

St. Cloud State University theRepository at St. Cloud State

Culminating Projects in Information Assurance

Department of Information Systems

3-2017

Security Log Analysis Using Hadoop

Harikrishna Annangi

Harikrishna Annangi, hannangi@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation

Annangi, Harikrishna, "Security Log Analysis Using Hadoop" (2017). *Culminating Projects in Information Assurance*. 19.
https://repository.stcloudstate.edu/msia_etds/19

This Starred Paper is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

Security Log Analysis Using Hadoop

by

Harikrishna Annangi

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Information Assurance

April, 2016

Starred Paper Committee:
Dr. Dennis Guster, Chairperson
Dr. Susantha Herath
Dr. Sneha Kalia

Abstract

Hadoop is used as a general-purpose storage and analysis platform for big data by industries. Commercial Hadoop support is available from large enterprises, like EMC, IBM, Microsoft and Oracle and Hadoop companies like Cloudera, Hortonworks, and Map Reduce.

Hadoop is a scheme written in Java that allows distributed processes of large data sets across clusters of computers using programming models. A Hadoop frame work application works in an environment that provides storage and computation across clusters of computers. This is designed to scale up from a single server to thousands of machines with local computation and storage.

Security breaches happen most frequently nowadays which can be found out by monitoring the server logs. This server-log analysis can be done by using Hadoop. This takes the analysis to the next level by improving security forensics which can be done as a low-cost platform.

Acknowledgement

The successful completion of this paper could not have been possible without the guidance of my beloved professors, Dr. Dennis Guster and Dr. Susantha Herath. I also would like to thank Professor Sneh Kalia for being part of the committee and finding the time to read my thesis.

I also would like to thank my mother A Venkata Ramana, my father A Lakshmi Venkata Ramana, and my friends who provided me immense support the entire way.

Table of Contents

		Page
List of Tables		5
Chapter		
I.	Introduction.....	6
	Hadoop.....	6
	Problem Statement	9
II.	Background and Literature Review	10
	Introduction.....	10
	Major Distributions of Hadoop.....	11
	Modules of Hadoop.....	12
	Hadoop I/O.....	19
III.	Methodology	23
	Introduction.....	23
IV.	Implementation	43
	Security Log Analysis Using Hadoop.....	43
V.	Timeline	54
VI.	Conclusion and Future Scope	55
References.....		56

List of Tables

Table	Page
1. Compression Formats	21
2. Dates and Description of the Task	54

Chapter I: Introduction

Hadoop

Hadoop was created by Doug Cutting, the creator of Apache and has its origin in Apache Nutch, which is an open source web search engine. In January 2008, Hadoop was made a top-level project at Apache. In April 2008, Hadoop broke the world record by becoming the fastest system to sort a terabyte of data.

In November 2008, Google announced its MapReduce implementation that sorted 1 terabyte in 68 seconds. In April 2009, a team at Yahoo announced that Hadoop sorted 1 terabyte of data in 62 seconds. In 2014 a team from DataBricks used 207 node spark clusters to sort 100 terabytes of data in 1406 seconds.

Data. In this digital universe, the total volume of data stored electronically is estimated to be 4.4 zettabytes in 2013 and by 2022 it is estimated to be 44 zettabytes. There is a large amount of data and this data effects the smaller and individual organizations. The digital streams produced by the individuals are occupying more space. The volume of data available publicly is increasing day by day and the success of organizations depends upon the ability to access and curate the data so that it can be utilized appropriately when making profitable decisions.

Data storage and analysis. The storage capacity of the hard drives has been increased over time, but the speed of reading the data is still the same. One typical drive from 1990 can store 1,390 MB of data and the speed to read the data is 4.4Mbps (Chansler, Kuang, Radia, Shvachko, & Srinivas, 2011). The other problem is to combine the data from one disk with other multiple sources and this can be done by various distributed systems but this will be a challenging task. MapReduce is a programming model which helps in reading, writing and

analyzing the data. MapReduce is an interface whereby analysis occurs with the combination of HDFS, here Hadoop provides the platform for storage and analysis of the data.

Querying data. MapReduce uses a brute force approach, it is a batch query processor which can run additional queries on the existing data sets which leads to new insights and provides an opportunity to think about the data in a new way and run a query in a timely manner. The analysis of the data capabilities when using different analyzing tools gives the engineers more scope to visualize the patterns and provide better service or support to the customers.

Beyond batch. MapReduce is not suitable for interactive analysis because it is a batch processing system. Hadoop has gone a step beyond batch processing system. Hadoop handles large scale data and many of the data processing units are supported by the Apache software foundation which provides access to the HTTP servers which process the batches individually. HBase is an online key value store that uses HDFS for storage. It provides online access to batch/individual operations which read and write data in bulk. Yarn is a cluster based resource management system in Hadoop which allows any distributed program to run data clusters.

The different processing patterns that work with Hadoop include:

- **SQL.** While working with large data sets dispensing with MapReduce, it is possible to achieve low latency responses in Hadoop.
- **Iterative processing.** Algorithms in machine language are iterative and efficient in nature and it is important to hold each intermediate working set in memory.
- **Stream processing.** Streaming systems like Apache Spark and Apache Storm make it possible to run real time distributed computations on unbounded streams of data and the results are stored in Hadoop.

- **Search.** The Apache Solr search platform can run on Hadoop clusters which help in indexing the documents that are added to HDFS which helps in serving queries from indexes stored in HDFS.

Comparison with other systems. Hadoop has the unique properties which make it different from other distributed and storage analysis data systems that are discussed below.

Relational database management system. It is the software that is used to manage data integrity, storage and access of the data. This software is used to update and work on point queries where small set of data is involved in which data sets have been integrated which has been indexed to deliver low latency and retrieval of small data. RDBMS has a defined format that is confined to a predefined schema. It is often normalized and retains integrity and redundancy.

Grid computing. Large scale data processing is performed by an application program interface which serves as a message passing interface within Grid computing and high performance computing (HPC). The approach used in these computing systems is to employ clusters of the machines and access shared files via a storage area network. This works for computing intensive jobs, but it becomes difficult when nodes need to access larger data volumes and the network bandwidth is a concern.

Volunteer computing. Volunteer computing solves the problem by breaking the work into work units which are then sent to different computers around the world for analysis. SETI@Home Search for Extra-Terrestrial Intelligence is a volunteer project in which CPU time from the ideal computers is used to analyze the radio computing data for signs of intelligence outside the earth. SETI@Home is a very CPU intensive project that is suitable for running on

thousands of computers around the world, here the work is broken into work units and it is done by the volunteers.

Problem Statement

As the IT firms and business administration sector is developing, associations are seeking more security answers to prepare for the constantly enlarging risk scene. Firewalls are in place but, due to constantly emerging new threats, the firewalls are still vulnerable to hostile spyware and web insurance apparatuses that are being taken down. Each one of these security tools will require extensive review of records and cautions. In this manner, numerous organizations and government offices are attempting to set up repeatable log accumulation, centralization and examination mechanisms.

Notwithstanding, when arranging, actualizing log gathering, and investigation frameworks, the associations regularly find that they are not understanding the full guarantee of such a framework. This happens because of the accompanying regular log investigation botches.

We will begin from the self-evident, however a basic mistake—not looking at the logs—is the first oversight. While gathering and putting away logs is imperative, it is an important obligation to view the logs at regular intervals. At that point, one can begin to recognize patterns that could be indicative of a bigger problem and then devise an offensive approach rather than a reactive one. Along these lines, once a new system is set up and activity logs are gathered, there should be a procedure for progressive audit checks that can act as a guide to improve the system where and when necessary. Log examination is vital but more urgent is the need to understand that the proactive approach is imperative in knowing when terrible things could happen or what might turn out to be a potentially devastating vulnerability.

Chapter II: Background and Literature Review

Introduction

Hadoop was brought into light by the creator of Apache Lucene, (a widely-used text library) Doug Cutting. In 2002, a search system emerged called Nutch, but its architecture could not handle the large storage of pages on the web.

In 2003, Google's GFS (Google's distributed file system) came into the picture, which provided the scaling for the storage needs of very large file systems on the web (Ghemawat, Hobbioff, & Leunk, 2003). GFS optimizes CPU usage and reallocates free time to the storage and indexing procedures, in 2004 Google started an implementation file system called Nutch distributed file system (NDFS).

In 2004, a MapReduce paper was published by Google (Dean & Ghemawat, 2004). In 2005, Nutch developers started working with MapReduce and the Nutch algorithm started running through MapReduce.

In 2006, they formed a new subproject called Hadoop and, at the same time, Doug Cutting joined Yahoo as well and a new project started called Hadoop at Yahoo (Zawodny, 2008). In February 2008, Yahoo's production search index was generated by Hadoop clusters (O'Malley, 2008).

Hadoop at Yahoo. The internet scale search engine was built with a huge amount of data and large number of machines to process it. Yahoo search consists of the following:

- Crawler. Downloads pages from webservers
- Web Map. Builds graph of own web

- Indexer. Builds reverse index to the page
- Runtime. Answers users' queries

Web Map consists of 1 trillion web links, 100 billion nodes representing distinct URLs. In 2005, Web Map has been redesigned to Dreadnaughts, which is used to analyze this large amount of graphical data. It is similar to MapReduce in many ways but it is more flexible and less structured.

To replace MapReduce, Eric Baldeschwieler started prototyping and redesigning GFS and MapReduce with C++ modules. But standardizing it with Yahoo search was a critical task and it was aborted. Initially there were only 20 nodes, but Yahoo decided to work with 200 nodes and bring it to the Yahoo search.

By 2008, Hadoop had become a priority with Apache and it was already working with Yahoo, Facebook and many other companies. In April 2008, Hadoop became the fastest system to sort a terabyte of data and broke the world record. It sorted one terabyte in 209 seconds (Google Official Blog, 2008). In November, Google reported that its MapReduce object sorted one terabyte in 68 seconds (O'Malley & Murthy, 2009). In April 2009, Yahoo published that Hadoop sorted one terabyte in 62 seconds (Chansler et al., 2011).

Hadoop is widely used in storage/analysis of big data in industries and mainframe streams and commercial purposes with vendors like ECM, IBM, Microsoft and Oracle.

