Module 2

Hadoop ecosystem

- **Sqoop:** It is used to import and export data to and from between HDFS and RDBMS.
- **Pig:** It is a procedural language platform used to develop a script for MapReduce operations.
- **Hbase**: HBase is a distributed column-oriented database built on top of the Hadoop file system.
- **Hive:** It is a platform used to develop SQL type scripts to do MapReduce operations.
- **Flume**: Used to handle streaming data on the top of Hadoop.
- **Oozie:** Apache Oozie is a workflow scheduler for Hadoop.

Introduction to Pig

- Pig raises the level of abstraction for processing large amount of datasets. It is a fundamental platform for analyzing large amount of data sets which consists of a high level language for expressing data analysis programs. It is an open source platform developed by yahoo.

Advantages of Pig

- Reusing the code
- Faster development
- Less number of lines of code
- Schema and type checking etc

What makes Pig Hadoop popular?

- Easy to learn read and write and implement if you know SQL.
- It implements a new approach of multi query.
- Provides a large number of nested data types such as Maps, Tuples and Bags which are not easily available in MapReduce along with some other data operations like Filters, Ordering and Joins.
- It consist of different user groups for instance up to 90% of Yahoo's MapReduce is done by Pig and up to 80% of Twitter's MapReduce is also done by Pig and various other companies like Sales force, LinkedIn and Nokia etc are majoritively using the Pig.

Pig Latin comes with the following features:

- Simple programming: it is easy to code, execute and manage the program.
- Better optimization: system can automatically optimize the execution as per the requirement raised.
- Extensive nature: Used to achieve highly specific processing tasks.

Pig can be used for following purposes:

- ETL data pipeline
- Research on raw data
- Iterative processing.

Data Types

The scalar data types in pig are in the form of int, float, double, long, chararray, and byte array. The complex data types in Pig are namely the map, tuple, and bag.

- Map: The data element consisting the data type chararray where element has pig data type include complex data type
- Example- [city'#'bang','pin'#560001]
- In this city and pin are data element mapping the values here.
- Tuple: Collection of data types and it has defined fixed length. It consists of multiple fields and those are ordered in sequence.
- Bag: It is a huge collection of tuples ,unordered sequence , tuples arranged in the bag are separated by comma.
- Example: {('Bangalore', 560001),('Mysore',570001),('Mumbai',400001)

Running Pig Programs

- There are namely 3 ways of executing Pig programs which works on both local and MapReduce mode:
- **Script**
- Pig can run a script file that contains Pig commands. For example, pig script.pig runs the commands in the local file script.pig. Alternatively, for very short scripts, you can use the -e option to run a script specified as a string on the command line.
- **Grunt**
- Grunt is an interactive shell programming for running Pig commands. Grunt is started when no file is specified for Pig to run, and the -e option apparently not used. It is also possible to run Pig scripts from within Grunt using run and exec.
- **Embedded**
- You can execute all the Pig programs from Java and can use JDBC to run SQL programs from Java.

Installation

- Download
- Extract
- Set Path

Run Example -> grunt

- pig -x local
- pig -x mapreduce
- A = load 'passwd' using PigStorage(':');
- B = foreach A generate $0 as id;
- dump B;

Run Example ->Script

- A = load 'passwd' using PigStorage(':');
- B = foreach A generate $0 as id;
- dump B;
- Svae in filename.pig

- Run using command
- pig -x local id.pig

Run Example ->Script in hadoop

- A = load 'passwd' using PigStorage(':');
- B = foreach A generate $0 as id;
- dump B;
- Save in filename.pig

- Start Hadoop
- Store input file in HDFS
- Hdfs dfs -mkdir /passwdDIR
- hdfs dfs -put passwd passwdDIR

- Run using command
- pig -x mapreduce id.pig

Apache Sqoop

- Sqoop is a tool designed to transfer data between Hadoop and relational databases. you can use Sqoop to import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDFS), transform the data in Hadoop, and then export the data back into an RDBMS.

Sqoop – "SQL to Hadoop and Hadoop to SQL"

Apache Sqoop

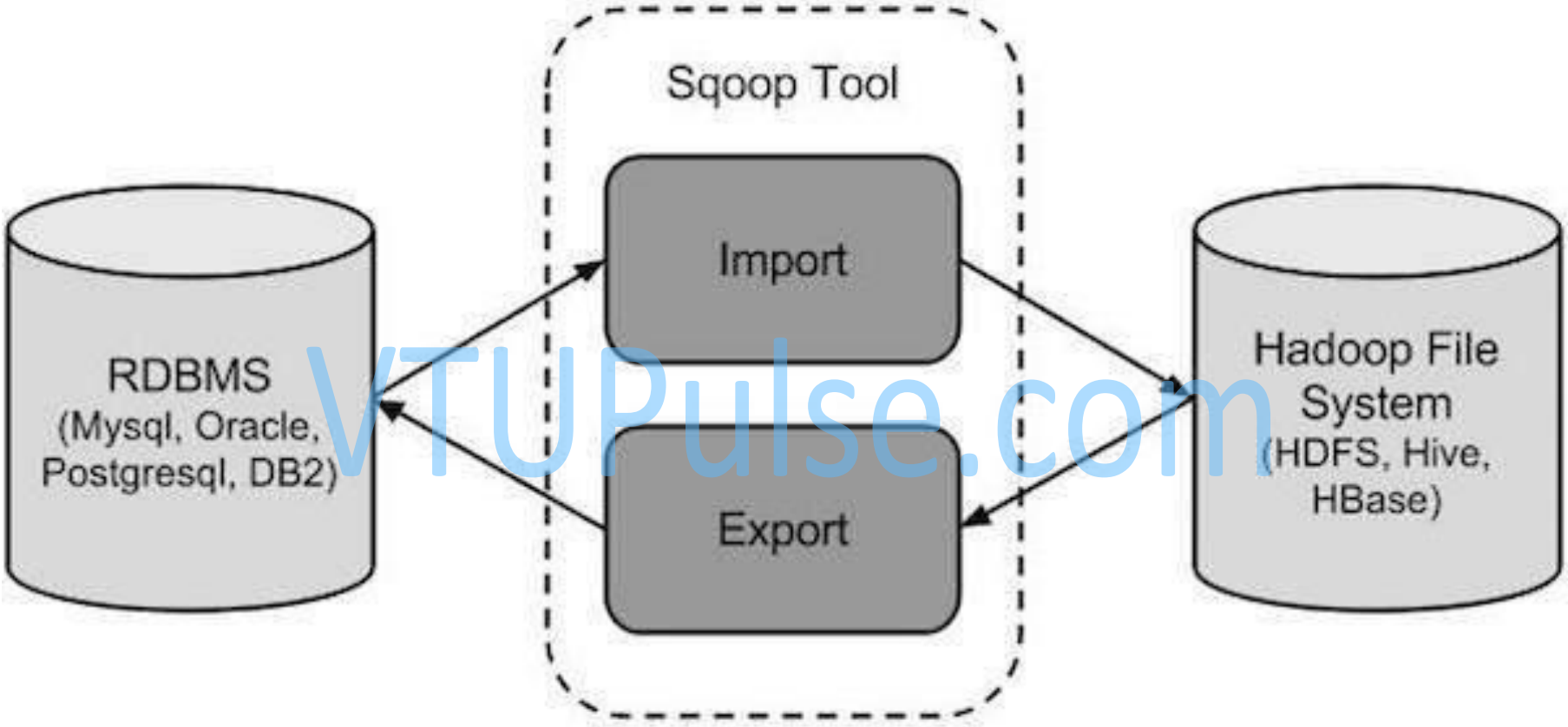
- The traditional application management system, that is, the interaction of applications with relational database using RDBMS, is one of the sources that generate Big Data. Such Big Data, generated by RDBMS, is stored in Relational **Database Servers** in the relational database structure.

- When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. Here, Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop's HDFS

Apache Sqoop

- Sqoop can be used with any Java Database Connectivity ODBC)—compliant database and has been tested on Microsoft SQL Server, PostgresSQL, MySQL, and Oracle. In version I of Sqoop, data were accessed using connectors written for specific databases. Version 2 (in beta) does not support connectors or version 1 data transfer from a RDBMS directly to Hive or HBase, or data transfer from Hive or HBase to your RDBMS.

How Sqoop Works?

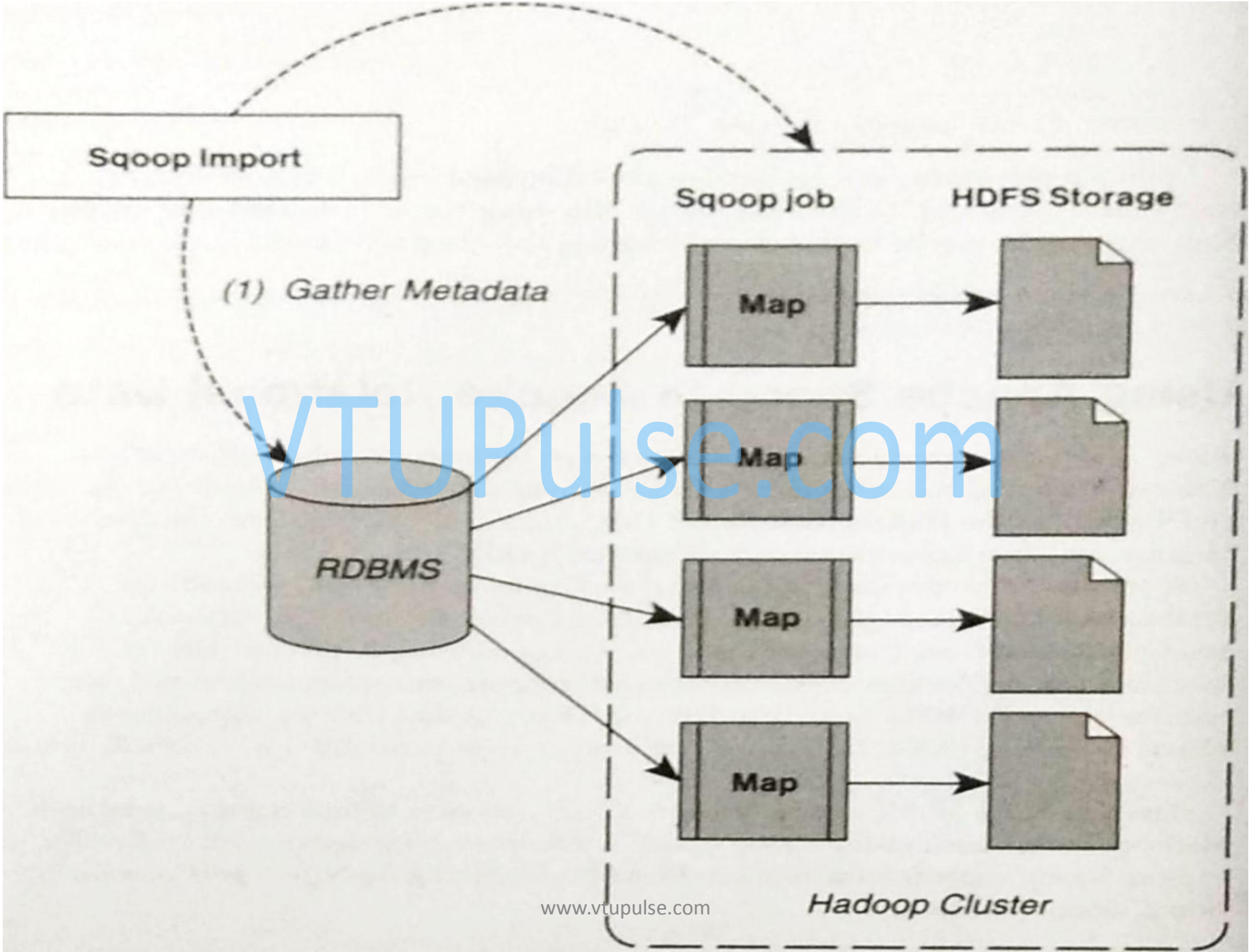


How Sqoop Works?

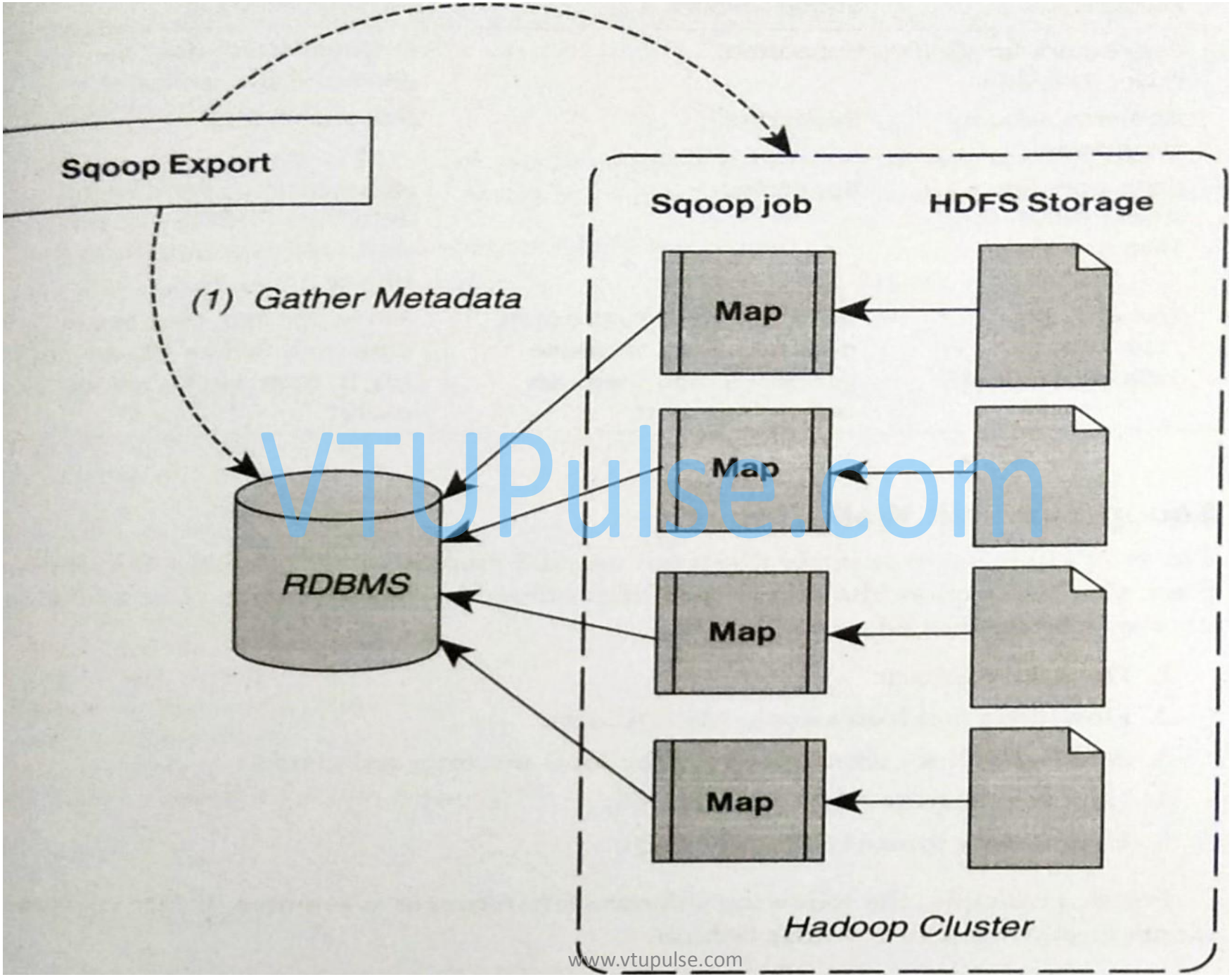
- Sqoop Import
  - The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.
- Sqoop Export
  - The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

Apache Sqoop Import and Export Methods

- Figure describes the Sqoop data import (to HDFS) process. The data import is done in two steps.
- In the first step, shown in the figure, Sqoop examines the database to gather the necessary metadata for the data to be imported.
- The second step is a map-only (no reduce step) Hadoop job that Sqoop submits to the cluster. This job does the actual data transfer using the metadata captured in the previous step. Note that each node doing the import must have access to the database.
- The imported data are saved in an HDFS directory. Sqoop will use the database name for the directory, or the user can specify any alternative directory where the files should be populated. By default, these files contain comma-delimited fields, with new lines searating different records.
- You can easily override the format in which data are copied over by explicitly specifying the field separator and record terminator characters. Once placed in HDFS, the data are ready for processing.



- Data export from the cluster works in a similar fashion. The export is done in two steps, as shown In Figure.
- As in the import process, the first step is to examine the database for metadata. The export step again uses a map-only Hadoop job to write the data to the database.
- Sqoop divides the input data set into splits, then uses individual map tasks to push the splits to the database. Again, this process assumes the map tasks have access to the database.



Apache Sqoop Version Changes

- Sqoop Version 1 uses specialized connectors to access external systems. These connectors are often optimized for varjous RDBMSs or for systems that do not support JDBC.
- Connectors are plug-in components based on Sqoop's extension framework and can be added to any existing Sqoop installation. Once a connector is installed, Sqoop can use it to efficiently transfer data between Hadoop and the external store supported by the connector.
- By default, Sqoop version 1 includes connectors for popular databases such as MySQL, PostgreSQL, Oracle, SQL Server, and DB2. It also supports direct transfer to and from the RDBMS to HBase or Hive.
- In contrast, to streamline the Sqoop input methods, Sqoop version 2 no longer supports specialized connectors or direct import into HBase or Hive. All imports and exports are done through the JDBC interface. Table summarizes the

Apache Sqoop Version Changes

Feature	Sqoop Version 1	Sqoop Version 1
Connectors for ail major ROSMSs	Supported.	Not supported. Use the generic JDBC connector.
Kerberos security integration	Supported.	Not supported.
Data transfer from RDBMS to Hive or HBase	Supported.	Not supported. First import data from RDBMS into HDFS, then load data into Hive or HBase manually.
Data transfer from Hive or HBase to RDBMS	Not supported. First export data from Hive or Hbase into HDFS, and then use Sqoop for export.	Not supported. First export data from Hive or HBase into HDFS, then use Sqoop for export

Sqoop Example Walk-Through

- The following simple example illustrates use of Sqoop. It can be used as a foundation from which to explore the other capabilities offered by Apache Sqoop. The following steps Will be performed:
  1. Download Sqoop.
  2. Download and load sample MySQL data.
  3. Add Sqoop user permissions for the local machine and cluster.
  4. Import data from MySQL to HDFS.
  5. Export data from HDFS to MySQL.

Step 1: Download Sqoop and Load Sample MySQL Database

- If you have not done so already, make sure Sqoop is installed on your cluster. Sqoop is needed on only a single node in your cluster. This Sqoop node will then serve as an entry point for all connecting Sqoop clients. Because the Sqoop node is a Hadoop MapReduce client, it requires both a Hadoop installation and access to HDFS. To install Sqoop using the HDP distribution RPM files, simply enter:

  – yum install sqoop sqoop-metastore

What is Flume?

- Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store.
- Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.

HDFS put Command

- The main challenge in handling the log data is in moving these logs produced by multiple servers to the Hadoop environment.
- Hadoop **File System Shell** provides commands to insert data into Hadoop and read from it. You can insert data into Hadoop using the **put** command as shown below.

$ Hadoop fs –put /path of the required file /path in HDFS where to save the file

Problem with put Command

We can use the **put** command of Hadoop to transfer data from these sources to HDFS. But, it suffers from the following drawbacks

- Using **put** command, we can transfer **only one file at a time** while the data generators generate data at a much higher rate. Since the analysis made on older data is less accurate, we need to have a solution to transfer data in real time.
- If we use **put** command, the data is needed to be packaged and should be ready for the upload. Since the webservers generate data continuously, it is a very difficult task.

What we need here is a solutions that can overcome the drawbacks of **put** command and transfer the "streaming data" from data generators to centralized stores (especially HDFS) with less delay.

Available Solutions

- Facebook's Scribe
- Apache Kafka
- Apache Flume

Applications of Flume

- Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue.
- Flume is used to move the log data generated by application servers into HDFS at a higher speed.

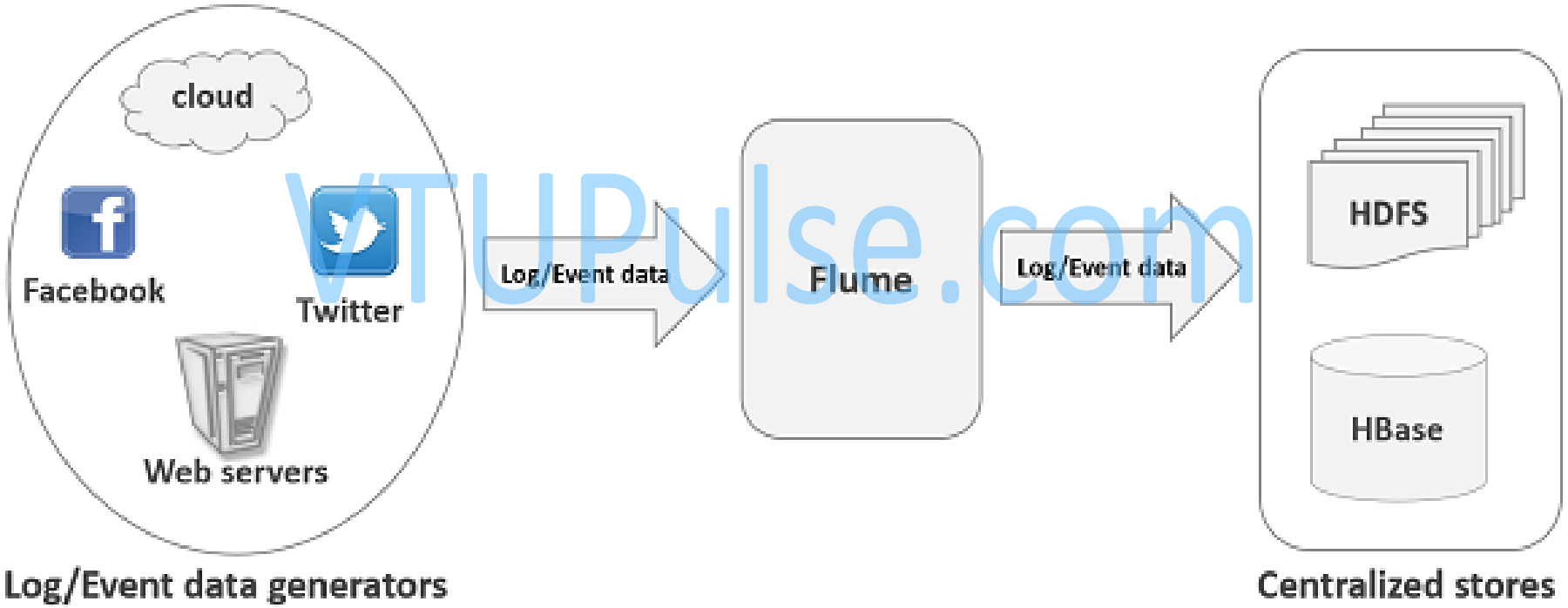
Advantages of Flume

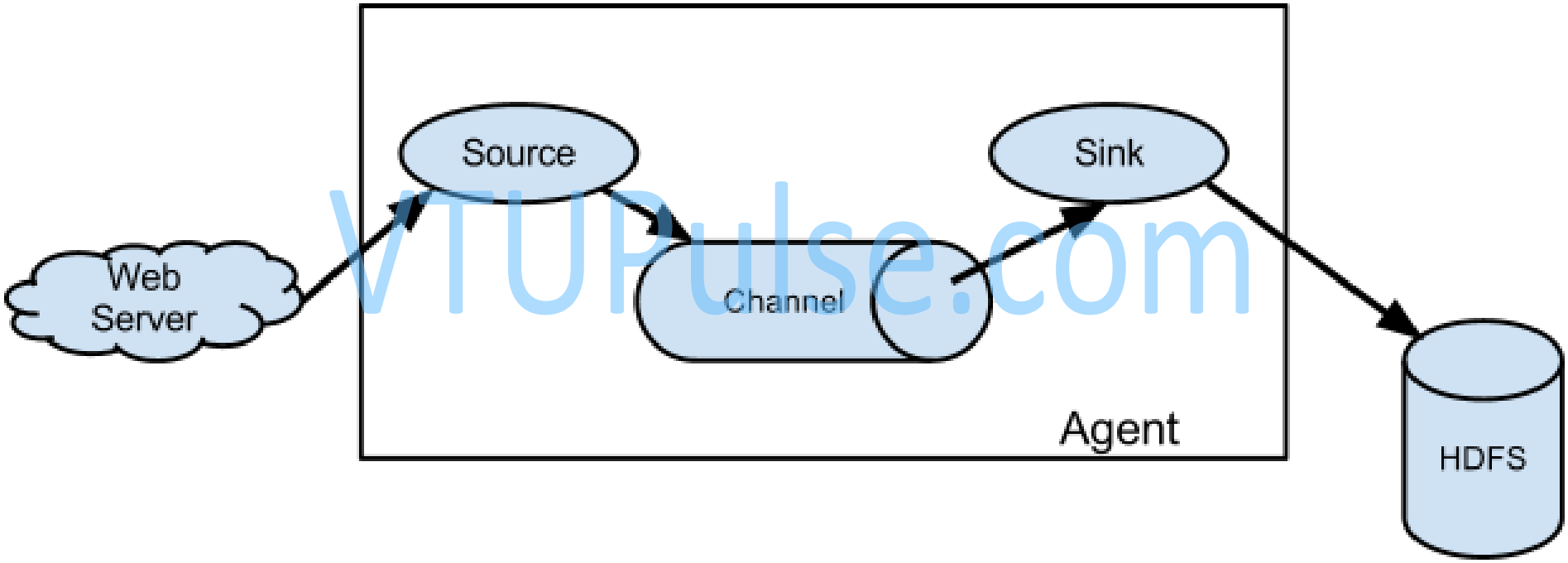
- Using Apache Flume we can store the data in to any of the centralized stores (HBase, HDFS).
- When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
- Flume provides the feature of **contextual routing**.
- The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
- Flume is reliable, fault tolerant, scalable, manageable, and customizable.

