp.hadoop.wc;

import java.io.IOException;

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

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

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                                    throws IOException, InterruptedException {
        int total = 0;

        for (IntWritable value : values) {
            total++;
        }

        context.write(key, new IntWritable(total));
    }
}
```

K2 V2 K3 V3

K2 V2

K3 V3



K *V*

```
1  Call me Ishmael. Some years ago--never mind how long precisely--having
2  little or no money in my purse, and nothing particular to interest me on
3  shore, I thought I would sail about a little and see the watery part of
4  the world. It is a way I have of driving off the spleen and regulating
5  the circulation.
```

map

K	*V*
Call	1
me	1
Ishmael.	1
Some	1
years	1
ago--never	1
mind	1
how	1
of	1
long	1
little	1
of	1
or	1
of	1

sort

K	*V*
ago--never	1
Call	1
how	1
Ishmael.	1
me	1
little	1
long	1
mind	1
of	1
of	1
of	1
or	1
Some	1
years	1

group

K	*V*
ago--never	1
Call	1
how	1
Ishmael.	1
me	1
little	1
long	1
mind	1
of	1,1,1
or	1
Some	1
years	1

reduce

K	*V*
ago--never	1
Call	1
how	1
Ishmael.	1
me	1
little	1
long	1
mind	1
of	3
or	1
Some	1
years	1

{K1,V1} → {K2, List<V2>} → {K3,V3}

Driver

```java
package com.manifestcorp.hadoop.wc;

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;

public class MyWordCount {

    public static void main(String[] args) throws Exception {
        Job job = new Job();
        job.setJobName("my word count");
        job.setJarByClass(MyWordCount.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.manifestcorp.hadoop</groupId>
  <artifactId>hadoop-mywordcount</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <hadoop.version>2.2.0</hadoop.version>
  </properties>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>${hadoop.version}</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

Run Hadoop Job

```
$ hadoop jar target/hadoop-mywordcount-0.0.1-SNAPSHOT.jar com.manifestcorp.hadoop.wc.MyWordCount /books out
```

Lab 5

1. Unzip /opt/data/hadoop-mywordcount-start.zip
2. Write Mapper class
3. Write Reducer class
4. Write Driver class
5. Build (mvn clean package)
6. Run mywordcount job
7. Review output

Unit Testing

https://mrunit.apache.org/

```
<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
 </dependency>
```

```
<dependency>
  <groupId>org.apache.mrunit</groupId>
  <artifactId>mrunit</artifactId>
  <version>1.1.0</version>
  <scope>test</scope>
  <classifier>hadoop2</classifier>
</dependency>
```

```java
package com.manifestcorp.hadoop.wc;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.apache.hadoop.mrunit.mapreduce.MapReduceDriver;
import org.apache.hadoop.mrunit.mapreduce.ReduceDriver;
import org.junit.Before;
import org.junit.Test;

public class WordCountTest {

  MapReduceDriver<LongWritable, Text, Text, IntWritable, Text, IntWritable> mapReduceDriver;
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;
  ReduceDriver<Text, IntWritable, Text, IntWritable> reduceDriver;

  @Before
  public void setUp() {
    WordCountMapper mapper = new WordCountMapper();
    WordCountReducer reducer = new WordCountReducer();

    mapDriver = new MapDriver<>(mapper);
    reduceDriver = new ReduceDriver<>(reducer);

    mapReduceDriver = new MapReduceDriver<>(mapper, reducer);
  }

  @Test
  public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("java hadoop java"));
    mapDriver.withOutput(new Text("java"), new IntWritable(1));
    mapDriver.withOutput(new Text("hadoop"), new IntWritable(1));
    mapDriver.withOutput(new Text("java"), new IntWritable(1));
    mapDriver.runTest();
  }
  // ...
}
```

```java
public class WordCountTest {

  MapReduceDriver<LongWritable, Text, Text, IntWritable, Text, IntWritable> mapReduceDriver;
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;
  ReduceDriver<Text, IntWritable, Text, IntWritable> reduceDriver;

  @Before
  public void setUp() {
    WordCountMapper mapper = new WordCountMapper();
    WordCountReducer reducer = new WordCountReducer();

    mapDriver = new MapDriver<>(mapper);
    reduceDriver = new ReduceDriver<>(reducer);

    mapReduceDriver = new MapReduceDriver<>(mapper, reducer);
  }

  @Test
  public void testMapper() throws IOException {
    mapDriver.withInput(new LongWritable(1), new Text("java hadoop java"));
    mapDriver.withOutput(new Text("java"), new IntWritable(1));
    mapDriver.withOutput(new Text("hadoop"), new IntWritable(1));
    mapDriver.withOutput(new Text("java"), new IntWritable(1));
    mapDriver.runTest();
  }

  @Test
  public void testReducer() throws IOException {
    List<IntWritable> values = new ArrayList<>();
    values.add(new IntWritable(1));
    values.add(new IntWritable(1));

    reduceDriver.withInput(new Text("java"), values);
    reduceDriver.withOutput(new Text("java"), new IntWritable(2));
    reduceDriver.runTest();
  }

  @Test
  public void testMapReduce() throws IOException {
    mapReduceDriver.withInput(new LongWritable(1), new Text("java hadoop java"));
    mapReduceDriver.withInput(new LongWritable(2), new Text("java spring java"));
    mapReduceDriver.addOutput(new Text("hadoop"), new IntWritable(1));
    mapReduceDriver.addOutput(new Text("java"), new IntWritable(4));
    mapReduceDriver.addOutput(new Text("spring"), new IntWritable(1));
    mapReduceDriver.runTest();
  }

}
```

Lab 6

Using the NASA Apache access log files found at

`/opt/data/NASA_access_log_Aug95.gz`
and
`/opt/data/NASA_access_log_Jul95.gz`
write a Mapper, Reducer, Driver and unit test to extract meaningful information from the logs.

Hint: there is a base project at ~/workspaces/hadoop-logs

Hadoop in the Cloud

Amazon
Elastic
MapReduce

Windows Azure

http://aws.amazon.com/elasticmapreduce/

AWS | Services | Edit | cmj @ 5637-0073-6850 | Global | Support

Upload | Create Folder | Actions | None | Properties | Transfers

All Buckets / cmj-emr

Name	Storage Class	Size	Last Modified
books	--	--	--
hadoop-mywordcount-0.0.1-SNAPSHOT.jar	Standard	5.4 KB	Tue Jan 06 22:01:56 GMT-500 2015
logs	--	--	--
wordcount	--	--	--

Welcome to Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

You do not appear to have any clusters. Create one now:

Create cluster

How Elastic MapReduce Works

Upload

Upload your data and processing application to S3.

Learn more

Create

Configure and create your cluster by specifying data inputs, outputs, cluster size, security settings, etc.

Learn more

Monitor

Monitor the health and progress of your cluster. Retrieve the output in S3.

Learn more

Additional Information

More about Elastic MapReduce

- EMR overview
- FAQs
- Pricing

More Help Using Elastic MapReduce

- Forum
- Documentation
- Developer Guide
- Quick Reference Card
- API Reference
- EMR on GitHub
- Help portal

AWS | Services | Edit | cmj @ 5637-0073-6850 | Oregon | Support

Elastic MapReduce | Create Cluster | EMR Help

Configure sample application

Cluster Configuration

Cluster name Word count

Termination protection Yes / No — Prevents accidental termination of the cluster: to shut down the cluster, you must turn off termination protection. Learn more

Logging Enabled — Copy the cluster's log files automatically to S3. Learn more

Log folder S3 location

s3://cmj-emr/logs/

s3://<bucket-name>/<folder>/

Debugging Enabled — Index logs to enable console debugging functionality (requires logging). Learn more

Tags

Optional: Add up to 10 tags to your EMR cluster. A tag consists of a case-sensitive key-value pair. Tags on EMR clusters are propagated to the underlying EC2 instances. Learn more about tagging your Amazon EMR clusters.

Key	Value (optional)
Add a key to create a tag	

Software Configuration

Hadoop distribution Amazon — Use Amazon's Hadoop distribution. Learn more

AMI version 3.3.1 — Determines the base configuration of the instances in your cluster, including the Hadoop version. Learn more

MapR — Use MapR's Hadoop distribution. Learn more

Applications to be installed	Version

Additional applications Select an application

Configure and add

File System Configuration

The EMR File System (EMRFS) and the Hadoop Distributed File System (HDFS) are both installed on your EMR cluster. HDFS stores data on an EMR cluster, while EMRFS allows EMR clusters to store data on S3. You can enable server-side encryption and consistent view for EMRFS below, or use a bootstrap action to configure additional settings for EMRFS.

Server-side encryption Enabled — Uses S3 server-side encryption on files written to S3 by EMRFS. Learn more

Consistent view Enabled — Monitors list and read-after-write (for new puts) consistency for files in S3. Learn more

Pricing for Amazon EMR and Amazon EC2 (On-Demand)

Region: US East (Northern Virginia)

	Amazon EC2 Price	Amazon Elastic MapReduce Price
Standard On-Demand Instances		
Small (Default)	$0.06 per hour	$0.015 per hour
Medium	$0.12 per hour	$0.03 per hour
Large	$0.24 per hour	$0.06 per hour
Extra Large	$0.48 per hour	$0.12 per hour
Hi-Memory On-Demand Instances		
Extra Large	$0.41 per hour	$0.09 per hour
Double Extra Large	$0.82 per hour	$0.21 per hour
Quadruple Extra Large	$1.64 per hour	$0.42 per hour
Hi-CPU On-Demand Instances		
Medium	$0.145 per hour	$0.03 per hour
Extra Large	$0.58 per hour	$0.12 per hour
Cluster Compute On-Demand Instances		
Quadruple Extra Large	$1.30 per hour	$0.27 per hour
Eight Extra Large	$2.40 per hour	$0.50 per hour
Cluster GPU On-Demand Instances		
Quadruple Extra Large	$2.10 per hour	$0.42 per hour
High-I/O On-Demand Instances		
Quadruple Extra Large	$3.10 per hour	$0.47 per hour
High-Storage On-Demand Instances		
Eight Extra Large	$4.60 per hour	$0.69 per hour

Making it more Real

http://aws.amazon.com/architecture/

amazon web services

Sign Up | My Account / Console | English

AWS Products & Solutions | AWS Product Information | Developers | Support

Architecture Center

- **Overview**
- AWS Simple Icons

Related Resources

- AWS Economics Center
- Security & Compliance
- AWS Products & Services
- AWS Solutions
- Case Studies

AWS Support

Please visit **AWS Support** for more details on getting one on one support for your architecture questions.

