

Designing and Building Big Data Applications



Introduction

Chapter 1



Course Chapters

- **Introduction**
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Chapter Topics

Introduction

- **About This Course**
- About Cloudera
- Course Logistics
- Introductions

Course Objectives (1)

After successfully completing this course, you will be able to:

- Determine which Hadoop-related tools are appropriate for specific tasks
- Understand how file formats, serialization, and data compression affect application compatibility and performance
- Design and evolve schemas in Apache Avro
- Create, populate, and access data sets with the Kite SDK
- Integrate with external systems using Apache Sqoop and Apache Flume
- Integrate Apache Flume with existing applications and develop custom components to extend Flume's capabilities
- Create, package, and deploy Oozie jobs to manage processing workflows
- Develop Java-based data processing pipelines with Apache Crunch

Course Objectives (2)

- **Implement user-defined functions for use in Apache Hive and Impala**
- **Index both static and streaming data sets with Cloudera Search**
- **Use Hue to build a Web-based interface for Search queries**
- **Integrate results from Impala and Cloudera Search into your applications**

Chapter Topics

Introduction

- About This Course
- **About Cloudera**
- Course Logistics
- Introductions

About Cloudera (1)



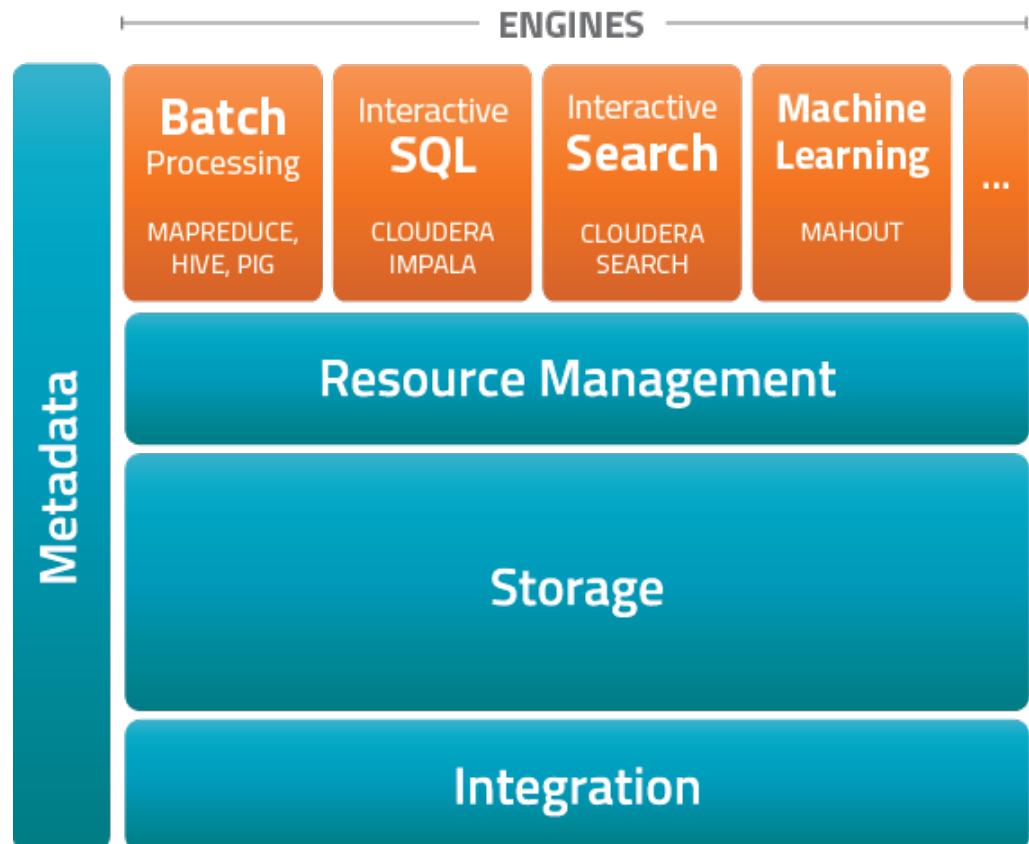
- The leader in Apache Hadoop-based software and services
- Founded by leading experts on Hadoop from Facebook, Yahoo, Google, and Oracle
- Provides support, consulting, training, and certification for Hadoop users
- Staff includes committers to virtually all Hadoop projects
- Many authors of industry-standard books on Apache Hadoop projects

About Cloudera (2)

- **Customers include many key users of Hadoop**
 - Allstate, AOL Advertising, Box, CBS Interactive, eBay, Experian, Groupon, National Cancer Institute, Orbitz, Social Security Administration, Trend Micro, Trulia, US Army, ...
- **Cloudera public training:**
 - Cloudera Developer Training for Apache Hadoop
 - Designing and Building Big Data Applications
 - Cloudera Administrator Training for Apache Hadoop
 - Cloudera Data Analyst Training: Using Pig, Hive, and Impala with Hadoop
 - Cloudera Training for Apache HBase
 - Introduction to Data Science: Building Recommender Systems
 - Cloudera Essentials for Apache Hadoop
- **Onsite and custom training is also available**

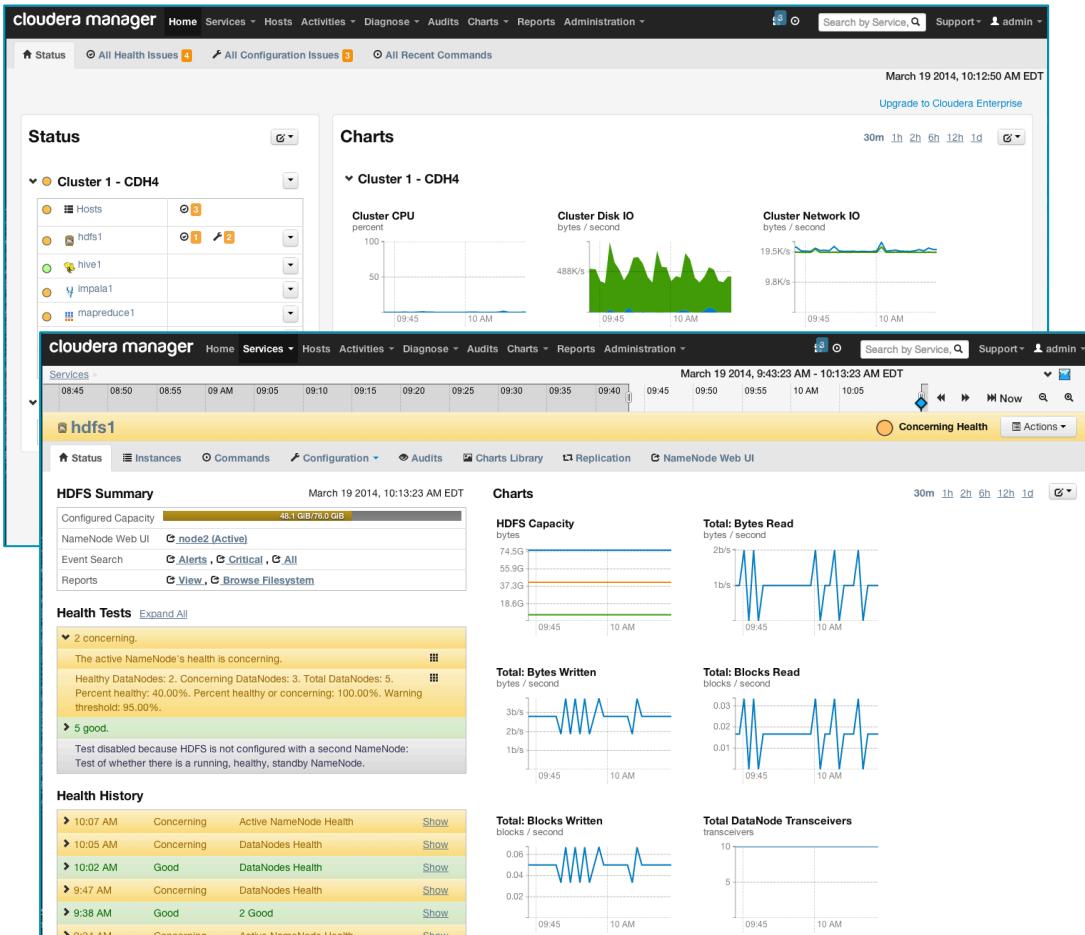
- **CDH (Cloudera's Distribution, including Apache Hadoop)**

- 100% open source, enterprise-ready distribution of Hadoop and related projects
- The most complete, tested, and widely-deployed distribution of Hadoop
- Integrates all the key Hadoop ecosystem projects
- Available as RPMs and Ubuntu/Debian/SuSE packages or as a tarball



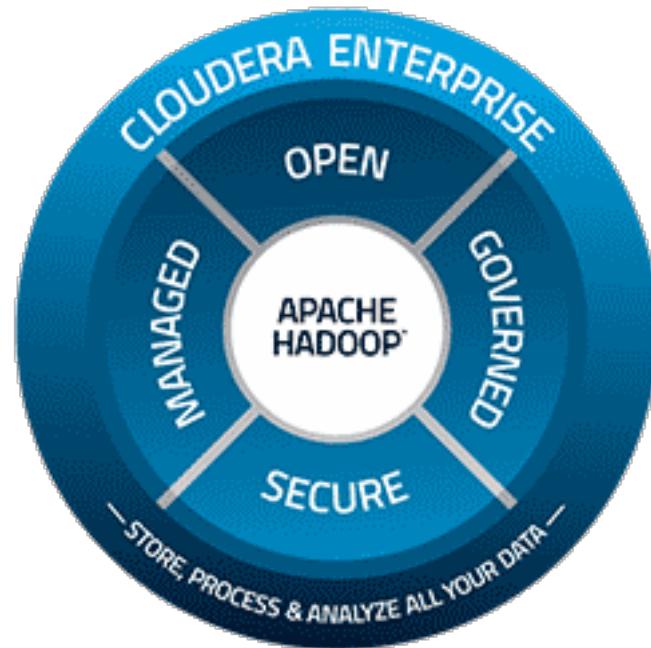
Cloudera Express

- **Cloudera Express**
 - Free download
- **The best way to get started with Hadoop**
- **Includes CDH**
- **Includes Cloudera Manager**
 - End-to-end administration for Hadoop
 - Deploy, manage, and monitor your cluster



Cloudera Enterprise

- **Cloudera Enterprise**
 - Subscription product including CDH and Cloudera Manager
- **Includes support**
- **Includes extra Cloudera Manager features**
 - Configuration history and rollbacks
 - Rolling updates
 - LDAP integration
 - SNMP support
 - Automated disaster recovery
 - ...and more



Chapter Topics

Introduction

- About This Course
- About Cloudera
- **Course Logistics**
- Introductions

Logistics

- Class start and finish times
- Lunch
- Breaks
- Restrooms
- Wi-Fi access
- Virtual machines
- Can I come in early/stay late?

Your instructor will give you details on how to access the course materials and exercise instructions for the class

Chapter Topics

Introduction

- About This Course
- About Cloudera
- Course Logistics
- **Introductions**

Introductions

- **About your instructor**
- **About you**
 - Where do you work? What do you do there?
 - What programming languages have you used?
 - How much Hadoop experience do you have?
 - What do you expect to gain from this course?

Application Architecture

Chapter 2



Course Chapters

- Introduction
- **Application Architecture**
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Application Architecture

In this chapter you will learn

- How the Hands-On Exercises relate to a working Big Data application
- Which tools we will use to develop, test, and build code
- What are some important factors to consider when choosing how to ingest, process, and present data

Chapter Topics

Application Architecture

- **Scenario Explanation**
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Scenario Explanation (1)

- **Hands-On Exercises let you practice what you have learned**
- **These exercises are based on a hypothetical scenario**
 - However, the concepts apply to nearly any organization
- **Loudacre Mobile is a fast-growing wireless carrier**
 - Founded in 2008 and headquartered in Palo Alto, California
 - Now provides service to 130,000 customers throughout the western USA
 - Ranked number one in customer support five years in a row



Scenario Explanation (2)

- **Unfortunately, customer satisfaction with support has plummeted**
 - Recent spike in growth outpaced our ability to hire and train staff
 - New representatives had little technical experience
 - Experienced representatives are too busy to train new hires

Scenario Explanation (3)

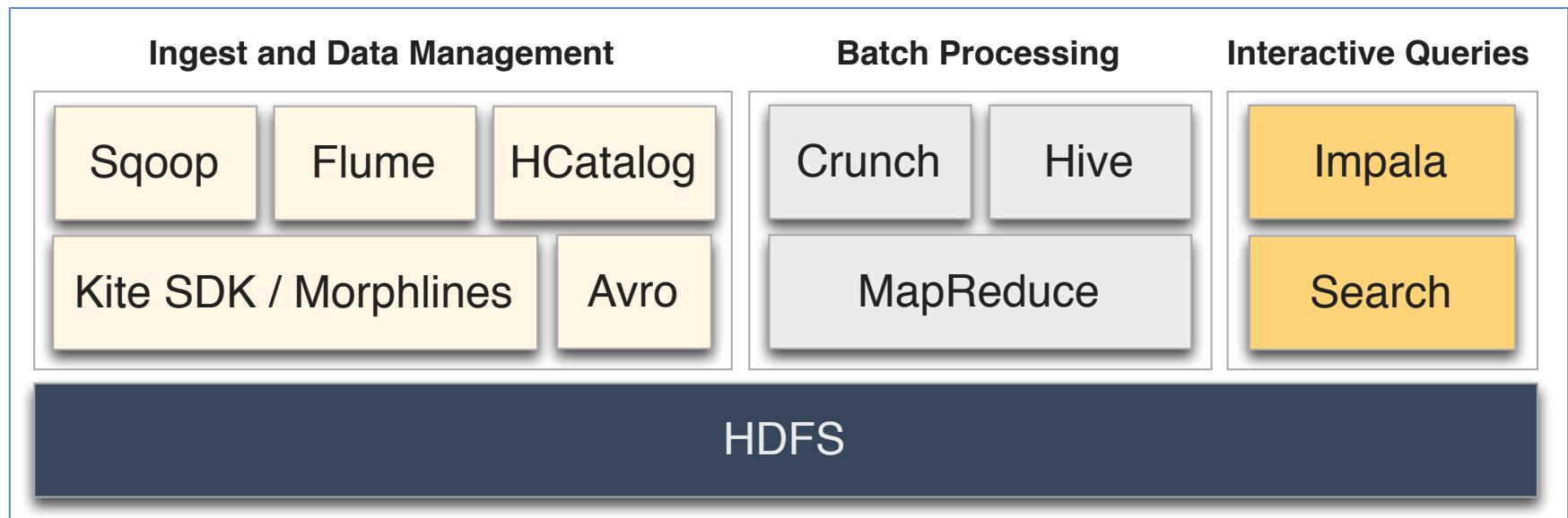
- **Average support call lasts 63% longer than last year**
 - Also 47% less likely to be resolved during the first call
- **This costs Loudacre millions of dollars per month**
 - Support center costs have risen dramatically
 - Frustrated customers are closing their accounts
 - Few new customers are signing up
- **Loudacre's CEO is counting on you to fix the problem**

Scenario Explanation (4)

- You will develop a revolutionary new support application
- This application will improve the accuracy and efficiency of the support team
 - Brings together data from disparate sources
 - Processes data to reveal important patterns
 - Presents results in a Web-based dashboard
- The Hands-On Exercises represent steps in developing the prototype
- We recommend that all “Big Data” applications start with a prototype
 - Start with a small amount of data
 - Ramp up in subsequent iterations
 - Automate the build and testing process
- This approach helps you to uncover problems earlier

Tools Used in Class

- During this course, you will use several tools from the Enterprise Data Hub
- All interact with data in HDFS and can be loosely divided into three groups
 - Some are used to load and manage data
 - Others are used for batch processing of large data sets
 - Still others are fast enough to power interactive applications



Chapter Topics

Application Architecture

- Scenario Explanation
- **Instructor-led Demo: Loudacre Mobile Support Portal**
- Understanding the Development Environment
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Demo: The Loudacre Mobile Support Portal

- Your instructor will now demonstrate the Loudacre Mobile Support Portal
 - You will work on building this application throughout the course

Chapter Topics

Application Architecture

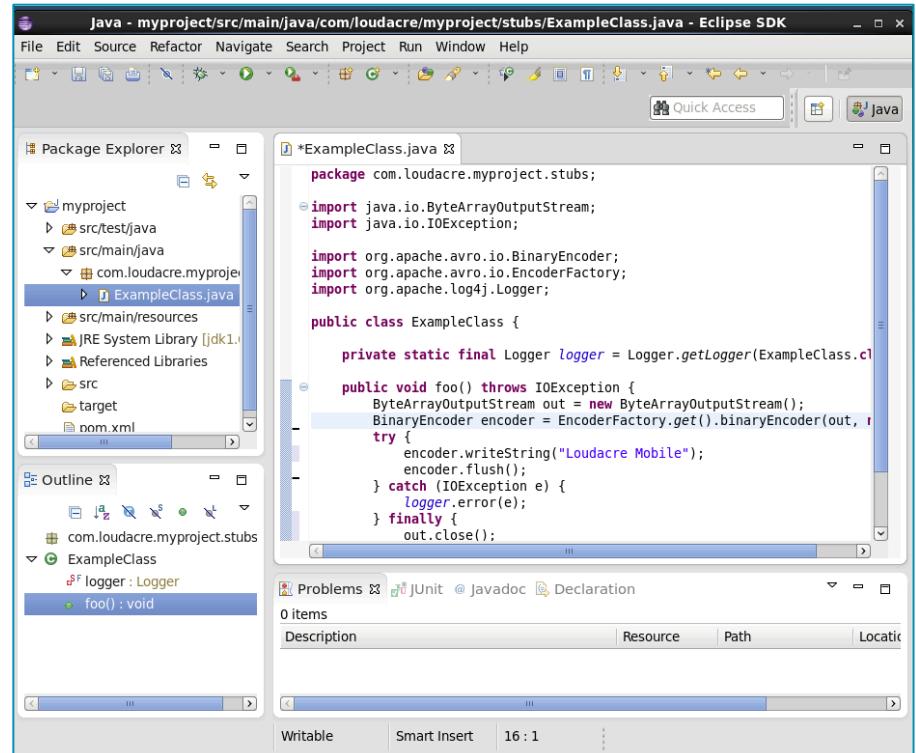
- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- **Understanding the Development Environment**
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Development Environment Overview

- Our virtual machine (VM) simulates a development cluster
 - VM hostname is dev.loudacre.com
 - CDH is installed in *pseudo-distributed mode*
 - A Hadoop cluster on a single machine
- Course uses the same tools engineers typically use for development
 - IDE (Eclipse)
 - Unit testing library (JUnit)
 - Build management tool (Maven)

Eclipse IDE

- Many Java Integrated Development Environments (IDEs) are available
 - Graphical tools for editing, compiling, testing, and running Java code
- Eclipse is one such IDE
 - Open source
 - Popular among Java developers
- Hands-On Exercises support Eclipse
 - Eclipse is installed on the VM
 - Hands-On Exercise Manual has instructions for using Eclipse if you are not familiar with it



JUnit Unit Testing Library

- **JUnit is a mature open source unit testing library**

- Tests should *assert* expected behavior of small code “units”
- Each typically tests a method under various conditions

- **Provides many assertions, e.g.,**

- assertEquals
- assertNull
- assertFalse

- **JUnit 4.x uses method annotation**

- Two most common are
 - @Before (pre-test setup)
 - @Test (marks something as a test)

The screenshot shows an IDE interface with two main panes. The top pane displays the Java code for a test class, `CalculatorTest.java`. It includes imports for `org.junit.Assert.*`, a private field `Calculator calc`, and two test methods: `verifyAddPositiveNumbers()` and `verifyMultiplyTimesZero()`. Both tests use the `assertEquals` assertion. The bottom pane shows the execution results in the JUnit view. It indicates the test completed "Finished after 0.082 seconds" with "Runs: 2/2", "Errors: 0", and "Failures: 0". A failure trace is shown for the first test, `verifyAddPositiveNumbers`, which took 0.070 seconds. The failure trace details the test method call.

```
CalculatorTest.java
import static org.junit.Assert.*;
public class CalculatorTest {
    private Calculator calc;
    @Before
    public void setUp() {
        calc = new Calculator();
    }
    @Test
    public void verifyAddPositiveNumbers() {
        assertEquals(5, calc.add(2, 3));
    }
    @Test
    public void verifyMultiplyTimesZero() {
        assertEquals(0, calc.multiply(8, 0));
    }
}
```

Problems JUnit @ Javadoc Declaration

Finished after 0.082 seconds

Runs: 2/2 Errors: 0 Failures: 0

Failure Trace

verifyAddPositiveNumbers (0.070 s)

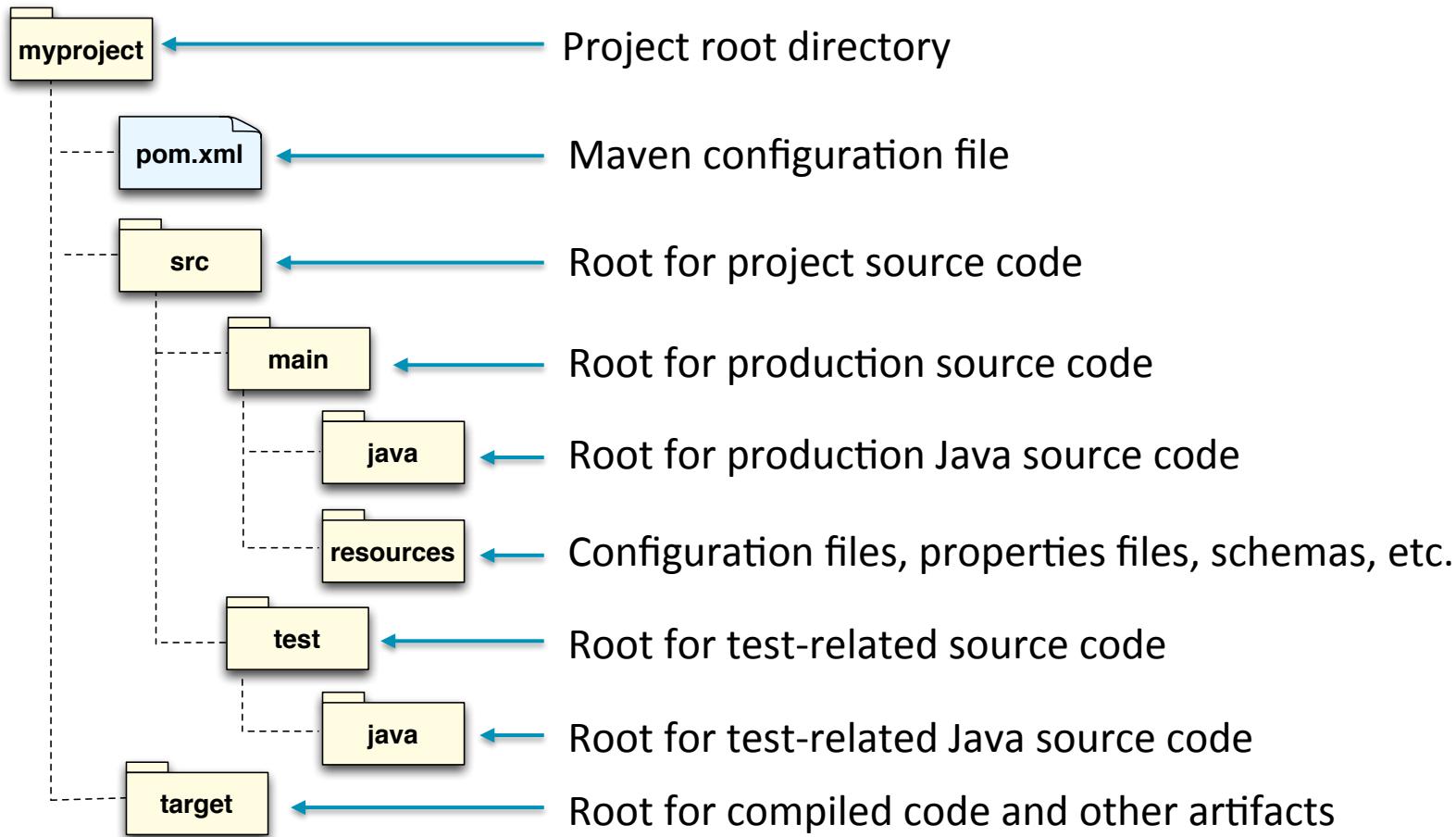
verifyMultiplyTimesZero (0.000 s)

What is Apache Maven?

- **Maven is a popular open source build tool**
 - Somewhat similar to Ant or Make
 - Can compile, test, execute, and package code for your project
 - Typically invoked from the command-line
- **Maven handles library dependency management**
 - Uses a declarative configuration in `pom.xml` file
 - Maven will find and download needed libraries from a *repository*
 - The VM we use is configured with a local repository
 - Contains all libraries needed during the course
 - Does not require Internet access
- **Maven can also create pre-configured Eclipse projects**
 - You will do this during the Hands-On Exercises

Maven Project Layout

- Standard Java project layout is based on “convention over configuration”



Maven Goals

- Build tasks are accomplished by invoking one or more “goals”
- Goals are specified as arguments to the `mvn` command
 - The `compile` goal will compile the project’s source code

```
$ mvn compile
```

- You may specify multiple goals at once
 - The following will compile the project’s source and run its unit tests

```
$ mvn compile test
```

- The `package` goal creates a JAR after compiling and testing the code

```
$ mvn package
```

Creating Eclipse Projects with Maven (1)

- **Maven already knows everything about your project**
 - Location of source code, unit tests, libraries for classpath, etc.
 - These are same details found in an IDE's project configuration
- **To avoid classpath problems, first add your Maven repository to Eclipse**
 - This is a one-time step (per machine, not per project)
 - Must restart Eclipse for this change to take effect

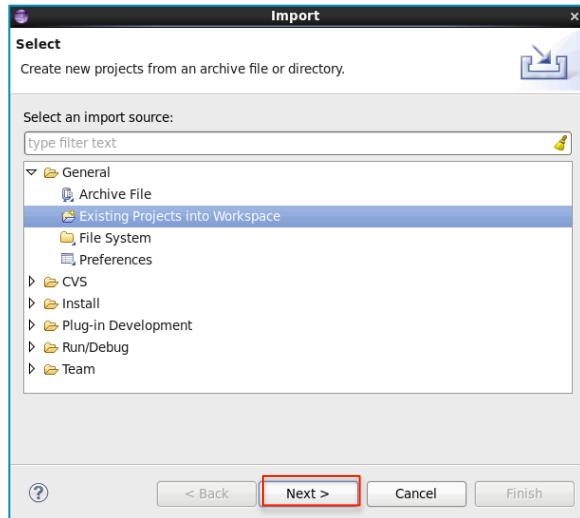
```
$ mvn -Dworkspace=/home/training/workspace/ \
eclipse:add-maven-repo
```

- **From within your project's directory, create the Eclipse project**

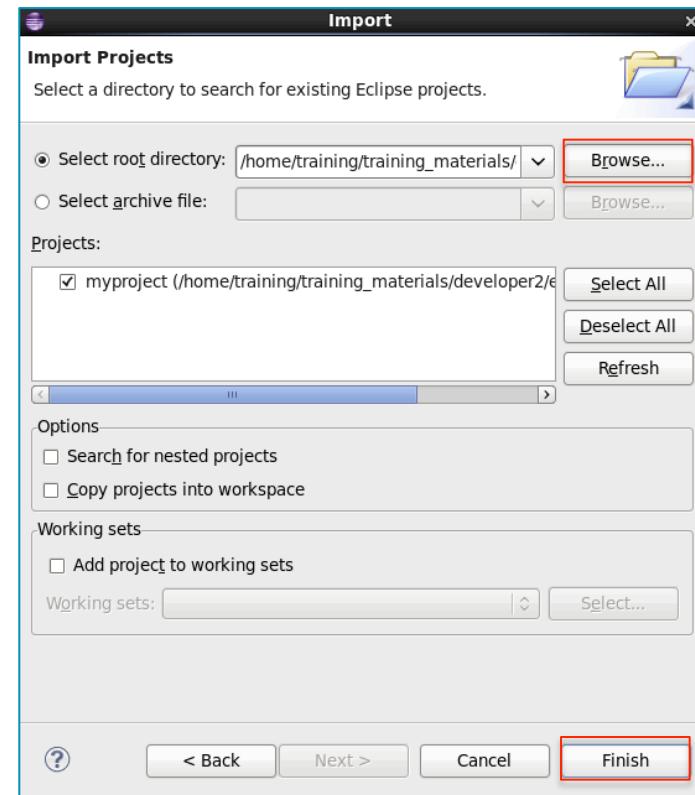
```
$ cd ~/training_materials/bigdata/exercises/myproject
$ mvn eclipse:eclipse
```

Creating Eclipse Projects with Maven (2)

- You may now import this directory as a project into your workspace
 - Start the import wizard by clicking the File menu and then selecting 'Import'



Wizard Page 1



Wizard Page 2

Chapter Topics

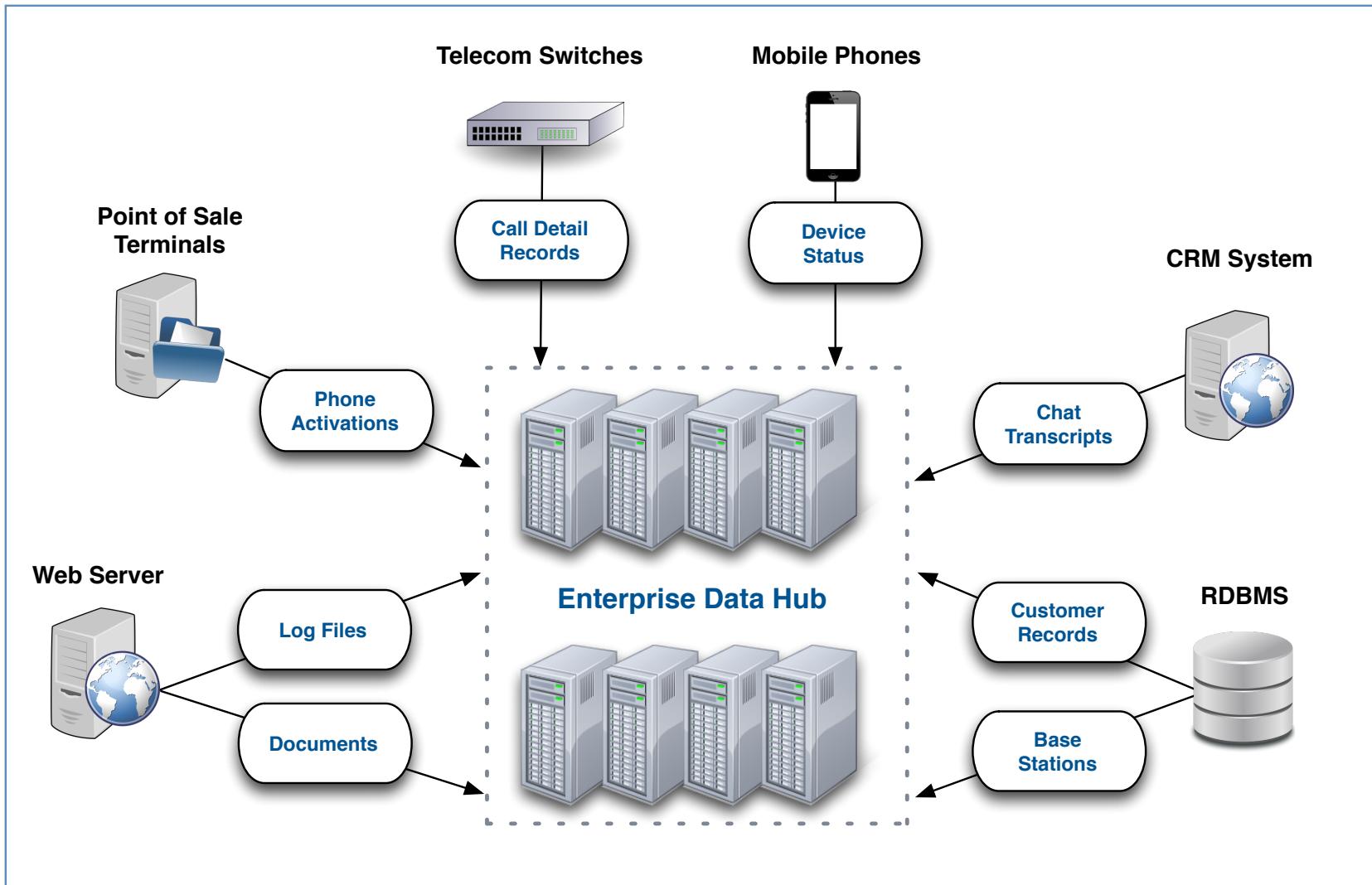
Application Architecture

- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- **Identifying and Collecting Input Data**
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Identifying Sources of Input Data

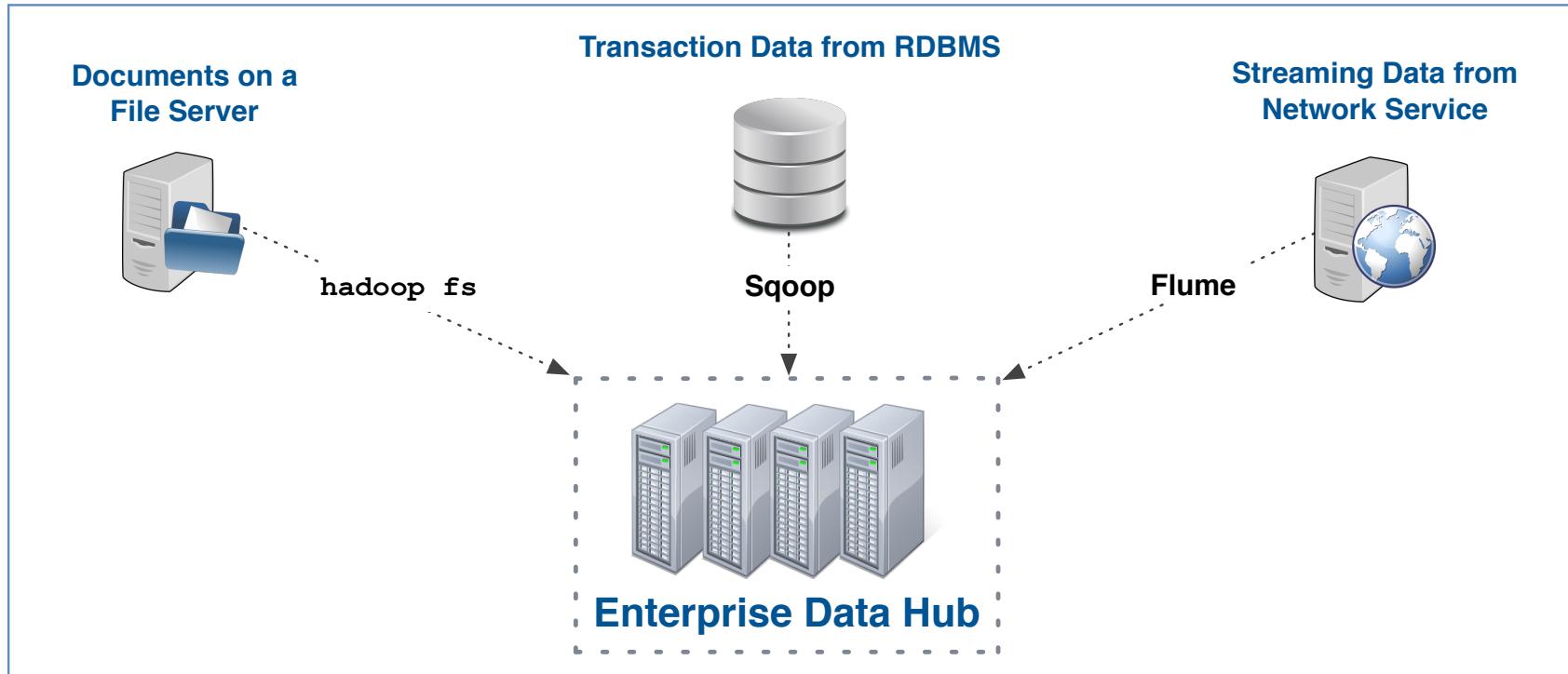
- **Loudacre's support agents require lots of data to solve problems**
 - Gathering it is currently a difficult and tedious manual process
 - Our application relieves support agents of this task
- **Our data comes from a variety of “siloed” systems, and includes**
 - Knowledge Base articles
 - Server logs
 - Records from relational database tables
 - Equipment status reports
 - Transcripts for online support chat sessions with customers

Input Data Sources in our Scenario



Collecting Input Data

- The Enterprise Data Hub offers several tools for ingesting data, such as
 - The hadoop fs command: Best for static documents
 - Sqoop: Used to exchange data with a database server
 - Flume: Collects data from various sources as it is generated



Considerations for Data Modification

- **The data you find may not be exactly what you need**
 - For example, you may prefer CSV but only fixed-width format is available
 - Data could have far more fields than you require
 - Records might contain a mix of date formats or timezones
- **How and when should you modify input data?**
 - The answer depends on many factors
- **In general, be cautious about modifying the *original* data set**
 - Particularly when removing fields
 - Modifications are usually permanent
- **HDFS storage is inexpensive – make a copy and modify *that* instead**
 - Allows you to tune performance for a specific use case

CSV: Comma-Separated Values

Chapter Topics

Application Architecture

- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- Identifying and Collecting Input Data
- **Selecting Tools for Data Processing and Analysis**
- Presenting Results to the User
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Tools for Analysis and Processing

- We are not limited to a single tool, technique, or programming language

Tool	Description
MapReduce	<ul style="list-style-type: none">• Low-level, batch-oriented processing
Apache Crunch	<ul style="list-style-type: none">• Java API for high-level processing pipelines via MapReduce• Faster and less tedious than writing MapReduce directly in Java
Apache Hive	<ul style="list-style-type: none">• SQL-like queries executed using MapReduce
Cloudera Impala	<ul style="list-style-type: none">• SQL-like queries on a custom execution engine• Lacks a few features of Hive, but queries run <i>much</i> faster
Cloudera Search	<ul style="list-style-type: none">• Extremely fast full-text search of data on Hadoop• Does not require the end-user to know a programming language

Chapter Topics

Application Architecture

- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- **Presenting Results to the User**
- Essential Points
- Hands-On Exercise: Natural Language Processing in MapReduce

Result Presentation

- **How does the end user access the results of your processing?**
 - It depends on what you produced
- **Static output from analysis that can be given directly to user**
 - Reports
 - Data sets
 - Batch-oriented tools (MapReduce, Pig, Hive, Crunch) are OK for this
- **Results that are used as input in a dynamic system**
 - Dashboard
 - Interactive application
 - High-performance tools (Impala and Search) are better for this

Chapter Topics

Application Architecture

- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- **Essential Points**
- Hands-On Exercise: Natural Language Processing in MapReduce

Bibliography

The following offer more information on topics discussed in this chapter

- **Eclipse IDE Web site**
 - <https://www.eclipse.org/>
- **JUnit Wiki**
 - <http://tiny.cloudera.com/adcc02a>
- **Apache Maven Web site**
 - <https://maven.apache.org/>
- ***Future of Data Management* keynote video from Strata 2013**
 - <http://tiny.cloudera.com/adcc02b>

