MODULE 2
ESSENTIAL HADOOP TOOLS, HADOOP YARN APPLICATIONS, MANAGING HADOOP WITH APACHE AMBARI, BASIC HADOOP ADMINISTRATION PROCEDURES

Essential Hadoop Tools

- **USING APACHE PIG**
- **USING APACHE HIVE**
- **USING APACHE SQOOP TO ACQUIRE RELATIONAL DATA**
- **USING APACHE FLUME TO ACQUIRE DATA STREAMS**
- **MANAGE HADOOP WORKFLOWS WITH APACHE OOZIE**
- **USING APACHE HBASE**

USING APACHE PIG

- Apache Pig is a high-level language that enables programmers to write complex MapReduce transformations using a simple scripting language.
- Pig Latin (the actual language) defines a set of transformations on a data set such as aggregate, join, and sort.
- Pig is often used to extract, transform, and load (ETL) data pipelines, quick research on raw data, and iterative data processing.

Apache Pig usage modes:

- The first is a local mode in which all processing is done on the local machine.
- The non-local (cluster) modes are MapReduce and Tez.
- These modes execute the job on the cluster using either the MapReduce engine or the optimized Tez engine. (Tez, which is Hindi for "speed," optimizes multistep Hadoop jobs such as those found in many Pig queries.)
- There are also interactive and batch modes available; they enable Pig applications to be developed locally in interactive modes, using small amounts of data, and then run at scale on the cluster in a production mode. The modes are summarized in Table 7.1.

Table 7.1 Apache Pig Usage Modes

	Local Mode	Tez Local Mode	MapReduce Mode	Tez Mode
Interactive Mode	Yes	Experimental	Yes	Yes
Batch Mode	Yes	Experimental	Yes	Yes

USING APACHE HIVE

- Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, ad hoc queries, and the analysis of large data sets using a SQL-like language called HiveQL.
- Hive is considered the de facto standard for interactive SQL queries over petabytes of data using Hadoop and offers the following features:
    1. Tools to enable easy data extraction, transformation, and loading (ETL)
    2. A mechanism to impose structure on a variety of data formats
    3. Access to files stored either directly in HDFS or in other data storage systems such as HBase
    4. Query execution via MapReduce and Tez (optimized MapReduce)
- Hive provides users who are already familiar with SQL the capability to query the data on Hadoop clusters.
- At the same time, Hive makes it possible for programmers who are familiar with the MapReduce framework to add their custom mappers and reducers to Hive queries.
- Hive queries can also be dramatically accelerated using the Apache Tez framework under YARN in Hadoop version 2.
- 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 can be used with any Java Database Connectivity (JDBC)– compliant database and has been tested on Microsoft SQL Server, PostgresSQL, MySQL, and Oracle.
- In version 1 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.
- Instead, version 2 offers more generalized ways to accomplish these tasks.

USING APACHE SQOOP TO ACQUIRE RELATIONAL DATA

Apache Sqoop Import and Export Methods

Figure 7.1 describes the Sqoop data import (to HDFS) process.

The data import is done in two steps.

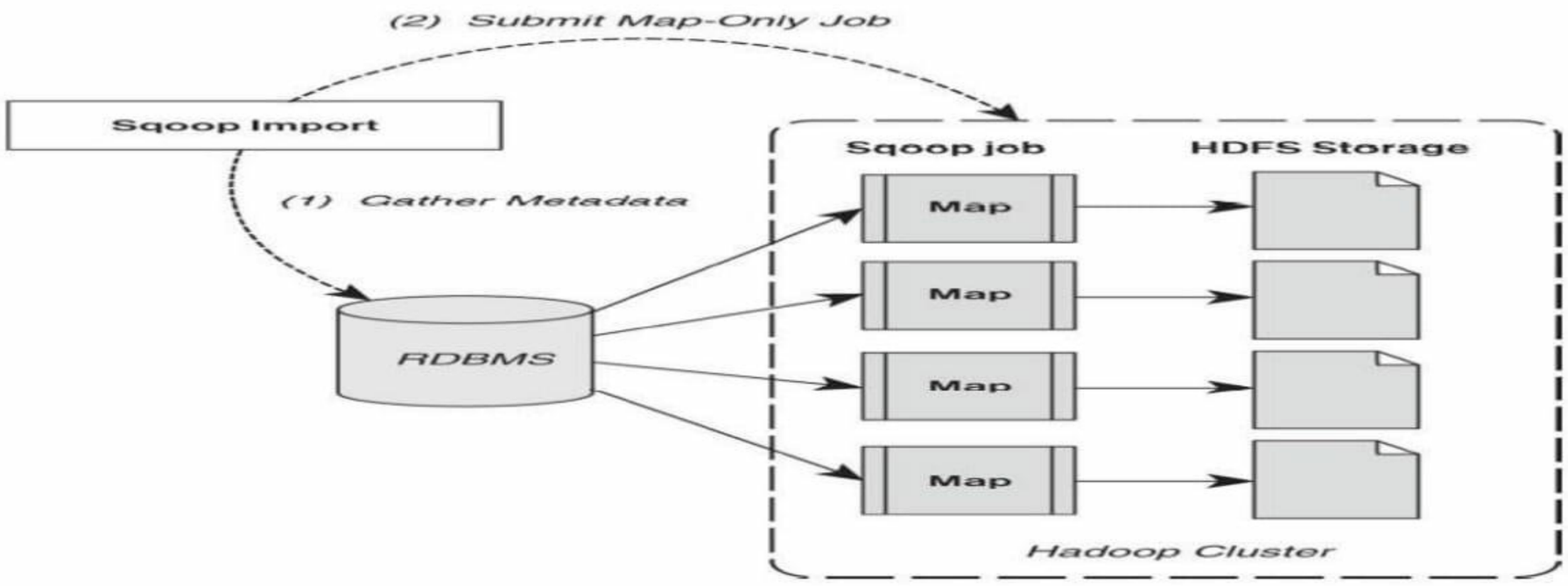
- In the **first step**, shown in the figure, Sqoop examines the database to gather the necessary metadata for the data to be imported.
- The **second step** is a map-only (no reduce step) Hadoop job that Sqoop submits to the cluster.
- This job does the actual data transfer using the metadata captured in the previous step.

- Note that each node doing the import must have access to the database.

Import method

- 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 separating 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.

Figure 7.1 Two-step Apache Sqoop data import method

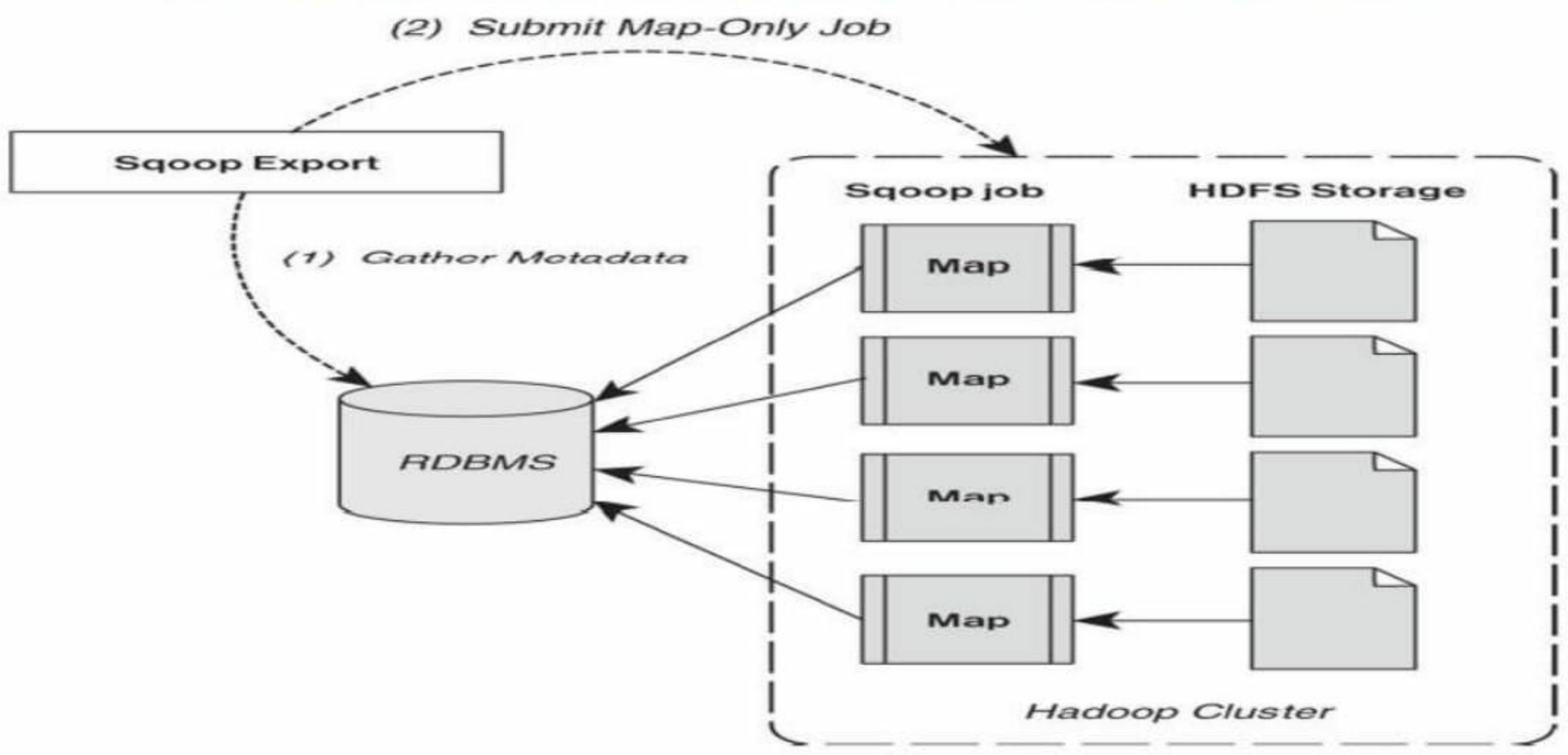


Export method

Data export from the cluster works in a similar fashion.

- The export is done in two steps, as shown in Figure 7.2. As in the import process, the **first step** is to examine the database for metadata.

Figure 7.2 Two-step Sqoop data export method



- 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 various 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 7.2 summarizes the changes from version 1 to version 2.
- Due to these changes, any new development should be done with Sqoop version 2.

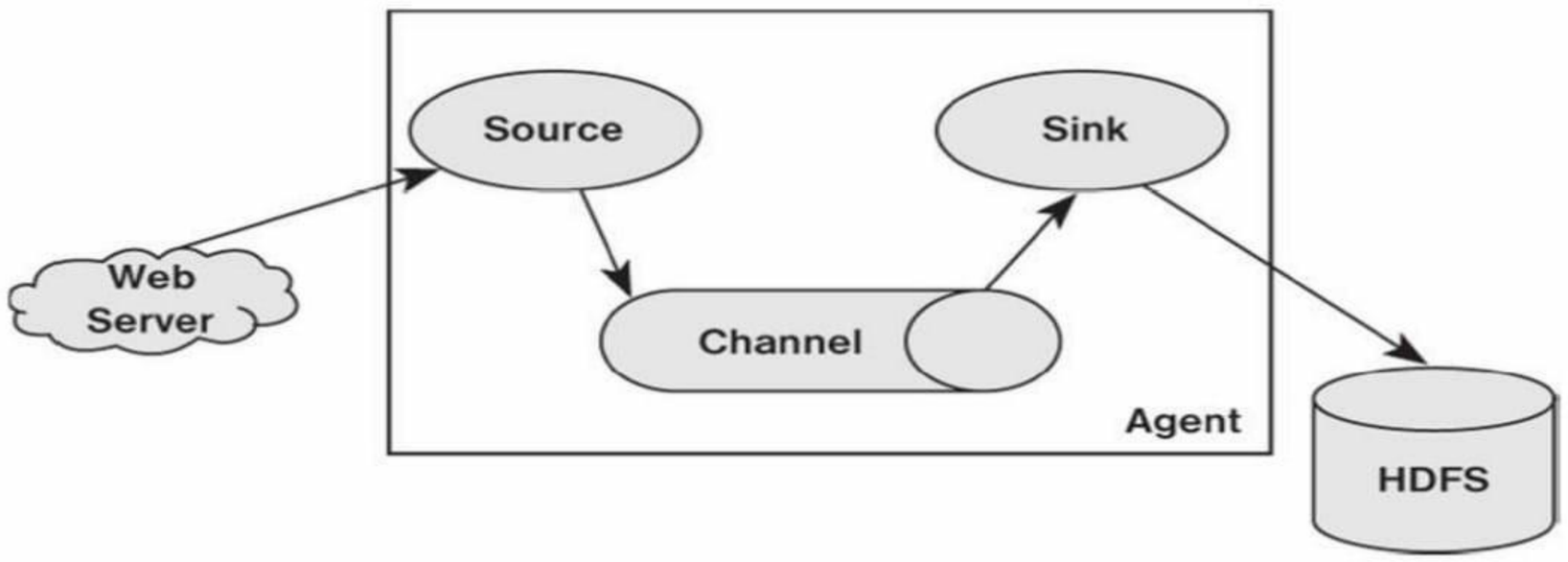
TABLE 7.2 APACHE SQOOP VERSION COMPARISON

Feature	Sqoop Version 1	Sqoop Version 2
Connectors for all major RDBMSs	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.