Apache Flume - Architecture

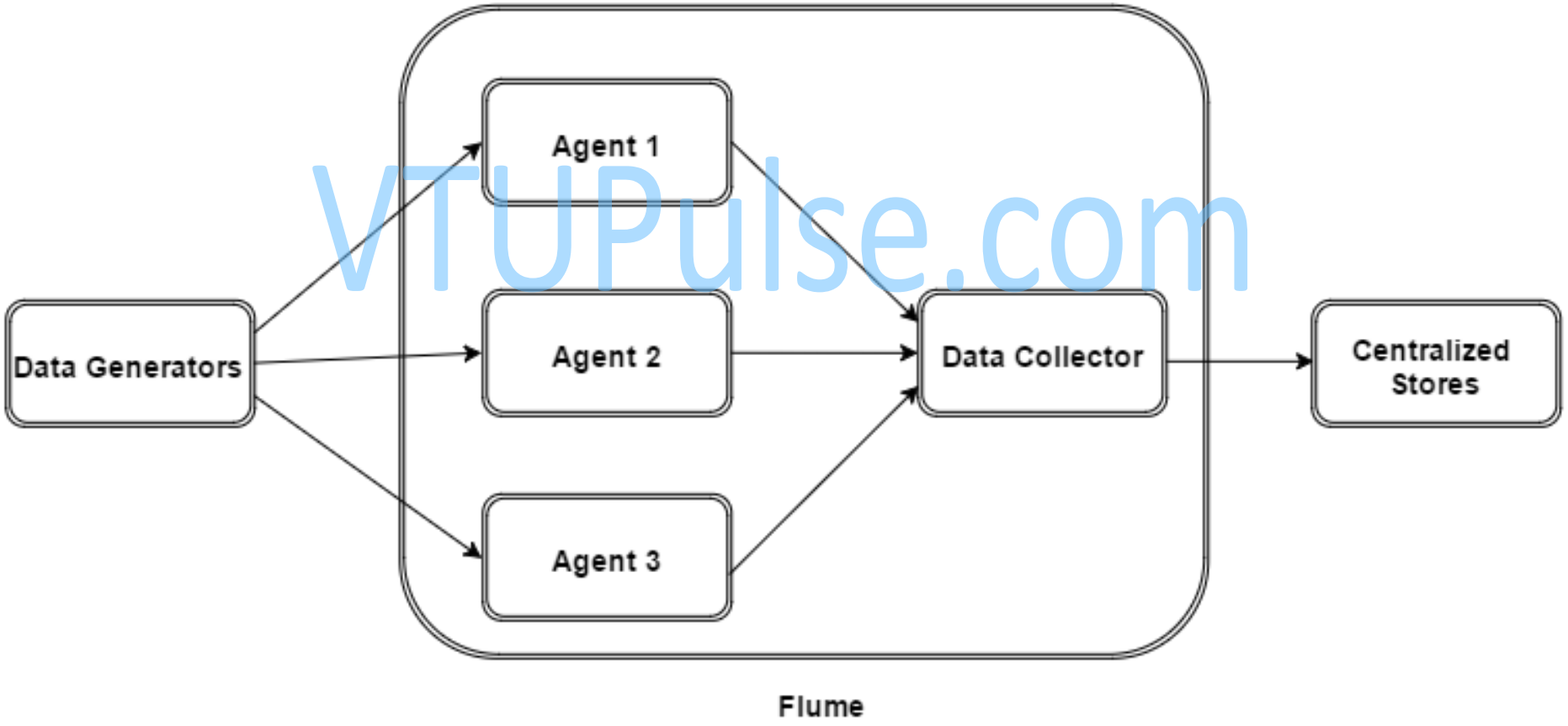
- The following illustration depicts the basic architecture of Flume. As shown in the illustration, **data generators** (such as Facebook, Twitter) generate data which gets collected by individual Flume **agents** running on them. Thereafter, a **data collector** (which is also an agent) collects the data from the agents which is aggregated and pushed into a centralized store such as HDFS or HBase.

What is Flume?





Apache Flume - Architecture



Flume Agent

- An **agent** is an independent daemon process (JVM) in Flume. It receives the data (events) from clients or other agents and forwards it to its next destination (sink or agent). Flume may have more than one agent. Following diagram represents a **Flume Agent**

Source

- A **source** is the component of an Agent which receives data from the data generators and transfers it to one or more channels in the form of Flume events.
- Apache Flume supports several types of sources and each source receives events from a specified data generator.
- **Example** − Facebook, Avro source, Thrift source, twitter 1% source etc.

Channel

- A **channel** is a transient store which receives the events from the source and buffers them till they are consumed by sinks. It acts as a bridge between the sources and the sinks.
- These channels are fully transactional and they can work with any number of sources and sinks.
- **Example** − JDBC channel, File system channel, Memory channel, etc.

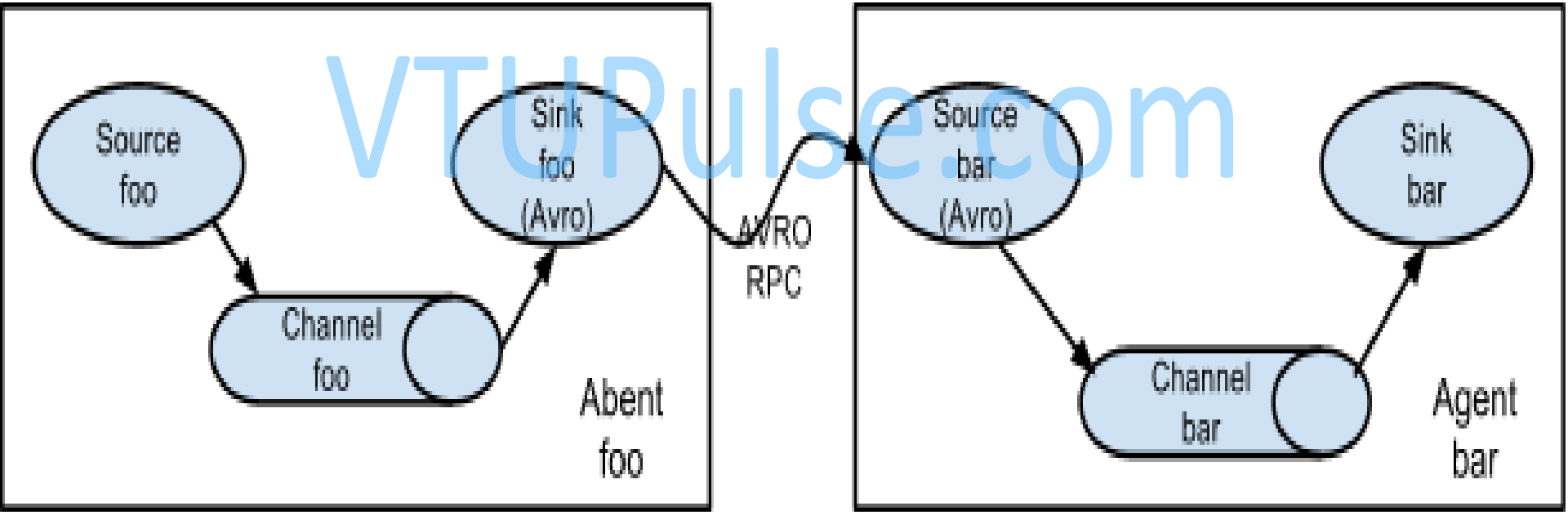
Sink

- A **sink** stores the data into centralized stores like HBase and HDFS. It consumes the data (events) from the channels and delivers it to the destination. The destination of the sink might be another agent or the central stores.
- **Example** − HDFS sink

Setting multi-agent flow

- In order to flow the data across multiple agents or hops, the sink of the previous agent and source of the current hop need to be avro type with the sink pointing to the hostname (or IP address) and port of the source.
- Within Flume, there can be multiple agents and before reaching the final destination, an event may travel through more than one agent. This is known as **multi-hop flow**.

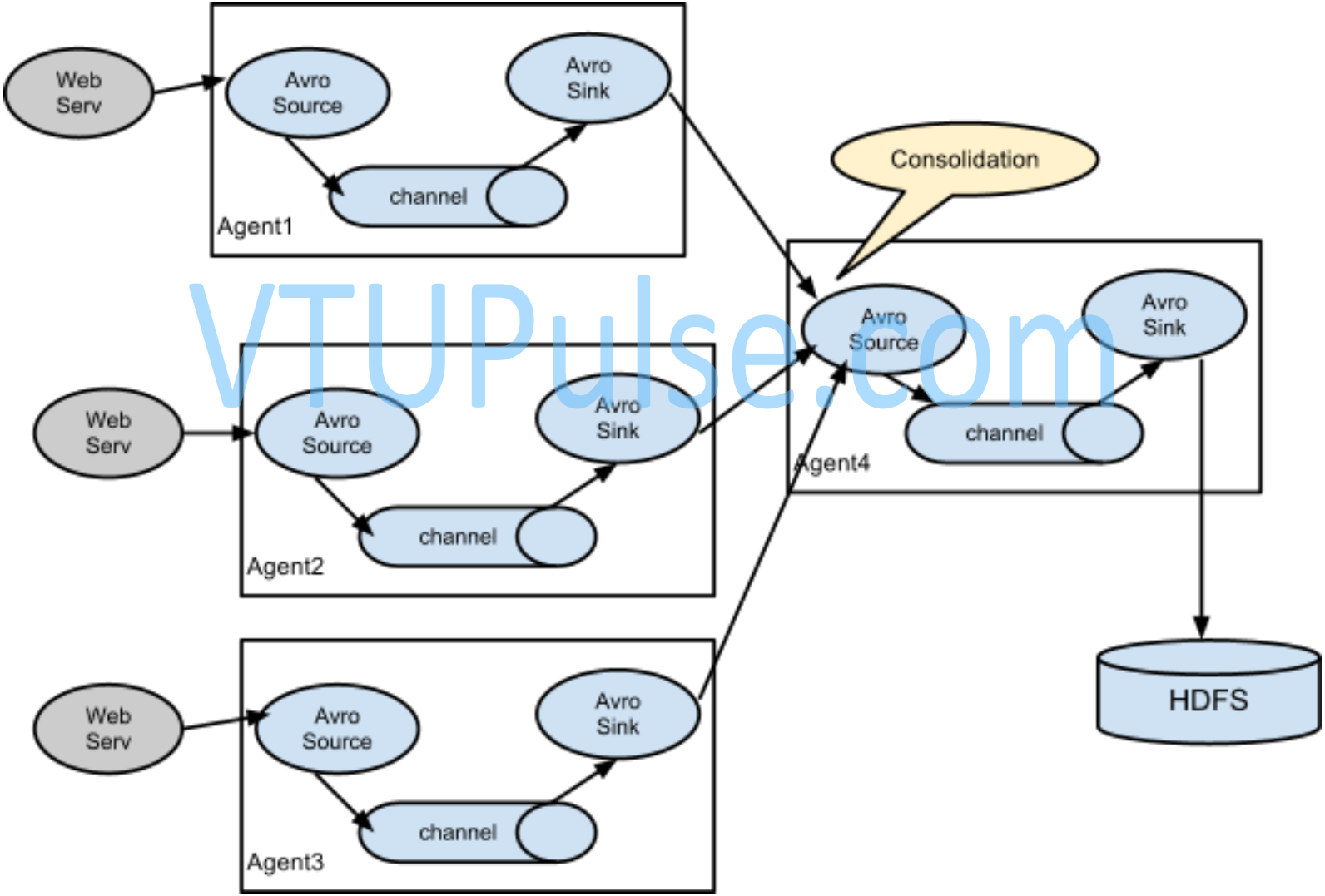
Setting multi-agent flow



Consolidation

- A very common scenario in log collection is a large number of log producing clients sending data to a few consumer agents that are attached to the storage subsystem. For example, logs collected from hundreds of web servers sent to a dozen of agents that write to HDFS cluster.
- This can be achieved in Flume by configuring a number of first tier agents with an avro sink, all pointing to an avro source of single agent (Again you could use the thrift sources/sinks/clients in such a scenario). This source on the second tier agent consolidates the received events into a single channel which is consumed by a sink to its final destination.

Consolidation



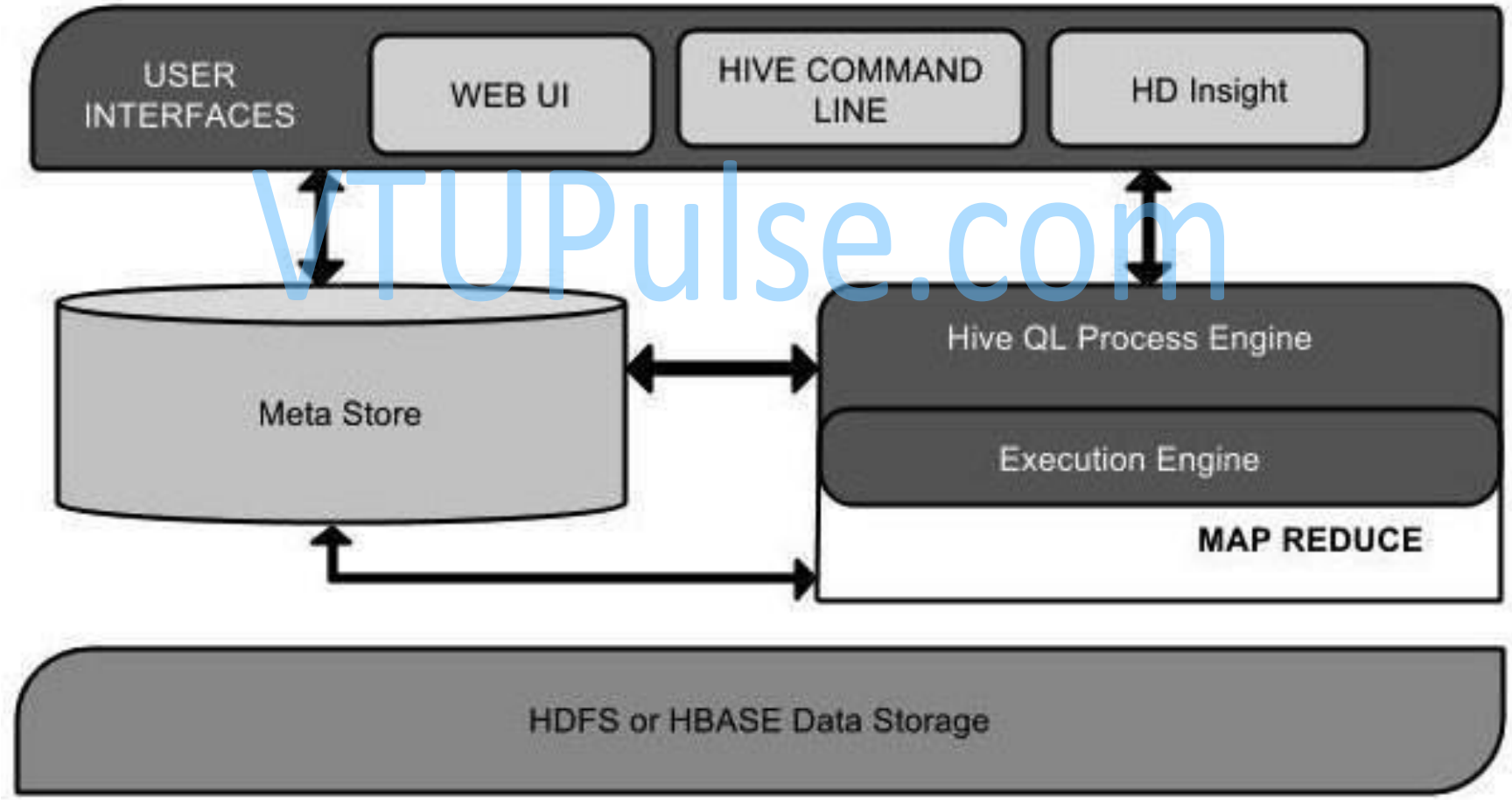
Apache Hive

- Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
- Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

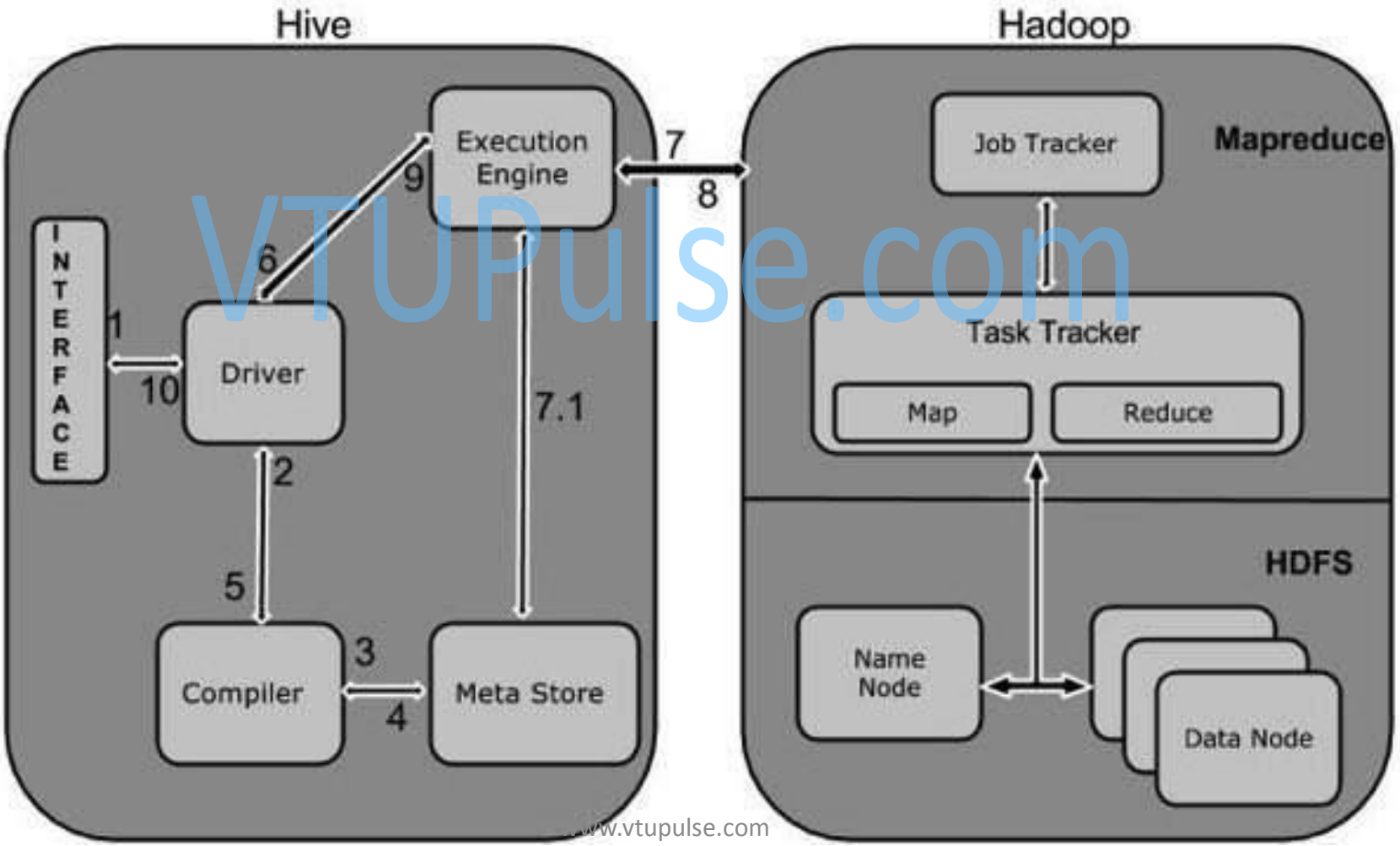
Architecture of Hive



Architecture of Hive

User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive



Working of Hive

Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.

Working of Hive

7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

Hive - Data Types

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Hive - Data Types - Column Types

Integral Types

- Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

String Types

- String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

Timestamp

- “YYYY-MM-DD HH:MM:SS.fffffffff”
- Date
- year/month/day format

Decimal

Unions

Hive - Data Types - Literals

- The following literals are used in Hive:
- Floating Point Types
- Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.
- Decimal Type
- Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10^{-308} to 10^{308}.

Hive - Data Types - Complex Types

Arrays

- Arrays in Hive are used the same way they are used in Java.

Maps

- Maps in Hive are similar to Java Maps.

Structs

- Structs in Hive is similar to using complex data with comment.

- hive> CREATE DATABASE [IF NOT EXISTS] userdb;
- hive> SHOW DATABASES;
- hive> DROP DATABASE IF EXISTS userdb;
- hive> DROP SCHEMA userdb;
- hive> CREATE TABLE IF NOT EXISTS employee (eid int, name String, salary String, destination String) COMMENT 'Employee details' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

  LINES TERMINATED BY '\n'

  STORED AS TEXTFILE;
- hive> ALTER TABLE employee RENAME TO emp;
- hive> DROP TABLE IF EXISTS employee;

Apache Oozie

- Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs the workflow of dependent jobs. Here, users are permitted to create **Directed Acyclic Graphs** of workflows, which can be run in parallel and sequentially in Hadoop.