Essential Points

- **HDFS provides low-cost centralized storage for all types of data**
- **Several tools are available for ingesting this data into your cluster**
 - Including hadoop fs commands, Sqoop, Flume
 - You should generally avoid making changes to the *original* data
- **There are many options for processing and querying data, such as**
 - MapReduce
 - Apache Crunch
 - Apache Hive
 - Cloudera Impala
 - Cloudera Search
- **Application development should start with a small-scale prototype**

Chapter Topics

Application Architecture

- Scenario Explanation
- Instructor-led Demo: Loudacre Mobile Support Portal
- Understanding the Development Environment
- Identifying and Collecting Input Data
- Selecting Tools for Data Processing and Analysis
- Presenting Results to the User
- Essential Points
- **Hands-On Exercise: Natural Language Processing in MapReduce**

How to Approach the Hands-On Exercises

- **Most Hands-On Exercises provide three sets of code**

Name	Description	Recommended for
Stubs	Minimal implementation	Advanced students who work <i>very</i> quickly
Hints	Partial implementation	Most students
Solutions	Complete implementation	Comparison with your own solution, or when you need an additional hint

- ***Red, italicized text* in the Exercise Manual indicates a value to change**
 - For example, to replace ‘solution’ in a file path with ‘hints’ or ‘stubs’
- **Most also provide one or more “bonus” exercises**
 - Please attempt these if time remains after completing the main exercise

Hands-On Exercise: Natural Language Processing in MapReduce

- In this Hands-On Exercise, you will use a custom Natural Language Processing (NLP) library to normalize keywords used in Loudacre's Knowledge Base articles
 - Please refer to the Hands-On Exercise Manual for instructions

Designing and Using Data Sets

Chapter 3



Course Chapters

- Introduction
- Application Architecture
- **Designing and Using Data Sets**
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Designing and Using Data Sets

In this chapter you will learn

- What metadata is and how it is managed in Hadoop
- How to define Avro schemas
- How Avro schemas can evolve to accommodate changing requirements
- How to extract data and metadata from an Avro data file
- How commonly-used Hadoop file formats compare to one another
- How compression and partitioning affect performance

Chapter Topics

Designing and Using Data Sets

- **Metadata Management**
 - Introduction to Apache Avro
 - Avro Schemas
 - Hands-On Exercise: Working with Avro Schemas
 - Avro Schema Evolution
 - Selecting a File Format
 - Performance Considerations
 - Essential Points

Data and Metadata

- **Data refers to the information you store and process**
 - Billing records, sensor readings, and server logs are examples of data
- **Metadata describes important aspects of that data**
 - Field name and order are examples of metadata

Metadata Data

	id	name	title	bonus
	108424	Alice	Salesperson	2500
	101837	Bob	Manager	3000
	107812	Chuck	President	9000
	105476	Dan	Accountant	3000
	104028	Eve	Manager	3500
	105293	Frank	Salesperson	2000

Metadata in a MapReduce Job

- The following Mapper excerpt shows how we might process this data
 - The input Path and InputFormat are also examples of metadata

```
@Override
public void map(LongWritable key, Text value, Context c)
    throws IOException, InterruptedException {

    String line = value.toString();
    String[] fields = line.split("\t");
    String title = fields[2];
    int bonus = Integer.valueOf(fields[3]);

    c.write(new Text(title), new IntWritable(bonus));
}
```

Metadata in Apache Hive

- **Hive is a high-level tool that uses an SQL-like syntax to query data**
 - It fulfills these queries by running MapReduce jobs on the cluster
 - The following yields the same result as the previous Mapper

```
hive> SELECT title, bonus FROM employees;
```

- **How does Hive know about the format and location of the data?**
 - You must specify them prior to using the “table” in a query

```
hive> CREATE TABLE employees
      (id INT, name STRING, title STRING, bonus INT)
      ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
      LOCATION '/loudacre/employees';
```

The Hive Metastore and HCatalog

- **Table metadata is managed by Hive's *metastore***
 - As the name implies, the Hive metastore is part of Apache Hive
 - The metastore uses a relational database to store the metadata
- **The ability to store and access metadata is useful outside of Hive too**
- **HCatalog is a Hive sub-project that provides access to the Metastore**
 - Accessible via command line and REST API
 - Allows you to define table-like abstractions using Hive syntax
 - Access those “tables” from Hive, Impala, MapReduce, and other tools
 - Ships with CDH 4.2.0 and later

Creating Tables in HCatalog

- HCatalog uses Hive's DDL (data definition language) syntax
 - You can specify a single command using the `-e` option

```
$ hcat -e "CREATE TABLE vendors \
(id INT, company STRING, email STRING) \
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
LOCATION '/loudacre/vendors'"
```

- Tip: save longer commands to a text file and use the `-f` option
 - If the file has more than one command, separate each with a semicolon

```
$ hcat -f createtable.txt
```

Displaying Metadata in HCatalog

- The **SHOW TABLES** command also shows tables created directly in Hive

```
$ hcat -e 'SHOW TABLES'  
employees  
vendors
```

- The **DESCRIBE** command lists the fields in a specified table
 - Use **DESCRIBE FORMATTED** instead to see detailed information

```
$ hcat -e 'DESCRIBE vendors'  
id      int  
company string  
email   string
```

Removing a Table in HCatalog

- The **DROP TABLE** command has the same behavior as it does in Hive
 - Caution: this will typically remove the data as well as the metadata!

```
$ hcat -e 'DROP TABLE vendors'
```

```
$ hcat -e 'SHOW TABLES'
```

```
employees
```

Chapter Topics

Designing and Using Data Sets

- Metadata Management
- **Introduction to Apache Avro**
- Avro Schemas
- Hands-On Exercise: Working with Avro Schemas
- Avro Schema Evolution
- Selecting a File Format
- Performance Considerations
- Essential Points

Data Serialization

- **To understand Avro, we must first define *serialization***
 - A way of representing data in memory as a series of bytes
 - Allows us to save data to disk or send it across the network
 - *Deserialization* allows us to read that data back into memory
- **For example, how do you serialize the number 108125150?**
 - 4 bytes when stored as a Java int
 - 9 bytes when stored as a Java String
- **Many programming languages and libraries support serialization**
 - Such as Serializable in Java or Writables in Hadoop
- **Backwards compatibility and cross-language support can be challenging**
 - Avro was developed to address these challenges

What is Apache Avro?

- **Avro is an efficient data serialization framework**
 - Apache project created by Doug Cutting (creator of Hadoop)
 - Widely supported throughout Hadoop and its ecosystem
- **Offers compatibility without sacrificing performance**
 - Data is serialized according to a *schema* you define
 - Read/write data in Java, C, C++, C#, Python, PHP, and other languages
 - Serializes data using a highly-optimized binary encoding
 - Specifies rules for *evolving* your schema over time
- **Avro also supports Remote Procedure Calls (RPC)**
 - Can be used for building custom network protocols
 - Flume uses this for internal communication



Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- **Avro Schemas**
- Hands-On Exercise: Working with Avro Schemas
- Avro Schema Evolution
- Selecting a File Format
- Performance Considerations
- Essential Points

Avro Schemas

- **Avro schemas define the structure of your data**
 - Similar in concept to CREATE TABLE in SQL, but more flexible
 - Schemas are represented using JSON
- **Three approaches for mapping a Java object to a schema**
 - **Generic:** Write code to map each schema field to a field in your object
 - Flexible, but sacrifices compile-time type safety
 - **Reflect:** Generate a schema from an existing class
 - Easy, but slower at runtime and may limit compatibility
 - **Specific:** Generate a Java class from your schema
 - Recommended, since it offers best performance and compatibility
 - There is even a Maven goal for doing this as part of your build

JSON: JavaScript Object Notation

Supported Types in Avro Schemas (Simple)

- A simple type holds exactly one value
- Avro's string type can be mapped to an Avro utility class, if desired
 - Using `org.apache.avro.util.Utf8` may improve performance

Name	Description	Java Equivalent
null	An absence of a value	null
boolean	A binary value	boolean
int	32-bit signed integer	int
long	64-bit signed integer	long
float	Single-precision floating point value	float
double	Double-precision floating point value	double
bytes	Sequence of 8-bit unsigned bytes	<code>java.nio.ByteBuffer</code>
string	Sequence of Unicode characters	<code>java.lang.CharSequence</code>

Supported Types in Avro Schemas (Complex)

- Avro also supports six complex types

Name	Description
record	A user-defined type composed of one or more named fields
enum	A specified set of values
array	Zero or more values of the same type
map	Set of key-value pairs; key is string while value is of specified type
union	Exactly one value matching a specified set of types
fixed	A fixed number of 8-bit unsigned bytes

- The **record** type is the most important
 - Main use of other types is to define a record's fields

Basic Schema Example

- Excerpt from a SQL CREATE TABLE statement

```
CREATE TABLE employees
  (id INT, name STRING, title STRING, bonus INT)
```

- Equivalent Avro schema

```
{"namespace": "com.loudacre.data",
"type": "record",
"name": "Employee",
"fields": [
  {"name": "id", "type": "int"},
  {"name": "name", "type": "string"},
  {"name": "title", "type": "string"},
  {"name": "bonus", "type": "int"}]
```

Specifying Default Values in the Schema

- The SQL CREATE TABLE statement allows you to specify a default value for a field
 - Used when no value was explicitly set in the data
- Avro also supports setting a default value in the schema

```
{ "namespace": "com.loudacre.data",
  "type": "record",
  "name": "Invoice",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "taxcode", "type": "int", "default": "39"},
    {"name": "lang", "type": "string", "default": "EN_US"}
  ]
}
```

Avro Schemas and Null Values

- Avro checks for null values when serializing the data
- Null values are only allowed *when explicitly specified* in the schema
 - The title and bonus fields in the example below allow null values
 - If using a default, its type must be listed first in the union

```
{"namespace": "com.loudacre.data",
"type": "record",
"name": "Employee",
"fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "title", "type": ["null", "string"]},
    {"name": "bonus", "type": ["null", "int"]}
]}
```

Schema Example with Complex Types

- The following example shows a record with an enum and a string array

```
{ "namespace": "com.loudacre.data",
  "type": "record",
  "name": "Ringtone",
  "fields": [
    { "name": "id", "type": "int" },
    { "name": "title", "type": "string" },
    { "name": "price", "type": "int" },
    { "name": "format", "type": {
        "name": "RingtoneFormat", "type": "enum",
        "symbols": [ "MP3", "AAC", "MIDI" ] }
    },
    { "name": "tags", "type": {
        "type": "array", "items": "string" }
    }
  ]
}
```

Documenting Your Schema

- It is good practice to document any ambiguities in a schema
 - All types (including record) support an optional `doc` attribute

```
{ "namespace": "com.loudacre.data",
  "type": "record",
  "name": "WebProduct",
  "doc": "Item currently sold in Loudacre's online store",
  "fields": [
    {"name": "id", "type": "int", "doc": "Product SKU"},
    {"name": "shipwt", "type": "int",
     "doc": "Shipping weight, in pounds"},
    {"name": "price", "type": "int",
     "doc": "Retail price, in cents (US)"}
  ]
}
```

Avro Container Format

- **Avro also defines a container file format for storing Avro records**
 - Also known as “Avro data file format”
 - Similar to Hadoop’s SequenceFile format
 - Cross-language support for reading and writing data
- **Supports compressing *blocks* (groups) of records**
 - It is “splittable” for efficient processing by MapReduce
- **This format is self-describing**
 - Each file contains a copy of the schema used to write its data
 - All records in a file must use the same schema

Inspecting Avro Data Files with Avro Tools

- **Avro data files are an efficient way to store data**
 - However, the binary format makes debugging difficult
- **Each Avro release contains an Avro Tools JAR file**
 - This tool allows you to read the schema or data for an Avro file
 - JAR name contains the version, such as `avro-tools-1.7.6.jar`
 - Available for download from the Avro Web site or Maven repository

```
$ java -jar avro-tools*.jar tojson mydatafile.avro
{"name": "Alice", "salary": 56500, "city": "Anaheim"}
{"name": "Bob", "salary": 51400, "city": "Bellevue"}
```



```
$ java -jar avro-tools*.jar getschema mydatafile.avro
{
  "type" : "record",
  "name" : "DeviceData",
  "namespace" : "com.loudacre.data",    ...rest of schema follows
```

Aside: Generating Java Source from an Avro Schema

- When using Avro “specific” records, you must first generate the Java code
- One way to generate code is to use the Avro Tools JAR file
 - Output directory must already exist
 - We recommend that you generate code to its own directory

```
$ mkdir generated-src  
$ java -jar avro-tools-*.jar compile schema \  
employee.avsc generated-src
```

- Using Maven to generate Java code from a schema is even easier
 - You will do this in an upcoming Hands-On Exercise
- You may notice that the fields in the generated code are deprecated
 - This is to encourage you to use their get and set methods instead

Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- Avro Schemas
- **Hands-On Exercise: Working with Avro Schemas**
- Avro Schema Evolution
- Selecting a File Format
- Performance Considerations
- Essential Points

Hands-on Exercise: Working with Avro Schemas

- **In this Hands-On Exercise, you will create and test an Avro schema**
 - Please refer to the Hands-On Exercise Manual for instructions

Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- Avro Schemas
- Hands-On Exercise: Working with Avro Schemas
- **Avro Schema Evolution**
- Selecting a File Format
- Performance Considerations
- Essential Points

Schema Evolution

- **The structure of your data will change over time**
 - Fields may be added, removed, changed, or renamed
 - In SQL, these are handled with ALTER TABLE statements
- **These changes can break compatibility with many formats**
 - Writable objects serialized in SequenceFiles become unreadable
- **Data written to Avro data files is always readable**
 - The schema used to write the data is embedded in the file itself
 - However, an application reading data might expect the *new* structure
- **Avro has a unique approach to maintaining forward compatibility**
 - A reader can use a different schema than the writer

Schema Evolution: A Practical Example (1)

- Imagine that we have written millions of records with this schema

```
{"namespace": "com.loudacre.data",
"type": "record",
"name": "CustomerContact",
"fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "faxNumber", "type": "string"}
]}
```

Schema Evolution: A Practical Example (2)

- We would like to modernize this based on the schema below
 - Renamed `id` field to `customerId`
 - Changed type from `int` to `long`
 - Removed `faxNumber` field
 - Added `email` field

```
{ "namespace": "com.loudacre.data",
  "type": "record",
  "name": "CustomerContact",
  "fields": [
    { "name": "customerId", "type": "long" },
    { "name": "name", "type": "string" },
    { "name": "email", "type": "string" }
  ]
}
```

Schema Evolution: A Practical Example (3)

- **We could use the new schema to write new data**
 - Applications that use the new schema could read the new data
- **Unfortunately, new applications wouldn't be able to read the *old* data**
 - We must make a few schema changes to improve compatibility

Schema Evolution: A Practical Example (4)

- If you rename a field, you must specify an alias for the old name(s)
 - Here, we map the old `id` field to the new `customerId` field

```
{"namespace": "com.loudacre.data",
"type": "record",
"name": "CustomerContact",
"fields": [
    {"name": "customerId", "type": "long",
     "aliases": ["id"]},
    {"name": "name", "type": "string"},
    {"name": "email", "type": "string"}]}  
}]}
```

Schema Evolution: A Practical Example (5)

- Newly-added fields will lack values for records previously written
 - You must specify a default value
 - In this case, the field is made nullable so null can be the default

```
{"namespace": "com.loudacre.data",
"type": "record",
"name": "CustomerContact",
"fields": [
    {"name": "customerId", "type": "long",
     "aliases": ["id"]},
    {"name": "name", "type": "string"},
    {"name": "email", "type": ["null", "string"],
     "default": null}
]}
```

Schema Evolution: Compatible Changes

- **The following changes will not affect existing readers**

- Adding, changing, or removing a `doc` attribute
- Changing a field's default value
- Adding a new field with a default value
- Removing a field that specified a default value
- Promoting a field to a wider type (e.g., `int` to `long`)
- Adding aliases for a field

Schema Evolution: Incompatible Changes

- **The following are some changes that may break compatibility**
 - Changing the record's name or namespace attributes
 - Adding a new field without a default value
 - Removing a symbol from an enum
 - Removing a type from a union
 - Modification to a field's type that could result in truncation
- **Such changes may require you to perform the following steps**
 - Read your old data (using the original schema)
 - Modify data as needed in your application
 - Write the new data (using the new schema)
 - Existing readers/writers may need to be updated to use new schema

Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- Avro Schemas
- Hands-On Exercise: Working with Avro Schemas
- Avro Schema Evolution
- **Selecting a File Format**
- Performance Considerations
- Essential Points

Considerations for Choosing a File Format

- **Hadoop and its ecosystem support many file formats**
 - May ingest data in one format, but convert to another as needed
- **Selecting the format for your data set involves several considerations**
 - Ingest pattern
 - Tool compatibility
 - Expected lifetime
 - Storage and performance requirements
- **Which format is best?**
 - It depends on *your* data and use cases
 - The following slides offer some general guidance on common formats

Hadoop File Formats: Text

- **Text files are the most basic file type in Hadoop**
 - Can be read or written from virtually any programming language
 - Comma- and tab-delimited files are compatible with many applications
- **Text files are human readable, since everything is a string**
 - Useful when debugging
- **At scale, this format is inefficient**
 - Representing numeric values as strings wastes storage space
 - Difficult to represent binary data such as images
 - Often resort to techniques such as Base64 encoding
 - Conversion to/from native types adds performance penalty
- **Verdict: Good interoperability, but poor performance**

Hadoop File Formats: SequenceFiles

- **SequenceFiles store key-value pairs in a binary container format**
 - Less verbose and more efficient than text files
 - Capable of storing binary data such as images
 - Format is Java-specific and tightly coupled to Hadoop
- **Verdict: Good performance, but poor interoperability**

Hadoop File Formats: Avro Data Files

- **Efficient storage due to optimized binary encoding**
- **Widely supported throughout the Hadoop ecosystem**
 - Can also be used outside of Hadoop
- **Ideal for long-term storage of important data**
 - Can read and write from many languages
 - Embeds schema in the file, so will always be readable
 - Schema evolution can accommodate changes
- **Verdict: Excellent interoperability and performance**
 - Best choice for general-purpose storage in Hadoop

Columnar Formats

- Hadoop also supports a few **columnar formats**
 - These organize data storage by column, rather than by row
 - Very efficient when selecting only a small subset of a table's columns

id	name	city	occupation	income	phone
1	Alice	Palo Alto	Accountant	85000	650-555-9748
2	Bob	Sunnyvale	Accountant	81500	650-555-8865
3	Bob	Palo Alto	Dentist	196000	650-555-7185
4	Bob	Palo Alto	Manager	87000	650-555-2518
5	Carol	Palo Alto	Manager	79000	650-555-3951
6	David	Sunnyvale	Mechanic	62000	650-555-4754

Organization of data in traditional row-based formats

id	name	city	occupation	income	phone
1	Alice	Palo Alto	Accountant	85000	650-555-9748
2	Bob	Sunnyvale	Accountant	81500	650-555-8865
3	Bob	Palo Alto	Dentist	196000	650-555-7185
4	Bob	Palo Alto	Manager	87000	650-555-2518
5	Carol	Palo Alto	Manager	79000	650-555-3951
6	David	Sunnyvale	Mechanic	62000	650-555-4754

Organization of data in columnar formats

Hadoop Columnar File Formats: RCFile and ORCFile

- **RCFile**

- A *column-oriented* format originally created for Hive tables
- All data stored as strings (inefficient)
- **Verdict:** poor performance, and limited interoperability

- **ORCFile**

- An improved version of RCFile currently under development
- More efficient than RCFile, but not yet well supported outside of Hive
- **Verdict:** improved performance, but limited interoperability

Hadoop File Formats: Parquet

- **Parquet is a columnar format developed by Cloudera and Twitter**
 - Supported in MapReduce, Hive, Pig, Impala, Crunch, and others
 - Contains embedded metadata
- **Uses advanced optimizations described in Google's Dremel paper**
 - Reduces storage space
 - Increases performance
- **Most efficient when adding many records at once**
 - Some optimizations rely on identifying repeated patterns
- **Verdict: Excellent interoperability and performance**
 - Best choice for column-based access patterns

Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- Avro Schemas
- Hands-On Exercise: Working with Avro Schemas
- Avro Schema Evolution
- Selecting a File Format
- **Performance Considerations**
- Essential Points

Performance Considerations

- You have just learned how different file formats affect performance
- Two other factors can significantly affect performance
 - Data compression
 - How data is organized on disk (partitioning)

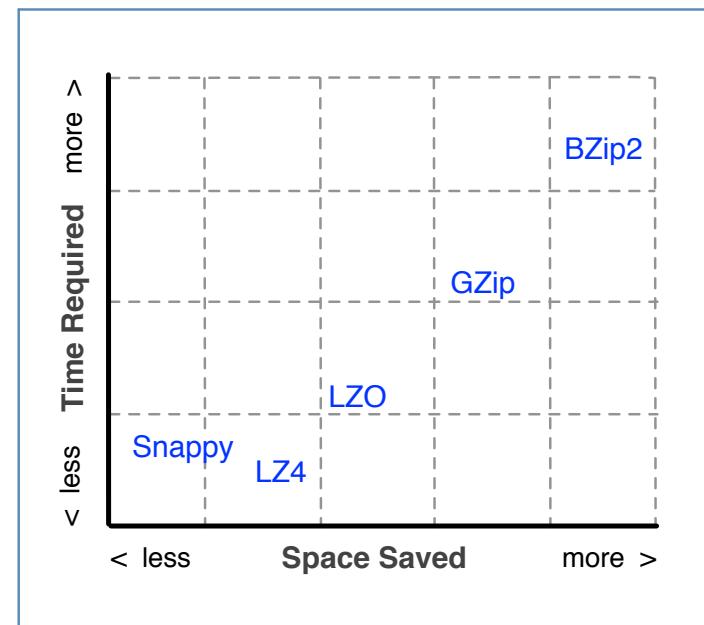
Data Compression

- **Each file format may also support compression**
 - This reduces amount of disk space required to store data
- **Compression is a tradeoff between CPU time and storage space**
 - Aggressive algorithms take a long time, but save more space
 - Less aggressive algorithms save less space but are much faster
- **Can significantly improve performance**
 - Many Hadoop jobs are I/O-bound
 - Using compression allows you to handle more data per I/O operation
 - Compression can also improve the performance of network transfers



Compression Codecs

- The implementation of a compression algorithm is known as a *codec*
 - Short for compressor/decompressor
- Many codecs are commonly used with Hadoop
 - Each has different performance characteristics
 - Not all Hadoop tools are compatible with all codecs
- Overall, BZip2 saves the most space
 - But LZ4 and Snappy are much faster
 - Impala supports Snappy but not LZ4
- For “hot” data, speed matters most
 - Better to compress by 40% in one second than by 80% in 10 seconds

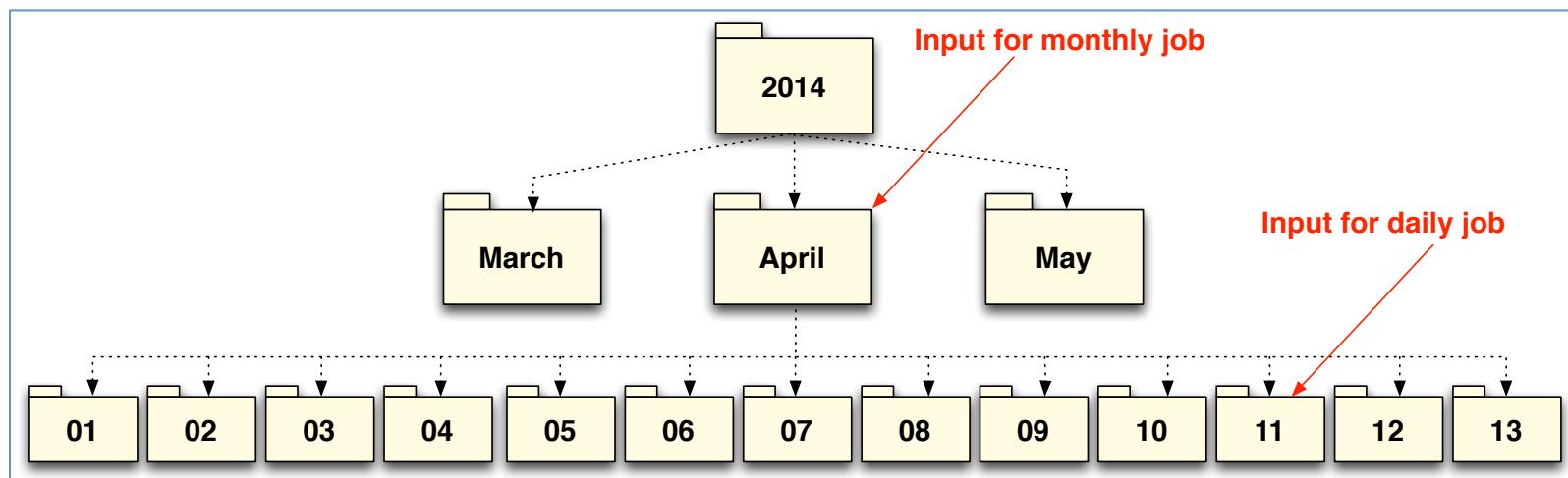


Data Partitioning

- **Partitioning refers to organizing data according to access patterns**
 - Improves performance by limiting the amount of input data to a job
- **Hive and Impala have built-in support for partitioning**
 - MapReduce can read input directories recursively if a configuration property is set
- **Common partitioning schemes**
 - Customers: partition by state, province, or region
 - Sales: separate by year, month, and day
- **We will now examine another example in greater detail...**

Partitioning Example

- Imagine that you store all your Web server log files in HDFS
 - Marketing runs *monthly* jobs to help with search engine optimization
 - Security runs *daily* jobs to identify attempted exploits
- In this case, it makes sense to partition into subdirectories by date
 - Logs from April 11, 2014 might go in /logs/web/2014/04/11
 - The daily job could use /logs/web/2014/04/11 as input
 - The monthly job could use /logs/web/2014/04 as input



Chapter Topics

Designing and Using Data Sets

- Metadata Management
- Introduction to Apache Avro
- Avro Schemas
- Hands-On Exercise: Working with Avro Schemas
- Avro Schema Evolution
- Selecting a File Format
- Performance Considerations
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Avro Getting Started Guide (Java)**
 - <http://tiny.cloudera.com/adcc03a>
- **Avro Specification**
 - <http://tiny.cloudera.com/adcc03b>
- **Announcing Parquet 1.0: Columnar Storage for Hadoop**
 - <http://tiny.cloudera.com/adcc03c>

Essential Points

- **Working with data sets in Hadoop requires many choices**
 - Such as file format, compression codec, and partitioning schema
 - Bad decisions can limit performance and interoperability
- **Avro is an language-neutral serialization system and file format**
 - Compact binary encodings provide good performance
 - Supports schema evolution for long-term storage
- **Parquet is a high-performance columnar format for Hadoop**
 - Good choice when accessing a small subset of columns in a table

Using the Kite SDK Data Module

Chapter 4



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- **Using the Kite SDK Data Module**
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Using the Kite SDK Data Module

In this chapter you will learn

- **What the Kite SDK is and why it was developed**
- **How Kite stores data and metadata**
- **What key components make up the Kite data module**
- **How to create, populate, read, and delete data sets with the Kite APIs**

Chapter Topics

Using the Kite SDK Data Module

- **What is the Kite SDK?**
- Fundamental Data Module Concepts
- Creating a New Data Set Using the Kite SDK
- Loading, Accessing, and Deleting a Data Set
- Essential Points
- Hands-On Exercise: Transforming Data with Kite

What is the Kite SDK?

- **The Kite SDK is a set of libraries and tools**
 - Open source, released under the Apache software license
 - Focuses on solving specific problems rather than using specific tools
 - Everything in Kite is client-side (requires no change to servers)
- **Aims to simplify the creation of data-oriented applications**
 - Encapsulates patterns and best practices from experts
 - Default values represent typical (sensible) choices
- **Allows developers to concentrate on business logic**
 - Instead of wasting time on low-level application “plumbing”

Kite SDK Modules

- **The Kite SDK is divided into several loosely-coupled modules**
 - You can select just the ones you need
 - Each module is designed to have minimal external dependencies
- **In this class, we will focus on the Kite Data Module**
 - Later, we will also use a Kite tool called Morphlines to do ETL processing

ETL: Extract, Transform, Load

Chapter Topics

Using the Kite SDK Data Module

- What is the Kite SDK?
- **Fundamental Data Module Concepts**
- Creating a New Data Set Using the Kite SDK
- Loading, Accessing, and Deleting a Data Set
- Essential Points
- Hands-On Exercise: Transforming Data with Kite

Kite SDK Data Module Overview

- HDFS defines *byte-oriented APIs* for reading and writing data
- The Kite Data module provides a *high-level API* for this
 - Avoids HDFS-specific constructs like directories, files, and streams
 - API helps to decouple schema from underlying storage
- This module is helpful for integrating applications with Hadoop
 - Makes it easy to create data and populate data sets from Java
 - For example, to parse XML from local files and write to HDFS as Avro

Defaults in Kite SDK Data Module

- **Default values balance performance and compatibility concerns**
- **Schema definition**
 - Always defined using Avro, regardless of underlying storage
- **File format**
 - Uses Avro data files by default, but also supports Parquet
- **Compression**
 - Automatically compresses data using Snappy
- **Partitioning**
 - Does not partition by default, but this is easy to enable
 - Supports several built-in partitioning strategies

Important Concepts

- **Data entity: a single logical unit of data**
 - Analogous to a record in a database table
- **Data set: a collection of data entities**
 - Analogous to a database table
- **Data set repository: maps logical data sets to physical storage**
 - Details are specified in a Data Set Descriptor
 - Includes path, format, schema, and partitioning strategy
 - Schemas in the Kite SDK are defined using Avro
- **Data sets are not directly instantiated**
 - Instead, they are accessed using the data set's repository

Chapter Topics

Using the Kite SDK Data Module

- What is the Kite SDK?
- Fundamental Data Module Concepts
- **Creating a New Data Set Using the Kite SDK**
- Loading, Accessing, and Deleting a Data Set
- Essential Points
- Hands-On Exercise: Transforming Data with Kite

Creating a Data Set with the Kite SDK (1)

- **Creating a new data set involves five steps**
 1. Define the Avro schema for the data set
 2. Optionally, generate Java code from your schema
 3. Open a connection to the data set repository
 4. Build a `DatasetDescriptor` with implementation details
 5. Pass the data set name and descriptor to the repo's `create` method
- **This is conceptually similar to the steps for creating a table using JDBC**

Creating a Data Set with the Kite SDK (2)

- The Avro schema for our example data set is shown below
 - Schemas are typically stored in the project's main/resources folder
 - By convention, the file name would be `employee.avsc`

```
{ "namespace": "com.loudacre.data",
  "type": "record",
  "name": "Employee",
  "fields": [
    { "name": "id", "type": "int" },
    { "name": "name", "type": "string" },
    { "name": "title", "type": "string" },
    { "name": "bonus", "type": "int" }
  ]
}
```

Creating a Data Set with the Kite SDK (3)

- The remaining steps are executed from a Java class
 - Typically based on one used to run a MapReduce job

```
public class DatasetCreator extends Configured implements Tool {  
  
    public static void main(String... args) throws Exception {  
        int status = ToolRunner.run(new DatasetCreator(), args);  
        System.exit(status);  
    }  
  
    @Override  
    public int run(String[] args) throws Exception {  
        // code for creating the data set goes here  
    }  
}
```

Creating a Data Set with the Kite SDK (4)

- **Next, open a reference to the DatasetRepository**
 - This represents the system that will store the data set

```
DatasetRepository repo = DatasetRepositories.open("URI");
```

- **The URI supplied to the open method implies the storage details**

Repository URI	Metadata Storage	Data Storage
repo:hive	Hive metastore	Hive-managed (in a subdirectory of /user/hive/warehouse)
repo:hive:/loudacre	Hive metastore	Self-managed (in a subdirectory of the specified path)
repo:hdfs:/loudacre	Within a hidden directory in HDFS	Self-managed (in a subdirectory of the specified path)

Creating a Data Set with the Kite SDK (5)

- Next, build a **DatasetDescriptor** to specify data set format details
 - This example uses the default settings (Snappy-compressed Avro data)

```
DatasetDescriptor descriptor =
    new DatasetDescriptor.Builder()
        .schemaUri("resource:employee.avsc")
        .build();
```

- This example will store data in Parquet format

```
DatasetDescriptor descriptor =
    new DatasetDescriptor.Builder()
        .format(Fормats.PARQUET)
        .schemaUri("resource:employee.avsc")
        .build());
```

Creating a Data Set with the Kite SDK (6)

- Finally, call the repository's **create** method with a name and descriptor

```
Dataset<Employee> dataset =  
    repo.create("employees", descriptor);
```

- The **dataset** reference can now be used to populate the data set
 - Or you can do this in a later session by loading the data set

Chapter Topics

Using the Kite SDK Data Module

- What is the Kite SDK?
- Fundamental Data Module Concepts
- Creating a New Data Set Using the Kite SDK
- **Loading, Accessing, and Deleting a Data Set**
- Essential Points
- Hands-On Exercise: Transforming Data with Kite

Populating a Data Set with the Kite SDK (1)

- **Populating the data set is a four-step process**
 1. Get a writer from the data set
 2. Open the writer
 3. Write records
 4. Close the writer
- **If you no longer have a reference to the Dataset, you can load one**

```
DatasetRepository repo =  
    DatasetRepositories.open("repo:hive");  
  
Dataset<Employee> dataset = repo.load("employees");
```

Populating a Data Set with the Kite SDK (2)

- Call the `newWriter()` method to retrieve a writer instance
 - Open the writer and write as many records as you need

```
DatasetWriter<Employee> writer = dataset.newWriter();
try {
    writer.open();

    writer.write(new Employee(1, "Jed", "Janitor", 3500));
    writer.write(new Employee(2, "Ned", "Lawyer", 1000));
    writer.write(new Employee(3, "Ted", "Plumber", 5000));
} finally {
    // As with any I/O code, be sure to close the writer
    writer.close();
}
```

Accessing a Data Set with the Kite SDK

- You can now read the records you have written
 - You can access them directly in HDFS (`hadoop fs -get`)
 - If the dataset uses the `repo:hive` URI, you can also query using Hive or Impala
- Reading the data in Kite is similar to (but even easier than) writing it

```
DatasetReader<Employee> reader = dataset.newReader();
try {
    reader.open();
    for (Employee employee : reader) {
        // use the data as needed in your application
        System.out.println(employee);
    }
} finally {
    reader.close();
}
```

Deleting a Data Set with the Kite SDK

- If you no longer need a data set, you can delete it
 - This is analogous to dropping a table in a relational database
 - Caution: this *will* remove metadata and *can* remove data

```
DatasetRepository repo =  
    DatasetRepositories.open("repo:hive");  
  
repo.delete("employees");
```

Chapter Topics

Using the Kite SDK Data Module

- What is the Kite SDK?
- Fundamental Data Module Concepts
- Creating a New Data Set Using the Kite SDK
- Loading, Accessing, and Deleting a Data Set
- **Essential Points**
- Hands-On Exercise: Transforming Data with Kite

Bibliography

The following offer more information on topics discussed in this chapter

- **Kite SDK Article by Tom White in Oracle's *Java Magazine***
 - <http://tiny.cloudera.com/adcc04a>
- **Kite Data Module Reference Guide**
 - <http://tiny.cloudera.com/adcc04b>
- **Code Examples for the Kite Data Module**
 - <http://tiny.cloudera.com/adcc04c>

Essential Points

- **Kite SDK is a set of client-side tools and libraries**
 - Developed and distributed as loosely-coupled modules
 - 100% Apache-licensed open source
- **Creating even a simple application on Hadoop involves many decisions**
 - File formats
 - Compression codecs
 - How to store data and metadata
 - Tool compatibility
- **The Kite SDK data module abstracts away these low-level details**
 - Provides sensible defaults and patterns based on best practices
 - Lets you focus on the business logic

Chapter Topics

Using the Kite SDK Data Module

- What is the Kite SDK?
- Fundamental Data Module Concepts
- Creating a New Data Set Using the Kite SDK
- Loading, Accessing, and Deleting a Data Set
- Essential Points
- **Hands-On Exercise: Transforming Data with Kite**

Hands-on Exercise: Transforming Data with Kite

- In this Hands-On Exercise, you will use the Kite Data API to create a data set in Avro format. You will then populate it using phone activation data stored in XML format
 - Please refer to the Hands-On Exercise Manual for instructions

Importing Relational Data with Apache Sqoop

Chapter 5



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- **Importing Relational Data with Apache Sqoop**
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Importing Relational Data with Apache Sqoop

In this chapter you will learn

- How to import tables from an RDBMS into your Hadoop cluster
- How to change the delimiter and file format of imported tables
- How to control which columns and rows are imported
- What techniques you can use to improve Sqoop's performance
- How the next-generation version of Sqoop compares to the original

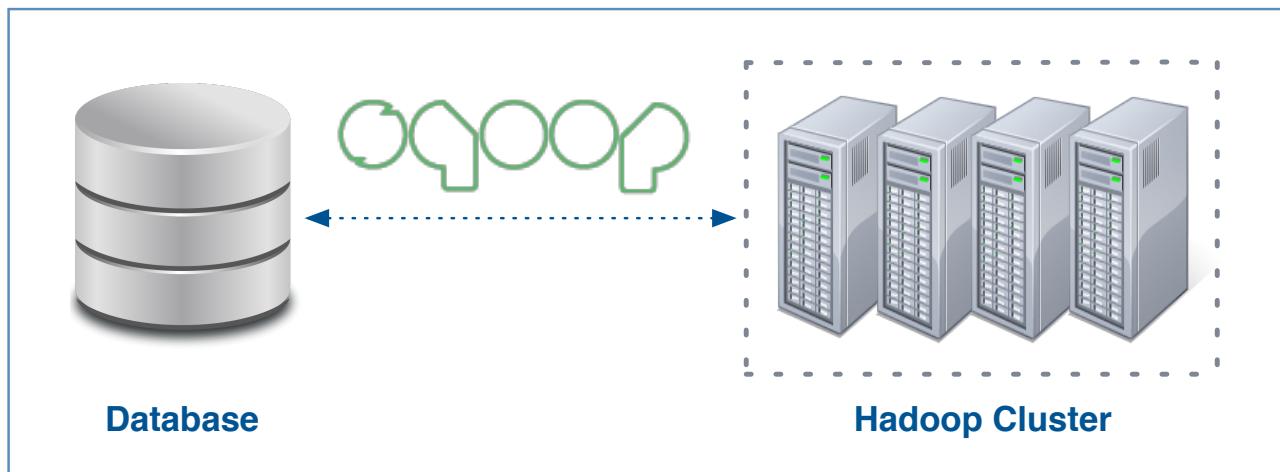
Chapter Topics

Importing Relational Data with Apache Sqoop

- **Sqoop Overview**
- Basic Imports and Exports
- Limiting Results
- Improving Sqoop's Performance
- Sqoop 2
- Essential Points
- Hands-On Exercise: Importing Customer Account Data

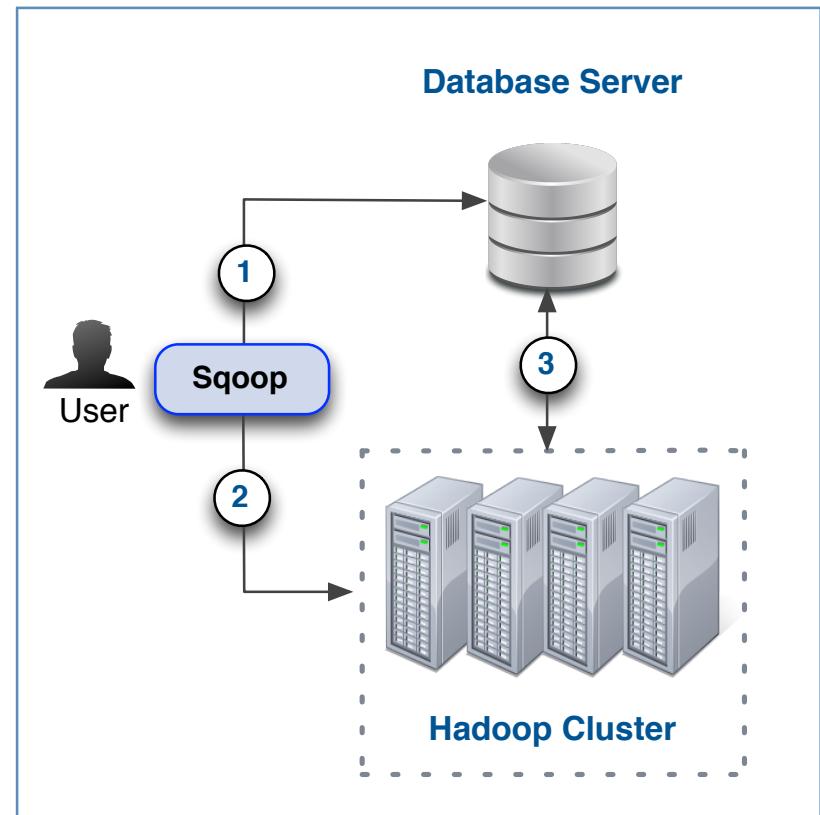
What is Apache Sqoop?

