# Tutorial Hadoop untuk Pemula

Last updated on Jan 22, 2020

Jumlah data telah meningkat pesat dalam satu dekade terakhir. Ini termasuk volume besar dari berbagai format data yang dibangkitkan pada kecepatan sangat tinggi. Pada masa awal, bukanlah tugas yang berat untuk mengelola data, tetapi dengan meningkatnya data, telah menjadi lebih sulit untuk menyimpan, memroses, dan menganalisisnya. Data demikian dikenal sebagai Big Data. Bagaimana kita mengelola big data? Gunakan Hadoop — suatu framework yang dapat digunakan untuk menyimpan (*store*), memroses dan menganalisis big data. Dalam tutorial ini kita akan mendiskusikan hal-hal berikut:

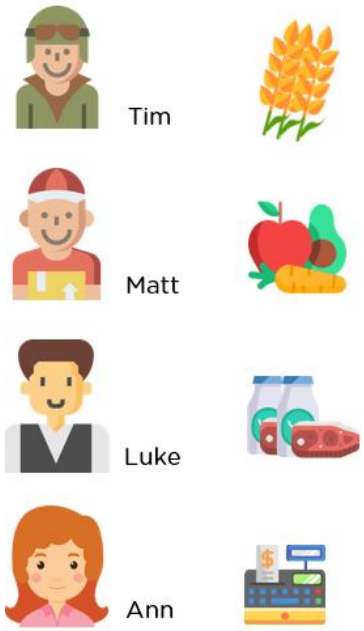
1. Mengapa Hadoop?
2. Apa itu Hadoop?
3. Hadoop HDFS
4. Hadoop MapReduce
5. Hadoop YARN
6. Kasus penggunaan Hadoop
7. Demo HDFS, MapReduce, dan YARN

## Analogi

Sekarang, marilah kita mencoba untuk memahami big data dan mengapa Hadoop diperlukan melalui suatu analogi yang sederhana. Bayangkan suatu skenario dimana kita mempunyai seorang *shopkeeper* (pemilik toko) bernama Tim yang menjual padi-padian. Para pelanggannya senang karena Tim sangat cepat menangani pesanan mereka.

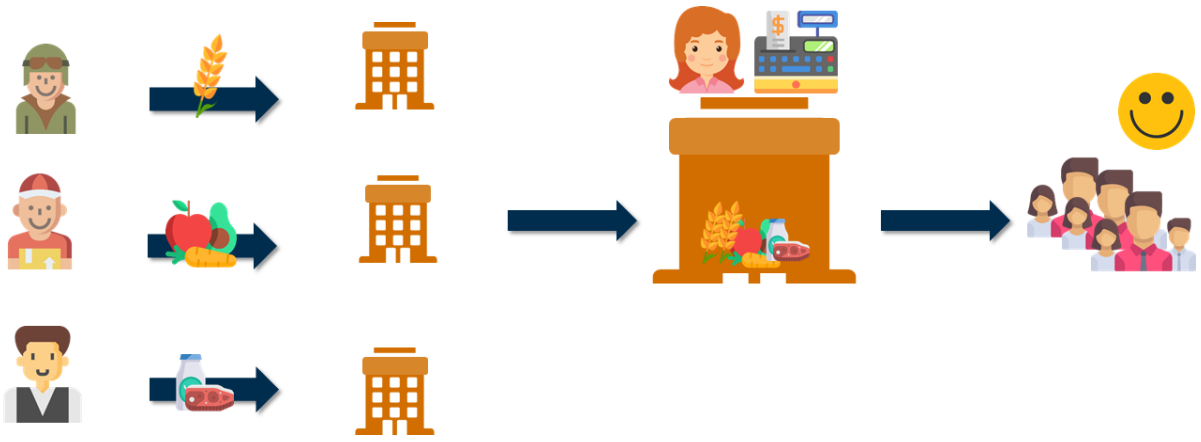
Setelah beberapa waktu, Tim merasakan adanya permintaan mendesak terhadap produk lain, sehingga dia berpikir untuk memperluas bisnisnya. Bersama dengan padi-padian, dia mulai menjual buah-buahan, sayur-sayuran, daging, dan produk susu (*dairy*). Saat jumlah pelanggannya meningkat, Tim mengalami kesulitan untuk menjaga kualitas pelayanan (yaitu menangani pesanan yang banyak dan terus-menerus).

Untuk mengatasi situasi ini, Tim memutuskan untuk mempekerjakan tiga orang (selain dirinya) untuk membantunya menyelesaikan pekerjaan. Mereka adalah Matt yang bertanggungjawab menangani bagian buah-buahan dan sayuran, Luke untuk menangani produk susu dan daging, dan terakhir Ann yang ditunjuk untuk menyelesaikan urusan pembayaran sebagai kasir.




Tetapi ini tidak menyelesaikan semua permasalahan Tim. Meskipun mempunyai tenaga kerja yang diperlukan, dia mulai kehabisan ruang (*running out of space*) dalam tokonya untuk menyimpan barang-barang yang diperlukan untuk memenuhi permintaan yang terus meningkat.

Tim menyelesaikan ini dengan mendistribusikan ruang antar lantai berbeda dari bangunan tokonya. Padi-padian dijual di lantai dasar, buah-buahan dan sayuran di lantai satu, produk susu (*dairy*) dan daging pada lantai dua, dan seterusnya.



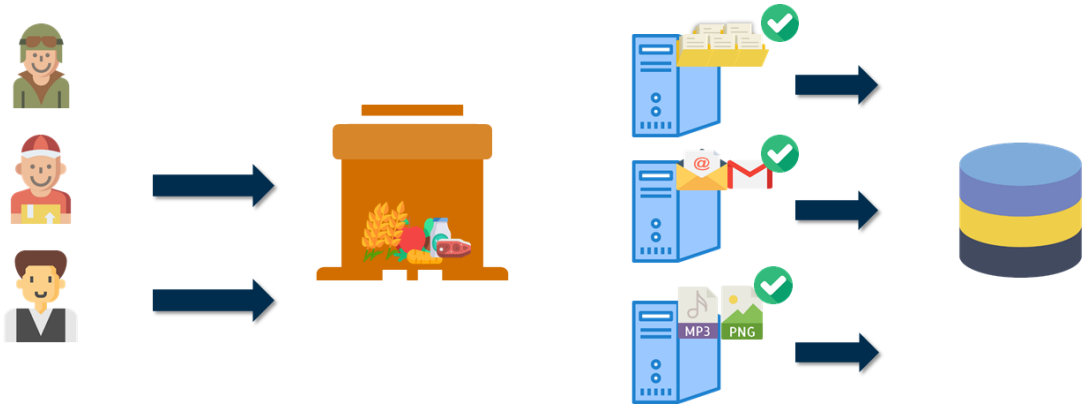
Inilah bagaimana Tim menuntaskan masalahnya; sekarang mari kita melihat bagaimana cerita ini dapat dibandingkan dengan big data dan Hadoop.

Dulu, pembangkitan data terbatas hanya untuk satu format. Data tersebut dapat dikelola dengan hanya satu unit *storage* dan satu *processor*. Generasi data secara bertahap mulai meningkat dan

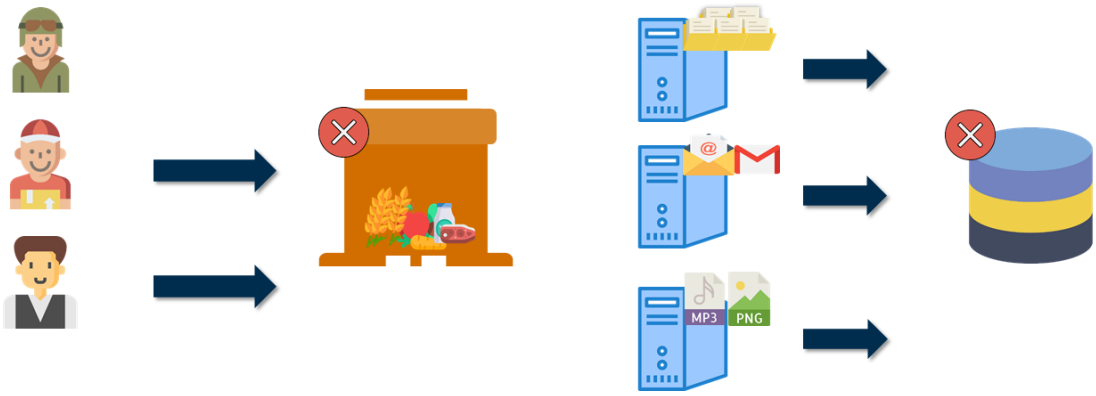
variasi baru dari data telah hadir. Karena data dihasilkan dengan kecepatan tinggi maka ini mengakibatkan lebih sulit untuk ditangani oleh satu prosesor.