USING APACHE FLUME TO ACQUIRE DATA STREAMS

- Apache Flume is an independent agent designed to collect, transport, and store data into HDFS.
- Often data transport involves a number of Flume agents that may traverse a series of machines and locations.
- Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source. As shown in Figure 7.3, a Flume agent is composed of three components.

Figure 7.3 Flume agent with source, channel, and sink



Source. The source component receives data and sends it to a channel. It can send the data to more than one channel. The input data can be from a realtime source (e.g., weblog) or another Flume agent.

Channel. A channel is a data queue that forwards the source data to the sink destination. It can be thought of as a buffer that manages input (source) and output (sink) flow rates.

Sink. The sink delivers data to destination such as HDFS, a local file, or another Flume agent.

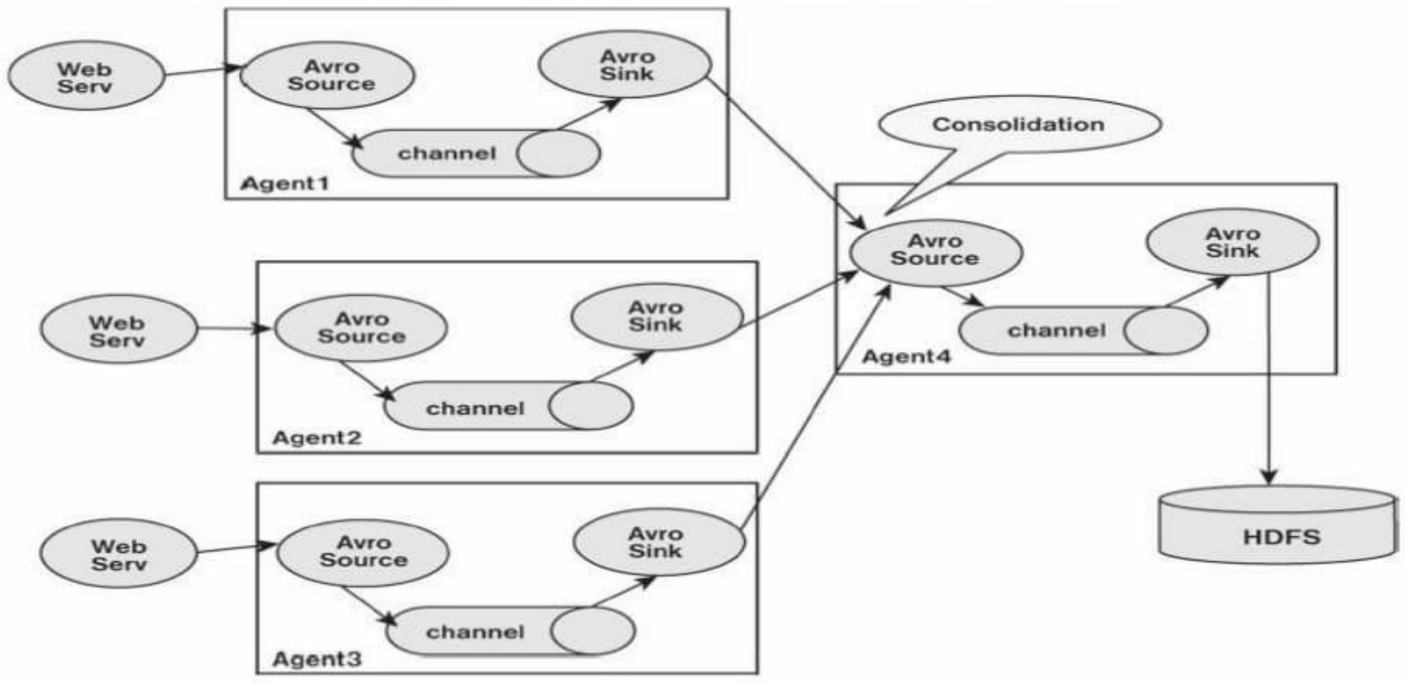
- A Flume agent must have all three of these components defined.
- A Flume agent can have several sources, channels, and sinks.
- Sources can write to multiple channels, but a sink can take data from only a single channel.
- Data written to a channel remain in the channel until a sink removes the data. By default, the data in a channel are kept in memory but may be optionally stored on disk to prevent data loss in the event of a network failure.
- As shown in Figure 7.4, Sqoop agents may be placed in a pipeline, possibly to traverse several machines or domains.
- This configuration is normally used when data are collected on one machine (e.g., a web server) and sent to another machine that has access to HDFS
- In a Flume pipeline, the sink from one agent is connected to the source of another.
- The data transfer format normally used by Flume, which is called Apache Avro, provides several useful features.
- First, Avro is a data serialization/deserialization system that uses a compact binary format.
- The schema is sent as part of the data exchange and is defined using JSON (JavaScript Object Notation).
- Avro also uses remote procedure calls (RPCs) to send data.
- That is, an Avro sink will contact an Avro source to send data.

Sqoop agents placed as pipeline Other configurations

- Another useful Flume configuration is shown in Figure 7.5.

- In this configuration, Flume is used to consolidate several data sources before committing them to HDFS.
- There are many possible ways to construct Flume transport networks.
- In addition, other Flume features not described in depth here include plug-ins and interceptors that can enhance Flume pipelines.

Figure 7.5 A Flume consolidation network



MANAGE HADOOP WORKFLOWS WITH APACHE OOZIE

- Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs.
- For instance, complete data input and analysis may require several discrete Hadoop jobs to be run as a workflow in which the output of one job serves as the input for a successive job.
- Oozie is designed to construct and manage these workflows.
- Oozie is not a substitute for the YARN scheduler.
- That is, YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control Hadoop jobs on the cluster.
- Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions. (DAGs are basically graphs that cannot have directed loops.)
- Three types of Oozie jobs are permitted:
    1. **Workflow**—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete.
    2. **Coordinator**—a scheduled workflow job that can run at various time intervals or when data become available.
    3. **Bundle**—a higher-level Oozie abstraction that will batch a set of coordinator jobs.

- Oozie is integrated with the rest of the Hadoop stack, supporting several types of Hadoop jobs out of the box (e.g., Java MapReduce, Streaming MapReduce, Pig,

Hive, and Sqoop) as well as systemspecific jobs (e.g., Java programs and shell scripts). Oozie also provides a CLI and a web UI for monitoring jobs.

Figure 7.6 A simple Oozie DAG workflow

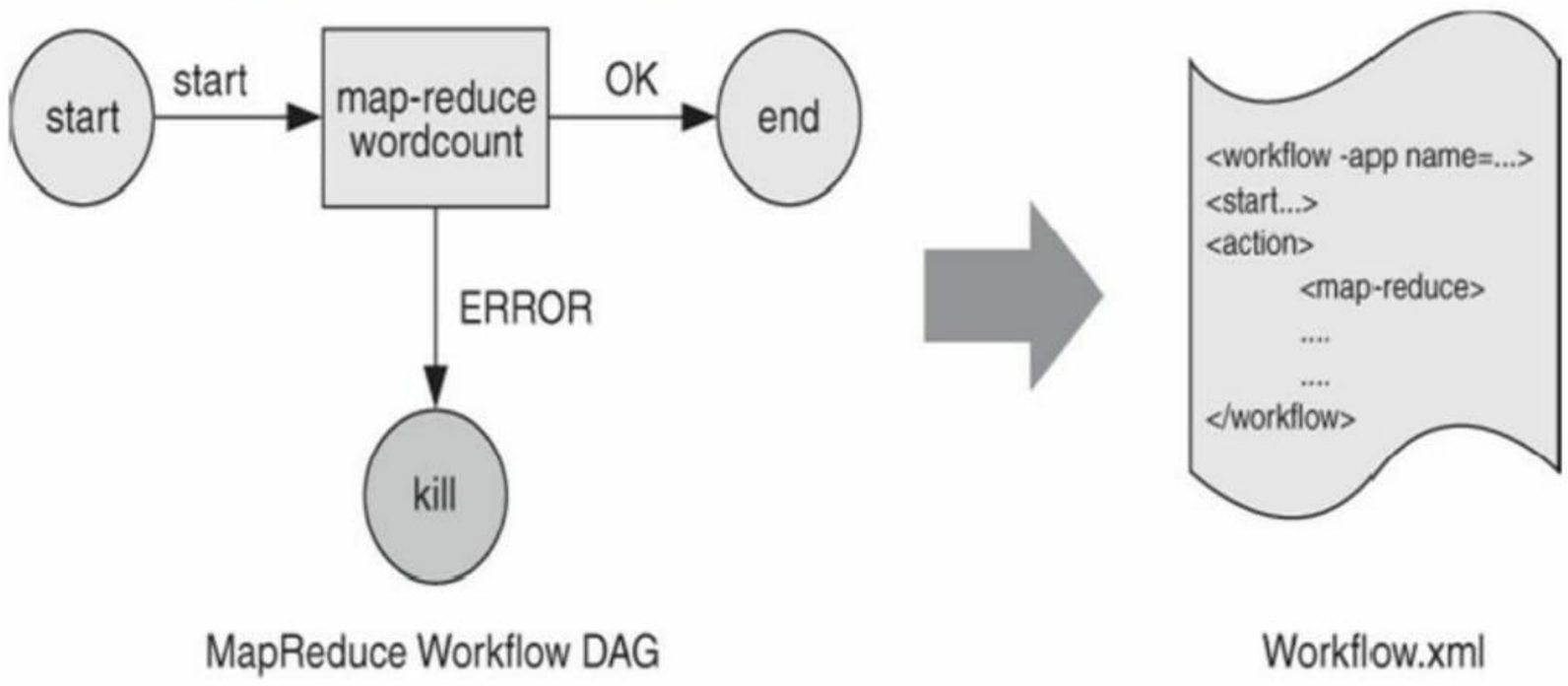
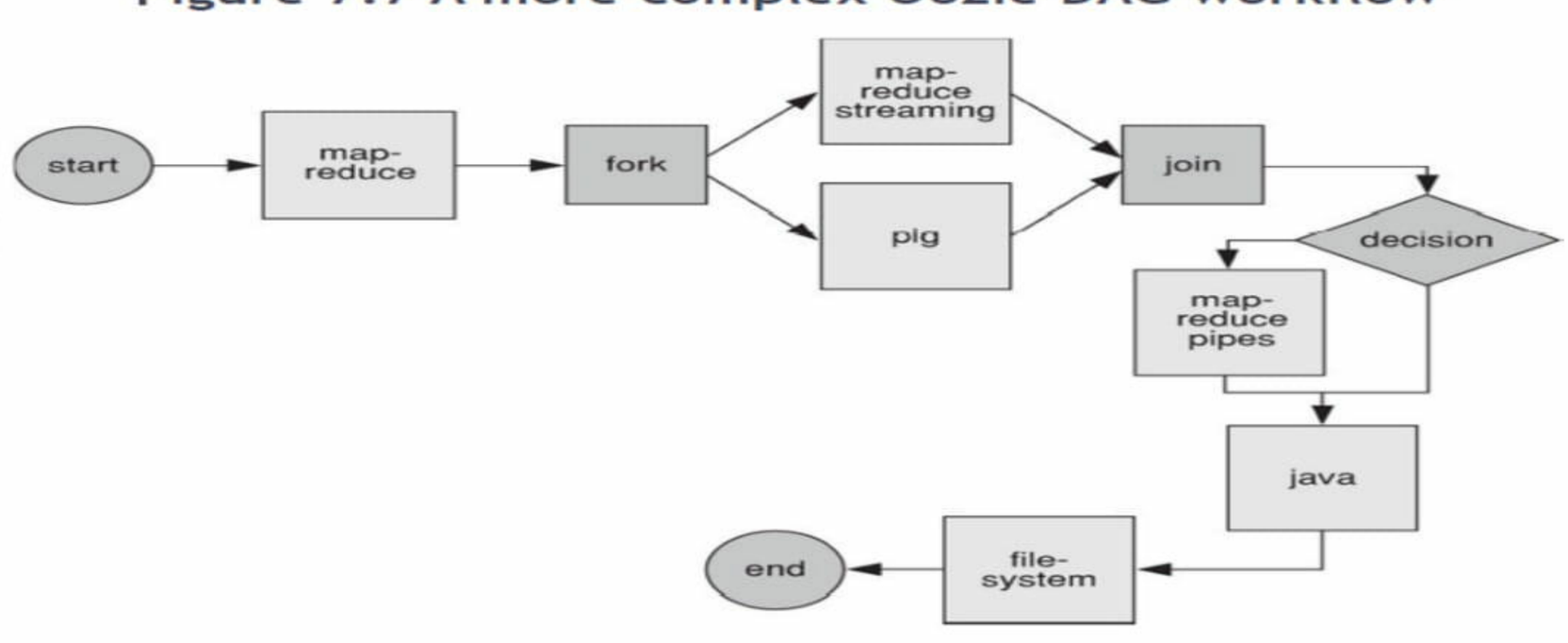


Figure 7.6 depicts a simple Oozie workflow. In this case, Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed. Oozie workflow definitions are written in hPDL (an XML Process Definition Language). Such workflows contain several types of nodes:

1. **Control flow** nodes define the beginning and the end of a workflow. They include start, end, and optional fail nodes.
2. **Action nodes** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands.
3. **Fork/join nodes** enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.
4. **Control flow nodes** enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g., file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language) that evaluate to either true or false.