- Open source Apache project originally developed by Cloudera
 - The name is a contraction of “SQL-to-Hadoop”
- Sqoop exchanges data between a database and HDFS
 - Can import all tables, a single table, or a partial table into HDFS
 - Data can be imported in delimited text, SequenceFile, or Avro format
 - Sqoop can also export data from HDFS to a database



How Does Sqoop Work?

- Sqoop is a client-side application that imports data using Map-only jobs
- A basic import involves three steps orchestrated by Sqoop
 1. Examine table details
 2. Create and submit job to cluster
 3. Fetch records from table and write this data to HDFS



Basic Syntax

- **Sqoop is a command-line utility with several subcommands, called *tools***
 - There are tools for import, export, listing database contents, and more
 - Run `sqoop help` to see a list of all tools
 - Run `sqoop help tool-name` for help on using a specific tool
- **Basic syntax of a Sqoop invocation**

```
$ sqoop tool-name [tool-options]
```

- **This command will list all tables in the `loudacre` database in MySQL**

```
$ sqoop list-tables \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username twalker \
  --password bigsecret
```

Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- **Basic Imports and Exports**
- Limiting Results
- Improving Sqoop's Performance
- Sqoop 2
- Essential Points
- Hands-On Exercise: Importing Customer Account Data

Overview of the Import Process

- **Imports are performed using Map-only jobs**
- **Sqoop begins by examining the table to be imported**
 - Determines the primary key, if possible
 - Runs a *boundary query* to see how many records will be imported
 - Divides result of boundary query by the number of mappers
 - Uses this to configure tasks so that they will have equal loads
- **Sqoop also generates a Java source file for each table being imported**
 - It compiles and uses this during the import process
 - The file remains after import, but can be safely deleted

Importing An Entire Database with Sqoop

- The **import-all-tables** tool imports an entire database
 - Stored as comma-delimited files below your HDFS home directory
 - Data will be in subdirectories corresponding to the name of each table

```
$ sqoop import-all-tables \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username twheeler --password bigsecret
```

- Use the **--warehouse-dir** option to specify a different base directory

```
$ sqoop import-all-tables \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username twheeler --password bigsecret \
  --warehouse-dir /loudacre
```

Importing A Single Table with Sqoop

- The **import** tool imports a single table
- This example imports the **accounts** table
 - It stores the data in HDFS as comma-delimited fields

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username training --password training
```

- This variation writes tab-delimited fields instead

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username training --password training \
  --fields-terminated-by "\t"
```

Specifying the File Format for Imported Data

- Sqoop can also store the imported data in Avro format

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --as-avrodatafile
```

- Use **--as-sequencefile** instead for SequenceFile format

Enabling Compression for Imported Data

- **Sqoop does not compress imported data by default**
 - Add the `-z` option to enable compression (default: Gzip)
 - Use `--compression-codec` option to specify an alternate codec

Codec	Value of <code>compression-codec</code> argument
Snappy	<code>org.apache.hadoop.io.compress.SnappyCodec</code>
BZip2	<code>org.apache.hadoop.io.compress.BZip2Codec</code>

Incremental Imports

- What if new records are added to the database?
 - Could re-import all records, but this is inefficient
- Sqoop's incremental append mode imports only *new* records
 - Based on the value of the last record in the specified column

```
$ sqoop import --table invoices \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --incremental append \
  --check-column id \
  --last-value 9478306
```

Exporting Data from Hadoop to RDBMS with Sqoop

- Sqoop's **import** tool pulls records from an RDBMS into HDFS
- It is sometimes necessary to push data in HDFS back to an RDBMS
 - Good solution when you must do batch processing on large data sets
 - Export the results to a relational database for access by other systems
- Sqoop supports this via the **export** tool
 - The RDBMS table must already exist prior to export

```
$ sqoop export \
  --connect jdbc:mysql://dev.loudacre.com/loudacre \
  --username training --password training \
  --export-dir /loudacre/recommender_output \
  --update-mode allowinsert
  --table product_recommendations
```

Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- Basic Imports and Exports
- **Limiting Results**
- Improving Sqoop's Performance
- Sqoop 2
- Essential Points
- Hands-On Exercise: Importing Customer Account Data

Importing Partial Tables with Sqoop

- Import only the specified columns from the accounts table

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --columns "id,first_name,last_name,state"
```

- Import only matching rows from the accounts table

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --where "state='CA'"
```

Using a Free-Form Query

- You can also import the results of a query, rather than a single table
- Supply a complete SQL query using the **--query** option
 - You must add the *literal* WHERE \$CONDITIONS token
 - Use --split-by to identify the field used to divide the work among Mappers

```
$ sqoop import \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --split-by accounts.id \
  --query "SELECT accounts.id, first_name,
last_name, bill_amount FROM accounts JOIN invoices ON
(accounts.id = invoices.cust_id) WHERE $CONDITIONS"
```

Using a Free-Form Query with WHERE Criteria

- The --where option is ignored in a free-form query
 - You must specify your criteria using AND following the WHERE clause

```
$ sqoop import \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username training --password training \
  --split-by accounts.id \
  --query "SELECT accounts.id, first_name,
last_name, bill_amount FROM accounts JOIN invoices ON
(accounts.id = invoices.cust_id) WHERE $CONDITIONS AND
bill_amount >= 40"
```

Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- Basic Imports and Exports
- Limiting Results
- **Improving Sqoop's Performance**
- Sqoop 2
- Essential Points
- Hands-On Exercise: Importing Customer Account Data

Options for Database Connectivity

- **Generic (JDBC)**

- Compatible with nearly any database
 - Overhead imposed by JDBC can limit performance

- **Direct Mode**

- Can improve performance through the use of database-specific utilities
 - Currently supports MySQL and Postgres (use `--direct` option)
 - Not all Sqoop features are available in direct mode

- **Cloudera and partners offer high-performance Sqoop connectors**

- These use native database protocols rather than JDBC
 - Connectors available for Netezza, Teradata, and Oracle
 - Download these from Cloudera's Web site
 - Not open source due to licensing issues, but free to use

Controlling Parallelism

- By default, Sqoop typically imports data using four Map tasks
 - Increasing the number of Map tasks might improve import speed
 - Caution: Each Map task adds load to your database server
- You can *influence* the number of Map tasks using the **-m** option
 - Sqoop views this only as a hint and might not honor it

```
$ sqoop import --table accounts \
  --connect jdbc:mysql://db.loudacre.com/loudacre \
  --username twalker --password bigsecret \
  -m 8
```

- Sqoop assumes all tables have an evenly-distributed numeric primary key
 - Sqoop uses this column to divide work among mappers
 - You can use a different column with the **--split-by** option

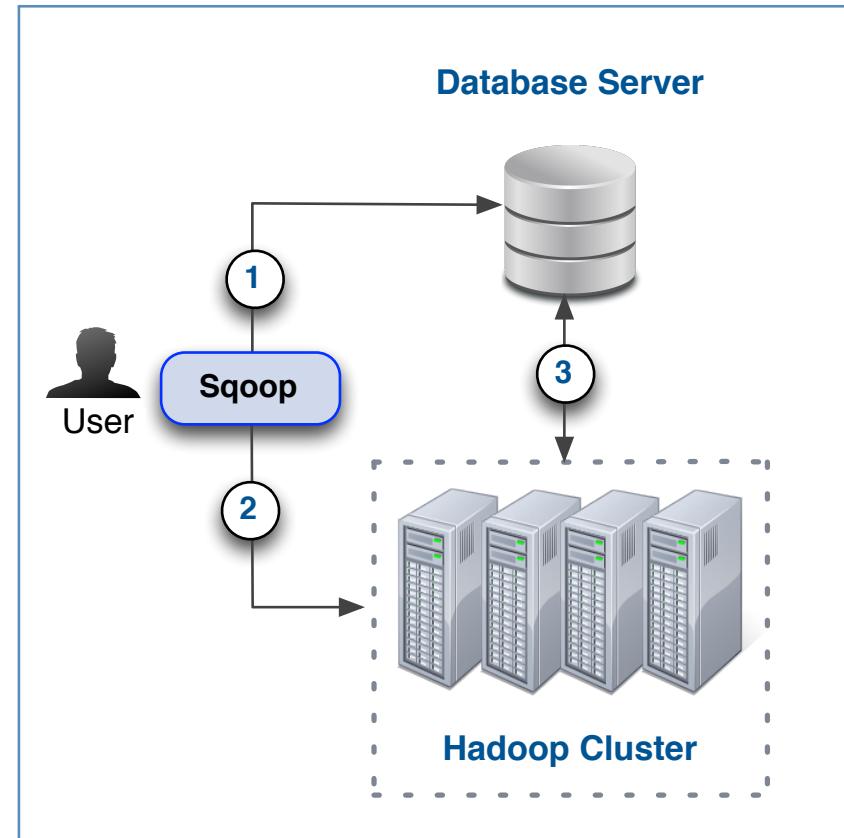
Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- Basic Imports and Exports
- Limiting Results
- Improving Sqoop's Performance
- **Sqoop 2**
- Essential Points
- Hands-On Exercise: Importing Customer Account Data

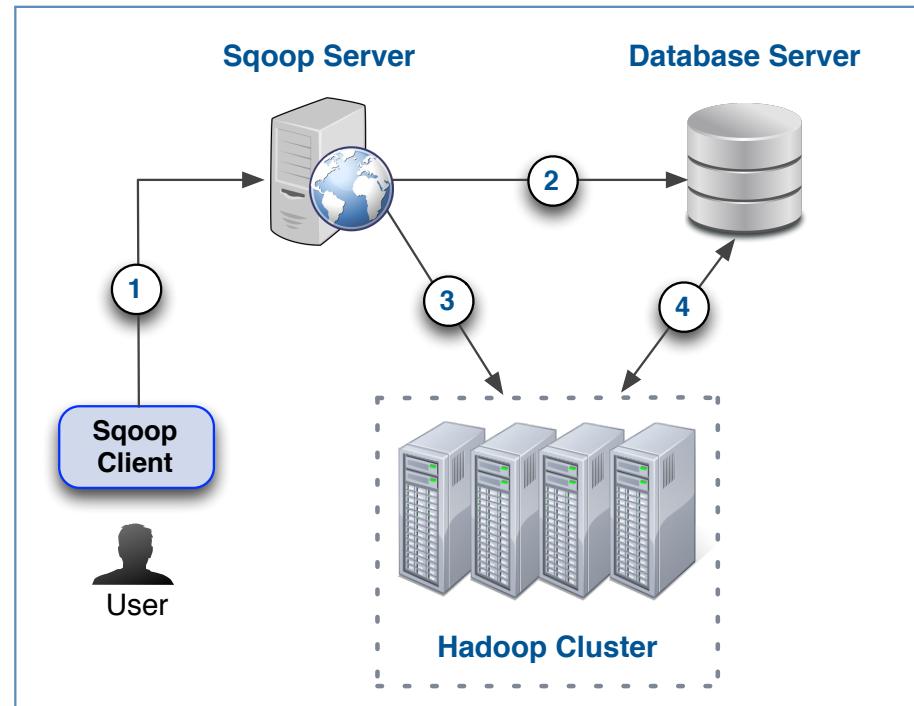
Limitations of Sqoop

- **Sqoop is stable and has been used successfully in production for years**
- **However, its client-side architecture does impose some limitations**
 - Requires connectivity to the RDBMS from the client
 - Requires connectivity to the cluster from the client
 - Requires the user to specify the RDBMS username and password
 - Difficult to integrate a CLI within external applications
- **Also tightly coupled to JDBC semantics**
 - A problem for NoSQL databases



Sqoop 2 Architecture

- **Sqoop 2 is the next-generation version of Sqoop**
 - Client-server design addresses the limitations described earlier
 - API changes also simplify development of other Sqoop connectors
- **Client requires connectivity only to the Sqoop server**
 - DB connections are configured on the server by a system administrator
 - End users no longer need to possess database credentials
 - Centralized audit trail
 - Better resource management
 - Sqoop server is accessible via CLI, REST API, and Web UI



Sqoop 2 Status

- **Sqoop 2 is being actively developed**
 - It began shipping (alongside Sqoop) starting in CDH 4.2.0
- **Sqoop 2 is not yet at feature parity with Sqoop**
 - Implemented features are regarded as stable
 - Consider using Sqoop 2 unless you require a feature it lacks
- **We use Sqoop, rather than Sqoop 2, in this class**
 - Primarily due to memory constraints in the VM

Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- Basic Imports and Exports
- Limiting Results
- Improving Sqoop's Performance
- Sqoop 2
- **Essential Points**
- Hands-On Exercise: Importing Customer Account Data

Bibliography

The following offer more information on topics discussed in this chapter

- **Sqoop User Guide**

- <http://tiny.cloudera.com/adcc05a>

- ***Apache Sqoop Cookbook* (published by O'Reilly)**

- <http://tiny.cloudera.com/adcc05b>

- **A New Generation of Data Transfer Tools for Hadoop: Sqoop 2**

- <http://tiny.cloudera.com/adcc05c>

Essential Points

- **Sqoop exchanges data between a database and the Hadoop cluster**
 - Provides subcommands (*tools*) for importing, exporting, and more
- **Tables are imported using Map-only jobs**
 - These are written as comma-delimited text by default
 - You can specify alternate delimiters or file formats
 - Uncompressed by default, but you can specify a codec to use
- **Sqoop provides many options to control imports**
 - You can select only certain columns or limit rows
 - Supports joins in free-form queries
- **Sqoop 2 is the next-generation version of Sqoop**
 - Client-server design improves administration and resource management

Chapter Topics

Importing Relational Data with Apache Sqoop

- Sqoop Overview
- Basic Imports and Exports
- Limiting Results
- Improving Sqoop's Performance
- Sqoop 2
- Essential Points
- **Hands-On Exercise: Importing Customer Account Data**

Hands-On Exercise: Importing Customer Account Data

- In this Hands-On Exercise, you will use Sqoop to perform incremental imports on customer account data
 - Please refer to the Hands-On Exercise Manual for instructions

Capturing Data with Apache Flume

Chapter 6



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- **Capturing Data with Apache Flume**
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Capturing Data with Apache Flume

In this chapter you will learn

- What are the main architectural components of Flume
- How these components are configured
- How to launch a Flume agent
- How to configure a standard Java application to log data using Flume

Chapter Topics

Capturing Data with Apache Flume

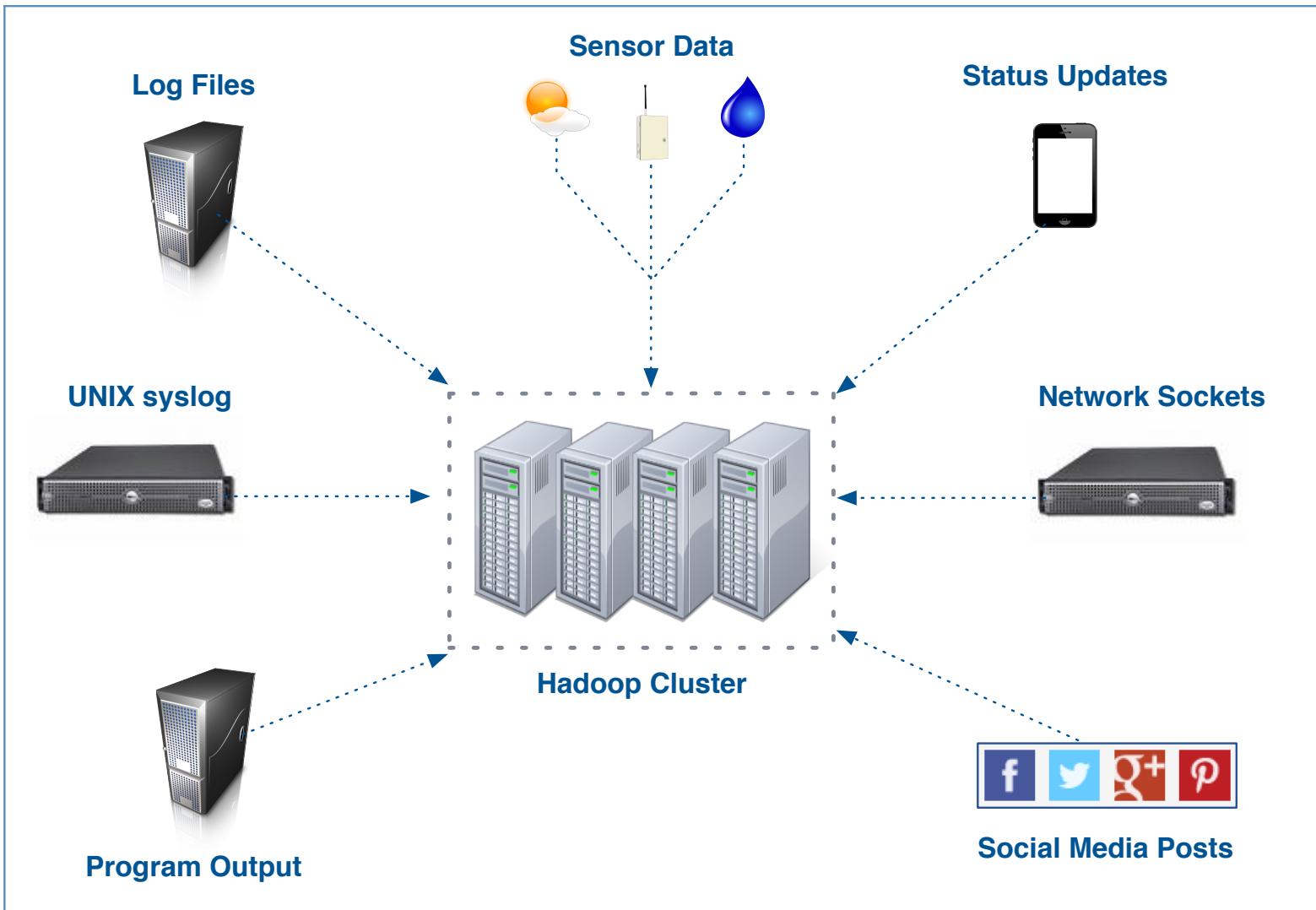
- **What is Apache Flume?**
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

What Is Apache Flume?

- **Apache Flume is a high-performance system for data collection**
 - Name derives from its original use case of near-real time log data ingestion
 - Now widely used for collection of any streaming event data
 - Supports aggregating data from many sources into HDFS
- **Originally developed by Cloudera**
 - Donated to Apache Software Foundation in 2011
 - Became a top-level Apache project in 2012
 - Flume OG gave way to Flume NG
- **Benefits of Flume**
 - Horizontally scalable
 - Extensible
 - Reliable

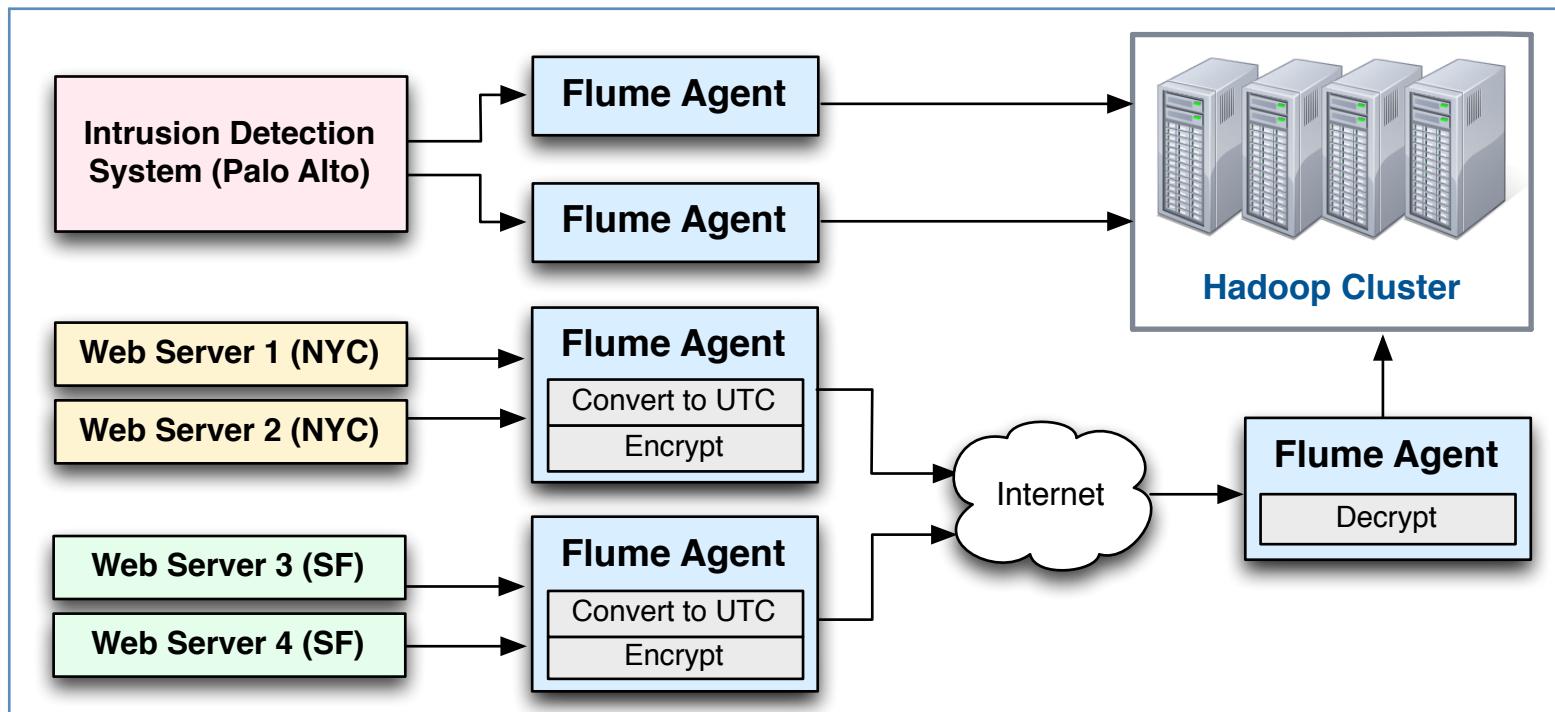


Common Flume Data Sources



Large-Scale Deployment Example

- Flume collects data using configurable “agents”
 - Agents can receive data from many sources, including other agents
 - Large-scale deployments use multiple tiers for scalability and reliability
 - Flume supports inspection and modification of in-flight data



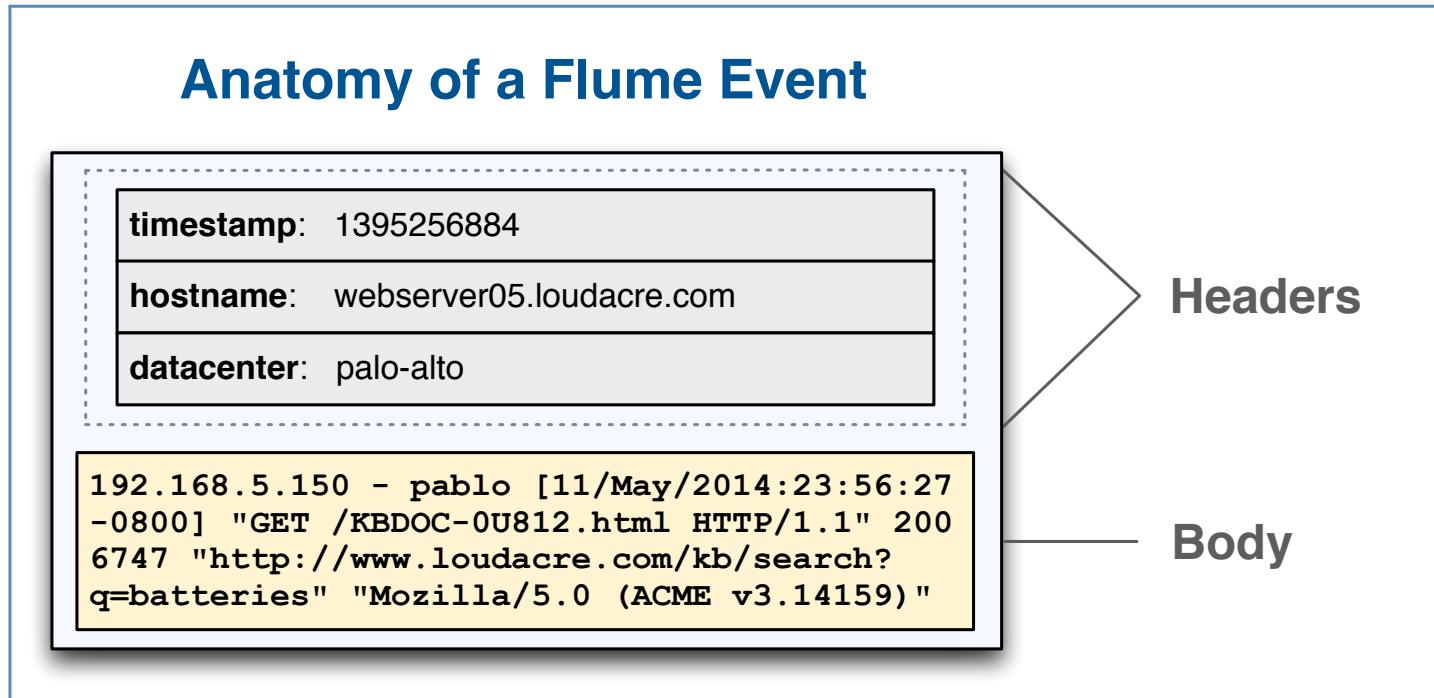
Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- **Basic Flume Architecture**
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Flume Events

- An **event** is the fundamental unit of data in Flume
 - Consists of a body (payload) and a collection of headers (metadata)
- Headers consist of name-value pairs
 - Headers are mainly used for directing output

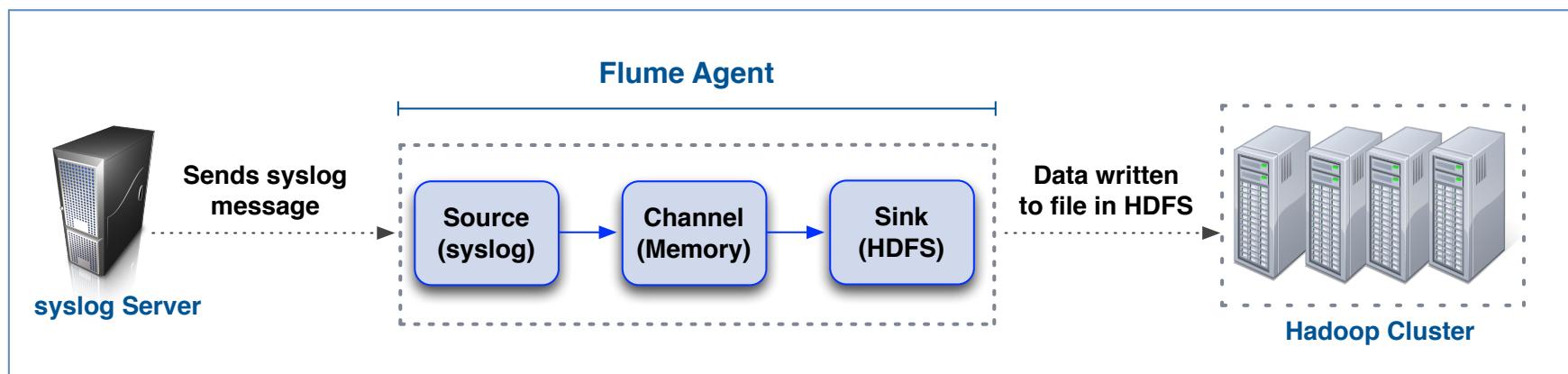


Components in Flume's Architecture

- **Source**
 - Receives events from the external actor that generates them
- **Sink**
 - Sends an event to its destination
- **Channel**
 - Buffers events from the source until they are drained by the sink
- **Agent**
 - Java process that configures and hosts the source, channel, and sink

Flume Data Flow

- This diagram illustrates how syslog data might be captured to HDFS
 - Message is logged on a server running a syslog daemon
 - Flume agent configured with syslog source receives event
 - Source pushes event to the channel, where it is buffered in memory
 - Sink pulls data from the channel and writes it to HDFS



Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- **Flume Sources**
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Notable Built-in Flume Sources

- **Syslog**
 - Captures messages from UNIX syslog daemon over the network
- **Netcat**
 - Captures any data written to a socket on an arbitrary TCP port
- **Exec**
 - Executes a UNIX program and reads events from standard output *
- **Spooldir**
 - Extracts events from files appearing in a specified (local) directory
- **HTTP Source**
 - Receives events from HTTP requests

* Asynchronous sources do not guarantee that events will be delivered

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- **Flume Sinks**
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Interesting Built-in Flume Sinks

- **Null**

- Discards all events ()Flume equivalent of /dev/null)

- **Logger**

- Logs event to INFO level using SLF4J

- **IRC**

- Sends event to a specified Internet Relay Chat channel

- **HDFS**

- Writes event to a file in the specified directory in HDFS

- **HBaseSink**

- Stores event in HBase

SLF4J: Simple Logging Façade for Java

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- **Flume Channels**
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Built-In Flume Channels

- **Memory**

- Stores events in the machine's RAM
 - Extremely fast, but not reliable (memory is volatile)

- **File**

- Stores events on the machine's local disk
 - Slower than RAM, but more reliable (data is written to disk)

- **JDBC**

- Stores events in a database table using JDBC
 - Slower than file channel

Chapter Topics

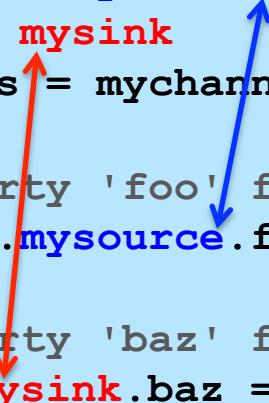
Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- **Flume Configuration**
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Flume Agent Configuration File

- Flume agent is configured through a Java properties file
 - Multiple agents can be configured in a single file
- The configuration file uses hierarchical references
 - Each component is assigned a user-defined ID
 - That ID is used in the names of additional properties

```
# Define sources, sinks, and channel for agent named 'agent1'  
agent1.sources = mysource  
agent1.sinks = mysink  
agent1.channels = mychannel  
  
# Sets a property 'foo' for the source associated with agent1  
agent1.sources.mysource.foo = bar  
  
# Sets a property 'baz' for the sink associated with agent1  
agent1.sinks.mysink.baz = bat
```



Configuring Flume Components

- Properties vary by component type (source, channel, and sink)
 - Properties also vary by subtype (e.g., netcat source vs. syslog source)
 - See the Flume user guide for full details on configuration

```
agent1.sources = src1
agent1.sinks = sink1
agent1.channels = ch1

agent1.channels.ch1.type = memory

agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/flume/incoming
agent1.sources.src1.channel = ch1

agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata
agent1.sinks.sink1.channel = ch1
```

Aside: HDFS Sink Configuration

- Path may contain patterns based on event headers, such as timestamp
- The HDFS sink writes uncompressed SequenceFiles by default
 - Specifying a codec will enable compression

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.codec = snappy
agent1.sinks.sink1.channel = ch1
```

- Setting fileType parameter to DataStream writes *raw* data
 - Can also specify a file extension, if desired

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .txt
agent1.sinks.sink1.channel = ch1
```

Starting a Flume Agent

- **Typical command line invocation**

- The `--name` argument must match the agent's name in the configuration file
 - Setting root logger as shown will display log messages in the terminal

```
$ sudo flume-ng agent \
  --conf /etc/flume-ng/conf \
  --conf-file /path/to/flume.conf \
  --name agent1 \
  -Dflume.root.logger=INFO,console
```

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- **Hands-On Exercise: Collecting Web Server Logs with Flume**
- Logging Application Events to Hadoop
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Hands-On Exercise: Collecting Web Server Logs with Flume

- In this Hands-On Exercise, you will configure Flume to ingest Web server log data to HDFS
 - Please refer to the Hands-On Exercise Manual for instructions

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- **Logging Application Events to Hadoop**
- Essential Points
- Hands-On Exercise: Collecting Application Data with Flume

Logging Application Events with Log4J

- Log4J is a popular logging library for Java
 - Alternative to using System.out.println throughout your code
 - Write log messages using different priority levels

```
import org.apache.log4j.Logger;

public class App {
    private static final Logger logger = Logger.getLogger(App.class);

    public static void main(String[] args) {
        logger.debug("App starting up");

        if (args.length == 0) {
            RuntimeException e = new IllegalArgumentException();
            logger.error("No argument specified", e);
            throw e;
        }
        logger.info("Invoked with: " + args[0]);
        // additional application code would follow...
    }
}
```

Log4J Configuration

- Log4J is typically configured using a properties file
 - Named `log4j.properties` and placed in the classpath at runtime
- Declaratively specifies details of how messages are logged
 - Message destination (appender)
 - Message format (layout)

```
# Log messages using a single appender named 'stdout'  
# Messages lower than INFO level (DEBUG or TRACE) are ignored  
log4j.rootLogger=INFO,stdout  
  
# Define the 'stdout' appender, which logs message to standard  
# output, preceded by the date/time and followed by a newline  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target=System.out  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd  
HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Log4J Example Output

- The following shows output from the sample application
 - When invoked with the required argument

```
2014-03-14 16:43:29 INFO App:14 - Invoked with: Cloudera
```

- When invoked without the required argument

```
2014-03-14 16:45:23 ERROR App:12 - No argument specified
Exception in thread "main" java.lang.IllegalArgumentException
java.lang.IllegalArgumentException
at App.main(App.java:11)
```

Log4J Flume Appender (1)

- Flume ships with a Log4J appender
 - Allows you to easily retrofit Java applications to log to HDFS via Flume
 - Does not require any changes to existing Java code
- Add these two dependencies to your application's `pom.xml` file

```
<dependency>
    <groupId>org.apache.flume</groupId>
    <artifactId>flume-ng-core</artifactId>
    <version>${flume.version}</version>
</dependency>
<dependency>
    <artifactId>flume-ng-log4jappender</artifactId>
    <groupId>org.apache.flume.flume-ng-clients</groupId>
    <version>${flume.version}</version>
</dependency>
```

Log4J Flume Appender (2)

- Next, edit `log4j.properties` to configure the Flume appender
 - Appender sends data to Flume using Avro RPC source (*not Avro format*)
 - Hostname and port must match your Flume agent's configuration
 - The layout pattern should *not* add a newline
 - Specify the package(s) and minimum log level that will log to Flume

```
# Define another appender that logs to Flume
log4j.appender.flume = org.apache.flume.clients.log4jappender.Log4jAppender
log4j.appender.flume.Hostname = dev.loudacre.com
log4j.appender.flume.Port = 12345

# Note that no newline (%n) is configured in this layout
log4j.appender.flume.layout=org.apache.log4j.PatternLayout
log4j.appender.flume.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} - %m

# Use the Flume appender only for messages logged from the
# com.cloudera.example package at the INFO level and higher
log4j.logger.com.cloudera.example = INFO,flume
```

Log4J Flume Appender (3)

- Next, create the Flume agent configuration file
 - Must use Avro source, but can use any channel or sink

```
agent1.sources = src1
agent1.sinks = sink1
agent1.channels = ch1

agent1.channels.ch1.type = memory

agent1.sources.src1.type = avro
agent1.sources.src1.bind = dev.loudacre.com
agent1.sources.src1.port = 12345
agent1.sources.src1.channels = ch1

# write data to HDFS (based on timestamp) as Snappy-compressed sequence files
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /user/training/logmessages/%y-%m-%d
agent1.sinks.sink1.hdfs.codec = snappy
agent1.sinks.sink1.channel = ch1
```

Log4J Flume Appender (4)

- Finally, start the Flume agent

- Must be started before your program
 - The agent must be running for the duration of your program

```
$ sudo flume-ng agent \
  --conf /etc/flume-ng/conf \
  --conf-file /path/to/flume.conf \
  --name agent1
```

- Limitations of the Flume Log4J appender

- Can only log strings, not Avro records
 - Logs data directly to files in HDFS, rather than a Kite data set

The Kite SDK Flume Module

- **The Kite SDK Data Module has a Flume sub-module**
 - Offers an enhanced version of the Log4J appender
- **Can log records *in Avro format* to a data set you have defined**
 - Access individual fields, rather than just a string
 - Immediately available for queries using Hive or Impala
 - If the Kite data set was defined using a `repo:hive` URI
- **This requires a few additional steps**
 - Must first create a data set
 - Must build Avro records according to the data set's schema
 - Must include `core-site.xml` and `hive-site.xml` config files

Log4J Configuration for the Kite SDK Flume Module

- Almost identical to the configuration used with the Log4J Flume appender
 - The class name of the appender is different
 - Since records are in Avro format, no layout is necessary
 - You must specify the name and repository URI of your data set

```
# Definition of root logger and unrelated appenders omitted

# Define another appender that logs to Flume
log4j.appenders.flume = org.kitesdk.data.flume.Log4jAppender
log4j.appenders.flume.Hostname = dev.loudacre.com
log4j.appenders.flume.Port = 12345
log4j.appenders.flume.DatasetRepositoryUri = repo:hive
log4j.appenders.flume.DatasetName = stationinfo

# Use the Flume appender only for messages logged from the
# com.cloudera.example package at the INFO level and higher
log4j.logger.com.cloudera.logkite = INFO,flume
```

Flume Agent Configuration for the Kite SDK Flume Module

- Almost identical to the configuration used with the Log4J Flume appender

```
agent1.sources = src1
agent1.sinks = sink1
agent1.channels = ch1

agent1.channels.ch1.type = memory

# receive log events over network via Avro RPC
agent1.sources.src1.type = avro
agent1.sources.src1.bind = dev.loudacre.com
agent1.sources.src1.port = 12345
agent1.sources.src1.channels = ch1

# Write data in Avro format to the location of the Hive-managed table
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /user/hive/warehouse/stationinfo
agent1.sinks.sink1.hdfs.fileSuffix = .avro
agent1.sinks.sink1.hdfs fileType = DataStream
agent1.sinks.sink1.serializer =
org.apache.flume.sink.hdfs.AvroEventSerializer$Builder
agent1.sinks.sink1.channel = ch1
```