Ini mirip dengan bagaimana Tim menemukan kesulitan dalam mengelola sendiri bisnisnya yang baru saja diperluas.

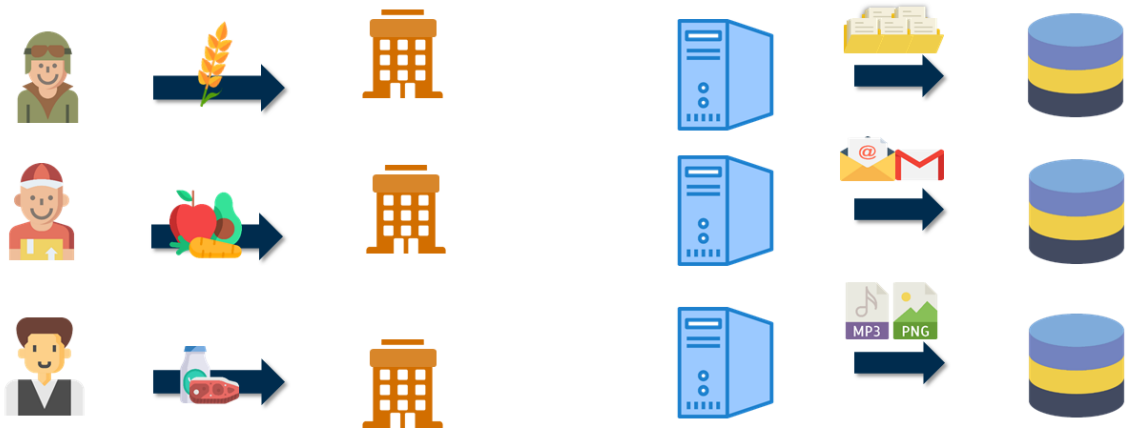

Jadi seperti bagaimana Tim memecahkan masalah ini dengan menambah tenaga kerja, banyak prosesor dapat digunakan untuk memproses setiap jenis data.



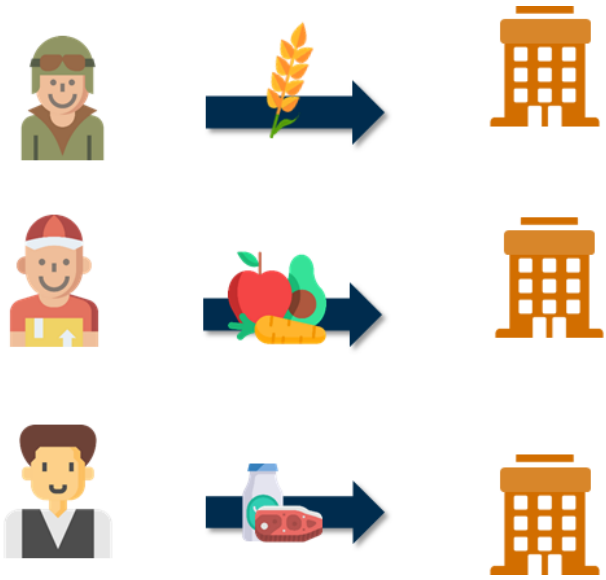

Namun, menjadi sulit bagi banyak prosesor untuk mengakses unit penyimpanan yang sama.



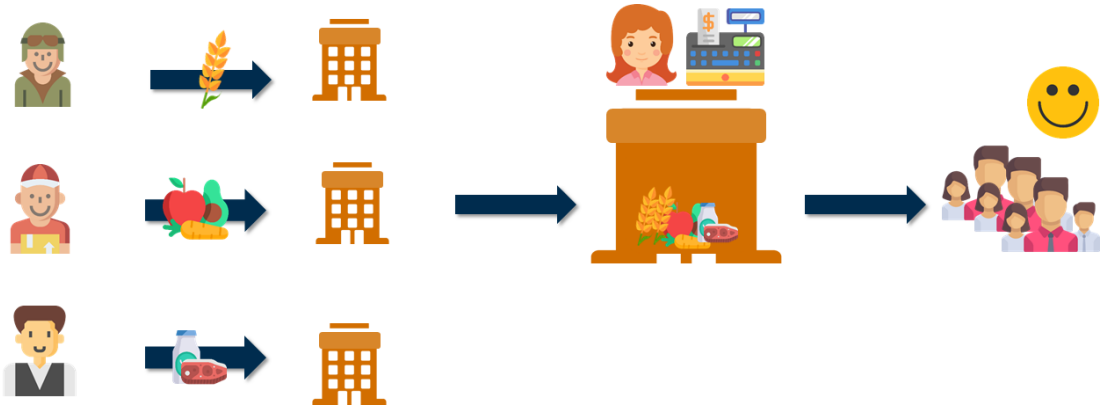

Akhirnya, seperti cara Tim mengadopsi pendekatan penyimpanan barang terdistribusi, sistem penyimpanan data juga dapat didistribusikan, dan dengan melakukan itu, data disimpan dalam basis data individual..




Kisah ini membantu Kita memahami bagaimana dua komponen utama dari Hadoop: HDFS dan MapReduce. HDFS mengacu pada ruang penyimpanan terdistribusi.

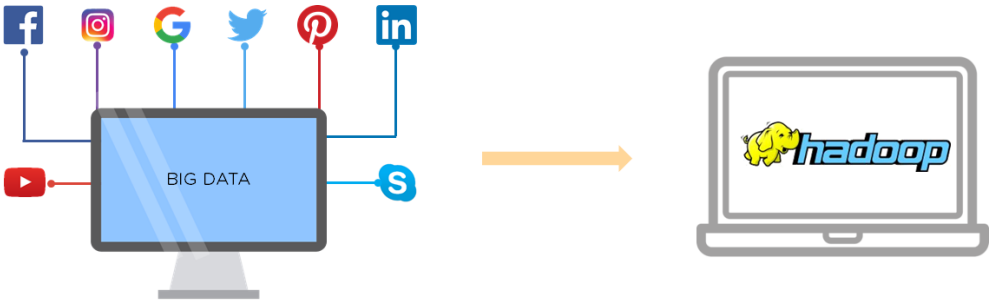


MapReduce, di sisi lain, analog dengan bagaimana setiap orang mengurus bagian yang terpisah, dan pada akhirnya pelanggan pergi ke kasir untuk penagihan akhir. Ini mirip dengan fase pengurangan (*reduce*).



# Hadoop

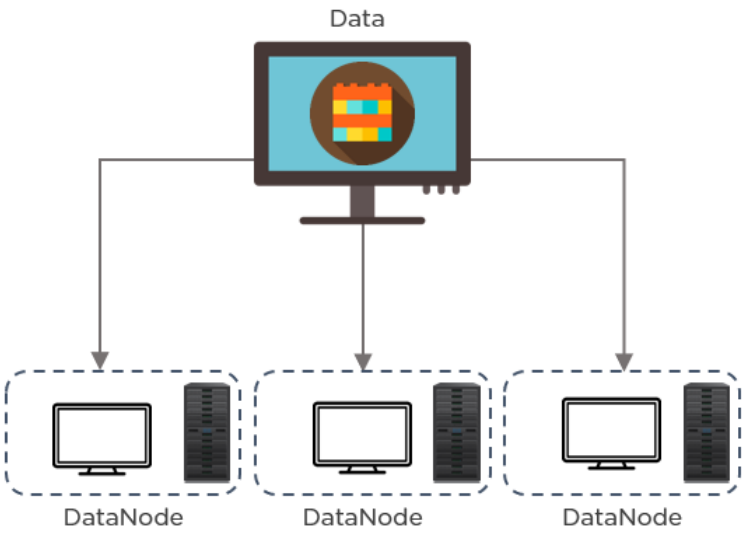
Hadoop adalah kerangka kerja yang menyimpan dan memproses data besar secara terdistribusi dan paralel.