Figure 7.7 A more complex Oozie DAG workflow



USING APACHE HBASE

- Apache HBase is an open source, distributed, versioned, **nonrelational** database modeled after **Google's Bigtable.**
- Like Bigtable, HBase leverages the **distributed data storage** provided by the underlying distributed file systems spread across **commodity servers**.
- Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

Some of the more important features include the following capabilities:

- **Linear** and **modular** scalability
- Strictly **consistent reads and writes**
- Automatic and **configurable sharding of tables**
- Automatic **failover** support between **RegionServers**
- Convenient **base classes** for **backing Hadoop MapReduce jobs** with Apache HBase tables
- **Easy-to-use Java API** for client access

HBase Data Model Overview

- A table in HBase is **similar** to other databases, having **rows and columns**.
- Columns in HBase are grouped into **column families**, all with the **same prefix**.
- For example, consider a table of daily stock prices. There may be a column family called "**price**" that has four *members— price:open, price:close, price:low, and price:high.*
- A column does not need to be a family. For instance, the stock table may have a column named "**volume**" indicating how many shares were traded.
- All **column family** members are **stored** together in the **physical file system**.
- Specific **HBase cell values** are identified by *a row key, column (column family and column), and version (timestamp).*
- It is possible to have many versions of data within an **HBase cell.**
- A version is specified as a **timestamp** and is created *each time data are written to a cell.*
- Almost anything can serve as a *row key*, from strings to binary representations of longs to *serialized data structures*.
- Rows are ***lexicographically*** sorted with the **lowest order appearing first in a table**.
- The **empty byte array denotes** both the start and the end of a ***table's namespace***.
- All ***table accesses*** are via the ***table row key***, which is considered its ***primary key***

YARN DISTRIBUTED-SHELL

- The Hadoop YARN project includes the Distributed-Shell application, which is an example of a ***Hadoop non-MapReduce application built on top of YARN***.
- Distributed-Shell is a ***simple mechanism for running shell commands and scripts in containers on multiple nodes*** in a Hadoop cluster.

- This application is not meant to be a **production administration tool**, but rather a demonstration of the non-MapReduce capability that can be implemented on top of YARN.
- There are multiple **mature implementations** of a distributed shell **that administrators** typically use to manage a cluster of machines.
- In addition, Distributed-Shell can be **used as a starting point** for **exploring and building** Hadoop YARN applications.

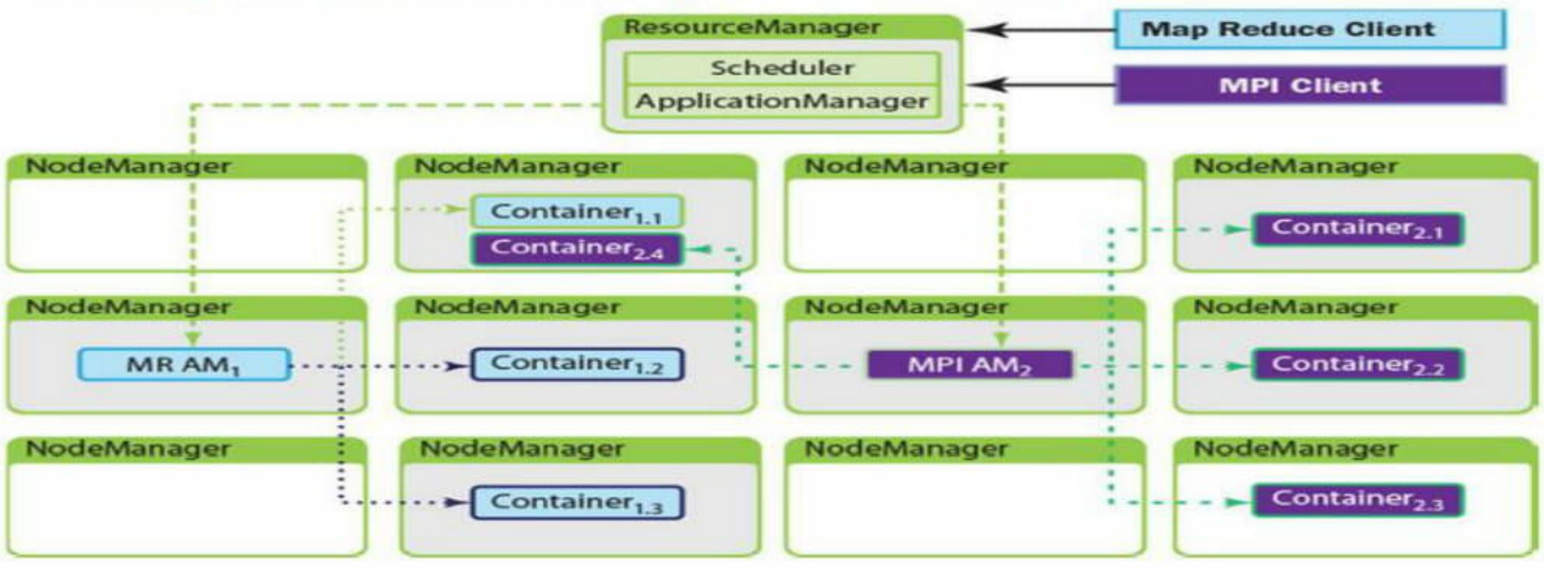
STRUCTURE OF YARN APPLICATIONS

- YARN ResourceManager runs as a scheduling ***daemon on a dedicated machine*** and acts as the central authority for allocating resources to the various competing applications in the cluster.
- The ResourceManager has a ***central and global view of all cluster resources*** and, therefore, can ***ensure fairness, capacity, and locality*** are shared across all users.
- Depending on the application ***demand, scheduling priorities, and resource availability***, the ResourceManager ***dynamically allocates*** resource containers to applications to run on particular nodes.
- A ***container*** is a logical **bundle of resources** (e.g., memory, cores) bound to a particular cluster node.
- To enforce and **track** such assignments, the ResourceManager interacts with a special **system daemon running on each node** called the **NodeManager.**
- Communications between the ResourceManager and NodeManagers are ***heartbeat based for scalability***.
- NodeManagers are **responsible** for ***local monitoring of resource availability, fault reporting, and container life-cycle management*** (e.g., starting and killing jobs).
- The ResourceManager depends on the NodeManagers for its "***global view"*** of the cluster.
- User applications are submitted to the ResourceManager via a **public protocol** and go through an admission control phase during which **security credentials** are validated and various **operational and administrative checks** are performed.
- Those applications that are accepted pass to the ***scheduler and are allowed to run***.
- Once the scheduler has enough ***resources to satisfy the request***, the application is ***moved from an accepted state to a running state.***
- Aside from internal bookkeeping, this process involves ***allocating a container for the single ApplicationMaster*** and spawning it on a node in the cluster. Often called container0, the ApplicationMaster does not have any additional ***resources at this point, but rather must request additional resources from the ResourceManager.***
- The ApplicationMaster is the "master" user job that ***manages all application life-cycle aspects***, including ***dynamically increasing and decreasing resource consumption*** (i.e., containers), ***managing the flow of execution*** (e.g., in case of

MapReduce jobs, running reducers against the output of maps), ***handling faults and computation skew, and performing other local optimizations***.

- The ApplicationMaster is designed to ***run arbitrary user code*** that can be written in any programming language, as all ***communication with the ResourceManager and NodeManager*** is encoded using extensible network protocols.
- YARN makes **few assumptions about the ApplicationMaster**, although in practice it expects most jobs will use a ***higher-level programming framework***.
- By ***delegating all these functions*** to ApplicationMasters, YARN's architecture gains a ***great deal of scalability, programming model flexibility, and improved user agility***.
- For example, upgrading and testing a new MapReduce framework can be done independently of other running MapReduce frameworks.
- Typically, an ApplicationMaster will need to harness the processing power of multiple servers to complete a job.
- To achieve this, the ApplicationMaster ***issues resource requests*** to the ResourceManager.
- The form of ***these requests includes*** specification of ***locality preferences*** (e.g., to accommodate HDFS use) ***and properties of the containers.***
- The ResourceManager will ***attempt to satisfy*** the resource requests coming from each application **according to availability and scheduling policies**.
- When a ***resource is scheduled on behalf*** of an ApplicationMaster, the ResourceManager generates a ***lease for the resource***, which is acquired by a ***subsequent ApplicationMaster heartbeat***.
- The ApplicationMaster then works with the NodeManagers to start the resource.
- A **token-based security** mechanism guarantees its **authenticity** when the ApplicationMaster presents the container lease to the NodeManager.
- In a typical situation, **running containers** will communicate with the ApplicationMaster through an **application-specific protocol** to report status and health information and to **receive framework-specific commands**.
- In this way, YARN provides a basic infrastructure for monitoring and **life-cycle management of containers**, while each framework manages application-specific semantics independently.

Figure 8.1 YARN architecture with two clients (MapReduce and MPI). The darker client (MPI AM2) is running an MPI application, and the lighter client (MR AM1) is running a MapReduce application.

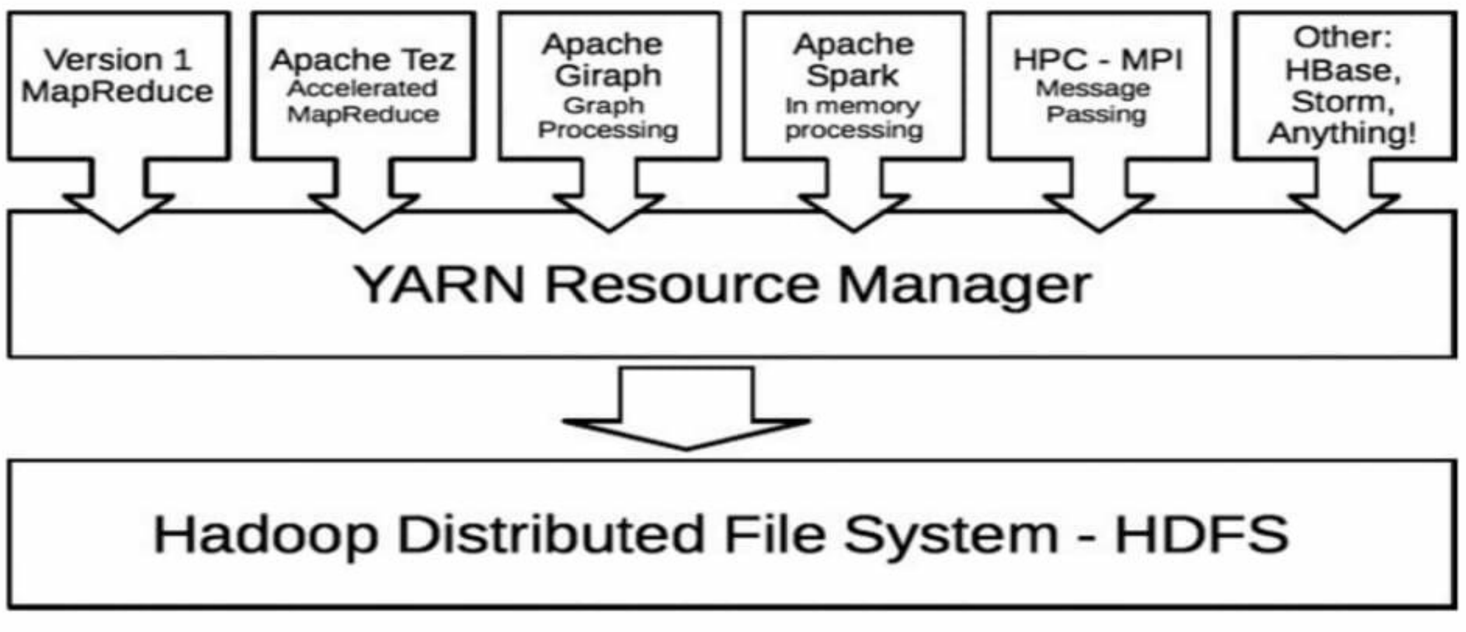


- This design stands in sharp contrast to the original Hadoop version 1 design, in which scheduling was designed and integrated around managing only MapReduce tasks.
- Figure 8.1 illustrates the relationship between the application and YARN components.
- The YARN components appear as the large outer boxes (ResourceManager and NodeManagers), and the two applications appear as smaller boxes (containers), one dark and one light.
- Each application uses a different ApplicationMaster; the darker client is running a Message Passing Interface (MPI) application and the lighter client is running a traditional MapReduce application.