Major Distributions of Hadoop

There are two major distributions of Hadoop—Cloudera and Horton Works.

Cloudera. Cloudera was founded in 2008 which provides support, services, and training to Apache-Hadoop. More than 50% of the engineering output is dedicated to the various

upstream projects like Apache HIVE, Apache Avro, Apache Hbase which combine to form the Hadoop platform. This offers software services and support in three different bundles:

- Cloudera Enterprise. This includes CDH and subscription licenses to cloud managers and technical support.
- Cloudera Express. This expresses versions of Cloudera manager lacking features such as rolling upgrades/disaster recovery.
- CDH. This may be downloaded through the Cloudera website with no technical support or Cloudera manager.

Horton works. Horton Works deals with the development and support of Apache Hadoop. This was formed in 2011, funded by 23\$ million from Yahoo and Benchmark Capital. Forrester was named Horton Work's echo system leader for Hadoop. The Horton Works data platform is an enterprise data platform that enables a centralized architecture for running batch jobs across shared data sets.

Modules of Hadoop

Hadoop depends upon the following modules: (a) MapReduce, (b) Yarn, (c) Hadoop I/O, and (d) HDFS.

MapReduce. Hadoop MapReduce is used in processing applications which analyze multi tetra data sets in parallel clusters, this programming model for data processing called MapReduce can run in different languages like Ruby, Python, and Java.

MapReduce works in such a way that data is divided into map and reduce, the data that is in the map is parallel processed in the clusters and sorted through a framework and the output of

the map is the input of the reduce (Yahoo Developer Network, n.d.). Map and reduce have the same storage and computing nodes.

The framework of the MapReduce consists of a master job tracker and a slave job tracker per node. The master job tracker schedules the tasks and checks the progress of the tasks and reschedules the failed tasks whereas the slave follows any directions by the master. Applications specify the input/output locations and provide the map used to the reduce tasks. Hadoop then submits the job and the configuration to the job tracker, which then reschedules the tasks, provides the status, monitors the tasks and then sends information to the job client.

Data flow. MapReduce job is a unit of work performed as per the requirements of the client; it is executed using input data, MapReduce and configuration information. The tasks run in Hadoop are of two types: Map tasks and reduce tasks. The scheduling of tasks is taken care by Yarn and they run on nodes. If tasks fail in a particular node, they tend to run on a different node automatically.

Hadoop splits the MapReduce tasks, called an input split, and it runs a thorough search of user records via one split per each record. A good split size is 128 MB which is the size of an HDFS. Hadoop follows the data locality optimization as it does not use valuable bandwidth clusters, this runs the best in map tasks where the input data is present in HDFS. The Map task object sends the output to local disk as the final output is thrown away once the job is done. While the output is consumed by the reduce tasks and, if the map task fails before the output is run, then Hadoop reruns the task automatically to a different node to obtain the output. Reduce tasks does not have a data locality as the output of all the mapping is shared across all tasks. The output of the reduce task is always stored in the HDFS for reliability. These reduce tasks do not

consume the bandwidth. The number of reduce tasks is not related to the size of the input, they are specified independently.

Hadoop streaming. Hadoop uses UNIX streaming which allows one to write the program in any language through which standard input is readable and an output can be written back to MapReduce. Map data is transferred to a map function which processes it line by line and sends it to a standard output. The reduce functions read the input line by line and produces a standard output which is compatible with UNIX standard output.

Yarn. Yet another resource negotiator, Yarn is one of the cluster management systems of Hadoop. It is introduced to support the distributed computing paradigm which also helps in implementations of MapReduce. Yarn provides API's for working with cluster resource, but these API's are not used by user code directly.

The distributed running frame work of Yarn works with applications like Spark, Tez, and MapReduce. Yarn operates on the cluster compute layer and the storage compute layer uses HDFS and HBase. The applications on the future layer that are built on the Yarn framework are PIG, HIVE and Crunch.

Anatomy of Yarn run. Yarn provides two types of core services:

- Resource Manager. This manages resources across clusters.
- Node Manager. This helps in launching and monitoring containers. This is an application specific process with set resources defined according to the Yarn process.

The containers used are the Unix process or Linux group.

Application run process in Yarn—to run an application in Yarn, a client contacts the application manager and initiates the application's master process. The resource manager, with

help from the node manager, launches the application manager in the container. Yarn applications use a form of remote communication to relay the status updates and results back to the client.

Resource request. Yarn is a flexible model when making a resource request; a request has a set of containers which express the amount of computer resources required as well as the locality constraints of the container. Distributed data processing uses clusters for efficient bandwidth, Yarn applications provide locality constraints when requesting containers. The Yarn application has the ability to make resource requests anytime when it is running.

Lifespan of Yarn application. The lifespan of Yarn applications varies, from a short-lived application which runs for few seconds, and a long-lived application which runs for a few hours or sometimes even months. In simple form, the user's job runs one application per job which is the same approach that MapReduce takes. The second form is that it runs one application per workflow or a session of jobs. This is an efficient method because containers can be reused between the jobs and there is no cached intermediate data between the jobs. The third example is a long running model shared by different users which acts in a coordination role. This kind of approach is used by Impala which provides a proxy application that communicates to request cluster resources.

Building Yarn application. Building or writing a Yarn application is difficult, but it is very easy to run with existing applications. The purpose of projects like Apache Slider is simplifying the building of Yarn applications. When using Apache Slider, it is possible to run existing distributed applications through Yarn. Users can run their own instances independently

with other users on the clusters. Different users can run different versions of the same applications.

Apache Twill is similar to Apache Slider and the only difference is that it provides a simple programming model for developing distributed applications on Yarn. This allows cluster processes defined through Java runnable to be run in Yarn containers in clusters. This will also support other applications like real time logging and command messaging.

If there are no running applications, then distributed shell applications can be used to write a Yarn application. This concept helps in understanding the use of API's and the way they communicate with the application master as well as with the Yarn daemons.

Yarn compared to MapReduce. The original version of Hadoop is referred to as MapReduce 1. There are two types of daemons that control the job execution process in Yarn—job tracker and task trackers. The job tracker takes care of the jobs that run on the system and helps in scheduling them to run on the task trackers. Task trackers run the task and send the report to the job tracker; the job tracker takes care of both the task progress monitoring and job scheduling in MapReduce 1. In Yarn, these are split into two separate entities—the resource manager and application master. The job tracker stores the job history of completed jobs in MapReduce and, in Yarn, the timeline server stores the job history.

Yarn is designed to address the limitations of the MapReduce 1, they are as follows: (a) Scalability, (b) Availability, (c) Utilization, and (d) Multi tenancy.

Scalability. MapReduce 1 runs with maximum of 4,000 nodes and 40,000 tasks. The job tracker manages both jobs and tasks. Conversely, Yarn runs on 10,000 nodes and 10,000 tasks with the virtue of a split resource manager and application master. In MapReduce, each

application instance has a dedicated application master which runs during the application; this is similar to Google MapReduce paper.

Availability. High availability is achieved when the events of a service daemon are failing there should be an availability of taking over the tasks by other daemon but, due to constant changes in the job trackers memory, it is difficult to fit the high availability service into the job trackers memory. In Yarn, the overhead of the resource manager and application master make high availability a difficult task. Hadoop 2 has been re-engineered to provide high availability in conjunction with the resource manager and application master.

Utilization. The task tracker in MapReduce is configured with a fixed allocation of slots, which are divided into map slots and reduce slots. Map slots run only with map task and reduce slots run only reduce tasks.

A node manager manages the set of resources in Yarn. In the case of a MapReduce task running with Yarn, it does not have to wait for a reduce task because only map slots are available on the clusters. In Yarn, resources are finely managed, whereby an application can request what it needs rather than an individual slot like in MapReduce.

Multi tenancy. Yarn gives the opportunity to run different versions of MapReduce on the same clusters, which gives Hadoop a platform to run applications other than just MapReduce. This helps in upgrading MapReduce and makes it more manageable.

Scheduling in Yarn. The Yarn scheduler assigns resources to the applications following a defined policy. Assigning resources without any policy is a difficult task and there is no best policy, hence Yarn provides a series of schedulers based on the type of resource choices.

Yarn has the following schedulers: (a) FIFO Scheduler, (b) Capacity Scheduler, (c) Fair Scheduler, and (d) Delay Scheduling.

FIFO scheduler. The FIFO scheduler accepts tasks from the applications in the order that they are submitted, starting from the task assigned first, regardless of the size or priority of the task. The scheduler completes one task at a time and then as resources become available, the next task is completed. It does not work on shared clusters, and is therefore only used to run small tasks so that the queue is handled efficiently.

Capacity scheduler. The capacity scheduler was developed by Yahoo! and is designed to run Hadoop applications in a manner that utilizes the cluster resources in a timely and efficient manner. This ensures that the organizations bear minimum overhead and reap the benefits of economies of scale by sharing the resources.

With the help of the capacity scheduler, organizations each share their cluster and are assigned a fraction of the resources to their queues at all times. The capacity scheduler creates queues for each organization with certain limits to ensure that no one organization can have excess resources and this provides a degree of elasticity, while also ensuring that organizations have guaranteed access to run their tasks at their peak hours. The scheduler also maintains hierarchical queues, which are sub-queues within an organization, and excess resources are only assigned to other queues once the requirements of one organization are fulfilled.

Fair scheduler. The fair scheduler allocates cluster resources to all the running applications in an interspersed manner so that, over time, all the applications receive a fair amount of shared resources, regardless of the size of the task. This is configured by an allocation file named fair scheduler.xml, which is loaded from the class path. If the allocation file is not present, the scheduler creates the queue dynamically when the user submits the first application. The files are named after the users.

Preemption helps the scheduler to kill the jobs in the container that are using more than their fair amount of resources so that the resources can be allocated to the jobs that are pending in the queue. There are two relating preemption settings available:

- For minimum share: If a queue waits as long as its minimum preemption share time out, then the scheduler will preempt the other containers.
- For fair share: If a queue remains below half of the fair share, as long as the fair share preemption time out, then the scheduler will preempt the other containers.

Delay scheduling. Delay scheduling waits for the task to launch with respect to fairness and locality, if the task does not launch locally it is skipped and other tasks in the queue are processed. This delay then causes the task to run locally.