Seperti yang disebutkan sebelumnya, Hadoop memiliki komponen individual untuk menyimpan dan memproses data. Mari kita pelajari lebih lanjut tentang lapisan penyimpanan Hadoop: ***Hadoop Distributed File System*** (HDFS).

# Hadoop HDFS

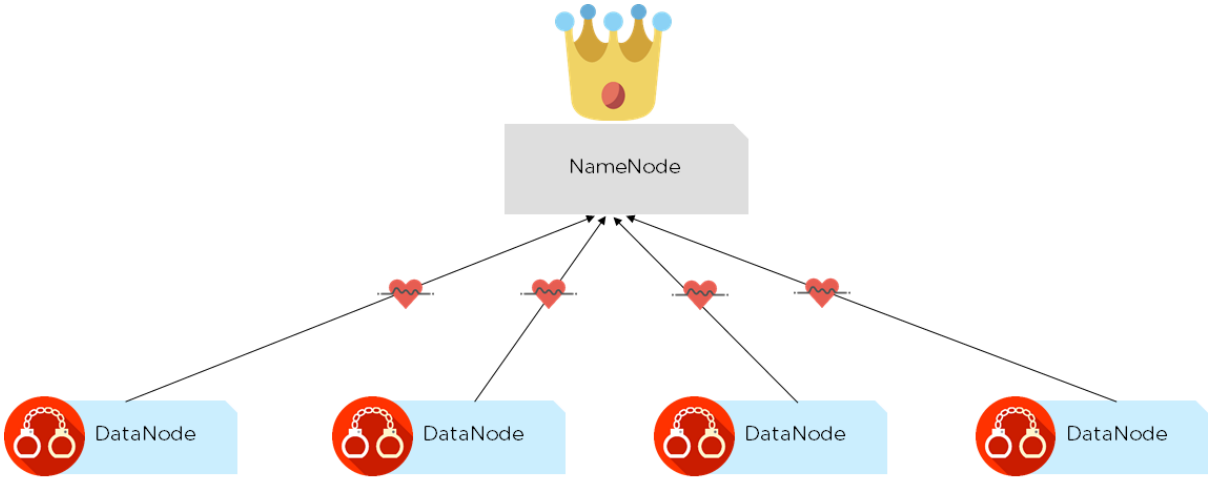
HDFS mirip dengan Google File System (GFS) karena menyimpan data di beberapa mesin. Data direplikasi secara otomatis ke berbagai mesin untuk mencegah hilangnya data. Dalam HDFS, data dipecah menjadi beberapa blok; masing-masing blok ini memiliki ukuran default 128 MB.



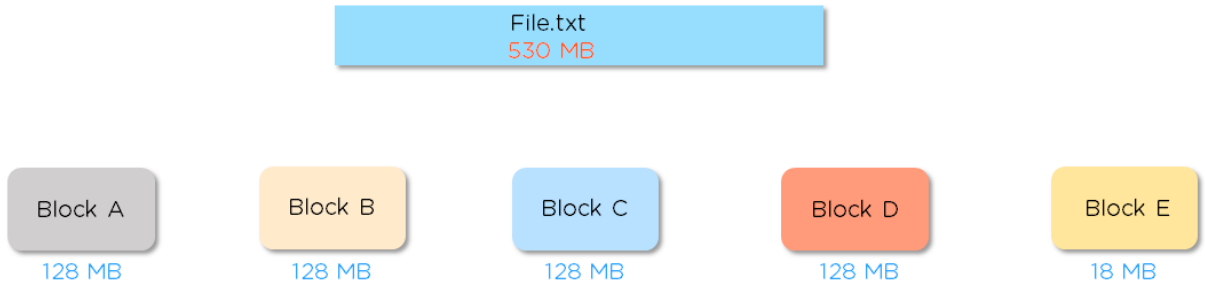

Jadi, bagaimana ini berbeda dari praktik penyimpanan tradisional? Perbedaannya adalah bahwa dalam sistem tradisional, semua data disimpan dalam satu basis data. Ini dapat menjadi masalah - jika database macet (*crash*); semua data akan hilang. Ini juga membebani database, dan sistem demikian sangat tidak toleran terhadap kesalahan (tidak *fault tolerance*). Masalah ini diatasi oleh HDFS dengan mendistribusikan data di antara beberapa mesin. Ini dirancang khusus untuk menyimpan kumpulan data besar-besaran di perangkat keras komoditas yang berarti kita dapat memiliki banyak mesin untuk digunakan (diskalakan). Selanjutnya, HDFS memiliki dua komponen yang berjalan di banyak mesin. yaitu:

1. **NameNode**. NameNode adalah master dari lapisan penyimpanan HDFS. Mesin ini menyimpan semua metadata. Jika mesin tempat proses NameNode lumpuh, gugus (*cluster*) tidak akan tersedia.
2. **DataNode**. DataNodes dikenal juga sebagai slave node. Mesin-mesin ini menyimpan data aktual, dan mereka melakukan operasi baca / tulis.

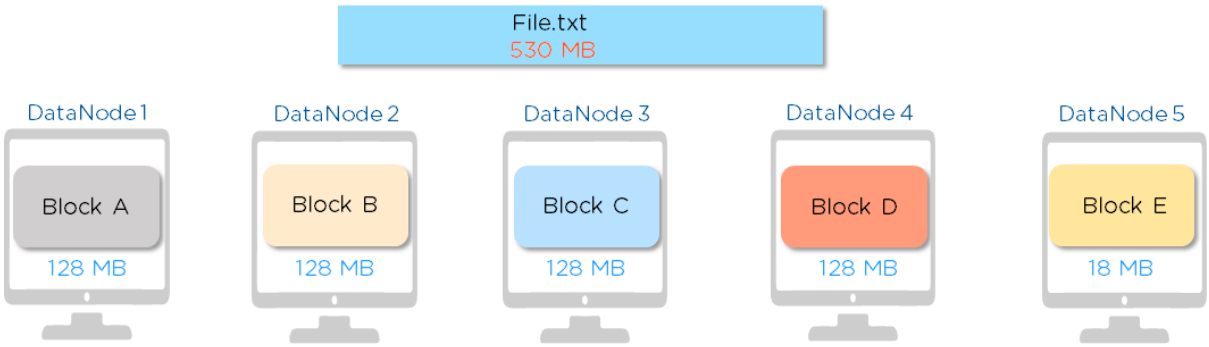
Pada dasarnya, NameNode mengelola semua DataNodes. Sinyal yang dikenal sebagai detak jantung (*heartbeats*) dikirim secara periodik oleh DataNodes ke NameNode untuk memberikan pembaruan status.




Sekarang, mari kita lihat bagaimana data dipecah (*split*) dalam HDFS.



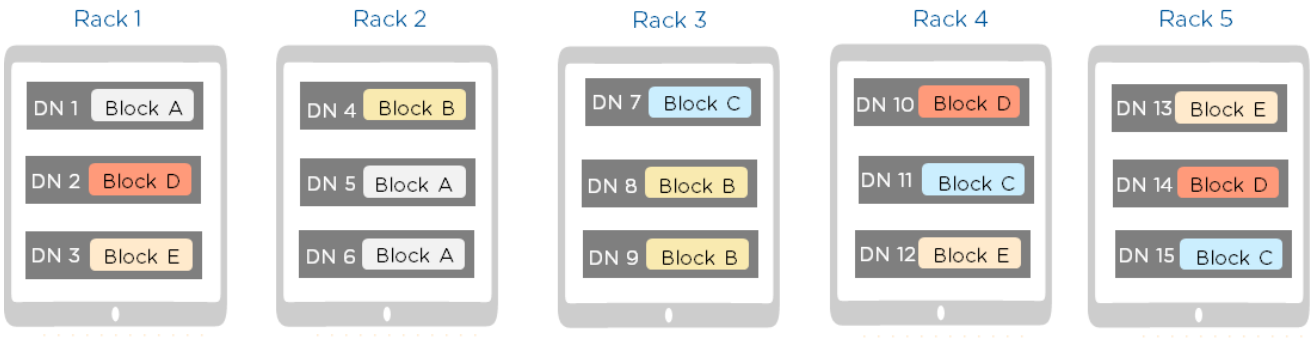

Seperti yang dapat terlihat dari contoh di atas, kita memiliki file berukuran 530 MB. File ini tidak akan disimpan apa adanya; tetapi akan dipecah menjadi lima blok yang berbeda (ukuran blok default 128MB). Blok terakhir hanya akan menggunakan ruang yang tersisa untuk penyimpanan. Blok data disimpan dalam beberapa DataNodes yang pada dasarnya hanya komputer.