YARN APPLICATION FRAMEWORKS

- One of the most exciting aspects of Hadoop version 2 is the capability to run all types of applications on a Hadoop cluster.
- In Hadoop version 1, the only processing model available to users is MapReduce.
- In Hadoop version 2, MapReduce is separated from the resource management layer of Hadoop and placed into its own application framework.
- Indeed, the growing number of YARN applications offers a high level and multifaceted interface to the Hadoop data lake
- YARN presents a resource management platform, which provides services such as scheduling, fault monitoring, data locality, and more to MapReduce and other frameworks.
- Figure 8.2 illustrates some of the various frameworks that will run under YARN

Figure 8.2 Example of the Hadoop version 2 ecosystem. Hadoop version 1 supports batch MapReduce applications only



Distributed-Shell

- Distributed-Shell is an example application included with the Hadoop core components that demonstrates how to write applications on top of YARN.
- It provides a simple method for running shell commands and scripts in containers in parallel on a Hadoop YARN cluster.

Hadoop MapReduce

- MapReduce was the first YARN framework and drove many of YARN's requirements.
- It is integrated tightly with the rest of the Hadoop ecosystem projects, such as Apache Pig, Apache Hive, and Apache Oozie.

Apache Tez

- Many Hadoop jobs involve the execution of a complex directed acyclic graph (DAG) of tasks using separate MapReduce stages.
- Apache Tez generalizes this process and enables these tasks to be spread across stages so that they can be run as a single, all-encompassing job.
- Tez can be used as a MapReduce replacement for projects such as Apache Hive and Apache Pig.

Apache Giraph

- Apache Giraph is an iterative graph processing system built for high scalability.
- Facebook, Twitter, and LinkedIn use it to create social graphs of users.
- Giraph was originally written to run on standard Hadoop V1 using the MapReduce framework, but that approach proved inefficient and totally unnatural for various reasons.
- The native Giraph implementation under YARN provides the user with an iterative processing model that is not directly available with MapReduce. Support for YARN has been present in Giraph since its own version 1.0 release.
- In addition, using the flexibility of YARN, the Giraph developers plan on implementing their own web interface to monitor job progress.

Hoya: HBase on YARN

- The Hoya project creates dynamic and elastic Apache HBase clusters on top of YARN.
- A client application creates the persistent configuration files, sets up the HBase cluster XML files, and then asks YARN to create an ApplicationMaster.
- YARN copies all files listed in the client's application-launch request from HDFS into the local file system of the chosen server, and then executes the command to start the Hoya ApplicationMaster.
- Hoya also asks YARN for the number of containers matching the number of HBase region servers it needs

Dryad on YARN

- Similar to Apache Tez, Microsoft's Dryad provides a DAG as the abstraction of execution flow.
- This framework is ported to run natively on YARN and is fully compatible with its non-YARN version.
- The code is written completely in native C++ and C# for worker nodes and uses a thin layer of Java within the application.

Apache Spark

- Spark was initially developed for applications in which keeping data in memory improves performance, such as iterative algorithms, which are common in machine learning, and interactive data mining.
- Spark differs from classic MapReduce in two important ways.
- First, Spark holds intermediate results in memory, rather than writing them to disk.
- Second, Spark supports more than just MapReduce functions; that is, it greatly expands the set of possible analyses that can be executed over HDFS data stores. It also provides APIs in Scala, Java, and Python.
- Since 2013, Spark has been running on production YARN clusters at Yahoo!. The advantage of porting and running Spark on top of YARN is the common resource management and a single underlying file system.

Apache Storm

- Traditional MapReduce jobs are expected to eventually finish, but Apache Storm continuously processes messages until it is stopped.
- This framework is designed to process unbounded streams of data in real time. It can be used in any programming language.
- The basic Storm use-cases include real-time analytics, online machine learning, continuous computation, distributed RPC (remote procedure calls), ETL (extract, transform, and load), and more.
- Storm provides fast performance, is scalable, is fault tolerant, and provides processing guarantees.
- It works directly under YARN and takes advantage of the common data and resource management substrate.

Apache REEF: Retainable Evaluator Execution Framework

- YARN's flexibility sometimes requires significant effort on the part of application implementers.
- The steps involved in writing a custom application on YARN include building your own ApplicationMaster, performing client and container management, and handling aspects of fault tolerance, execution flow, coordination, and other concerns.
- The REEF project by Microsoft recognizes this challenge and factors out several components that are common to many applications, such as storage management, data caching, fault detection, and checkpoints.
- Framework designers can build their applications on top of REEF more easily than they can build those same applications directly on YARN, and can reuse these common services/libraries.
- REEF's design makes it suitable for both MapReduce and DAG-like executions as well as iterative and interactive computations

Hamster: Hadoop and MPI on the Same Cluster

- The Message Passing Interface (MPI) is widely used in high-performance computing (HPC).
- MPI is primarily a set of optimized message-passing library calls for C, C++, and Fortran that operate over popular server interconnects such as Ethernet and InfiniBand.
- Because users have full control over their YARN containers, there is no reason why MPI applications cannot run within a Hadoop cluster.
- The Hamster effort is a work-in-progress that provides a good discussion of the issues involved in mapping MPI to a YARN cluster.
- Currently, an alpha version of MPICH2 is available for YARN that can be used to run MPI applications.

Apache Flink: Scalable Batch and Stream Data Processing

- Apache Flink is a platform for efficient, distributed, generalpurpose data processing.
- It features powerful programming abstractions in Java and Scala, a high-performance run time, and automatic program optimization.
- It also offers native support for iterations, incremental iterations, and programs consisting of large DAGs of operations.
- Flink is primarily a stream-processing framework that can look like a batch-processing environment.
- The immediate benefit from this approach is the ability to use the same algorithms for both streaming and batch modes
- However, Flink can provide low-latency similar to that found in Apache Storm, but which is not available in Apache Spark.

Apache Slider: Dynamic Application Management

- Apache Slider (incubating) is a YARN application to deploy existing distributed applications on YARN, monitor them, and make them larger or smaller as desired in real time.
- Applications can be stopped and then started; the distribution of the deployed application across the YARN cluster is persistent and allows for best-effort placement close to the previous locations.
- Applications that remember the previous placement of data (such as HBase) can exhibit fast startup times by capitalizing on this feature.
- YARN monitors the health of "YARN containers" that are hosting parts of the deployed applications.
- If a container fails, the Slider manager is notified. Slider then requests a new replacement container from the YARN ResourceManager.

- Some of Slider's other features include user creation of on-demand applications, the ability to stop and restart applications as needed (preemption), and the ability to expand or reduce the number of application containers as needed.
- The Slider tool is a Java command-line application

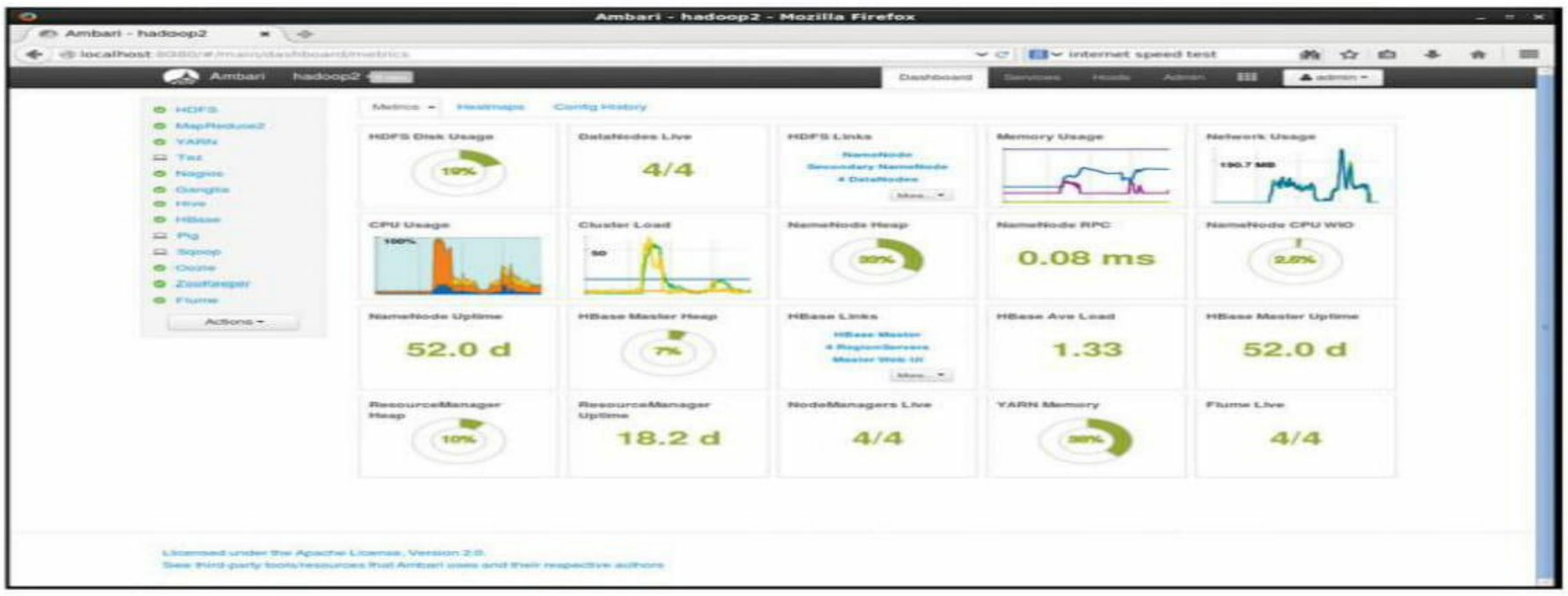
Managing Hadoop with Apache Ambari

- QUICK TOUR OF APACHE AMBARI
- Dashboard View
- Services View
- Hosts View
- Admin View
- Views View
- MANAGING HADOOP SERVICES
- CHANGING HADOOP PROPERTIES

QUICK TOUR OF APACHE AMBARI

- After completing the initial installation and logging into Ambari (as explained in Chapter 2), a dashboard similar to that shown in Figure 9.1 is presented. The same four-node cluster as created.
- If you need to reopen the Ambari dashboard interface,simply enter the following command **$ firefox localhost:8080** The default login and password are admin and admin, respectively. Before continuing any further, you should change the default password.52
- To change the password, select Manage Ambari from the Admin pull-down menu in the upper right corner.
- In the management window, click Users under User + Group Management, and then click the admin username.
- Select Change Password and enter a new password. When you are finished, click the Go To Dashboard link on the left side of the window to return to the dashboard view.