Apache Oozie

It consists of Three parts:

- Workflow engine: Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.
- **Coordinator engine**: It runs workflow jobs based on predefined schedules and availability of data.
- Bundle: Higher level abstraction that will batch a set of coordinator jobs

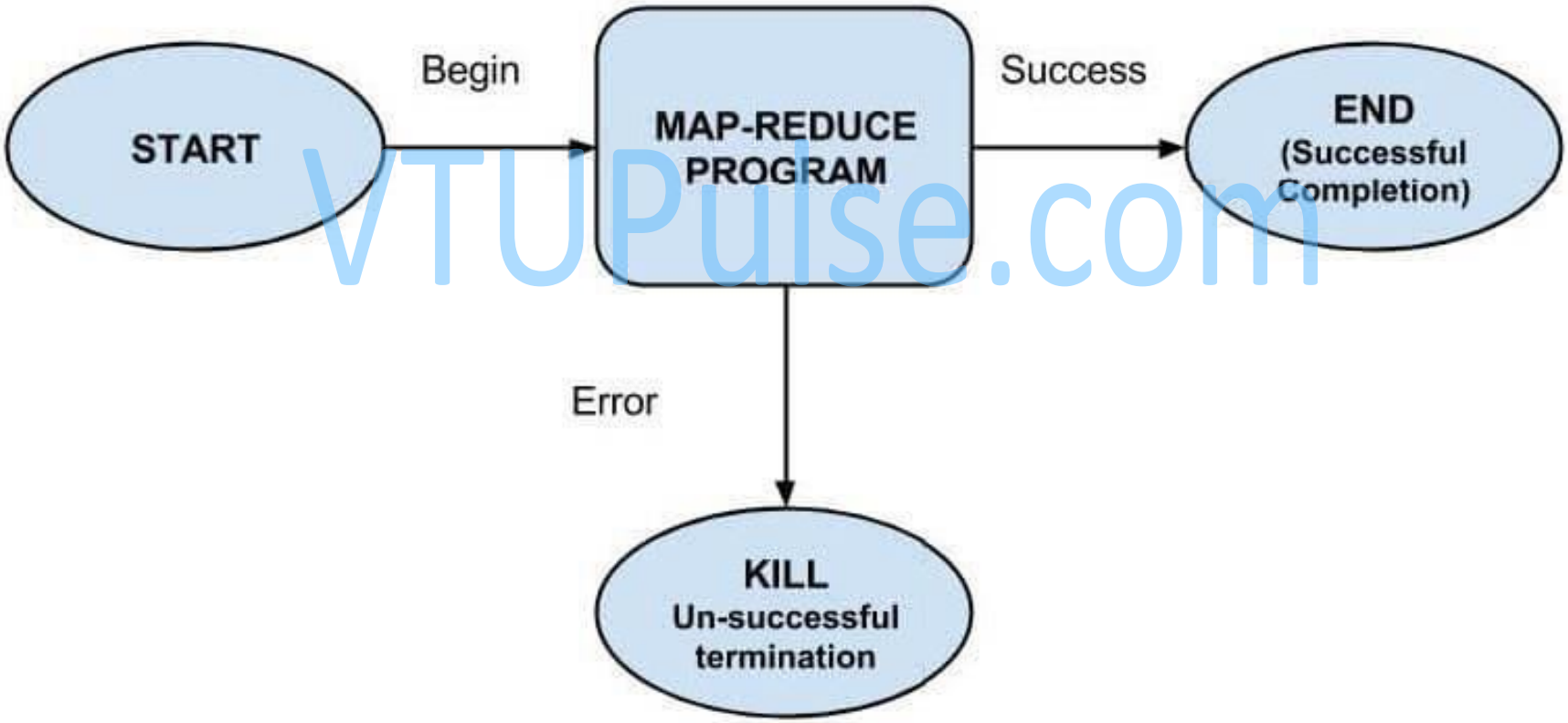
Apache Oozie

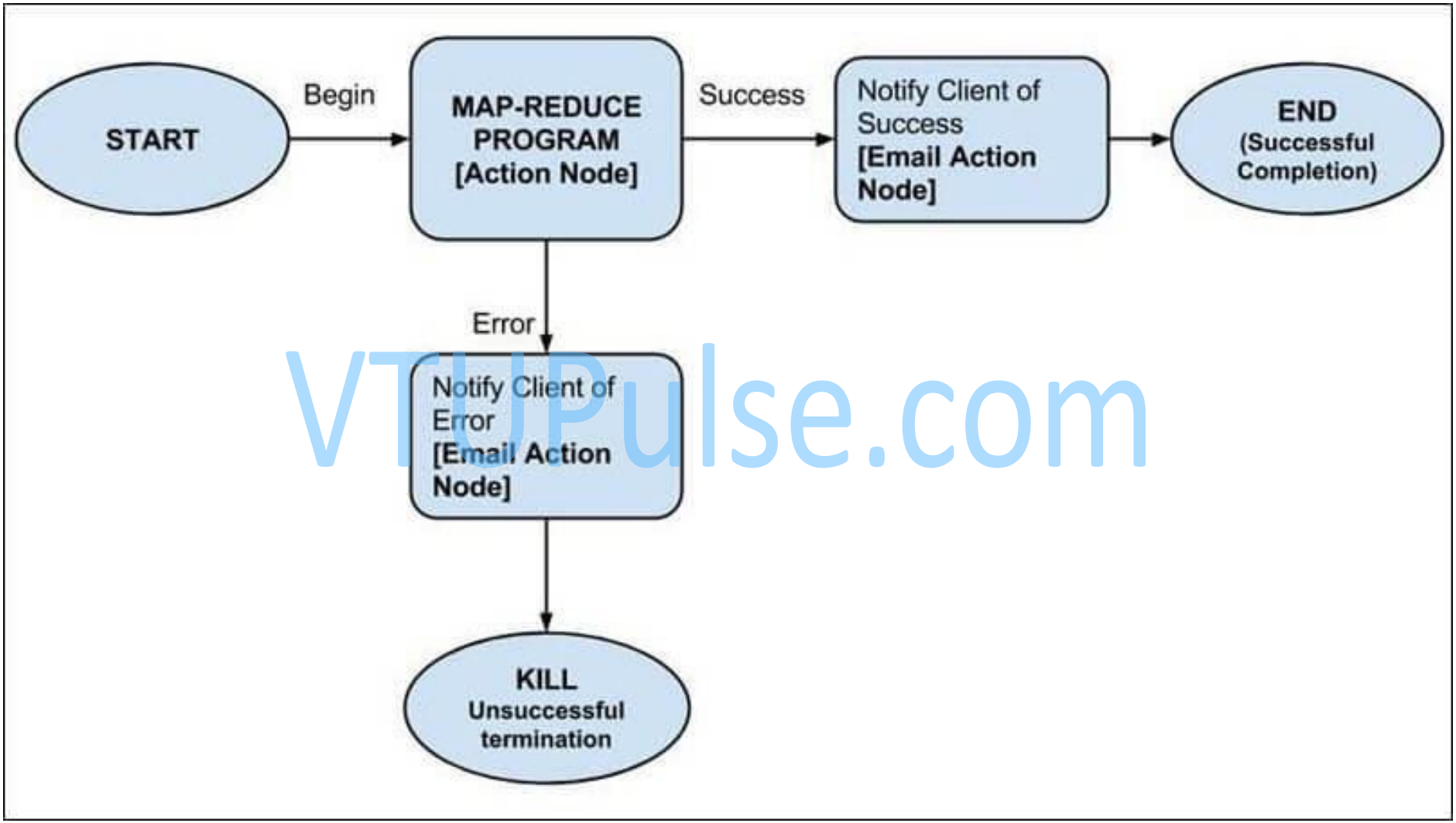
- Oozie is **scalable** and can manage the timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster.
- Oozie is very much **flexible**, as well. One can easily start, stop, suspend and rerun jobs. Oozie makes it very easy to rerun failed workflows. One can easily understand how difficult it can be to catch up missed or failed jobs due to downtime or failure. It is even possible to skip a specific failed node.

How does OOZIE work?

- Oozie runs as a service in the cluster and clients submit workflow definitions for immediate or later processing.
- Oozie workflow consists of **action nodes** and **control-flow nodes**.
- A **control-flow node** controls the workflow execution between actions by allowing constructs like conditional logic wherein different branches may be followed depending on the result of earlier action node.
- **Start Node**, **End Node**, and **Error Node** fall under this category of nodes.
- **Start Node,** designates the start of the workflow job.
- **End Node,** signals end of the job.
- **Error Node** designates the occurrence of an error and corresponding error message to be printed.
- An **action node** represents a workflow task, e.g., moving files into HDFS, running a MapReduce, Pig or Hive jobs, importing data using Sqoop or running a shell script of a program written in Java.

Example Workflow Diagram





HBase

- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data.
- Since 1970, RDBMS is the solution for data storage and maintenance related problems. After the advent of big data, companies realized the benefit of processing big data and started opting for solutions like Hadoop.

Limitations of Hadoop

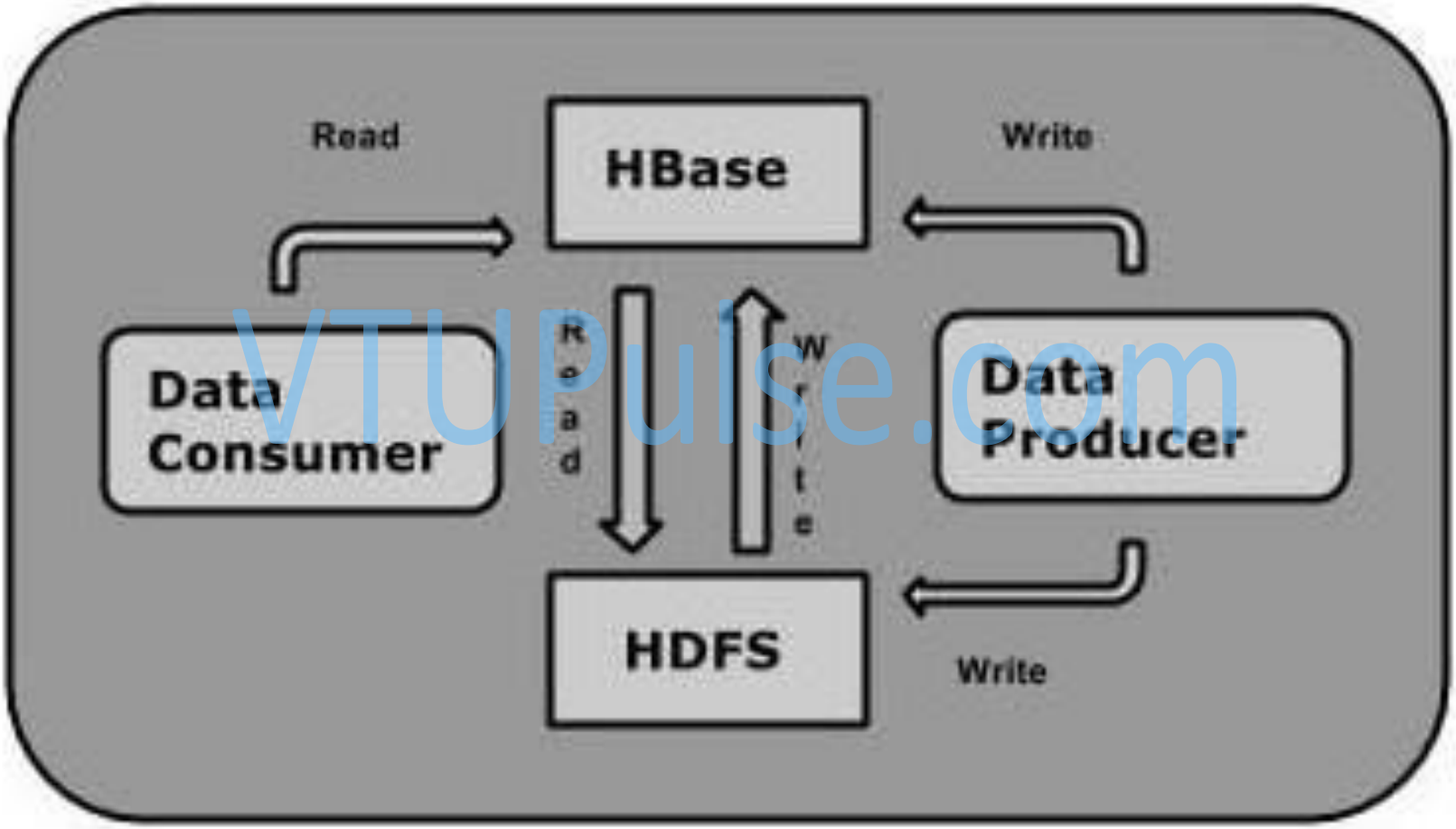
- Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.
- A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

What is HBase?

- HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

What is HBase?



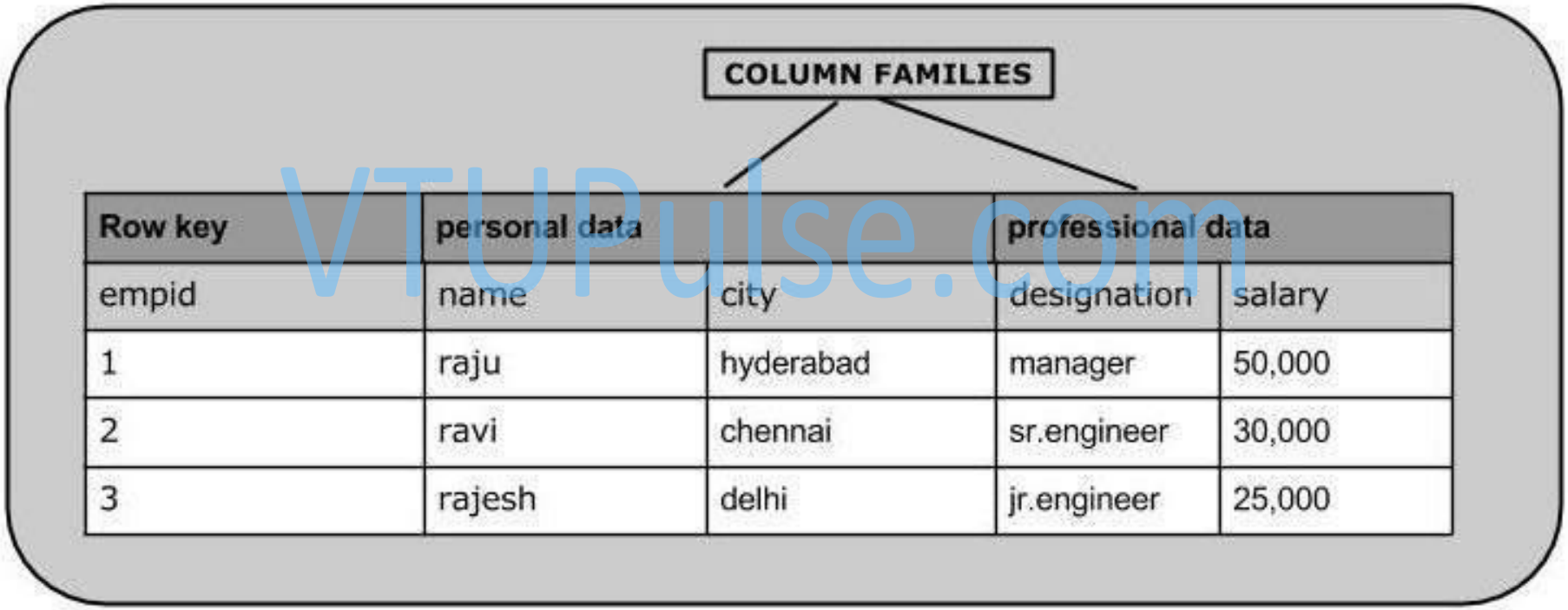
Storage Mechanism in HBase

HBase is a column-oriented database and the tables in it are sorted by row.

In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Storage Mechanism in HBase



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

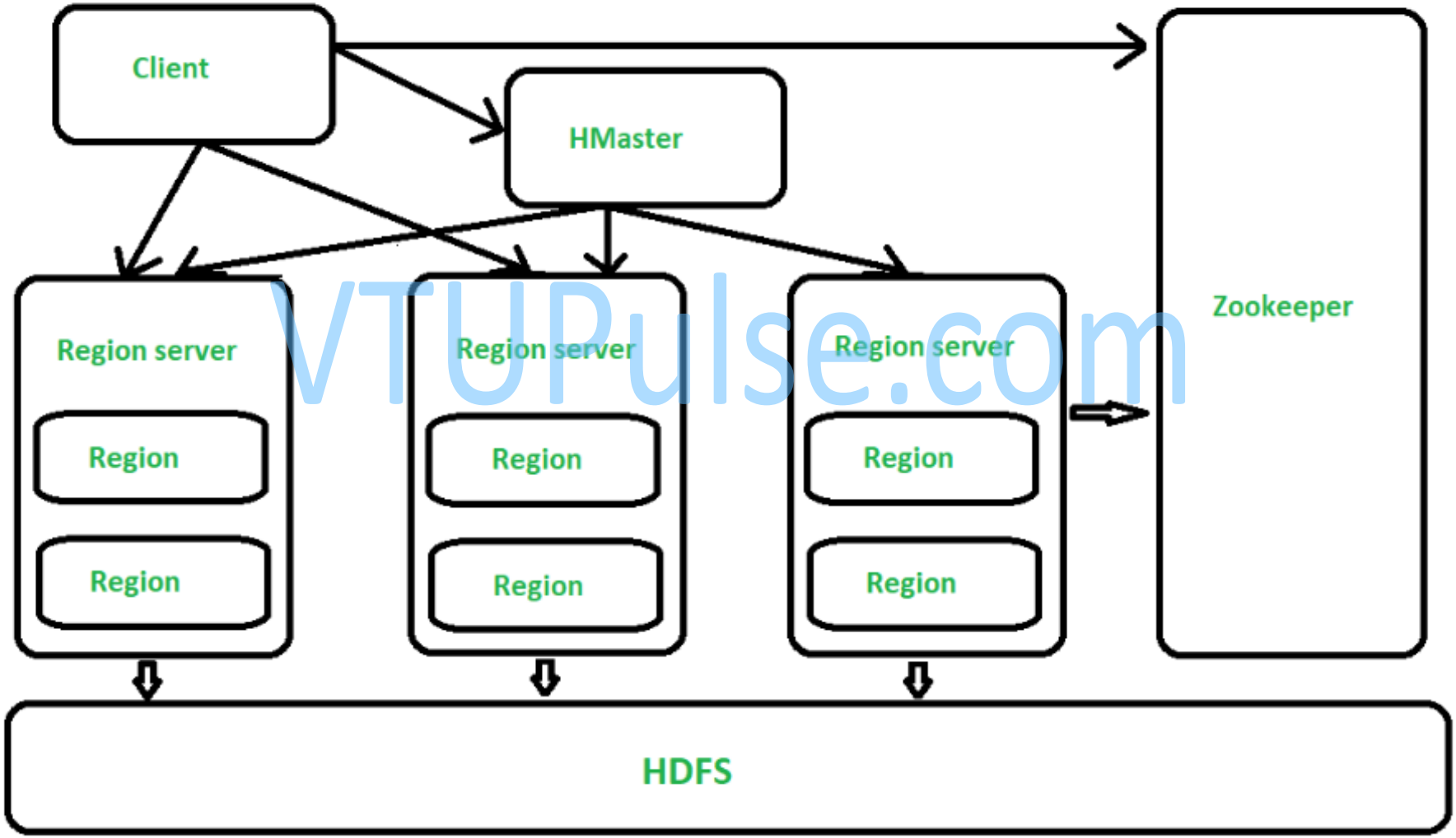
Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.

- It hosts very large tables on top of clusters of commodity hardware.

- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache

Hbase Architecture



Hbase Installation

- Download
- http://archive.apache.org/dist/hbase/0.98.24/
- Extract
- sudo tar -zxvf hbase-0.98.24-hadoop2-bin.tar.gz
- Move
- sudo mv hbase-0.98.24-hadoop2 /usr/local/Hbase
- cd /usr/local/Hbase/

- <property>
- <name>hbase.rootdir</name>
- <value>file:/usr/local/hadoop/HBase/HFiles</value>
- </property>

- //Here you have to set the path where you want HBase to store its built in zookeeper

Hbase Shell

- hbase shell
- Create Database and insert data
- create ‘apple’, ‘price’, ‘volume’
- put 'apple', '17-April-19', 'price:open', '125'
- put 'apple', '17-April-19', 'price:high', '126'
- put 'apple', '17-April-19', 'price:low', '124'
- put 'apple', '17-April-19', 'price:close', '125.5'
- put 'apple', '17-April-19', 'volume', '1000'

Inspect Database

- scan 'apple'
- ROW COLUMN+CELL
- 17-April-19 column=price:close, timestamp=1555508855040, value=122.5
- 17-April-19 column=price:high, timestamp=1555508840180, value=126
- 17-April-19 column=price:low, timestamp=1555508846589, value=124
- 17-April-19 column=price:open, timestamp=1555508823773, value=125
- 17-April-19 column=volume:, timestamp=1555508892705, value=1000

Get a row

- get 'apple', '17-April-19'
- COLUMN CELL
- price:close timestamp=1555508855040, value=122.5
- price:high timestamp=1555508840180, value=126
- price:low timestamp=1555508846589, value=124
- price:open timestamp=1555508823773, value=125
- volume: timestamp=1555508892705, value=1000

Get table Cell

- get 'apple', '17-April-19', {COLUMN => 'price:low'}
- COLUMN CELL
- price:low timestamp=1555508846589, value=124

- get 'apple', '17-April-19', {COLUMN => ['price:low', 'price:close']}
- COLUMN CELL
- price:close timestamp=1555508855040, value=122.5
- price:low timestamp=1555508846589, value=124

Get table Cell

- get 'apple', '17-April-19', { COLUMN => ['volume', 'price:low']}
- COLUMN CELL
- price:low timestamp=1555508846589, value=124
- volume: timestamp=1555508892705, value=1000

Delete Cell, Row and Table

- delete 'apple', '17-April-19', 'price:low'
- deleteall 'apple', '17-April-19'
- disable 'apple'
- drop 'apple'

Scripting

- echo "create 'apple', 'price', 'volume'"
- echo "create 'apple', 'price', 'volume'" | hbase shell
- Create test.sh file with contents
- echo "create 'mango', 'price', 'volume'"
- echo "put 'mango', ‘123’, 'price', '100'"
- Then run the following commands
- sudo sh test.sh | hbase shell

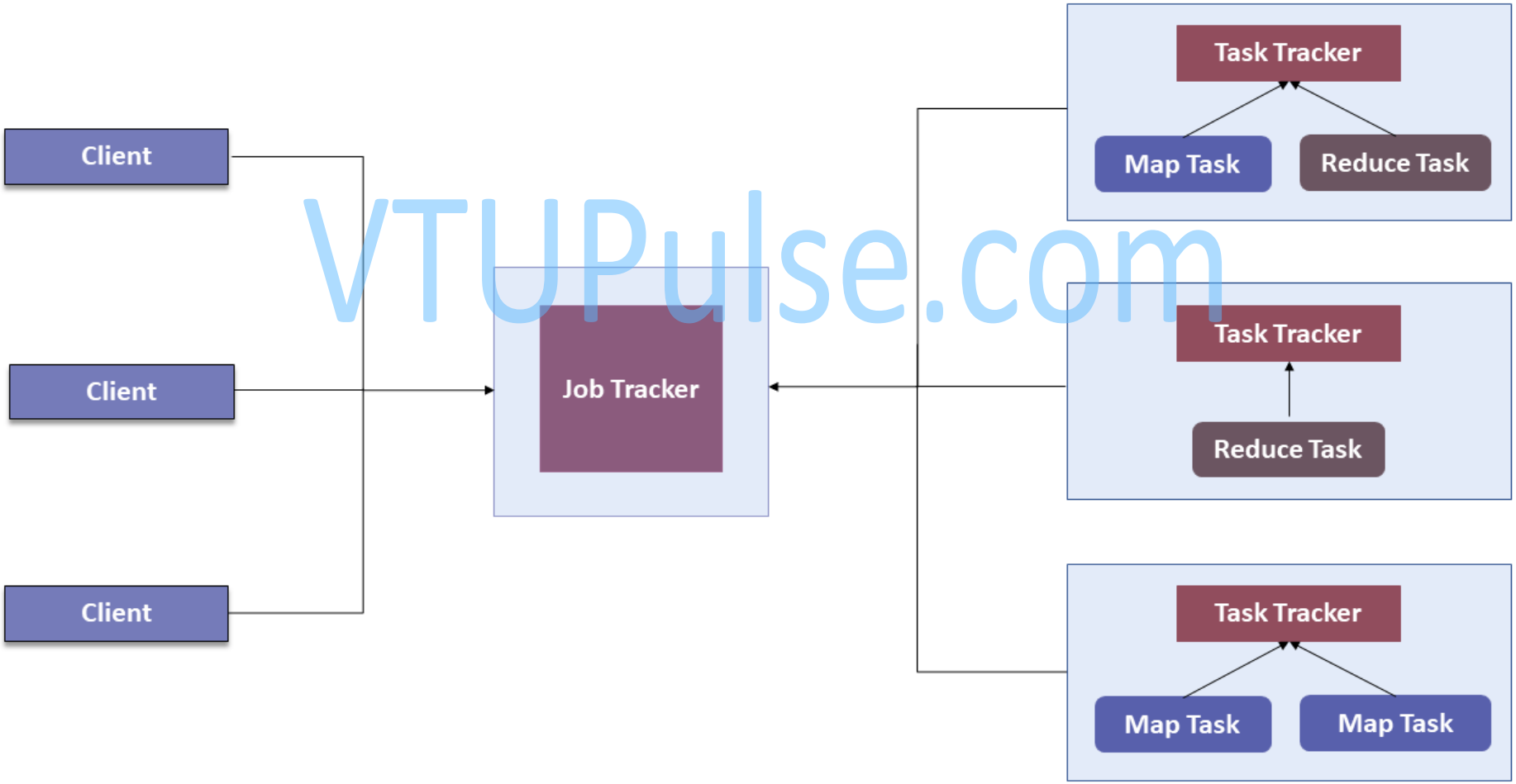
Web Interface

- http://localhost:60010

Why YARN?