Jadi, apa yang terjadi jika komputer yang berisi Blok A lumpuh? Apakah kita akan kehilangan data kita? TIDAK: itulah keindahan HDFS; salah satu fitur utama HDFS adalah replikasi data.

HDFS membuat salinan data di beberapa mesin, dan dengan cara ini jika komputer yang memegang blok A *crash*, data kita akan aman pada komputer yang lain. Faktor replikasi default dalam HDFS adalah tiga. Ini berarti secara keseluruhan kita memiliki tiga salinan dari setiap blok data.

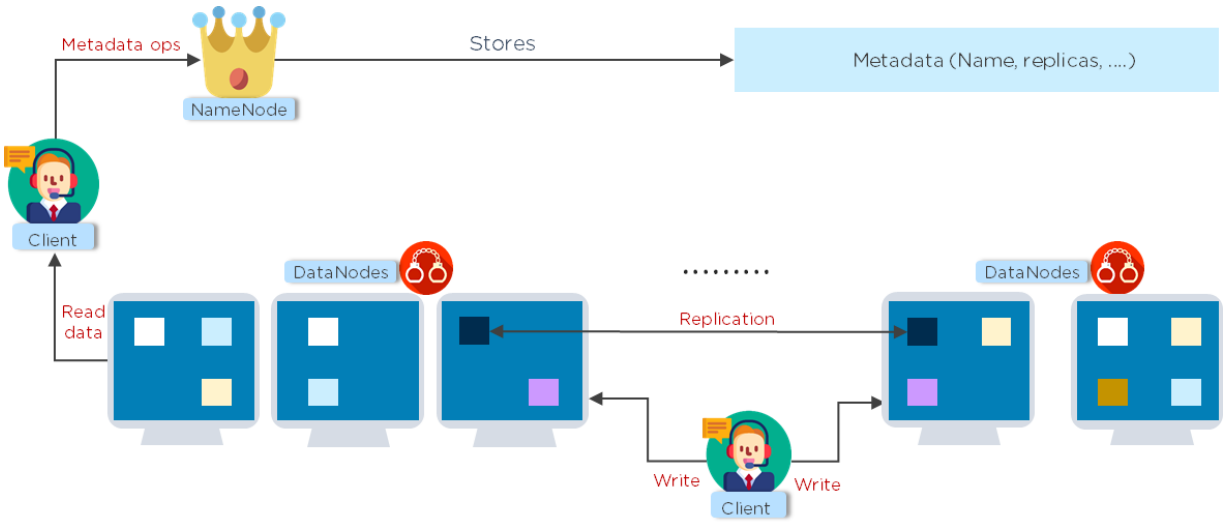
Mari kita lihat lebih dekat konsep dari replikasi. Konsep Kesadaran Rak (*Rack Awareness*) membantu untuk memutuskan di mana replika blok data harus disimpan. Di sini, rak mengacu pada koleksi 30-40 DataNodes. Menurut aturan replikasi, blok data dan salinannya tidak dapat disimpan pada DataNode yang sama.




Dari gambar di atas, dapat dilihat bahwa kita memiliki Blok A di rak 1 dan rak 2.

Sudah menjadi aturan, kita tidak dapat memiliki blok dan replikanya semua berada di rak yang sama. Namun, sebagai contoh dari Blok D, juga tidak ideal untuk memiliki blok yang tersebar di semua rak, karena akan meningkatkan kebutuhan *bandwidth*. Oleh karena itu, jika cluster yang dibangun bersifat sadar rak, sebaran yang bagus adalah seperti terlihat pada Blok A dan Blok B. Di sini, blok data tidak semua disimpan pada rak yang sama, karena itu jika satu rak *crash* maka kita tidak kehilangan data karena kita memiliki salinan di rak lain. Inilah cara HDFS menyediakan toleransi kesalahan. Harus diingat bahwa ukuran blok dan faktor replikasi dapat disesuaikan.

Sekarang mari kita beralih ke arsitektur HDFS. Gambar di bawah ini menunjukkan bagaimana HDFS beroperasi. Kita memiliki NameNode, DataNodes, dan permintaan klien (*request*).




Pada dasarnya sistem file HDFS menyediakan dua operasi, yaitu operasi membaca (*read*) dan menulis (*write*).

Pertama, NameNode menyimpan semua metadata dalam RAM-nya dan juga dalam disk-nya. Ketika *cluster* tertentu dimulai, kita memiliki DataNodes dan NameNode yang aktif. Seperti yang terlihat sebelumnya, DataNodes akan mulai mengirimkan detak jantung ke NameNode setiap tiga detik begitu mereka aktif. Ini akan terdaftar dalam RAM NameNode. Pindah ke disk NameNode, kita memiliki informasi pemformatan yang diatur saat memulai cluster. Jika *cluster* tersebut sadar rak, maka semua blok data tidak akan berada di rak yang sama. Jika klien ingin membaca data, permintaan masuk ke NameNode. NameNode kemudian akan melihat metadata terkait dan mencari di mana informasi itu berada. Setelah selesai, klien dapat membaca data dari DataNodes yang mendasarinya. Ini terjadi secara paralel dari beberapa DataNodes.

Ketika datang *request* untuk menulis data, prosesnya mirip dengan operasi baca. Klien meminta kepada NameNode dan kemudian NameNode mencari DataNodes yang tersedia. Setelah daftar siap, klien menuliskan data ke dalam DataNodes yang diberikan oleh NameNode. Inilah cara HDFS menangani operasi baca dan tulis.

Berikut ini adalah beberapa fitur unggulan dari HDFS:

1. HDFS toleran terhadap kesalahan (*fault tolerance*) karena ada banyak salinan data yang dibuat
2. HDFS menyediakan enkripsi ujung ke ujung (*end-to-end encryption*) untuk melindungi data pengguna
3. Dalam HDFS, beberapa node dapat ditambahkan ke *cluster* tergantung pada kebutuhan
4. Hadoop HDFS fleksibel dalam menyimpan semua jenis data, seperti data terstruktur, semi-terstruktur atau tidak terstruktur

Sekarang semua data disimpan dalam HDFS, langkah selanjutnya adalah mengolahnya untuk mendapatkan informasi yang bermakna. Untuk menyelesaikan pemrosesan, kita menggunakan Hadoop MapReduce.

# Hadoop MapReduce

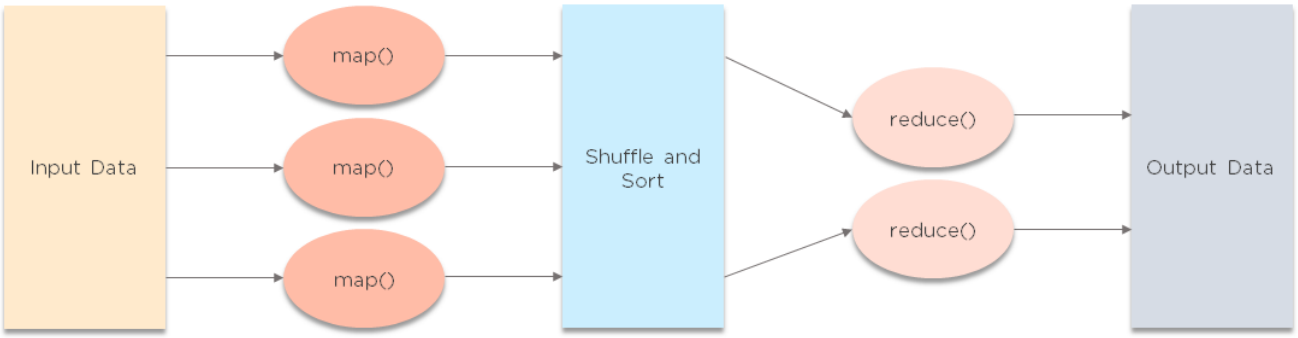
Mengapa MapReduce diperlukan sejak awal hadirnya Hadoop? Dalam pendekatan tradisional, data besar diproses di master node. Ini adalah kerugian, karena dibutuhkan lebih banyak waktu untuk memroses berbagai jenis data. Untuk mengatasi masalah ini, data diproses pada setiap slave node, dan hasil akhirnya dikirim ke master node.