Figure 9.1 Apache Ambari dashboard view of a Hadoop cluster

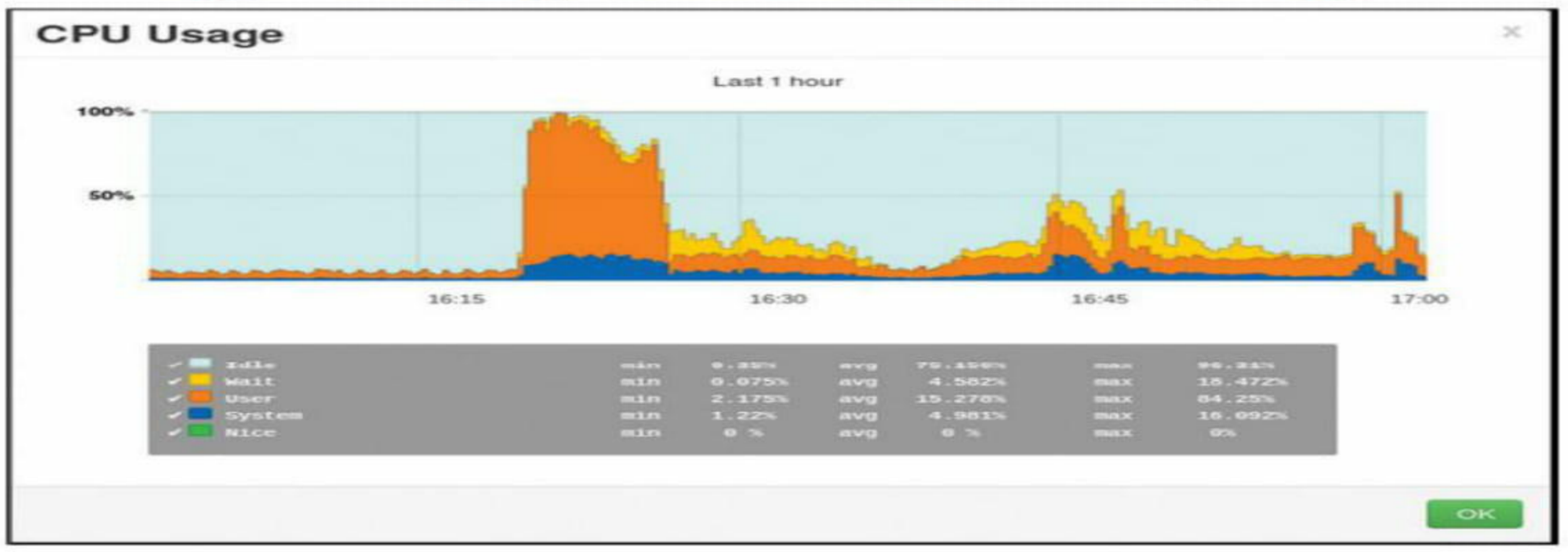


- To leave the Ambari interface, select the Admin pulldown menu at the left side of the main menu bar and click Sign out.
- The dashboard view provides a number of high-level metrics for many of the installed services. A glance at the dashboard should allow you to get a sense of how the cluster is performing.

Dashboard View

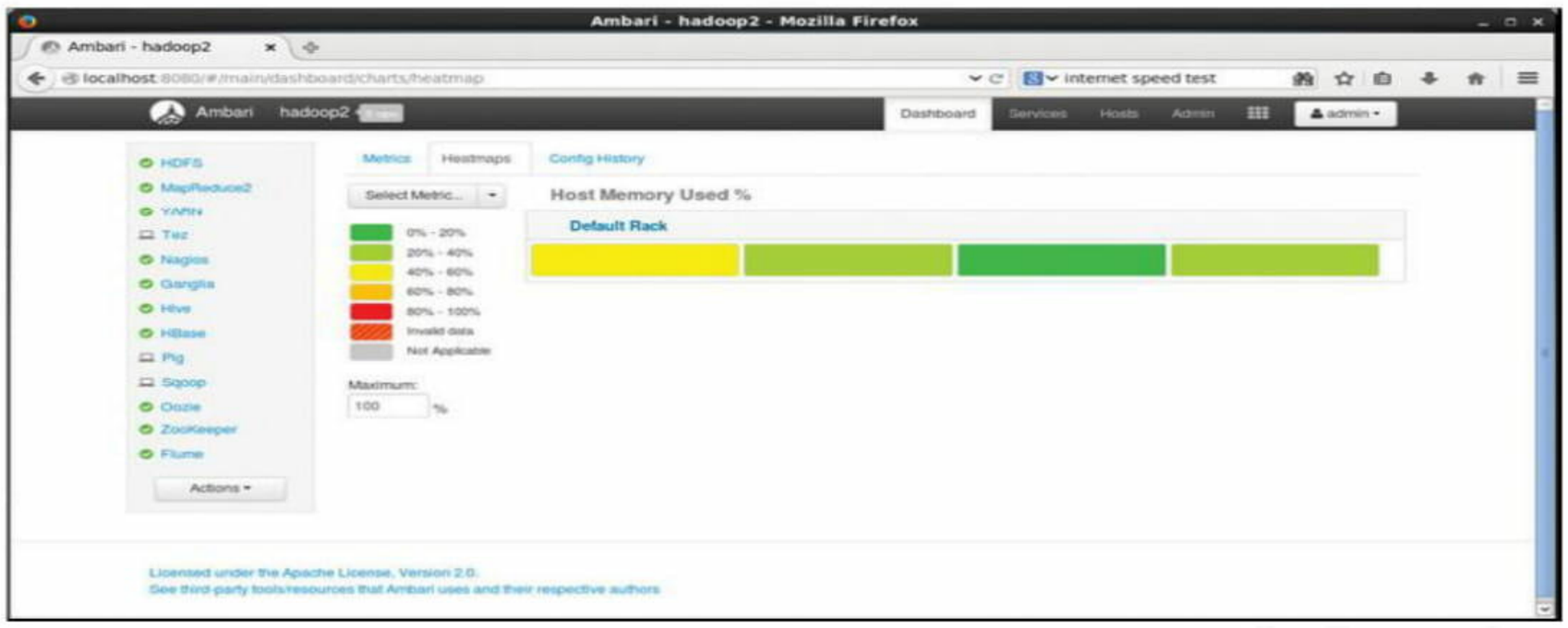
- The Dashboard view provides small status widgets for many of the services running on the cluster.
- The actual services are listed on the left-side vertical menu. You can move, edit, remove, or add these widgets as follows:
  - **Moving:** Click and hold a widget while it is moved about the grid.
  - **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.
- Some widgets provide additional information when you move the mouse over them. For instance, the DataNodes widget displays the number of live, dead, and decommissioning hosts.

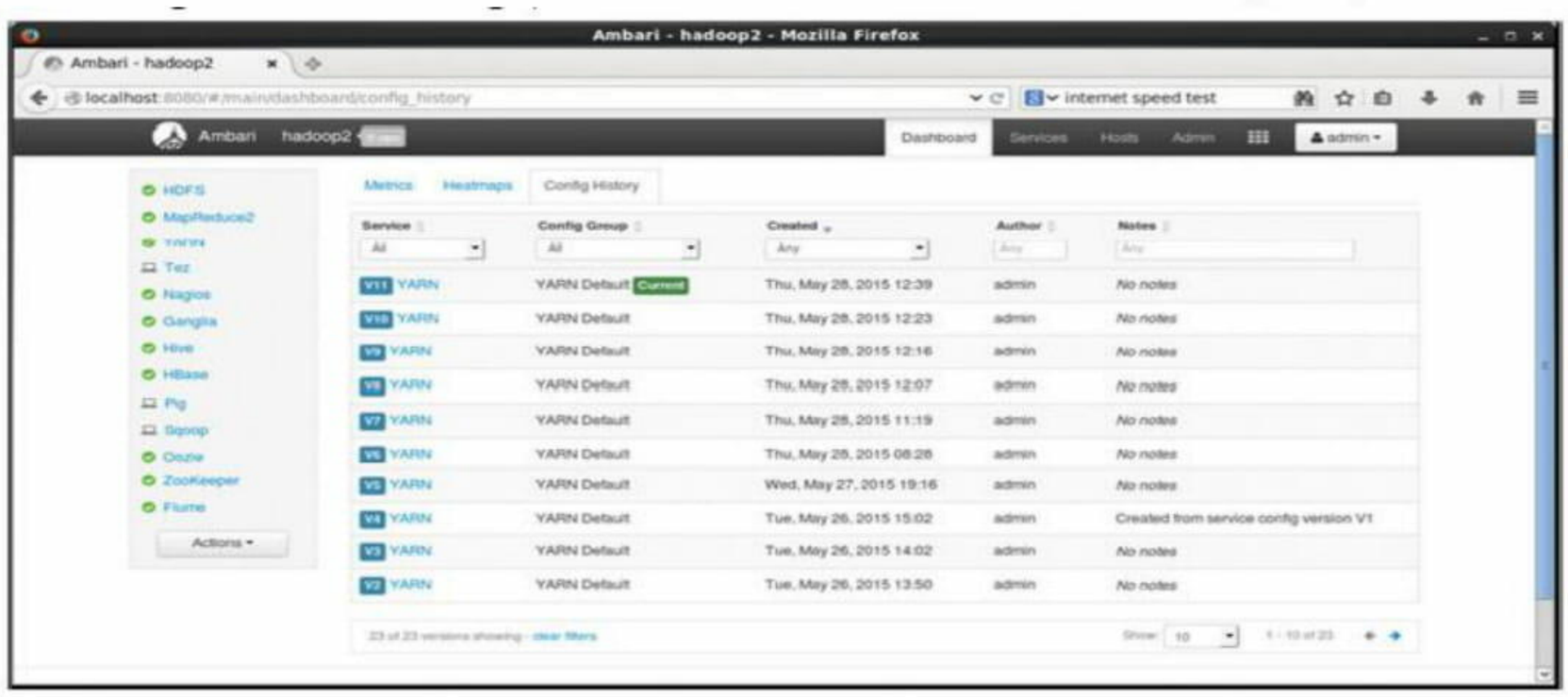
Figure 9.2 Enlarged view of Ambari CPU Usage widget



- The Dashboard view also includes a heatmap view of the cluster. Cluster heatmaps physically map selected metrics across the cluster.
- When you click the Heatmaps tab, a heatmap for the cluster will be displayed.
- To select the metric used for the heatmap, choose the desired option from the Select Metric pull-down menu.
- Note that the scale and color ranges are different for each metric.
- The heatmap for percentage host memory used is displayed in Figure 9.3.

The heatmap for percentage host memory used is displayed in Figure 9.3.





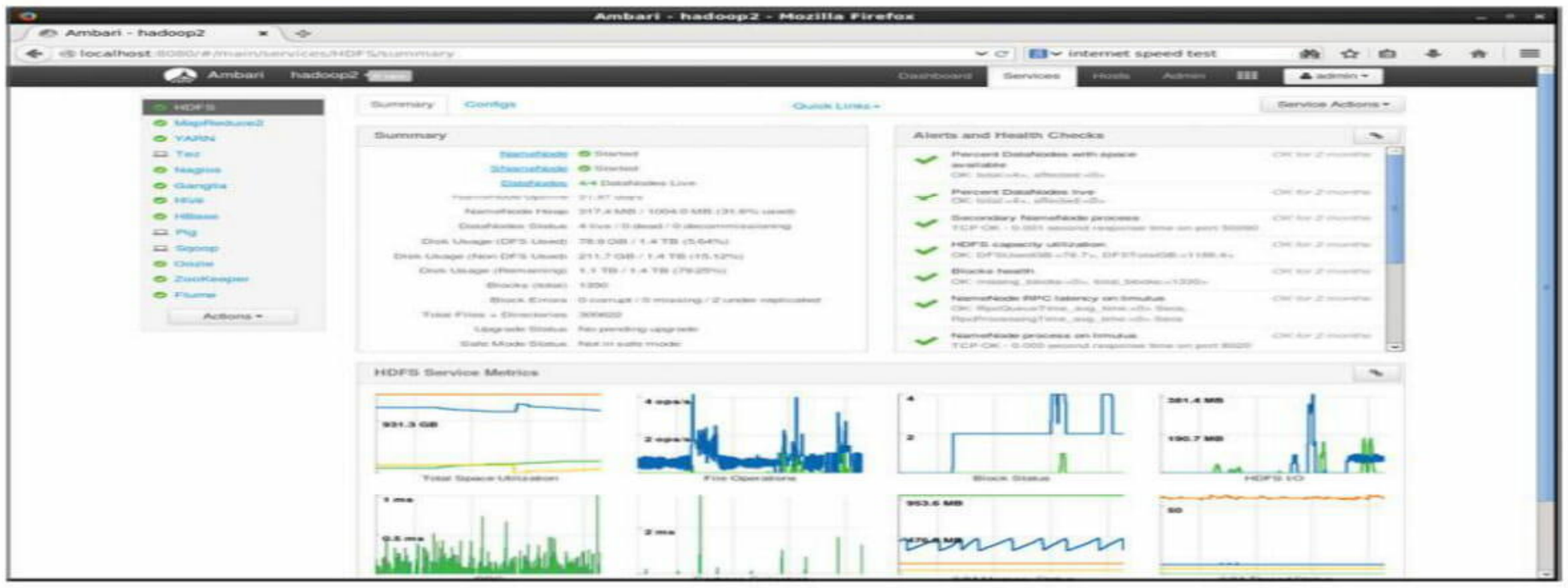
Configuration history is the final tab in the dashboard window. This view provides a list of configuration changes made to the cluster. As shown in Figure 9.4, Ambari enables configurations to be sorted by Service, Configuration Group, Data, and Author. To find the specific configuration settings, click the service name.58

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 hand-editing the /etc/hadoop/confXML 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 left-side 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.
- For example, Figure 9.5 shows the Service view for HDFS. Important information such as the status of NameNode, SecondaryNameNode, DataNodes, uptime, and available disk space is displayed in the Summary window.