Hadoop I/O

When dealing with multi-terabyte data, data integrity and compression are the techniques that are comparatively easy to understand within the Hadoop structure.

Data integrity. All data processes on the network or the disk will cause some data to be lost and will include a small number of errors, but Hadoop has the capability to handle data errors at a high rate. Hadoop uses a checksum routine to ensure the integrity of the data: if the newly generated checksum is not equal to the original checksum, then the data may be deemed corrupt. Unfortunately, the checksum only serves to indicate that there is data corruption or error but cannot point out exactly what is wrong or fix it.

CRC-32 is a commonly used checksum or error detecting code which computes a 32-bit integer checksum for data of any size. CRC-32 is used by HDFS, which is a more efficient variant.

Local file system. The Hadoop local file system performs client side checksums. When a file is created, it automatically creates a transparent file in the background with the file name.crc, which uses check chunks to check the file. Each chunk can check a segment up to 512 bytes, the chunk of data is divided by the file.byte-per-checksum property, and the chunk is then stored as metadata in a .crc file. The file can be read correctly though the settings of the files might change, and if an error is detected then the local system throws a checksum exception.

Checksum file system. Local file systems use checksum file systems as a security measure to ensure that the data is not corrupt or damaged in any way. In this file system, the underlying file system is called the raw file system, if an error is detected while working on the checksum file system, it will call reportchecksumfailure(). Here the local system moves the effected file to another directory as a file titled as bad_file. It is then the responsibility of an administrator to keep a check on these bad_files and take the necessary action.

Compression. Compression has two major benefits—it creates space for a file and it also increases the speed of data transfer to a disk or drive. The following are the commonly used methods of compression in Hadoop: (a) Deflate, (b) Gzip, (c) Bizip2, (d) Lzo, (e) Lz4, and (f) Snappy.

All these compression methods primarily provide optimization of speed and storage space, and they all have different characteristics and advantages. Gzip is a general compressor used to clear the space and performs faster than bzip2, but the decompression speed of bzip2 is good. Lzo, Lz4 and Snappy can be optimized as required and, hence, are the better tools in comparison to the others.

Codecs. A codec is an algorithm that is used to perform compression and decompression of a data stream to transmit or store it. In Hadoop, these compression and decompression operations run with different codecs and with different compression formats.

Table 1

Compression Formats

Compression Format	Hadoop Compression Codec
DEFLATE	org.apache.hadoop.io.compress.defaultcodec
Gzip	Org.apache.hadoop.io.copress.gzipcodec
Bzip2	org.apache.hadoop.io.compress.bzip2
Lzo	com.hadoop.compression.lzo.lzocodec
Lz4	org.apache.hadoop.io.compress.lz
Snappy	org.apache.hadoop.io.compress.snappy

Serialization. Serialization is turning structured objects into byte streams and deserialization is turning byte streams into structured objects. Serialization is divided in two methods of data processing: intercrossing communication and data storage. Intercrossing communication between nodes is processing that uses remote procedure calls (RPC's). In RPC, the data is converted to the binary system and is then transferred to a remote node where the data is de-serialized into the original message. The lifespan of RPC is less than a second. Properties of RPC serialization format are (a) Compact, (b) Fast, (c) Extensible, and (d) Interoperable.

Hadoop uses its own serialization format called Writable. It is written in Java and is fast as well as compact. The other serialization framework supported by Hadoop is Avro.

File-based data structures. Hadoop has developed several file structures that work well with MapReduce, depending on the application being used. Each structure has its strengths and weaknesses.

SequenceFile. SequenceFile is a commonly used Java based file format in Hadoop for storing and processing smaller files. Since Hadoop is primarily structured to handle large files, SequenceFile helps to mitigate the problem of excessive overhead for processing smaller files such as the excessive use of memory for the NameNode. A sequence file format consists of a header with records. The first three bytes of the sequence are called the magic numbers, followed by a single byte representing the sequence numbers. The header contains information such as the name of the key, value classes, compression details, metadata defined by the user, and a sync marker. There are three formats available for the formatting of records in SequenceFile: uncompressed, compressed, and block compressed.

Mapfile. A Mapfile is a directory, containing a data file, containing all the keys and values in the map, and an index file, containing a fraction of the keys. The Mapfile provides a similar interface as the SequenceFile, but allows random access read/write updating. The map entries must be added in order or they are thrown to an IO exception.

Chapter III: Methodology

Introduction

The Hadoop distributed file system. Hadoop works with HDFS, which is the Hadoop distributed file system. Even though there are many similarities between HDFS and existing distributed file systems, the differences are quite significant. It was initially built to support the Apache Nutch web search engine project but eventually became a Hadoop subproject.

Design of HDFS. HDFS is a file system that is designed for storing large data files with streaming access to data and can be run on commodity hardware (Yahoo Developer Network, n.d.).

- Very Large files: The files that are in megabytes, gigabytes and terabytes in size (Shvacho, 2010).
- Very large files: The files that are in megabytes, gigabytes, and terabytes in size (Shvacho, 2010).
- Streaming data access: The most efficient dataset processing is done by HDFS. A dataset is generated or copied from the source and the analysis of the datasets involves a large amount of data.
- Commodity hardware: Hadoop is designed to run on the clusters of commonly used hardware. The cluster failure rate is high, but it is designed with fault tolerance in such a way that it does not result in any interruption during the failure.

Backdrops of Hadoop. As with all systems, Hadoop has certain areas which require improvement that are discussed below.

- Low latency data access: Applications with low latency in the tens of million seconds' range will not work with HDFS. HBase can be used for low latency access.
- Lots of small files: The NameNode manages the memory in the file system, the limit to number of files allowed in the file system vary by the amount of memory on the NameNode. The storage of millions of files is feasible but billions cannot be stored because the capacity of the hardware does not allow for it (Hadoop, n.d.).
- Multiple writers: HDFS does not allow multiple writes in the modification of datasets.

Concepts of HDFS. The following are the concepts of HDFS: (a) Blocks, (b) Name and data nodes, (c) Block caching, (d) HDFS federation, and (e) HDFS high availability.

Blocks. Every disk has a block size, which is the minimum amount of physical data that it can read and write. File systems blocks are a few kilobytes in size whereas disk blocks are 512 bytes. Tools like DF and FSCK are used for maintenance of file systems operate on the block level. HDFS blocks are large units that are 128MB by default. The HDFS files are blocks that are broken into chunks and are stored as independent units.

There are many benefits of a distributed file system:

- A file can be larger than any single disk in the network, but it can be broken down into blocks and be stored on any disk in the cluster.
- A unit of abstraction is preferred to be stored as a block rather than a file system to simplify the storage subsystem. Simplicity is very important for the distributed systems whose failure modes are varied. The storage subsystem deals with blocks, simplifying storage management and eliminating metadata.

- Blocks go well with replication and provide a certain level of fault tolerance and availability. Each block is divided into smaller machines, if one block is unavailable due to corruption or machine failure, a copy of it can be read from its different locations.
- Some applications set a high replication factor to spread the read load in the cluster.

NameNodes and DataNodes. The HDFS cluster architecture consists of MasterNode, also known as the NameNode, and WorkerNode patterns which are the number of datanodes.

The namenode manages the file system namespace; all the files systems, directories, and metadata which is present in the tree are maintained by the namenode. The information is stored in the local disk in the form of namespace and edit log. The namenode also regulates client access to the files.

The datanode stores HDFS files in its local file system. The datanode does not create all the files in the same directory and it does not have any knowledge about the HDFS files, rather it creates an optimal number of files per directory and creates subdirectories as needed. Each time the datanode starts up, it generates a list of all HDFS data blocks that correspond to each of the files located in the local file system, and generates a Blockreport that it sends to the NameNode.

Hadoop provides two mechanisms for the namenode:

- Backup the file that makes up the filesystem state of metadata. Hadoop can be configured as the namenode gives its state to multiple file systems, the usual choice is to write to the local disk as well as the NFS mount.
- The role of secondary namenode is to prevent the edits log from becoming too large, for this purpose it periodically merges the namespace image and the edits log file.

The secondary namenode is always set up on a separate machine as its memory and system requirements are the same as the primary namenode. The secondary namenode stores the latest checkpoint in a similar directory as the primary namenode and hence acts as a backup copy, if necessary. They run on different machines because there is a need for massive CPU and memory resources when the name node is run. This design also saves a copy of merged data which can be retrieved upon failure.

Block caching. The frequently accessed file blocks are cached in the memory of the datanode in what is called block cache. Job schedulers take advantage of the cached blocks when running the task on datanodes which increases performance.

HDFS federation. A reference of every file and block is kept in the memory of the file system, as there is limited storage of files on the large clusters. HDFS Federation brings up the 2.x series release in the clusters that add namespace in the portion of every namenode which manages the file system.

In the federation, each namenode manages the namespace volumes and the back pool contains all the blocks for files in the namespace. The namespace volumes do not communicate with each other as they are independent and if there is a failure of one namenode it does not affect the other namenodes in the cluster, as the block pool is not partitioned the data nodes on the cluster store blocks from multiple block pools.

HDFS high availability. Creating a secondary namenode and replicating metadata on multiple file systems protects from data loss but does not provide high availability of the file systems. The namenode is a single point of contact. If a namenode did not fail, all clients, such as

MapReduce, will not be able to read, write or list as the namenode is the depository of metadata and the file to block mapping. In this case, Hadoop is taken out of the scenario until a new service comes up online.

To retrieve data from a failed namenode, the administrator starts a new namenode, with replicas from the metadata, and configures the datanodes and clients to use this network. To work on the request, the new namenode should have the following properties:

- Load the namespace image into the memory.
- Reconfigure edit logs.
- Get blockreports from the datanodes to leave safe mode.

It takes 30 minutes or more for a namenode to start working on the large clusters with many blocks and files. Unexpected failure of the namenodes is very rare and it is always important for planned down time. In Hadoop 2, HDFS High Availability support is present which prevents such situations. These helps in the continuous process of the client request without any interruption. The changes while working on namenodes:

- To share the edit logs, namenodes must have high available shared storage. When the system is in standby with the standby namenodes, it reads the edit logs to synchronize with an active name node.
- Block mapping is stored in the memory of the namenode and datanodes send the blockreports to namenodes for block data mapping.
- By using transparent machines, the clients can configure the failure structure of the namenodes.