Inilah gunanya MapReduce. Di sini data diproses di mana pun data tersebut disimpan. MapReduce didefinisikan sebagai model pemrograman di mana sejumlah besar data diproses secara paralel dan terdistribusi. Namun, kerangka kerja MapReduce tidak bergantung pada satu bahasa tertentu. Kode program MapReduce dapat ditulis dalam Java, Python, atau bahasa pemrograman lainnya.

Seperti namanya, MapReduce terdiri dari dua tugas (*task*):

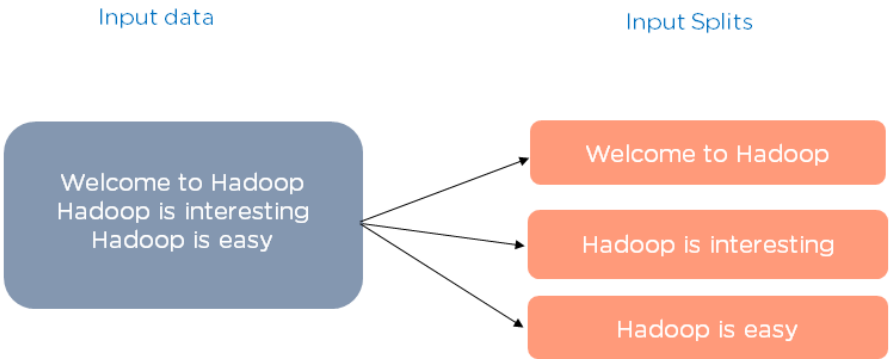
1. Map tasks
2. Reduce tasks

Mapper adalah fungsi yang menangani fase pemetaan dan fungsi Reducer menangani fase reduksi. Kedua fungsi ini menjalankan tugas Map dan Reduce secara internal. Dari diagram berikut dapat dilihat bagaimana setiap langkah dieksekusi di dalam MapReduce.

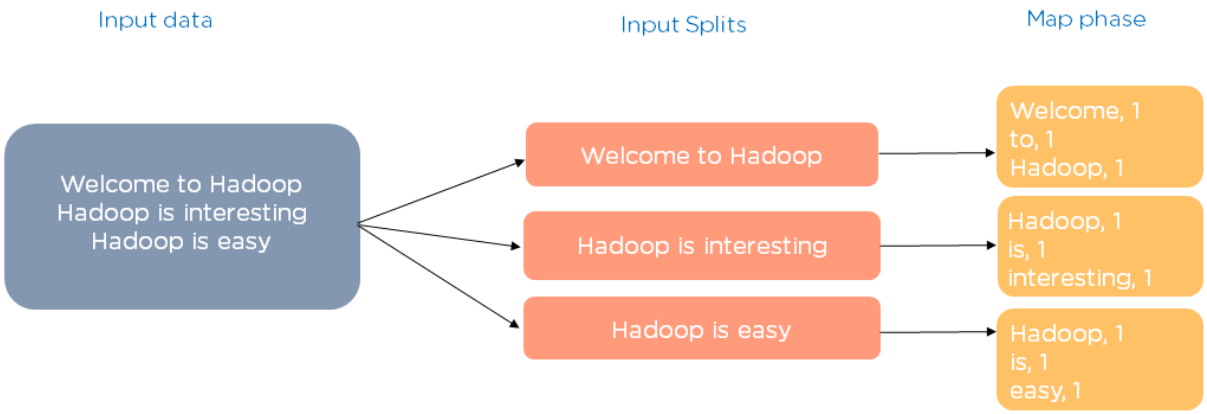



Pertama, data input akan dibagi (di*split*) menjadi sejumlah blok data. Pada fase pemetaan, fungsi mapper yang terdiri dari beberapa kode akan berjalan pada satu atau lebih (banyak) split. Hasil dari fase pemataan akan masuk ke fase pengocokan (*shuffling*) dan pengurutan (*sorting*), di mana output dari fase pemetaan akan dikelompokkan untuk diproses lebih lanjut. Kemudian dalam fase pengurangan (*reduce*) hasilnya dikumpulkan dan nilai output tunggal dikirimkan. Pada framework ini pengembang menyediakan tugas ***mapper*** dan ***reducer***. Framework MapReduce sendiri menangani *shuffling*, pengurutan, dan pemartisian. Dalam contoh berikut, data input akan dipecah, di*shuffle*, dan digabungkan (*aggregate*) untuk mendapatkan hasil akhir. Mari kita lihat, langkah demi langkah:

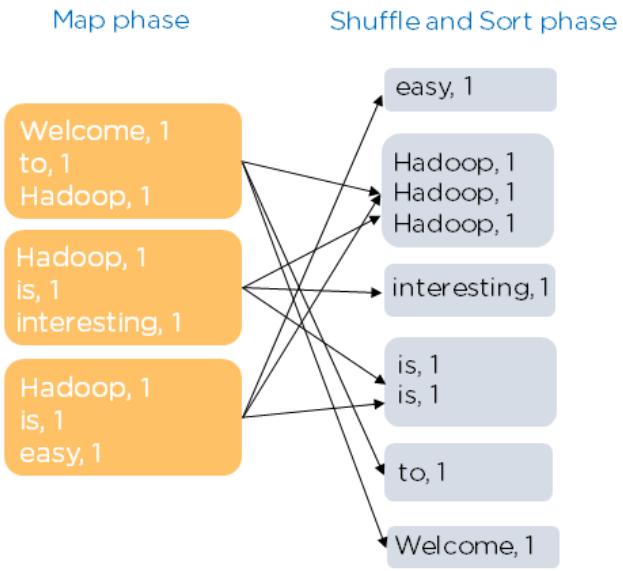
i) Data input displit, baris demi baris (pada contoh menjadi 3 split).



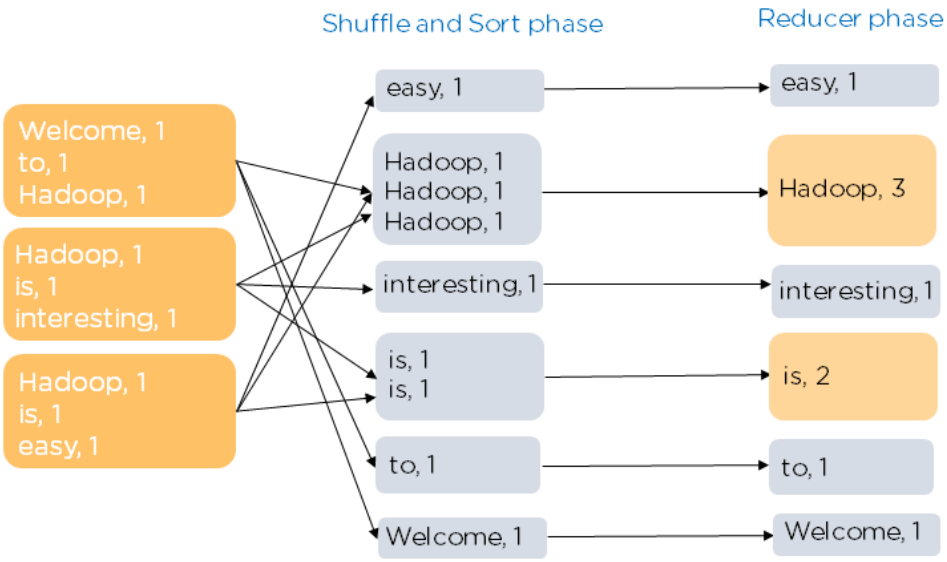

ii) Fungsi ***mapper*** kemudian bekerja pada setiap input split yang bekerja mirip dengan model jumlah kata. Data dipetakan ke pasangan (kunci, nilai). Di sini, kita memiliki kata sebagai kunci dan nilainya sebagai satu.