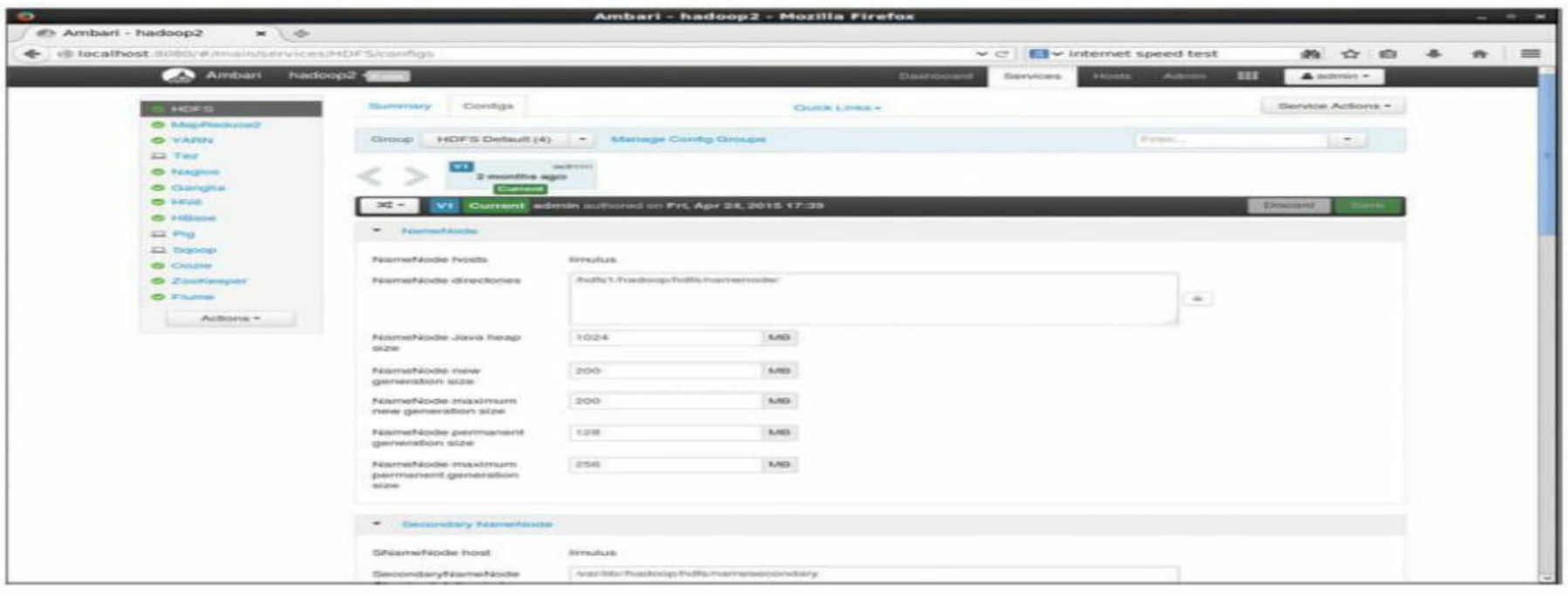
- The Alerts and Health Checks window provides the latest status of the service and its component systems.
- Finally, several important real-time service metrics are displayed as widgets at the bottom of the screen.
- As on the dashboard, these widgets can be expanded to display a more detailed view.

Figure 9.5 HDFS service summary window



- Clicking the Configs tab will open an options form, shown in Figure 9.6, for the service.
- The options (properties) are the same ones that are set in the Hadoop XML files.
- When using Ambari, the user has complete control over the XML files and should manage them only through the Ambari interface—that is, the user should not edit the files by hand.
- The current settings available for each service are shown in the form.
- The administrator can set each of these properties by changing the values in the form.
- Placing the mouse in the input box of the property displays a short description of each property.
- Where possible, properties are grouped by functionality.
- The form also has provisions for adding properties that are not listed.
- An example of changing service properties and restarting the service components is provided in the "Managing Hadoop Services" section.
- If a service provides its own graphical interface (e.g., HDFS, YARN, Oozie), then that interface can be opened in a separate browser tab by using the Quick Links pulldown menu located in top middle of the window.

Figure 9.6 Ambari service options for HDFS

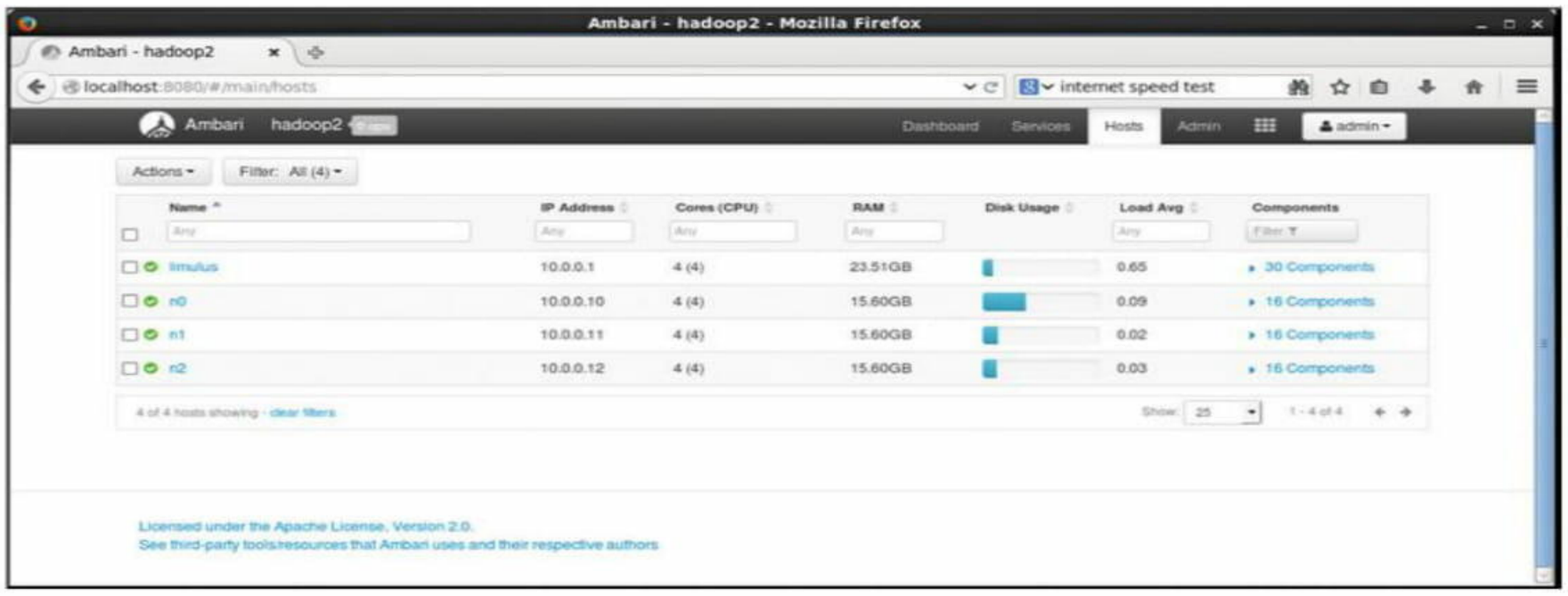


- Finally, the Service Action pull-down menu in the upperleft corner provides a method for starting and stopping each service and/or its component daemons across the cluster.
- Some services may have a set of unique actions (such as rebalancing HDFS) that apply to only certain situations.
- Finally, every service has a Service Check option to make sure the service is working property.
- The service check is initially run as part of the installation process and can be valuable when diagnosing problems.

Hosts View

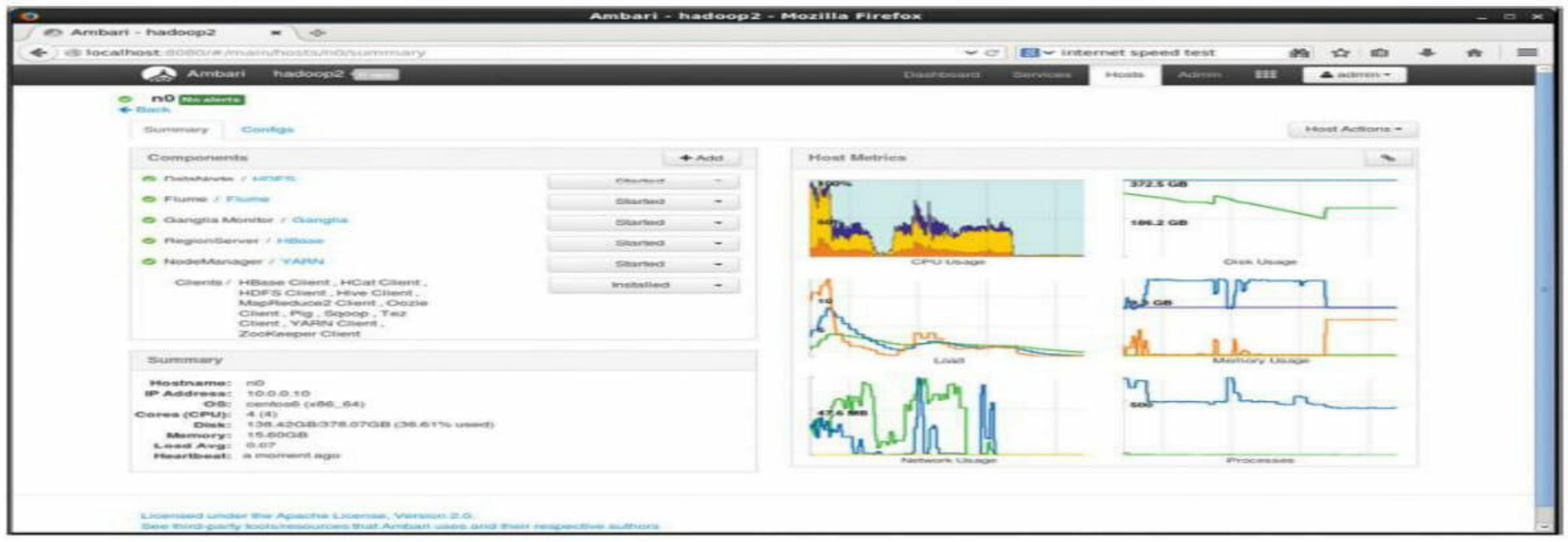
- Selecting the Hosts menu item provides the information shown in Figure 9.7.
- The host name, IP address, number of cores, memory, disk usage, current load average, and Hadoop components are listed in this window in tabular form.
- To display the Hadoop components installed on each host, click the links in the rightmost columns.
- You can also add new hosts by using the Actions pulldown menu. The new host must be running the Ambari agent (or the root SSH key must be entered) and have the base software.
- The remaining options in the Actions pull-down menu provide control over the various service components running on the hosts.

Figure 9.7 Ambari main Hosts screen



- Further details for a particular host can be found by clicking the host name in the left column. As shown in Figure 9.8, the individual host view provides three subwindows: Components, Host Metrics, and Summary information.
- The Components window lists the services that are currently running on the host. Each service can be stopped, restarted, decommissioned, or placed in maintenance mode.
- The Metrics window displays widgets that provide important metrics (e.g., CPU, memory, disk, and network usage).
- Clicking the widget displays a larger version of the graphic.
- The Summary window provides basic information about the host, including the last time a heartbeat was received.