- The rate of secondary namenode polling which defines the interval of namenode periodic check points.

There are two highly available shared storages, a NFS filer, and Quorum journal manager (QJM). This is a recommended for HDFS installations and it is designed for highly available edit logs. Each edit is run against the majority of journal nodes. The arrangement of journal nodes is similar to a zookeeper's arrangement; there are three journal nodes available to make the system run if one is lost.

If the active name node fails, the system takes about a minute to recognize the failure, and then it gets the data from the memory because the memory contains the active namenode with edit log entries and up to date data mapping. If the active instance fails and the standby is down, we can still run the standby through the administrator in cold fashion.

Failover and fencing. The transitions of the active node and the standby node are maintained by the failover controller. There are many failover controllers but the default one runs in conjunction with the zookeeper to make sure that one namenode is active and running. The QJM allows one namenode to write edit logs at a time and the previous active nodes are stale but can serve the requests of the clients. The NFS filer for shared edit logs is required, but it writes to only one namenode at a time. Fencing mechanisms include revoking namenode access from a shared directory and remotely disabling the associated network port. By a technique called STONITH, the previously active name node can be fenced.

Hadoop file systems. HDFS is one of the file systems within Hadoop. There are many other file systems in Hadoop. The java implementation `org.apache.hadoop.fs.FileSystem` is the interface to the file systems in Hadoop.

The following are the file systems within Hadoop:

- Local file system. This is the java implementation of `fs.localfilesystems`. The locally connected file systems use checksum for parity while `rawlocalfilesystem` does not.
- HDFS File system. The java implementation of HDFS is `Hdfs.distributedfilesystem`. It is designed to work with conjunction with Mapreduce.
- WebHDFS File system—`hdfs.web.webhdfsfilesystem`. This provides the authentication for reading and writing with http.
- Secure webHdfs file system—`hdfs.web.swebhdfsfilesystem`. This provides access to the secured web hdfs which is Https.
- HAR file system—`Fs.Harfilesystem`. This is a file system which is layered with another file system for creating files. Hadoop archive puts together a large number files into a single file and stores it to reduce the memory usage of the namenode and creates new Har files.
- View files system—`viewfs.viewfilesystem`. This file system creates mount points to the namenodes.
- FTP—`fs.ftp.FTP filesystem`. This is backed by the FTP file system.
- S3—`fs.s3a.Safilesystem`. This file system is backed by the amazon s3 file system.
- Azure—`fs.azure.nativeazurefilesystem`. This file system is backed by Microsoft Azure.

Hadoop provides different interfaces to its filesystem and uses a URI scheme to communicate with file system instances. It is possible to access the data of MapReduce programs

using file systems that use large data. In such cases, a distributed file system is chosen which has data optimization instantiated locally.

Interfaces. Hadoop is written in Java, and most of Hadoop interactions are through the Java API. The following are the interactions that work with HDFS; many of them will also work with the Hadoop file system.

- HTTP: The non-java file systems get exposed to the HDFS file system by this web interface. Because it runs with non-java file systems, it makes it easier for other language file systems to interact with HDFS. This HTTP system is slower when compared to Java interactions so large data file transfers cannot be run in this interaction. There are two ways to use HDFS over HTTP, they are Direct access and Proxy Access.
 - Direct access: The web serves in datanode and namenode act as WebHDFS endpoints. The file metadata operations are sent first to the namenode, redirected to the HTTP and then to the client.
 - Proxy access: This depends on one or more proxy servers. To hide the namenode and the datanode from the client, the traffic in the cluster is generated through proxy. This helps to manage the proxy and the default setting for the firewall. The Httpfs proxy like HTTP used the same interface as webHDFS.
- C: Hadoop has a C library, called the libhdfs, which is similar to the Java filesystem interface. This works with the Java native interface to call javafilesystem client. The libwebhdfs library uses the webhdfs interface. The C hdfs library is similar to Java, but it does not have the updated versions of Java. This just has a header file hdfs.h

which includes apache the Hadoop toolbar. The apache Hadoop system is a prebuilt 64 bit binary linux platform. To add additional features to these platforms, the building.txt instructions need to be added.

- NFS: By using the HDFS Nv3, a gateway is possible which can mount the HDFS on a local client system. Unix utilities are used to interact with filesystem, upload file systems and use proxy libraries to access file system from any of the libraries from the other programming languages.
- FUSE: These file systems are integrated with the Unix file system, which are implemented in the users' interface. Fuse-DFS is a file system that is implemented in C, using libhdfs as an interface for HDFS. HDFS NFS is robust means for mounting HDFS which is preferred over Fuse-DFS.

Flume. Apache Flume is used to aggregate, store, and analyze data using Hadoop.

Hadoop is used to process very large data however a lot of systems only stream data. Flume is used to collect log files from web servers and move the log events to the HDFS files for processing. It is designed to manage large volumes of event based data.

The building block of Flume is a source-channel-sink combination. The Flume agent runs a long-lived Java process to run the sources and sinks which are connected by channels. The installation of Flume is a collection of agents running in a distributed topology. Agents on the edge of the system collect the data and forward that to agents who are responsible for aggregating and storing of data. Flume usually writes the data to HDFS, but also supports Solr and HBase.

Transactions and reliability. A transaction is used for delivery of events from the source event to the sink; Flume uses different transactions to deliver from source to channel and channel to sink. In case if the transaction is not logged due to a failure or an unusual condition, it rolls back the transaction and the event is stored in the channel for later delivery.

A file channel is used, which is durable and, once the event is logged, it will not be lost and remains in the channels, though it is restarted by the agent. Every event produced by the source will reach the sink, and there are chances of forming duplicates; these duplicates are produced either in source or in sink. These duplicates are removed by semantics. Semantics require a two-way commit protocol which is expensive, this differentiates Flume as a high volume parallel ingest system from traditional enterprise systems.

Batching. Flume tries processing the events in batches for high efficiency. Batching helps the performance of the file channel, where every transaction is written on disk and synchronized via fsync. The batch size depends on the query and its components which can be configurable in many cases.

Partitioning and interceptors. Flume data is to be partitioned by time. If a large data set is being processed in Flume, it is always the case that the transformers complete events within complete partitions. The large data sets are always partitioned because the processing becomes much easier and there are less chances of duplication. If duplicates occur, they are easy to remove.

File formats. Binary format is always preferred to run the file systems, as the resulting files are small and the resulting performance will be better when compared to other events. The

HDFS sink file format is controlled using `hdfs.filetype` with a combination of many other properties.

Fan-out It is the term used to describe delivery events from one source to different channels which are then transferred to multiple sinks.

Delivery guarantee. Flume uses different directories for different transactions which are transferred from a spooling directory to each channel. If a transaction is happening, one transaction is needed to channel feed the HDFS sink and other batch feed is used to deliver the batch of events to the channel logger sink. This uses transactions to ensure that batch events are reliable and delivered from one source to the channel as well as from the channel to the sink.

Integrating flume with applications. This is a module that facilitates the java rpc Client in transferring events to an Avro endpoint. Clients can be configured through failover or load balancing between endpoints. The Flume agents offer similar endpoints and it runs in the Java application. It has a single special source that runs the application and sends the calling events to Flume which is a method within an embedded agent object.

PIG. PIG helps in abstraction of processing large data; the data structures are multivalued and structured. This makes data transformations more powerful. It is made up of two components: (a) language used to express data flows, Pig Latin, and (b) the environment to run the Pig Latin program.

There are two Pig Latin environments: (a) local execution in a single JVM and (B) distributed execution in a Hadoop cluster.

Pig Latin is a program made up of a series of operations which are applied to the input data to produce output. Pig turns the transformation into series of MapReduce jobs. This is a

scripting language for exploring large data sets. Pig can process terabytes of data by issuing half dozen lines of Pig Latin from the console. This provides several commands to the programmer for integrating the data structure into a written program. It is extensible and all parts of the program are customizable like storage, loading, filtering, grouping, and joining which are designed to be user defined functions.

Installing and running Pig. Pig is a client side application; it can be run on Hadoop clusters without any extra installations, Pig launches jobs and interacts with the file systems from one's work station. It has straight forward installation.

Execution types. Pig has two modes of executions—local mode and MapReduce mode.

- **Local Mode.** In this mode, Pig runs in single jvm to access the local file system and it is suitable for small data bases.
- **MapReduce Mode.** In this mode, Pig transfers jobs to MapReduce and runs on the Hadoop clusters. The cluster can be either a pseudo or a fully distributed cluster. This mode is suitable to run large data sets.

Pig honors the Hadoop home environment for finding which Hadoop client has to be run, if this is not set, Pig uses a bundle copy of Hadoop libraries. Pig has to be pointed at the namenode and the resource manager in the cluster. These properties can be saved in the pig properties files in the pig confg directory.

Running Pig program. There are three ways of running Pig program in both MapReduce and the local environment—Script, Grunt, and Embedded.

- **Script.** Pig runs a script file that contains Pig commands.

- Example: The `pigscript.pig` run command in the local file `script.pig` can be used for short scripts. `-e` is used to indicate that the script specified string on the command line should be run.
- Grunt. This is an interactive shell for running pig commands. Grunt is started when no file is specified to run the `-e` option. By running `run` and `exec` files one can run pig scripts in Grunt. Like GNU `rule` grunt has line editing facilities. History can be recalled in Grunt using commands like `Cl+P` and `Cl+N`. Incomplete Pig Latin keywords and functions are completed when the tab key is pressed. The customization tokens can be created by file name auto complete.
- Embedded. `Pigserver` class is used to run pig programs in java while `JDBC` is used to run sql programs from java.

Pig Latin editor: The Hadoop distribution comes with the Hue web interface which includes the Pig script editor and launcher, the Pig Latin syntax highlighters such as ECLIPS, intellij Idea, vim, emacs, and tectmate.

Pig Latin: This program explains the syntax and semantics of the programming language. This is collection of statements that can be thought of as an operation or a command. The statements have to be terminated with semicolon and can be split across multiple lines. Pig Latin has two types of commands: double hype and single hype. Everything from the first hype to the end of line can be ignored by Pig Latin interpreter.