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- **Essential Points**
- Hands-On Exercise: Collecting Application Data with Flume

Bibliography

The following offer more information on topics discussed in this chapter

- **Flume User Guide**

- <http://tiny.cloudera.com/adcc06a>

- **Log4J Web Site**

- <http://tiny.cloudera.com/adcc06b>

Essential Points

- **Apache Flume is a high-performance system for data collection**
 - Scalable, extensible, and reliable
- **A Flume agent manages the source, channels, and sink**
 - Source receives event data from its origin
 - Sink sends the event to its destination
 - Channel buffers events between the source and sink
- **The Flume agent is configured using a properties file**
 - Each component is given a user-defined ID
 - This ID is used to define properties of that component
- **Application events can be captured using Log4J**
 - Flume log appender is trivial to implement, but limited to strings
 - Kite supports logging to data sets in Avro format

Chapter Topics

Capturing Data with Apache Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Hands-On Exercise: Collecting Web Server Logs with Flume
- Logging Application Events to Hadoop
- Essential Points
- **Hands-On Exercise: Collecting Application Data with Flume**

Hands-On Exercise: Collecting Application Data with Flume

- In this Hands-On Exercise, you will use Log4J and Flume to capture data from a legacy application
 - Please refer to the Hands-On Exercise Manual for instructions

Developing Custom Flume Components

Chapter 7



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- **Developing Custom Flume Components**
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Developing Custom Flume Components

In this chapter you will learn

- **How data flows through a Flume agent**
- **What are some common extension points in Flume**
- **How the two types of Flume sources differ**
- **How to create a custom Flume source**
- **How to create a custom Flume interceptor**

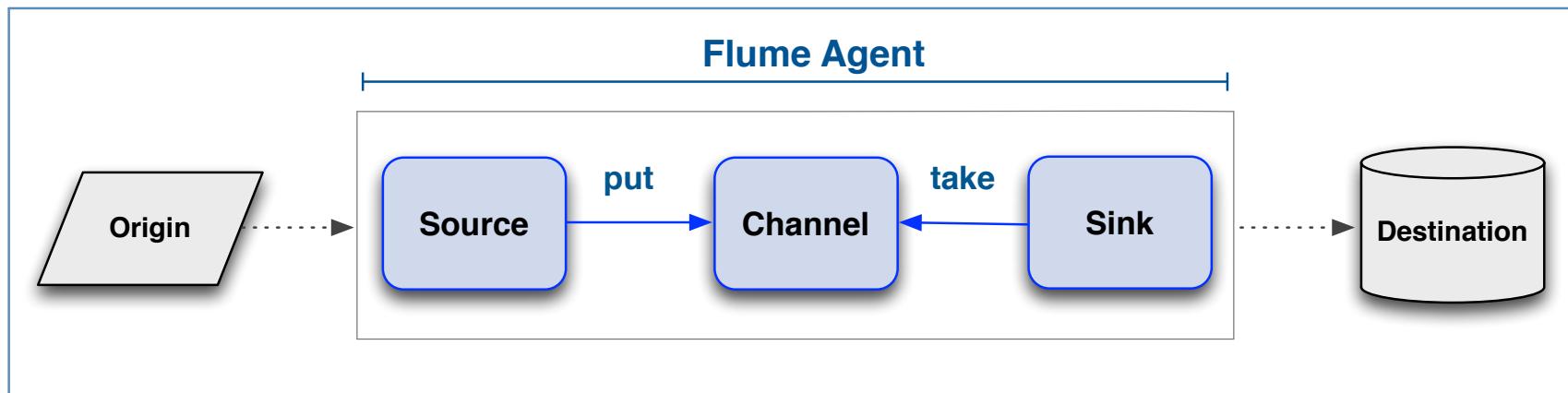
Chapter Topics

Developing Custom Flume Components

- **Flume Data Flow and Common Extension Points**
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Basic Flume Architecture Recap

- Data flows through Flume agent components to its destination
- The source knows how to read data from the origin
 - Source puts events into the channel
- A channel is a passive component that serves as a buffer
- The sink knows how to send data to its destination
 - Sink takes events from the channel

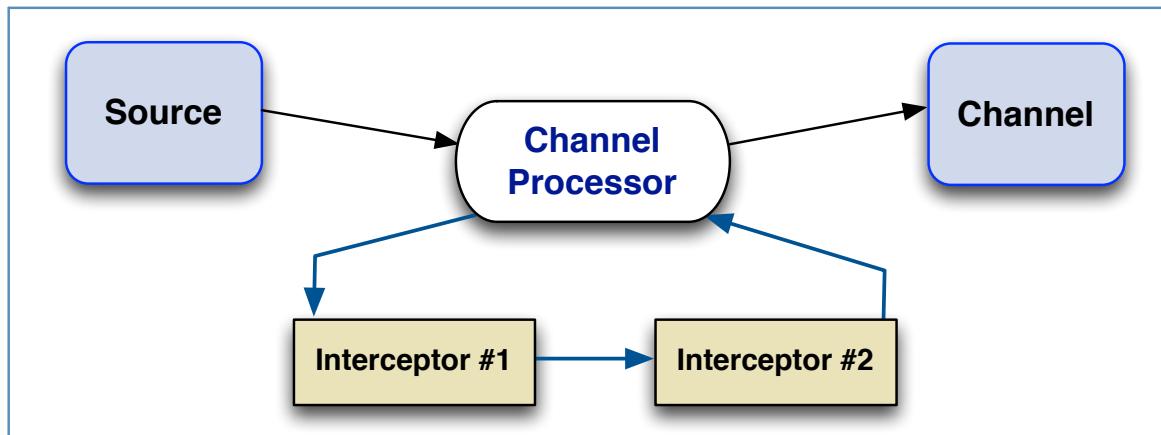


Data Flow Customizations

- **Flume comes with many built-in sources, sinks, and channels**
 - It is also possible to create your own implementations
- **Additionally, Flume supports a few other ways to customize data flow**
 - Can inspect, modify, or transform event data
 - Can control the format of messages written to the destination

Flume Interceptors

- **Interceptors support on-the-fly inspection and modification of events**
 - Useful for filtering or transformation
- **Often used to append or modify event headers**
 - Helpful when writing to HDFS paths based on timestamp or hostname
- **Invoked on events as they travel between a source and channel**
 - Multiple interceptors can be ‘chained’ together in a sequence
 - Coordinated by a Flume component called the *Channel Processor*



Built-in Flume Interceptors

- Flume has several built-in interceptors, such as:
 - Timestamp: adds a timestamp header to each event
 - Host: adds a header with the machine's hostname to each event
 - Regex Filtering: drops/passes event if the event body matches a pattern

```
# Example demonstrating how to configure a chain of two built-in
# interceptors (remainder of file omitted for brevity)
agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/flume/incoming
agent1.sources.src1.interceptors = inter1 inter2
agent1.sources.src1.interceptors.inter1.type = timestamp
agent1.sources.src1.interceptors.inter2.type = host
```

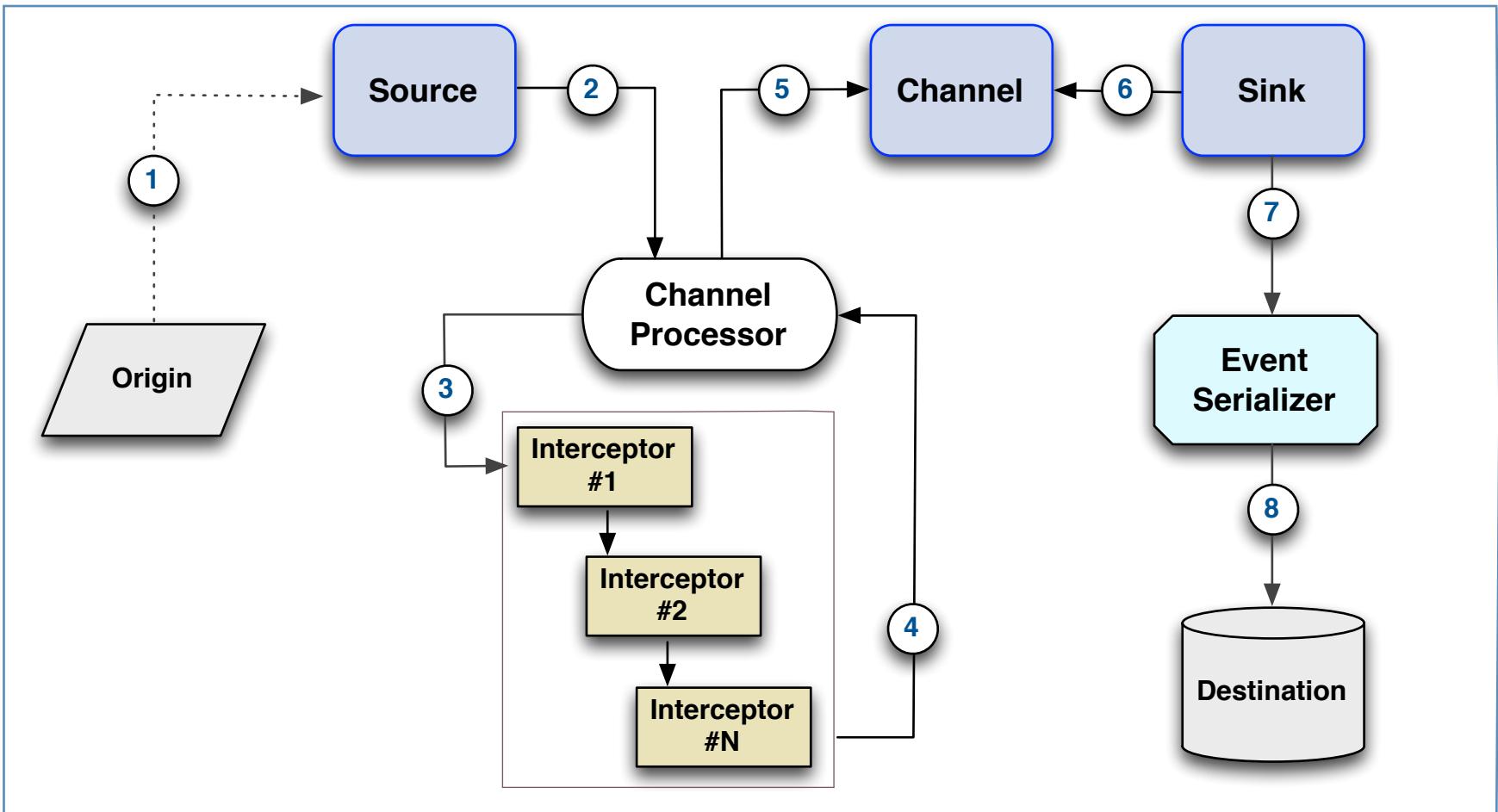
- You will see how to create a custom interceptor later in this chapter
 - You will also create one in the Hands-On Exercise

Event Serializers

- **Event serializers control the output format of event data**
 - The serializer is associated with a sink in the configuration file
 - Not all sinks support serializers
- **Example: specifying an event serializer with the HDFS sink**
 - The default serializer (`text`) writes only the event body
 - The `avro_event` serializer writes headers *and* body in Avro format

```
# remainder of config file omitted
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata
agent1.sinks.sink1.hdfs.fileSuffix = .avro
agent1.sinks.sink1.hdfs fileType = DataStream
agent1.sinks.sink1.serializer = avro_event
```

More Complete Flume Architecture



Common Extension Points

- Many components in the Flume data flow are pluggable
- Custom sources and interceptors are the most common
 - Covered later in this chapter
- Custom channels are the most complex by far
 - Luckily, it is almost never necessary to develop these!

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- **Custom Flume Sources**
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Developing a Custom Source in Flume

- **Custom sources implement the Source interface**
 - Typically by extending an abstract base class
 - Flume API has base classes for both event-driven and polling sources
- **Flume event processing uses transactional semantics for reliability**
 - Source or sink begins a transaction with the channel
 - Put or take one or more events
 - Source or sink commits if successful, or rolls back if failed
 - For a custom source, the ChannelProcessor handles this for us

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- **Developing a Flume Pollable Source**
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Pollable Source

- **The example below shows a skeletal PollableSource implementation**
 - Pollable sources are asked to look for data periodically
- **The process method does most of the work**
 - Flume invokes this method from within a loop
 - The source must create events and put them into the channel

```
public class MySource extends AbstractSource
    implements PollableSource, Configurable {

    public void configure(Context context) {}

    public void start() {}

    public void stop() {}

    public Status process() {}
}
```

Pollable Source Example (1)

- Following is a complete example of a pollable source
 - Source creates events when the number of files in a directory changes

```
package com.cloudera.example;

public class FileCountMonitorSource extends AbstractSource
    implements PollableSource, Configurable {

    private File dir;
    private int lastFileCount;

    @Override
    public void configure(Context context) throws ConfigurationException {
        // get the path property from the config file, default to /tmp
        String path = context.getString("path", "/tmp");
        dir = new File(path);

        if (! (dir.exists() && dir.isDirectory())) {
            throw new ConfigurationException("Invalid path specified");
        }
    }
}
```

Pollable Source Example (2)

- **Optionally, you can override the `stop()` and `start()` methods**
 - Implement any necessary initialization and cleanup tasks here
 - Used here to store and retrieve file count
 - Be sure to call the superclass's methods if you implement your own

```
@Override
public void stop() {
    // called only when source is stopped: (do cleanup here)
    MyUtilityClass.storeCurrentValue(lastFileCount);

    super.stop();
}

@Override
public void start() {
    // called only when source is started: (do setup here)
    lastFileCount = MyUtilityClass.readPreviousValue();

    super.start();
}
```

Pollable Source Example (3)

- Finally, implement the business logic in the `process()` method

```
@Override
public Status process() throws EventDeliveryException {
    int count = dir.list().length;
    if (count == lastFileCount) {
        return Status.BACKOFF; // try again later
    }
    lastFileCount = count;

    String msg = "filecount:" + count;
    byte[] body = msg.getBytes();

    Map<String, String> headers = new HashMap<String, String>();
    headers.put("path", dir.getPath());
    Event event = EventBuilder.withBody(body, headers);

    getChannelProcessor().processEvent(event);

    return Status.READY;
}
```

Aside: Adding Custom Libraries to Flume NG

- Flume plugins, such as custom sources, are packaged as JAR files
- Must be made available in Flume's runtime classpath
 - Several possible ways to do this
 - Steps vary based on Flume version and cluster configuration
 - Consult your administrator regarding production deployment
- Recent versions of Flume support a `plugins.d` directory
 - This is the preferred approach for deployment
- Plugin directory is located at `/etc/flume-ng/plugins.d` in our VM
 - Create a subdirectory for your plugin, such as `filecountsource`
 - Then, create a subdirectory named `lib` below that
 - Copy your JAR, plus any dependencies, into this `lib` directory

Configuring a Custom Source

- Edit your Flume agent's configuration to use the new source

```
# remainder of config file omitted

# The type must be set to the fully-qualified class name of your source
agent1.sources.src1.type = com.cloudera.example.FileCountMonitorSource

# Afterwards, configure any source-specific properties
agent1.sources.src1.path = /home/training/Documents
```

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- **Developing a Flume Event-Driven Source**
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Event-Driven Sources

- Flume executes a pollable source's `process()` method in a loop
- In contrast, event-driven sources have some internal stimulus
 - This is typically a timer or application event
 - No need for a `process()` method
 - Implementation is otherwise very similar to pollable sources
- The following example shows an event-driven source
 - Generates random numbers at specific intervals
 - The interval defaults to 500 milliseconds, but is configurable
 - Uses a `java.util.Timer` instance for scheduling
- Follow the same steps as before for deployment and configuration

Event-Driven Source Example (1)

```
package com.cloudera.example;

public class RandomNumberSource extends AbstractSource
    implements Configurable, EventDrivenSource {

    private long interval;
    private Timer timer;

    @Override
    public void configure(Context context) throws FlumeException {
        this.interval = context.getLong("interval", 500L); // milliseconds
    }

    @Override
    public void start() throws FlumeException {
        timer = new Timer();
        timer.schedule(new RandomNumberGeneratorTask(), 0, interval);
        super.start();
    }
}
```

Event-Driven Source Example (2)

```
@Override  
public void stop() throws FlumeException {  
    if (timer != null) {  
        timer.cancel();  
    }  
  
    super.stop();  
}  
  
class RandomNumberGeneratorTask extends TimerTask {  
  
    @Override  
    public void run() {  
        byte[] body = String.valueOf(Math.random()).getBytes();  
        Event event = EventBuilder.withBody(body);  
  
        getChannelProcessor().processEvent(event);  
    }  
}
```

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- **Custom Flume Interceptors**
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

The Interceptor Interface

- All interceptors implement the `Interceptor` interface
 - Flume API has no base classes for interceptors
- Each event is passed to the `intercept(Event)` method
 - Inspect, modify, or filter events
 - Process batches in the `intercept(List<Event>)` method
- You must also create a class that implements `Interceptor.Builder`
 - Typically implemented as a static inner class
 - Required to have a public zero-argument constructor
 - Must instantiate, configure, and return your interceptor

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- **Developing a Header-Modifying Flume Interceptor**
- Developing a Filtering Flume Interceptor
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Header-Adding Interceptor (1)

- The following example adds a header named `length`
 - Value is set to the number of bytes in the payload

```
package com.cloudera.example;

public class LengthHeaderInterceptor implements Interceptor {

    public static class Builder implements Interceptor.Builder {
        @Override
        public Interceptor build() {
            return new LengthHeaderInterceptor();
        }

        @Override
        public void configure(Context context) {
            // nothing to configure in this example
        }
    }

    @Override
    public void initialize() { }

    @Override
    public void close() { }
```

Header-Adding Interceptor (2)

```
@Override  
public Event intercept(Event event) {  
    int len = event.getBody().length;  
  
    event.getHeaders().put("length", String.valueOf(len));  
  
    return event;  
}  
  
@Override  
public List<Event> intercept(List<Event> events) {  
    List<Event> result = new ArrayList<Event>();  
    for (Event event : events) {  
        Event intercepted = intercept(event);  
        result.add(intercepted);  
    }  
  
    return result;  
}  
}
```

Custom Interceptor Deployment and Configuration

- Deployment process is identical to that for custom sources
 - Compile the code, package the JAR, and place it in Flume's classpath
- Configuration is similar to built-in interceptors
 - The type property must specify the fully-qualified class name of builder

```
# remainder of config file omitted
agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/flume/incoming
agent1.sources.src1.interceptors = inter1
agent1.sources.src1.interceptors.inter1.type =
    com.cloudera.example.LengthHeaderInterceptor$Builder
```



The value should be on the same line as the property name

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- **Developing a Filtering Flume Interceptor**
- Essential Points
- Hands-On Exercise: Writing a Custom Flume Interceptor

Filtering Interceptors

- **Interceptors are also used to filter incoming events**
 - Using any criteria the developer chooses
- **Filtering interceptors also implement the `Interceptor` interface**
 - Return `null` from `intercept(Event)` to filter out an event
 - Do not return `null` from the `intercept(List<Event>)` method
 - Return an empty list if all events are filtered out
 - Individual values can be null
- **The following example filters out events at random**
 - Might be useful for creating a data sample for testing purposes
 - The percentage of records to pass is configurable

Filtering Interceptor (1)

```
package com.cloudera.example;

public class RandomSamplingInterceptor implements Interceptor {
    private final float passPercent;

    public static class Builder implements Interceptor.Builder {
        private float passPercent;

        @Override
        public void configure(Context ctx) throws ConfigurationException {
            String pctString = ctx.getString("passPercent", "0.01");
            pct = Float.valueOf(pctString);

            if (pct < 0 || pct > 1.0) {
                throw new ConfigurationException("Percentage is out of range");
            }
        }

        @Override
        public Interceptor build() {
            return new RandomSamplingInterceptor(pct);
        }
    }
}
```

Filtering Interceptor (2)

```
private RandomSamplingInterceptor(float passPercent) {
    this.passPercent = passPercent;
}
// initialize() and close() omitted for brevity

@Override
public Event intercept(Event event) {
    if (Math.random() > passPercent) {
        return null;
    }
    return event;
}

@Override
public List<Event> intercept(List<Event> events) {
    List<Event> result = Lists.newArrayList();
    for (Event event : events) {
        Event intercepted = intercept(event);
        result.add(intercepted);
    }
    return result;
}
}
```

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- **Essential Points**
- Hands-On Exercise: Writing a Custom Flume Interceptor

Bibliography

The following offer more information on topics discussed in this chapter

- **Architecture of Flume NG**

- <http://tiny.cloudera.com/adcc07a>

- **Flume Developers Guide**

- <http://tiny.cloudera.com/adcc07b>

Essential Points

- **Data flows through Flume agent components to its destination**
 - Sources put events into a channel
 - Sinks take events from a channel
 - Channels are passive components
 - Event serializers control the output format
- **Flume has two main types of sources**
 - Pollable sources are executed in a loop
 - Event-driven sources use an internal stimulus
- **Interceptors process messages between the source and channel**
 - Useful for filtering and transforming events
 - Interceptors can be chained in the configuration file

Chapter Topics

Developing Custom Flume Components

- Flume Data Flow and Common Extension Points
- Custom Flume Sources
- Developing a Flume Pollable Source
- Developing a Flume Event-Driven Source
- Custom Flume Interceptors
- Developing a Header-Modifying Flume Interceptor
- Developing a Filtering Flume Interceptor
- Essential Points
- **Hands-On Exercise: Writing a Custom Flume Interceptor**

Hands-On Exercise: Writing a Custom Flume Interceptor

- In this Hands-On Exercise, you will write a custom interceptor for Flume that will process incoming device status records
 - Please refer to the Hands-On Exercise Manual for instructions

Managing Workflows with Apache Oozie

Chapter 8



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- **Managing Workflows with Apache Oozie**
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Managing Workflows with Apache Oozie

In this chapter you will learn

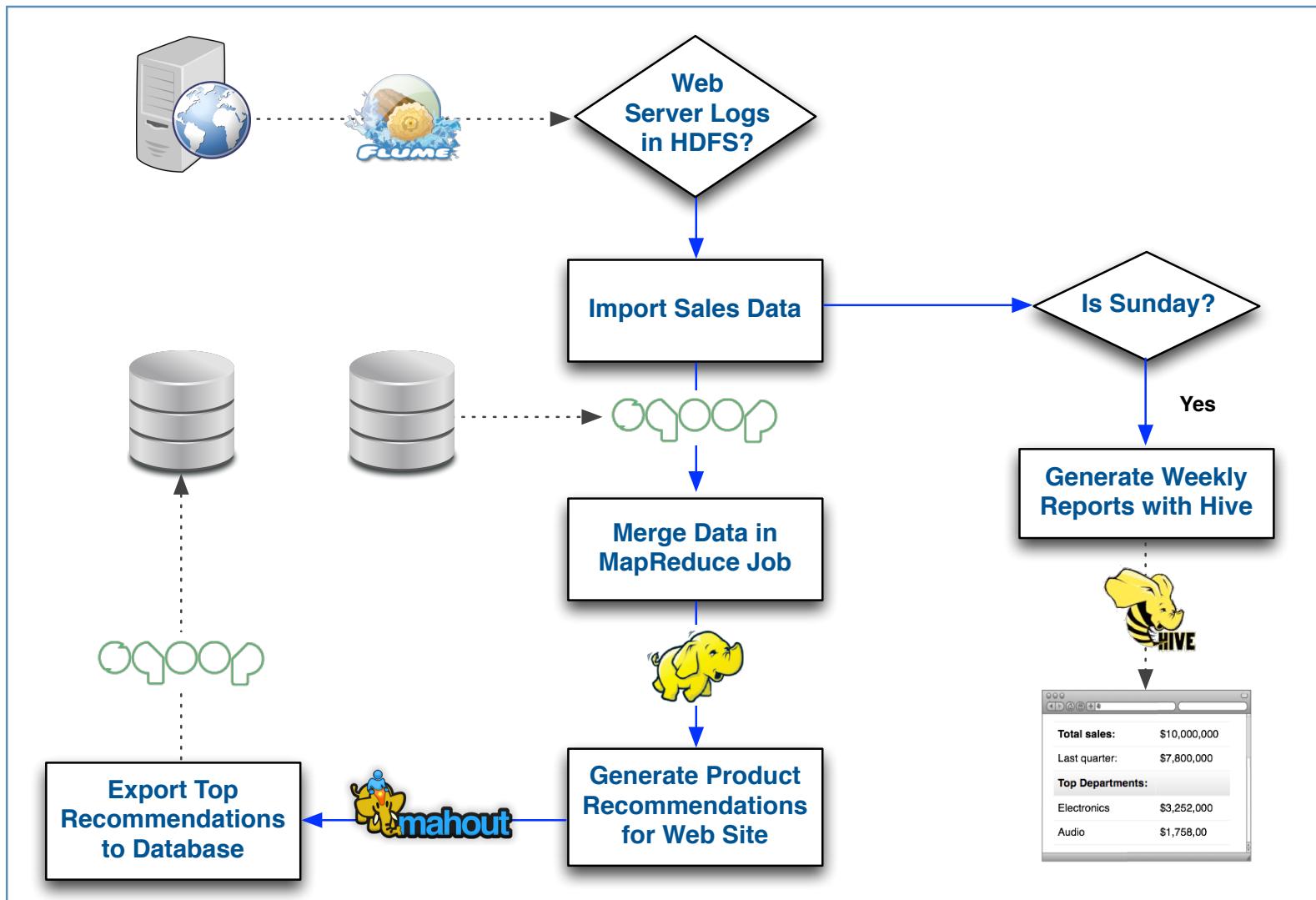
- How Apache Oozie helps manage the execution of complex Hadoop jobs
- What commonly-used actions are supported in Oozie
- How to create an Oozie workflow using XML or the Hue Web UI
- How to package, deploy, and execute an Oozie workflow
- How to schedule recurring workflows

Chapter Topics

Managing Workflows with Apache Oozie

- **The Need for Workflow Management**
- What is Apache Oozie?
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Example Workflow



Managing Complex Workflows

- **The previous slide illustrated a complex but realistic workflow**
 - Acquires data from multiple sources
 - Processes it using diverse tools, as appropriate for the task
 - Involves decision points and forks
 - Probably runs on a recurring schedule
- **What initiates this workflow?**
 - Manual processes are tedious and error prone
 - Automation saves time and helps ensure consistent results
- **However, even automated processes may fail**
 - We must ensure that failures are handled properly
- **Orchestrating such a workflow can be challenging without special tools**

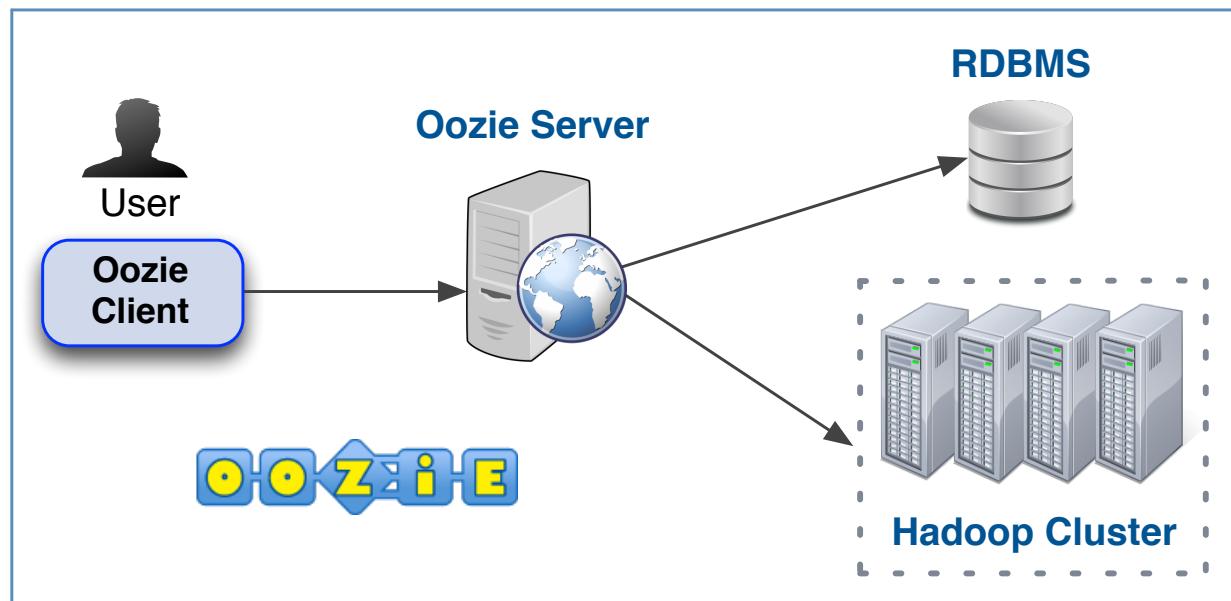
Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- **What is Apache Oozie?**
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

What is Apache Oozie?

- **Oozie is a versatile workflow engine for the Hadoop ecosystem**
- **Implemented as a client-server system**
 - Client submits the workflow definition and required libraries to the Oozie server
 - Oozie server submits individual jobs to Hadoop based on this workflow
 - Client can obtain job status and manage jobs through the Oozie server



Oozie Terminology

- **Action: a single step**
 - Examples: importing a table in Sqoop or executing a MapReduce job
- **Control flow: the sequence of execution**
 - Examples: starting a job, ending a job, or running an action conditionally
- **Workflow: a collection of actions and control flows**
 - Each Oozie workflow is a directed acyclic graph (DAG)
- **Coordinator: provides support for automatic workflow execution**
 - Can run workflows at defined intervals or based on the availability of input data
- **Bundle: a collection of coordinators**
 - Allows one to manage them as a single unit
- **Two types of *nodes* in a workflow: action and control flow**

Chapter Topics

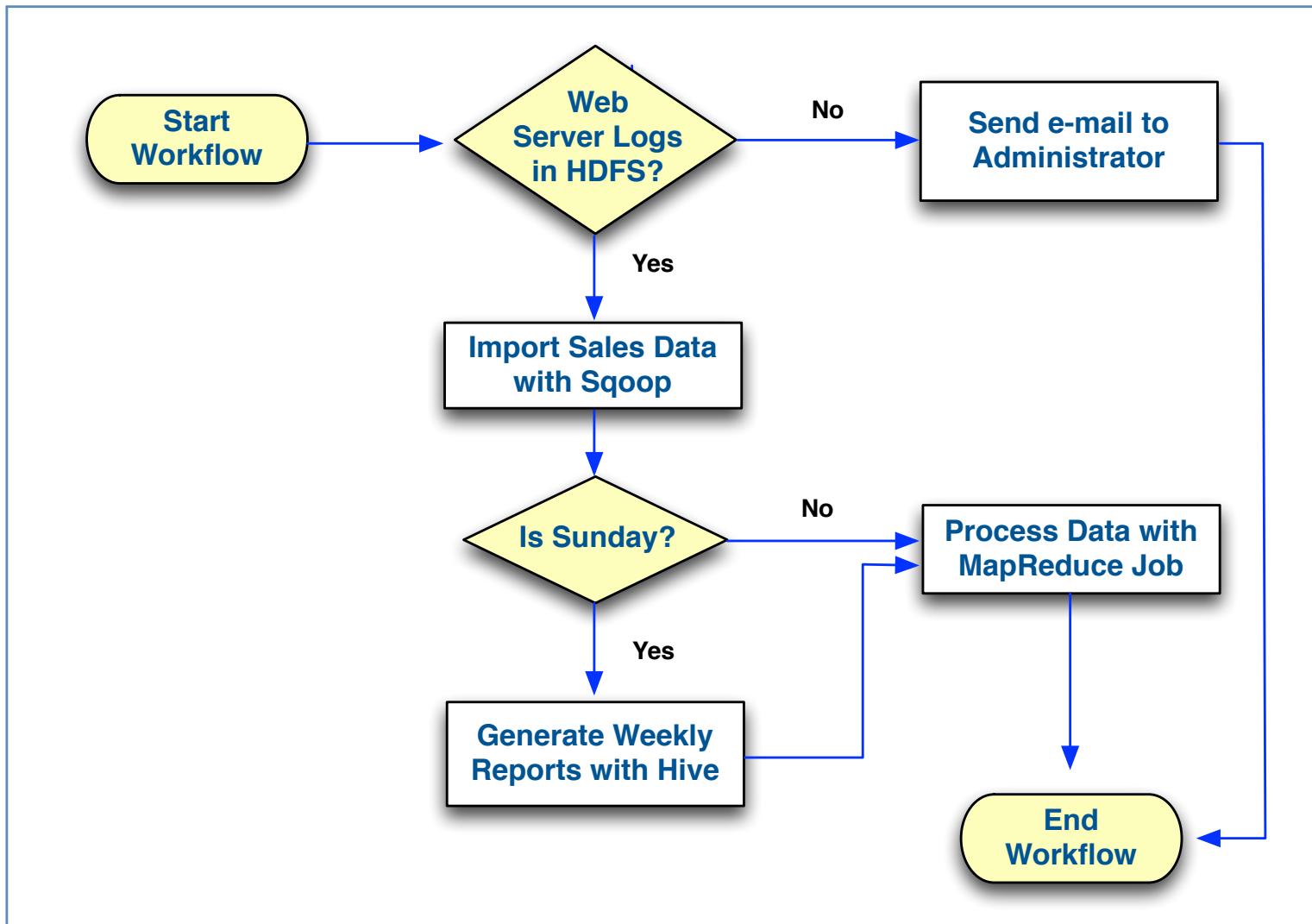
Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- **Defining an Oozie Workflow**
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Control Flow Nodes

- ***Control flow nodes*** are used to define the execution path for a workflow
 - Starting, ending, or killing jobs
 - Running jobs conditionally based on some criteria
- Examples of how control flow nodes are used
 - Initiating jobs based on the presence of data in a directory in HDFS
 - Forking several jobs and running another action after they have all completed
 - Killing downstream jobs if the upstream job failed

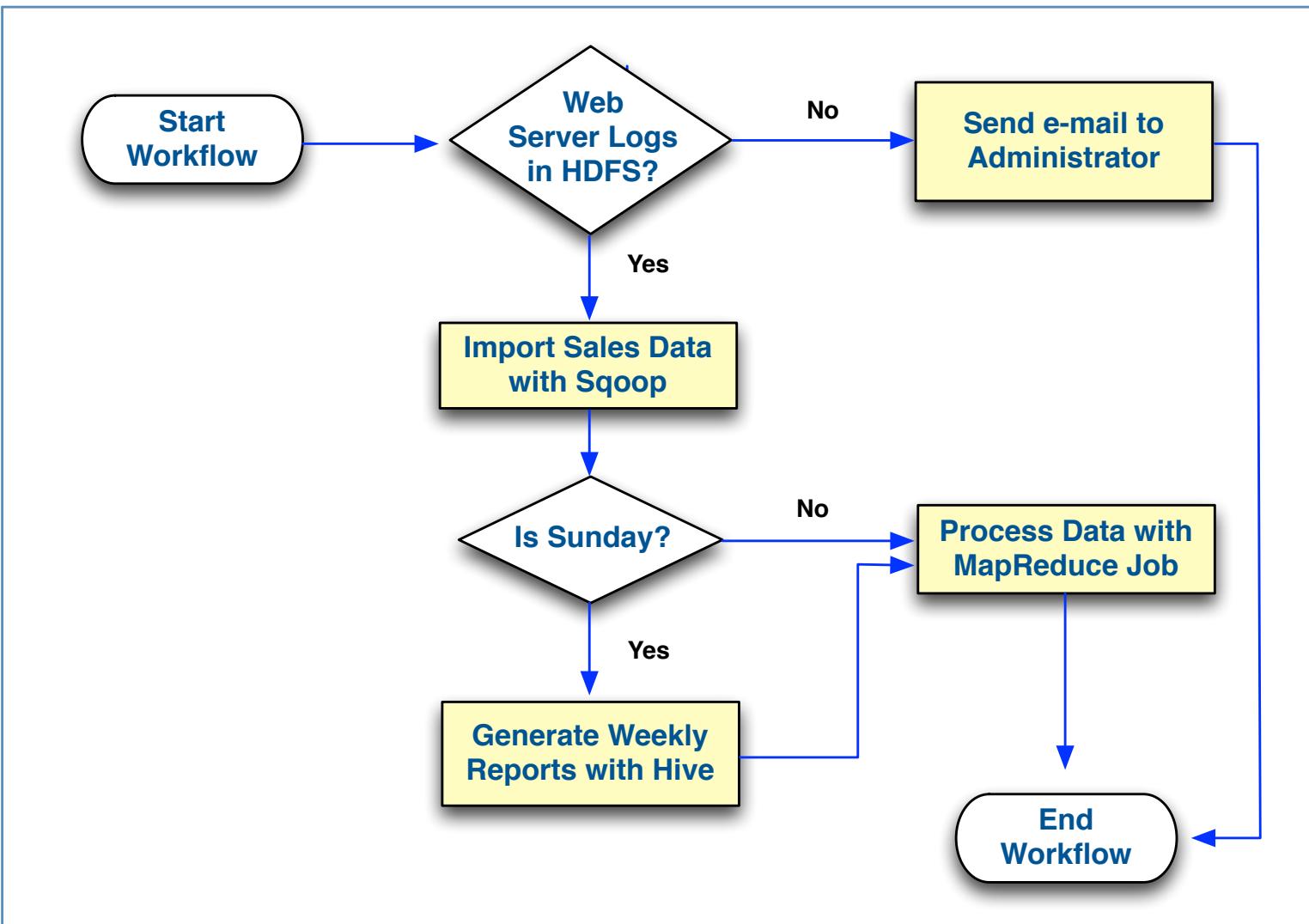
Control Flow Nodes in an Oozie Workflow



Actions Supported in Oozie

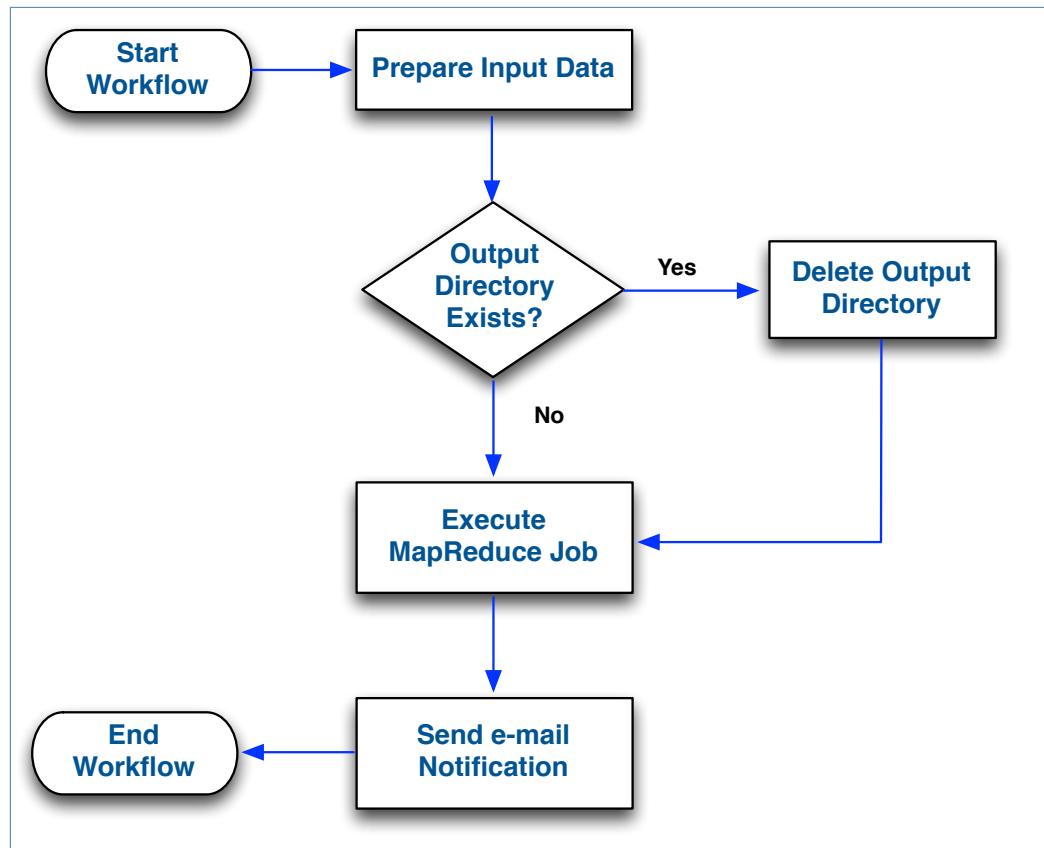
- Based on the flow of control, Oozie can run one or more *action nodes*
- Oozie supports many action types, including:
 - Executing MapReduce jobs
 - Running Pig or Hive scripts
 - Executing standard Java or shell programs
 - Manipulating data via HDFS commands
 - Running remote commands with SSH
 - Sending e-mail messages
- Most actions are executed from a MapReduce job on the Hadoop cluster
 - Caution: “local” filesystem paths do not refer to the *client’s* filesystem

Action Nodes in an Oozie Workflow



Workflow Example (1)

- The following example shows a basic but complete Oozie workflow
- Uses many of the features just discussed
 - Filesystem commands
 - Decision in control flow
 - MapReduce job
 - Notification via e-mail
- The next several slides will show how to create and run this workflow
- Oozie workflows are defined using an XML syntax



Workflow Example (2)

- The workflow is defined in an XML file named `workflow.xml`
 - We will look at overall structure first, then cover each section in detail

```
<workflow-app name="myworkflow" xmlns="uri:oozie:workflow:0.4">
  <start to="prepInputData"/>

  <action name="prepInputData">
    <!-- details shown later -->
    <ok to="checkOutputDirectory"/>
    <error to="kill"/>
  </action>

  <decision name="checkOutputDirectory">
    <switch>
      <case to="deleteOldOutput">EXPRESSION-TO-CHECK-DIRECTORY-EXISTENCE</case>
      <default to="runMapReduceJob"/>
    </switch>
  </decision>

  <action name="deleteOldOutput">
    <!-- details shown later -->
    <ok to="runMapReduceJob"/>
    <error to="kill"/>
  </action>
```

Cont'd...