AWS Architecture Center

The AWS Architecture Center is designed to provide you with the necessary guidance and best practices to build highly scalable and reliable applications in the AWS Cloud. These resources will help you understand the AWS platform, its services and features, and will provide architectural guidance for design and implementation of systems that run on the AWS infrastructure.

Featured

Reference Implementation: Deploy a Microsoft SharePoint 2010 Server Farm in the AWS Cloud in 6 Simple Steps

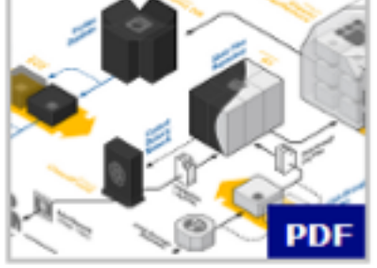
SHAW MEDIA

Read Shaw Media Case Study "Our average uptime increased rapidly from 98.8% to 99.9% without re-architecting applications"

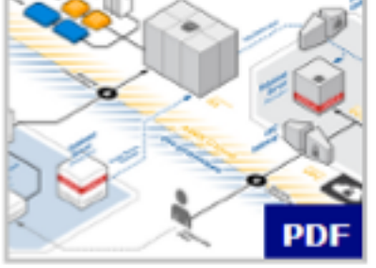
AWS Reference Architectures

The flexibility of AWS allows you to design your application architectures the way you like. AWS Reference Architecture Datasheets provide you with the architectural guidance you need in order to build an application that takes full advantage of the AWS cloud. Each datasheet includes a visual representation of the architecture and basic description of how each service is used.

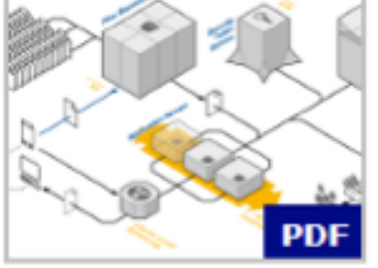
Large Scale Processing and Huge Data sets
Build high-performance computing systems that involve Big Data (PDF)



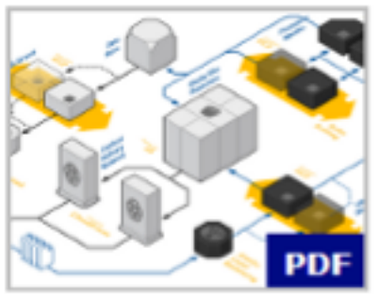
Ad Serving
Build highly-scalable online ad serving solutions (PDF)



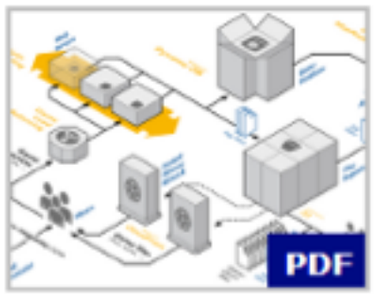
Disaster Recovery for Local Applications
Build cost-effective Disaster Recovery solutions for on-premises applications (PDF)



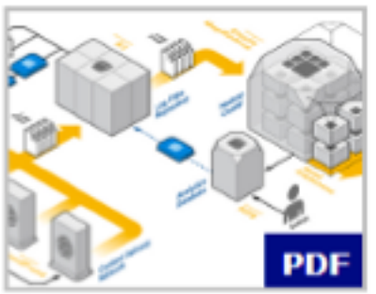
File Synchronization
Build simple file synchronization service (PDF)



Media Sharing
Cloud-powered Media Sharing Framework (PDF)

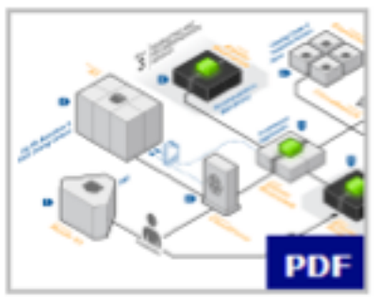


Online Games
Build powerful online games (PDF)

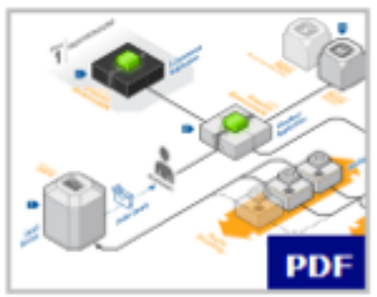


Log Analysis
Analyze massive volumes of log data in the cloud (PDF)

Financial Services Grid Computing
Build highly scalable and elastic grids for the Financial Services Sector (PDF)



E-Commerce Website Part 1: Web Frontend
Build elastic Web Front-ends for an e-Commerce website (PDF)



E-Commerce Website Part 2: Checkout Pipeline
Build highly scalable checkout pipeline for an e-Commerce website (PDF)

E-Commerce Website Part 3: Marketing and Recommendations
Build highly scalable recommendation engine for an e-Commerce website (PDF)

AWS Reference Architectures

- Amazon EC2
- Amazon EC2 Spot
- Amazon EMR
- Amazon S3
- Amazon CloudFront
- Amazon RDS

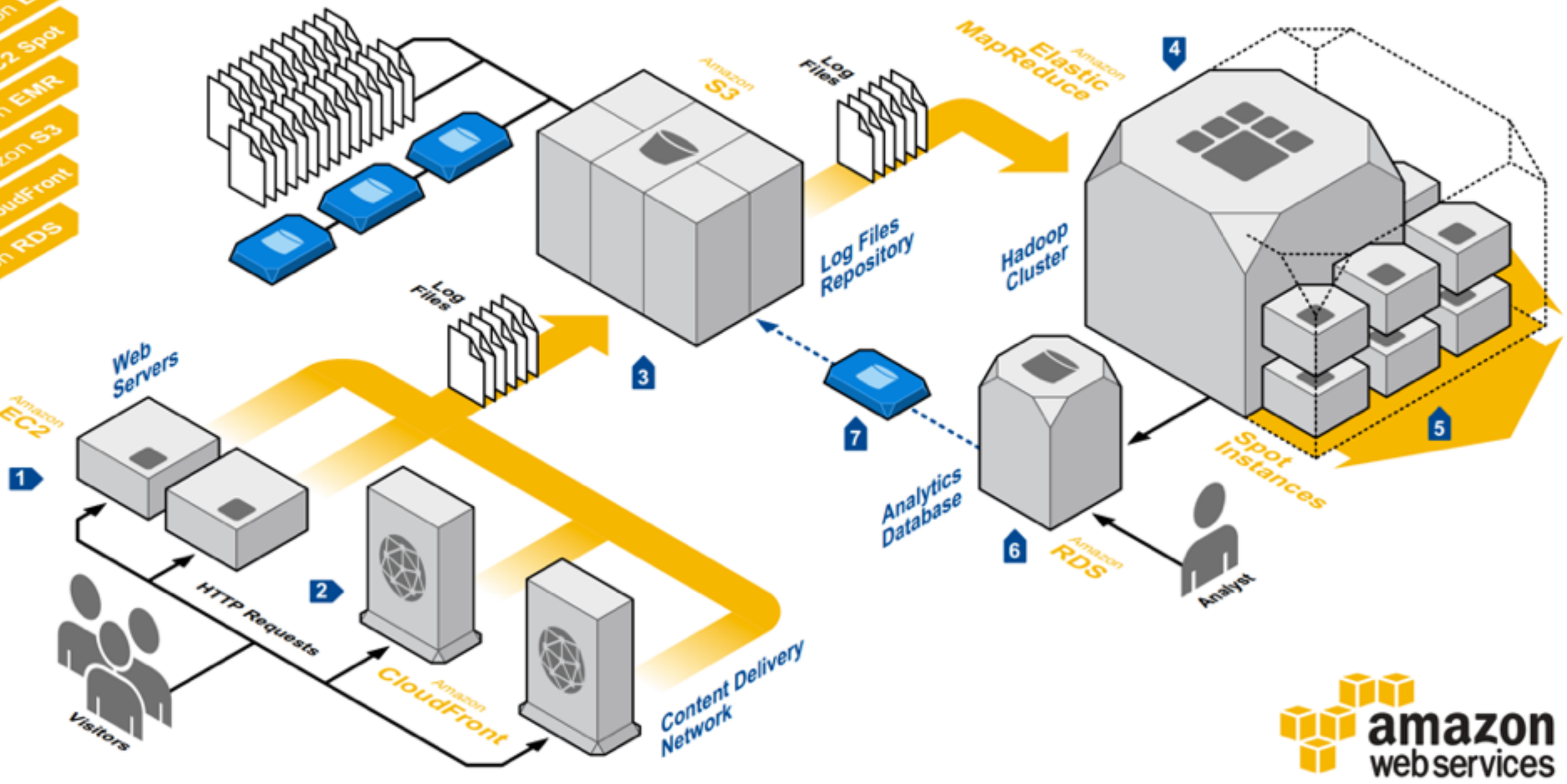
WEB LOG ANALYSIS

Amazon Web Services provides services and infrastructure to build reliable, fault-tolerant, and highly available web applications in the cloud. In production environments, these applications can generate huge amounts of log information.

This data can be an important source of knowledge for any company that is operating web applications. Analyzing logs can reveal information such as traffic patterns, user behavior, marketing profiles, etc.

However, as the web application grows and the number of visitors increases, storing and analyzing web logs becomes increasingly challenging.