- In Hadoop version 1.0 which is also referred to as MRV1(MapReduce Version 1), MapReduce performed both processing and resource management functions.
- It consisted of a Job Tracker which was the single master.
- The Job Tracker allocated the resources, performed scheduling and monitored the processing jobs.
- It assigned map and reduce tasks on a number of subordinate processes called the Task Trackers.
- The Task Trackers periodically reported their progress to the Job Tracker.

Why YARN?



Why YARN?

- This design resulted in scalability bottleneck due to a single Job Tracker. IBM mentioned in its article that according to Yahoo!, the practical limits of such a design are reached with a cluster of 5000 nodes and 40,000 tasks running concurrently.
- Apart from this limitation, the utilization of computational resources is inefficient in MRV1. Also, the Hadoop framework became limited only to MapReduce processing paradigm.

Why YARN?

- To overcome all these issues, YARN was introduced in Hadoop version 2.0 in the year 2012 by Yahoo and Hortonworks.
- The basic idea behind YARN is to relieve MapReduce by taking over the responsibility of Resource Management and Job Scheduling.
- YARN started to give Hadoop the ability to run non-MapReduce jobs within the Hadoop framework.

Why YARN ?

- MapReduce is a powerful **distributed framework** and **programming model** that allows batch-based parallelized work to be performed on a cluster of multiple nodes.
- Despite being very efficient at what it does, though, MapReduce has some disadvantages; principally that it's **batch-based**, and as a result isn't suited to real-time or even near-real-time data processing.
- Historically this has meant that processing models such as graph, iterative, and real-time data processing are not a natural fit for **MapReduce**.

Introduction to Hadoop YARN

Hadoop v1.0

MapReduce
Data Processing
& Resource Management

HDFS
Distributed File Storage

Hadoop v2.0

MapReduce

Other Data Processing Frameworks

YARN
Resource Management

HDFS
Distributed File Storage

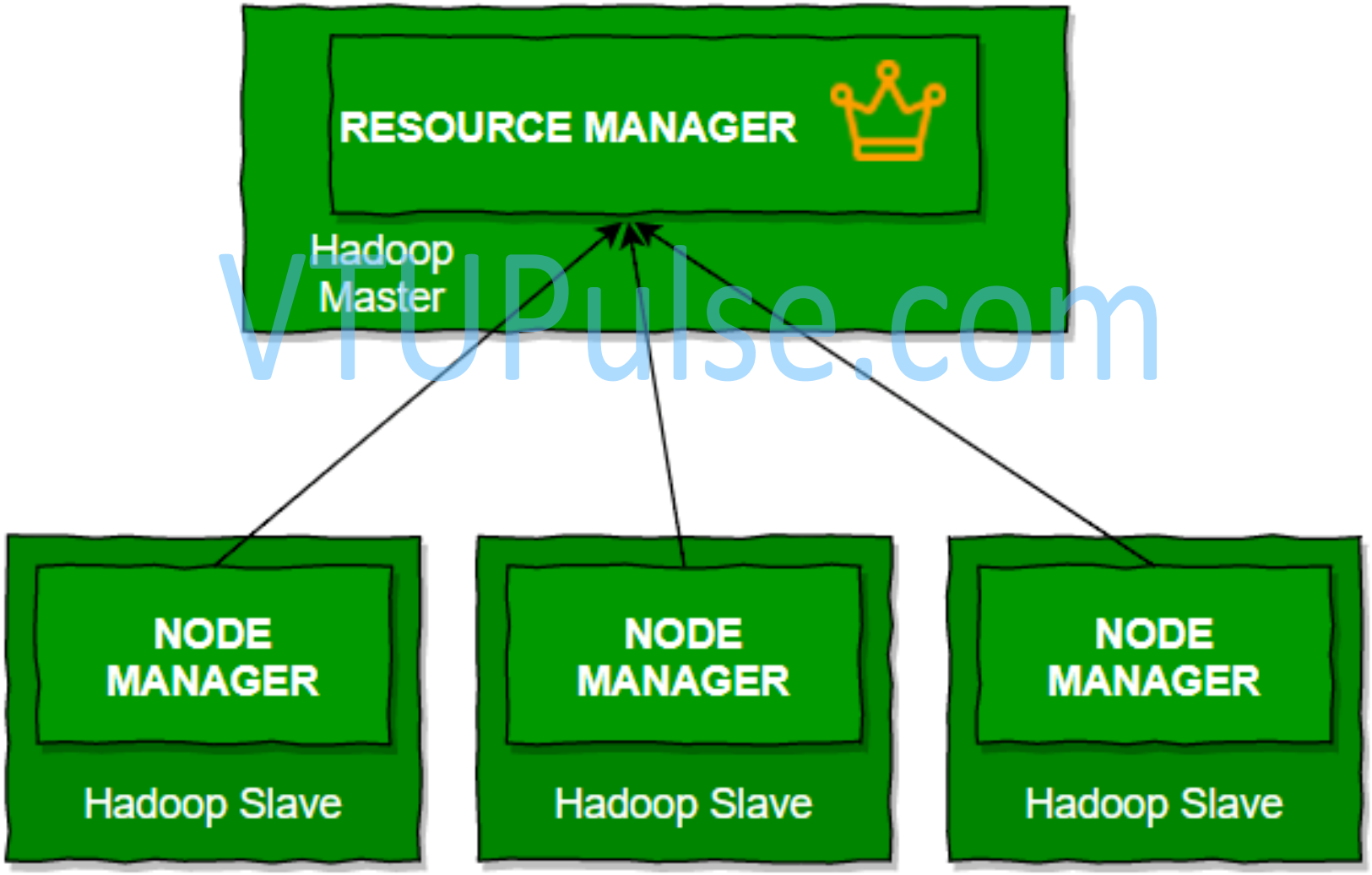
Components of YARN

Apart from Resource Management, YARN also performs Job Scheduling. YARN performs all your processing activities by allocating resources and scheduling tasks.

Apache Hadoop YARN Architecture consists of the following main components :

- **Resource Manager:** Runs on a master daemon and manages the resource allocation in the cluster.
- **Node Manager:** They run on the slave daemons and are responsible for the execution of a task on every single Data Node.

Components of YARN



Components of YARN - ResourceManager

- The ResourceManager is the YARN master process.
- A Hadoop cluster has a single ResourceManager (RM) for the entire cluster. Its sole function is to arbitrate all the available resources on a Hadoop cluster.
- ResourceManager tracks usage of resources, monitors the health of various nodes in the cluster, enforces resource-allocation invariants, and arbitrates conflicts among users.
- The components of resource manager
  - Scheduler
  - ApplicationsManager

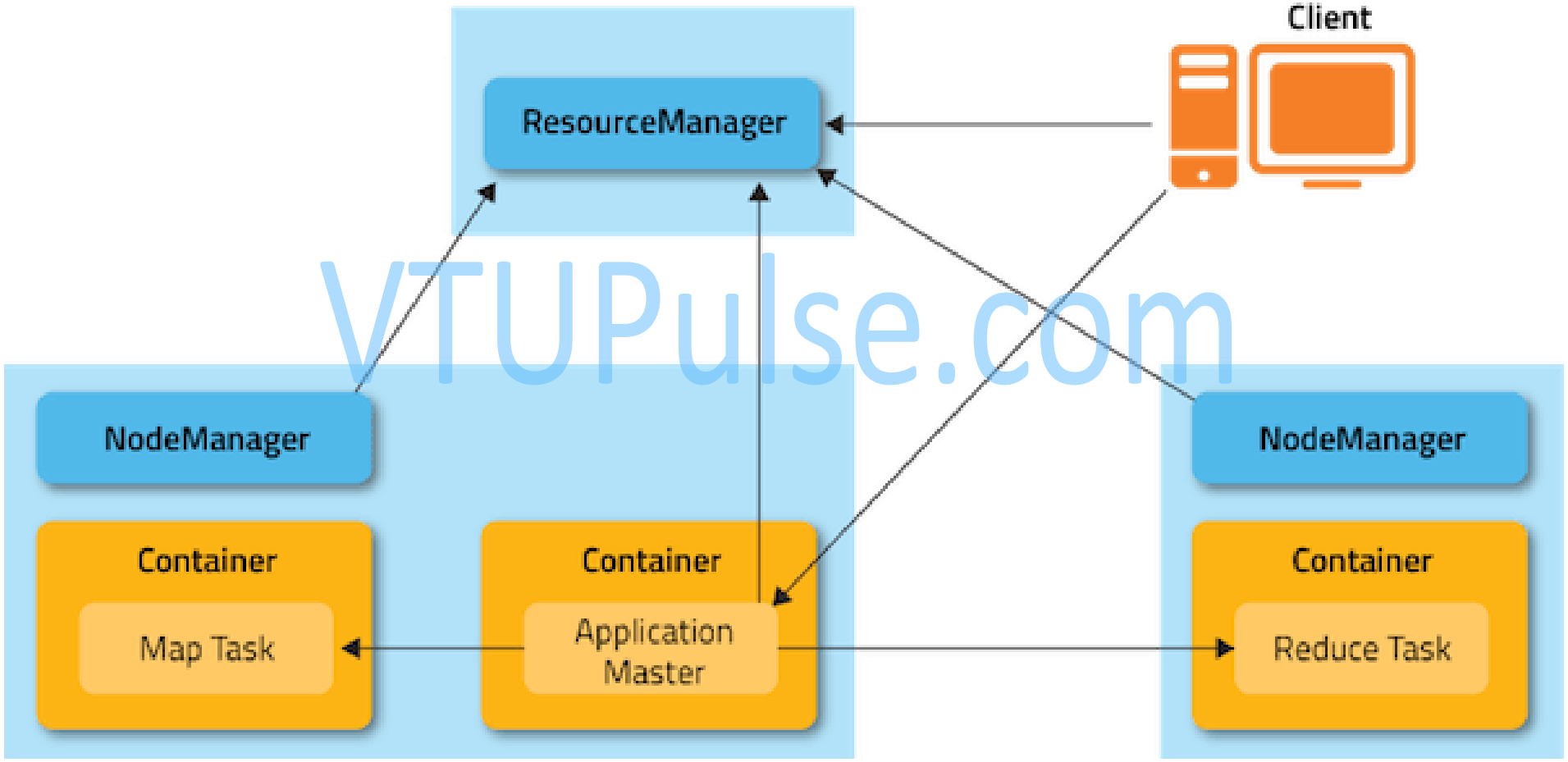
Components of YARN - NodeManager

- The NodeManager is the slave process of YARN.
- It runs on every data node in a cluster.
- Its job is to create, monitor, and kill containers.
- It services requests from the ResourceManager and ApplicationMaster to create containers, and it reports on the status of the containers to the ResourceManager. The ResourceManager uses the data contained in these status messages to make scheduling decisions for new container requests.
- On start-up, the NodeManager registers with the ResourceManager; it then sends heartbeats with its status and waits for instructions. Its primary goal is to manage application containers assigned to it by the ResourceManager.

YARN Applications

- The YARN framework/platform exists to manage applications, so let's take a look at what components a YARN application is composed of.
- A YARN application implements a specific function that runs on Hadoop. A YARN application involves 3 components:
  - Client
  - ApplicationMaster(AM)
  - Container

YARN Applications



YARN Applications - YARN Client

- Launching a new YARN application starts with a YARN client communicating with the ResourceManager to create a new YARN ApplicationMaster instance.
- Part of this process involves the YARN client informing the ResourceManager of the ApplicationMaster's physical resource requirements.

YARN Applications - YARN ApplicationMaster

- The ApplicationMaster is the master process of a YARN application.
- It doesn't perform any application-specific work, as these functions are delegated to the containers. Instead, it's responsible for managing the application-specific containers.
- Once the ApplicationMaster is started (as a container), it will periodically send heartbeats to the ResourceManager to affirm its health and to update the record of its resource demands.

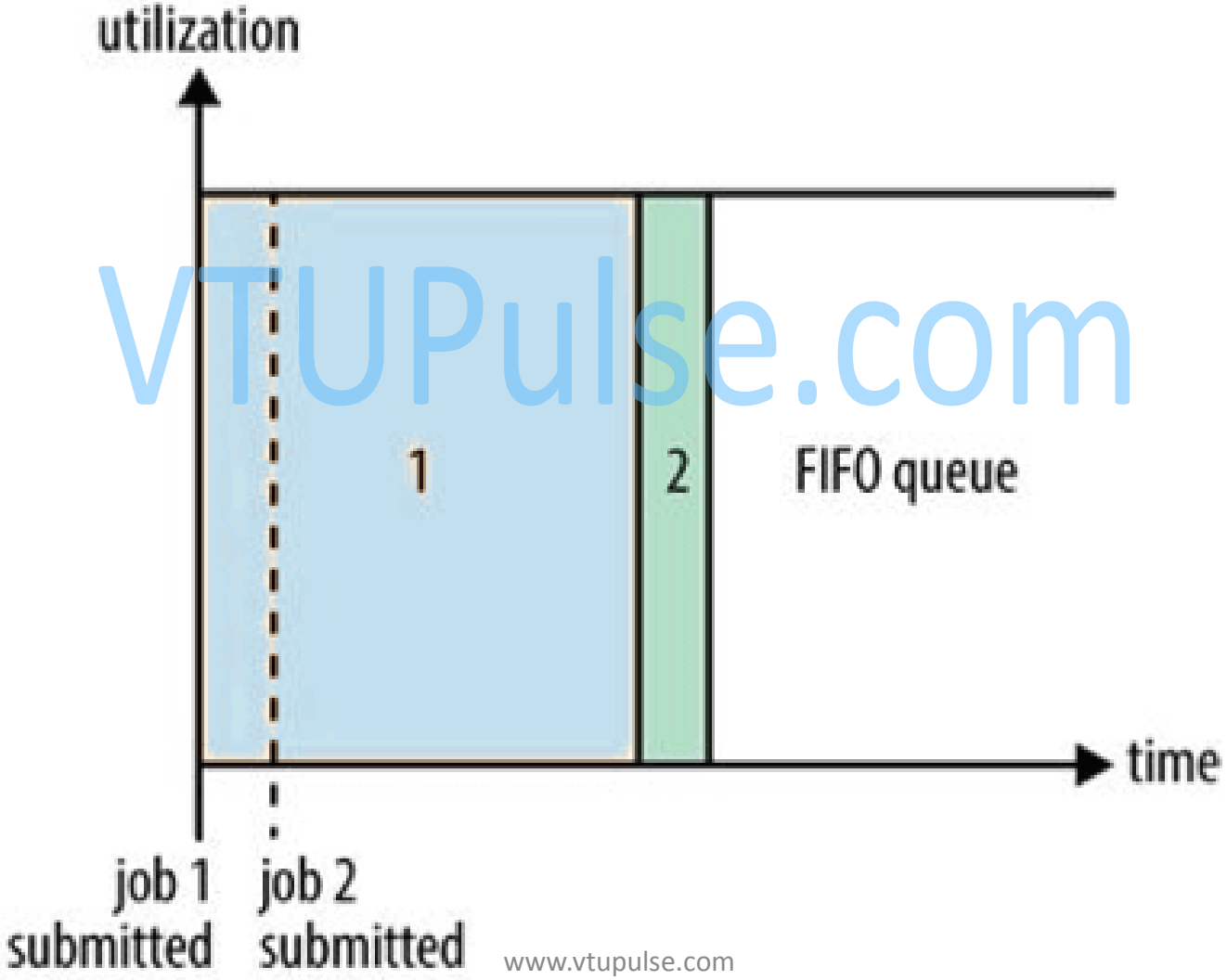
YARN Applications - YARN Container

- A container is an application-specific process that's created by a NodeManager on behalf of an ApplicationMaster.
- At the fundamental level, a container is a collection of physical resources such as RAM, CPU cores, and disks on a single node.

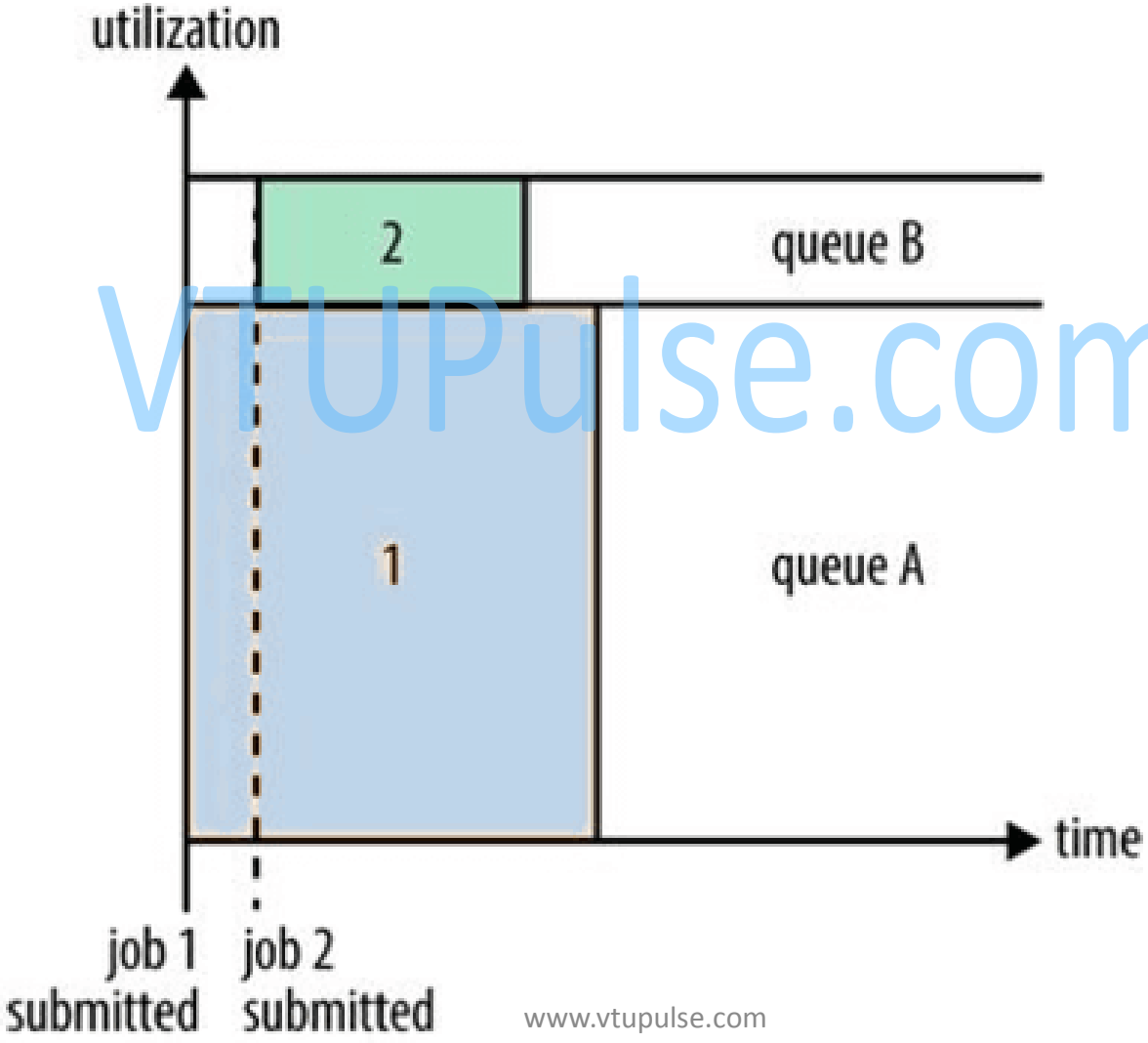
YARN scheduler policies

- In an ideal world, the requests that a YARN application makes would be granted immediately.
- In the real world, however, resources are limited, and on a busy cluster, an application will often need to wait to have some of its requests fulfilled.
- The FIFO scheduler
- The Capacity scheduler
- The Fair scheduler