Workflow Example (3)

- Here is the second half of the structural view of workflow.xml

```
<action name="runMapReduceJob">
    <!-- details shown later -->
    <ok to="notifyEmailSuccess"/>
    <error to="notifyEmailFailure"/>
</action>

<action name="notifyEmailSuccess">
    <!-- details shown later -->
    <ok to="end"/>
    <error to="end"/>
</action>

<action name="notifyEmailFailure">
    <!-- details shown later -->
    <ok to="end"/>
    <error to="end"/>
</action>

<kill name="kill">
    <!-- details shown later -->
</kill>

<end name="end"/>
</workflow-app>
```

Workflow Example (4)

- Next, we will examine the complete XML for each part of the workflow
- The version number in the XML namespace is significant
- start node is required and contains only a reference to another node
- The prepInputData node performs some filesystem (HDFS) actions
 - Values like \${inputDir} are property references (more later)
 - Using properties instead of hardcoding values is a best practice

```
<workflow-app name="myworkflow" xmlns="uri:oozie:workflow:0.4">
  <start to="prepInputData"/>

  <action name="prepInputData">
    <fs>
      <mkdir path='${inputDir}'/>
      <move source='${dataFile}' target="${inputDir}" />
    </fs>
    <ok to="checkOutputDirectory"/>
    <error to="kill"/>
  </action>
```

Workflow Example (5)

- A decision node is similar to a switch statement in Java
 - Supports one default element and any number of case elements
- Each case element contains a JSP Expression Language (EL) predicate
 - Evaluated in order of appearance until one evaluates as true
 - The default element is selected if no predicate evaluates as true

```
<decision name="checkOutputDirectory">
    <switch>
        <case to="deleteOldOutput">${fs:exists(outputDir) == "true"}</case>
        <default to="runMapReduceJob"/>
    </switch>
</decision>

<action name="deleteOldOutput">
    <fs>
        <delete path='${outputDir}'/>
    </fs>
    <ok to="runMapReduceJob"/>
    <error to="kill"/>
</action>
```

Aside: More EL Examples

- Here are some more examples of EL expressions
 - See documentation for a complete list of functions, constants, and tests

Expression	Description
<code> \${fs:fileSize(inputFile) gt 5 * GB}</code>	True if the file in HDFS referenced by the <code>inputFile</code> property is larger than 5 GB
<code> \${fs:fileSize('/tmp/x.txt') gt 5 * GB}</code>	Similar, but uses a literal path
<code> \${fs:fileSize(inputFile) ge 64 * MB}</code>	True if the file in HDFS referenced by the <code>inputFile</code> property is larger than or equal to 64 MB.
<code> \${fs:dirSize(inputDir) lt 3 * TB}</code>	True if all files in the HDFS directory referenced by the <code>inputDir</code> property are collectively smaller than 3 TB. This does not include any subdirectories.
<code> \${fs:isDir('/user/alice/data')}</code>	True if this path refers to a directory.

Workflow Example (6)

- **The map-reduce action executes a MapReduce (or Map-only) job**
 - Oozie does not require a “driver” class
 - The properties set here are equivalent to methods called by your driver

```
<action name="runMapReduceJob">
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <configuration>
      <property>
        <name>mapred.input.dir</name>
        <value>${inputDir}</value>
      </property>
      <property>
        <name>mapred.output.dir</name>
        <value>${outputDir}</value>
      </property>
      <!-- additional properties are described on the next slide -->
    </configuration>
  </map-reduce>
  <ok to="notifyEmailSuccess"/>
  <error to="notifyEmailFailure"/>
</action>
```

Workflow Example (7)

- **The previous slides showed how to specify input and output paths**
 - Here are other properties set based on the driver replaced by Oozie
 - Some property names and values may differ if using the MapReduce Old API

Property Name	Property Value
mapreduce.map.class	com.cloudera.example.MyMapper
mapreduce.reduce.class	com.cloudera.example.MyReducer
mapred.mapoutput.key.class	org.apache.hadoop.io.Text
mapred.mapoutput.value.class	org.apache.hadoop.io.IntWritable
mapred.mapper.new-api	true
mapred.reducer.new-api	true

Aside: The <java> Action

- This action is used to execute an arbitrary Java program
 - Content of <java-opts> is passed to the JVM
 - Content of each <arg> element is passed to the class's main method

```
<action name="my-java-action">
  <java>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <main-class>com.loudacre.example.MyClass</main-class>
    <java-opts>-Dmode=debug -Denvironment=dev -Xmx1024m</java-opts>
    <arg>${myFirstArgument}</arg>
    <arg>${mySecondArgument}</arg>
  </java>
  <ok to="my-next-action"/>
  <error to="fail"/>
</action>
```

Aside: Specifying Configuration Properties in an External File

- Many of these properties will be the same for each map-reduce action
- The job-xml element is an alternative to duplicating these properties
 - Properties defined in external XML file, but inserted at runtime

`workflow.xml`

```
<map-reduce>
  <job-tracker>${jobTracker}</job-tracker>
  <name-node>${nameNode}</name-node>
  <job-xml>config.xml</job-xml>←
  <configuration>
    <property>
      <name>mapred.input.dir</name>
      <value>${inputDir}</value>
    </property>
    <property>
      <name>mapred.output.dir</name>
      <value>${outputDir}</value>
    </property>
  </configuration>
</map-reduce>
```

`config.xml`

```
<configuration>
  <property>
    <name>mapred.mapper.new-api</name>
    <value>true</value>
  </property>
  <property>
    <name>mapred.reducer.new-api</name>
    <value>true</value>
  </property>
</configuration>
```

Workflow Example (8)

- The `email` action is an example of an Oozie extension
 - Extensions specify a namespace with a version number
 - Prerequisite: admin must first configure mail server properties in Oozie

```
<action name="notifyEmailSuccess">
    <email xmlns="uri:oozie:email-action:0.1">
        <to>${recipientEmailAddress}</to>
        <subject>Oozie Workflow Completed Successfully</subject>
        <body>Workflow ${wf:id()} is done; see output at ${outputDir}</body>
    </email>
    <ok to="end"/>
    <error to="end"/>
</action>

<action name="notifyEmailFailure">
    <email xmlns="uri:oozie:email-action:0.1">
        <to>${recipientEmailAddress}</to>
        <subject>Oozie Workflow Failed</subject>
        <body>Workflow ${wf:id()} failed. The error message associated with
              the failure is: ${wf:errorMessage(wf:lastErrorNode())}.</body>
    </email>
    <ok to="end"/>
    <error to="end"/>
</action>
```

Workflow Example (9)

- **The kill node specifies what happens when the job is killed**
 - In this case, we output an error message (will be shown in client)
 - The Oozie EL functions shown in the example below extract information about the last failure
- **The end node represents the ultimate destination of the workflow**
 - Like the start node, it is required but has no content

```
<kill name="kill">
    <message>Action failed, error message[$wf:errorMessage(wf:lastErrorNode())]
    </message>
</kill>

<end name="end"/>
</workflow-app>
```

The <prepare> Element

- **MapReduce jobs fail if the output directory already exists in HDFS**
 - Other types of jobs may require that a specific directory *does* exist
- **Most actions support <prepare> following <name-node>**
 - Performs basic HDFS tasks before the action is executed
 - Only <mkdir> and <delete> are supported
 - Simplifies workflow for common cases

```
<action name="run-mapreduce-job">
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
      <delete path="${outputDir}"/>
      <mkdir path="${myTempDir}"/>
    </prepare>
```

Using Sqoop from Oozie

- The `<sqoop>` action is an extension, and requires its own namespace
- Specify all Sqoop arguments in a single `<command>` element

```
<sqoop xmlns="uri:oozie:sqoop-action:0.2">
  <job-tracker>${jobTracker}</job-tracker>
  <name-node>${nameNode}</name-node>
  <command>import --connect jdbc:mysql://dev.loudacre.com/loudacre
--table basestations --username training --password training --target-dir
${tablePath} --fields-terminated-by "\t"</command>
</sqoop>
```

- Alternatively, specify each argument in its own `<arg>` element

```
<sqoop xmlns="uri:oozie:sqoop-action:0.2">
  <job-tracker>${jobTracker}</job-tracker>
  <name-node>${nameNode}</name-node>
  <arg>import</arg>
  <arg>--connect</arg>
  <arg>jdbc:mysql://dev.loudacre.com/loudacre</arg>
  <!-- additional arguments omitted here for brevity -->
</sqoop>
```

Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- Defining an Oozie Workflow
- **Validation, Packaging, and Deployment**
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Validating an Oozie Workflow

- The verbose XML format of the workflow lends itself to minor errors
- You can validate the XML using the `oozie` command line utility
 - Example: valid workflow XML document

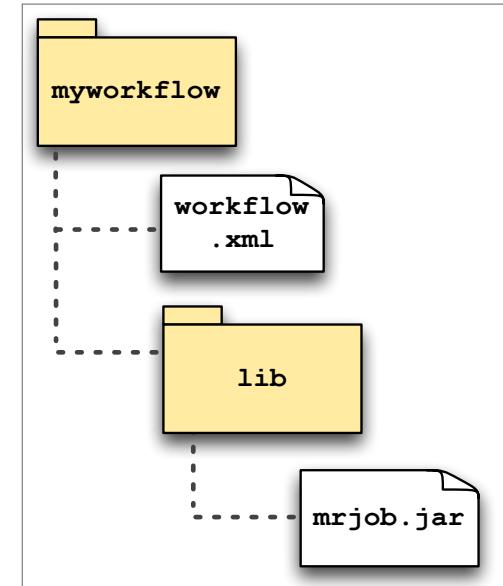
```
$ oozie validate workflow.xml
Valid worflow-app
```

- Example: invalid workflow XML document

```
$ oozie validate workflow.xml
Error: Invalid workflow-app, org.xml.sax.SAXParseException;
lineNumber: 4; columnNumber: 47;
cvc-complex-type.3.2.2: Attribute 'foo' is not allowed to appear
in element 'action'.
```

Packaging and Deploying a Workflow

- **On-disk structure of an Oozie workflow**
 - Base directory (user-defined name)
 - The `workflow.xml` file
- **The lib subdirectory contains required libraries**
 - Optional: not all workflows require libraries
 - More on library handling later in this chapter
- **Copy the base directory into HDFS to deploy it**
 - No restriction on where in HDFS it should be placed



```
$ hadoop fs -put myworkflow /loudacre/etl/
```

- Note that *running* the workflow is a separate operation

Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- **Running and Tracking Workflows using the CLI**
- The Hue UI for Oozie
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Defining Job Properties

- Recall that the `workflow.xml` contained several property references
 - This facilitates reuse and maintainability of Oozie workflows
- Must be defined in a *local* properties file before running the workflow
 - By convention, this is named `job.properties`
 - The `oozie.wf.application.path` property is required
 - Java system properties, such as `${user.name}`, are inherent

```
jobTracker=dev.loudacre.com:8021
nameNode=hdfs://dev.loudacre.com:8020

# Path to the base directory for this workflow in HDFS
# This property must be defined
oozie.wf.application.path=${nameNode}/loudacre/etl/myworkflow

inputDir=${nameNode}/user/${user.name}/example/input_data
dataFile=${nameNode}/user/${user.name}/example/data.tsv
outputDir=${nameNode}/user/${user.name}/example/output_data
recipientEmailAddress=alice@example.com
```

Aside: Handling Libraries in Oozie

- All libraries required for your job must be available at runtime
 - This includes libraries related to Hive, Sqoop, Streaming, and so on
 - JAR files can be placed in the workflow's lib subdirectory, as discussed earlier
- Oozie supports two additional ways to make libraries available
 1. Define the `oozie.libpath` property in `job.properties`
 - Value is a comma-delimited list of directories in HDFS
 - All JAR files in each directory will be added to the runtime classpath
 2. Use Oozie's ShareLib
 - This is a directory in HDFS configured by the Administrator
 - Includes all libraries needed for Hive, Pig, Sqoop, and Streaming
 - Must set `oozie.use.system.libpath=true` (e.g., in `job.properties`) for your job to include these libraries
- Not mutually exclusive – you can use any combination of these

Running a Workflow

- You can run a workflow from the command line
 - Oozie will respond with the new job's ID

```
$ oozie job -oozie http://oozie.example.com:11000/oozie \
  -config job.properties \
  -run
job: 0000039-140115110315378-oozie-oozi-W
```

- Tip: set the **OOZIE_URL** environment variable to your Oozie server URL
 - The `-oozie` argument will then no longer be required

```
$ export OOZIE_URL=http://dev.loudacre.com:11000/oozie
$ oozie job -config job.properties -run
job: 0000039-140115110315378-oozie-oozi-W
```

Getting Information on a Job

- Use the job's ID to get information on the job

```
$ oozie job -info 0000039-140115110315378-oozie-oozi-W
Job ID : 0000039-140115110315378-oozie-oozi-W
-----
Workflow Name : myworkflow
App Path      : hdfs://nn.example.com:8020/loudacre/etl/myworkflow
Status        : KILLED
User          : alice
Started       : 2014-02-27 02:06 GMT
Ended         : 2014-02-27 02:07 GMT
Actions
-----
ID           Status
-----
0000039-140115110315378-oozie-oozi-W@start:      OK
-----
0000039-140115110315378-oozie-oozi-W@prepInputData    ERROR
-----
0000039-140115110315378-oozie-oozi-W@kill        OK
-----
```

Note: command output has been edited and reformatted to fit the screen

Getting Information on a Specific Action within a Job

- Specify the action's ID (rather than the job ID) to debug the failure
 - In this case, we tried to move a non-existent file in HDFS

```
$ oozie job -info 0000039-140115110315378-oozie-oozi-W@prepInputData
ID : 0000039-140115110315378-oozie-oozi-W@prepInputData
-----
Error Message   : FS006: move, source path
                  [hdfs://nn.example.com:8020/user/alice/example/data.tsv]
                  does not exist
External Status: ERROR
Name            : prepInputData
Type            : fs
Started         : 2014-02-27 02:06 GMT
Status          : ERROR
Ended           : 2014-02-27 02:07 GMT
-----
```

Note: command output has been edited and reformatted to fit the screen

Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- **The Hue UI for Oozie**
- Essential Points
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Support for Oozie in Hue

- **Oozie has its own Web UI**
 - Depends on the ExtJS JavaScript library, which must be obtained separately
 - Mainly focused on managing job execution
 - Does not support defining or editing workflows
- **The Hue Web UI in CDH now offers extensive support for Oozie, including:**
 - Defining and editing workflows
 - Submitting workflows for execution
 - Viewing and managing job execution
 - Viewing job output through the file browser
 - Scheduling and managing recurring jobs (coordinators)

The Oozie Dashboard

- The dashboard is the starting point for Oozie-related support in Hue
 - Accessible at: `http://hue_hostname:8888/oozie/`

The screenshot shows the Hue Oozie Editor/Dashboard interface. At the top, there's a toolbar with various icons for navigation and operations. Below the toolbar, a menu bar has tabs for Dashboard, Workflows, Coordinators, and Bundles. The main area is titled "Dashboard". It features a search bar and a filter section where you can choose to show results for 1, 7, 15, or 30 days with specific statuses: Succeeded (green), Running (orange), or Killed (red). Below this, there are two main sections: "Running" and "Completed". The "Running" section has a header with columns for Submission, Status, Name, Progress, Submitter, Created, Last Modified, Run, Id, and Action. A message says "No data available". Below this, it says "Showing 0 to 0 of 0 entries" and includes "Previous" and "Next" navigation buttons. The "Completed" section is partially visible below the "Running" section.

Defining a Workflow in Hue (1)

- Start by clicking the “Workflows” item in the upper navigation area

The screenshot shows the Hue Workflows Dashboard. At the top, there is a navigation bar with icons for Home, Oozie, Coordinators, and Bundles. Below the navigation bar, the word "Dashboard" is displayed. A red box highlights the "Workflows" tab, which is currently selected. The main area is titled "Dashboard" and contains three tabs: "Workflows" (selected), "Coordinators", and "Bundles". Below the tabs is a search bar with placeholder text "Search for username, name, etc...". To the right of the search bar are buttons for filtering results by status: "Show only" (with options 1, 7, 15, 30 days) and "days with status" (with buttons for "Succeeded" (green), "Running" (orange), and "Killed" (red)). The main content area is titled "Running" and displays a table header with columns: "Submission", "Status", "Name", "Progress", "Submitter", "Created", "Last Modified", "Run", "Id", and "Action". The table body below the header shows the message "No data available". At the bottom of the page, there is a footer with the text "Showing 0 to 0 of 0 entries" and navigation links "← Previous" and "Next →".

Defining a Workflow in Hue (2)

- The Workflow Manager lists all Oozie workflows available to Hue
 - Click the “Create” button to begin defining a new workflow

The screenshot shows the Hue Workflow Manager interface. At the top, there is a toolbar with various icons and a dropdown menu labeled "training". Below the toolbar, a navigation bar has tabs for "Dashboard", "Workflows" (which is selected), "Coordinators", and "Bundles". On the right side of the navigation bar, there are buttons for "Submit", "Schedule", "Copy", "Move to trash", "View trash", "Create" (which is highlighted with a red border), and "Import". Below the navigation bar, the main area is titled "Workflow Manager". It features a search bar and several buttons. A table header with columns for "Name", "Description", "Last Modified", "Steps", "Status", and "Owner" is shown. Below the header, a message says "No data available". At the bottom, it shows "Showing 0 to 0 of 0 entries" and navigation buttons for "Previous" and "Next".

Defining a Workflow in Hue (3)

- Specify a name and description for the workflow, then click “Save”

Create Workflow

Properties

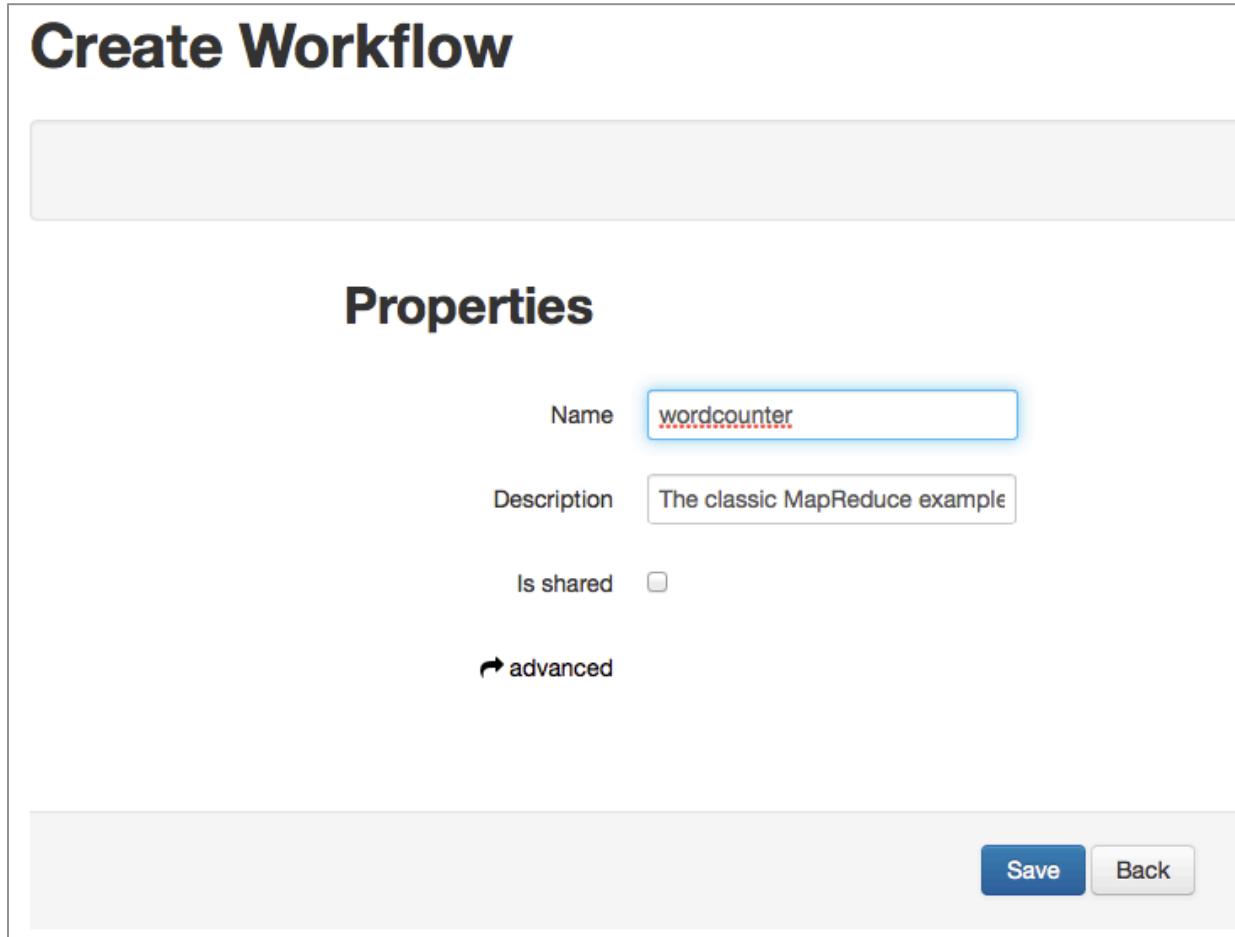
Name

Description

Is shared

[advanced](#)

Save **Back**

A screenshot of the Hue 'Create Workflow' interface. The title 'Create Workflow' is at the top. Below it is a large empty text area. Underneath is a section titled 'Properties'. It contains three fields: 'Name' with the value 'wordcounter', 'Description' with the value 'The classic MapReduce example', and 'Is shared' with an unchecked checkbox. Below these fields is a link 'advanced'. At the bottom right are two buttons: a blue 'Save' button and a white 'Back' button.

Defining a Workflow in Hue (4)

- This takes you to the workflow editor
 - The start and end nodes are automatically present

The screenshot shows the Hue Workflow Editor interface. On the left is a sidebar with the following menu items:

- EDITOR
- Workflow (selected)
- Properties
- Workspace
- ADVANCED
- Import action
- Kill node
- History
- ACTIONS
- Submit
- Schedule
- Copy

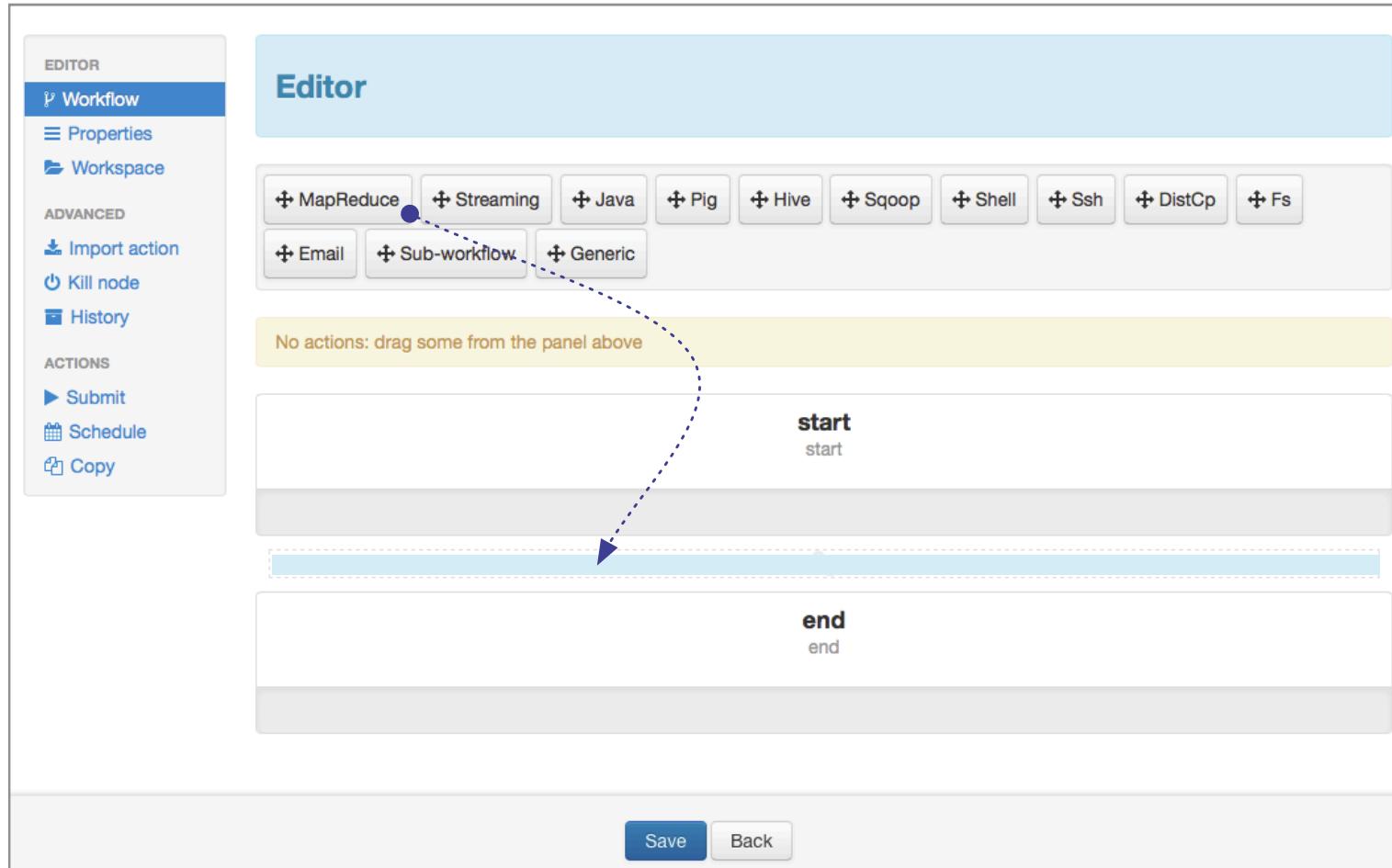
The main area is titled "Editor" and contains a toolbar with various action icons:

- MapReduce, Streaming, Java, Pig, Hive, Sqoop, Shell, Ssh, DistCp, Fs
- Email, Sub-workflow, Generic

A message at the top of the workspace says "No actions: drag some from the panel above". Below this, there are two nodes: "start" and "end". The "start" node is positioned above a dashed line, and the "end" node is positioned below it. At the bottom of the editor are "Save" and "Back" buttons.

Defining a Workflow in Hue (5)

- Drag and drop an action to the desired position in the workflow



Defining a Workflow in Hue (6)

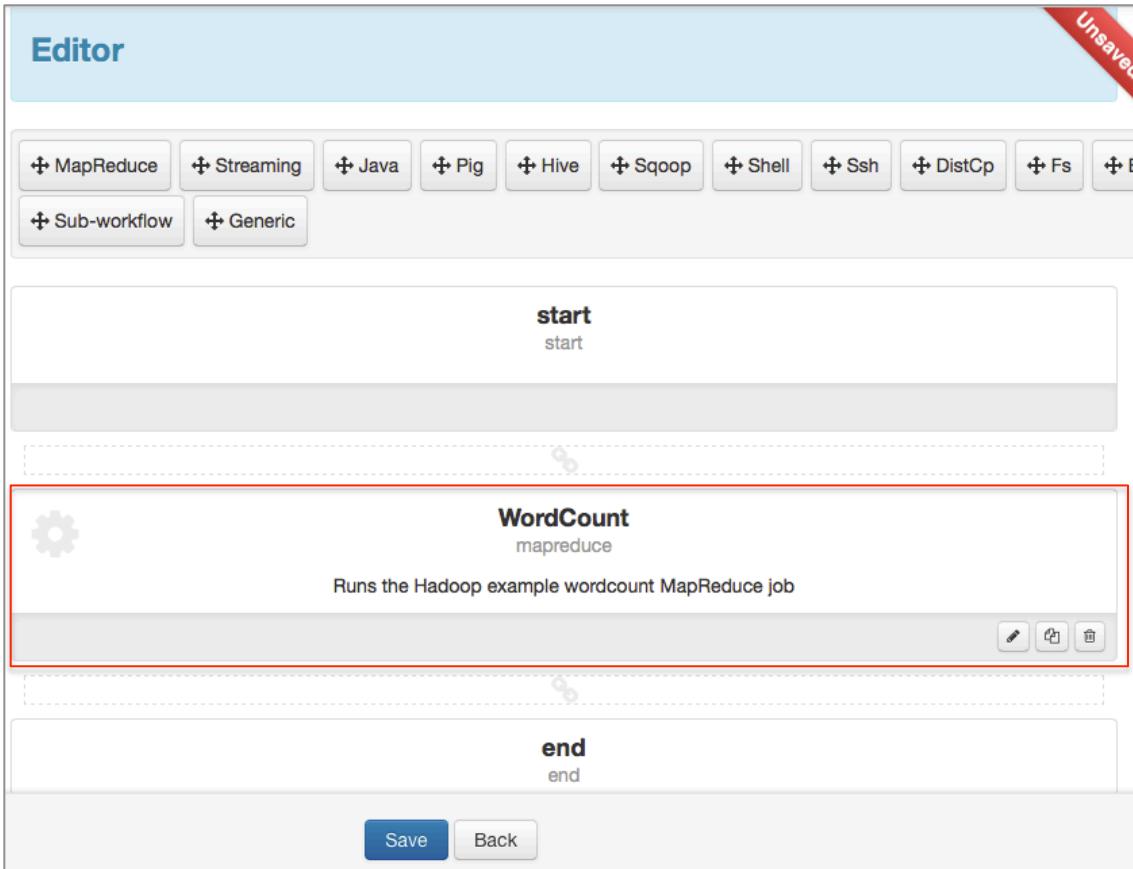
- This immediately opens the editor for that action
 - Populate the form as needed and then click the “Done” button

Edit Node: WordCount

Name	WordCount										
Description	Runs the Hadoop example <code>wordcount</code> MapReduce job										
Action type	mapreduce										
<p>All the paths are relative to the deployment directory. They can be absolute but this is not recommended. You can parameterize values using case sensitive <code> \${parameter} </code>.</p>											
Jar name	<input type="text" value="hadoop-examples.jar"/> ..										
Prepare	<input type="button" value="Add delete"/> <input type="button" value="Add mkdir"/>										
Job properties	<table border="1"><thead><tr><th>Property name</th><th>Value</th></tr></thead><tbody><tr><td><code>mapreduce.map.class</code></td><td><input type="text" value="org.apache.hadoop.examples.WordCount\$Tol"/> .. <input type="button" value="Delete"/></td></tr><tr><td><code>mapreduce.reduce.class</code></td><td><input type="text" value="org.apache.hadoop.examples.WordCount\$Int"/> .. <input type="button" value="Delete"/></td></tr><tr><td><code>mapred.input.dir</code></td><td><input type="text" value="/user/training/wordcount-input"/> .. <input type="button" value="Delete"/></td></tr><tr><td><code>mapred.output.dir</code></td><td><input type="text" value="/user/training/wordcount-output"/> .. <input type="button" value="Delete"/></td></tr></tbody></table>	Property name	Value	<code>mapreduce.map.class</code>	<input type="text" value="org.apache.hadoop.examples.WordCount\$Tol"/> .. <input type="button" value="Delete"/>	<code>mapreduce.reduce.class</code>	<input type="text" value="org.apache.hadoop.examples.WordCount\$Int"/> .. <input type="button" value="Delete"/>	<code>mapred.input.dir</code>	<input type="text" value="/user/training/wordcount-input"/> .. <input type="button" value="Delete"/>	<code>mapred.output.dir</code>	<input type="text" value="/user/training/wordcount-output"/> .. <input type="button" value="Delete"/>
Property name	Value										
<code>mapreduce.map.class</code>	<input type="text" value="org.apache.hadoop.examples.WordCount\$Tol"/> .. <input type="button" value="Delete"/>										
<code>mapreduce.reduce.class</code>	<input type="text" value="org.apache.hadoop.examples.WordCount\$Int"/> .. <input type="button" value="Delete"/>										
<code>mapred.input.dir</code>	<input type="text" value="/user/training/wordcount-input"/> .. <input type="button" value="Delete"/>										
<code>mapred.output.dir</code>	<input type="text" value="/user/training/wordcount-output"/> .. <input type="button" value="Delete"/>										
<input type="button" value="Cancel"/> <input type="button" value="Done"/>											

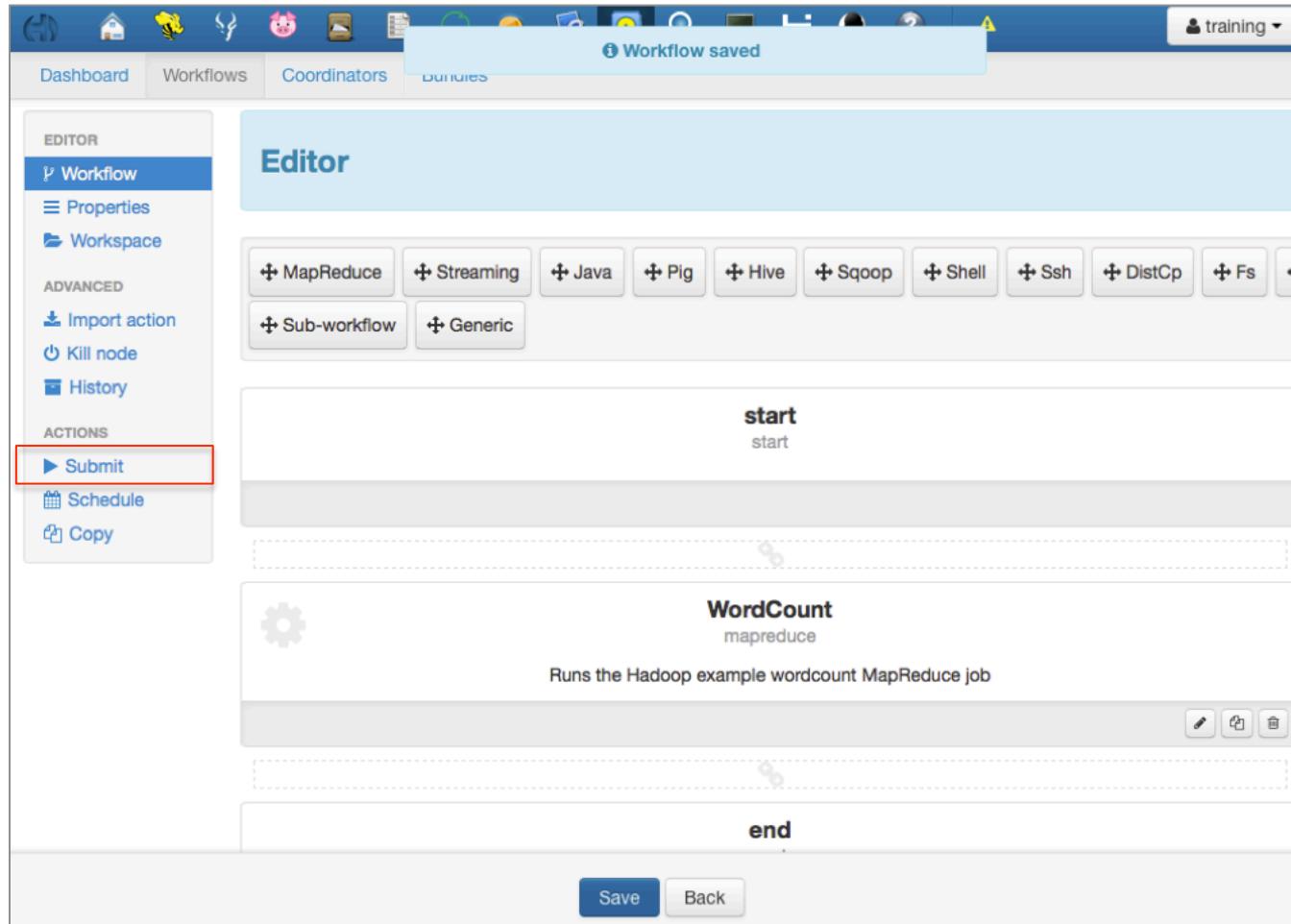
Defining a Workflow in Hue (7)

- The configured action now appears in the workflow
 - Add more actions if desired and then click the “Save” button