This diagram shows how to use Amazon Web Services to build a scalable and reliable large-scale log analytics platform. The core component of this architecture is Amazon Elastic MapReduce, a web service that enables analysts to process large amounts of data easily and cost-effectively using a Hadoop hosted framework.



amazon web services

System Overview

1 The web front-end servers are running on **Amazon Elastic Compute Cloud** (Amazon EC2) instances.

2 **Amazon CloudFront** is a content delivery network that uses low latency and high data transfer speeds to distribute static files to customers. This service also generates valuable log information.

3 Log files are periodically uploaded to **Amazon Simple Storage Service** (Amazon S3), a highly available and reliable data store. Data is sent in parallel from multiple web servers or edge locations.

4 An **Amazon Elastic MapReduce** cluster processes the data set. **Amazon Elastic MapReduce** utilizes a hosted Hadoop framework, which processes the data in a parallel job flow.

5 When **Amazon EC2** has unused capacity, it offers EC2 instances at a reduced cost, called the *Spot Price*. This price fluctuates based on availability and demand. If your workload is flexible in terms of time of completion or required capacity, you can dynamically extend the capacity of your cluster using **Spot Instances** and significantly reduce the cost of running your job flows.

6 Data processing results are pushed back to a relational database using tools like *Apache Hive*. The database can be an **Amazon Relational Database Service** (Amazon RDS) instance. **Amazon RDS** makes it easy to set up, operate, and scale a relational database in the cloud.

7 Like many services, **Amazon RDS** instances are priced on a pay-as-you-go model. After analysis, the database can be backed-up into **Amazon S3** as a database snapshot, and then terminated. The database can then be recreated from the snapshot whenever needed.

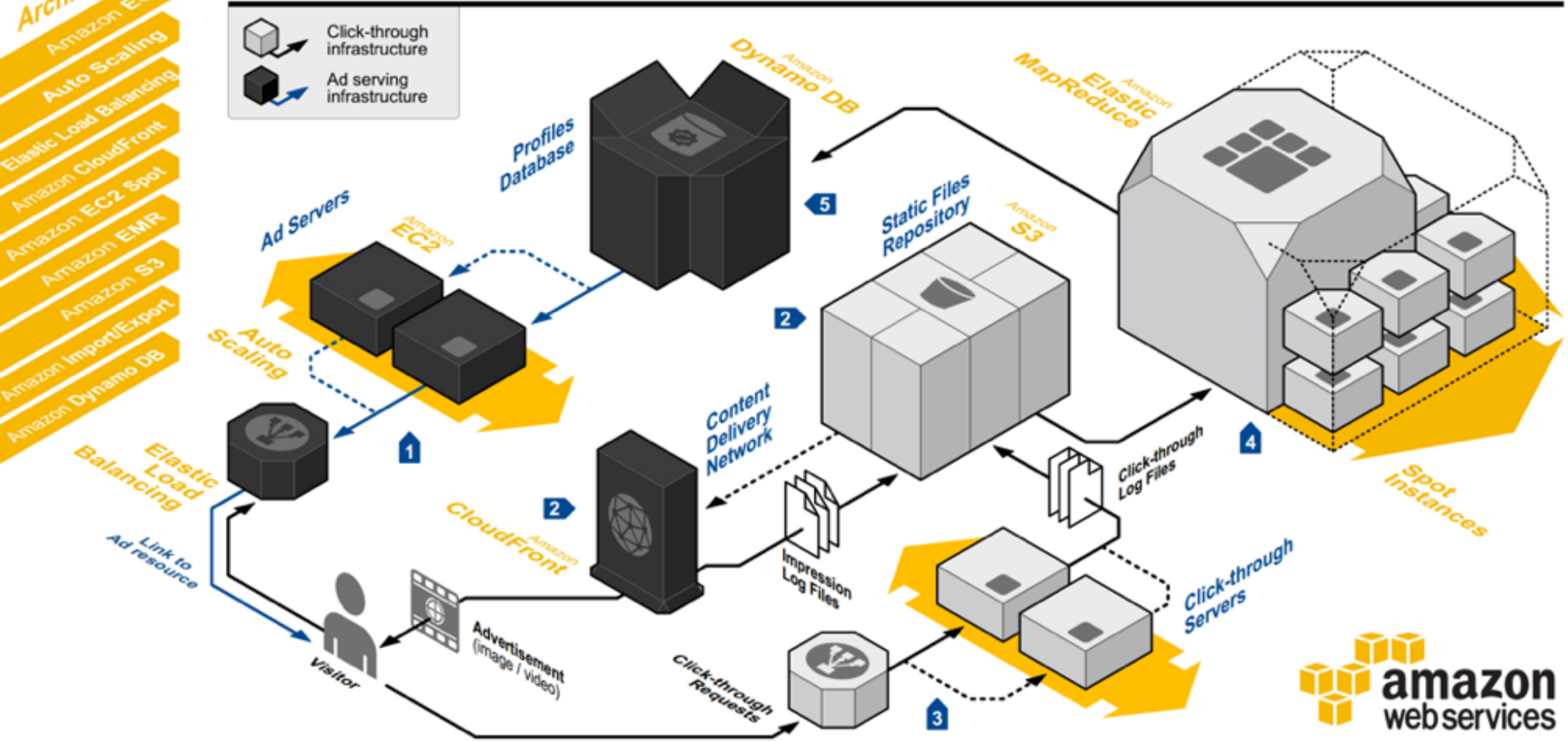
AWS Reference Architectures

- Amazon EC2
- Auto Scaling
- Elastic Load Balancing
- Amazon CloudFront
- Amazon EC2 Spot
- Amazon EMR
- Amazon S3
- Amazon Import/Export
- Amazon Dynamo DB

ADVERTISEMENT SERVING

Internet advertising services need to serve targeted advertising and must do so under limited time. These are just two of multiple technical challenges they face.

Amazon Web Services provides services and infrastructure to build reliable, fault-tolerant, and highly available ad serving platforms in the cloud. In this document, we describe the two main parts of such a system: ad serving infrastructure and click-through collection featuring a data analysis cluster.



System Overview

1 When visitors load a web page, ad servers return a pointer to the ad resource to be displayed. These servers are running on **Amazon Elastic Compute Cloud** (Amazon EC2) instances. They query a data set stored in an **Amazon DynamoDB** table to find relevant ads depending on the user's profile.

2 Ad files are downloaded from **Amazon CloudFront**, a content delivery service with low latency, high data-transfer speeds, and no commitments. Log information from displayed ads is stored on **Amazon Simple Storage Service** (Amazon S3), a highly available data store.

3 The click-through servers are a group of **Amazon EC2** instances dedicated to collecting click-through data. This information is contained in the log files of the click-through web servers, which are periodically uploaded to **Amazon S3**.

4 Ad impression and click-through data are retrieved and processed by an **Amazon Elastic MapReduce** cluster using a hosted Hadoop framework to process the data in a parallel job flow. The cluster's capacity can be dynamically extended using **Spot Instances** to reduce the processing time and the cost of running the job flow.

5 Data processing results are pushed back into **Amazon DynamoDB**, a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. **Amazon DynamoDB** tables can store and retrieve any amount of data, and serve any level of request traffic, both of which are specific requirements for storing and quickly retrieving visitors' profile information.

The high availability and fast performance of **Amazon DynamoDB** enable ad server front-ends to serve requests with predictable response time, even with high traffic volumes or large profile's data sets.

AWS Reference Architectures

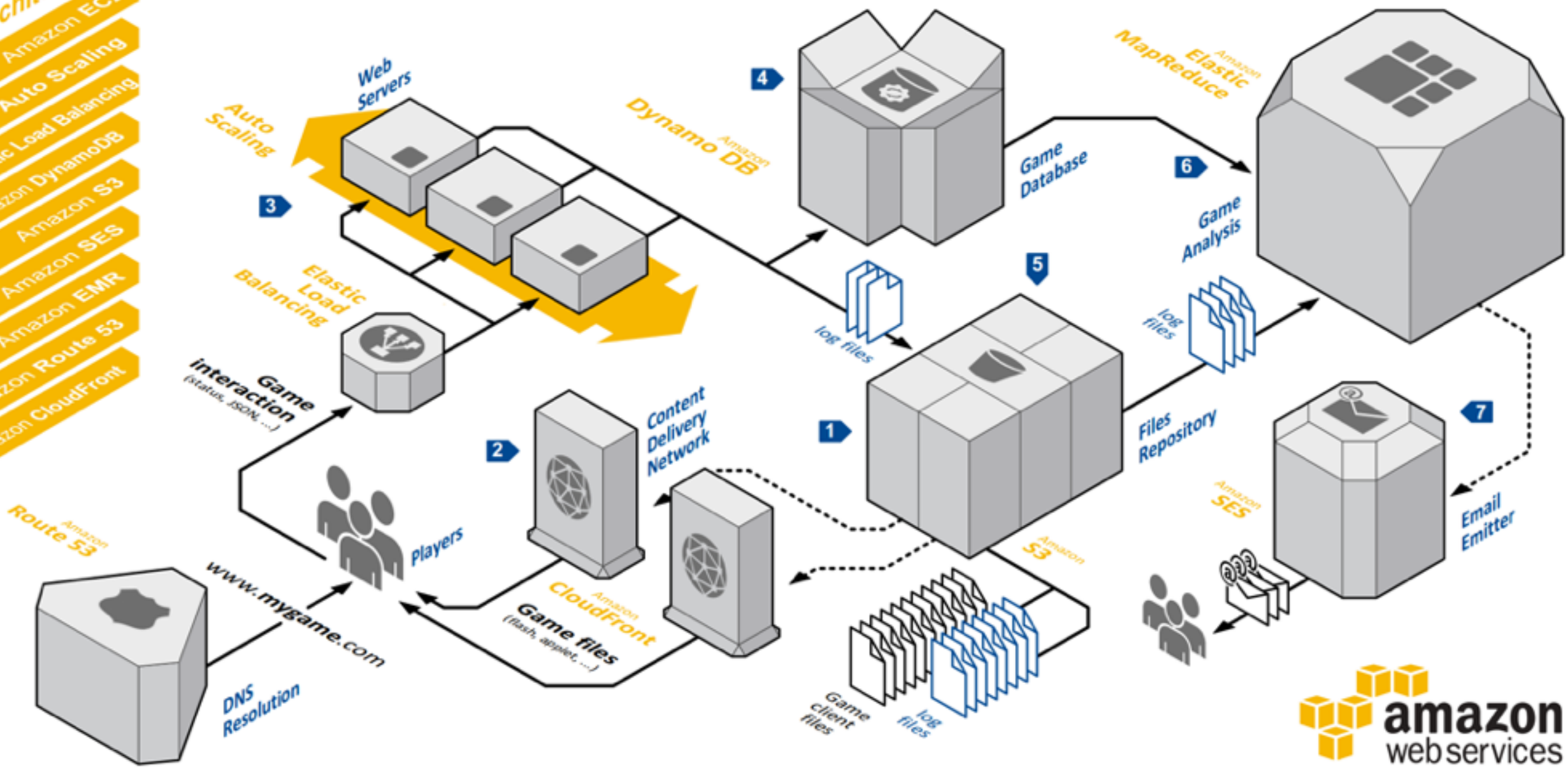
- Amazon EC2
- Auto Scaling
- Elastic Load Balancing
- Amazon DynamoDB
- Amazon S3
- Amazon SES
- Amazon EMR
- Amazon Route 53
- Amazon CloudFront

ONLINE GAMES

Online games back-end infrastructures can be challenging to maintain and operate. Peak usage periods, multiple players, and high volumes of write operations are some of the most common problems that operations teams face.