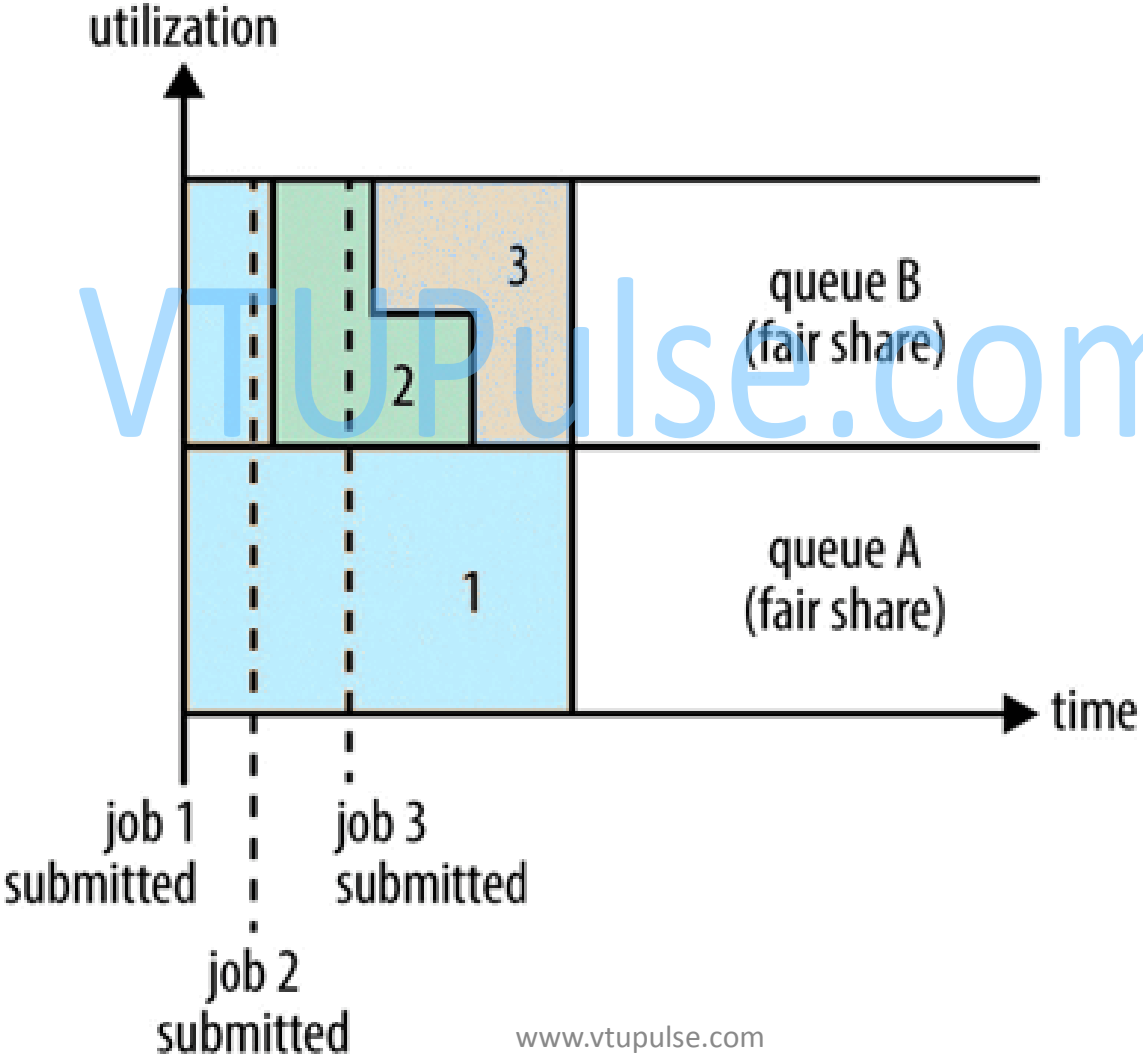
YARN scheduler policies - The FIFO scheduler



YARN scheduler policies - The Capacity scheduler



YARN scheduler policies - The Fair scheduler



Apache Ambari

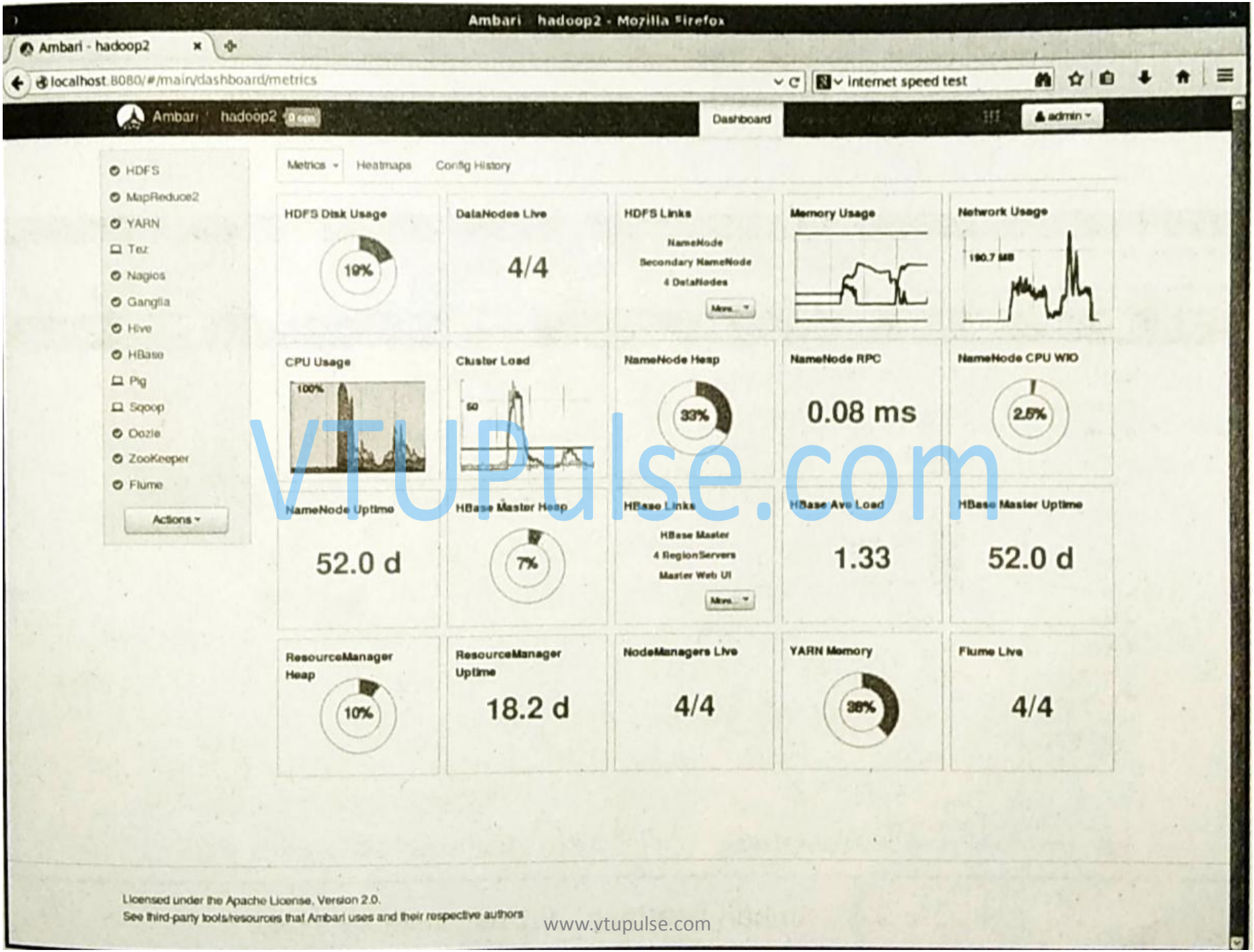
- Managing a Hadoop installation by hand can be tedious and time consuming. In addition to keeping configuration files synchronized across a cluster, starting, stopping, and restarting Hadoop services and dependent services in the right order is not a simple task.
- The Apache Ambari graphical management tool is designed to help you easily manage these and other Hadoop administrative issues. This chapter provides some basic navigation and usage scenarios for Apache Ambari

Apache Ambari

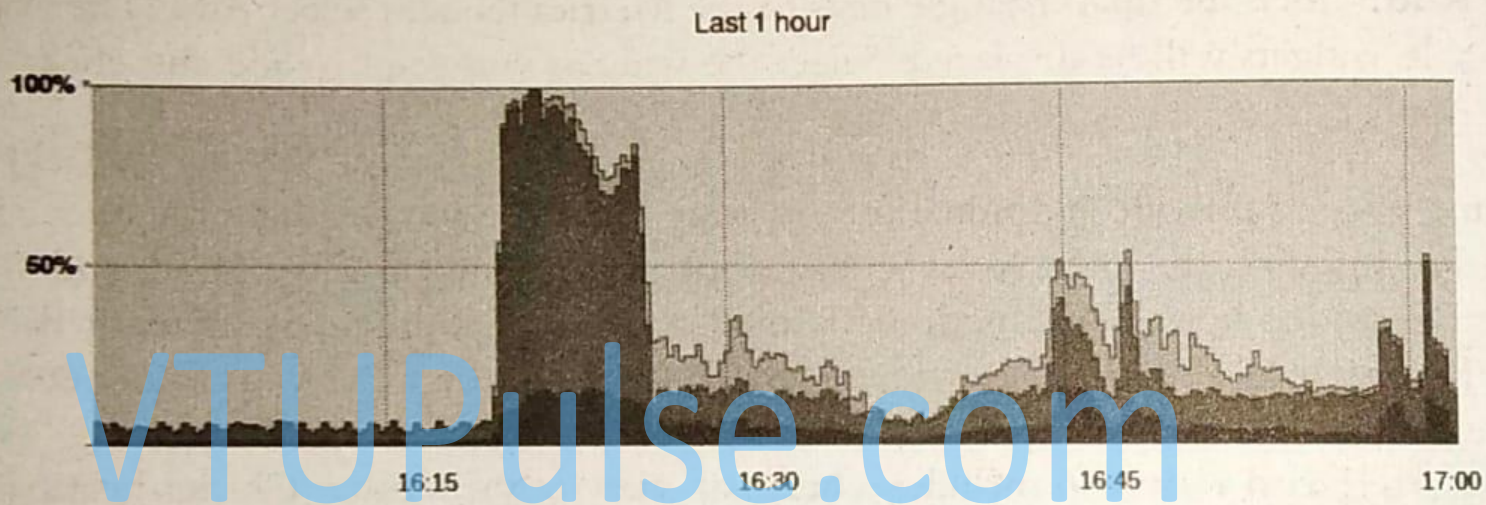
- Along with being an installation tool, Ambari can be used as a centralized point of administration for a Hadoop cluster. Using Ambari, the user can configure cluster services, monitor the status of cluster hosts (nodes) or services, visualize hotspots by service metric, start or stop services, and add new hosts to the cluster.
- All of these features infuse a high level of agility into the processes of managing and monitoring a distributed computing environment. Ambari also attempts to provide real-time reporting of important metrics.

Dashboard View

- Moving: Click and hold a widget while it is moved about the grid.
- Edit: Place the mouse on the widget and click the gray edit symbol in the upper-right corner of the widget. You can change several different aspects (including thresholds) of the widget.
- Remove: Place the mouse on the widget and click the X in the upper-left corner.
- Add: Click the small triangle next to the Metrics tab and select Add. The available widgets will be displayed. Select the widgets you want to add and click Apply.

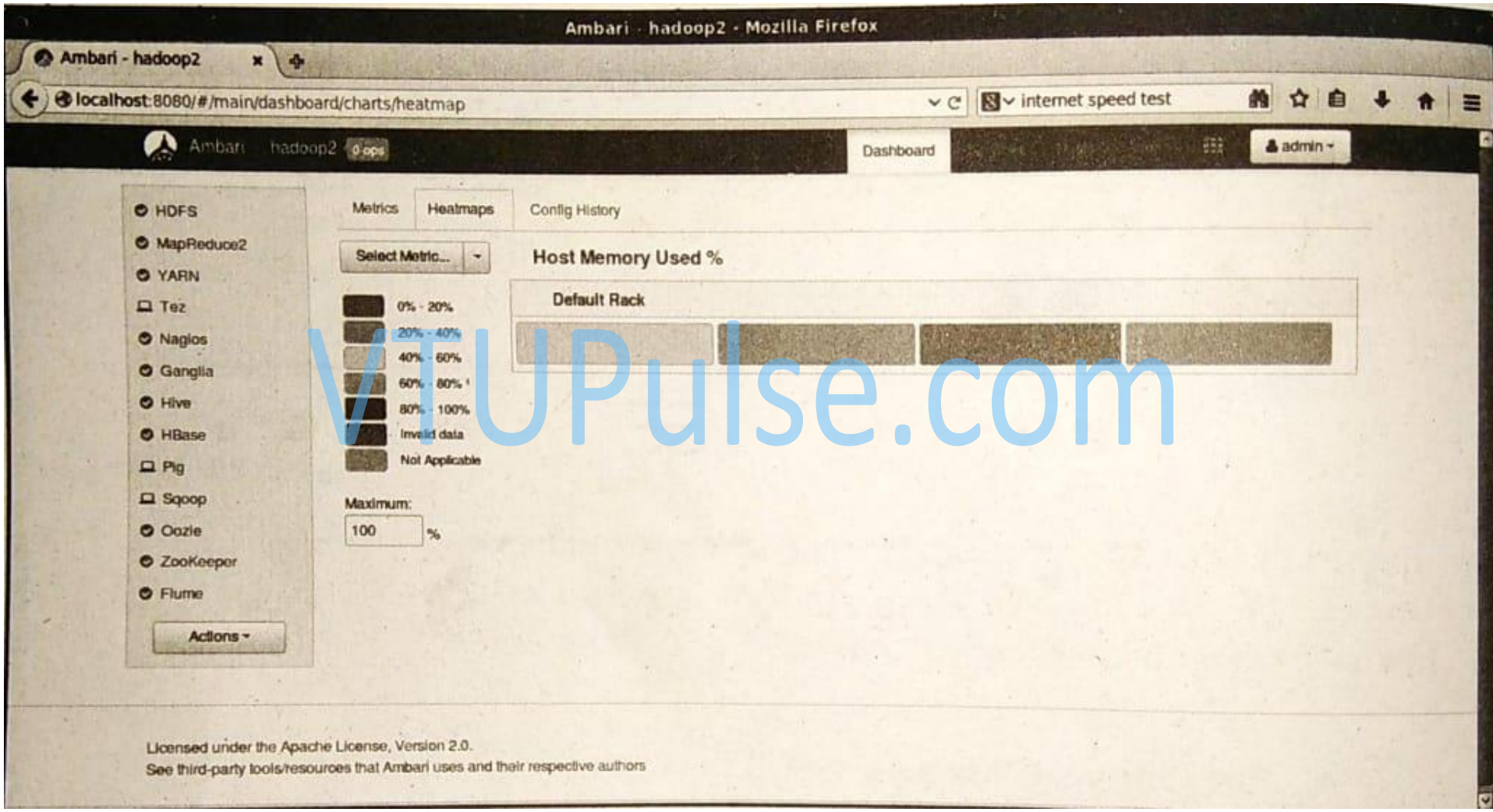


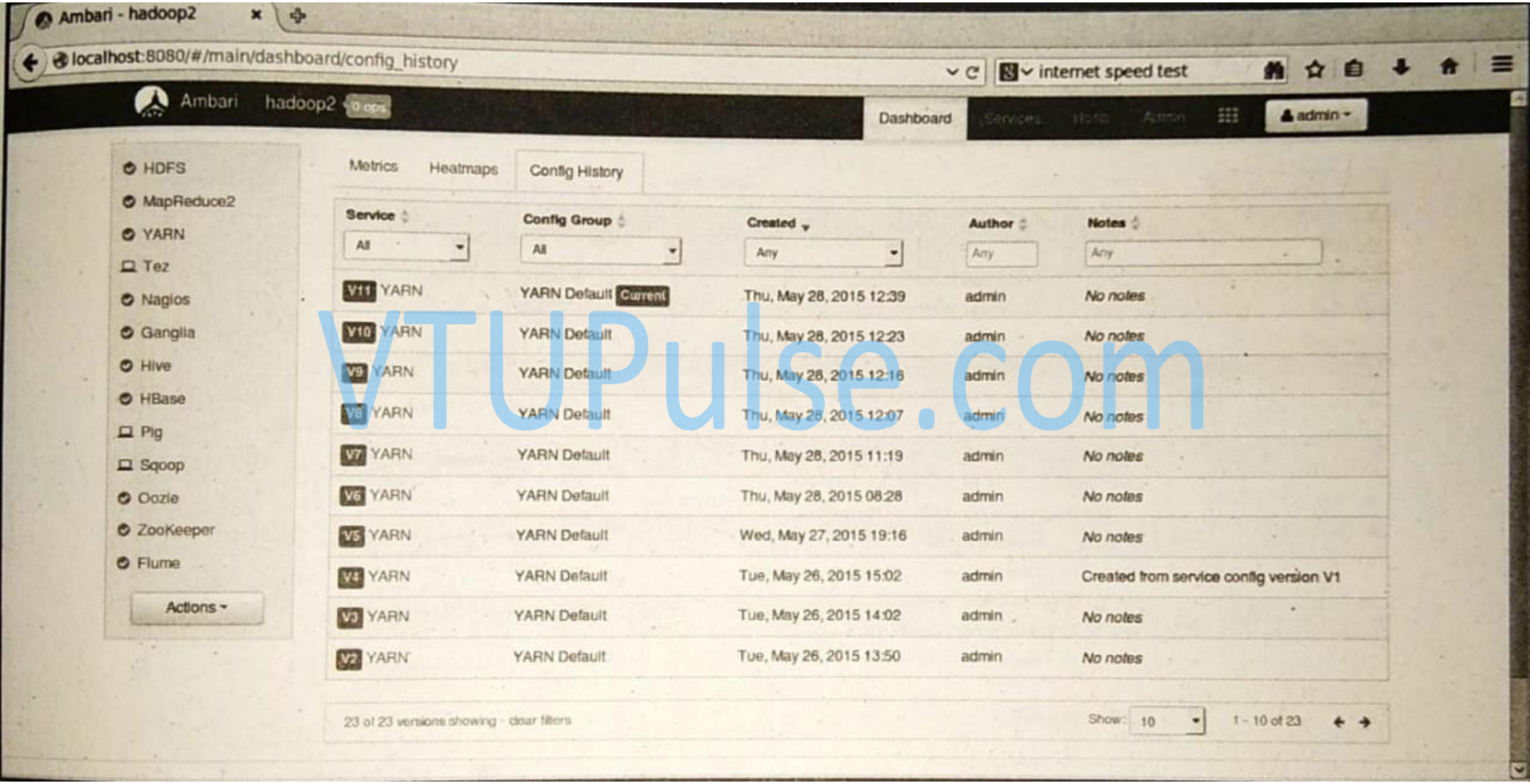
CPU Usage



✓ Idle	min	0.35%	avg	75.156%	max	96.31%
✓ Wait	min	0.075%	avg	4.582%	max	18.472%
✓ User	min	2.175%	avg	15.278%	max	84.25%
✓ System	min	1.22%	avg	4.981%	max	16.092%
✓ Nice	min	0 %	avg	0 %	max	0%

OK





Services View

- The Services menu provides a detailed look at each service running on the cluster. It also provides a graphical method for configuring each service (i.e., instead of handediting the /etc/hadoop/conf XML files).
- The summary tab provides a current Summary view of important service metrics and an Alerts and Health Checks sub-window.
- Similar to the Dashboard view, the currently installed services are listed on the leftside menu. To select a service, click the service name in the menu. When applicable, each service will have its own Summary, Alerts and Health Monitoring, and Service Metrics windows.

Ambari - hadoop2

localhost:8080/#/main/services/HDFS/summary

internet speed test

Ambari hadoop2 0 ops

Services

admin

HDFS
MapReduce2
YARN
Tez
Nagios
Ganglia
Hive
HBase
Pig
Sqoop
Oozie
ZooKeeper
Flume

Actions

Summary Configs Quick Links

Service Actions

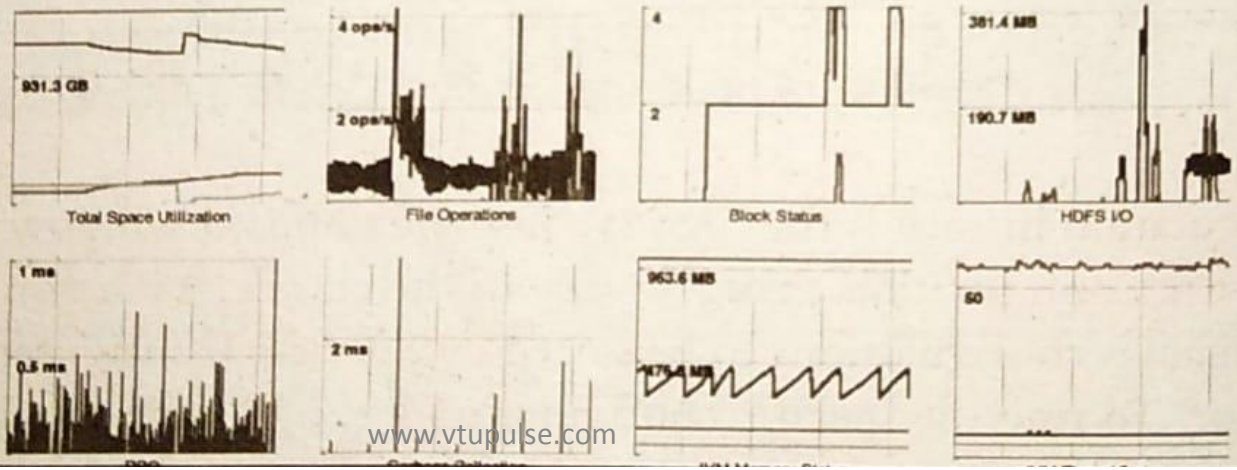
Summary

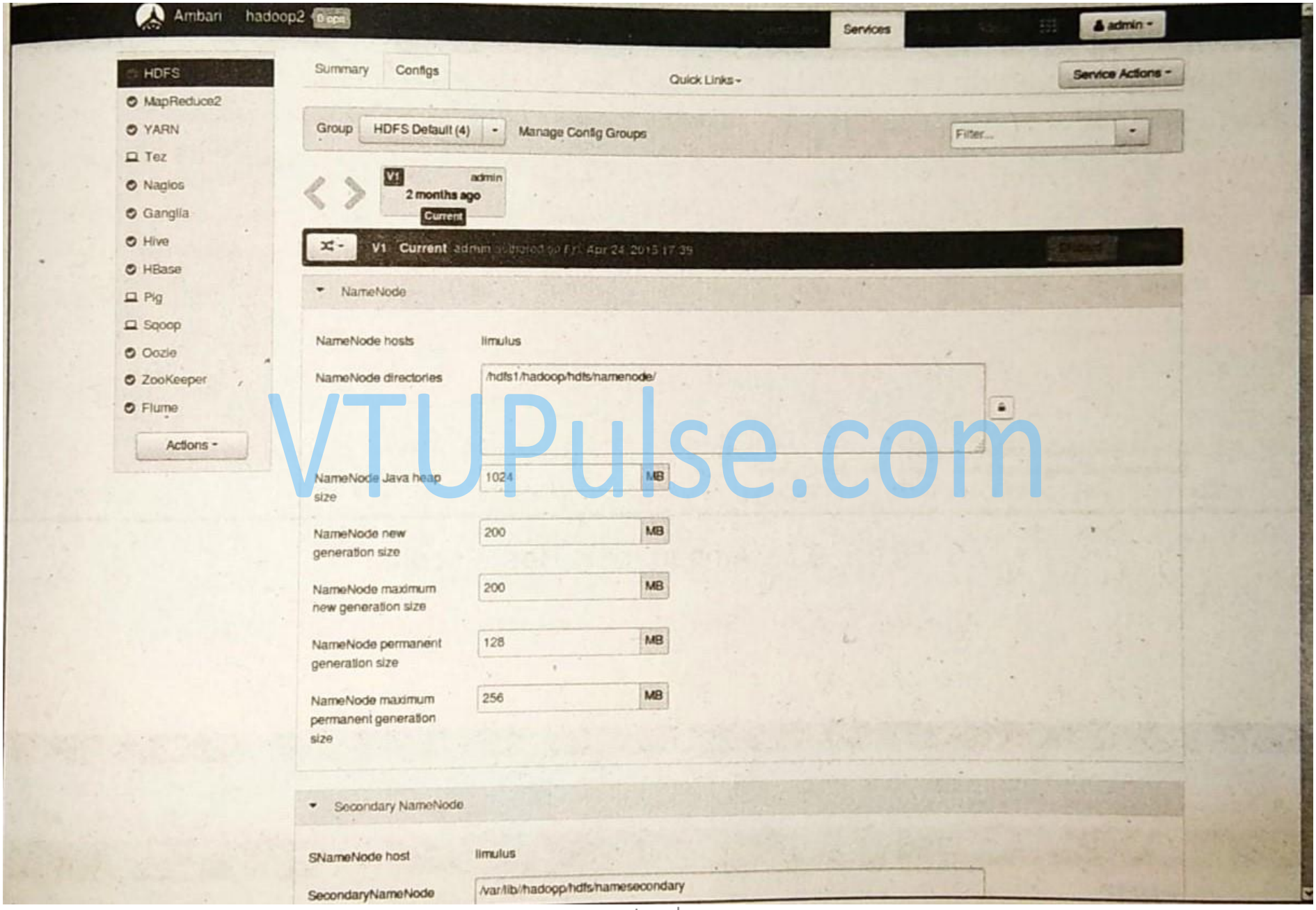
NameNode	Started
SNameNode	Started
DataNodes	4/4 DataNodes Live
NameNode Uptime	51.97 days
NameNode Heap	317.4 MB / 1004.0 MB (31.6% used)
DataNodes Status	4 live / 0 dead / 0 decommissioning
Disk Usage (DFS Used)	78.9 GB / 1.4 TB (5.64%)
Disk Usage (Non DFS Used)	211.7 GB / 1.4 TB (15.12%)
Disk Usage (Remaining)	1.1 TB / 1.4 TB (79.25%)
Blocks (total)	1330
Block Errors	0 corrupt / 0 missing / 2 under replicated
Total Files + Directories	300622
Upgrade Status	No pending upgrade
Safe Mode Status	Not in safe mode