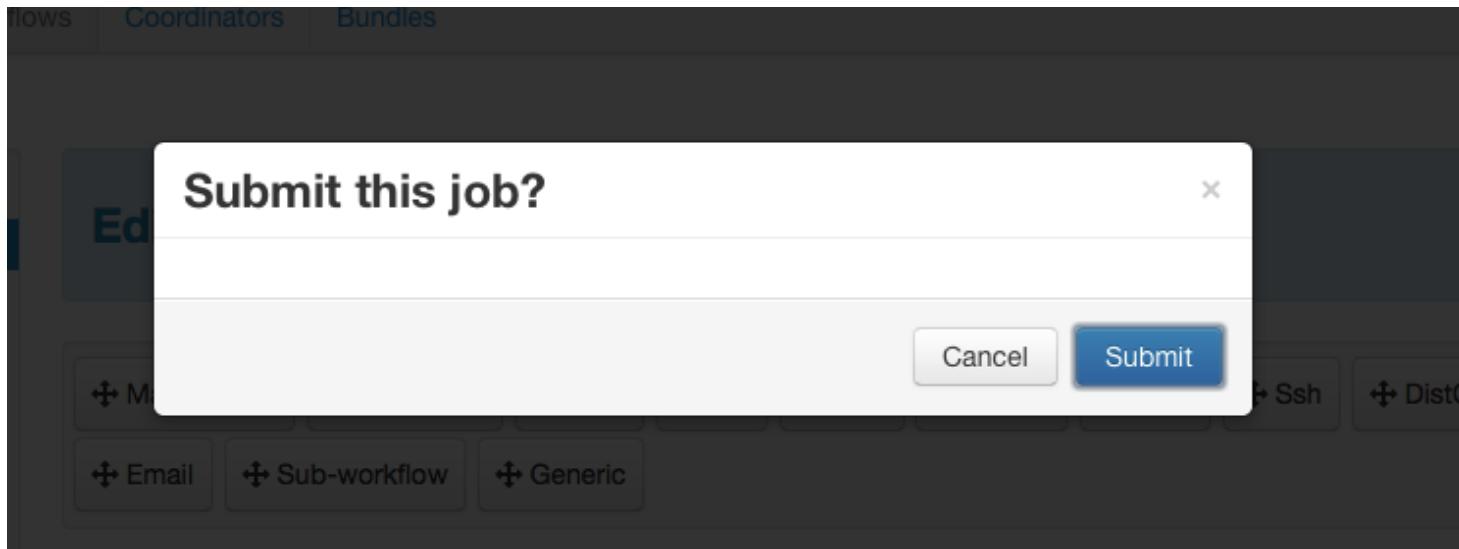
Running a Workflow in Hue (1)

- To execute the new workflow, click the “Submit” icon on the left



Running a Workflow in Hue (2)

- This shows an overlay confirming your intent
 - Click the “Submit” button



Running a Workflow in Hue (2)

- The workflow's progress is now shown in the dashboard

Dashboard

Workflows Coordinators Bundles Oozie

Workflow wordcounter

WORKFLOW
wordcounter - training

SUBMITTER
training

STATUS
RUNNING

PROGRESS
33%

ID
0000064-140115110315378-oozie-oozi-W

VARIABLES

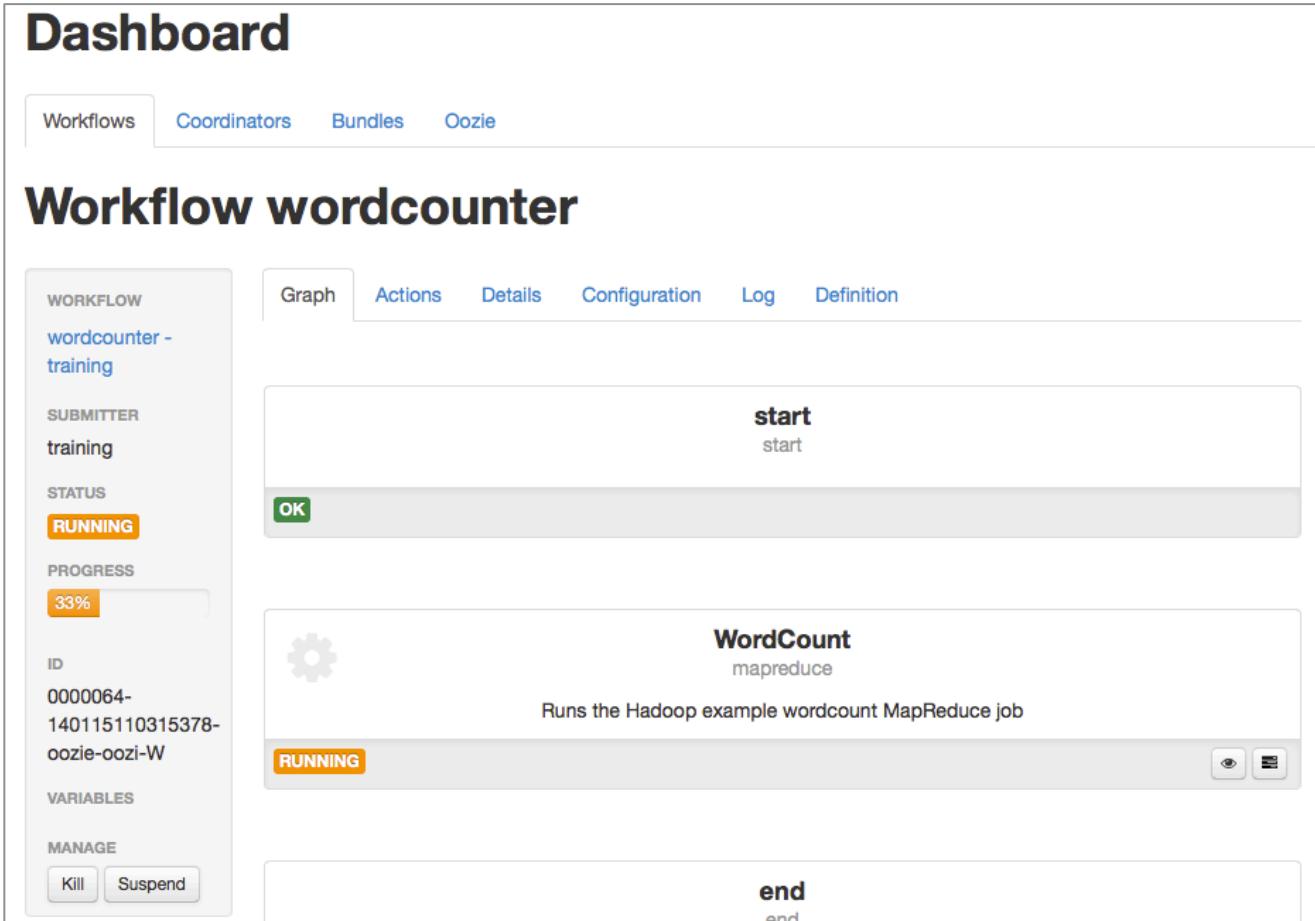
MANAGE
Kill Suspend

Graph Actions Details Configuration Log Definition

start
start
OK

WordCount
mapreduce
Runs the Hadoop example wordcount MapReduce job
RUNNING

end
end



Inspecting the Workflow Definition

- Hue can also show the workflow as an XML document

Workflow wordcounter

WORKFLOW
wordcounter - training
SUBMITTER
training
STATUS
SUCCEEDED
PROGRESS
100%
ID
0000064-140115110315378-oozie-oozi-W
VARIABLES
MANAGE
Rerun

Graph Actions Details Configuration Log **Definition**

```
1 <workflow-app name="wordcounter" xmlns="uri:oozie:workflow:0.4">
2   <start to="WordCount"/>
3   <action name="WordCount">
4     <map-reduce>
5       <job-tracker>${jobTracker}</job-tracker>
6       <name-node>${nameNode}</name-node>
7       <configuration>
8         <property>
9           <name>mapreduce.map.class</name>
10          <value>org.apache.hadoop.examples.WordCount$TokenizerMapper</value>
11        </property>
12        <property>
13          <name>mapreduce.reduce.class</name>
14          <value>org.apache.hadoop.examples.WordCount$IntSumReducer</value>
15        </property>
16        <property>
17          <name>mapred.input.dir</name>
18          <value>/user/training/wordcount-input</value>
19        </property>
20        <property>
21          <name>mapred.output.dir</name>
22          <value>/user/training/wordcount-output</value>
23        </property>
24        <property>
25          <name>mapred.output.key.class</name>
26          <value>org.apache.hadoop.io.Text</value>
27        </property>
28        <property>
29          <name>mapred.output.value.class</name>
30          <value>org.apache.hadoop.io.IntWritable</value>
31        </property>
```

Scheduling Recurring Workflows (Coordinators) in Hue

- Hue makes it easy to define a coordinator
 - Coordinators are used to automate execution of a workflow

The screenshot shows the Hue Workflow Editor interface. On the left, a sidebar menu includes options like 'Workflow' (selected), 'Properties', 'Workspace', 'Import action', 'Kill node', 'History', 'Submit', 'Schedule' (which is highlighted with a red box), and 'Copy'. The main area is titled 'Editor' and contains a toolbar with various job types: MapReduce, Streaming, Java, Pig, Hive, Sqoop, Shell, Ssh, DistCp, Fs, Email, Sub-workflow, and Generic. Below the toolbar, there's a 'start' node labeled 'start' and 'start'. A 'WordCount' node is shown, described as 'Runs the Hadoop example wordcount MapReduce job'. At the bottom, there's an 'end' node labeled 'end' and 'end'. At the very bottom of the editor are 'Save' and 'Back' buttons.

Hue Coordinator Editor (1)

- Specify a name and description and click “Next”

Coordinator Editor

Step 1: Details Step 2: Frequency Step 3: Inputs Step 4: Outputs Step 5: Advanced settings

Coordinator data

Name Name of the job, which must be unique per user.

Description The purpose of the job.

Workflow The workflow to schedule repeatedly.

Is shared Enable other users to have access to this job.

Hue Coordinator Editor (2)

- Here we specify when the job starts and ends, and how often it recurs

Coordinator Editor

Step 1: Details Step 2: Frequency Step 3: Inputs Step 4: Outputs Step 5: Advanced settings

Frequency

Frequency number: 20

The number of units of the rate at which data is periodically created.

Frequency unit: Minutes

The unit of the rate at which data is periodically created.

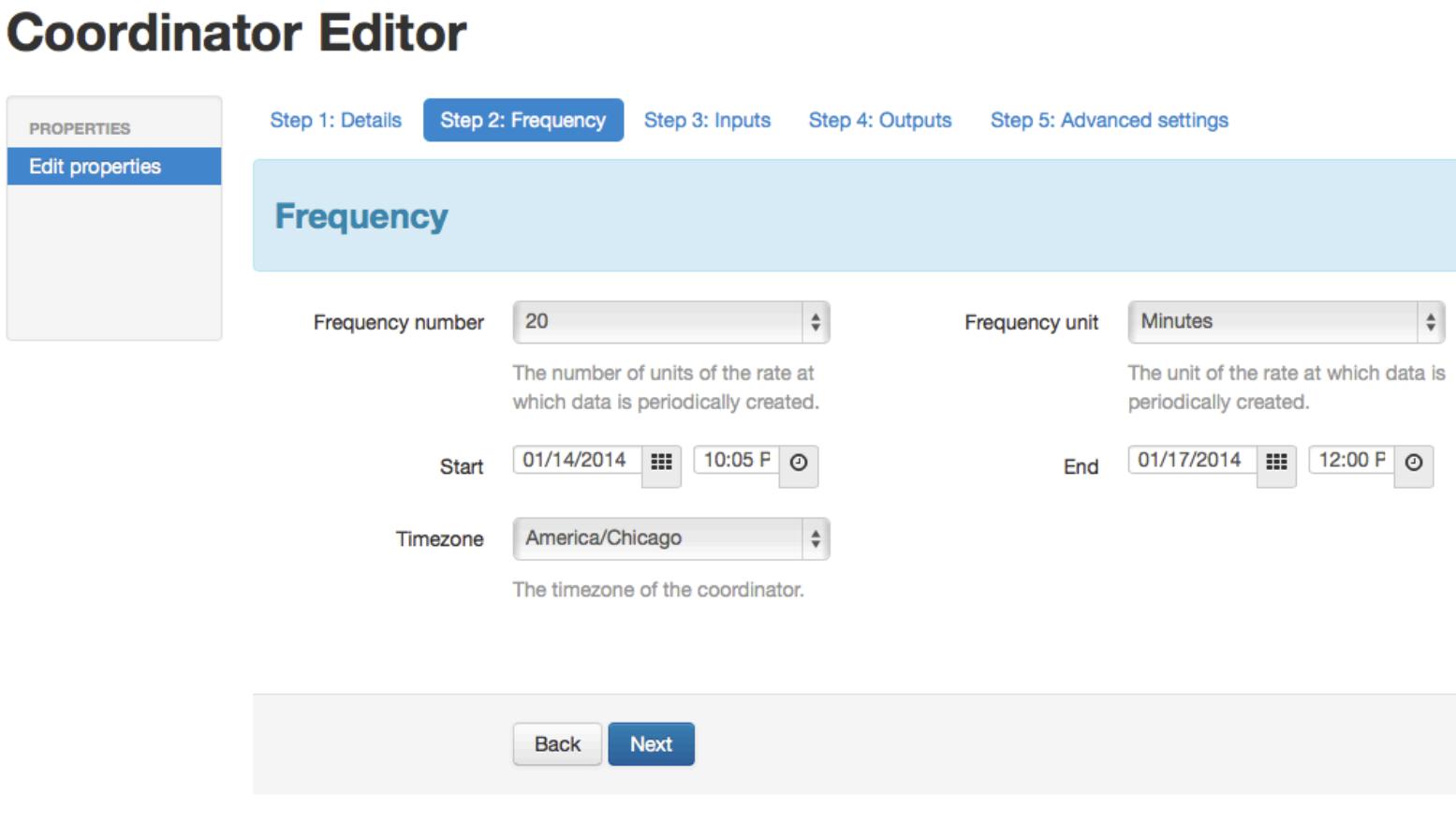
Start: 01/14/2014 10:05 P

End: 01/17/2014 12:00 P

Timezone: America/Chicago

The timezone of the coordinator.

Back Next



Hue Coordinator Editor (3)

- **We can skip the remaining steps in our example**
 - These mainly relate to defining input and output ‘datasets’
- **Datasets are often used to start a workflow when data becomes available**
 - Also useful for defining data dependencies between workflows
- **However, our coordinator executes the workflow at a fixed interval**

Running a Coordinator Job in Hue

- This process is identical to running the workflow

Coordinator Editor : recurring-wordcounter

Step 1: General Step 2: Frequency Step 3: Inputs Step 4: Outputs Step 5: Advanced settings

Coordinator data

Name Name of the job, which must be unique per user.

Description The purpose of the job.

Workflow The workflow to schedule repeatedly.

Is shared Enable other users to have access to this job.

[Back](#) [Next](#)

Properties
Edit properties
Workflow
wordcounter - training
Datasets
Create new
Show existing
History
Show history
Actions
Submit
Copy

Managing a Coordinator Job in Hue

- Use the Dashboard to view and manage your coordinator

The screenshot shows the Hue Coordinator dashboard. At the top, there's a navigation bar with icons for Home, Workflows, Coordinators, Bundles, and a search bar. Below the navigation bar, the word "Dashboard" is displayed. In the center, there's a sub-navigation bar with tabs for Workflows, Coordinators (which is selected and highlighted with a red border), Bundles, and Oozie. Below this, there's a search input field and a filter section for "Show only" days with status: Succeeded (green), Running (orange), or Killed (red). The main content area is titled "Running" and lists a single job: "recurring-wordcounter". The job details include: Next Submission (Mon, 27 Jan 2014 04:00:00), Status (RUNNING), Name (recurring-wordcounter), Start Time (Thu, 23 Jan 2014 14:05:00), Id (0000146-140115110315378-oozie-oozi-C), and Action buttons for Kill and Suspend. At the bottom right, there are navigation buttons for Previous, 1, and Next.

Next Submission	Status	Name	Start Time	Id	Action
Mon, 27 Jan 2014 04:00:00	RUNNING	recurring-wordcounter	Thu, 23 Jan 2014 14:05:00	0000146-140115110315378-oozie-oozi-C	Kill Suspend

Note: some columns have been removed to fit the screen

Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- **Essential Points**
- Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

Bibliography

The following offer more information on topics discussed in this chapter

- **Apache Oozie Web Site**

- <http://oozie.apache.org/>

- **Oozie-related Articles on Cloudera's Blog**

- <http://tiny.cloudera.com/adcc08a>

- **Oozie-related Video Tutorials for Hue**

- <http://tiny.cloudera.com/adcc08b>

Essential Points

- **Hadoop and related tools are often used in complex workflows**
 - Oozie is a versatile workflow engine for the Hadoop ecosystem
- **Oozie is implemented as a client-server system**
 - Users submit workflows to Oozie server
 - The Oozie server then submits jobs to the Hadoop cluster
 - Users can perform job management via the shell or the Hue UI
- **A workflow consists of control flow and action nodes**
 - Oozie has actions for MapReduce, Hive, Sqoop, HDFS, and others
 - Control flow nodes define the sequence of events
- **Coordinators run workflows based on schedules or external events**

Chapter Topics

Managing Workflows with Apache Oozie

- The Need for Workflow Management
- What is Apache Oozie?
- Defining an Oozie Workflow
- Validation, Packaging, and Deployment
- Running and Tracking Workflows using the CLI
- The Hue UI for Oozie
- Essential Points
- **Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie**

Hands-On Exercise: Creating a Multi-Stage Workflow with Oozie

- In this Hands-On Exercise, you will create and execute a workflow that uses Sqoop to import a data set and MapReduce to process it
 - Please refer to the Hands-On Exercise Manual for instructions

Processing Data Pipelines with Apache Crunch

Chapter 9



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- **Processing Data Pipelines with Apache Crunch**
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Processing Data Pipelines with Apache Crunch

In this chapter you will learn

- What Apache Crunch is and how it compares to MapReduce
- How to create Crunch projects with Maven
- How to read and write data in Crunch
- How to use Crunch's parallel collection classes
- How to define and invoke processing functions
- How to represent data structures using custom types
- How to test and execute your Crunch code

Chapter Topics

Processing Data Pipelines with Apache Crunch

- **What is Apache Crunch?**
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

The Motivation for Apache Crunch

- **MapReduce is a powerful framework for large-scale data processing**
- **However, writing MapReduce code directly in Java can be tedious**
 - Business logic typically comprises just a fraction of overall code
 - Many real-world computations involve a sequence of jobs
 - Chaining multiple MapReduce jobs increases the complexity
- **Crunch is designed to address these problems**

What is Apache Crunch?

- **Crunch is a library that simplifies parallel processing**
 - An open-source implementation of a library developed at Google
 - Initially created at Cloudera
 - Now an active top-level Apache project
- **Crunch provides a high-level API targeted at Java developers**
 - Does not require detailed knowledge of MapReduce
 - More productive than writing MapReduce code directly
- **An alternative to tools like Apache Pig**
 - Crunch retains the full power and expressiveness of Java

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- **Comparing Crunch to Java MapReduce**
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Comparing Crunch to MapReduce

- We can learn the basics of Crunch by examining a simple job
- The following slides compare two implementations of WordCount
 - First, using the MapReduce Java API
 - Second, using the Crunch API
- The input and output of each is exactly the same
 - Both can be executed using the `hadoop jar` command
 - The execution time for both is also approximately the same
 - The Crunch version simply takes less time to write and test

WordCount with Hadoop's MapReduce Java API (1)

- Excerpt from the driver used in the MapReduce version of WordCount

```
public int run(String[] args) throws Exception {
    Job job = new Job();

    job.setJobName("Word Count");
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(WordMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

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

    boolean success = job.waitForCompletion(true);
    return success ? 0 : 1;
}
```

Note: portions of the driver class omitted for brevity

WordCount with Hadoop's MapReduce Java API (2)

- The Mapper splits each line of text into individual words

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

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        for (String word : line.split("\\s+")) {
            context.write(new Text(word), new IntWritable(1));
        }
    }
}
```

WordCount with Hadoop's MapReduce Java API (3)

- The Reducer sums the number of occurrences for each word

```
public class SumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
                      Context context) throws IOException, InterruptedException {

        int wordCount = 0;
        for (IntWritable value : values) {
            wordCount += value.get();
        }

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

WordCount with the Crunch API (1)

- **In Crunch, there is no need for a separate driver class**
 - You need only specify a few details like input and output paths
 - Extending CrunchTool provides convenience methods for this
 - Crunch configures and submits jobs for you
- **Crunch's API does not have specific map and reduce methods**
 - Processing logic is provided through simple function objects
- **Crunch allows you to use normal Java classes for processing**
 - It can convert these to your choice of Writables or Avro

WordCount with the Crunch API (2)

- Most of Crunch's WordCount implementation fits on one slide

```
public class WordCount extends CrunchTool {  
    public static void main(String[] args) throws Exception {  
        ToolRunner.run(new Configuration(), new WordCount(), args);  
    }  
  
    @Override  
    public int run(String[] args) throws Exception {  
        PCollection<String> lines = read(From.textFile(args[0]));  
  
        // implementation of ExtractWordsFn is shown on the next slide  
        PCollection<String> words = lines.parallelDo("extract words",  
            new ExtractWordsFn(), Writables.strings());  
  
        PTable<String, Long> counts = words.count();  
  
        counts.write(To.textFile(args[1]));  
        return done().succeeded() ? 0 : 1;  
    }  
}
```

WordCount with the Crunch API (3)

- Our line-splitting function is implemented as a static inner class
 - Logic is the same as the Mapper shown earlier

```
static class ExtractWordsFn extends DoFn<String, String> {  
    @Override  
    public void process(String input, Emitter<String> output) {  
        for (String word : input.split("\\s+")) {  
            output.emit(word);  
        }  
    }  
}
```

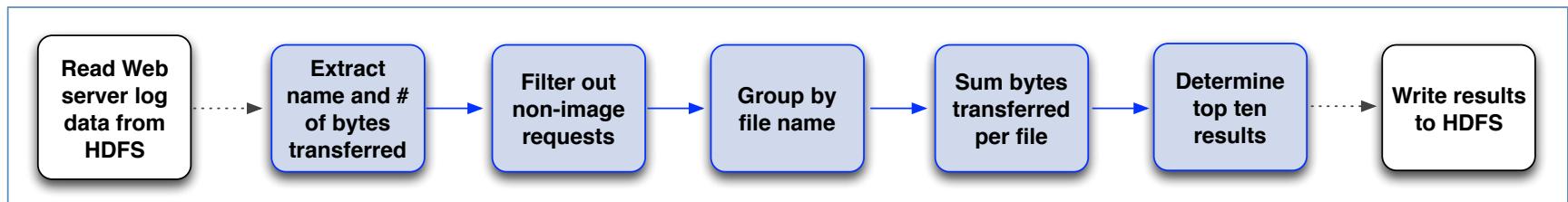
Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- **Understanding the Crunch Pipeline**
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Processing Pipelines in Crunch

- **The notion of “processing pipelines” is fundamental to Crunch**
 - Pipelines provide an abstraction for a logical computation
 - These are defined in Java using the Crunch API
- **The pipeline specifies the location of input and output data**
 - Data is read into immutable PCollection objects
 - Pipelines also define a sequence of functions used to process the data



Reading Data

- **Input data is read into a PCollection from a Source**
- **A Source has three responsibilities**
 - Identify the location of the input data
 - Select the correct InputFormat to use for this data
 - Specify the data type returned by the source
- **You can instantiate a Source with the From utility class**
 - The input path can be either a Hadoop Path object or a String

```
Source<String> source = From.textFile(args[0]);
PCollection<String> lines = getPipeline().read(source);

// one-liner equivalent using a convenience method in CrunchTool
PCollection<String> lines = read(From.textFile(args[0]));
```

Additional Crunch Source Implementations

- Crunch has sources for many common Hadoop file types

```
// read a collection of IntWritables from values in a SequenceFile  
PCollection<IntWritable> prices = read(From.sequenceFile(  
    args[0], IntWritable.class));  
  
// The specifics method is used for Java classes generated from Avro  
PCollection<StationInfo> stationInfo = read(From.avroFile(  
    Avros.specs(StationInfo.class)));
```

- The Kite SDK also provides a Source for reading its data sets

```
DatasetRepository repo = DatasetRepositories.open("repo:hive:/loudacre");  
Dataset<Product> productsDataset = repo.load("products");  
  
PCollection<Product> products = read(CrunchDatasets.asSource(  
    productsDataset, Product.class)));
```

Functions in Crunch

- **Processing logic is encapsulated in function objects**
 - Functions are subclasses of `DoFn<S, T>` in the Crunch API
- **Functions are applied to a PCollection**
 - The result is a new PCollection
- **They tend to be relatively short and simple**
 - Take a specific type as input
 - Return a specific type as output
- **A processing pipeline will often chain together many functions**

A Simple Function Example

- All functions extend the `DoFn<S, T>` abstract class
 - S represents the input value type
 - T represents the output value type
- Implementer is only required to provide the `process` method
 - Called once for each element in the input collection
 - The Emitter class collects the output
 - Can emit zero or more output values

```
public class StringLengthFn extends DoFn<String, Integer> {  
    @Override  
    public void process(String input, Emitter<Integer> output) {  
        output.emit(input.length());  
    }  
}
```

Invoking a Function

- Functions are usually applied using the `parallelDo` method
 - The first argument, description, is optional but recommended
 - The second argument is an instance of the function itself
 - The third argument is an instance of `PType`
- A `PType` specifies data type and serialization format of the *output value*
 - This example uses `Writable` serialization format with `int` data type

```
// read lines from text files as input
PCollection<String> lines = read(From.textFile(args[0]));

// calculate the length of each line
PCollection<Integer> lengths = lines.parallelDo("measure string length",
    new StringLengthFn(), Writables.ints());
```

Accessing MapReduce Features from a Function

- The Crunch API abstracts away many details of MapReduce
- Many MapReduce-related methods are still accessible from within a DoFn
 - Use progress () or setStatus (String) to signal activity
 - Several variations of the increment method for updating counters
 - The getContext () method returns a Hadoop Context object

```
@Override
public void process(String input, Emitter<Double> output) {
    try {
        double value = Double.valueOf(input);
        setStatus("Performing long-running calculation");
        double result = doSomeIntensiveCalculation(value);
    } catch (NumberFormatException nfe) {
        increment("INPUT_RECORDS", "UNPARSEABLE");
    }
}
```

Writing Data

- **Data collections containing final results are written to a Target**
 - Similar role as a Source but used for output instead of input
 - The To utility class is often used to create a Target
 - The output path can be either a Hadoop Path object or a String
- **No need to specify data types, since a PCollection knows its type**

```
lines.write(To.textFile(outputPath)) ;  
  
prices.write(To.sequenceFile(outputPath)) ;  
  
customers.write(To.avroFile(outputPath)) ;
```

Pipeline Execution

- **The pipeline manages the entire execution sequence**
 - May create a series of MapReduce jobs to produce desired results
 - Reads and writes input, output, and intermediate data as needed
- **Crunch also supports alternate pipeline execution strategies**
 - For example, it can execute a pipeline in memory (for testing)
- **The `done()` method should *always* be the last call in a pipeline**
 - Executes the job and performs any necessary cleanup afterward

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- **Unit Testing**
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Unit Testing Your Functions

- **It is important to test your functions**
 - Does not require a special testing library – JUnit is sufficient
 - Use the `InMemoryEmitter` class so you can retrieve values
- **The next slide shows how to write a unit test for this function**

```
public class StringLengthFn extends DoFn<String, Integer> {  
  
    @Override  
    public void process(String input, Emitter<Integer> output) {  
        output.emit(input.length);  
    }  
}
```

Unit Test Example

```
public class StringLengthFnTest {

    private StringLengthFn function;
    private InMemoryEmitter<Integer> emitter;

    @Before
    public void setUp() {
        function = new StringLengthFn();
        emitter = new InMemoryEmitter<Integer>();

        function.process("cat", emitter);
        function.process("wombat", emitter);
        function.process("aardvark", emitter);
    }

    @Test
    public void verifyWordLengths() throws Exception {
        assertEquals(3, emitter.getOutput().get(0)); // cat
        assertEquals(6, emitter.getOutput().get(1)); // wombat
        assertEquals(8, emitter.getOutput().get(2)); // aardvark
    }
}
```

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- **Working with Crunch Projects**
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Creating a Crunch Project

- **Crunch is a Java library you use in your project**
 - If using Maven, add a section like this to your pom.xml file

```
<dependency>
    <groupId>org.apache.crunch</groupId>
    <artifactId>crunch-core</artifactId>
    <version>${crunch.version}</version>
</dependency>
```

- **Value of crunch.version varies based on use of Hadoop 1.x or 2.x**
 - Always specify the 2.x version with CDH 4.x and later

Building and Running a Crunch Project (1)

- Getting all necessary libraries into runtime classpath can be tricky
- Suggestion: create your project using Maven's Crunch project archetype

```
$ mvn archetype:generate \
-Dfilter=org.apache.crunch:crunch-archetype
```

- This will use Maven's assembly plugin
 - Allows you to create a single JAR containing all dependencies
 - Projects in upcoming Crunch Hands-On Exercises already use this plugin

Building and Running a Crunch Project (2)

- **Package your Crunch code and any dependencies into a JAR**
 - You can then use `hadoop jar` to run your driver, as for any other job
- **Builds that use the assembly plugin will produce two JAR files**
 - `wordcount-1.0.jar`
 - `wordcount-1.0-job.jar`
 - Use the one ending with `-job` with the `hadoop jar` command

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- **Hands-On Exercise: File Type Processing with Crunch**
- Data Collections and Serialization in the Crunch API
- Utility Classes in the Crunch API
- Essential Points
- Hands-On Exercise: Analyzing Web Server Log Data with Crunch

Hands-On Exercise: File Type Processing with Crunch

- In this Hands-On Exercise, you will use Crunch to count the number of requests for each file type in Web server log files
 - Please refer to the Hands-On Exercise Manual for instructions

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- **Data Collections and Serialization in the Crunch API**
- Built-in Utility Classes
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Working with PCollections

- All data used in Crunch is encapsulated in a PCollection
 - PCollections represents data distributed throughout the cluster
- A PCollection is similar to a list or array
 - Each element contains a single object
 - All of these objects are of the same type

```
PCollection<String> lines = read(From.textFile(args[0]));
```

- A PCollection differs from lists and arrays in two important ways
 - Elements must be serializable using Writables or Avro
 - A PCollection is immutable

Representing Complex Data (1)

- A PCollection is a collection of exactly one type of object
 - Such as PCollection<String> or PCollection<Integer>
 - How can we represent complex data?
- The Crunch API provides the Pair utility class
 - Use this to combine two values into a single object
 - Also provides Tuple classes for three or more elements

```
Pair<String, Integer> products = new Pair<String, Integer>(name, price);  
  
Tuple3<Integer, String, String> employees =  
    new Tuple3<Integer, String, String>(id, firstName, lastName);
```

Representing Complex Data (2)

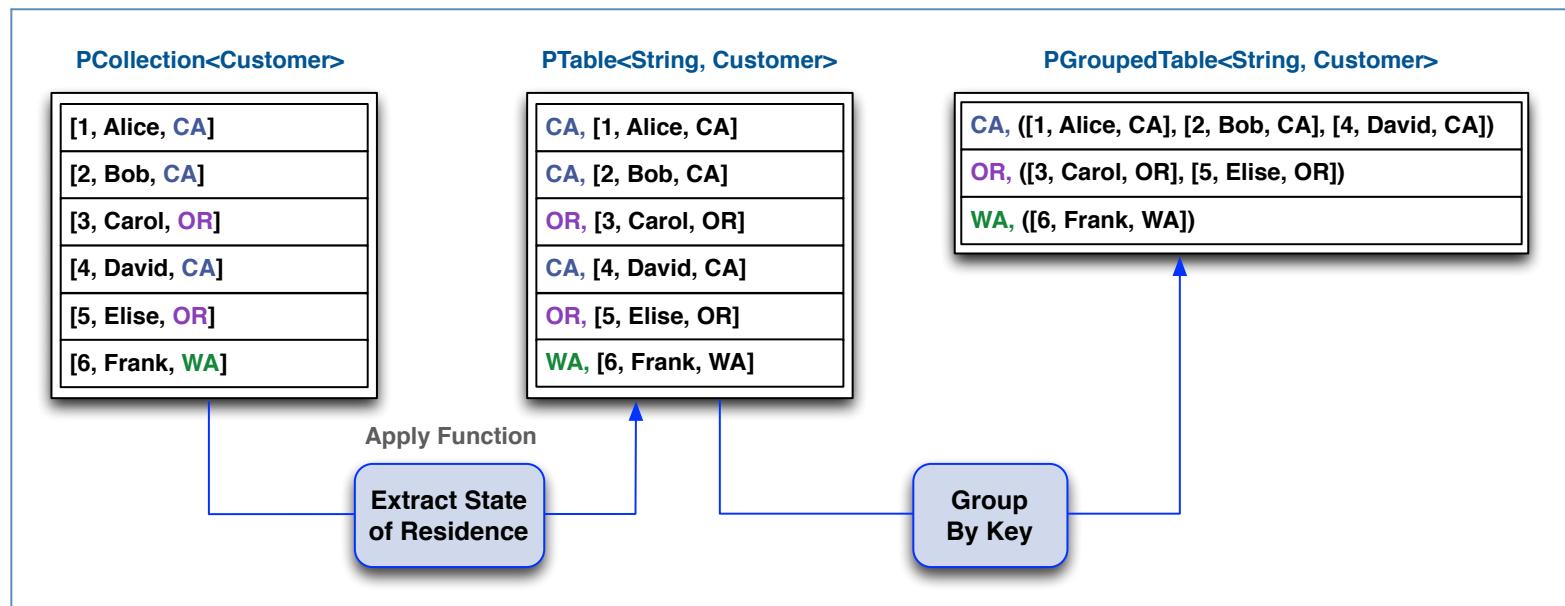
- Type specification can become complicated when using pairs and tuples

```
PCollection<Pair<String, Pair<Integer, String>>> data;
```

- Avoid “angle bracket syndrome” by using a custom type instead
 - Custom types have specific fields to represent specific values
 - A better choice if you will use this data structure frequently
- Crunch supports custom types using Writable or Avro
 - More on this in a moment...

Subtypes of PCollection

- There are two subclasses of PCollection
- A PTable is similar to a map
 - It contains two items (key and value) per element
- A PGroupedTable is created by grouping a PTable by its key
 - Each element contains a single key and an Iterable of values



Utility Methods in PCollection and PTable

- Some of the utility methods defined in PCollection:

Method	Description
count ()	Counts number of unique elements
min ()	Identifies smallest object in collection
max ()	Identifies largest object in collection
union (PCollection<S>)	Combines another collection of same type

- PTable inherits all of those, and also adds several more:

Method	Description
top (int n)	Returns n largest values
bottom (int n)	Returns n smallest values
keys ()	Extracts only keys into a PCollection
values ()	Extracts only values into a PCollection

Working With PTypes

- PTypes are used to signal how data collections should be serialized
- This example uses the String data type and Avro serialization format

```
PCollection<String> words =  
    lines.parallelDo(new ExtractWordsFn(), Avros.strings());
```

- This example uses a custom data type that implements Writable

```
PCollection<LogEntry> parsed = logLines.parallelDo(  
    new ParseLogEntryFn(), Writables.writables(LogEntry.class));
```

Using Custom Objects with Avro

- **Can also use a custom data type with Avro's on-the-fly schema generation**
 - Allow you to use a Java object without having to implement Writable
- **Caveats of the Avros . reflect method**
 - Object must have a zero-argument constructor
 - All fields must have an equivalent type in Avro

```
PCollection<Customer> customers = rawData.parallelDo(  
    new ParseCustomerDataFn(), Avros.reflects(Customer.class));
```

- **Use the Avros . specifics method for classes generated from schema**
 - Preferred approach, especially if long-term storage is likely

Creating PTypes for Composite Objects

- It is common to use a **Pair** as output from a function
 - Both Writables and Avros also have a `pairs` method

```
PCollection<Pair<String, Integer>> wordLengths =  
    lines.parallelDo(new StringLengthFn(),  
        Writables.pairs(Writables.strings(), Writables.ints()));
```

- Writables and Avros also both have a **tableOf** method
 - Use this when working with PTables

```
PTable<String, Customer> statesAndCustomers =  
    customers.parallelDo(new ExtractStateFn(),  
        Avros.tableOf(Avros.strings(), Avros.reflects(Customer.class)));
```

The Function Lifecycle

- There are four important methods in the DoFn lifecycle
 - You are only *required* to implement the process method

1. `configure(Configuration)`

- Called before the job is run (can modify configuration)

2. `initialize()`

- Called before any processing for this instance begins

3. `process(S input, Emitter<T> output)`

- Called once for each element in the collection being processed

4. `cleanup(Emitter<T> output)`

- Called after all processing for this instance is complete

Serialization in Functions (1)

- The DoFn class implements the `java.io.Serializable` interface
 - Therefore, all DoFn implementations are Serializable
 - This can cause problems for developers

```
public class ParseLogEntryFn extends DoFn<String, LogEntry> {  
  
    private LogLineParser parser; // non-serializable class  
  
    public ParseLogEntryFn() {  
        parser = new LogLineParser();  
    }  
  
    @Override  
    public void process(String input, Emitter<LogEntry> output) {  
        LogEntry parsed = parser.parseLine(input);  
    }  
}
```

Serialization in Functions (2)

- The previous example can fail at runtime, but this is easily fixed
 - Mark the member as transient
 - Instantiate it in the `initialize()` method

```
public class ParseLogEntryFn extends DoFn<String, LogEntry> {  
  
    private transient LogLineParser parser;  
  
    @Override  
    public void initialize() {  
        parser = new LogLineParser();  
    }  
  
    @Override  
    public void process(String input, Emitter<LogEntry> output) {  
        LogEntry parsed = parser.parseLine(input);  
    }  
}
```

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- **Built-In Utility Classes**
- Essential Points
- Hands-On Exercise: Log Processing with Crunch

Other Processing Features in Crunch

- The `org.apache.crunch.lib` package contains many utilities, e.g.:
 - Sample
 - Distinct
 - Join
- We will cover two more of these, with examples
 - Sort
 - Secondary Sort

Using the Sort Class

- The Sort class is used to sort a collection
 - Sorted according to natural ordering of its elements

```
PCollection<String> sorted = Sort.sort(lines, Sort.Order.ASCENDING);
```

- Also provides methods for sorting Pairs and Tuples
 - These support sorting by a given column
 - The column indexing is 1-based (like SQL) not 0-based (like Java)

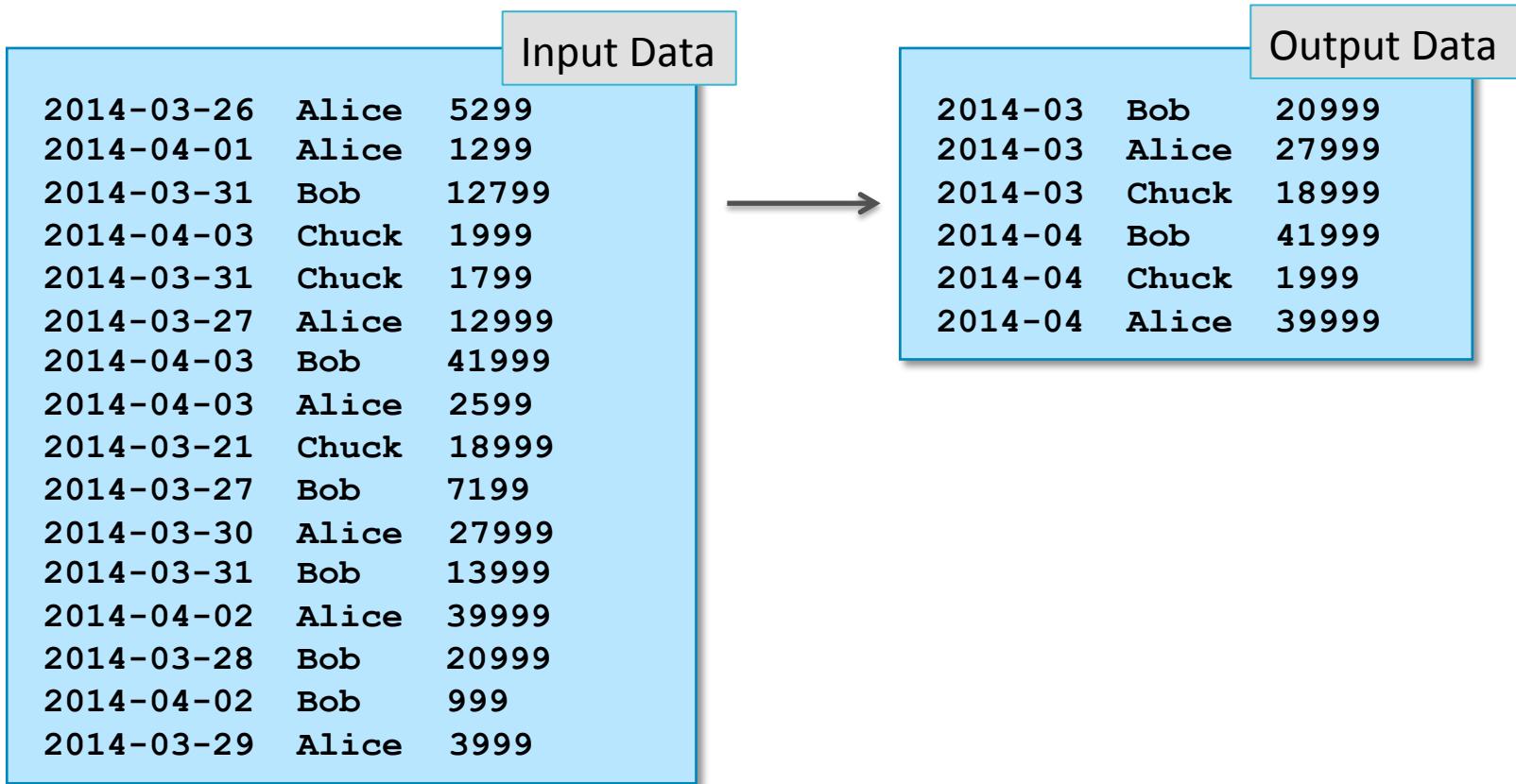
```
PCollection<Pair<String, Integer>> salaries = ... // read data from file  
PCollection<Pair<String, Integer>> sorted = Sort.sortPairs(salaries,  
    new Sort.ColumnOrder(2, Sort.Order.DESCENDING));
```

Secondary Sort

- As with MapReduce, a secondary sort allows you to sort **values**
 - This provides ability to group by both a primary and secondary key
 - Often used for sessionization
- SecondarySort provides several methods called **sortAndApply**
- One such method takes two arguments: **PTable** and **DoFn**
 - Table is of the form PTable<K, Pair<V1, V2>>
 - Where K is the primary grouping key
 - And V1 is the secondary grouping key
 - Function is used to perform additional processing on the sorted records
 - Takes a Pair<K, Iterable<Pair<V1, V2>>> as input
 - Returns a PCollection<T> as output
- Let's look at an example

Sessionization Example: Overview

- Imagine that we have input data containing sales transactions
 - Our goal is to find largest sale for each salesperson every month
 - This is an example of sessionization (by month)



Sessionization Example: Pipeline Code

- Here is a code excerpt from the run method
 - ParseSalesDataFn extracts year/month, name, and price

```
PCollection<String> lines = readTextFile(args[0]);  
  
PTable<String, Pair<String, Integer>> parsed = lines.parallelDo(  
    "parse sales data", new ParseSalesDataFn(),  
    Writables.tableOf(Writables.strings(),  
        Writables.pairs(Writables.strings(), Writables.ints())));  
  
PCollection<String> maxByMonth = SecondarySort.sortAndApply(parsed,  
    new FindMaxSaleFn(), Writables.strings());  
  
// then write maxByMonth as output
```

Sessionization Example: Parse Function

- The parsing function is straightforward

```
static class ParseSalesDataFn extends
    DoFn<String, Pair<String, Pair<String, Integer>>> {

    @Override
    public void process(String input,
        Emitter<Pair<String, Pair<String, Integer>>> output) {

        String[] fields = input.split("\t");

        String yearAndMonth = fields[0].substring(0, 7);
        String salesperson = fields[1];
        int price = Integer.valueOf(fields[2]);

        Pair<String, Pair<String, Integer>> data =
            new Pair (yearAndMonth, Pair.of(salesperson, price));
        output.emit(data);
    }
}
```

Sessionization Example: Find Max Sale Function (1)

- This function iterates over the sorted list of year/month pairs

```
static class FindMaxSaleFn extends
    DoFn<Pair<String, Iterable<Pair<String, Integer>>>, String> {

    @Override
    public void process(Pair<String,
        Iterable<Pair<String, Integer>>> input, Emitter<String> out) {

        String yearAndMonth = input.first();

        // iterate through all sales for the current month
        Iterator<Pair<String, Integer>> iterator =
            input.second().iterator();

        String previousSalesPerson = null;
        int biggestSale = 0;
        while (iterator.hasNext()) {
            Pair<String, Integer> pair = iterator.next();
            String salesPerson = pair.first();
            int price = pair.second();
```

Cont'd...