But the most difficult challenge is ensuring flexibility in the scale of that system. A popular game might suddenly receive millions of users in a matter of hours, yet it must continue to provide a satisfactory player experience. Amazon Web Services provides different tools and services that can be used for building online games that scale under high usage traffic patterns.

This document presents a cost-effective online game architecture featuring automatic capacity adjustment, a highly available and high-speed database, and a data processing cluster for player behavior analysis.



System Overview

1 Browser games can be represented as client-server applications. The client generally consists of static files, such as images, sounds, flash applications, or Java applets. Those files are hosted on **Amazon Simple Storage Service** (Amazon S3), a highly available and reliable data store.

2 As the user base grows and becomes more geographically distributed, a high-performance cache like **Amazon CloudFront** can provide substantial improvements in latency, fault tolerance, and cost. By using **Amazon S3** as the origin server for the **Amazon CloudFront** distribution, the game infrastructure benefits from fast network data transfer rates and a simple publishing/caching workflow.

3 Requests from the game application are distributed by **Elastic Load Balancing** to a group of web servers running on **Amazon Elastic Compute Cloud** (Amazon EC2) instances. **Auto Scaling** automatically adjusts the size of this group, depending on rules like network load, CPU usage, and so on.

4 Player data is persisted on **Amazon DynamoDB**, a fully managed NoSQL database service. As the player population grows, **Amazon DynamoDB** provides predictable performance with seamless scalability.

5 Log files generated by each web server are pushed back into **Amazon S3** for long-term storage.

6 Managing and analyzing high data volumes produced by online games platforms can be challenging. **Amazon Elastic MapReduce** (Amazon EMR) is a service that processes vast amounts of data easily. Input data can be retrieved from web server logs stored on **Amazon S3** or from player data stored in **Amazon DynamoDB** tables to run analytics on player behavior, usage patterns, etc. Those results can be stored again on **Amazon S3**, or inserted in a relational database for further analysis with classic business intelligence tools.

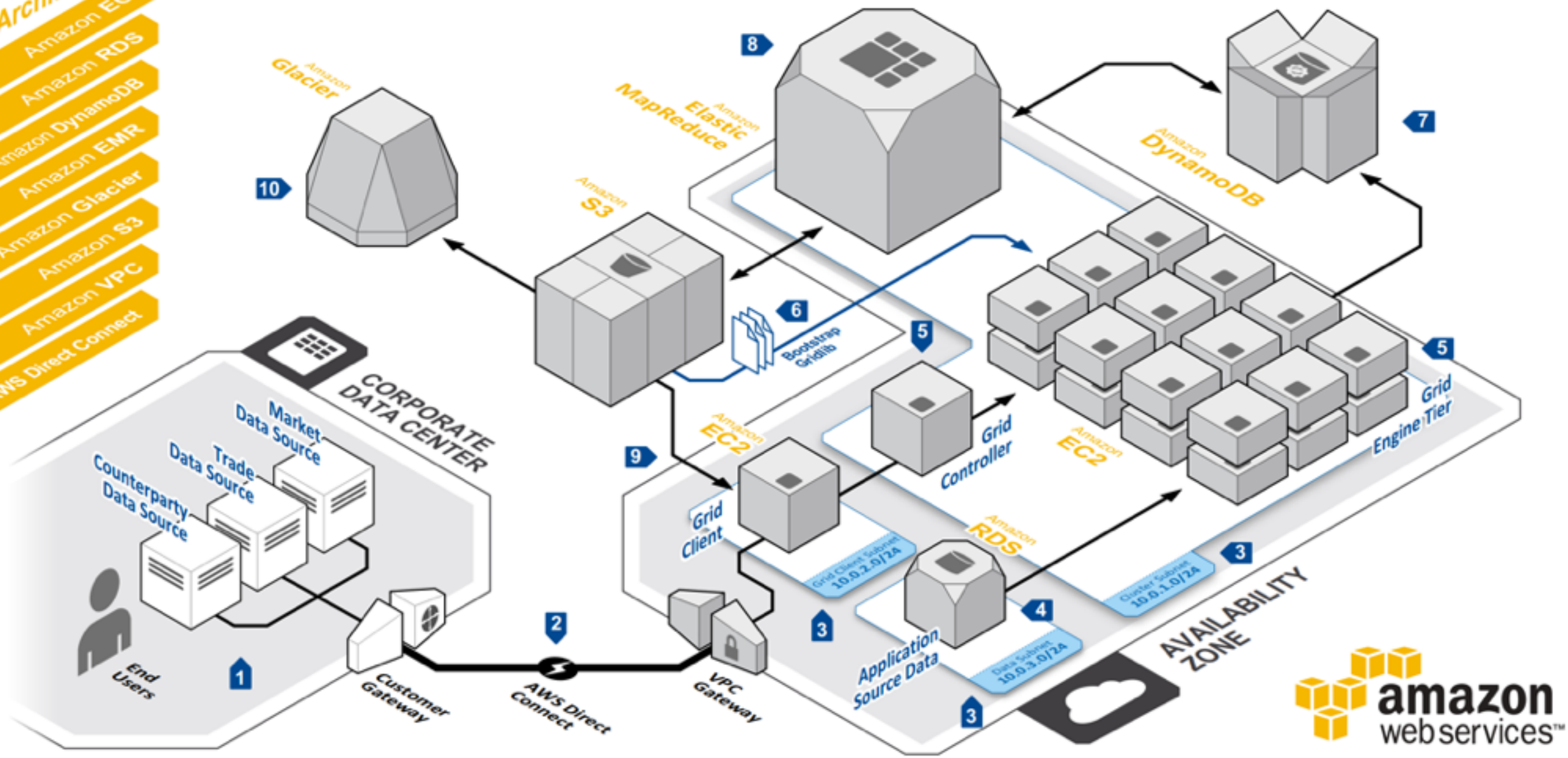
7 Based on the needs of the game, **Amazon Simple Email Service** (Amazon SES) can be used to send email to players in a cost-effective and scalable way.

AWS Reference Architectures

- Amazon EC2
- Amazon RDS
- Amazon DynamoDB
- Amazon EMR
- Amazon Glacier
- Amazon S3
- Amazon VPC
- AWS Direct Connect

FINANCIAL SERVICES GRID COMPUTING

Financial services grid computing on the cloud provides dynamic scalability and elasticity for operation when compute jobs are required, and utilizing services for aggregation that simplify the development of grid software.
On demand provisioning of hardware, and template driven deployment, combined with low latency access to existing on-premise data sources make AWS a powerful platform for high performance grid computing systems.



System Overview

1. Date sources for market, trade, and counterparties are installed on startup from on premise data sources, or from **Amazon Simple Storage Service** (Amazon S3).

2. **AWS DirectConnect** can be used to establish a low latency and reliable connection between the corporate data center site and AWS, in 1 to 10Gbit increments. For situations with lower bandwidth requirements, a VPN connection to the **VPC Gateway** can be established.

3. Private subnetworks are specifically created for customer source data, compute grid clients, and the grid controller and engines.

4. Application and corporate data can be securely stored in the cloud using the **Amazon Relational Database Service** (Amazon RDS).

5. Grid controllers and grid engines are running **Amazon Elastic Compute Cloud** (Amazon EC2) instances started on demand from **Amazon Machine Images** (AMIs) that contain the operating system and grid software.

6. Static data such as holiday calendars and QA libraries and additional gridlib bootstrapping data can be downloaded on startup by grid engines from **Amazon S3**.

7. Grid engine results can be stored in **Amazon DynamoDB**, a fully managed database providing configurable read and write throughput, allowing scalability on demand.

8. Results in **Amazon DynamoDB** are aggregated using a map/reduce job in **Amazon Elastic MapReduce** (Amazon EMR) and final output is stored in **Amazon S3**.