Alerts and Health Checks

Percent DataNodes with space available OK: total:<4>, affected:<0>	OK for 2 months
Percent DataNodes live OK: total:<4>, affected:<0>	OK for 2 months
Secondary NameNode process TCP OK - 0.001 second response time on port 50090	OK for 2 months
HDFS capacity utilization OK: DFSUsedGB:<78.7>, DFSTotalGB:<1188.4>	OK for 2 months
Blocks health OK: missing_blocks:<0>, total_blocks:<1330>	OK for 2 months
NameNode RPC latency on limulus OK: RpcQueueTime_avg_time:<0> Secs, RpcProcessingTime_avg_time:<0> Secs	OK for 2 months
NameNode process on limulus TCP OK - 0.000 second response time on port 8020	OK for 2 months

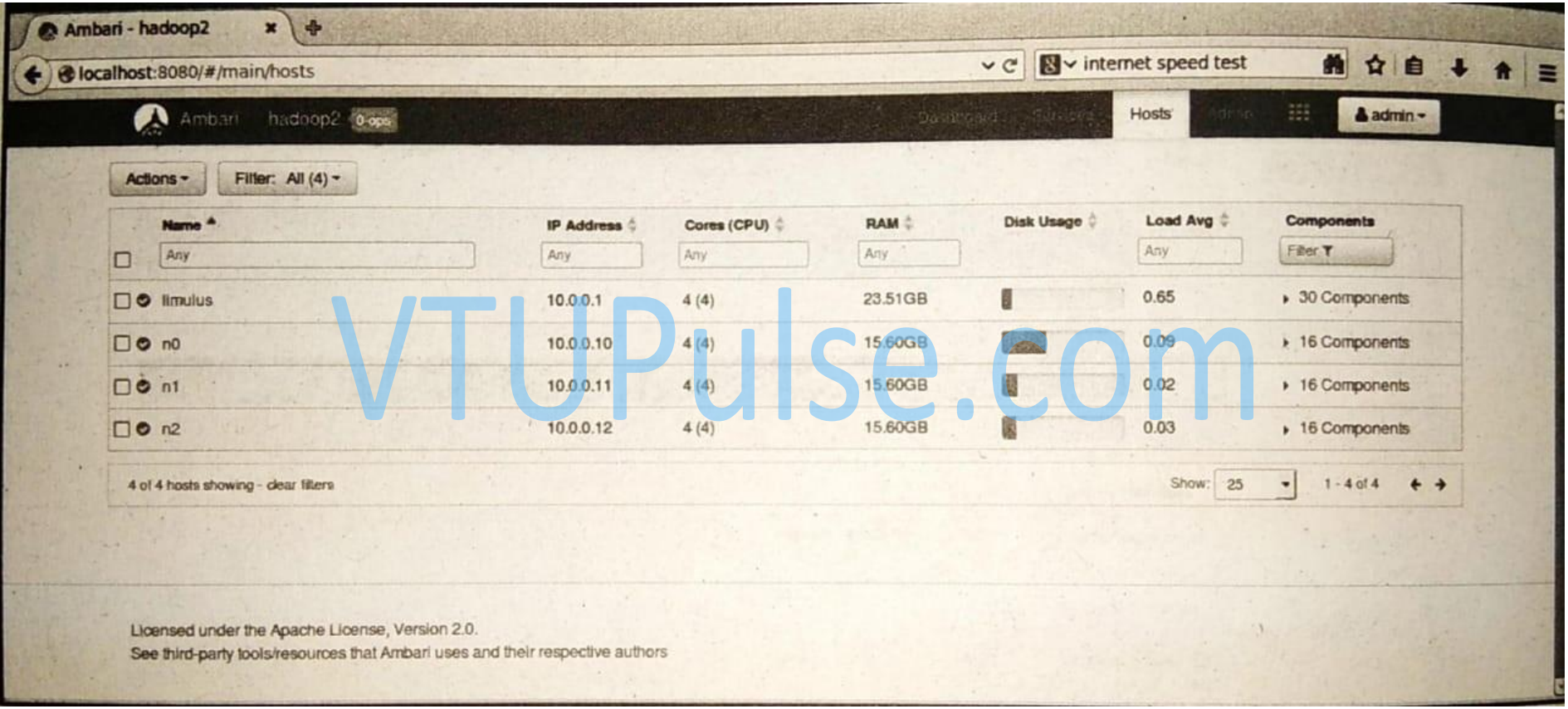
HDFS Service Metrics





Hosts View

- The host name, IP address, number of cores, memory, disk usage, current load average, and Hadoop components are listed in this window in tabular form.
- To display the Hadoop components installed on each host, click the links in the rightmost columns. You can also add new hosts by using the Actions pull-down menu.



Ambari - hadoop2 - Mozilla Firefox

Ambari - hadoop2

localhost:8080/#/main/hosts/n0/summary

internet speed test

Ambari hadoop2 0 ops

Dashboard Services Hosts Alerts admin

n0 No alerts

Back

Summary Configs

Host Actions

Components

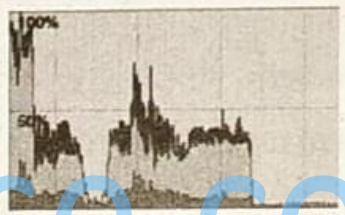
+ Add

DataNode / HDFS	Started
Flume / Flume	Started
Ganglia Monitor / Ganglia	Started
RegionServer / HBase	Started
NodeManager / YARN	Started
Clients / HBase Client , HCat Client , HDFS Client , Hive Client , MapReduce2 Client , Oozie Client , Pig , Sqoop , Tez Client , YARN Client , ZooKeeper Client	Installed

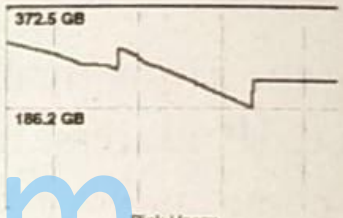
Summary

Hostname: n0
IP Address: 10.0.0.10
OS: centos6 (x86_64)
Cores (CPU): 4 (4)
Disk: 138.42GB/378.07GB (36.61% used)
Memory: 15.60GB
Load Avg: 0.07
Heartbeat: a moment ago

Host Metrics

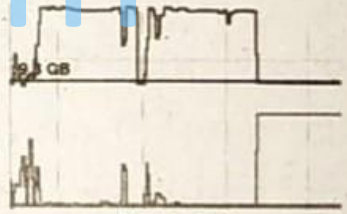


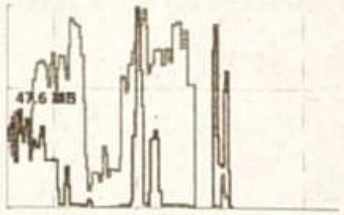
CPU Usage

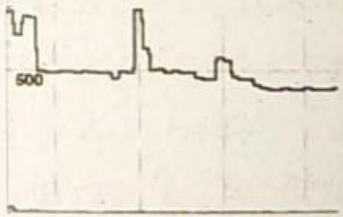


Disk Usage

Load


Memory Usage


Network Usage



Processes

Admin View

- The Administration (Admin) view provides three options. The first, as shown in Figure, displays a list of installed software. This Repositories listing generally reflects the version of Hortonworks Data Platform (HDP) used during the installation process.
- Second, the Service Accounts option lists the service accounts added when the system was installed. These accounts are used to run various services and tests for Ambari.
- The third option, Security, sets the security on the cluster. A fully secured Hadoop cluster is important in many instances and should be explored if a secure environment is needed. This aspect of Ambari is beyond the scope of this book.

Ambari hadoop2 0 ops Dashboard Services Hosts Admin admin

- Repositories
- Service Accounts
- Security

Cluster Stack Version: HDP-2.2

Service	Version	Description
Falcon	0.6.0.2.2.0.0	Data management and processing platform
Flume	1.5.2.2.2.0.0	A distributed service for collecting, aggregating, and moving large amounts of streaming data into HDFS
Ganglia	3.5.0	Ganglia Metrics Collection system (RRDTool will be installed too)
HBase	0.98.4.2.2.0.0	Non-relational distributed database and centralized service for configuration management & synchronization
HDFS	2.6.0.2.2.0.0	Apache Hadoop Distributed File System
Hive	0.14.0.2.2.0.0	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
Kafka	0.8.1.2.2.0.0	A high-throughput distributed messaging system
Knox	0.5.0.2.2.0.0	Provides a single point of authentication and access for Apache Hadoop services in a cluster
Nagios	3.5.0	Nagios Monitoring and Alerting system
Oozie	4.1.0.2.2.0.0	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the optional Oozie Web Console which relies on and will install the ExtJS Library.
Pig	0.14.0.2.2.0.0	Scripting platform for analyzing large datasets
Slider	0.60.0.2.2.0.0	A framework for deploying, managing and monitoring existing distributed applications on YARN.
Sqoop	1.4.5.2.2.0.0	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
Storm	0.9.3.2.2.0.0	Apache Hadoop Stream processing framework
Tez	0.5.2.2.2.0.0	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
YARN + MapReduce2	2.6.0.2.2.0.0	Apache Hadoop NextGen MapReduce (YARN)
ZooKeeper	3.4.6.2.2.0.0	Centralized service which provides highly reliable distributed coordination

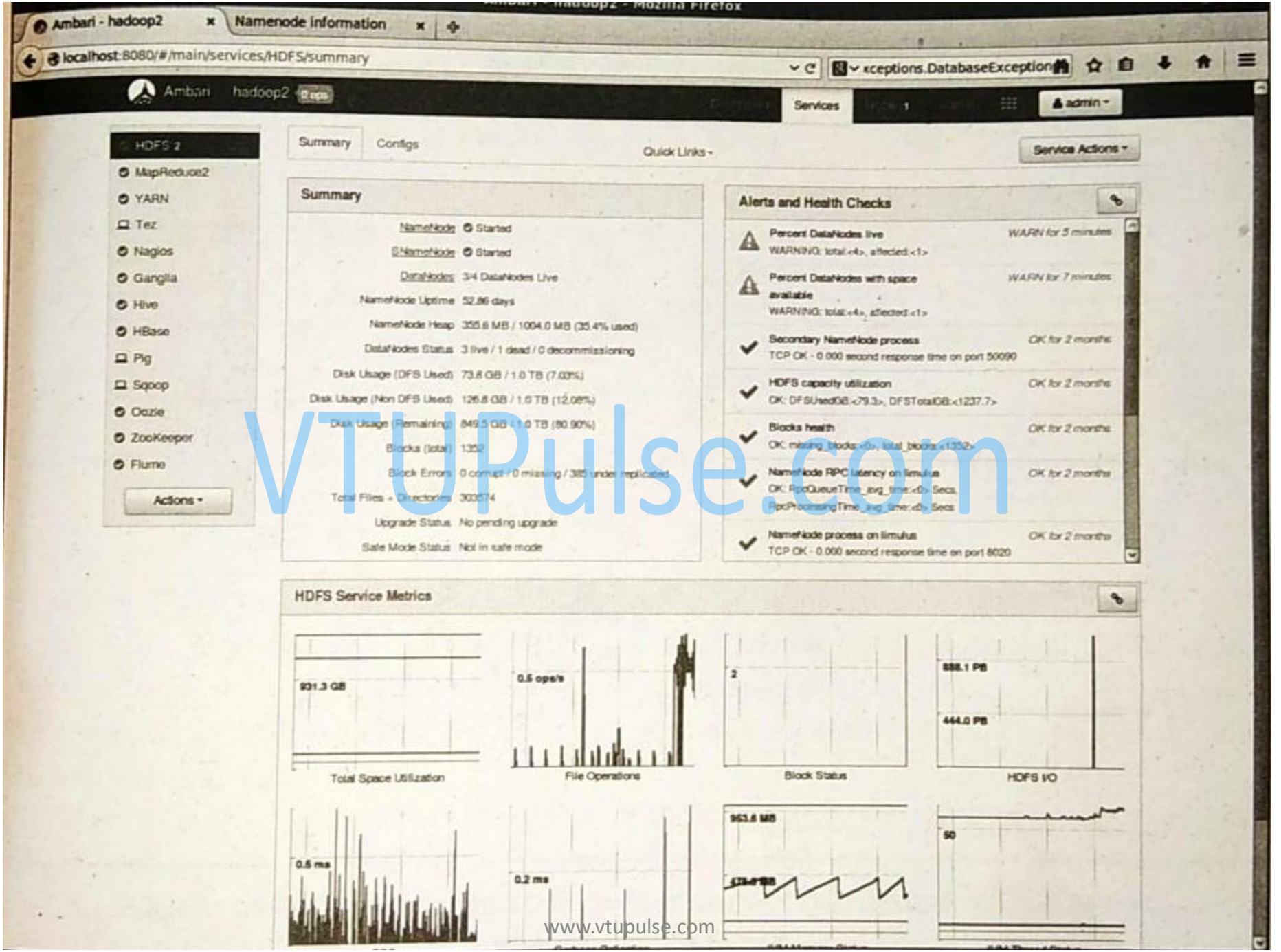
Repositories

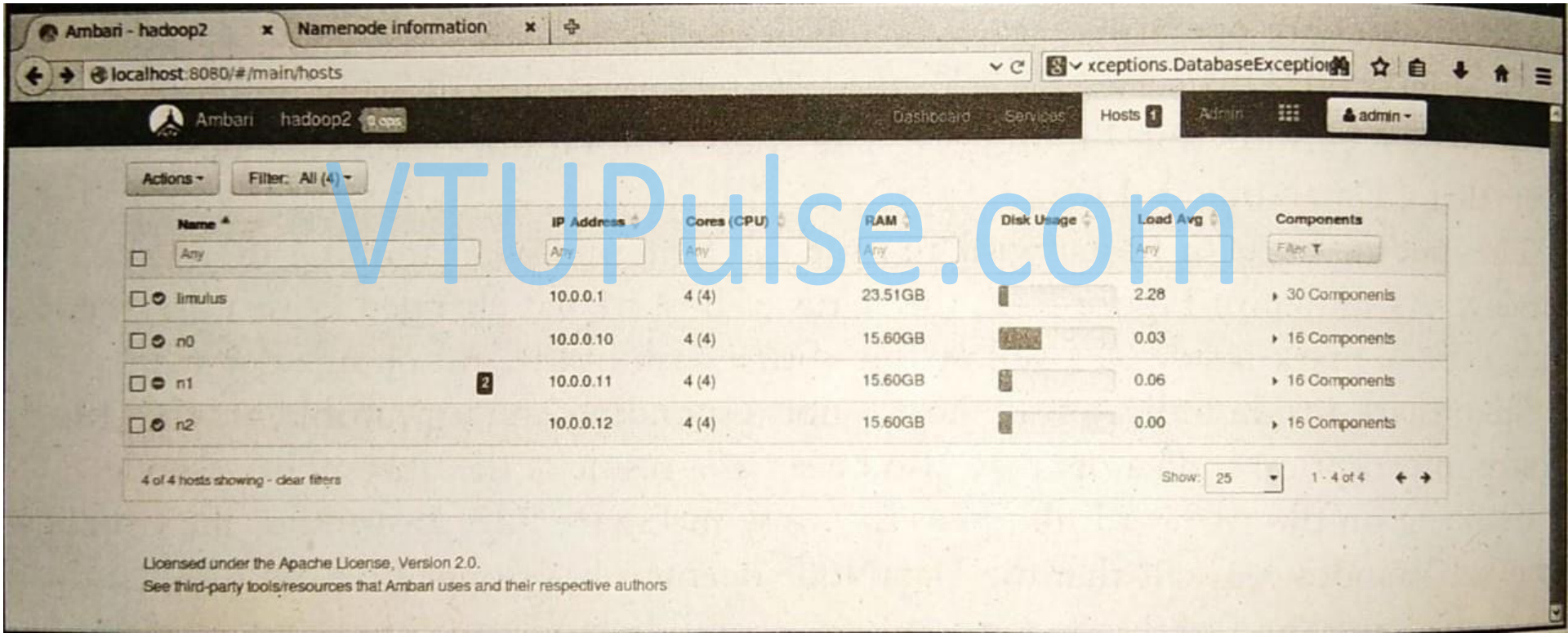
2.2

OS	Name	Base URL

Managing Hadoop Services

- During the course of normal Hadoop cluster operation, services may fail for any number of reasons.
- Ambari monitors all of the Hadoop services and reports any service interruption to the dashboard.
- In addition, when the system was installed, an administrative email for the Nagios monitoring system was required. All service interruption notifications are sent to this email address.
- Figure 9.10 shows the Ambari dashboard reporting a down DataNode.





Ambari - hadoop2 | Namenode information

localhost:8080/#/main/hosts/n1/summary

Ambari hadoop2 | Dashboard | Services | Hosts 1 | Admin | admin

n1

Back

Summary | Configs | Host Actions

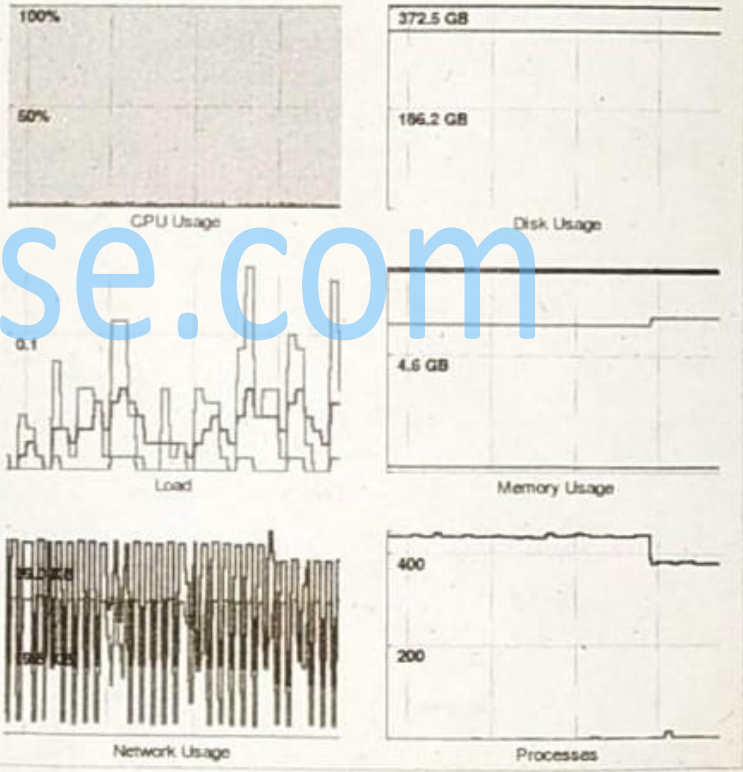
Components + Add

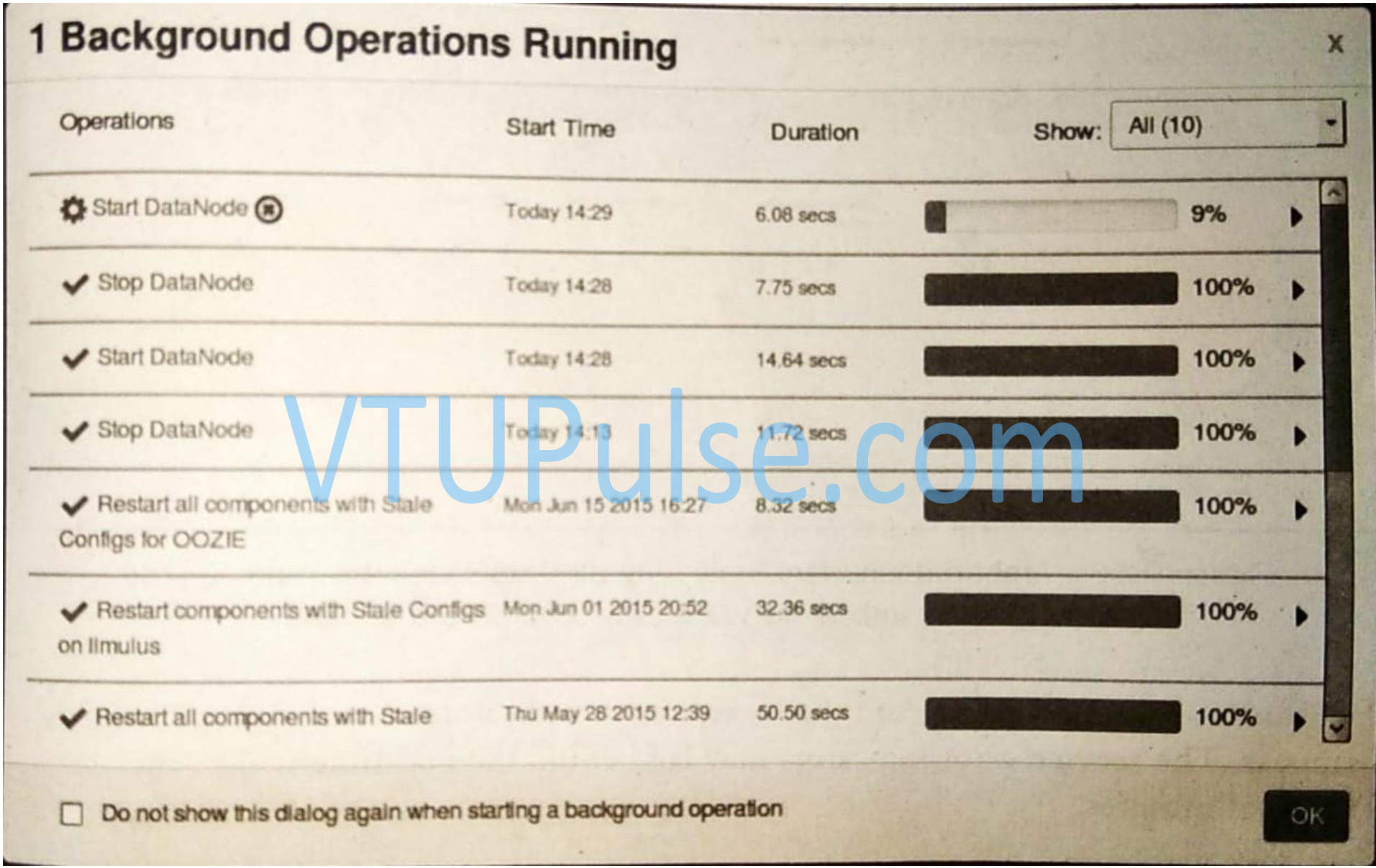
DataNode / HDFS	Stopped
Flume / Flume	Started
Ganglia Monitor / Ganglia	Started
RegionServer / HBase	Started
NodeManager / YARN	Started
Clients / HBase Client, HCat Client, HDFS Client, Hive Client, MapReduce2 Client, Oozie Client, Pig, Sqoop, Tez Client, YARN Client, ZooKeeper Client	Installed