Sessionization Example: Find Max Sale Function (2)

```
int price = pair.second();
if (previousSalesPerson == null) {
    previousSalesPerson = salesPerson;
}

if (salesPerson.equals(previousSalesPerson)) {
    biggestSale = Math.max(price, biggestSale);
} else {
    out.emit(yearAndMonth + "\t"
        + previousSalesPerson + "\t" + biggestSale);

    previousSalesPerson = salesPerson;
    biggestSale = price;
}

out.emit(yearAndMonth + "\t"
    + previousSalesPerson + "\t" + biggestSale);
}
```

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- **Essential Points**
- Hands-On Exercise: Log Processing with Crunch

Bibliography

The following offer more information on topics discussed in this chapter

- **Apache Crunch *Getting Started Guide***

- <http://tiny.cloudera.com/adcc09a>

- **Apache Crunch *User Guide***

- <http://tiny.cloudera.com/adcc09b>

Essential Points

- **Apache Crunch is an open source library that simplifies parallel processing**
 - Offers a high-level API targeted at Java developers
 - Strikes a good balance between power and productivity
 - Many MapReduce details are abstracted but still available if needed
- **Input, processing, and output are encapsulated in a pipeline**
 - Crunch uses this to lazily create and execute a series of MapReduce jobs
- **Processing logic is encapsulated into DoFn subclasses**
 - These are applied to Crunch PCollection objects
 - The function receives each input record from the PCollection
 - It emits zero or more output records to populate a new PCollection
 - Be careful with serialization in your DoFn subclasses!

Chapter Topics

Processing Data Pipelines with Apache Crunch

- What is Apache Crunch?
- Comparing Crunch to Java MapReduce
- Understanding the Crunch Pipeline
- Unit Testing
- Working with Crunch Projects
- Hands-On Exercise: File Type Processing with Crunch
- Data Collections and Serialization in the Crunch API
- Built-in Utility Classes
- Essential Points
- **Hands-On Exercise: Log Processing with Crunch**

Hands-On Exercise: Log Processing with Crunch

- In this Hands-On Exercise, you will develop Crunch jobs to analyze Web server logs to determine which Knowledge Base articles are the most popular, and which are likely to be the best at solving customers' problems
 - Please refer to the Hands-On Exercise Manual for instructions

Working with Tables in Apache Hive

Chapter 10



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- **Working with Tables in Apache Hive**
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Working with Tables in Apache Hive

In this chapter you will learn

- How Hive generates and runs MapReduce jobs from SQL-like queries
- How to access Hive from the command line, query tool, and Hue
- How to create, populate, query, and delete tables
- How Hive determines the structure and location of table data
- How to perform basic queries in Hive
- What a SerDe is and how it is used in Hive
- How to use the RegexSerDe with semi-structured and fixed-width formats

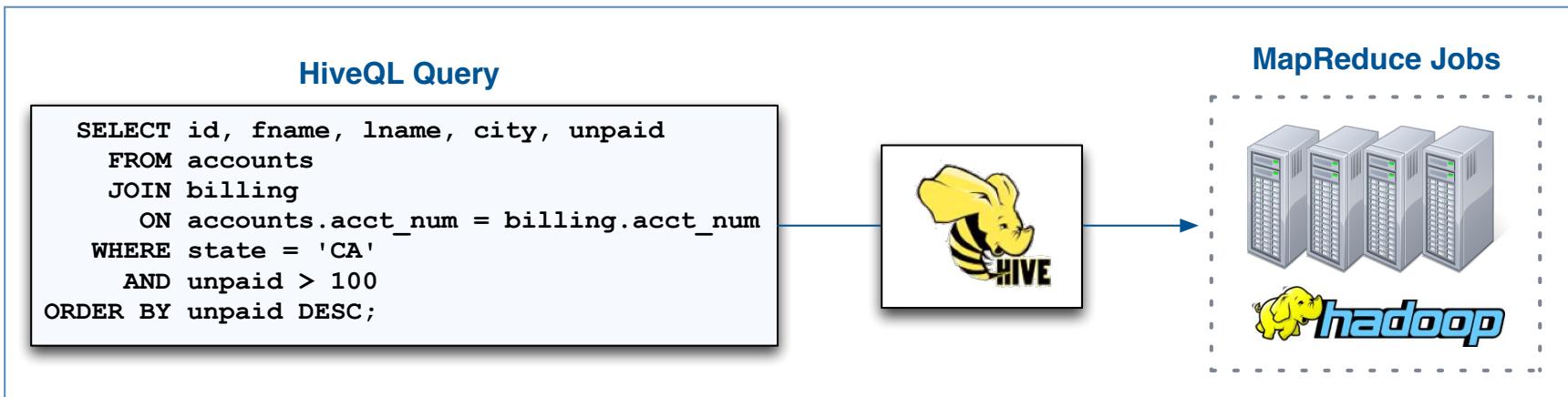
Chapter Topics

Working with Tables in Apache Hive

- **Hive Overview**
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

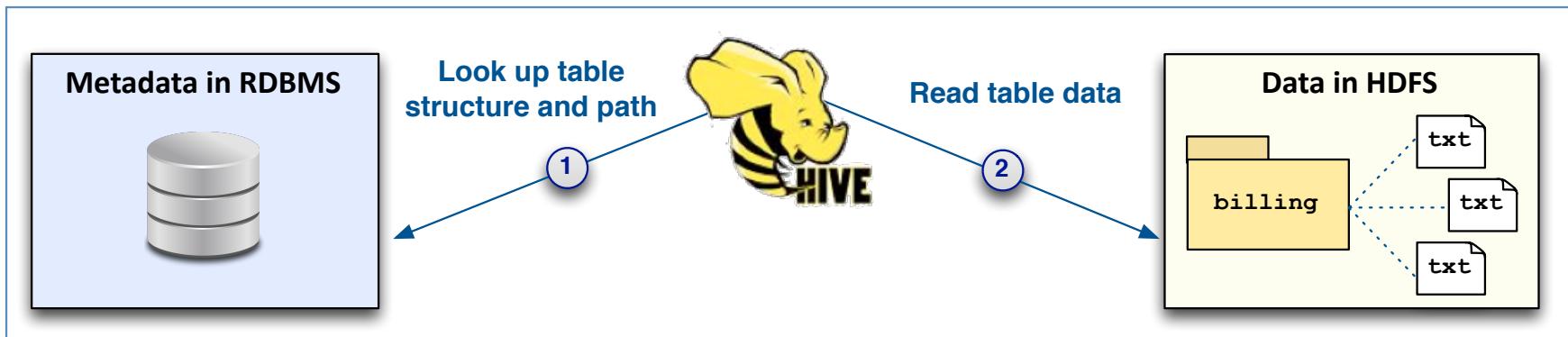
Hive Overview

- Apache Hive is a high-level abstraction on top of MapReduce
 - Interprets a language called HiveQL, based on SQL-92
 - Generates MapReduce jobs that run on the Hadoop cluster
 - Hive does *not* turn your cluster into a database server!
- Easier and faster than writing your own Java MapReduce jobs
 - But amount of *execution time* will be about the same



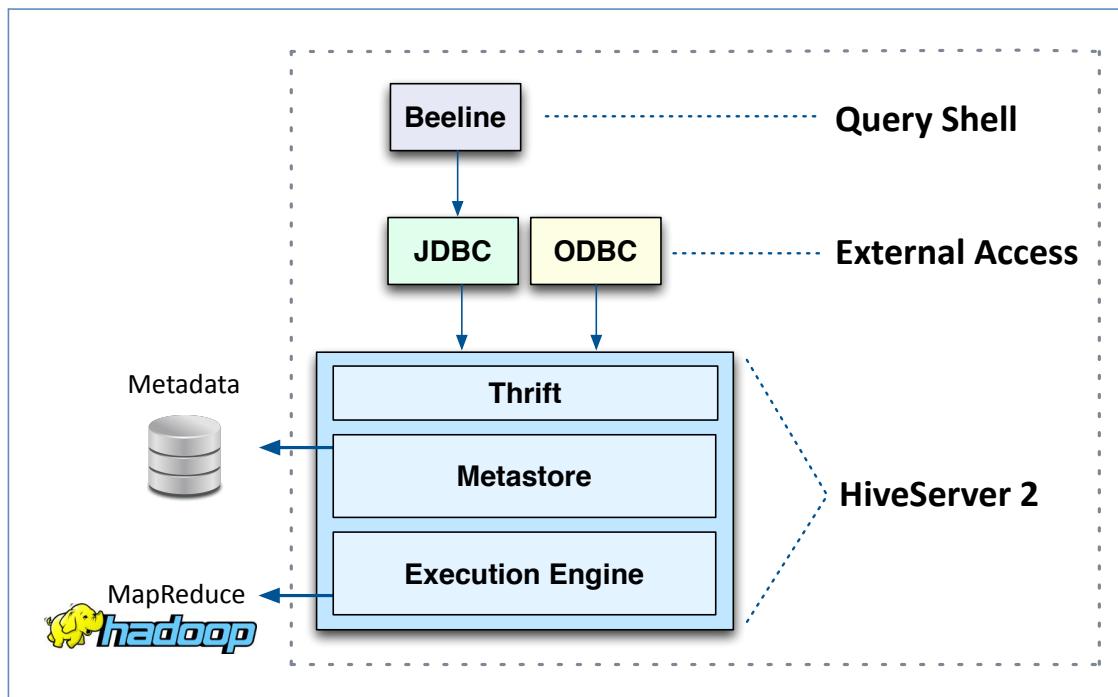
How Data is Stored in Hive

- **Hive's queries operate on tables, as with a relational database**
 - But Hive tables are just a façade for a directory of data in HDFS
 - Default file format is delimited text, but Hive supports many others
- **How does Hive know the structure and location of tables?**
 - These are specified when tables are created
 - This *metadata* is stored in an RDBMS such as MySQL
 - Kite data sets created with the `repo:hive` URI are already known to Hive



Basic Architecture

- A system called **HiveServer 2** is the container for Hive's execution engine
 - Available in Hive 0.11 (CDH 4.1) and later
 - Improves scalability and security compared to earlier versions
- Accessible via Beeline (command line shell), JDBC, ODBC, or Hue (Web UI)



Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- **Accessing Hive**
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

Accessing Hive Using the Beeline Shell

- You can execute HiveQL with Beeline, a replacement for the old Hive shell
 - Start Beeline by specifying login credentials and a JDBC URL
 - Connection details vary based on cluster settings (ask your sysadmin)

```
$ beeline -n alice -p swordfish \
-u jdbc:hive2://dev.loudacre.com/default
```

- You will then see some startup messages followed by a prompt *

```
0: jdbc:hive2://dev.loudacre.com/default>
```

- Use this prompt to enter HiveQL statements (terminated by semicolons)
- Type !quit or press Ctrl+d to exit Beeline and return to the shell

* We will abbreviate this prompt as > in future slides

Executing HiveQL Statements in Batch Mode

- It is sometimes convenient to execute HiveQL statements in batch
 - Rather than interactively from within Beeline
 - As with interactive mode, results are printed to the terminal window
- You can specify HiveQL statements using the **-e** option for Beeline

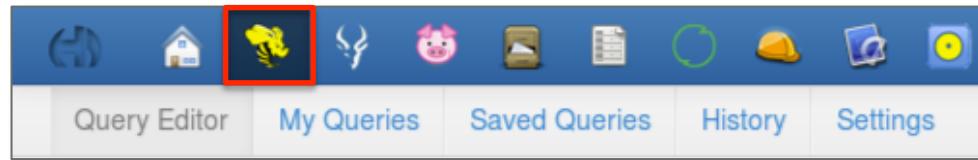
```
$ beeline -n training -p training \
-u jdbc:hive2://dev.loudacre.com/default \
-e 'SELECT * FROM employees WHERE salary > 50000'
```

- Alternatively, save HiveQL statements to a text file and use the **-f** option
 - Especially convenient for automation

```
$ beeline -n training -p training \
-u jdbc:hive2://dev.loudacre.com/default \
-f myqueries.hql
```

Accessing Hive with Hue

- To use Hue, browse to `http://hue_server:8888/`
- Hue provides a Web-based interface to Hive
 - Launch by clicking its icon
- This interface supports:
 - Creating tables
 - Running queries
 - Browsing tables
 - Saving queries for later execution



Executing a Hive Query in Hue (1)

The screenshot shows the Hue web interface for executing Hive queries. The title bar reads "Hue - Hive Query" and the URL is "https://hueserver.example.com:8888/beeswax/". The top navigation bar includes icons for Home, Recent, Print, and Google search, along with a user dropdown for "training". Below the navigation is a toolbar with various icons: a house, a yellow bird, a bull, a pig, a book, a document, a green circle, a hard hat, a blue square, a magnifying glass, and a question mark. The main menu bar has tabs for "Query Editor", "My Queries", "Saved Queries", "History", and "Settings".

The left sidebar contains several sections with "Add" buttons:

- DATABASE:** A dropdown set to "default".
- SETTINGS:** An "Add" button.
- FILE RESOURCES:** An "Add" button.
- USER-DEFINED FUNCTIONS:** An "Add" button.
- PARAMETERIZATION:** A checked checkbox for "Enable Parameterization".
- EMAIL NOTIFICATION:** An unchecked checkbox for "Email me on completion".

The central area is titled "Query Editor" and displays the following SQL query:

```
1 SELECT acct_num, acct_create_dt, first_name, last_name, city
2 FROM account
3 WHERE state = 'CA'
4 ORDER BY acct_create_dt DESC;
```

At the bottom of the query editor are four buttons: "Execute", "Save as...", "Explain", and "or create a New query".

Executing a Hive Query in Hue (2)

The screenshot shows the Hue web interface for executing Hive queries. At the top, there's a browser header with the URL <https://hueserver.example.com:8888/beeswax/watch/6?context=design>. Below the header is a blue navigation bar with various icons and a dropdown menu set to "training". The main content area has tabs for "Query Editor", "My Queries", "Saved Queries", "History", and "Settings". The "Query Editor" tab is active. A large title "Waiting for query... Unsaved Query" is displayed. Below it, there are two tabs: "Log" (selected) and "Query". The "Log" tab displays a log of Hadoop job output:

```
14/02/11 14:28:40 INFO exec.ExecDriver: adding libjars: file:///usr/lib/hive/lib/hive-builtins-0.10.0-cdh4.5.0.jar
14/02/11 14:28:40 INFO exec.ExecDriver: Processing alias account
14/02/11 14:28:40 INFO exec.ExecDriver: Adding input file hdfs://localhost.localdomain:8020/loudacre/account
14/02/11 14:28:40 INFO exec.Utilities: Content Summary not cached for hdfs://localhost.localdomain:8020/loudacre/account
14/02/11 14:28:40 INFO ql.Context: New scratch dir is hdfs://localhost.localdomain:8020/tmp/hive-beeswax-training/hive_2014-02-11_14-28-40_869_1170646905001180238-3
14/02/11 14:28:41 INFO exec.ExecDriver: Making Temp Directory: hdfs://localhost.localdomain:8020/tmp/hive-beeswax-training/hive_2014-02-11_14-28-40_869_1170646905001180238-1-ext-10001
14/02/11 14:28:41 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
14/02/11 14:28:41 INFO io.CombineHiveInputFormat: CombineHiveInputSplit creating pool for hdfs://localhost.localdomain:8020/loudacre/account; using filter path hdfs://localhost.localdomain:8020/loudacre/account
14/02/11 14:28:41 INFO mapred.FileInputFormat: Total input paths to process : 4
14/02/11 14:28:41 INFO io.CombineHiveInputFormat: number of splits 1
Starting Job = job_201402102109_0024, Tracking URL = http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201402102109_0024
14/02/11 14:28:41 INFO exec.Task: Starting Job = job_201402102109_0024, Tracking URL = http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201402102109_0024
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_201402102109_0024
14/02/11 14:28:41 INFO exec.Task: Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_201402102109_0024
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
14/02/11 14:28:45 INFO exec.Task: Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
14/02/11 14:28:45 WARN mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. Use org.apache.hadoop.mapreduce.TaskCounter instead
2014-02-11 14:28:45,426 Stage-1 map = 0%, reduce = 0%
14/02/11 14:28:45 INFO exec.Task: 2014-02-11 14:28:45,426 Stage-1 map = 0%, reduce = 0%
```

Executing a Hive Query in Hue (3)

The screenshot shows the Hue web interface for executing Hive queries. The top navigation bar includes icons for Home, Beeswax, File Browser, History, and Settings, along with a user dropdown for 'training'. The main content area displays the results of an unsaved query. On the left, there's a sidebar with download options (CSV, XLS) and a save button. A 'Did you know?' box provides tips for managing large results. The main table lists 15 rows of data with columns: acct_num, acct_create_dt, first_name, last_name, and city.

	acct_num	acct_create_dt	first_name	last_name	city
0	124754	2014-03-14 23:54:52	Jason	Knox	Industry
1	127332	2014-03-14 22:36:18	Jimmy	Delamora	North Hollywood
2	127758	2014-03-14 22:26:11	John	Leclair	Inglewood
3	121693	2014-03-14 22:22:25	Joel	Carlos	Berkeley
4	124738	2014-03-14 22:11:37	Deborah	Harris	Oakland
5	122322	2014-03-14 21:57:31	Vance	Porter	Palm Springs
6	128693	2014-03-14 21:29:48	Edward	Berg	Fresno
7	127975	2014-03-14 21:14:37	Donna	Gilder	Palo Alto
8	123919	2014-03-14 21:11:31	Sandra	Conrad	Bakersfield
9	123142	2014-03-14 21:01:53	Constance	Lo	Redding
10	124435	2014-03-14 20:53:12	David	Smith	Fresno
11	128048	2014-03-14 20:50:58	Jack	Aronson	Oxnard
12	127006	2014-03-14 20:34:32	Herbert	Snell	Inglewood
13	124608	2014-03-14 20:27:03	Brian	Ward	Oakland
14	126964	2014-03-14 20:17:55	Stacy	Evans	San Bernardino
15	122707	2014-03-14 20:01:20	Doug	Fugate	Bakersfield

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- **Basic Query Syntax**
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

Exploring Hive Tables

- The **SHOW TABLES** command lists all tables in the current Hive database

```
> SHOW TABLES;  
+-----+  
| tab_name |  
+-----+  
| accounts |  
| billing |  
| employees |  
| vendors |  
+-----+
```

- The **DESCRIBE** command lists the fields in the specified table

```
> DESCRIBE vendors;  
+-----+-----+-----+  
| col_name | data_type | comment |  
+-----+-----+-----+  
| vendor_id | int | |  
| date_approved | timestamp | |  
| company_name | string | |  
| phone_number | string | |  
+-----+-----+-----+
```

Basic HiveQL Syntax

- Hive keywords are not case-sensitive, but often capitalized by convention
- Statements may span lines and are terminated by a semicolon
- Comments begin with -- (*double hyphen*)
 - Supported in Hive scripts and Hue, but not in Beeline

```
$ cat nearby_customers.hql

SELECT acct_num, first_name, last_name
FROM accounts
WHERE zipcode='94306';    -- Loudacre headquarters
```

Selecting Data from a Hive Table

- The **SELECT** statement retrieves data from Hive tables
 - Can specify an ordered list of individual columns

```
> SELECT first_name, last_name, city FROM accounts;
```

- An asterisk matches all columns in the table

```
> SELECT * FROM accounts;
```

- You can assign aliases using the **AS** keyword
 - Helpful when creating new columns using an expression

```
> SELECT acct_num AS id, total * 0.1 AS commission  
      FROM sales;
```

Limiting and Sorting Query Results

- The **LIMIT** clause sets the maximum number of rows returned

```
> SELECT acct_num, city FROM accounts LIMIT 10;
```

- Caution: no guarantee regarding *which* 10 results are returned
 - Use ORDER BY for top-N queries
 - The field(s) used in ORDER BY *must* be selected

```
> SELECT acct_num, city FROM customers  
      ORDER BY city DESC LIMIT 10;
```

Using a WHERE Clause to Restrict Results

- WHERE clause restricts rows to those matching specified criteria
 - String comparisons are case-sensitive

```
> SELECT * FROM accounts WHERE acct_num=1287;
```

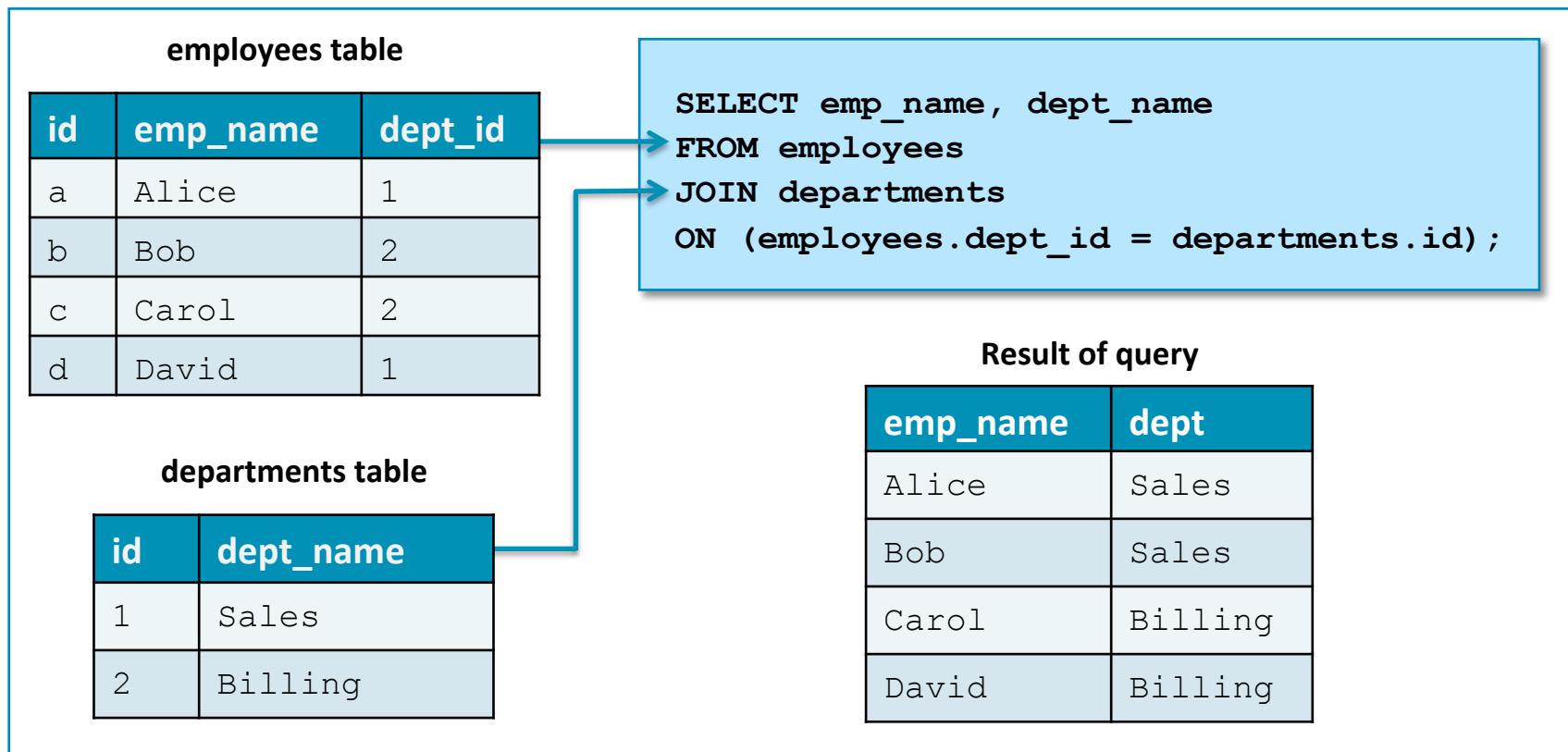
```
> SELECT * FROM accounts WHERE first_name='Anne';
```

- You can combine expressions using AND or OR
- Use the LIKE keyword and a wildcard (%) for inexact string matching

```
hive> SELECT * FROM accounts
      WHERE first_name LIKE 'Ann%'
      AND (city='Seattle' OR city='Portland');
```

Joins in Hive

- Joining disparate data sets is a common use of Hive
 - Caution: note the JOIN ... ON syntax required by Hive
 - For best performance, list the largest table last in your query



Hive Functions

- Hive offers dozens of built-in functions to use in queries
 - Many are identical to those found in SQL
 - Others are Hive-specific
- Example function invocation
 - Function names are not case-sensitive

```
> SELECT CONCAT(first_name, ' ', last_name)  
      AS full_name FROM accounts;
```

Getting Help with Functions

- The SHOW FUNCTIONS command lists all available functions
- Use DESCRIBE FUNCTION to learn about a specific function

```
> DESCRIBE FUNCTION UPPER;  
UPPER(str) - Returns str with all characters  
changed to uppercase
```

- DESCRIBE FUNCTION EXTENDED shows additional information

```
> DESCRIBE FUNCTION EXTENDED UPPER;  
UPPER(str) - Returns str with all characters  
changed to uppercase  
Synonyms: upper, ucase  
Example:  
> SELECT UPPER('Facebook') FROM src LIMIT 1;  
'FACEBOOK'
```

Common Built-in Functions

- These built-in functions operate on one value at a time

Function Description	Example Invocation	Input	Output
Rounds to specified places	ROUND(total_price, 2)	23.492	23.49
Returns nearest integer above	CEIL(total_price)	23.492	24
Returns nearest integer below	FLOOR(total_price)	23.492	23
Extracts year from timestamp	YEAR(order_dt)	2013-06-14 16:51:05	2013
Extracts portion of string	SUBSTRING(name, 0, 3)	Benjamin	Ben
Converts timestamp from specified timezone to UTC	TO_UTC_TIMESTAMP(ts, 'PST')	2014-02-27 15:03:14	
Converts to another type	CAST(weight as INT)	3.581	3

Record Grouping for Use with Aggregate Functions

- **GROUP BY** groups selected data by one or more columns
 - Columns not part of aggregation *must* be listed in GROUP BY

vendors table

id	city	state	region
a	Anaheim	CA	SOUTH
b	Bellevue	WA	NORTH
c	Camarillo	CA	SOUTH
d	Davis	CA	SOUTH
e	Eugene	OR	NORTH
f	Flagstaff	AZ	SOUTH
g	George	WA	NORTH

```
SELECT region, state,  
       COUNT(id) AS num  
  FROM vendors  
 GROUP BY region, state;
```

Result of query

region	state	num
NORTH	OR	1
NORTH	WA	2
SOUTH	AZ	1
SOUTH	CA	3

Built-in Aggregate Functions

- Hive has many built-in aggregate functions, including

Function Description	Example Invocation
Counts all rows	COUNT (*)
Counts number of non-null values for a given field	COUNT (first_name)
Counts number of unique, non-null values for field	COUNT (DISTINCT fname)
Returns the largest value	MAX (salary)
Returns the smallest value	MIN (salary)
Returns total of selected values	SUM (price)
Returns the average of all supplied values	AVG (salary)

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- **Creating and Populating Hive Tables**
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

Hive Data Types

- **Each column in a Hive table is assigned a specific data type**
 - These are specified when the table is created
 - Hive returns NULL values for non-conforming data in HDFS
- **Here are some common data types in Hive**
 - Hive also supports a few complex types such as maps and arrays

Name	Description	Example Value
STRING	Character data (of any length)	Alice
BOOLEAN	True or False	TRUE
TIMESTAMP	Instant in time	2014-03-14 17:01:29
INT	Range: same as Java int	84127213
BIGINT	Range: same as Java long	7613292936514215317
FLOAT	Range: same as Java float	3.14159
DOUBLE	Range: same as Java double	3.1415926535897932385

Creating a Table in Hive

- The following example creates a new table named **products**
 - Data stored in text file format with four delimited fields per record

```
CREATE TABLE products (
    id INT,
    name STRING,
    price INT,
    date_added TIMESTAMP
);
```

- Default field delimiter is \001 (Ctrl-A) and line ending is \n (newline)
- Example of corresponding records for the table above

```
1^AUSB Cable^A799^A2014-03-11 17:23:19\n
2^AScreen Cover^A3499^A2014-03-12 08:41:26\n
```

Changing the Default Field Delimiter When Creating a Table

- If you have tab-delimited data, you would create the table like this
 - Data stored as text with four tab-delimited fields per record

```
CREATE TABLE products (
    id INT,
    name STRING,
    price INT,
    date_added TIMESTAMP
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

- Example of corresponding records for the table above

```
1\tUSB Cable\t799\t2014-03-11 17:23:19\n
2\tScreen Cover\t3499\t2014-03-12 08:41:26\n
```

Creating a Table with Sequence File Format

- Creating tables that will be populated with SequenceFiles is also easy

```
CREATE TABLE products (
    id INT,
    name STRING,
    price INT,
    date_added TIMESTAMP
)
STORED AS SEQUENCEFILE;
```

- STORED AS TEXTFILE is the default and is rarely specified explicitly

Creating a Table with Avro File Format

- **Avro-based tables must specify a few more details**
 - Of these, only the table name and schema URL typically change
 - Field names and types are defined in the schema

```
CREATE TABLE products
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS
INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
TBLPROPERTIES
  ('avro.schema.url'='hdfs://dev.loudacre.com:8020/schemas/products.avsc');
```

Removing a Table

- Use **DROP TABLE** to remove a table from Hive

```
> DROP TABLE products;
```

- Add **IF EXISTS** to avoid an error if the table does not already exist
 - Useful for scripting tasks

```
> DROP TABLE IF EXISTS products;
```

- Caution: dropping a table is a destructive operation
 - This *will* remove metadata and *may* remove data from HDFS
 - Hive does not have a rollback or undo feature

Specifying Table Data Location

- By default, table data is stored beneath Hive's 'warehouse' directory
 - Path will be /user/hive/warehouse/<tablename>
- Storing data below Hive's warehouse directory is not always ideal
 - Data might already exist in a different location
- Use LOCATION clause during creation to specify alternate directory

```
CREATE TABLE products (
    id INT,
    name STRING,
    price INT,
    date_added TIMESTAMP
)
LOCATION '/loudacre/products';
```

Self-Managed (External) Tables

- Related to this is Hive's management of the data
 - Dropping a table removes data in HDFS
- Using EXTERNAL when creating the table avoids this behavior
 - Dropping this table affects only *metadata*
 - This is a better choice if you intend use this data outside of Hive
 - Almost always used in conjunction with the LOCATION keyword

```
CREATE EXTERNAL TABLE products (
    id INT,
    name STRING,
    price INT,
    date_added TIMESTAMP
)
LOCATION '/loudacre/products' ;
```

Loading Data Into Hive Tables

- Remember, each Hive table is mapped to a directory in HDFS
 - Can populate a table by adding one or more files to this directory
 - Place file(s) in the root of table directory (subdirectories not allowed)

```
$ hadoop fs -mv newaccounts.csv /user/hive/warehouse/accounts/
```

- Hive also provides a LOAD DATA command
 - Equivalent to above, but does not require you to specify table path

```
> LOAD DATA INPATH 'newaccounts.csv' INTO TABLE accounts;
```

- Unlike an RDBMS, Hive does not validate data on insert
 - Missing or invalid data will be represented as NULL in query results

Appending Selected Records to a Table

- Another way to populate a table is through a query
 - Use INSERT INTO to append results to an *existing* Hive table

```
> INSERT INTO TABLE accounts_copy  
      SELECT * FROM accounts;
```

- Specify a WHERE clause to control which records are appended

```
> INSERT INTO TABLE loyal_customers  
      SELECT * FROM accounts  
      WHERE YEAR(acct_create_dt) = 2008  
            AND acct_close_dt IS NULL;
```

Creating and Populating a Tables using CTAS

- You can also create *and* populate a table with a single statement
 - This technique is known as ‘Create Table As Select’ (CTAS)
 - Column names and types are derived from the source table

```
> CREATE TABLE loyal_customers AS  
    SELECT * FROM accounts  
    WHERE YEAR(acct_create_dt) = 2008  
        AND acct_close_dt IS NULL;
```

- New table uses Hive’s default format, but you can specify an alternate format

```
> CREATE TABLE california_customers  
    STORED AS SEQUENCEFILE  
    AS  
    SELECT * FROM accounts  
    WHERE state = 'CA';
```

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- **Hands-On Exercise: Running Queries in Hive**
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

Hands-On Exercise: Running Queries in Hive

- In this Hands-On Exercise, you will create Hive tables for data you have created in earlier exercises, and then query these new tables
 - Please refer to the Hands-On Exercise Manual for instructions

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- **How Hive Reads Data**
- Using the RegexSerDe in Hive
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

Recap: Reading Data in MapReduce

- A MapReduce job's `InputFormat` class controls how records are read from files
 - `TextInputFormat` (default) supplies each line of text as the value
 - It is *your* responsibility to parse content into specific fields

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

    @Override
    public void map(LongWritable key, Text value, Context ctx)
        throws IOException, InterruptedException {

        String line = value.toString();

        String[] fields = line.split("\t");
        int id = Integer.valueOf(fields[0]);
        String salesperson = fields[1];
        int price = Integer.valueOf(fields[2]);
```

How Hive Reads Records from Files

- **Hive queries are executed as MapReduce jobs**
 - Records are read using the `InputFormat` associated with the table
 - Similarly, records are written using the table's `OutputFormat`
 - These formats are specified or implied during table creation
- **As with MapReduce, `TextInputFormat` is used by default**
 - Each line from the input file(s) is considered a record

```
CREATE TABLE sales (
    id INT,
    salesperson STRING,
    price INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

Specifying the Input and Output Format in Hive

- The file type implies which input and output formats are used for the table

```
CREATE TABLE people (
    id INT,
    name STRING
)
STORED AS SEQUENCEFILE;
```

- This is an alternative to specifying input and output formats explicitly

```
CREATE TABLE people (
    id INT,
    name STRING
)
STORED AS
    INPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileInputFormat'
    OUTPUTFORMAT 'org.apache.hadoop.hive.io.HiveSequenceFileOutputFormat'
```

How Hive Extracts Fields from Records

- You have learned how Hive reads records from a file
 - How does Hive retrieve fields from those records?
- Hive uses a class called a SerDe to extracts fields from each record
 - SerDe is short for ‘Serializer/Deserializer’
- The default SerDe (`LazySimpleSerDe`) parses fields based on delimiter
 - Used if `ROW FORMAT DELIMITED` specified in `CREATE TABLE`
 - Also used if `ROW FORMAT` is missing from `CREATE TABLE`

```
CREATE TABLE sales (
    id INT,
    salesperson STRING,
    price INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

Common SerDe Implementations

- Hive in CDH ships with many SerDes, including:

Class Name	Reads and Writes Records
LazySimpleSerDe	Using specified field delimiters (default)
AvroSerDe	Using Avro, according to specified schema
ColumnarSerDe	Using the RCFile columnar format
ParquetHiveSerDe	Using the Parquet columnar format
RegexSerDe	Using the supplied regular expression

- See Hive documentation for fully-qualified class names
- Not all SerDes are compatible with all file formats
 - Cannot use AvroSerDe with SequenceFiles, for example

Specifying the SerDe Type

- A SerDe can be specified explicitly in the CREATE TABLE statement

```
CREATE TABLE products
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS
INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
TBLPROPERTIES
  ('avro.schema.url'='hdfs://dev.loudacre.com:8020/schemas/products.avsc');
```

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- **Using the RegexSerDe in Hive**
- Essential Points
- Hands-On Exercise: Using the RegexSerDe in Hive

About the RegexSerDe

- Hadoop is a great fit for semi-structured data
 - Many formats typically lack a consistent delimiter character

```
03/14/2014@15:25:47 415-555-2854:312-555-7819 "Call placed"
03/14/2014@15:25:53 415-555-2854:312-555-7819 "Call dropped"
```

- RegexSerDe can help you to parse such data
 - Regular expressions provide great flexibility
 - Example: using RegexSerDe allows you to perform queries on log files
- Points to note about RegexSerDe
 - Only supports STRING column type in Hive 0.10 and earlier
 - Backslashes in regular expressions must be escaped with a backslash

Using the RegexSerDe

03/14/2014@15:25:47 415-555-2854:312-555-7819 "Call placed"
03/14/2014@15:25:53 415-555-2854:312-555-7819 "Call dropped"

Input

```
CREATE TABLE calls (
    event_date STRING,
    event_time STRING,
    from_number STRING,
    to_number STRING,
    event_desc STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" =
    "(.*?)@(.*) (.*):(.*) \"([^\"]*)\"");

```

SerDe

event_date	event_time	from_number	to_number	event_desc	Table
03/14/2014	15:25:47	415-555-2854	312-555-7819	Call placed	
03/14/2014	15:25:53	415-555-2854	312-555-7819	Call dropped	

Table

Supporting Fixed-Width Formats with RegexSerDe

- Hive does not have a specific SerDe to support fixed-width formats
 - Yet these are commonly produced by older applications
 - Fixed-width data is easy to read with RegexSerDe

2014-03-1212345673661845I love the guitar solo
2014-03-1216431444025652Hard to dance to this one
2014-03-1412957863971103It needs more cowbell

Raw Data

```
CREATE TABLE ... WITH SERDEPROPERTIES ("input.regex" =  
    "(.{10})(\\d{6})(\\d{7})(\\d)(.*)");
```

SerDe (Excerpt)

comment_date	acct_num	prod_id	rating	comments
2014-03-12	123456	7366184	5	I love the guitar solo
2014-03-12	164314	4402565	2	Hard to dance to this one
2014-03-14	129578	6397110	3	It needs more cowbell

Resulting Table

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- **Essential Points**
- Hands-On Exercise: Using the RegexSerDe in Hive

Bibliography

The following offer more information on topics discussed in this chapter

- **Apache Hive Web Site**

- <http://hive.apache.org/>

- **Demo: Analyzing Data with Hue and Hive**

- <http://tiny.cloudera.com/adcc10a>

- **HiveQL Language Manual**

- <http://tiny.cloudera.com/adcc10b>

- **How HiveServer2 Brings Security and Concurrency to Apache Hive**

- <http://tiny.cloudera.com/adcc10c>

- **Migrating from Hive CLI to Beeline: A Primer**

- <http://tiny.cloudera.com/adcc10d>

Essential Points

- **Hive is a high-level abstraction on top of MapReduce**
 - Interprets queries in a SQL-like language called HiveQL
 - Fulfils queries by executing MapReduce jobs on the Hadoop cluster
- **Hive tables are simply directories in HDFS**
 - Only the table's metadata is stored in a relational database
 - Data structure and format is specified in the CREATE TABLE statement
 - Table data is delimited text by default, but Hive supports many formats

Chapter Topics

Working with Tables in Apache Hive

- Hive Overview
- Accessing Hive
- Basic Query Syntax
- Creating and Populating Hive Tables
- Hands-On Exercise: Running Queries in Hive
- How Hive Reads Data
- Using the RegexSerDe in Hive
- Essential Points
- **Hands-On Exercise: Using the RegexSerDe in Hive**

Hands-On Exercise: Using the RegexSerDe in Hive

- In this Hands-On Exercise, you will use the RegexSerDe in Hive to create a table based on a fixed-width data set
 - Please refer to the Hands-On Exercise Manual for instructions

Developing User-Defined Functions

Chapter 11



Course Chapters

- Introduction
- Application Architecture
- Simplifying Development with the Kite SDK
- Designing and Using Data Sets
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- **Developing User-Defined Functions**
- Executing Interactive Queries with Impala
- Enabling Full-Text Search of Data Stored in HDFS
- Presenting Results to Users
- Conclusion

Developing User-Defined Functions

In this chapter you will learn

- Which types of user-defined functions Hive supports
- How to implement a user-defined function
- How to deploy and register a user-defined function for use with Hive

Chapter Topics

Developing User-Defined Functions

- **User-Defined Function Overview**
- Implementing a User-Defined Function
- Deploying Custom Libraries in Hive
- Registering a User-Defined Function in Hive
- Essential Points
- Hands-On Exercise: Implementing a User-Defined Function in Hive

Why Create User-Defined Functions?