Pig Latin has a set of keywords that cannot be used as identifiers, these include the operator (`LOAD`, `ILLUSTRATOR`), commands (`Cat`,`ls`) expressions (`matches`, `flatten`) and functions (`diff`,

max). In this the operator and the commands are not case sensitive but the aliases and functions are case sensitive.

Functions: Pig has four types of functions—Eval function, Filter function, Load function, and Store function.

- **Eval Function.** This function takes one expression and returns another expression. Max is an example of a built in algebraic eval function. It is an example of an aggregate function that operates on data to produce a scalar value. Most aggregate functions are algebraic, in MapReduce algebraic functions work with a combiner and are calculated incrementally.
- **Filter Function.** Filter function is an eval function that gives a Boolean result which can be used to FILTER and remove unwanted rows. This works in relational operations where a Boolean function is used.
- **Load Function.** The Load function helps in understanding how to load data from external resources.
- **Store function.** The Store function states how to save the data of a relational function in external storage.

Pig in practice. There are certain techniques one should follow while working with the Pig program.

- **Parallelism.** While working with MapReduce, the degree of parallelism should match the size of data sets. Pig runs by introducing reducers for 1 GB of data blocks. In this case, the operators that run in reduced mode would need to use the parallel clause.
- **Annoys relations.** For the most defined relationships, diagnostic operators like DUMP can be applied. Pig has access to a previous relation and it matches a name to each

one of these relations. It is recommended that the syntax used should not contain aliases.

- **Parameter Substitution.** Pig supports parameter substitution where the script is substituted by using values supplied at runtime. These parameters are preceded with the \$ character.
- **Dynamic parameters.** These are the parameters that are operated using the param option. It is easy to make the value dynamic by running a command script.
- **Parameter substitution processing.** This is a preprocessing step that occurs before the script has been run. The preprocessors are run on Pig by -dryrun. During the -dryrun mode Pig performs the parameter substitution and generates an original script with a parameter value. Before running it in normal mode, one can check the script and the substitution values.

HIVE. Hive was built by Jeff Hammerbacher and is defined as the information platform which ingests and processes information throughout an organization (Segaran & Hammerbacher, 2009). Hive is a data warehousing infrastructure built on top of Hadoop. Hive was developed by Facebook, which needed to process a large amount of data that accumulated over time as its user base expanded. After trying many systems, Hadoop was chosen by the team because it is cost effective and met Facebook's scalability needs (White, 2015).

Hive was created to analyze a large amount of data using the SQL (HiveQL) skill set this also helps in running the queries on Facebook which are stored in HDFS.

Installing HIVE. Hive can be run on any work station and converts a SQL query into series of jobs which are run on Hadoop clusters. This runs on tables of data which creates the

data structure that is stored in HDFS. Metadata is stored in database called Metastore. To install Hive, the same version of Hadoop is required as in the cluster version.

The Hive shell. This is the primary way to interact with Hive by using commands in HiveQL. HiveQL is a query language which is similar to MySQL, and the commands are case sensitive.

Configuring Hive. Hive is configured as an XML configuration file, called hive-site.xml, and it is similar to Hadoop. In this file, one can set up all the properties that must be implemented while running Hive. The file also contains Hive-default.xml, this documents Hive properties and defaults.

The hive-site.xml is the default place set up a cluster connection, by using Hadoop properties one can specify the file system and the resources. Hive permits one to set up properties which are based on per-session by using the hiveconfig command. The Hive settings can be changed per query within the session using the SET command.

There is a hierarchy to set up all the properties from lower to higher priority:

- Hive Set command
- The command line -hiveconf option
- Hive-site.xml files and Hadoop site files
- The default files of Hive and Hadoop

Hive services. Hive shell is one of the services that can run within the Hive command structure. Hive has the following useful list of services available:

CLI is a default service and it is a command line interface to Hive

Hiveserver2 runs Hive as a server, exposing thrift services and it exposes access to clients in different languages; this improves the original Hive server by supporting multiple user concurrencies. Applications using connectors like Thrift, JDBC, and ODBC run on the Hive server to communicate with Hive.

Beeline is the hive component which works in embedded mode with a command line interface.

Hwi is a simple interface used as an alternate to CLI without installing any client software's.

Jar is convenient way to run java applications that include both Hadoop and hive applications that run with java applications.

Metastore runs on the same process as hive services, but metastore uses a standalone process and sets up specific ports.

Hive clients:

If hive is run as a server, there are many mechanisms an application can use to obtain connectivity: Thrift client, JDBC Drivers, and ODBC drivers.

- **Thrift Client:** Thrift client is a Hive server that is advertised as a thrift server which can be contacted through any language.
- **JDBC Drivers:** Hive provides type 4 JDBC drivers, which run with java applications connected to the Hive server running on separate host and ports. An alternate method to connect to Hive via JDBC is an embedded mode. In this case, Hive runs jvm as an application and it does not use thrift services.

- ODBC Drivers: ODBC allows applications to support ODBC portal. This does not ship with the ODBC servers; several vendors make this driver freely available.

Hive SQL:

Hive SQL is combination of SQL-62 and MySQL. This allowed certain features from SQL standards and the nonstandard extensions to SQL to be adopted by MapReduce.

Data Types:

Hive supports the following data types:

- Primitive data types. This includes numeric, Booleans, strings, and time stamp types.
- Complex data types. This include arrays, maps and data structures.

Hive has three data types for text storage, these come under the primitive data types:

- String. It is a variable length character with no maximum length.
- Varchar. They are declared with maximum number from 1 to 65355.
- Char. These are fixed length strings with trailing spaces if required.

In the complex data types, there is an attribute level of nesting, which uses angle braced notion for complex data types which specifies the type of fields in the collection.

User defined functions: Hive produces built-in functions that cannot be expressed easily. Hive plugs in its own processing code and invokes it from Hive query. There are three types of UDFs (User Defined Functions) in Hive—Regular UDFs, User defined aggregate function, and User defined table generating function.

- Regular UDF. Most of these functions are mathematical and string functions. This type generates a single row as its output.

- User defined aggregate function. This works on multiple input rows and creates a single output row.
- User defined table generating function. This operates on a single row and produces multiple rows.

Apache Oozie. This is a system for running workflows on dependent jobs. There are two main parts—Workflow engine, and Coordinate engine.

- Workflow Engine: Workflow engines store and run workflows which contain different types of Hadoop jobs.
- Coordinate engine: Coordinate engines run workflows based on the schedule and availability of data.

In a Hadoop cluster, Oozie is designed to scale and manage thousands of executions, this makes rerunning failed work flows more traceable. Oozie runs as a service of clusters and submits work flow definitions for immediate or delayed executions. Action nodes perform work flow tasks, such as moving a file to HDFS, running MapReduce, streaming Pig, and running arbitrary shell scripts or java programs. When the work flow is completed, it calls back to HTTP to inform upon the status of the task.

Defining a Workflow Oozie. Workflow Oozie is written in XML, using the Hadoop process definition language. The work flow nodes consist of three control flow nodes, an action flow node, a kill control node, and an end control node. The first two elements, like job tracker and namenode, are specified to use a Yarn specified resource manager.

Packing and deploying an Oozie in workflow application. Workflow applications depend upon the workflow definition and all the associated resources. Applications adhere to

simple directory structure and are deployed to HDFS to get access via Oozie. For this application, all the directories are kept in a base called max-temp-workflow.

Chapter IV: Implementation

Security Log Analysis Using Hadoop

- Download and extract the server log files.
- Install, configure, and start Flume.
- Generate the server log data.
- Import the server log data into Microsoft Excel.
- Visualize the server log data using Microsoft Excel Power View.

Download and extract the server log files. Make a zip of the server log files.

You can save and explore the `ServerLogFiles.zip` archive on your computer.

Login in to the Hadoop virtual machine using the following credentials:

```
login: root
Password: hadoop
```

If you are trying to connect using SSH you can use the following command:

```
ssh root@localhost -p 2222
```

After login, you can see the following command prompt:

```
[root@Sandbox ~]\#:
```

Now you can unzip the serverlogfiles.

```
unzip serverfiles.zip
```

Lastly, we need to create the following directories for Flume:

```
mkdir /var/flume/
mkdir /var/flume/checkpoint/
mkdir /var/flume/data/
chmod 777 -R /var/flume
```

The mkdir command is used to create new directories.

Install, configure, and start Flume. First, we are going to make a backup of the existing flume configuration.

```
cd /etc/flume/conf
mv flume.conf flume.conf.bak
```

The MV command is used to rename a file in UNIX.

Next, we are going to setup a flume agent with new configurations. Copy the following agent config into new file called flume.conf under /etc/flume/conf. Run vi flume.conf, then hit the **I** key to enter insert mode in vi. Then copy the following config. Vi is the editor that is widely used in Unix operating systems.

```
# Flume agent config
sandbox.sources = eventlog
sandbox.channels = file_channel
sandbox.sinks = sink_to_hdfs

# Define / Configure source
sandbox.sources.eventlog.type = exec
sandbox.sources.eventlog.command = tail -F /var/log/eventlog-demo.log
sandbox.sources.eventlog.restart = true
sandbox.sources.eventlog.batchSize = 1000
#sandbox.sources.eventlog.type = seq

# HDFS sinks
sandbox.sinks.sink_to_hdfs.type = hdfs
sandbox.sinks.sink_to_hdfs.hdfs.fileType = DataStream
sandbox.sinks.sink_to_hdfs.hdfs.path = /flume/events
sandbox.sinks.sink_to_hdfs.hdfs.filePrefix = eventlog
sandbox.sinks.sink_to_hdfs.hdfs.fileSuffix = .log
sandbox.sinks.sink_to_hdfs.hdfs.batchSize = 1000

# Use a channel which buffers events in memory
sandbox.channels.file_channel.type = file
sandbox.channels.file_channel.checkpointDir = /var/flume/checkpoint
sandbox.channels.file_channel.dataDirs = /var/flume/data

# Bind the source and sink to the channel
sandbox.sources.eventlog.channels = file_channel
sandbox.sinks.sink_to_hdfs.channel = file_channel
```