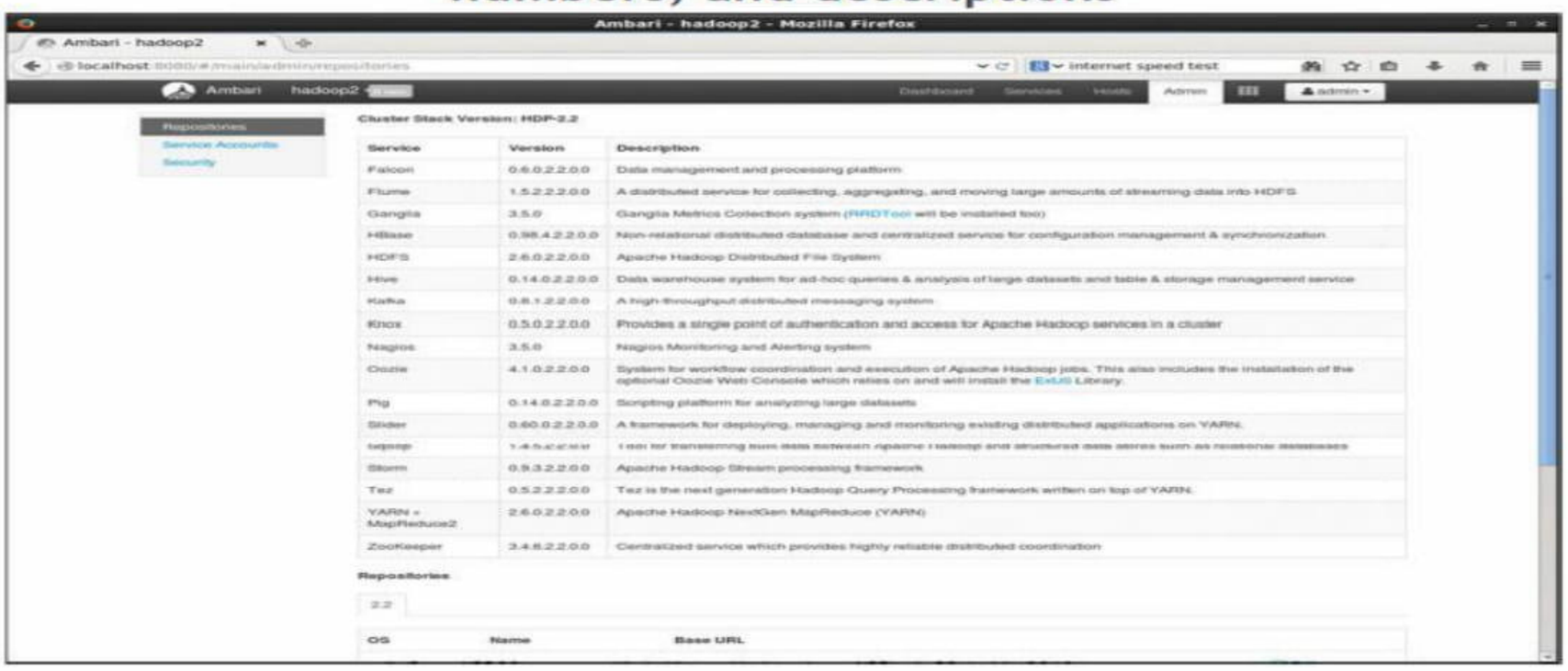
Figure 9.8 Ambari cluster host detail view



Admin View

- The Administration (Admin) view provides three options. The first, as shown in Figure 9.9, displays a list of installed software.
- This Repositories listing generally reflects the version of Horton works Data Platform (HDP) used during the installation process.
- 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.

Figure 9.9 Ambari installed packages with versions, numbers, and descriptions



Views View

- Ambari Views is a framework offering a systematic way to plug in user interface capabilities that provide for custom visualization, management, and monitoring features in Ambari.
- Views allows you to extend and customize Ambari to meet your specific needs.

Admin Pull-Down Menu

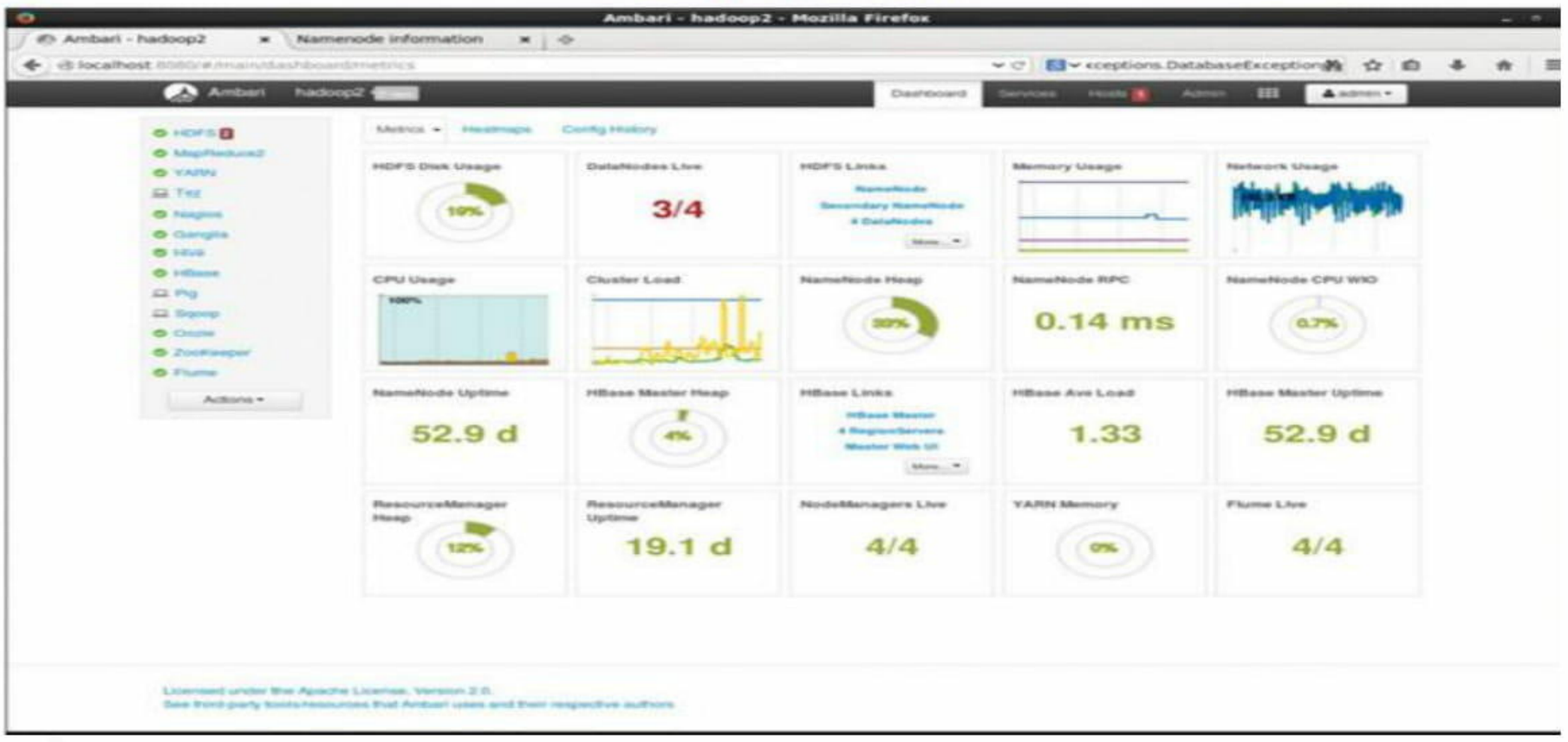
The Administrative (Admin) pull-down menu provides the following options:

- **About**—Provides the current version of Ambari.
- **Manage Ambari**—Open the management screen where Users, Groups, Permissions, and Ambari Views can be created and configured.
- **Settings**—Provides the option to turn off the progress window. (See Figure 9.15.)
- **Sign Out**—Exits the interface.—Exits the interface.

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.
- The service error indicator numbers next to the HDFS service and Hosts menu item indicate this condition.
- The DataNode widget also has turned red and indicates that 3/4 DataNodes are operating.

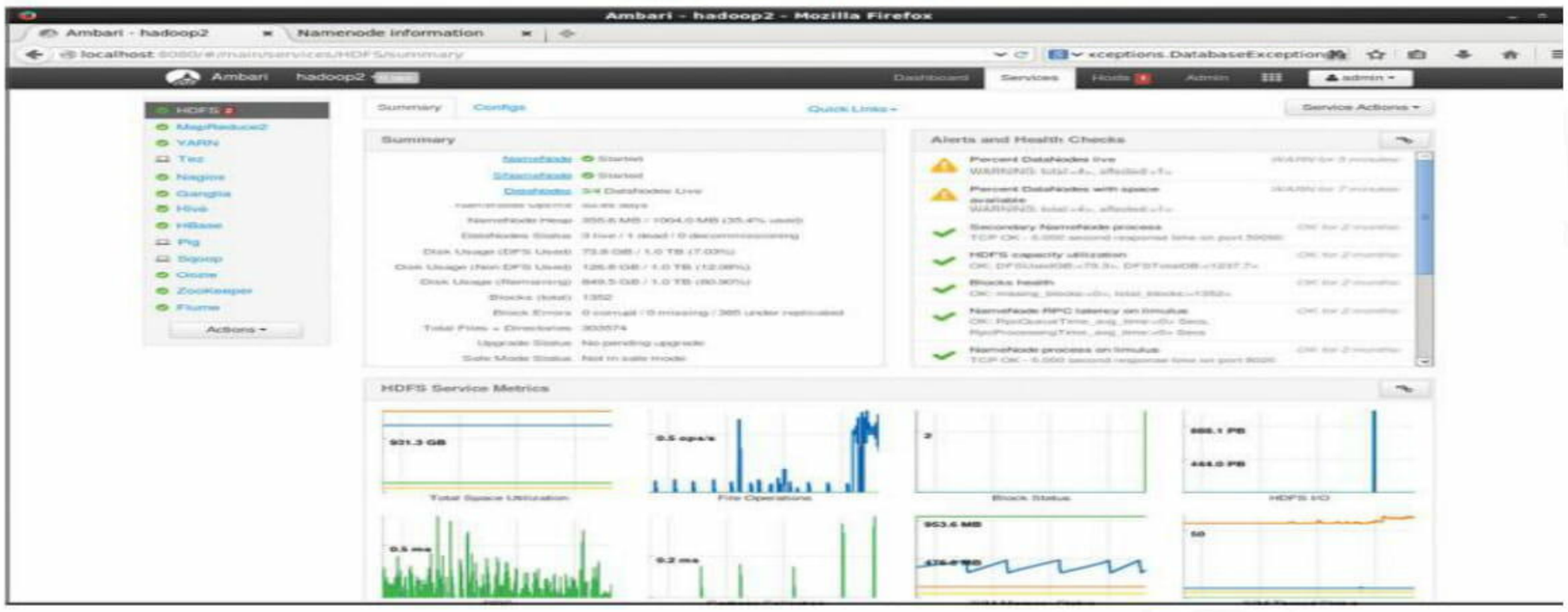
Figure 9.10 Ambari main dashboard indicating a DataNode issue



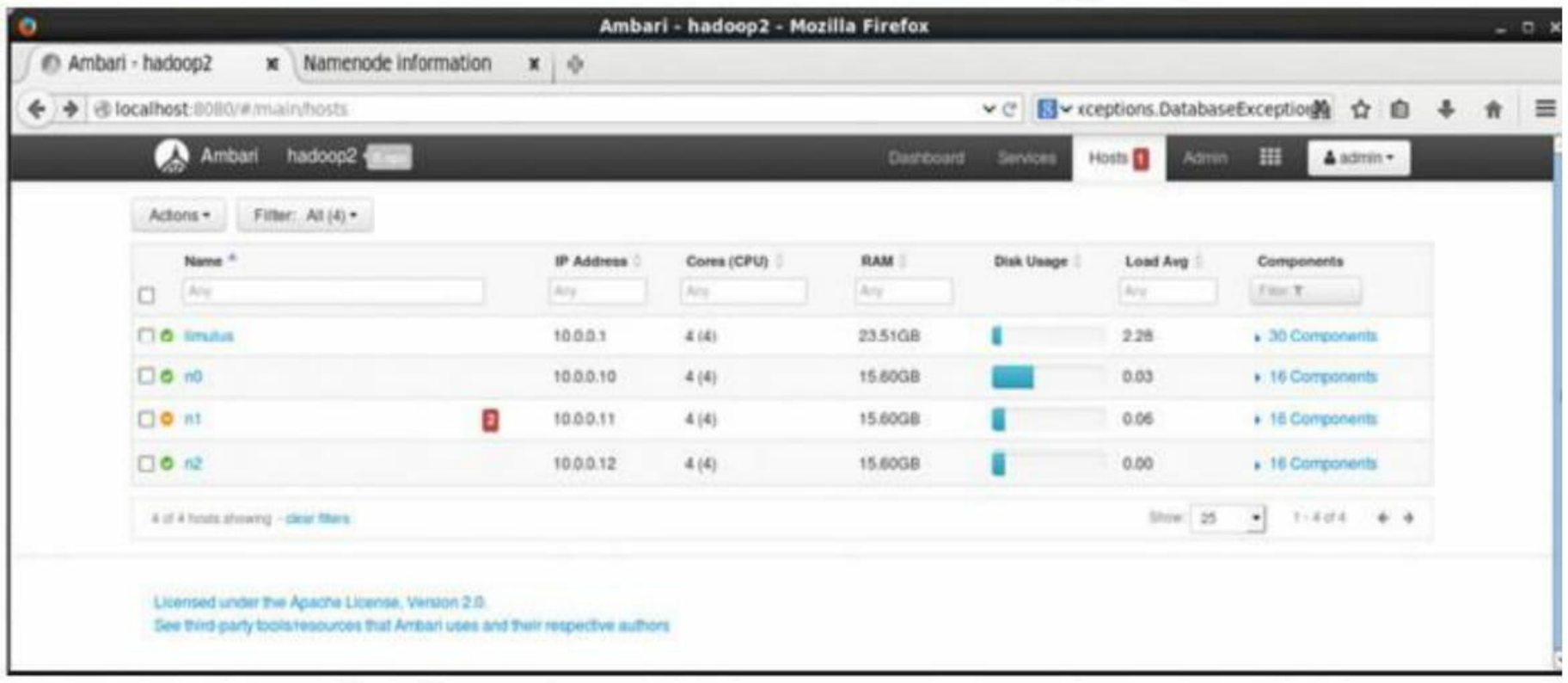
The specific host (or hosts) with an issue can be found by examining the Hosts window. As shown in Figure 9.12, the status of host n1 has changed from a green dot with a check mark inside to a yellow dot with a dash inside.