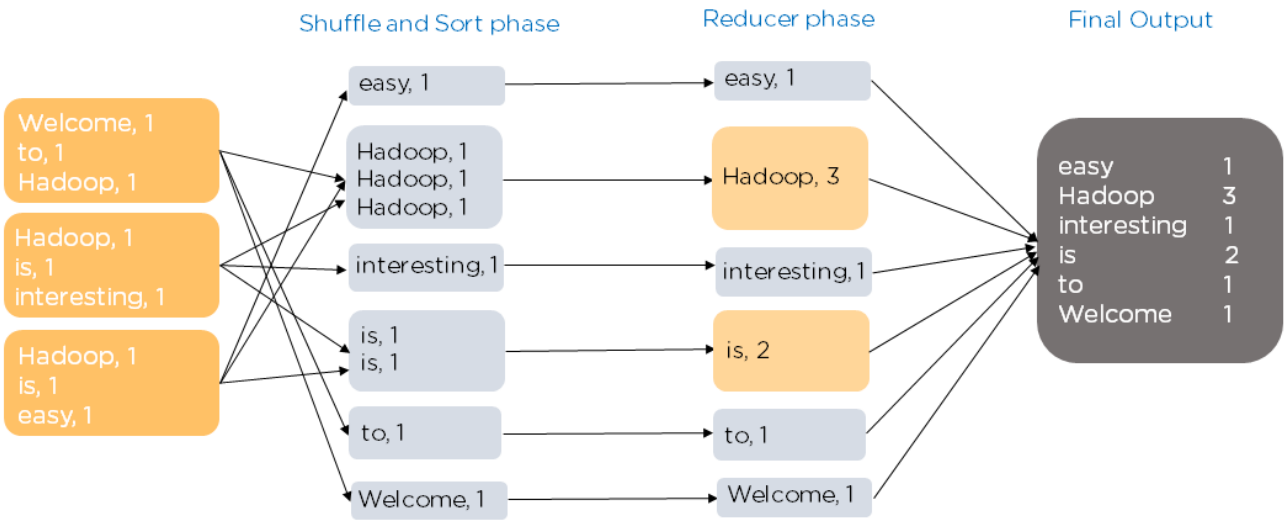

iii) Pada langkah berikutnya, data dikocok dan disortir untuk mendapatkan kunci (kata) yang sama.




iv) Fase ***reducer*** mengumpulkan nilai untuk kunci yang serupa.




v) Akhirnya, output akan terdiri dari daftar kata dan nilainya yang menampilkan jumlah kemunculannya.




Ini adalah contoh bagaimana tugas-tugas MapReduce dilakukan. Beberapa fitur MapReduce termasuk:

1. Penyeimbangan muatan (*load balancing*) ditingkatkan karena tahapan dibagi ke dalam Map dan Reduce.
2. Ada eksekusi ulang otomatis jika tugas tertentu gagal
3. MapReduce memiliki salah satu model pemrograman paling sederhana, yang didasarkan pada Java.

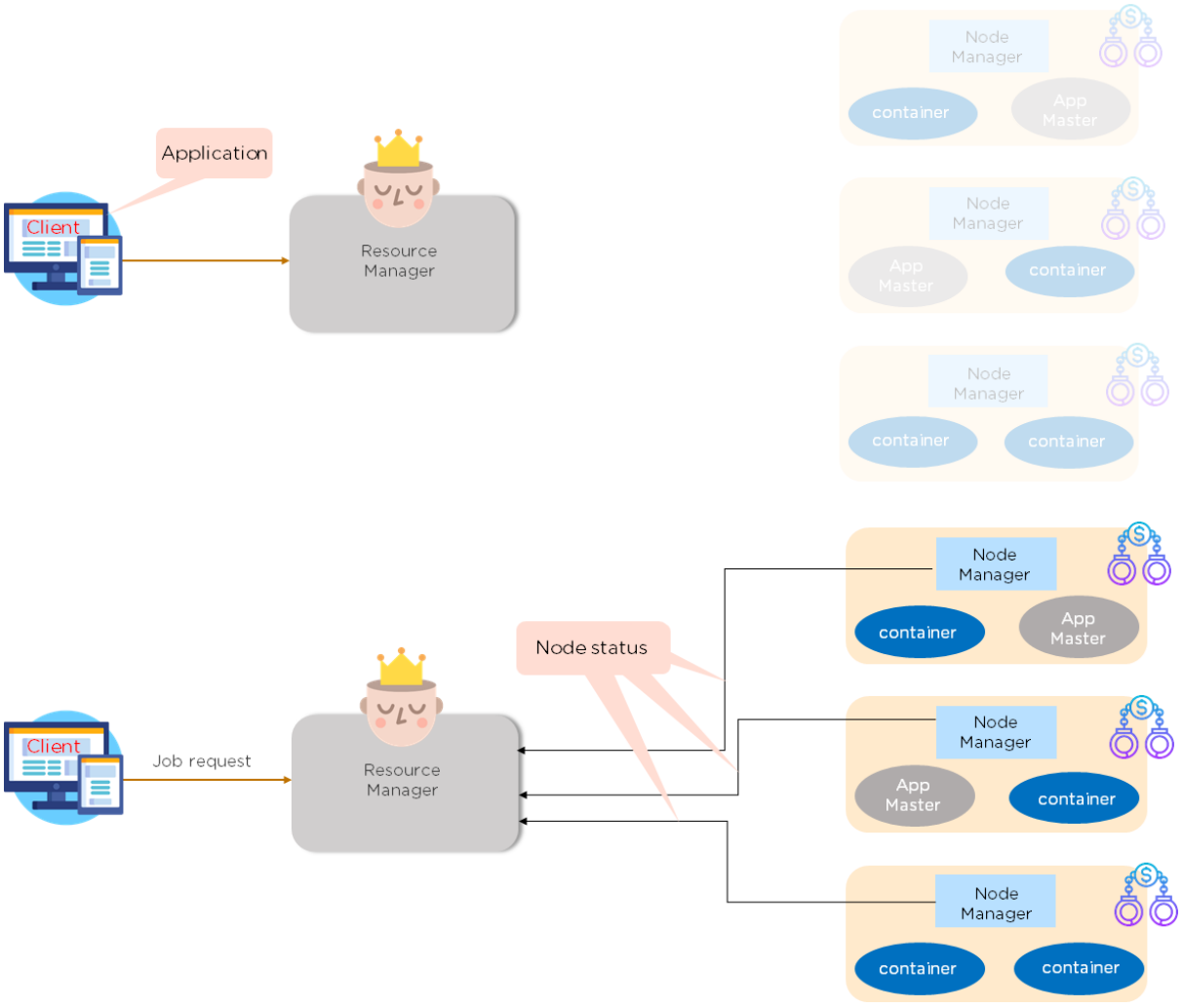
HDFS dan MapReduce adalah dua unit Hadoop 1.0. Versi ini memiliki masalah karena JobTracker melakukan pemrosesan data dan alokasi sumber daya. Hal ini mengakibatkan JobTracker menjadi terbebani. Untuk mengatasi masalah ini, Hadoop 2.0 memperkenalkan YARN sebagai lapisan (*layer*) pemrosesan.