You will see the following screen:

```

1. root@sandbox:/usr/hdp/current/flume-server/conf (ssh)
# Flume agent config
sandbox.sources = eventlog
sandbox.channels = file_channel
sandbox.sinks = sink_to_hdfs

# Define / Configure source
sandbox.sources.eventlog.type = exec
sandbox.sources.eventlog.command = tail -F /var/log/eventlog-demo.log
sandbox.sources.eventlog.restart = true
sandbox.sources.eventlog.batchSize = 1000
#sandbox.sources.eventlog.type = seq

# HDFS sinks
sandbox.sinks.sink_to_hdfs.type = hdfs
sandbox.sinks.sink_to_hdfs.hdfs.fileType = DataStream
sandbox.sinks.sink_to_hdfs.hdfs.path = /flume/events
sandbox.sinks.sink_to_hdfs.hdfs.filePrefix = eventlog
sandbox.sinks.sink_to_hdfs.hdfs.fileSuffix = .log
sandbox.sinks.sink_to_hdfs.hdfs.batchSize = 1000

# Use a channel which buffers events in memory
sandbox.channels.file_channel.type = file
sandbox.channels.file_channel.checkpointDir = /var/flume/checkpoint
sandbox.channels.file_channel.dataDirs = /var/flume/data

# Bind the source and sink to the channel
sandbox.sources.eventlog.channels = file_channel

```

Then exit insert mode by hitting **Esc**. Finally, exit `vi` by typing `:wq`

Next, we are going to need to edit a flume configuration file.

```
vi /etc/flume/conf/log4j.properties`
```

```
[root@sandbox ~]# vi /etc/flume/conf/log4j.properties
```

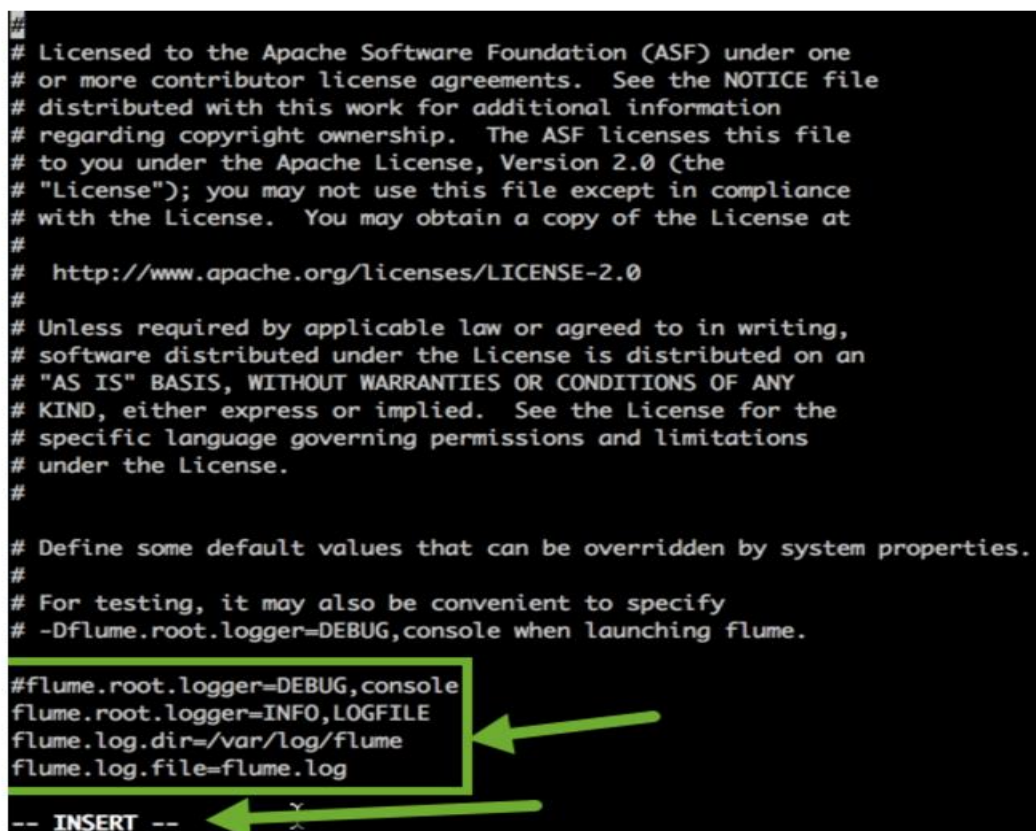
Press the “i” key to switch to insert mode. “–INSERT–” will appear at the bottom of the command prompt window. Use the down-arrow keys to scroll down to see the following code in the log4j properties.

```
flume.root.logger=INFO,LOGFILE
flume.log.dir=./logs
flume.log.file=flume.log
```

Use the arrow keys to position the cursor at the end of the second line. Use the Backspace (Windows) key to delete “./logs”, then type in “/var/log/flume”.

```
flume.root.logger=INFO,LOGFILE
flume.log.dir=/var/log/flume
flume.log.file=flume.log
```

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements. See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership. The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the license.
#
# Define some default values that can be overridden by system properties.
#
# For testing, it may also be convenient to specify
# -Dflume.root.logger=DEBUG,console when launching flume.
#flume.root.logger=DEBUG,console
flume.root.logger=INFO,LOGFILE
flume.log.dir=/var/log/flume
flume.log.file=flume.log
-- INSERT --
```



Press ESC key and then it will move from INSERT mode, then enter the following command at the end of the file.

```
:wq
```

```

#flume.root.logger=DEBUG,console
flume.root.logger=INFO,LOGFILE
flume.log.dir=/var/log/flume
flume.log.file=flume.log
:wc ←

```

This command saves the file and exits the “Vi” editor.

Restart flume. While still at your terminal, we will need to restart flume. Use the following command to restart or start the flume:

```
flume-ng -n sandbox --conf /etc/flume/conf -f /etc/flume/conf/flume.conf
```

Click on flume service go to service actions and select start if it is not already started, then restart it.

Generate the server log data. Because flume is running, we will use a Python script to produce the server log data. Then create the table HCatalog from the data.

```
python generate_logs.py
```

When the log file has generated, a timestamp will show up and the command prompt will appear as normal ([root@Sandbox ~]#). It may take several seconds to see the log file.

```

root@sandbox:~ (ssh)  bash  1. root@sandbox:~ (ssh)
[root@sandbox ~]# vi /etc/flume/conf/log4j.properties
[root@sandbox ~]# ls
anaconda-ks.cfg  build.out  install.log.syslog  serverfiles.zip  start_ambari.sh  start_solr.sh
blueprint.json  install.log  sandbox.info  ServerLogFiles  start_hbase.sh  stop_solr.sh
[root@sandbox ~]# clear
[root@sandbox ~]# python ServerLogFiles/generate_logs.py
2016-02-12T18:29:11
[root@sandbox ~]#

```


Next, we will create a Hive table using the log file. Open the Ambari UI and go to the views dropdown. Select **Hive** and then execute the following query.

```
CREATE TABLE FIREWALL_LOGS(time STRING, ip STRING, country STRING, status STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LOCATION '/flume/events';
```

If the query did not execute properly, it may be due to the permissions setting, in which case we execute the following commands:

```
sudo -u hdfs hadoop fs -chmod -R 777 /flume
sudo -u hdfs hadoop fs -chown -R admin /flume
```

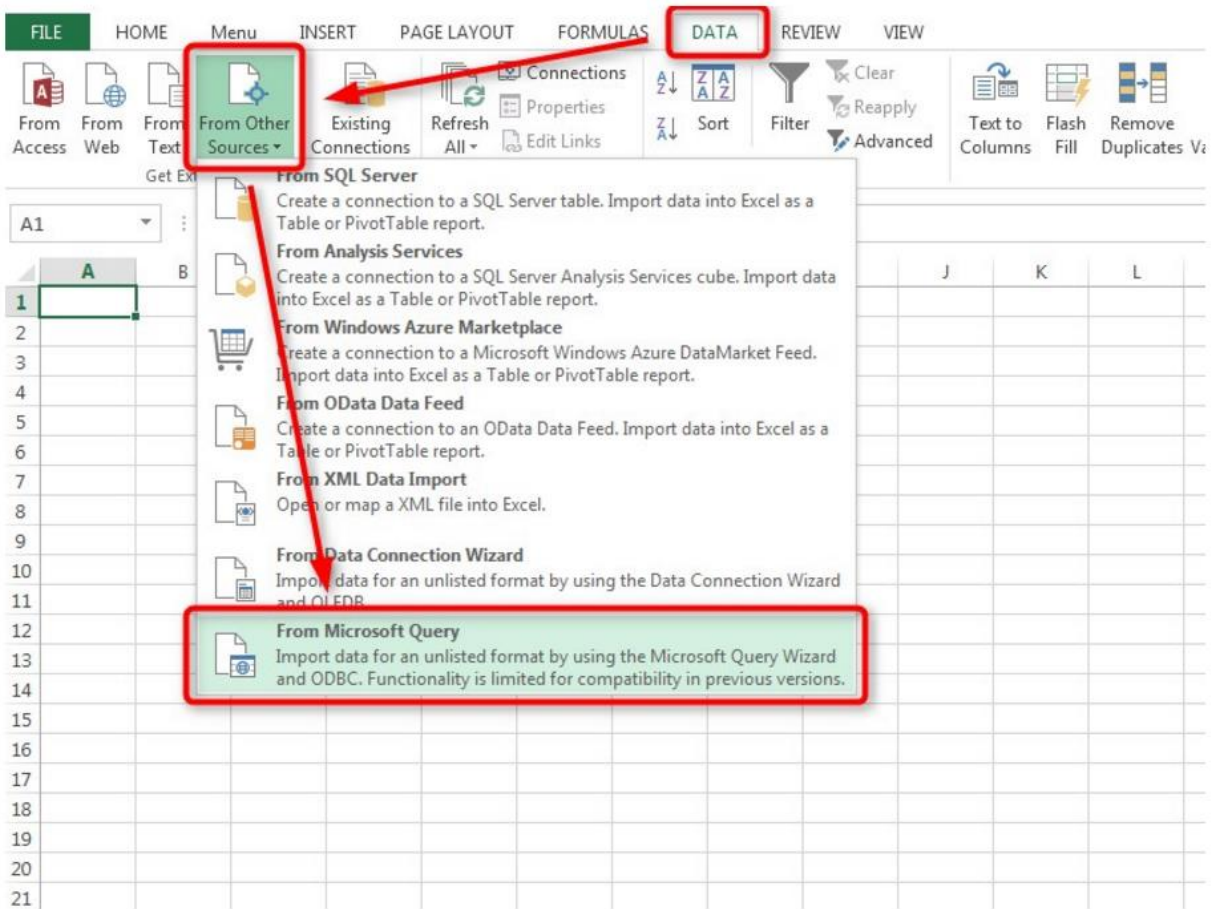
When the table has been created, the data can be seen using the following query.