- An orange dot with a question mark inside indicates the host is not responding and is probably down.
- Other service interruption indicators may also be set as a result of the

Figure 9.11 Ambari HDFS service summary window indicating a down DataNode



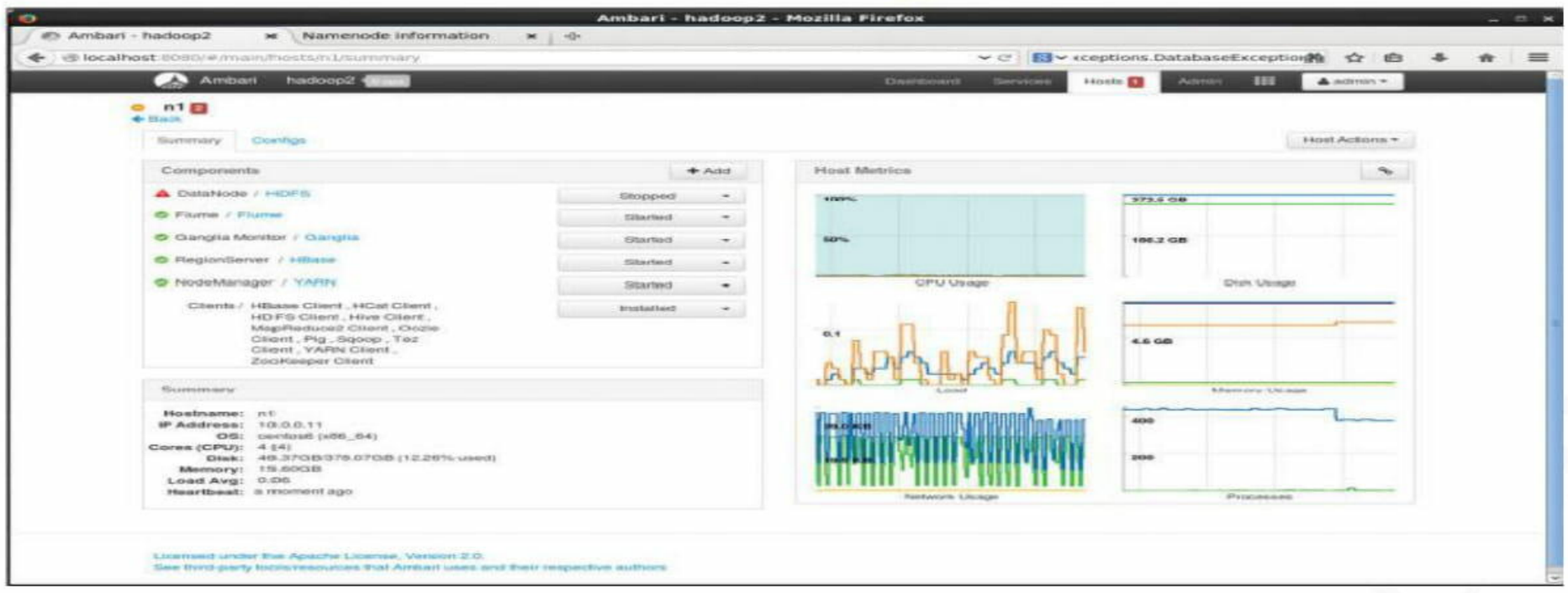
unresponsive node.



Clicking on the n1 host link opens the view in Figure 9.13.

- Inspecting the Components sub-window reveals that the DataNode daemon has stopped on the host.
- At this point, checking the DataNode logs on host n1 will help identify the actual cause of the failure.
- Assuming the failure is resolved, the DataNode daemon can be started using the Start option in the pull-down menu next to the service name.

Figure 9.13 Ambari window for host n1 indicating the DataNode/HDFS service has stopped



When the DataNode daemon is restarted, a confirmation similar to Figure 9.14 is required from the user.

- When a service daemon is started or stopped, a progress window similar to Figure 9.15 is opened.
- The progress bar indicates the status of each action.
- Note that previous actions are part of this window.
- If something goes wrong during the action, the progress bar will turn red.
- If the system generates a warning about the action, the process bar will turn orange.

Figure 9.15 Ambari progress window for DataNode restart

When these background operations are running, the small ops (operations) bubble on the top menu bar will indicate how many operations are running. (If different service daemons are started or stopped, each process will be run to completion before the next one starts.)

- Once the DataNode has been restarted successfully, the dashboard will reflect the new status (e.g., 4/4 DataNodes are Live).
- As shown in Figure 9.16, all four DataNodes are now working and the service error indicators are beginning to slowly disappear.

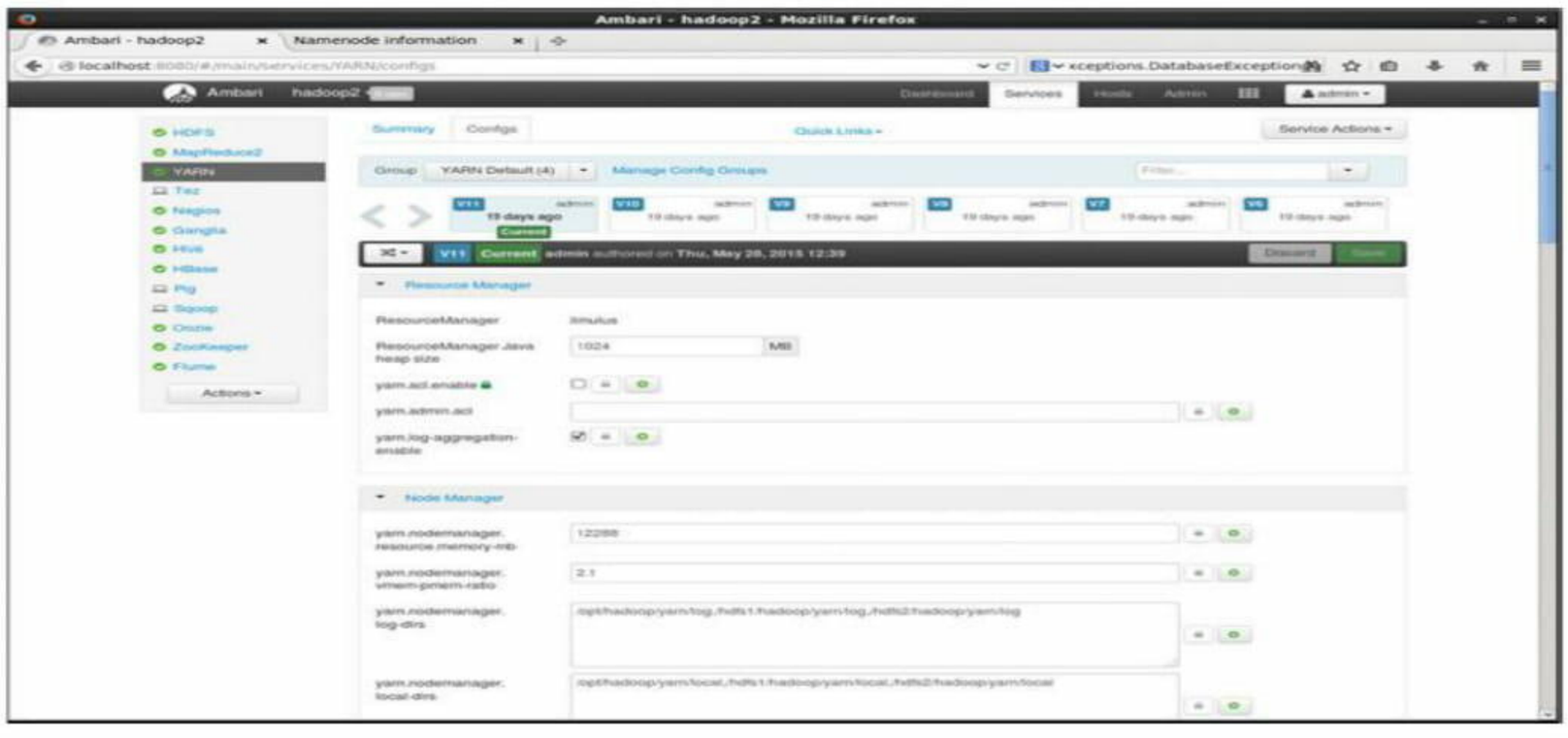
- The service error indicators may lag behind the real time widget updates for several minutes.

Figure 9.16 Ambari dashboard indicating all DataNodes are running

HANGING HADOOP PROPERTIES

- One of the challenges of managing a Hadoop cluster is managing changes to cluster-wide 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 9.17.

Figure 9.17 Ambari YARN properties view



To easily view the application logs, this property must be set to true.

- This property is normally on by default.

- As an example, for our purposes here, we will use the Ambari interface to disable this feature.
- As shown in Figure 9.18, when a property is changed, the green Save button becomes activated.

Basic Hadoop Administration Procedures

Adding Users to HDFS

To quickly create user accounts manually on a Linux-based system, perform the following steps:

1. 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>
```

2. Create the username directory in HDFS.

```
hdfs dfs -mkdir /user/<username>
```

3. Give that account ownership over its directory in HDFS.

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

Perform an FSCK on HDFS

To check the health of HDFS, you can issue the hdfs fsck <path> (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.
- -includeSnapshots includes snapshot data. The path indicates the existence of a snapshottable directory or the presence of snapshottabledirectories under it.
- -list-corruptfileblocks prints out a list of missing blocks andthe files to which they belong.
- -blocks prints out a block report.
- -locations prints out locations for every block.
- -racks prints out network topology for data-node locations.

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.

The HDFS superuser must run the balancer. The simplest way to run the balancer is to enter the following command:

```
$ hdfs balancer
```

- By default, the balancer will continue to rebalance the nodes until the number of data blocks on all DataNodes are within 10% of each other.
- The balancer can be stopped, without harming HDFS, at any time by entering a Ctrl-C.
- Lower or higher-thresholds can be set using the -threshold argument. For example, giving the following command sets a 5% threshold:

```
$ hdfs balancer -threshold 5
```

- The lower the threshold, the longer the balancer will run.
- To ensure the balancer does not swamp the cluster networks, you can set a bandwidth limit before running the balancer, as follows:

```
$ dfsadmin -setBalancerBandwidth newbandwidth
```

The newbandwidth option is the maximum amount of network bandwidth, in bytes per second, that each DataNode can use during the balancing operation.

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 (e.g., a full DataNode). 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
```

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.

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-1.

```
$ hdfs dfs -createSnapshot /user/hdfs/war-and-peace-input wapi-snap-1
Created snapshot /user/hdfs/war-and-peace-input/.snapshot/wapi-snap-1
```

The path of the snapshot is /user/hdfs/war-and-peaceinput/.snapshot/wapi-snap-1. The /user/hdfs/war-andpeace-input directory has one file, as shown by issuing the followingcommand:

```
$ hdfs dfs -ls /user/hdfs/war-and-peace-input/
Found 1 items
-rw-r--r-- 2 hdfs hdfs 3288746 2015-06-24 19:56 /user/hdfs/warand-
peaceinput/war-and-peace.txt
```

If the file is deleted, it can be restored from the snapshot:

```
$ hdfs dfs -rm -skipTrash /user/hdfs/war-and-peace-input/war-andpeace.
txt
Deleted /user/hdfs/war-and-peace-input/war-and-peace.txt
$ hdfs dfs -ls /user/hdfs/war-and-peace-input/
```

The restoration process is basically a simple copy from the snapshot to the previous directory (or anywhere else). Note the use of the ~/.snapshot/wapi-snap-1 path to restore the file:

```
$ hdfs dfs -cp /user/hdfs/war-and-peace-input/.snapshot/wapi-snap-1/war-
and-peace.txt /user/hdfs/war-and-peace-input
```

Confirmation that the file has been restored can be obtained by issuing the following command:

```
$ hdfs dfs -ls /user/hdfs/war-and-peace-input/
Found 1 items
-rw-r--r-- 2 hdfs hdfs 3288746 2015-06-24 21:12 /user/hdfs/warand-
peace-input/war-and-peace.txt
```

References:

- Douglas Eadline,**"Hadoop 2 Quick-Start Guide: Learn the Essentials of Big Data Computing in the Apache Hadoop 2 Ecosystem",** 1stEdition, Pearson Education, 2016. ISBN-13: 978-9332570351