# Hadoop YARN - Yet Another Resource Negotiator

Di bagian sebelumnya, kita membahas bagaimana data disimpan dan diproses. Tetapi bagaimana unit alokasi sumber daya diurus? Bagaimana sumber daya dinegosiasikan di seluruh *cluster*? YARN menangani ini dan bertindak sebagai unit manajemen sumber daya Hadoop. Apache YARN terdiri dari:

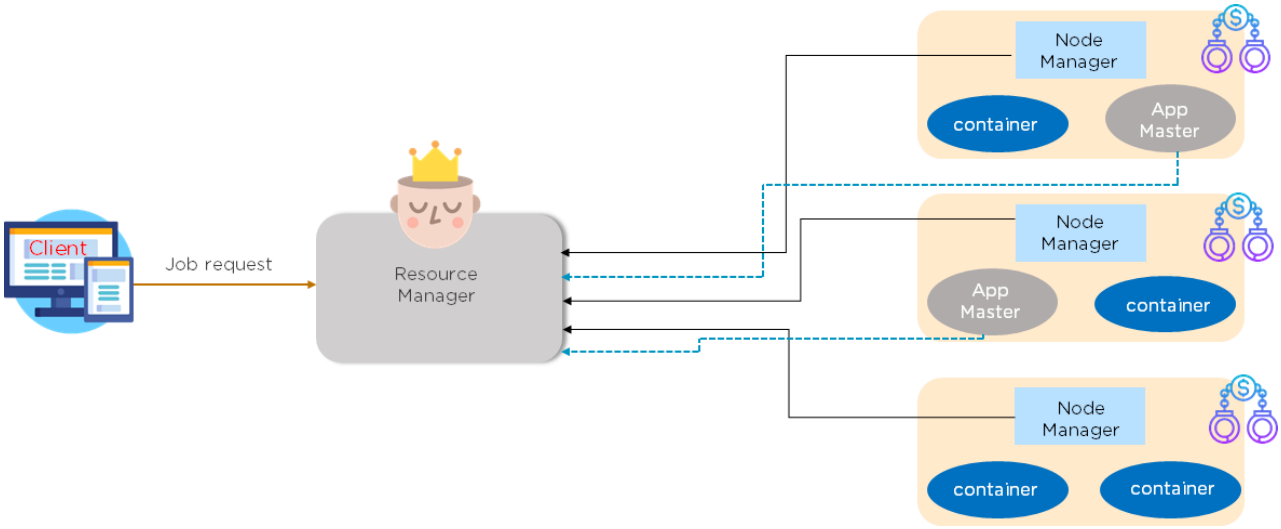
1. **Resource Manager** yang bertindak sebagai master daemon dan bertanggungjawab dalam penugasan CPU, memori, dll.
2. **Node Manager** yang bekerja sebagai *slave daemon* dan melaporkan pekerjaannya kepada Resource Manager.
3. **Application Master** yang bekerja dengan Resource Manager dan Node Manager dalam menegosiasikan sumber daya.

Mari kita lihat bagaimana YARN bekerja. Seperti terlihat sebelumnya, proses dimulai oleh klien dengan berinteraksi dengan NameNode untuk mengetahui DataNodes mana yang tersedia untuk pemrosesan data. Setelah langkah itu selesai, klien berinteraksi dengan Resource Manager untuk melacak sumber daya yang tersedia yang dimiliki Node Manager. Node Manager juga mengirim detak jantung ke Resource Manager.



Ketika klien menghubungi Resource Manager untuk pemrosesan data, Resource Manager ini akan meminta sumber daya yang diperlukan dari beberapa Node Manager. Di sini, Container adalah kumpulan sumber daya fisik seperti CPU dan RAM. Bergantung pada ketersediaan container ini, Node Manager memberikan respon kepada Resource Manager. Setelah ketersediaan dikonfirmasi, Resource Manager akan mulai menjalan master aplikasi (App. Master).

App. Master adalah suatu jenis kode yang mengeksekusi aplikasi. Ini berjalan di salah satu container dan menggunakan container lainnya untuk menjalankan tugas. Jika App. Master membutuhkan sumber daya tambahan, maka ia tidak dapat menghubungi Node Manager secara langsung tetapu harus tetap menghubungi Resource Manager.

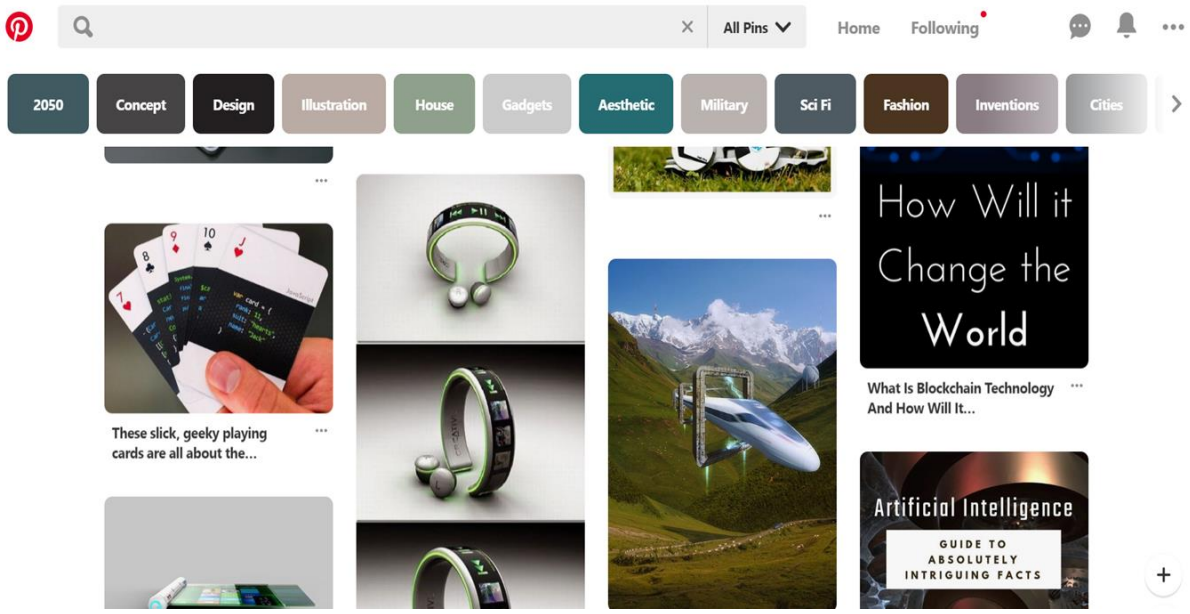



Berikut ini adalah beberapa fitur YARN:

1. YARN bertanggung jawab untuk memproses permintaan pekerjaan dan mengalokasikan sumber daya
2. Versi MapReduce yang berbeda dapat berjalan di YARN sehingga upgrade MapReduce dapat dikelola
3. Bergantung pada kebutuhan, kita dapat menambahkan node.

# Kasus Penggunaan Hadoop: Pinterest

Banyak perusahaan memanfaatkan Hadoop untuk mengelola kumpulan data besar mereka. Mari kita tinjau bagaimana situs web berbagi gambar yang populer, Pinterest, menggunakan Hadoop.




Pinterest adalah platform media sosial yang memungkinkan Kita untuk "menyematkan" (*pin*) informasi menarik yang ditemukan online di situs tersebut. Layanan ini memiliki lebih dari 250 juta pengguna dan hampir 30 miliar pin. Platform menghasilkan sejumlah besar data seperti detail login, perilaku pengguna, pin yang paling banyak dilihat, dll.

Di masa lalu, Pinterest memiliki masalah serius dalam mengelola semua data ini. Perusahaan juga mengalami kesulitan yang signifikan dalam menganalisis data mana yang perlu ditampilkan di mesin penemuan pribadi milik pengguna. Mereka menemukan solusi, yaitu Hadoop. Analisis data yang berkelanjutan memungkinkan Pinterest untuk menyediakan berbagai fitur kepada penggunanya, seperti pin terkait, pencarian terpandu, dan sebagainya.

# Demo Hadoop

Melalui demo individual, kita akan melihat bagaimana HDFS, MapReduce dan YARN dapat digunakan.