9. The compute grid client collects aggregate results from **Amazon S3**.

10. Aggregate results can be archived using **Amazon Glacier**, a low-cost, secure, and durable storage service.

AWS Reference Architectures

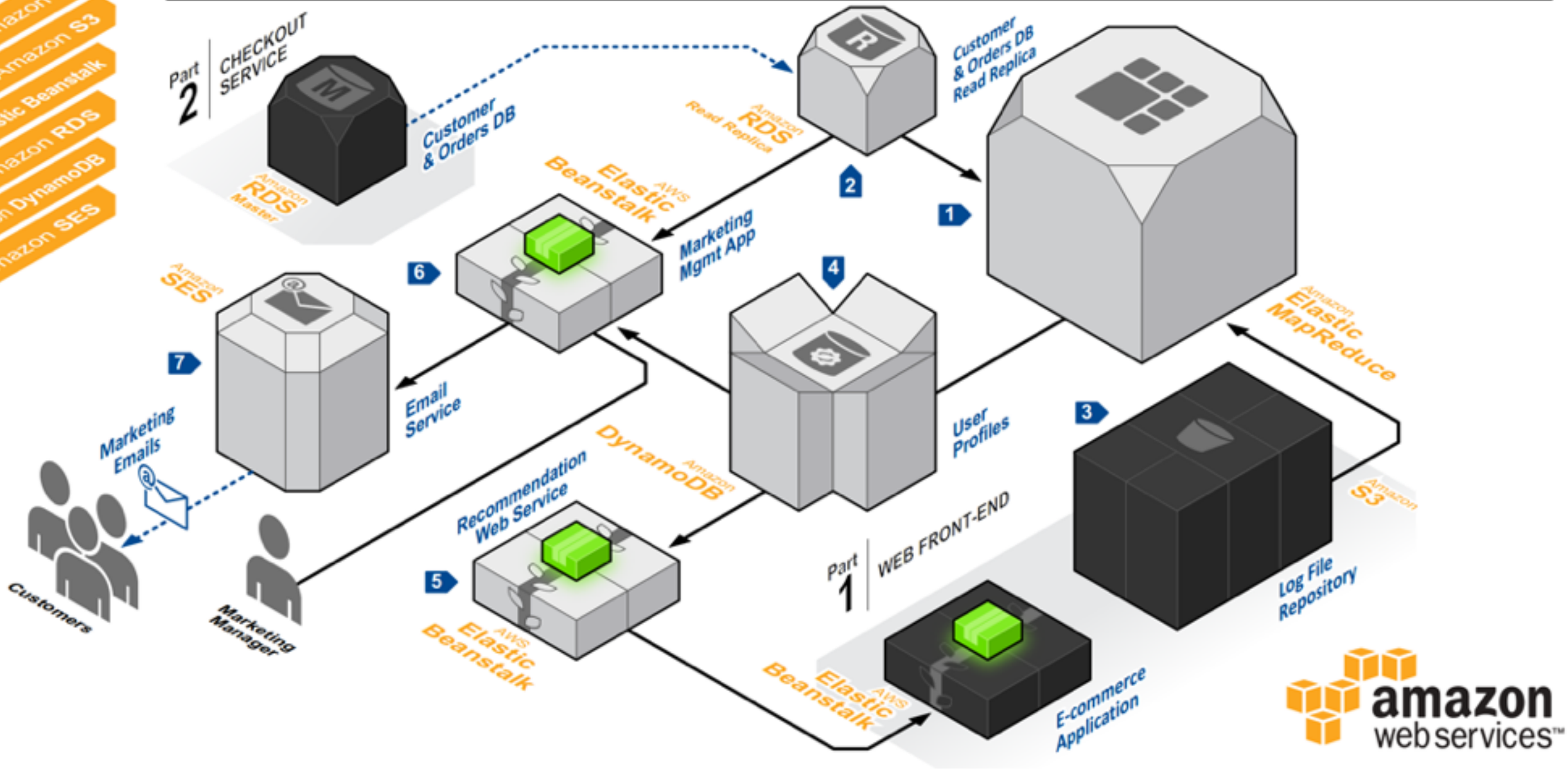
- Amazon EMR
- Amazon S3
- AWS Elastic Beanstalk
- Amazon RDS
- Amazon DynamoDB
- Amazon SES

E-COMMERCE WEBSITE

PART 3: MARKETING & RECOMMENDATIONS

With Amazon Web Services, you can build a recommendation and marketing service to manage targeted marketing campaigns and offer personalized product recommendations to customers who are browsing your e-commerce site.
In order to build such a service, you have to process very large amounts of data from multiple data sources. The resulting user profile information has to be available to deliver real-time product recommendations on your e-commerce website.

The insights that you gain about your customers can also be used to manage personalized marketing campaigns targeted at specific customer segments.
With the tools that AWS provides, you can build highly scalable recommendation services that can be consumed by different channels, such as dynamic product recommendations on the e - commerce website or targeted email campaigns for your customers.



System Overview

1 **Amazon Elastic MapReduce** (Amazon EMR) is a hosted Hadoop framework that runs on Amazon Elastic Compute Cloud (Amazon EC2) instances. It aggregates and processes user data from server log files and from the customer´s purchase history.

2 An **Amazon Relational Database Services** (Amazon RDS) Read Replica of customer and order databases is used by Amazon EMR to compute user profiles and by **Amazon Simple Email Service** (Amazon SES) to send targeted marketing emails to customers.

3 Log files produced by the e-commerce web front end have been stored on **Amazon Simple Storage Service** (Amazon S3) and are consumed by the Amazon EMR cluster to compute user profiles.

4 User profile information generated by the Amazon EMR cluster is stored in **Amazon DynamoDB**, a scalable, high-performance managed NoSQL database that can serve recommendations with low latency.

5 A recommendation web service used by the web front end is deployed by **AWS Elastic Beanstalk**. This service uses the profile information stored on Amazon DynamoDB to provide personalized recommendations to be shown on the e-commerce web front end.

6 A marketing administration application deployed by **AWS Elastic Beanstalk** is being used by marketing managers to send targeted email campaigns to customers with specific user profiles. The application reads customer email addresses from an **Amazon RDS Read Replica** of the customer database.

7 **Amazon SES** is used to send marketing emails to customers. **Amazon SES** is based on the scalable technology used by Amazon web sites around the world to send billions of messages a year.

amazon web services™

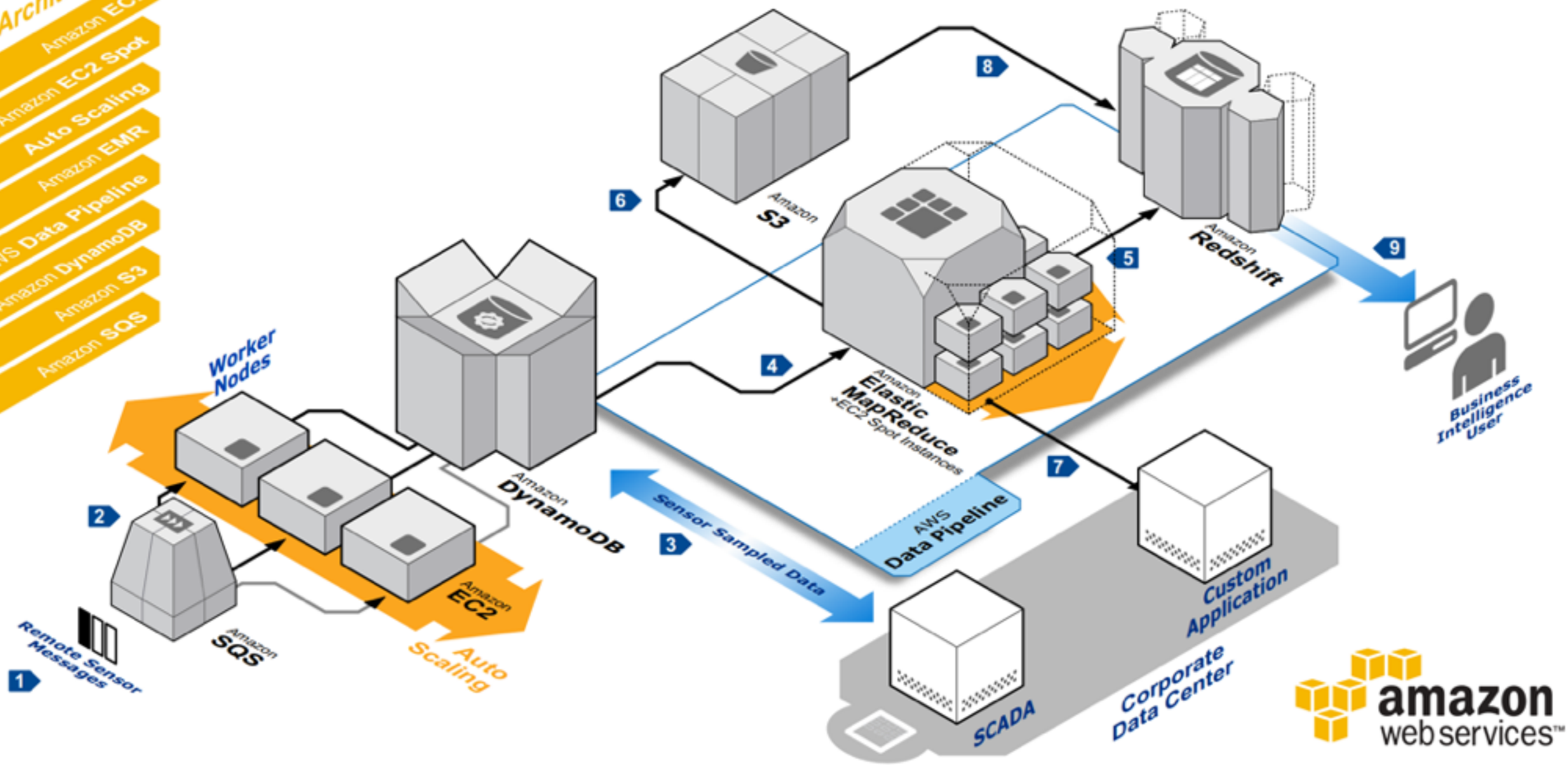
AWS Reference Architectures

- Amazon EC2
- Amazon EC2 Spot
- Auto Scaling
- Amazon EMR
- AWS Data Pipeline
- Amazon DynamoDB
- Amazon S3
- Amazon SQS

TIME SERIES PROCESSING

When data arrives as a succession of regular measurements, it is known as time series information. Processing of time series information poses systems scaling challenges that the elasticity of AWS services is uniquely positioned to address.

This elasticity is achieved by using Auto Scaling groups for ingest processing, AWS Data Pipeline for scheduled Amazon Elastic MapReduce jobs, AWS Data Pipeline for intersystem data orchestration, and Amazon Redshift for potentially massive-scale analysis. Key architectural throttle points involving Amazon SQS for sensor message buffering and less frequent AWS Data Pipeline scheduling keep the overall solution costs predictable and controlled.