Summary

Hostname: n1
IP Address: 10.0.0.11
OS: centos6 (x86_64)
Cores (CPU): 4 (4)
Disk: 46.37GB/378.07GB (12.26% used)
Memory: 15.60GB
Load Avg: 0.06
Heartbeat: a moment ago

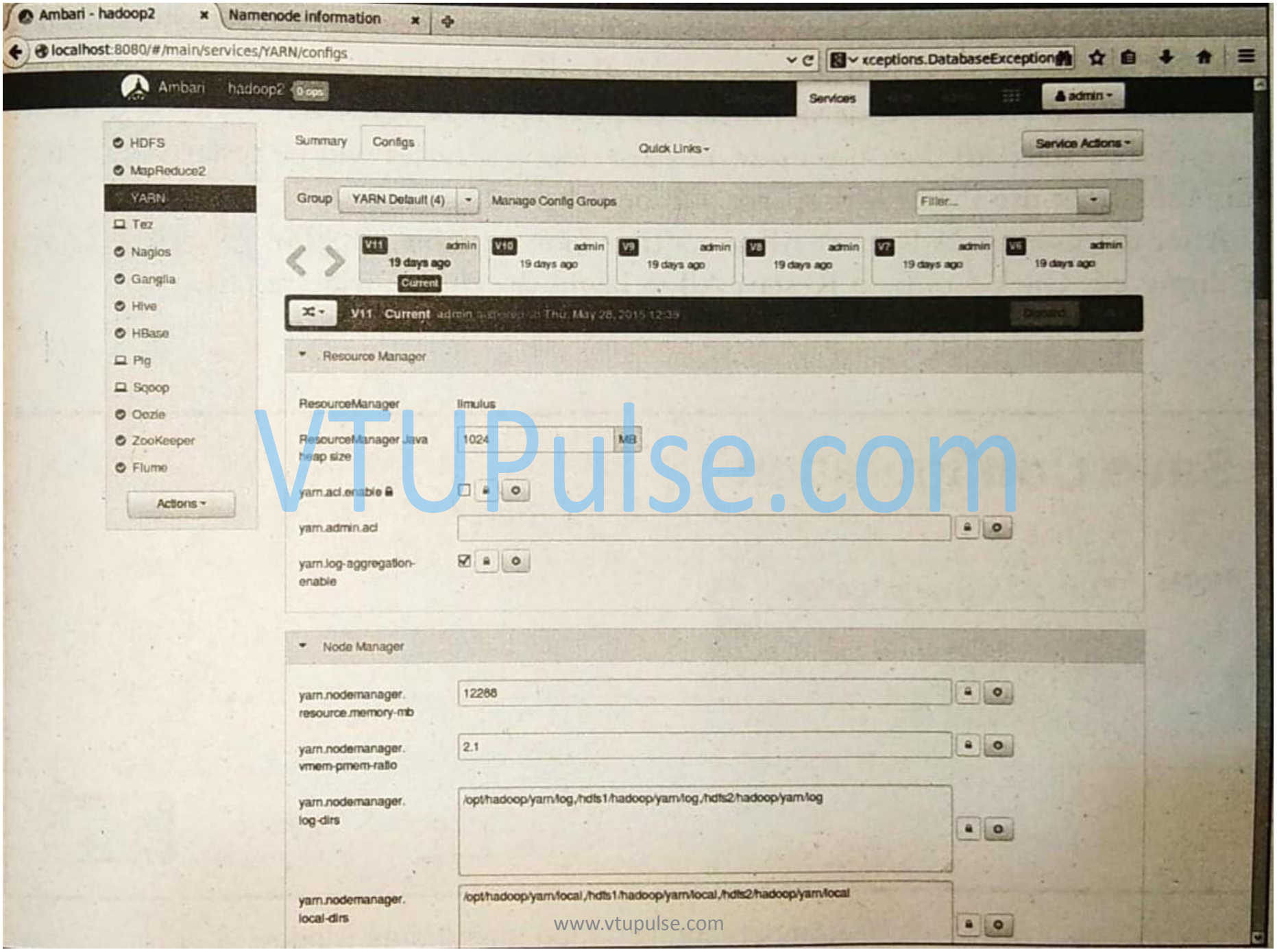
Host Metrics

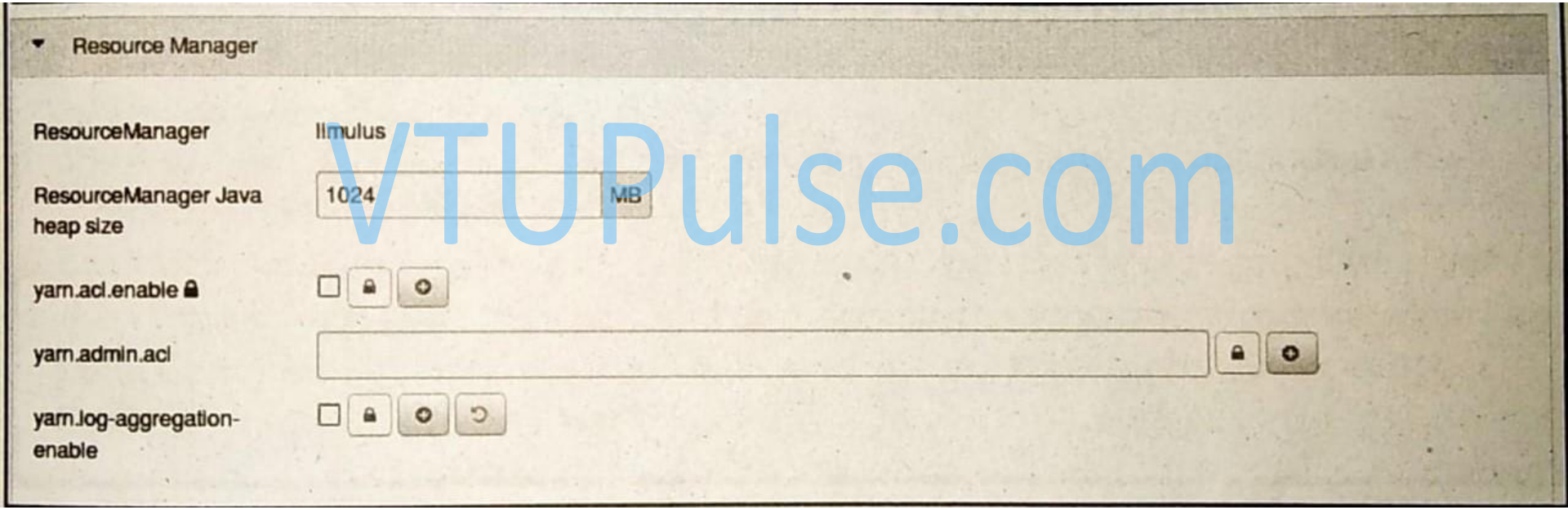


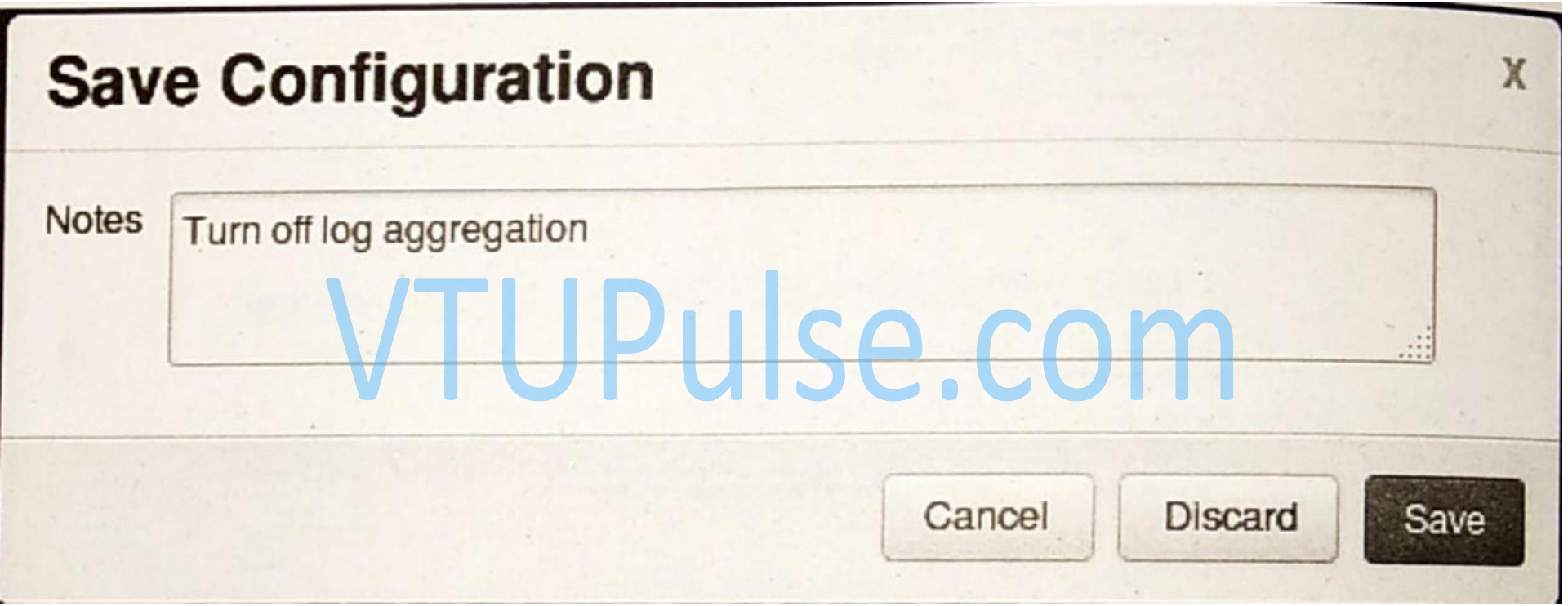


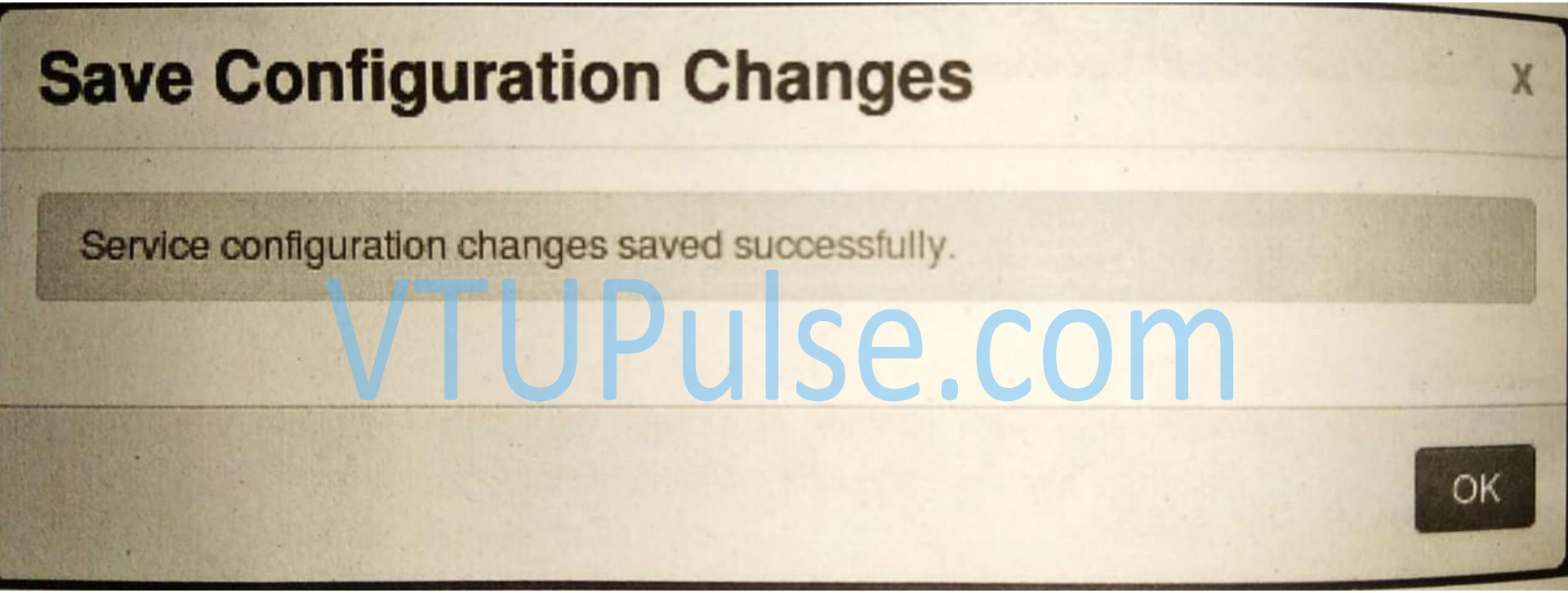
Changing Hadoop Properties

- One of the challenges of managing a Hadoop cluster is managing changes to Clusterwide configuration properties. In addition to modifying a large number of properties, making changes to a property often requires restarting daemons (and dependent daemons) across the entire cluster. This process is tedious and time consuming. Fortunately, Ambari provides an easy way to manage this process.
- As described previously, each service provides a Configs tab that opens a form displaying all the possible service properties. Any service property can be changed (or added) using this interface. As an example, the configuration properties for the YARN scheduler are shown in Figure.









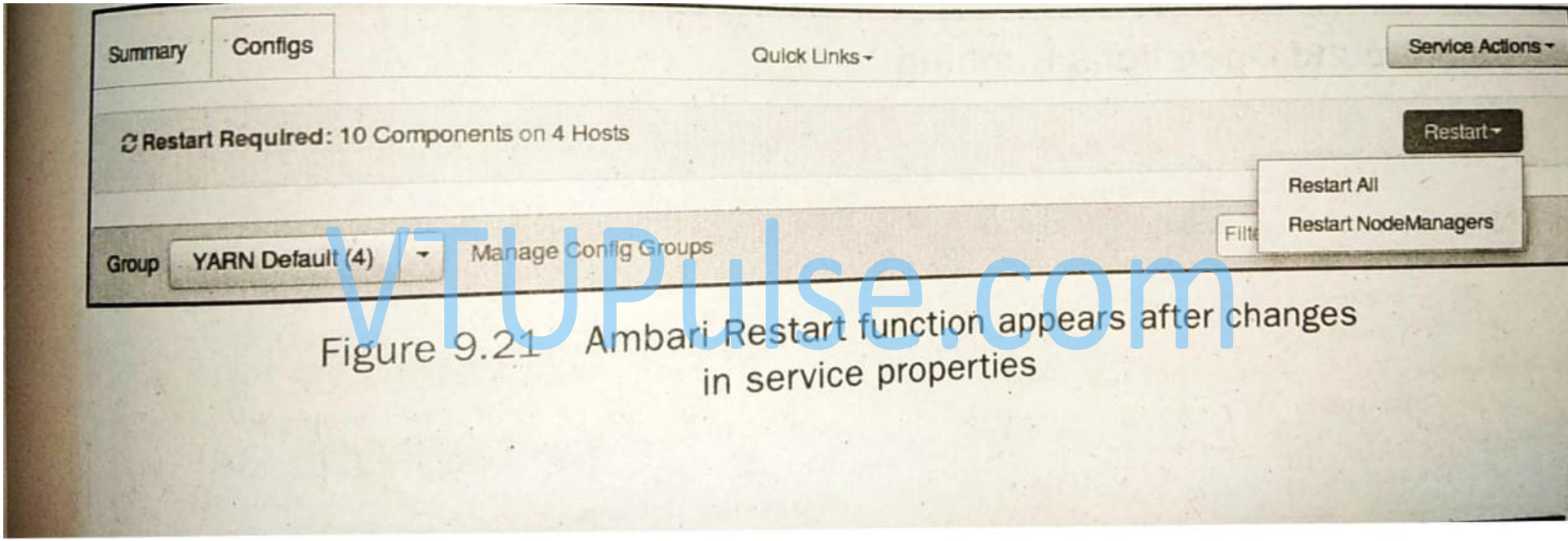
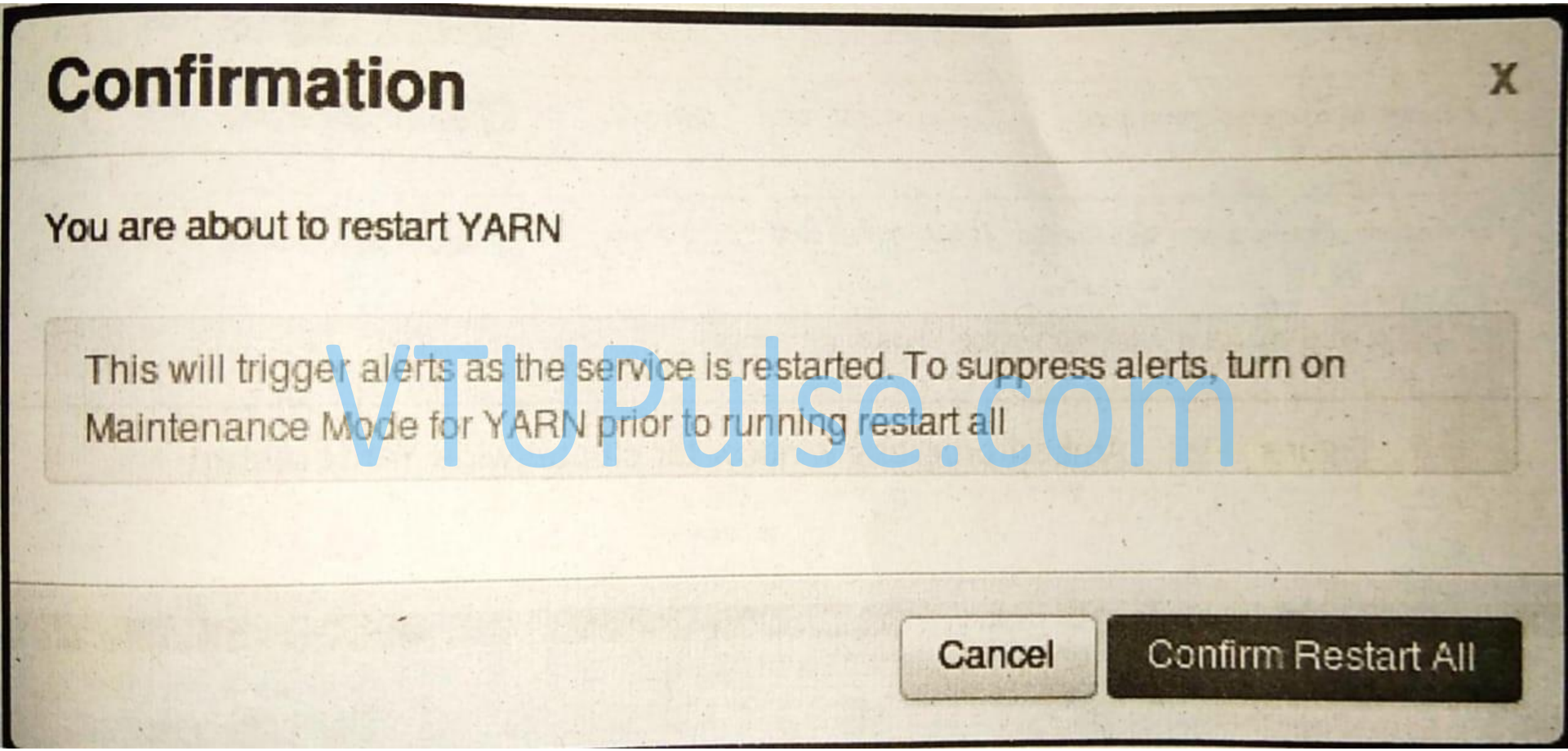
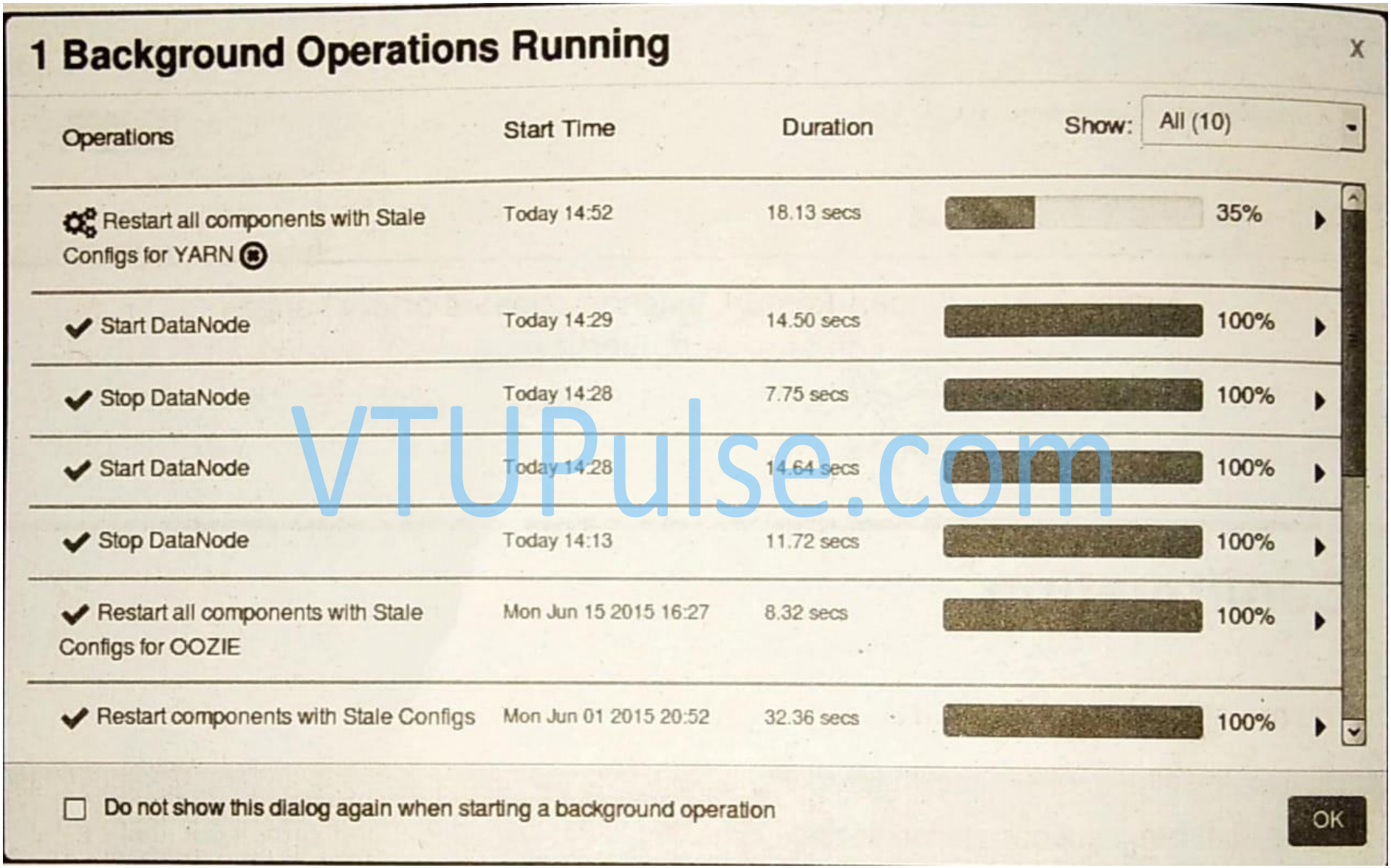
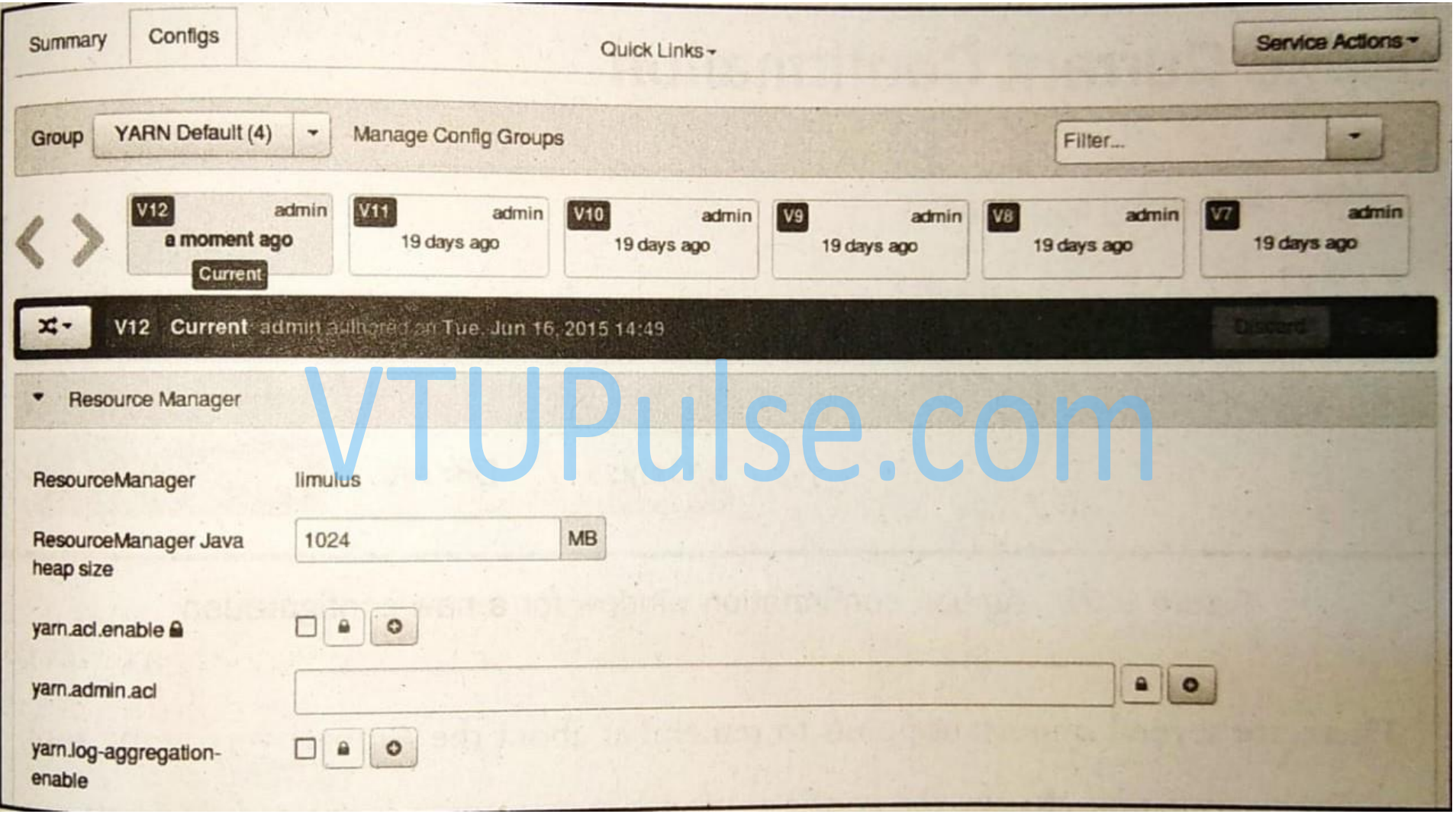
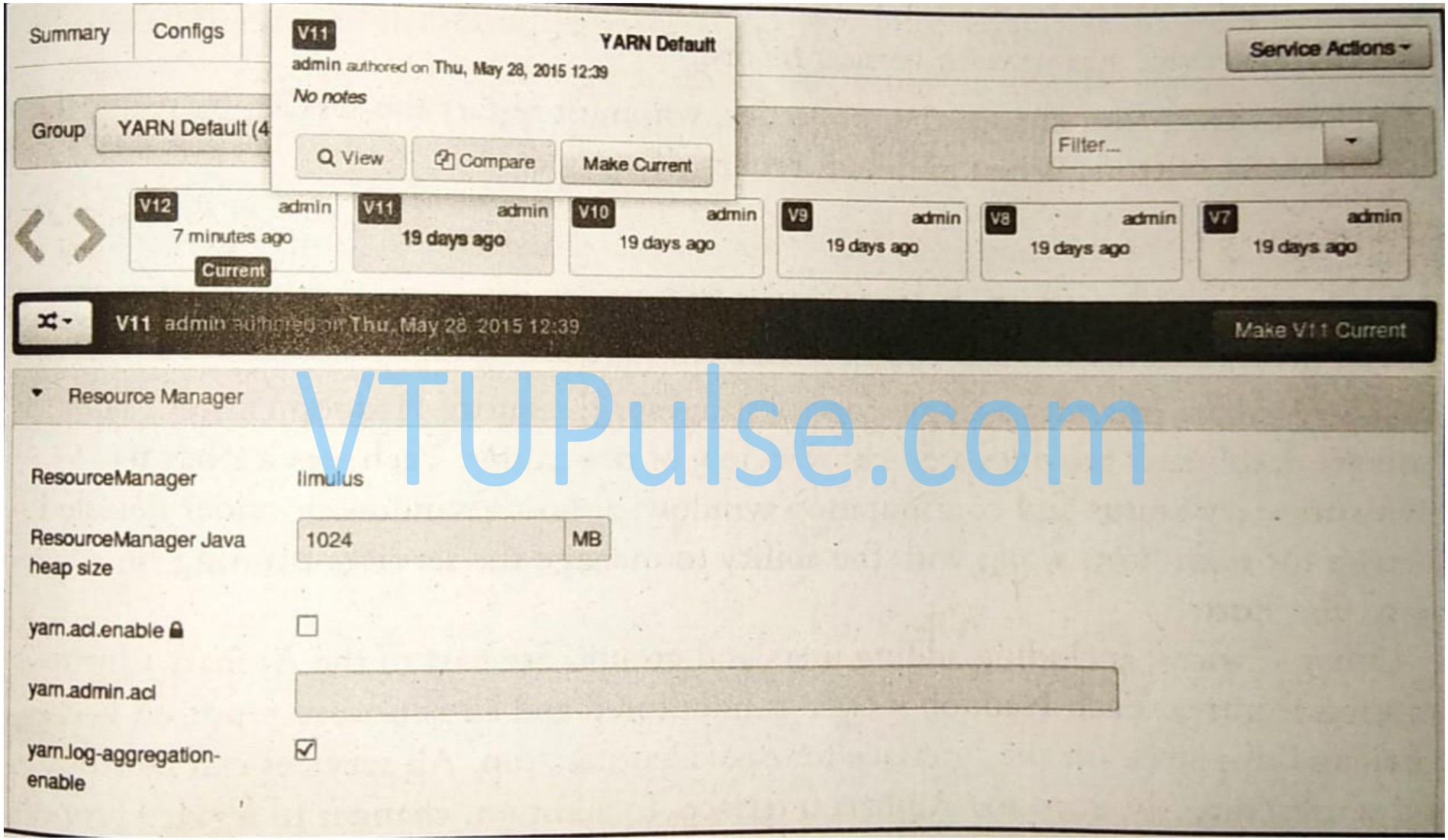