## 1. Demo HDFS

Dalam demo ini, kita akan melihat perintah-perintah yang akan membantu kita menuliskan data ke suatu *cluster* dua-node yang memiliki dua DataNodes, dua NodeManager, dan satu mesin Master. Ada tiga cara untuk menulis data:

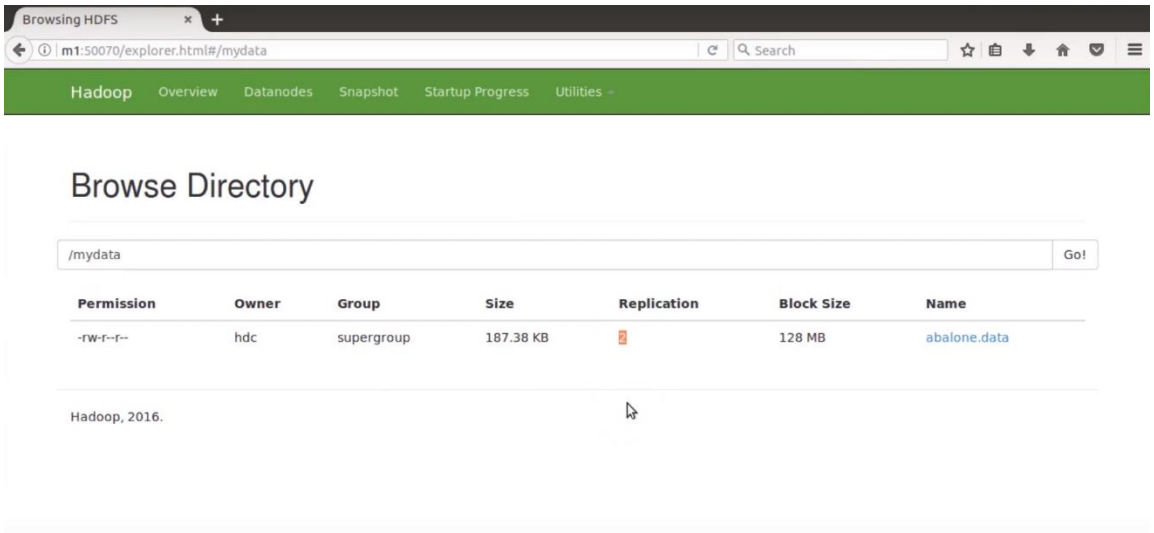
1. Melalui perintah langsung (*command line*)
2. Dengan menuliskan kode program
3. Menggunakan suatu *Graphical User interface* (GUI)

Sebelum dimulai, kita harus mengunduh beberapa dataset sampel yang akan digunakan untuk menulis data ke dalam HDFS. Sekarang, kita akan melihat cara menjalankan beberapa perintah pada HDFS. Ini adalah perintah yang dapat dimulai:

- `hdfs dfs -mkdir /mydata // To create a directory on HDFS`
- `ls // This lists down the files`

- `hdfs dfs -copyFromLocal aba* /mydata // Copies file from local file system to HDFS`
- `hdfs dfs -ls /mydata // Lists the directory`

Setelah perintah ini, menggunakan antarmuka web, kita dapat memeriksa apakah file telah direplikasi. Jika direplikasi, layar akan terlihat seperti gambar berikut:




Sekarang, mari kita berikan perintah tambahan:

```
cp hadoop-hdc-datanode-m1.log cp hadoop-hdc-datanode-m2.log
cp hadoop-hdc-datanode-m1.log cp hadoop-hdc-datanode-m3.log
cp hadoop-hdc-datanode-m1.log cp hadoop-hdc-datanode-m3.log

// Above commands creates multiple files

hdfs dfs -mkdir /mydata2 // Creates a new directory on HDFS
hdfs dfs -put hadoop-hdc-datanode-m* /mydata2 // Copies multiple files
hdfs dfs -setrep -R -w 2 /mydata2 // Sets replication factor to 2
hdfs dfs -rm -R /mydata2 // Removes data from HDFS
```

## 2. Demo MapReduce

Dalam demo MapReduce ini, kita akan melihat cara mendapatkan jumlah total URL yang paling sering dikunjungi.

Pertama, kita harus menggunakan file sampel yang memiliki daftar URL yang kemudian kita hitung kemunculan totalnya. Sampel dataset dapat berasal dari log proxy server atau web server. Untuk mengimplementasikan program ini menggunakan pendekatan MapReduce, silakan ikuti langkah-langkah ini:

1. Gunakan program Mapper di bawah ini yang ditulis untuk melakukan tugas map:

URLCountM.java

```java
package org.example.HCodes;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

public class URLCountM extends Mapper<Text, Text,Text, IntWritable> {
    private static final Logger LOG = Logger.getLogger(URLCountM.class.getName());

    public final IntWritable iw = new IntWritable();

        @Override
    public void map(Text key, Text value, Context context){

        try{
                    LOG.log(Level.INFO, "MAPPER_KEY: ".concat(key.toString()).concat(" MAPPER_VALUE: ".concat(
        context.write(key, new IntWritable(Integer.valueOf(value.toString())));
        }
        catch(NumberFormatException | IOException | InterruptedException e){
            LOG.log(Level.SEVERE, "ERROR: ".concat(e.toString()));
        }
    }
}
```

2. Gunakan program Reducer di bawah ini untuk melakukan agregasi:

URLCountR.java

```java
package org.example.HCodes;

import java.io.IOException;

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

    private static final Logger LOG = Logger.getLogger(URLCountR.class.getName());

    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context
    ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        LOG.log(Level.INFO, "REDUCER_VALUE: ".concat(result.toString()));
        context.write(key, result);
    }
}
```

3. Gunakan program driver di bawah ini untuk memahami kelas mapper, kelas reducer, format kunci output, dan format nilai

URLCount.java

```java
package org.example.HCodes;

import org.apache.hadoop.conf.Configuration;

public class URLCount extends Configured implements Tool {

    public static void main(String[] args) throws Exception {

        int res = ToolRunner.run(new Configuration(), new URLCount(), args);
        System.exit(res);

    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", " ");
        Job job = Job.getInstance(conf, "URLCount");
        job.setJarByClass(getClass());
        job.setInputFormatClass(KeyValueTextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setMapperClass(URLCountM.class);
        job.setReducerClass(URLCountR.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
```

URLCount.java

```java
    public static void main(String[] args) throws Exception {

        int res = ToolRunner.run(new Configuration(), new URLCount(), args);
        System.exit(res);

    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", " ");
        Job job = Job.getInstance(conf, "URLCount");
        job.setJarByClass(getClass());
        job.setInputFormatClass(KeyValueTextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setMapperClass(URLCountM.class);
        job.setReducerClass(URLCountR.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return (job.waitForCompletion(true) == true ? 0 : -1);
    }

}
```

4. Setelah menulis kode, kita dapat mengekspor ini ke bentuk file jar. Selain file jar, perlu ada file dalam HDFS untuk melaksanakan MapReduce. Untuk itu, kita harus masuk (*log in*) ke *cluster* tersebut terlebih dahulu.

5. Masukkan file sampel (file yang memiliki URL) ke HDFS menggunakan perintah **`-put`**.

6. Menggunakan perintah di bawah ini, kita akan menjalankan program MapReduce dan mendapatkan jumlah kumulatif berapa kali suatu URL dikunjungi.

```
hadoop jar URLCount.jar org.example.HCodes.URLCount /user(//mention directory where the input is present) /user(//mention directory where the output should be seen - destination path)
```

7. Setelah menjalankan kode di atas, kita akan melihat bahwa pekerjaan MapReduce dikirimkan ke cluster YARN. Setiap kali program MapReduce berjalan, kita akan memiliki satu atau lebih file bagian yang dibuat sebagai output. Dalam hal ini, Kita memiliki satu tugas map dan satu tugas reduce, dan karenanya jumlah file bagian juga akan menjadi satu. Kode berikut digunakan untuk menampilkan jumlah file bagian dan hasil akhir:

```
hdfs dfs -ls /user(give your destination path) //Displays the number of part files

hdfs dfs -cat /user(give your destination path)/part-r-00000(mention the part details) //Displays the final output
```

### 3. Demo YARN

Di bawah ini adalah beberapa perintah YARN yang paling sering digunakan:

```
yarn version //Displays the Hadoop and vendor-specific distribution version
yarn application -list //Lists all the applications running
yarn application -list -appSTATES -FINISHED //Lists the services that are finished running
yarn application -status give application ID //Prints the status of the applications
yarn application -kill give application ID //Kills a running application
yarn node -list //Gives the list of node managers
yarn rmadmin -getGroups hdfs //Gives the group HDFS belongs to
```

## Kesimpulan

Kami harap tutorial ini membantu Anda memahami Apache Hadoop. Kita telah belajar tentang big data, Hadoop dan keterkaitan keduanya. Kita telah melihat apa dan bagaimana kerja dari HDFS, MapReduce, dan YARN. Terakhir, kita belajar bagaimana komponen-komponen Hadoop ini bekerja melalui berbagai demo. Jika Anda ingin mempelajari lebih lanjut tentang big data dan Hadoop, bacalah beberapa buku tentang Big data dan Hadoop, ikut pelatihan khusus tentang ini yang sering ada di Universitas Trunojoyo Madura. Jangan lupa, praktek mandiri juga penting sekali!