System Overview

1. Remote devices such as power meters, mobile clients, ad-network clients, industrial meters, satellites, and environmental meters measure the world around them and send sampled sensor data as messages via HTTP(S) for processing.

2. Send messages to an **Amazon Simple Queue Service** queue for processing into Amazon DynamoDB using autoscaled **Amazon EC2** workers. Or, if the sensor source can do so, post sensor samples directly to **Amazon DynamoDB**. Try starting with a DynamoDB table that is a week-oriented, time-based table structure.

3. If a Supervisory Control and Data Acquisition (SCADA) system exists, create a flow of samples to or from **Amazon DynamoDB** to support additional cloud processing or other existing systems, respectively.

4. Using **AWS Data Pipeline**, create a pipeline with a regular **Amazon Elastic MapReduce** job that both calculates expensive sample processing and delivers samples and results.

5. The pipeline places results into **Amazon Redshift** for additional analysis.

6. The pipeline exports historical week-oriented sample tables, from **Amazon DynamoDB** to **Amazon Simple Storage Service** (Amazon S3)

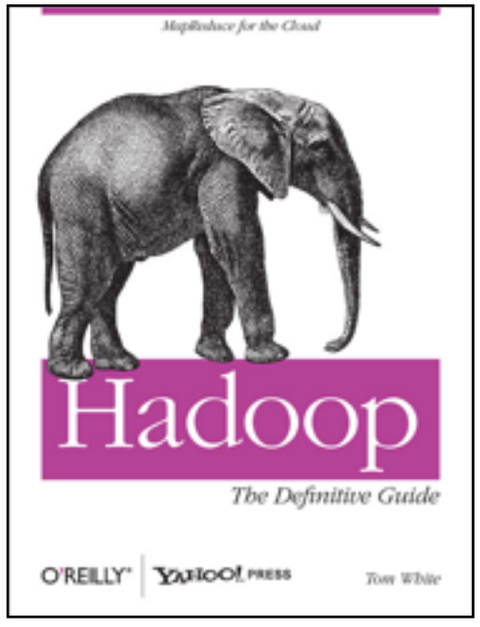
7. The pipeline also optionally exports results in a format custom applications can accept.

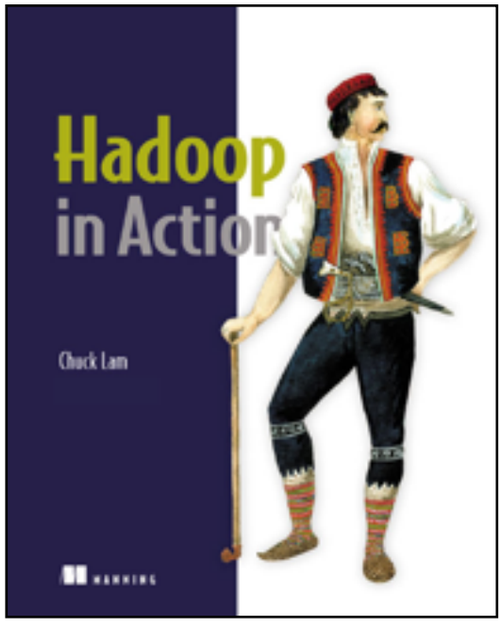
8. **Amazon Redshift** optionally imports historic samples to reside with calculated results.

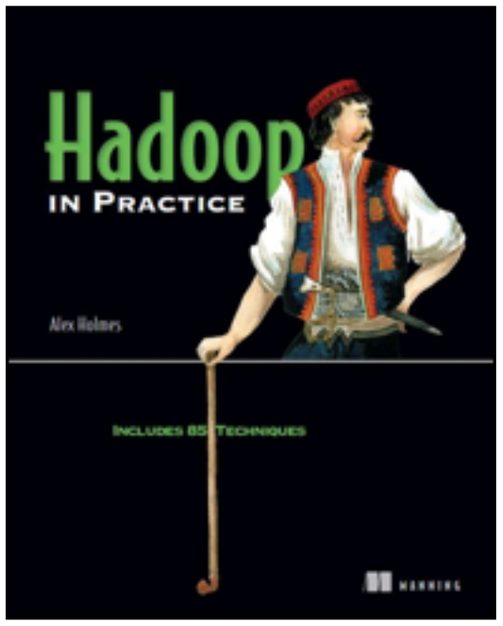
9. Using in-house or Amazon partner business intelligence solutions, **Amazon Redshift** supports additional analysis on a potentially massive scale.

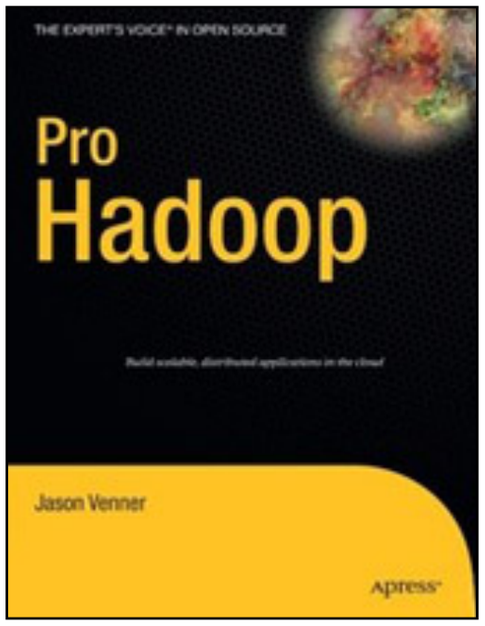
amazon web services™

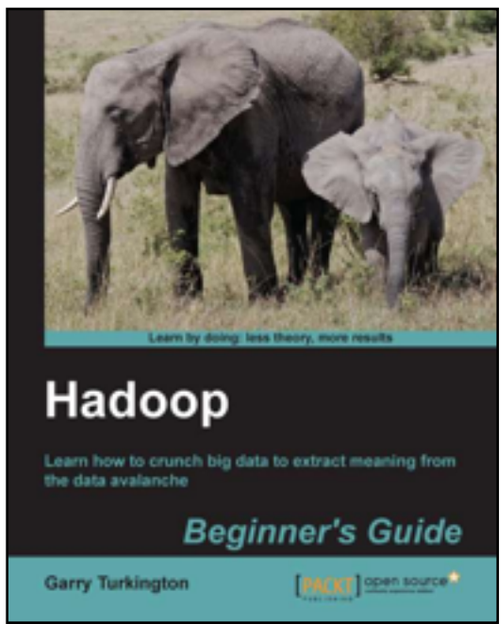
Resources

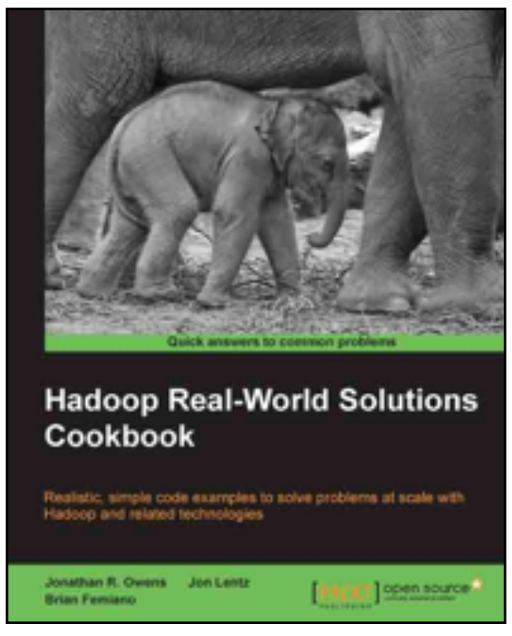

MapReduce for the Cloud
Hadoop
The Definitive Guide
O'REILLY
YAHOO! PRESS
Tom White

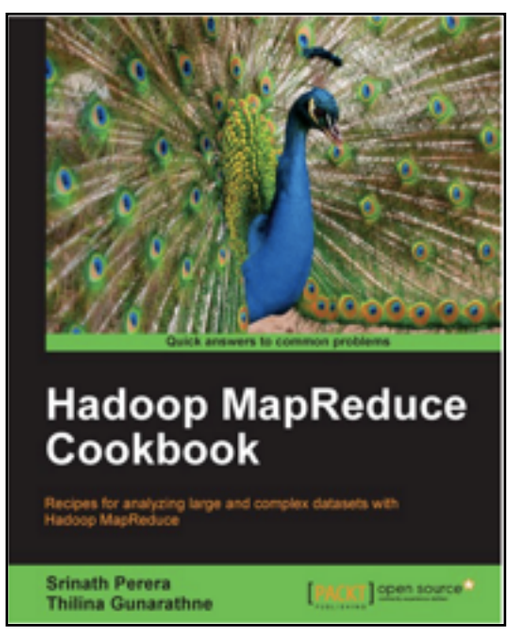

Hadoop
in Action
Chuck Lam
MANNING

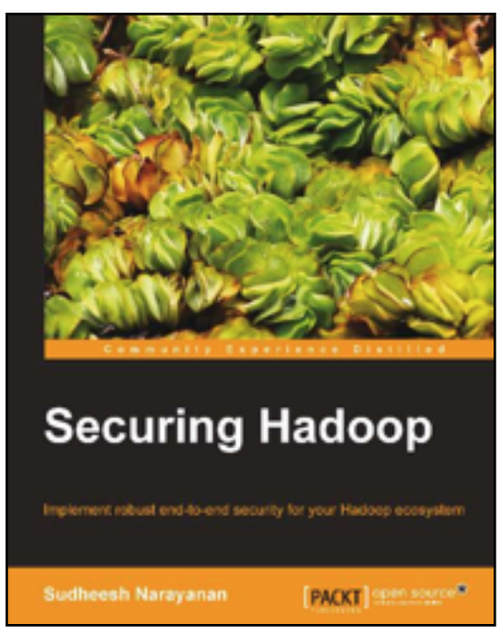

Hadoop
IN PRACTICE
Alex Holmes
INCLUDES 85 TECHNIQUES
MANNING


THE EXPERT'S VOICE® IN OPEN SOURCE
Pro
Hadoop
Jason Venner
Apress®


Learn by doing: less theory, more results
Hadoop
Learn how to crunch big data to extract meaning from the data avalanche
Beginner's Guide
Garry Turkington
PACKT open source


Quick answers to common problems
Hadoop Real-World Solutions Cookbook
Realistic, simple code examples to solve problems at scale with Hadoop and related technologies
Jonathan R. Owens
Jon Lentz
Brian Femiano
PACKT open source


Quick answers to common problems
Hadoop MapReduce Cookbook
Recipes for analyzing large and complex datasets with Hadoop MapReduce
Srinath Perera
Thilina Gunarathne
PACKT open source


Community Experience Distilled
Securing Hadoop
Implement robust end-to-end security for your Hadoop ecosystem
Sudheesh Narayanan
PACKT open source

Resources

117 DZone REFCARDZ

BROUGHT TO YOU BY: MAPR

Get More Refcardz! Visit Refcardz.com

CONTENTS
» Design Concepts
» Hadoop Components
» HDFS
» YARN
» YARN Applications
» MapReduce
» And More...