```
Select * from FIREWALL_LOGS LIMIT 100;
```

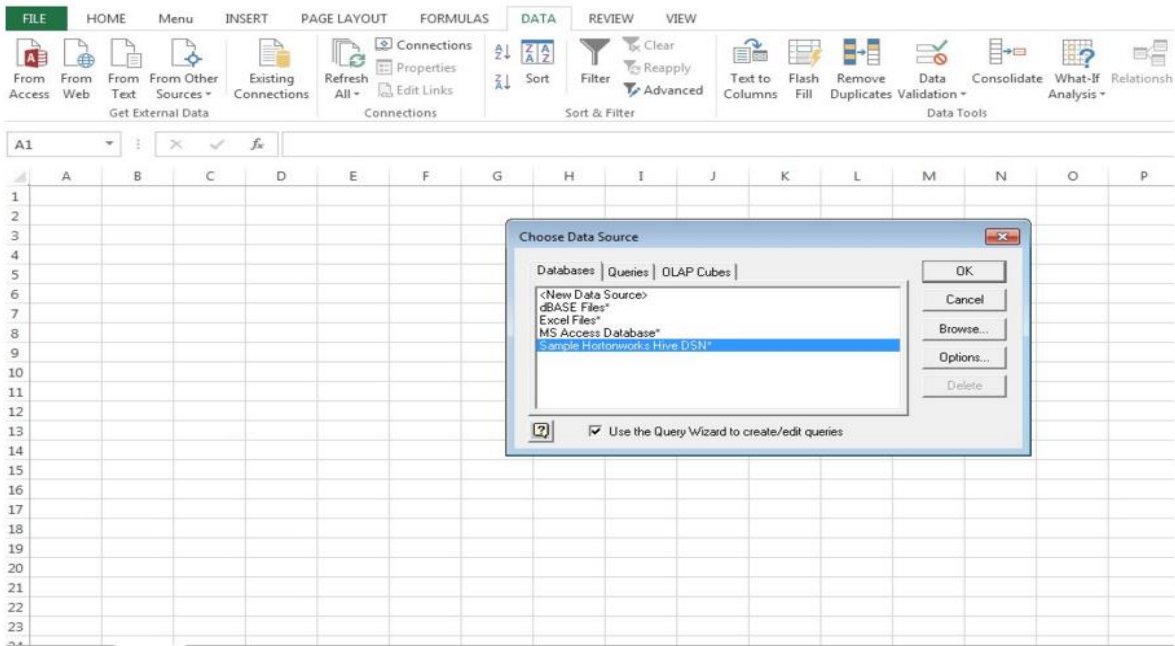
The screenshot shows the Ambari UI interface. On the left, the Database Explorer shows a tree view of databases, with 'firewall_logs' highlighted. The main area is the Query Editor, which contains the query: `SELECT * FROM firewall_logs LIMIT 100;`. Below the query editor, the Query Process Results section shows the status 'Succeeded' and a table of results. The table has four columns: 'firewall_logs.time', 'firewall_logs.ip', 'firewall_logs.country', and 'firewall_logs.status'. The results are as follows:

firewall_logs.time	firewall_logs.ip	firewall_logs.country	firewall_logs.status
2016-01-21T21:17:03	254.173.216.140	KR	ERROR
2016-01-21T21:17:03	180.23.254.139	KR	ERROR
2016-01-21T21:17:03	68.183.85.165	MW	ERROR

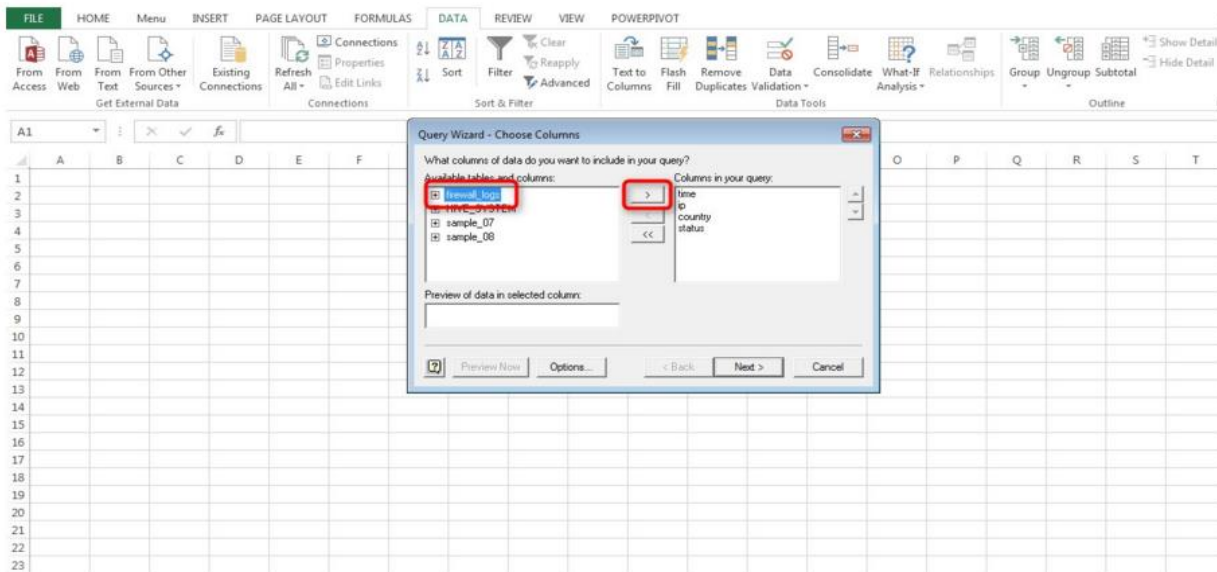
Import the server log data into Microsoft Excel. In Windows, open a new Excel workbook, then select Data > From Other Sources > from Microsoft Query.



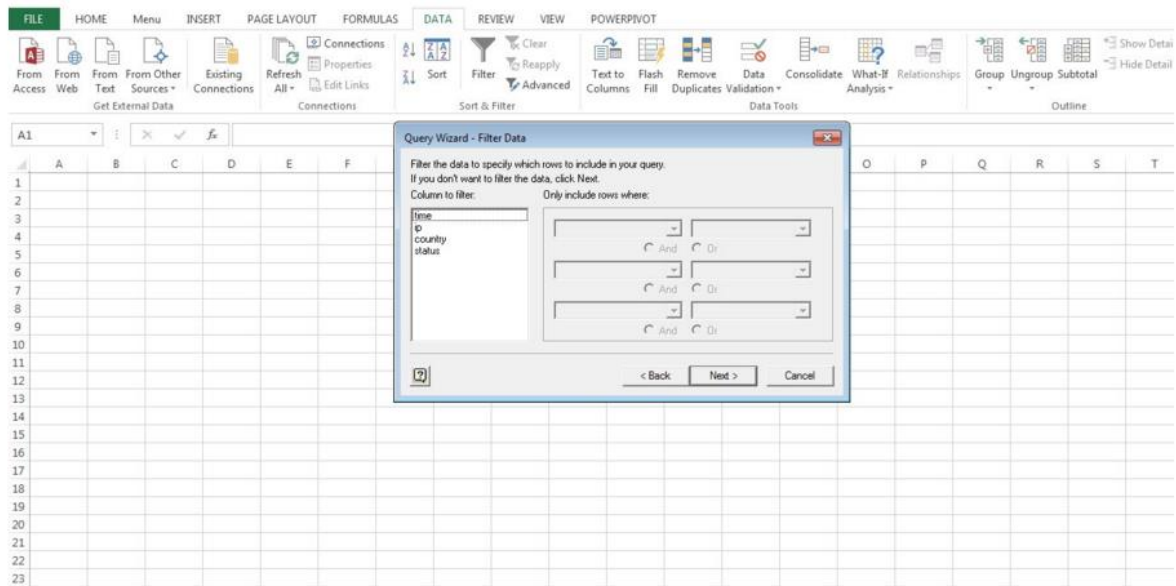
In Data source pop-up, select the Hortonworks ODBC data connection that is installed and then click okay. Hortonworks ODBC driver enables you to access the Hortonworks data using Excel and other Business Intelligence (BI) programs which support ODBC.



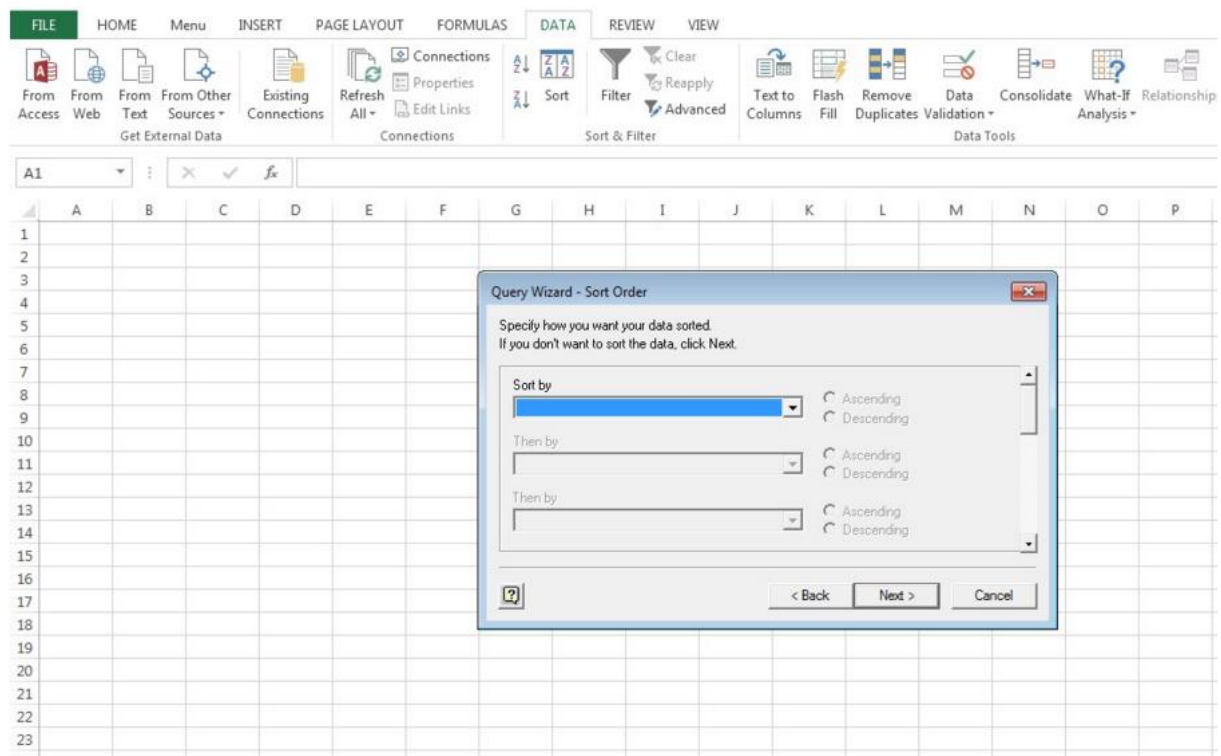
After the connection to Sandbox is done, the query wizard will appear. Select the “firewall_logs” table in the Available Tables column box then click right arrow to move the entire table to the query and then click NEXT continue.



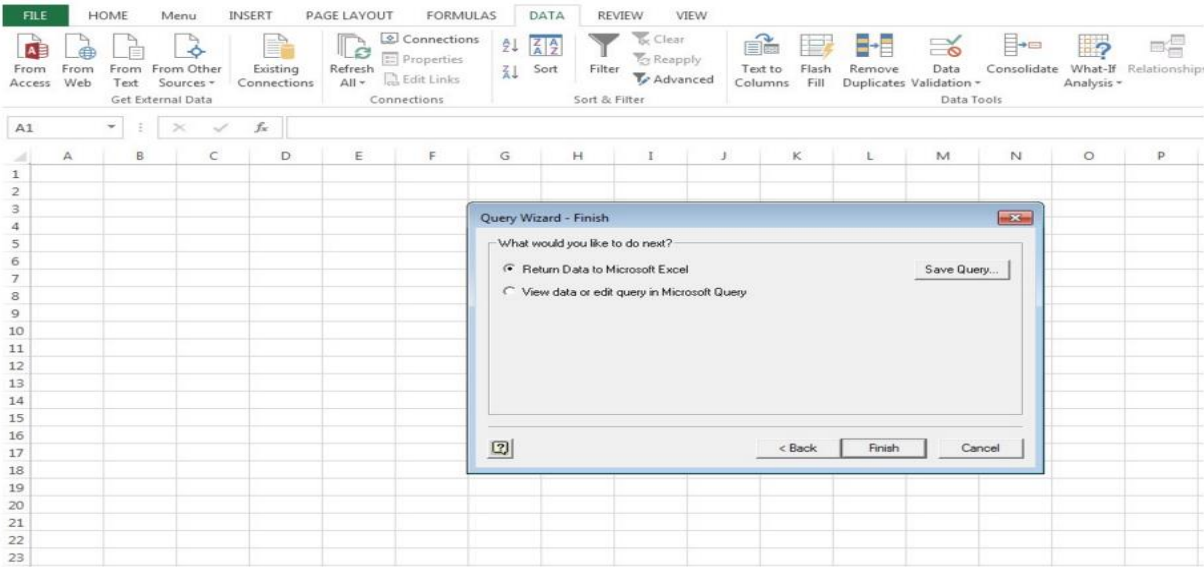
Click next without filtering the data



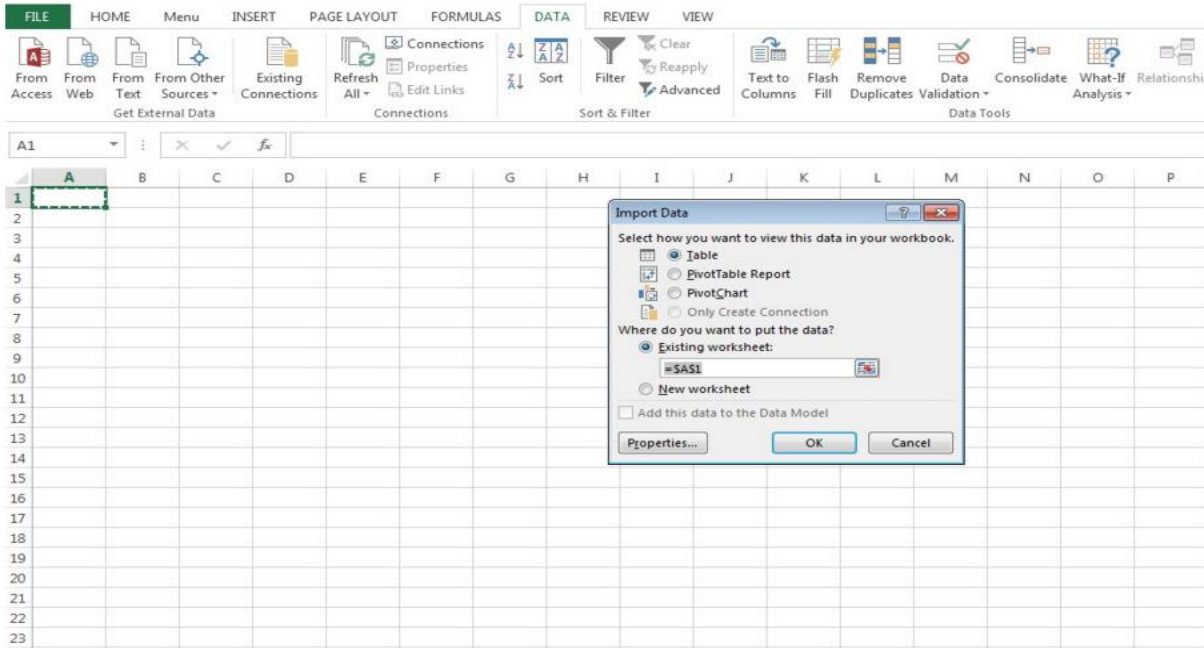
Click next without sorting the data



Click **Finish** on the Query Wizard to retrieve the query data from the Sandbox and import it into Microsoft Excel.



Click **OK** for default settings and import the data as table.



Chapter V: Timeline

Table 2

Dates and Description of the Task

Task(s)			
Start Date	End Date	Description	Duration (Days)
8/16/2015	8/29/2015	Winding up the Data Collection	14
9/1/2015	9/7/2015	Acquiring Servers	7
9/9/2015	9/20/2015	Required Software	12
9/25/2015	10/15/2015	Installation and Configuration	11
10/20/2015	11/09/2015	All Necessary Features Installation	20
12/1/2015	12/20/2015	Implementing the replication of the Horton Works Module for Log Analysis	20
1/2/2015	1/20/2015	Testing the Module used for Log Analysis	18
2/2/2015	2/28/2015	Complete report on findings	26
3/16/2015	4/08/2015	Get ready for final defense and working on paper	23

I have met the estimated timeline as planned, though I have faced some problems.