Figure 9.21 Ambari Restart function appears after changes in service properties









Basic Hadoop Administration Procedures

- Hadoop has two main areas of administration: the YARN resource manager and the HDFS file system. Other application frameworks (e.g., the MapReduce framework) and tools have their own management files.
- Hadoop configuration is accomplished through the use of XML configuration files. The basic files and their function are as follows:
  - **core-default.xml:** System-wide properties
  - **hdfs-default.xml:** Hadoop Distributed File System properties
  - **mapred-default.xml:** Properties for the YARN MapReduce framework
  - **yarn-default .xml:** YARN properties

Basic Hadoop YARN Administration

- YARN has several built-in administrative features and commands.
- To find out more about them, examine the YARN commands documentation at

https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YarnCommands.html#Administration_Commands

Basic Hadoop YARN Administration

Decommissioning YARN Nodes

- If a NodeManager host/node needs to be removed from the cluster, it should be decommissioned first.
- Assuming the node is responding, you can easily decommission it from the Ambari web UI.
- Simply go to the Hosts view, click on the host, and select Decommission from the pull-down menu next to the NodeManager component.

Basic Hadoop YARN Administration

YARN WebProxy

- The Web Application Proxy is a separate proxy server in YARN that addresses security issues with the cluster web interface on ApplicationMasters.
- By default, the proxy runs as part of the Resource Manager itself, but it can be configured to run in a stand-alone mode by adding the configuration property yarn.web-proxy.address to yarn-site.xml

Basic Hadoop YARN Administration

Using the JobHistoryServer

- The removal of the JobTracker and migration of MapReduce from a system to an application-level framework necessitated creation of a place to store MapReduce job history.
- The JobHistoryServer provides all YARN MapReduce applications with a central location in which to aggregate completed jobs for historical reference and debugging.
- The settings for the JobHistoryServer can be found in the mapred-site.xml file.

Basic Hadoop YARN Administration

Managing YARN Jobs

- YARN jobs can be managed using the yarn application command. The following options, including -kill, -list, and -status, are available to the administrator with this command.
- MapReduce jobs can also be controlled with the **mapred** job command.
- Usage

application –option [application ID]

Basic Hadoop YARN Administration

Setting Container Memory

- YARN manages application resource containers over the entire cluster. Controlling the amount of container memory takes place through three important values in the yarn-site.xml file:
- **yarn.nodemanager.resource.memory-mb** is the amount of memory the NodeManager can use for containers.
- **Yarn.scheduler. minimum-allocation-mb** is the smallest container allowed by the Resourc Manager. A requested container smaller than this value will result in an allocated container of this size (default 1024MB).
- **yarn. scheduler. maximum-allocation-mb** is the largest container allowed by the Resource Manager (default 8192MB)

Basic Hadoop YARN Administration

Setting Container Cores

- **yarn.scheduler.minimum-allocation-vcores:** The minimum allocation for every container request at the Resource Manager, in terms of virtual CPU cores. Requests smaller than this allocation will not take effect, and the specified value will be allocated the minimum number of cores. The default is 1 core.
- **yarn.scheduler.maximum-allocation-vcores**: The maximum allocation for every container request at the Resource Manager, in terms of virtual CPU cores. Requests larger than this allocation will not take effect, and the number of cores will be capped at this value. The default is 32.
- **yarn.nodemanager.resource.cpu-vcores**: The number of CPU cores that can be allocated for containers. The default is 8.

Basic Hadoop YARN Administration

Setting MapReduce Properties

- As noted throughout this book, MapReduce now runs as a YARN application. Consequently, it may be necessary to adjust some of the mapred-site.xlnl properties as they relate to the map and reduce containers. The following properties are used to set some Java arguments and memory size for both the map and reduce containers:
  - **mapred.child.java.opts** provides a larger or smaller heap size for child J VMS of maps
  - **mapreduce.map.memory.mb** provides a larger or smaller resource limit for maps (default ¯- 1536MB).
  - **mapreduce.reduce.memory.mb** provides a larger heap size for child JVMS of maps (default = 3072MB).
  - **mapreduce.reduce.java.opts** provides a larger or smaller heap size for child reducers.

Basic HDFS Administration

The NameNode User Interface

- Monitoring HDFS can be done in several ways. One of the more convenient ways to get a quick view of HDFS status is through the NameNode user interface.
- This web-based tool provides essential information about HDFS and offers the capability to browse the HDFS namespace and logs.
- URL to open UI: http://10ca1host:50070
- There are five tabs on the UI: Overview, Datanodes, Snapshot, Startup Progress, and Utilities.
- The **Overview** page provides much of the essential information that the command-line tools also offer, but in a much easier-to-read format.
- The **Datanodes** tab displays node information like that shown in Figure.
- The **Snapshot** window lists the "snapshottable" directories and the snapshots.
- Startup **progress** gives you the details process timings
- The **utility** menu offers two options: First, file system browser, where you can easily explore the HDFS namespace and second, Links to various name node logs.

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Overview 'limulus:8020' (active)

Started:	Tue Jun 23 21:36:29 EDT 2015
Version:	2.6.0.2.2.4.2-2, r22a563ebe448969d07902aed869ac13c652b2872
Compiled:	2015-03-31T19:49Z by jenkins from (no branch)
Cluster ID:	CID-b24693ba-9cc7-4750-bbc8-08c4d88f26e7
Block Pool ID:	BP-1208979959-10.0.0.1-1429639633828

Summary

Security is off.

Safemode is off.

351446 files and directories, 1063 blocks = 352509 total filesystem object(s).

Heap Memory used 221.75 MB of 1004 MB Heap Memory. Max Heap Memory is 1004 MB.

Non Heap Memory used 48.97 MB of 134.56 MB Commited Non Heap Memory. Max Non Heap Memory is 304 MB.

Configured Capacity:	1.37 TB
DFS Used:	55.35 GB
Non DFS Used:	89.95 GB
DFS Remaining:	1.23 TB
DFS Used%:	3.95%

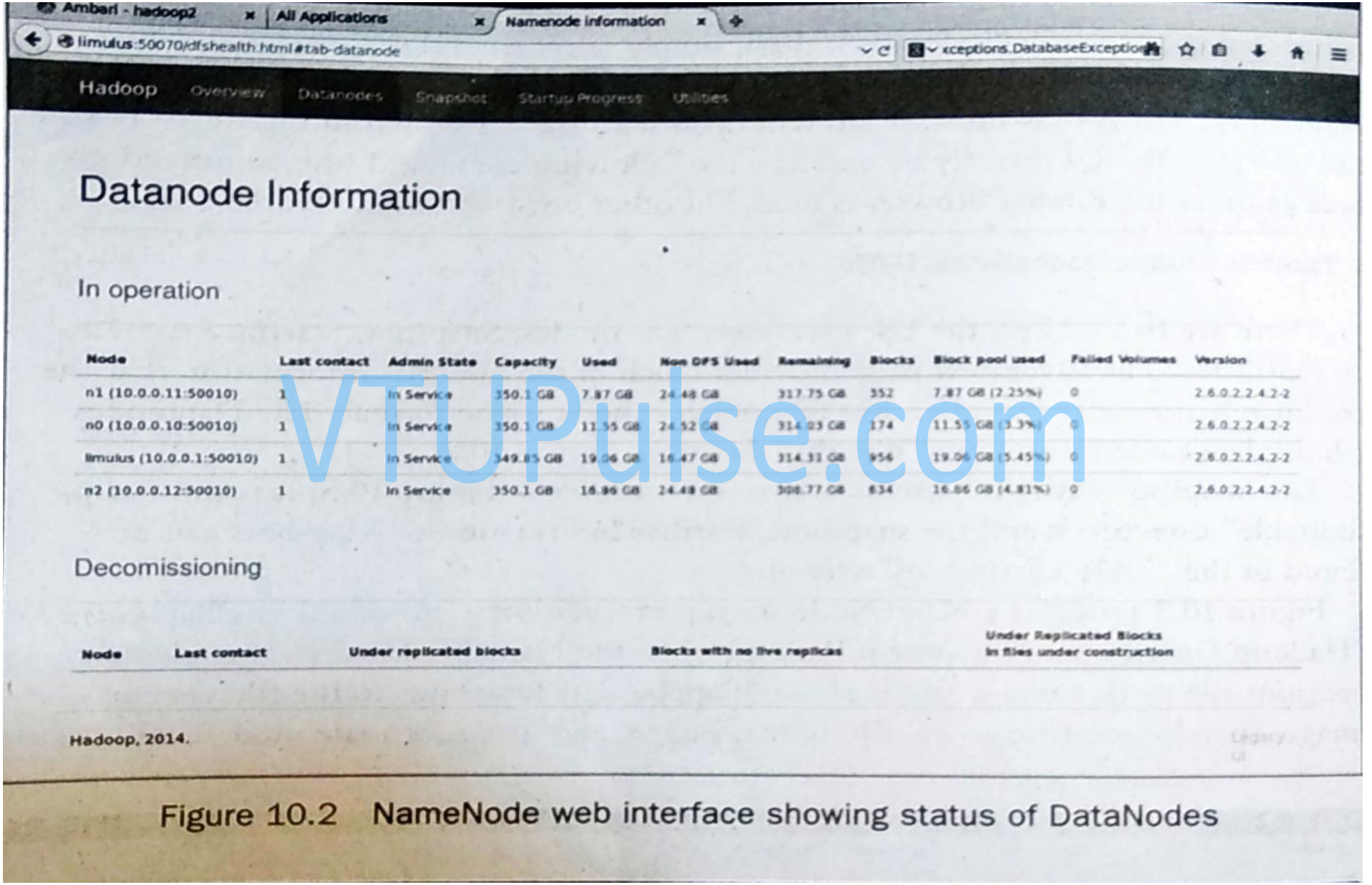


Figure 10.2 NameNode web interface showing status of DataNodes

Startup Progress

Elapsed Time: 2 mins, 6 sec, Percent Complete: 100%

Phase	Completion	Elapsed Time
Loading fsimage /hdfs1/hadoop/hdfs/namenode/current/fsimage_0000000000000999637 26.35 MB	**100%**	**2 sec**
inodes (0/0)	100%	
delegation tokens (0/0)	100%	
cache pools (0/0)	100%	
Loading edits	**100%**	**0 sec**
/hdfs1/hadoop/hdfs/namenode/current/edits_0000000000000999638-0000000000000999793 1 MB (156/156)	100%	
Saving checkpoint	**100%**	**0 sec**
Safe mode	**100%**	**2 mins, 3 se**
awaiting reported blocks (971/971)	100%	

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxrwxrwx	yarn	hadoop	0 B	0	0 B	app-logs
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	apps
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	benchmarks
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	hdp
drwxr-xr-x	mapred	hdfs	0 B	0	0 B	mapred
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	mr-history
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	system
drwxrwxrwx	hdfs	hdfs	0 B	0	0 B	tmp
drwxr-xr-x	hdfs	hdfs	0 B	0	0 B	user
drwx-wx-wx	hdfs	hdfs	0 B	0	0 B	var

Hadoop, 2014.

Basic HDFS Administration

Adding Users to HDFS

- To quickly create user accounts manually on a Linux-based system, perform the following steps:
- Add the user to the group for your operating system on the HDFS client system. In most cases, the groupname should be that of the HDFS superuser, which is often hadoop or hdfs.

  useradd -G <groupname> <username>
- Create the username directory in HDFS.

  hdfs dfs -mkdir / user/<username>
- Give that account ownership over its .directory in HDFS.

  hdfs dfs -chown <usernames : <groupname> /user/<username>

Basic HDFS Administration

Perform an FSCK on HDFS

- To check the health of HDFS, you can issue the hdfs fsck (file system check) command.
- The entire HDFS namespace can be checked, or a subdirectory can be entered as an argument to the command.
- The following example checks the entire HDFS namespace.

  hdfs fsck /
- Other options provide more detail, include snapshots and open files, and management of corrupted files.

  -move moves corrupted files to / lost+found.

  -delete deletes corrupted files.

  -files prints out files being checked.

  -openforwrite prints out files opened for writes during check.

  -list-corruptfileblocks prints out a list of missing blocks

  -blocks prints out a block report.

  -locations prints out locations for every block.

  -racks prints out network topology for data-node locations.

```
.....................................................................
Status: HEALTHY
 Total size:    100433565781 B (Total open files size: 498 B)
 Total dirs:    201331
 Total files:   1003
 Total symlinks:                0 (Files currently being written: 6)
 Total blocks (validated):      1735 (avg. block size 57886781 B) (Total open file
                                blocks (not validated): 6)
 Minimally replicated blocks:           1735 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:                 0 (0.0 %)
 Default replication factor: 2
 Average block replication:     1.7850144
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:                  4
 Number of racks:               1
FSCK ended at Fri May 29 14:48:03 EDT 2015 in 1853 milliseconds
```

Basic HDFS Administration

Balancing HDFS

- Based on usage patterns and DataNode availability, the number of data blocks across the DataNodes may become unbalanced.
- To avoid over-utilized DataNodes, the HDFS balancer tool rebalances data blocks across the available DataNodes.
- Data blocks are moved from over-utilized to under-utilized nodes to within a certain percent threshold.
- Rebalancing can be done when new DataNodes are added or when a DataNode is removed from service.
- This step does not create more space in HDFS, but rather improves efficiency.

hdfs balancer

Basic HDFS Administration

HDFS Safe Mode

- When the NameNode starts, it loads the file system state from the fsimage and then applies the edits log file.
- It then waits for DataNodes to report their blocks. During this time, the NameNode stays in a read-only Safe Mode. The NameNode leaves Safe Mode automatically after the DataNodes have reported that most file system blocks are available.
- The administrator can place HDFS in Safe Mode by giving the following command:

  hdfs dfsadmin -safemode enter
- Entering the following command turns off Safe Mode:

  hdfs dfsadmin -safemode leave
- HDFS may drop into Safe Mode if a major issue arises within the file system. The file system will not leave Safe Mode until the situation is resolved. To check whether HDFS is in Safe Mode, enter the following command:

  hdfs dfsadmin -safemode get

Basic HDFS Administration

Decommissioning HDFS Nodes

- If you need to remove a DataNode host/node from the cluster, you should decommission it first.
- Assuming the node is responding, it can be easily decommissioned from the Ambari web UI.
- Simply go to the Hosts view, click on the host, and selected Decommission from the pull-down menu next to the DataNode component.
- Note that the host may also be acting as a Yarn NodeManager. Use the Ambari Hosts view to decommission the YARN host in a similar fashion.

Basic HDFS Administration

SecondaryNameNode

- To avoid long NameNode restarts and other issues, the performance of the SecondaryNameNode should be verified.
- The SecondaryNameNode takes the previous file system image file (fsimage*) and adds the NameNode file system edits to create a new file system image file for the NameNode to use when it restarts.
- The **hdfs-site.xml** defines a property called **fs.checkpoint.period**.
- This property provides the time in seconds between the SecondaryNameNode checkpoints.

Basic HDFS Administration

HDFS Snapshots

- HDFS snapshots are read-only, point-in-time copies of HDFS. Snapshots can be taken on a subtree of the file system or the entire file system.
- Some common use-cases for snapshots are data backup, protection against user errors, and disaster recovery.
- Snapshots can be taken on any directory once the directory has been set as snapshottable. A snapshottable directory is able to accommodate 65,536 simultaneous snapshots.
- There is no limit on the number of snapshottable directories.
- Administrators may set any directory to be snapshottable, but nested snapshottable directories are not allowed.
- For example, a directory cannot be set to snapshottable if one of its ancestors/ descendants is a snapshottable directory.

Basic HDFS Administration

HDFS Snapshots

- The following example walks through the procedure for creating a snapshot. The first step is to declare a directory as "snapshottable" using the following command:

hdfs dfsadmin -allowSnapshot / user/hdfs/war-and-peace-input

- Allowing snapshot on / user/hdfs/war-and-peace-input succeeded
- Once the directory has been made snapshottable, the snapshot can be taken with the following command. The command requires the directory path and a name for the snapshot—in this case, wapi-snap-l.

hdfs dfs -createSnapshot /user/hdfs/war-and-peace-input wapi-snap-1

- Created snapshot / user/hdfs/war-and-peace-input/ .snapshot/wapi-snap-1