Getting Started with Apache Hadoop

By Adam Kawa and Piotr Krewski

INTRODUCTION

This Refcard presents Apache Hadoop, a software framework that enables distributed storage and processing of large datasets using simple high-level programming models. We cover the most important concepts of Hadoop, describe its architecture, guide how to start using it as well as write and execute various applications on Hadoop.

In the nutshell, Hadoop is an open-source project of the Apache Software Foundation that can be installed on a set of standard machines, so that these machines can communicate and work together to store and process large datasets. Hadoop has become very successful in recent years thanks to its ability to effectively crunch big data. It allows companies to store all of their data in one system and perform analysis on this data that would be otherwise impossible or very expensive to do with traditional solutions.

Many companion tools built around Hadoop offer a wide variety of processing techniques. Integration with ancillary systems and utilities is excellent, making real-world work with Hadoop easier and more productive. These tools together form the Hadoop Ecosystem.

HOT TIP: Visit http://hadoop.apache.org to get more information about the project and access detailed documentation.

Note: By a standard machine, we mean typical servers that are available from many vendors and have components that are expected to fail and be replaced on a regular base. Because Hadoop scales nicely and provides many fault-tolerance mechanisms, you do not need to break the bank to purchase expensive top-end servers to minimize the risk of hardware failure and increase storage capacity and processing power.

DESIGN CONCEPTS

To solve the challenge of processing and storing large datasets, Hadoop was built according to the following core characteristics:

- Distribution - instead of building one big supercomputer, storage and processing are spread across a cluster of smaller machines that communicate and work together.
- Horizontal scalability - it is easy to extend a Hadoop cluster by just adding new machines. Every new machine increases total storage and processing power of the Hadoop cluster.
- Fault-tolerance - Hadoop continues to operate even when a few hardware or software components fail to work properly.
- Cost-optimization - Hadoop runs on standard hardware; it does not require expensive servers.
- Programming abstraction - Hadoop takes care of all messy details related to distributed computing. Thanks to a high-level API, users can focus on implementing business logic that solves their real-world problems.
- Data locality – don't move large datasets to where application is running, but run the application where the data already is.

HADOOP COMPONENTS

Hadoop is divided into two core components

- HDFS - a distributed file system
- YARN - a cluster resource management technology

HOT TIP: Many execution frameworks run on top of YARN, each tuned for a specific use-case. The most important are discussed under 'YARN Applications' below.

Let's take a closer look on their architecture and describe how they cooperate.

Note: YARN is the new framework that replaces the former implementation of the processing layer in Hadoop. You can find how YARN addresses shortcomings of previous version on the Yahoo blog: https://developer.yahoo.com/blogs/hadoop/next-generation-apache-hadoop-mapreduce-3061.html.

HDFS

HDFS is a Hadoop distributed file system. It can be installed on commodity servers and run on as many servers as you need - HDFS easily scales to thousands of nodes and petabytes of data.

The larger HDFS setup is, the bigger probability that some disks, servers or network switches will fail. HDFS survives these types of failures by replicating data on multiple servers. HDFS automatically detects that a given component has failed and takes necessary recovery actions that happen transparently to the user.

HDFS is designed for storing large files of the magnitude of hundreds of megabytes or gigabytes and provides high-throughput streaming data access to them. Last but not least, HDFS supports the write-once-read-many model. For this use case HDFS works like a charm. If you need, however, to store a large number of small files with a random read-write access, then other systems like RDBMS and Apache HBase can do a better job.

Note: HDFS does not allow you to modify a file's content. There is only support for appending data at the end of the file. However, Hadoop was designed with HDFS to be one of many pluggable storage options – for example, with MapR-Fs, a proprietary filesystem, files are fully read-write. Other HDFS alternatives include Amazon S3 and IBM GPFS.

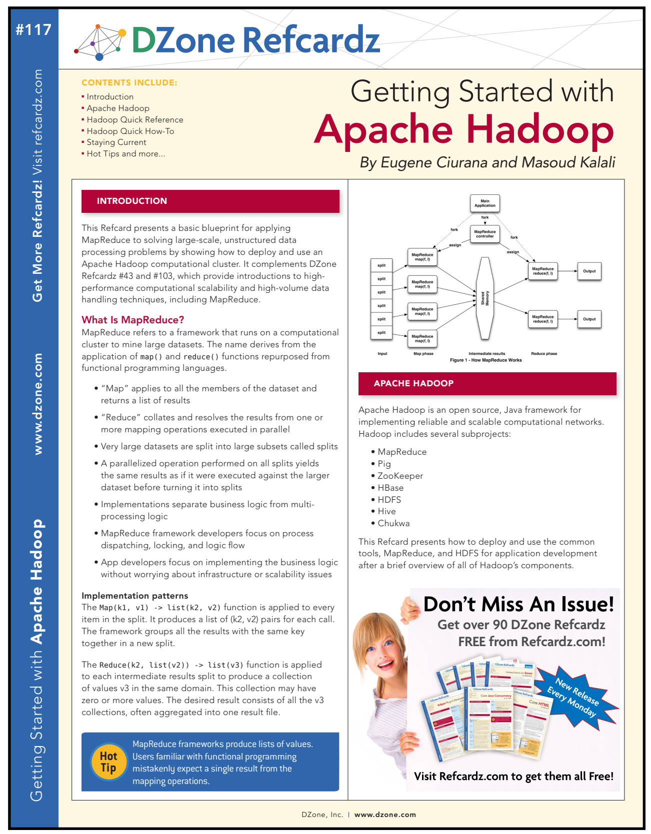
ARCHITECTURE OF HDFS

HDFS consists of following daemons that are installed and run on selected cluster nodes:

MAPR Sandbox
Take the Fastest On-Ramp to Hadoop with the MapR Sandbox
Try the sandbox FREE today at www.mapr.com/sandbox.

APACHE HADOOP

© DZONE, INC. | DZONE.COM



#117 DZone Refcardz

Get More Refcardz! Visit refcardz.com | www.dzone.com | Getting Started with Apache Hadoop

CONTENTS INCLUDE:
- Introduction
- Apache Hadoop
- Hadoop Quick Reference
- Hadoop Quick How-To
- Staying Current
- Hot Tips and more...

Getting Started with Apache Hadoop

By Eugene Ciurana and Masoud Kalali

INTRODUCTION

This Refcard presents a basic blueprint for applying MapReduce to solving large-scale, unstructured data processing problems by showing how to deploy and use an Apache Hadoop computational cluster. It complements DZone Refcardz #43 and #103, which provide introductions to high-performance computational scalability and high-volume data handling techniques, including MapReduce.

What Is MapReduce?

MapReduce refers to a framework that runs on a computational cluster to mine large datasets. The name derives from the application of `map()` and `reduce()` functions repurposed from functional programming languages.

- "Map" applies to all the members of the dataset and returns a list of results
- "Reduce" collates and resolves the results from one or more mapping operations executed in parallel
- Very large datasets are split into large subsets called splits
- A parallelized operation performed on all splits yields the same results as if it were executed against the larger dataset before turning it into splits
- Implementations separate business logic from multi-processing logic
- MapReduce framework developers focus on process dispatching, locking, and logic flow
- App developers focus on implementing the business logic without worrying about infrastructure or scalability issues

Implementation patterns

The `Map(k1, v1) -> list(k2, v2)` function is applied to every item in the split. It produces a list of (k2, v2) pairs for each call. The framework groups all the results with the same key together in a new split.

The `Reduce(k2, list(v2)) -> list(v3)` function is applied to each intermediate results split to produce a collection of values v3 in the same domain. This collection may have zero or more values. The desired result consists of all the v3 collections, often aggregated into one result file.

Hot Tip: MapReduce frameworks produce lists of values. Users familiar with functional programming mistakenly expect a single result from the mapping operations.

Figure 1 - How MapReduce Works

APACHE HADOOP

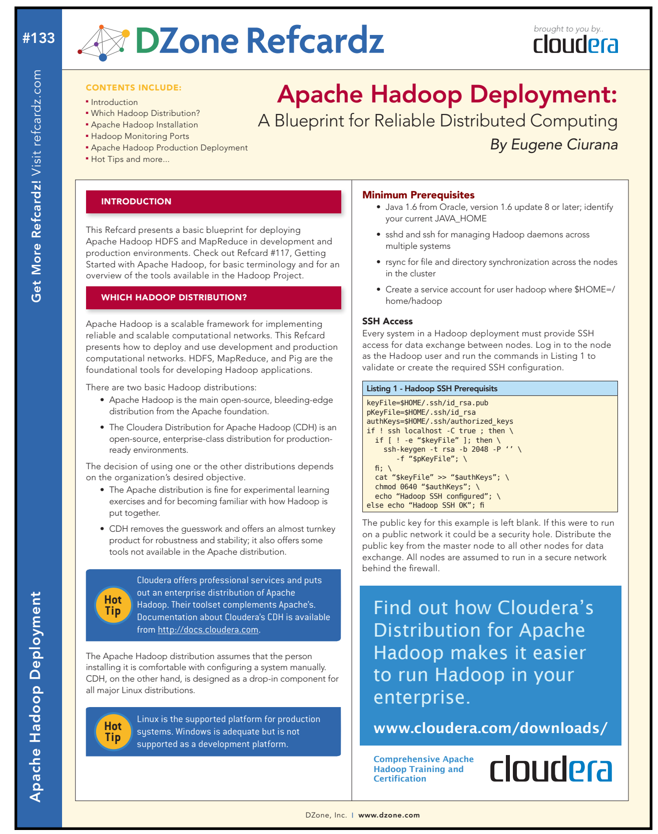
Apache Hadoop is an open source, Java framework for implementing reliable and scalable computational networks. Hadoop includes several subprojects:

- MapReduce
- Pig
- ZooKeeper
- HBase
- HDFS
- Hive
- Chukwa

This Refcard presents how to deploy and use the common tools, MapReduce, and HDFS for application development after a brief overview of all of Hadoop's components.