Chapter VI: Conclusion and Future Scope

By using the Hortonworks module for secure log data analysis, we can easily get an Excel view of the log data in any organization with minimal infrastructure.

Visualize the server log data using Microsoft Excel Power View

- Review the network traffic by country.

After the successful export of the log data into Excel, there is an option in Insert which is Power View. By using this we can get the network traffic by country.

- Zoom in on one particular country

In Power View Fields, select the country and status which will get us map view and we can zoom into one country.

- Generate a list of attacking IP addresses

Select all the data and sort it out with ERROR so that it will show the attacking IP addresses.

References

- Chansler, R., Kuang, H., Radia, S., Shvachko, K., & Srinivas, S. (2011). The Hadoop distributed file system. In A. Brown & G. Wilson (Eds.) *The architecture of open source applications: elegance, evolution, and a few fearless hacks* (pp. 121-134). Retrieved from <http://www.aosabook.org/en/hdfs.html>
- Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified data processing on large clusters*. Retrieved from <http://research.google.com/archive/mapreduce.html>
- Ghemawat, S., Hoberk, H., & Leung, S-T. (2003). *The Google file system*. Retrieved from <http://research.google.com/archive/gfs.html>
- Google Official Blog. (2008). *Sorting 1PB with MapReduce*. Retrieved from <http://googleblog.blogspot.com/2008/11/sorting-1pbwith-mapreduce.html>
- Hadoop. (n.d.). *Welcome to Apache pig*. Retrieved from <http://pig.apache.org/>
- O'Malley, O. (2008). *Terabyte sort on apache hadoop*. Retrieved from <http://sortbenchmark.org/YahooHadoop.pdf>
- O'Malley, O., & Murthy, A. C. (2009). *Winning a 60 second dash with a yellow elephant*. Retrieved from <http://sortbenchmark.org/Yahoo2009.pdf>
- Segaran, T., & Hammerbacher, T. (2009). *Beautiful data: The stories behind elegant data solutions*. Boston, MA: O'Reilly Media.
- Shvachko, K. V. (2010). *HDFS scalability: The limits to growth*. Retrieved from <https://www.usenix.org/publications/login/april-2010-volume-35-number-2/hdfs-scalability-limits-growth>
- White, T. (2015). *Hadoop: The definitive guide* (4th edition). Boston, MA: O'Reilly Media.

Yahoo Developer Network. (n.d.). *Scaling hadoop to 4000 nodes at Yahoo!* Retrieved from

<https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html>

Zawodny, J. (2008). *Yahoo! launches world's largest hadoop production application*. Retrieved

from <https://developer.yahoo.com/blogs/hadoop/yahoo-launches-world-largest-hadoop-production-application-398.html>