Don't Miss An Issue!
Get over 90 DZone Refcardz FREE from Refcardz.com!
New Release Every Monday
Visit Refcardz.com to get them all Free!

DZone, Inc. | www.dzone.com



#133 DZone Refcardz

brought to you by.. cloudera

Get More Refcardz! Visit refcardz.com | Apache Hadoop Deployment

CONTENTS INCLUDE:
- Introduction
- Which Hadoop Distribution?
- Apache Hadoop Installation
- Hadoop Monitoring Ports
- Apache Hadoop Production Deployment
- Hot Tips and more...

Apache Hadoop Deployment:

A Blueprint for Reliable Distributed Computing

By Eugene Ciurana

INTRODUCTION

This Refcard presents a basic blueprint for deploying Apache Hadoop HDFS and MapReduce in development and production environments. Check out Refcard #117, Getting Started with Apache Hadoop, for basic terminology and for an overview of the tools available in the Hadoop Project.

WHICH HADOOP DISTRIBUTION?

Apache Hadoop is a scalable framework for implementing reliable and scalable computational networks. This Refcard presents how to deploy and use development and production computational networks. HDFS, MapReduce, and Pig are the foundational tools for developing Hadoop applications.

There are two basic Hadoop distributions:

- Apache Hadoop is the main open-source, bleeding-edge distribution from the Apache foundation.
- The Cloudera Distribution for Apache Hadoop (CDH) is an open-source, enterprise-class distribution for production-ready environments.

The decision of using one or the other distributions depends on the organization's desired objective.

- The Apache distribution is fine for experimental learning exercises and for becoming familiar with how Hadoop is put together.
- CDH removes the guesswork and offers an almost turnkey product for robustness and stability; it also offers some tools not available in the Apache distribution.

Hot Tip: Cloudera offers professional services and puts out an enterprise distribution of Apache Hadoop. Their toolset complements Apache's. Documentation about Cloudera's CDH is available from http://docs.cloudera.com.

The Apache Hadoop distribution assumes that the person installing it is comfortable with configuring a system manually. CDH, on the other hand, is designed as a drop-in component for all major Linux distributions.

Hot Tip: Linux is the supported platform for production systems. Windows is adequate but is not supported as a development platform.

Minimum Prerequisites

- Java 1.6 from Oracle, version 1.6 update 8 or later; identify your current JAVA_HOME
- sshd and ssh for managing Hadoop daemons across multiple systems
- rsync for file and directory synchronization across the nodes in the cluster
- Create a service account for user hadoop where $HOME=/home/hadoop

SSH Access

Every system in a Hadoop deployment must provide SSH access for data exchange between nodes. Log in to the node as the Hadoop user and run the commands in Listing 1 to validate or create the required SSH configuration.

Listing 1 - Hadoop SSH Prerequisits

```
keyFile=$HOME/.ssh/id_rsa.pub
pKeyFile=$HOME/.ssh/id_rsa
authKeys=$HOME/.ssh/authorized_keys
if ! ssh localhost -C true ; then \
  if [ ! -e "$keyFile" ]; then \
    ssh-keygen -t rsa -b 2048 -P '' \
      -f "$pKeyFile"; \
  fi; \
  cat "$keyFile" >> "$authKeys"; \
  chmod 0640 "$authKeys"; \
  echo "Hadoop SSH configured"; \
else echo "Hadoop SSH OK"; fi
```

The public key for this example is left blank. If this were to run on a public network it could be a security hole. Distribute the public key from the master node to all other nodes for data exchange. All nodes are assumed to run in a secure network behind the firewall.

Find out how Cloudera's Distribution for Apache Hadoop makes it easier to run Hadoop in your enterprise.
www.cloudera.com/downloads/

Comprehensive Apache Hadoop Training and Certification — cloudera

DZone, Inc. | www.dzone.com

#159 DZone Refcardz

Get More Refcardz! Visit refcardz.com | Apache HBase

CONTENTS INCLUDE:
- Configuration
- Start/Stop
- HBase Shell
- Java API
- Web UI: Master & Slaves
- and More!

Apache HBase

The NoSQL Database for Hadoop and Big Data

By Alex Baranau and Otis Gospodnetić

ABOUT HBASE

HBase is the Hadoop database. Think of it as a distributed, scalable Big Data store.

Use HBase when you need random, real-time read/write access to your Big Data. The goal of the HBase project is to host very large tables — billions of rows multiplied by millions of columns — on clusters built with commodity hardware. HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable. Just as Bigtable leverages the distributed data storage provided by the Google File System, HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

CONFIGURATION

OS & Other Pre-requisites

HBase uses the local hostname to self-report its IP address. Both forward- and reverse-DNS resolving should work.

HBase uses many files simultaneously. The default maximum number of allowed open-file descriptors (1024 on most *nix systems) is often insufficient. Increase this setting for any Hbase user.

The nproc setting for a user running HBase also often needs to be increased — when under a load, a low nproc setting can result in the OutOfMemoryError.

Because HBase depends on Hadoop, it bundles an instance of the Hadoop jar under its /lib directory. The bundled jar is ONLY for use in standalone mode. In the distributed mode, it is critical that the version of Hadoop on your cluster matches what is under HBase. If the versions do not match, replace the Hadoop jar in the HBase /lib directory with the Hadoop jar from your cluster.

To increase the maximum number of files HDFS DataNode can serve at one time in hadoop/conf/hdfs-site.xml, just do this:

```
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>4096</value>
</property>
```

hbase-env.sh

You can set HBase environment variables in this file.

Env Variable	Description
HBASE_HEAPSIZE	Shows the maximum amount of heap to use, in MB. Default is 1000. It is essential to give HBase as much memory as you can (avoid swapping!) to achieve good performance.
HBASE_OPTS	Shows extra Java run-time options. You can also add the following to watch for GC: export HBASE_OPTS="$HBASE_OPTS -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps $HBASE_GC_OPTS"

hbase-site.xml

Specific customizations go into this file in the following file format:

```
<configuration>
  <property>
    <name>property_name</name>
    <value>property_value</value>
  </property>
  …
</configuration>
```

For the list of configurable properties, refer to http://hbase.apache.org/book.html#hbase_default_configurations (or view the raw /conf/hbase-default.xml source file).

These are the most important properties:

Property	Value	Description
hbase.cluster.distributed	true	Set value to true when running in distributed mode.
hbase.zookeeper.quorum	my.zk.server1,my.zk.server2,	HBase depends on a running ZooKeeper cluster. Configure using external ZK. (If not configured, internal instance of ZK is started.)
hbase.rootdir	hdfs://my.hdfs.server/hbase	The directory shared by region servers and where HBase persists. The URL should be 'fully qualified' to include the filesystem scheme.

START/STOP

Running Modes

HBase Cluster: client, ZooKeeper, HMaster, RegionServer

AnswerHub Social Q&A for the Enterprise
1/2 of the Top 10 StackExchange 1.0 Sites Now Run on AnswerHub
Discover Why Now!

DZone, Inc. | www.dzone.com

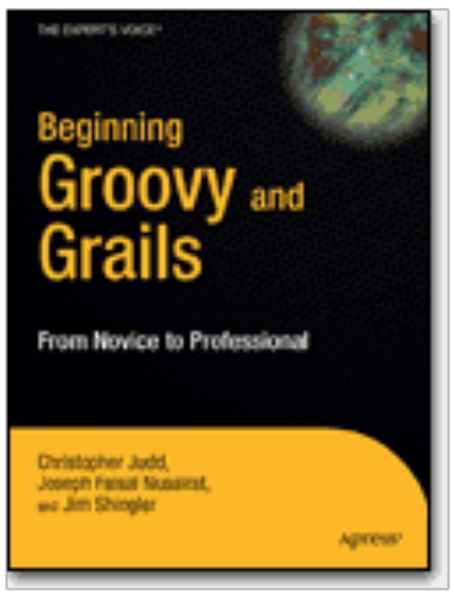
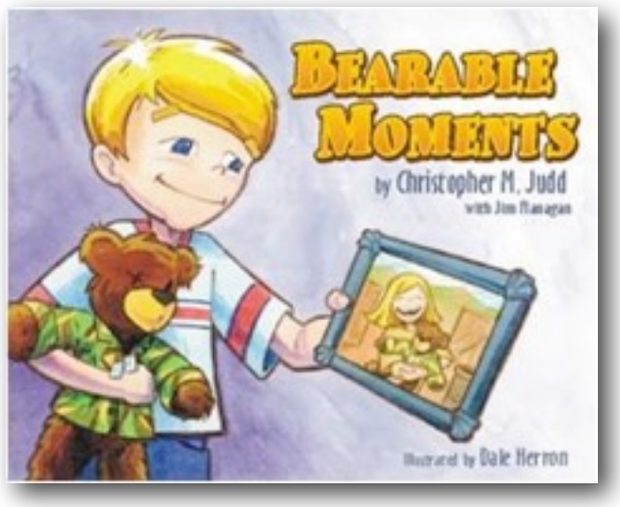
Christopher M. Judd

Manifest Solutions

CTO and Partner

email: cjudd@juddsolutions.com

web: www.juddsolutions.com

blog: juddsolutions.blogspot.com

twitter: javajudd