- **User-defined functions allow you to extend the capabilities of Hive**
 - You might need to reformat values in a query
 - You might need to add new mathematical functions
 - You might need to include support for custom processing

Types of User-Defined Functions

- There are three distinct types of user-defined functions
- UDF (User-Defined Function)
 - Operates on exactly one record and returns a single value
 - Examples include SUBSTRING and ROUND
- UDAF (User-Defined Aggregate Function)
 - Operates on multiple records and returns a single value
 - Usually used in conjunction with a GROUP BY clause
 - Examples include SUM and AVG
- UDTF (User-Defined Table Function)
 - Operates on a single record but returns multiple records
 - The EXPLODE function is an example of a UDTF
- The first type is the most common and we will focus on it in this chapter

Chapter Topics

Developing User-Defined Functions

- User-Defined Function Overview
- **Implementing a User-Defined Function**
- Deploying Custom Libraries in Hive
- Registering a User-Defined Function in Hive
- Essential Points
- Hands-On Exercise: Implementing a User-Defined Function in Hive

Maven Configuration for Custom UDF Development

- To develop a custom UDF, first add the following to your `pom.xml`

```
<dependency>
  <artifactId>hive-exec</artifactId>
  <groupId>org.apache.hive</groupId>
  <version>${hive.version}</version>
</dependency>
```

- Value of `hive.version` varies based on your CDH release
 - Use the Maven repository Web UI to search for available versions
 - <https://repository.cloudera.com/>

Developing and Using a Custom UDF

- **The easiest way to develop a UDF is to extend the UDF class**
 - Implementation differs for UDAFs and UDTFs
- **Your UDF must implement one or more methods named evaluate**
 - Each method can accept a different type of argument
 - Methods *should* accept and return Writable types
- **The UDF should also use a Description annotation**
 - Provides information used in SHOW FUNCTION command

Simple UDF Example

- The simplest UDF implementation just returns a constant value

```
package com.loudacre.example;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.DoubleWritable;

@Description (
    name="KPM",
    value="_FUNC_() - returns the number of kilometers per mile",
    extended = "Example:\n"
        + " > SELECT _FUNC_() * distance FROM locations LIMIT 1;\n"
        + " 482.802")
public class KilometersPerMile extends UDF {

    private final DoubleWritable result = new DoubleWritable(1.60934);

    public DoubleWritable evaluate() {
        return result;
    }
}
```

Price Formatting UDF Example (1)

- This example can accept either one or two arguments
 - Example: FORMAT_CENTS (375) returns \$3.75
 - Example: FORMAT_CENTS (375, '€') returns €3.75

```
package com.loudacre.example;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

@Description(
    name = "FORMAT_CENTS",
    value = "_FUNC_(cents) - Formats INT value as dollars and cents.",
    extended = "Example:\n"
        + " > SELECT _FUNC_(price_in_cents) FROM sales;\n"
        + " > SELECT _FUNC_(price_in_eurocents, '€') FROM sales;\n"
)
public class FormatCents extends UDF {
    // continued on next slide
```

Price Formatting UDF Example (2)

- The class defines two **Text** object as member variables
 - It is more efficient to reuse objects rather than creating new instances each time
- This **evaluate** method invokes the two-argument version
 - By supplying the default symbol

```
// continued from previous slide
private static final Text DEFAULT_SYMBOL = new Text("$");
private final Text result = new Text();

public Text evaluate(IntWritable centsAsWritable) {
    return evaluate(centsAsWritable, DEFAULT_SYMBOL);
}

// continued on next slide
```

Price Formatting UDF Example (3)

```
// continued from previous slide

public Text evaluate(IntWritable centsAsWritable, Text symbol) {
    int cents = centsAsWritable.get();

    StringBuilder sb = new StringBuilder();
    if (cents <= 9) {
        sb.append("00");
    } else if (cents <= 99) {
        sb.append("0");
    }

    sb.append(cents);
    sb.insert(sb.length() - 2, ".");
    sb.insert(0, symbol);

    result.set(sb.toString());

    return result;
}
}
```

Chapter Topics

Developing User-Defined Functions

- User-Defined Function Overview
- Implementing a User-Defined Function
- **Deploying Custom Libraries in Hive**
- Registering a User-Defined Function in Hive
- Essential Points
- Hands-On Exercise: Implementing a User-Defined Function in Hive

Deploying Custom Libraries to Hive (1)

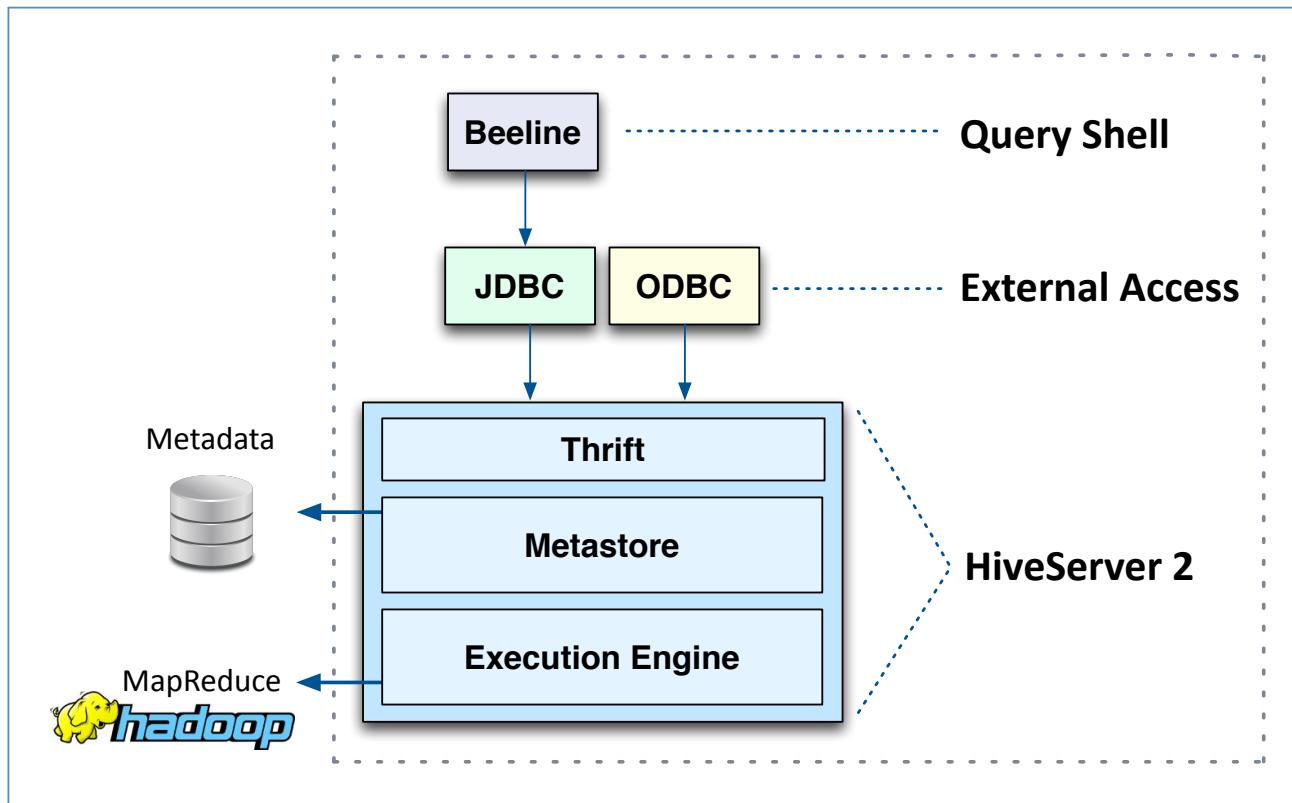
- The first step in deployment is to package your UDF as a JAR file

```
$ mvn package
```

- Deployment may require assistance from your system administrator
 - Your account may lack permissions to make the necessary changes

Deploying Custom Libraries to Hive (2)

- Older versions of Hive used the ADD JAR command
 - This approach does not work with HiveServer2
 - We need to update the HiveServer2 daemon's classpath



Deploying Custom Libraries to Hive (3)

- Edit `/etc/hive/conf/hive-site.xml` on the node running `HiveServer2`
 - Add or edit the `hive.aux.jars.path` property
 - Set its value to the fully-qualified URI to your JAR file(s)
 - Multiple URIs are separated by commas for this property

```
<property>
  <name>hive.aux.jars.path</name>
  <value>file:///home/bob/myudf.jar,file:///opt/lib/foo.jar</value>
</property>
```

- `HiveServer2` will add these JARs to the distributed cache when its MapReduce jobs are run
- We could also upload the JAR file to HDFS
 - In this case, the URI would begin with `hdfs://` instead of `file://`

Deploying Custom Libraries to Hive (4)

- **Edit /etc/default/hive-server2 on node running HiveServer2**
 - Set AUX_CLASSPATH environment variable to include all your JARs
 - The value follows the typical classpath format (colon delimited)
 - The value must always consist of *local* paths
 - This step adds the libraries to HiveServer2's runtime classpath

```
export AUX_CLASSPATH=/home/bob/myudf.jar:/opt/lib/foo.jar
```

- **Finally, restart the hive-server2 service**
 - You must redeploy the JAR and restart the service whenever you modify the code

```
$ sudo service hive-server2 restart
```

Deploying Custom Libraries to Hive (5)

- **Error messages in Beeline can be cryptic**
- **If you have problems with a newly-registered UDF, check the logs**
 - Typically located in the `/var/log/hive` directory

Chapter Topics

Developing User-Defined Functions

- User-Defined Function Overview
- Implementing a User-Defined Function
- Deploying Custom Libraries in Hive
- **Registering a User-Defined Function in Hive**
- Essential Points
- Hands-On Exercise: Implementing a User-Defined Function in Hive

Registering a User-Defined Function in Hive

- Your UDF must be registered before using it in a query
- This is done using the CREATE TEMPORARY FUNCTION command in Hive

```
> CREATE TEMPORARY FUNCTION KPM  
    AS 'com.loudacre.example.KilometersPerMile';  
  
> SELECT KPM() * miles_to_headquarters AS km_to_headquarters  
  FROM store_locations  
 WHERE store_id=1234;
```

- Repeat this step during every Hive session where it is used

Chapter Topics

Developing User-Defined Functions

- User-Defined Function Overview
- Implementing a User-Defined Function
- Deploying Custom Libraries in Hive
- Registering a User-Defined Function in Hive
- **Essential Points**
- Hands-On Exercise: Implementing a User-Defined Function in Hive

Bibliography

The following offer more information on topics discussed in this chapter

- **Programming Hive (book)**

- <http://tiny.cloudera.com/adcc11a>

- **Hive Built-In Functions**

- <http://tiny.cloudera.com/adcc11b>

- **Using a UDF in Hue**

- <http://tiny.cloudera.com/adcc11c>

Essential Points

- **The easiest way to create a UDF in Hive is to extend the UDF class**
 - The Description annotation is used by SHOW FUNCTION
- **UDF logic is implemented one or more methods named evaluate**
 - Each method can accept a different type of argument
 - Methods should accept and return Writable types
- **The UDF JAR and its dependencies are deployed to HiveServer2**
 - Involves editing two configuration files and restarting the daemon
 - May require assistance from your system administrator
 - You must redeploy following any update to the UDF code
- **Register UDF with CREATE TEMPORARY FUNCTION**
 - Required step in any Hive session where the function is used

Chapter Topics

Developing User-Defined Functions

- User-Defined Function Overview
- Implementing a User-Defined Function
- Deploying Custom Libraries in Hive
- Registering a User-Defined Function in Hive
- Essential Points
- **Hands-On Exercise: Implementing a User-Defined Function in Hive**

Hands-On Exercise: Implementing a User-Defined Function in Hive

- In this Hands-On Exercise, you will create a custom UDF that formats phone numbers. You will then deploy the UDF and use it in queries
 - Please refer to the Hands-On Exercise Manual for instructions

Executing Interactive Queries with Impala

Chapter 12



Course Chapters

- Introduction
- Application Architecture
- Simplifying Development with the Kite SDK
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- **Executing Interactive Queries with Impala**
- Enabling Full-Text Search of Data Stored in HDFS
- Presenting Results to Users
- Conclusion

Executing Interactive Queries with Impala

In this chapter you will learn

- How Impala compares to Hive and relational databases
- How to run Impala queries from the shell or browser
- How Impala's query syntax compares to HiveQL
- How Impala achieves high performance for interactive queries
- How and when to refresh Impala's metadata cache
- How to use a custom User-Defined Function (UDF) in Impala

Chapter Topics

Executing Interactive Queries with Impala

- **What is Impala?**
- Comparing Impala with Hive
- Running Queries in Impala
- Support for User-Defined Functions
- Data and Metadata Management
- Essential Points
- Optional Hands-On Exercise: Running Queries in Impala

What is Impala?

- **Impala is a high-performance SQL engine for data stored in Hadoop**
 - Inspired by Google's Dremel project
 - Reads and writes data in common Hadoop file formats
- **Impala supports a subset of SQL-92**
 - Syntax is nearly identical to HiveQL
- **Designed for low latency, ideal for interactive queries**
 - Impala is *significantly* faster than Hive
 - Accessible from Impala shell, Hue, ODBC, and JDBC
- **Developed by Cloudera and released as 100% open source**
 - Uses the same Apache software license as Hadoop itself



cloudera®
IMPALA

Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- **Comparing Impala with Hive**
- Running Queries in Impala
- Support for User-Defined Functions
- Data and Metadata Management
- Essential Points
- Optional Hands-On Exercise: Running Queries in Impala

Comparing Impala to Hive

- **Impala and Hive are similar in many ways**
 - Use a variant of SQL to query data stored in Hadoop
 - More productive than writing MapReduce code in Java
 - Tables created in Hive are visible in Impala (and vice versa)
- **Hive and Impala support many of the same features**
 - Executing queries via interactive shell, Hue, ODBC, and JDBC
 - Grouping, joining, and filtering data
 - Reading and writing data in multiple formats
 - Support Custom User-Defined Functions (UDFs and UDAFs)

Contrasting Impala with Hive

- **Unlike Hive, Impala does not execute queries via MapReduce**
 - Impala uses a custom execution engine written in C++
 - Tuned for peak performance with interactive queries
 - Queries can complete in a fraction of a second
- **However, Impala currently lacks a few features from Hive, including**
 - Nested data types like arrays and maps
 - Support for custom SerDes
 - Support for User-Defined Table Functions (UDTFs)
- **Many of these features are being considered for future releases**
- **Hive is better for large, long-running batch processes**
 - Impala is best for interactive/ad hoc queries

Query Fault Tolerance in Impala

- **Queries in both Hive and Impala are distributed across nodes**
- **Hive answers queries by running MapReduce jobs**
 - Takes advantage of Hadoop's fault tolerance
 - If a node fails during the query, MapReduce runs the task elsewhere
- **Impala does not use MapReduce**
 - Its query engine currently lacks fault tolerance
 - If a node fails during a query, the query will fail
 - Workaround: just re-run the query

Impala Query Size Limitations

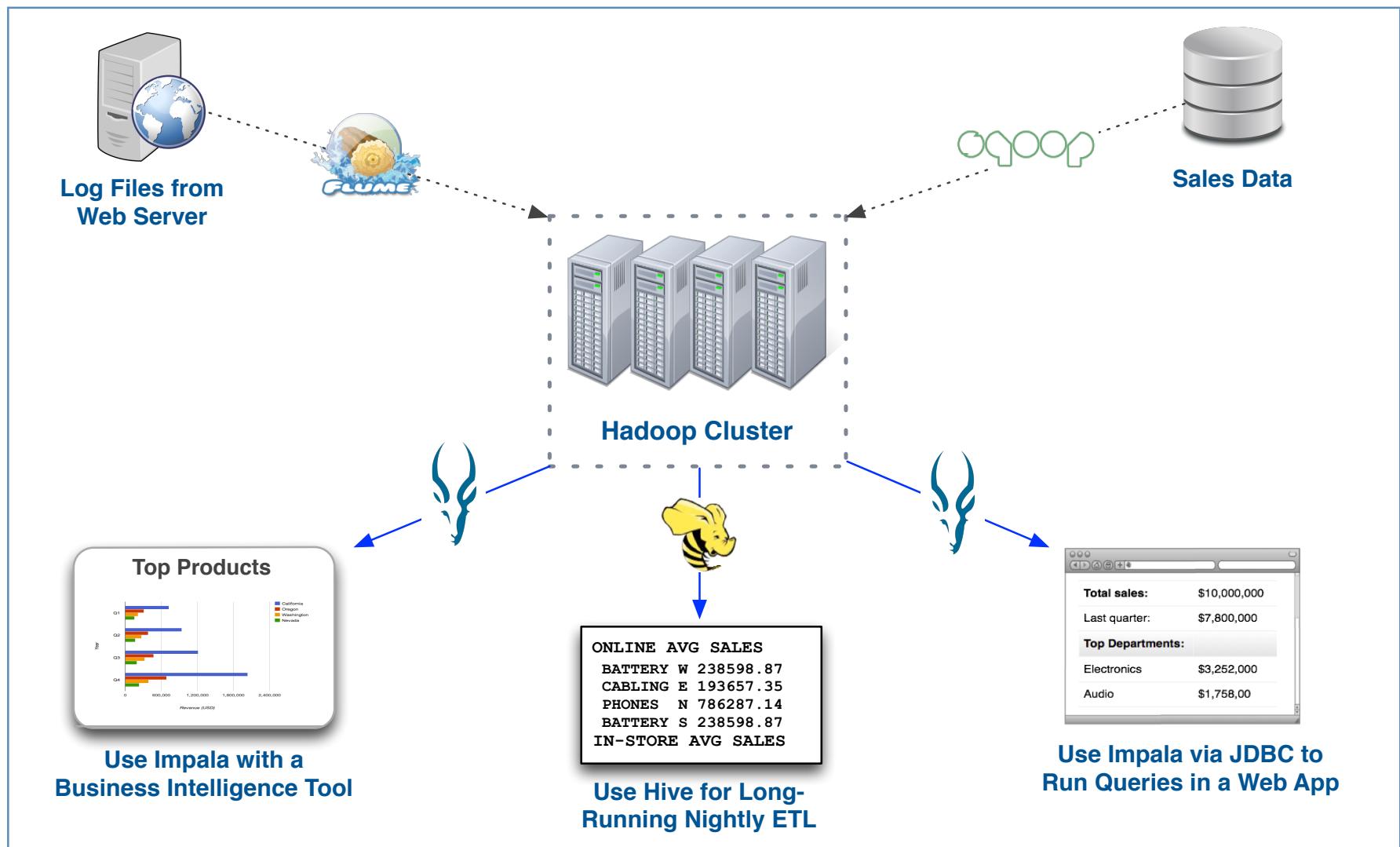
- **Query size is based on a query's *working set size***
- **The working set of a query contains all records after**
 - Filtering rows
 - Pruning unused columns
 - Performing aggregation, if applicable
- **Impala query results must fit into the cluster's aggregate memory**
 - For aggregation, this is the working set size for all tables in the query
 - For joins, this is the working set size for all but the largest table
- **Use Hive if your query exceeds these limits**
 - Alternatively, ask your system administrator to install more RAM

Comparing Impala To A Relational Database

- Unlike Hive, Impala is suitable for interactive applications
 - However, it is not a replacement for a transactional RDBMS

Feature	Relational Database	Impala
Update individual records	Yes	No
Delete individual records	Yes	No
Transactions	Yes	No
Stored procedures	Yes	No
Referential integrity	Yes	No
When schema is applied	On Write	On Read
Scalability	Low	High
Data Storage Cost	Expensive	Inexpensive

Choosing the Right Tool for the Job



Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- Comparing Impala with Hive
- **Running Queries in Impala**
- Support for User-Defined Functions
- Data and Metadata Management
- Essential Points
- Optional Hands-On Exercise: Running Queries in Impala

Starting the Impala Shell

- You can execute statements in the Impala's shell
 - This interactive tool is similar to the Beeline shell for Hive
- Execute the `impala-shell` command to start the shell

```
$ impala-shell
Connected to dev.loudacre.com:21000
Server version: impalad version 1.2.3
Welcome to the Impala shell.
[dev.loudacre.com:21000] >
```

- (Some log messages truncated to better fit the slide)
- Use `-i hostname:port` option to connect to another server

```
$ impala-shell -i myserver.example.com:21000
[myserver.example.com:21000] >
```

Using the Impala Shell

- Enter semicolon-terminated statements at the prompt
 - Hit Enter to execute the query
 - Use the quit command to exit the Impala shell

```
$ impala-shell
> SELECT acct_num, first_name, last_name FROM accounts
   WHERE zipcode='90210' LIMIT 2;
+-----+-----+-----+
| acct_num | first_name | last_name |
+-----+-----+-----+
| 6029     | Brian      | Ferrara   |
| 9493     | Mickey     | Kunkle    |
+-----+-----+-----+
> quit;
$
```

Note: shell prompt abbreviated as >

Running Queries from the Command Line

- You can execute a file containing queries using the **-f** option

```
$ impala-shell -f myquery.sql
```

- Run queries directly from the command line with the **-q** option

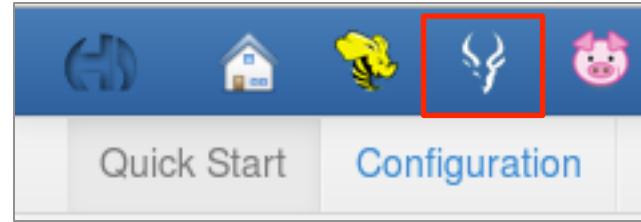
```
$ impala-shell -q 'SELECT COUNT(*) FROM accounts'
```

- Use **-o** (and optionally specify a delimiter) to capture output to file

```
$ impala-shell -f myquery.sql \
-o results.txt \
--output_file_field_delim='\t'
```

Accessing Impala with Hue (1)

- Alternatively, you can access Impala through Hue
- To use Hue, browse to `http://hue_server:8888/`
- Launch Hue's Impala interface by clicking its icon



Impala icon in Hue

Accessing Impala with Hue (2)

- Hue allows you to run Impala queries from your Web browser

The screenshot shows the Hue - Hive Query interface in a web browser. The URL is <https://hueserver.example.com:8888/impala/>. The top navigation bar includes icons for Home, Recent, and Help, along with a user dropdown labeled "training". Below the bar are tabs for "Query Editor", "My Queries", "Saved Queries", "History", and "Settings".

The main area is titled "Query Editor". On the left, there's a sidebar with sections for "DATABASE" (set to "default"), "SETTINGS" (with an "Add" button), "PARAMETERIZATION" (checkbox checked for "Enable Parameterization"), and "METASTORE CATALOG" (checkbox checked for "Sync tables tips").

The central query editor pane contains the following SQL code:

```
1  SELECT HOUR(call_begin) AS hour,
2      MINUTE(call_begin) AS minute,
3      COUNT(call_begin) AS failures
4  FROM calldetail
5  WHERE status = 'FAILED'
6  GROUP BY hour, minute;
7  ORDER BY failures DESC
8  LIMIT 100;
```

At the bottom of the editor are buttons for "Execute", "Download", "Save as...", "Explain", and "or create a New query".

Accessing Impala with Hue (3)

- Hue displays the results in a sortable table

The screenshot shows the Hue web interface for Hive queries. The title bar reads "Hue - Hive Query" and the URL is "https://hueserver.example.com:8888/impala/". The top navigation bar includes icons for Home, Recent, Saved, History, and Settings, along with a user dropdown for "training". Below the navigation is a tab bar with "Query Editor" (selected), "My Queries", "Saved Queries", "History", and "Settings". The main content area is titled "Query Results: Unsaved Query". A table displays 15 rows of data with columns "hour", "minute", and "failures". The table is sortable by clicking on the column headers.

	hour	minute	failures
0	15	30	59
1	15	24	55
2	10	54	55
3	23	6	54
4	19	17	54
5	22	43	54
6	13	56	53
7	22	2	52
8	21	37	52
9	11	17	52
10	16	9	52
11	17	15	52
12	15	14	52
13	16	25	51
14	12	58	51

Overview of Impala Query Syntax

- **Impala's query language is a subset of SQL-92**
 - Plus a few extensions from the Oracle and MySQL dialects
- **Syntax is virtually identical to HiveQL**
 - Differences mainly related to features unsupported in Impala
 - Most Hive queries can be executed verbatim in Impala
- **Impala may also support statements that Hive does not**
 - Such as the ability to insert individual rows *

```
> INSERT INTO departments VALUES (9, 'Engineering');
```

* Not intended for bulk loads

Case-Sensitivity and Comments

- **Keywords are not case-sensitive, but are often capitalized by convention**
- **Supports single- and multi-line comments**
 - These are allowed in scripts, the command line shell, and Hue

```
$ cat find_wealthy_accounts.sql

/*
    This script will query the accounts table
    and find all accounts in a given ZIP code
*/
SELECT acct_num, first_name, last_name
    FROM accounts
   WHERE zipcode='90210';      -- Beverly Hills
```

Limiting and Sorting Query Results

- The **LIMIT** clause sets the maximum number of rows returned

```
> SELECT acct_num, city FROM accounts LIMIT 10;
```

- As with Hive, no guarantee regarding which 10 results are returned
 - Use ORDER BY for top-N queries
 - The field(s) you ORDER BY must be selected
- When using ORDER BY, the LIMIT clause is mandatory in Impala

```
> SELECT acct_num, city FROM accounts  
      ORDER BY acct_num DESC LIMIT 10;
```

Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- Comparing Impala with Hive
- Running Queries in Impala
- **Support for User-Defined Functions**
- Data and Metadata Management
- Essential Points
- Optional Hands-On Exercise: Running Queries in Impala

Built-In Functions in Impala

- **Impala supports many of the same built-in functions as Hive**
 - Lacks some others, particularly for formatting and text processing
 - Support for the SHOW FUNCTIONS command varies by release
 - See the Impala documentation for a complete list of functions
- **These functions are invoked just as they are in HiveQL and SQL**

```
> SELECT CONCAT_WS(' ', ' ', last_name, first_name) AS full_name  
      FROM accounts WHERE acct_num=871573;  
  
+-----+  
| full_name |  
+-----+  
| Klopeck, Reuben |  
+-----+
```

Custom User-Defined Functions (UDFs) in Impala

- **Impala UDFs can be written in Java or C++**
 - UDFs were introduced in Impala 1.2
- **Java-based UDFs use the same API as in Hive**
 - Many Hive UDFs can be reused in Impala without change
- **UDFs written in C++ offer significantly better performance**
 - Recommended approach for most new functions
 - UDFs implemented in C++ cannot be used in Hive
 - Out of scope for this class, since C++ is not a prerequisite

Limitations of Hive UDF Support in Impala

- **Aggregate (UDAF) and table-generating (UDTF) functions are unsupported**
 - Impala does support UDAFs implemented in C++
- **Certain types of data are unsupported as parameters or return values**
 - Nested types like MAP or ARRAY
 - TIMESTAMP and BINARY
 - Objects that are not subclasses of Writable

Using a Java UDF in Impala

- Perform these steps to use a Java-based UDF in Impala
 - Repeat them – and restart Impala – whenever code changes
- Package the compiled code into a JAR file and upload to HDFS

```
$ hadoop fs -put priceformatter.jar /loudacre/udfs
```

- Register the UDF in Impala

```
> CREATE FUNCTION FORMAT_CENTS(INT) RETURNS STRING  
    LOCATION '/loudacre/udfs/priceformatter.jar'  
    SYMBOL='com.loudacre.example.FormatCents';
```

- Use the function in your query

```
> SELECT FORMAT_CENTS(price) AS formatted FROM sales;
```

Using a C++ UDF in Impala

- The steps are almost identical when using a UDF written in C++
 - Compile the code into a shared object
 - Upload that shared object file to a directory in HDFS

```
$ hadoop fs -put sampleudfs.so /loudacre/udfs
```

- Register the UDF in Impala

```
> CREATE FUNCTION COUNT_VOWELS(STRING) RETURNS INT  
  LOCATION '/loudacre/udfs/sampleudfs.so'  
  SYMBOL='CountVowels';
```

- Use the function in your query

```
> SELECT COUNT_VOWELS(city) FROM accounts;
```

Unregistering a Function

- **Use `DROP FUNCTION` to unregister the UDF**

- Must do this before you can register another with same signature
 - Invocation is the same for Java and C++ functions

```
> DROP FUNCTION FORMAT_CENTS(INT);
```

```
> DROP FUNCTION COUNT_VOWELS(STRING);
```

Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- Comparing Impala with Hive
- Running Queries in Impala
- Support for User-Defined Functions
- **Data and Metadata Management**
- Essential Points
- Optional Hands-On Exercise: Running Queries in Impala

How Impala Executes a Query

- **Each slave node in the cluster runs an Impala daemon**
 - Co-located with the HDFS DataNode
- **Impala daemon plans the query issued by the client**
 - Checks the local metadata cache
 - Distributes the query across other Impala daemons in the cluster
 - Daemons read data from HDFS and stream results to the client
- **Two other daemons running on master nodes support query execution**
 - State Store daemon
 - Provides lookup service and status checks for Impala daemons
 - Catalog daemon
 - Relays metadata changes to all the Impala daemons in a cluster

Metadata Caching in Impala

- **Impala shares the metastore with Hive**
 - Tables created in Hive are visible in Impala (and vice versa)
- **Impala caches metadata**
 - The tables' schema definitions
 - The locations of tables' HDFS blocks
- **Metadata updates made *from Impala* are broadcast throughout cluster**
 - Metadata cache is updated automatically on all nodes running Impala

Updating the Impala Metadata Cache

- **Impala is unaware of metadata updates made *outside of Impala***
 - For example, changes made in Hive, or changes to table data in HDFS
 - You must manually update the metadata cache following such changes
- **Procedure for updating metadata cache varies by Impala version**
 - Check the documentation for details specific to your version
- **In the version of Impala on the VM, `INVALIDATE METADATA` reloads *all* metadata**

```
> INVALIDATE METADATA;
```

- If a table name is specified, only that table's metadata will be reloaded

```
> INVALIDATE METADATA products;
```

Creating Tables in Impala

- Data definition language is generally identical to Hive
 - Supports most of the same data types as Hive
 - Tables created in Impala are immediately available in Hive

```
> CREATE EXTERNAL TABLE prospects
  (id INT,
   name STRING,
   email STRING,
   last_contact TIMESTAMP)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LOCATION '/dept/sales/prospects';
```

Using Hive Tables in Impala

- **Impala does not support Hive SerDes**
 - It does support most formats commonly used in Hive
 - Tables with unsupported formats are unreadable in Impala
- **Workaround: use a CTAS *in Hive* to create a new table for Impala**
 - The logdata table in this example uses RegexSerDe
 - Impala can read the tab-delimited format of the new table

```
hive> CREATE TABLE delimited_logdata
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      AS
      SELECT * FROM logdata;
```

Data Formats Supported by Impala

- **Impala can query data in several formats**
 - Table below summarizes compatibility
- **Create and load with Hive if those operations are unsupported in Impala**
 - Impala does not support enum, bytes, or complex types in Avro

Format	Description of Format	Read	Create	Insert
TEXTFILE	Delimited text file format	Yes	Yes	Yes
SEQUENCEFILE	Binary flat file format	Yes	Yes	No
Avro	Structured cross-platform binary format	Yes	No	No
RCFILE	Columnar format compatible with Hive	Yes	Yes	No
PARQUET	High-performance columnar format	Yes	Yes	Yes

Creating a Parquet Table in Impala

- Ideal format when only a subset of columns are typically accessed

```
> CREATE TABLE survey_results
  (contact_id INT,
   interviewer_id INT,
   when_interviewed TIMESTAMP,
   annual_income INT,
   occupation STRING,
   postal_code STRING,
   signal_strength_rating TINYINT,
   product_quality_rating TINYINT,
   support_staff_rating TINYINT,
   overall_value_rating TINYINT,
   would_recommend BOOLEAN)
STORED AS PARQUET;
```

Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- Comparing Impala with Hive
- Running Queries in Impala
- Support for User-Defined Functions
- Data and Metadata Management
- **Essential Points**
- Optional Hands-On Exercise: Running Queries in Impala

Bibliography

The following offer more information on topics discussed in this chapter

- **Impala Concepts and Architecture**

- <http://tiny.cloudera.com/adcc12a>

- **Impala FAQ**

- <http://tiny.cloudera.com/adcc12b>

- ***Cloudera Impala (free e-book published by O'Reilly)***

- <http://tiny.cloudera.com/adcc12c>

- **Impala UDF Examples in Java and C++**

- <http://tiny.cloudera.com/adcc12d>

- **Impala-related Articles on Cloudera's Blog**

- <http://tiny.cloudera.com/adcc12e>

Essential Points

- **Impala is a high-performance SQL engine for Hadoop**
 - Reads data in HDFS and shares metadata with Hive
 - Queries are expressed in an SQL dialect similar to HiveQL
 - Custom execution engine avoids the latency of MapReduce
- **Impala is best suited to ad hoc/interactive queries**
 - Hive is better for long-running batch processes
 - Not all Hive features are currently supported in Impala
- **External changes to table data requires a metadata cache update**
- **Impala supports User-Defined Functions in Java and C++**
 - Java provides compatibility with many Hive UDFs
 - We recommend writing functions in C++ for best performance

Chapter Topics

Executing Interactive Queries with Impala

- What is Impala?
- Comparing Impala with Hive
- Running Queries in Impala
- Support for User-Defined Functions
- Data and Metadata Management
- Essential Points
- **Optional Hands-On Exercise: Running Queries in Impala**

Optional Hands-On Exercise: Running Queries in Impala

- In this optional Hands-On Exercise, you will use Hue and the Impala shell to query data in the Hive tables you created during an earlier exercise
 - Please refer to the Hands-On Exercise Manual for instructions

Understanding Cloudera Search

Chapter 13



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- **Understanding Cloudera Search**
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Understanding Cloudera Search

In this chapter you will learn

- The fundamentals of Cloudera Search
- Which open source projects make up Cloudera Search
- Two ways Cloudera Search can index data
- How indexes are split into shards and replicated throughout the cluster
- Some of the document formats supported by Search

Chapter Topics

Understanding Cloudera Search

- **What is Cloudera Search?**
- Search Architecture
- Supported Document Formats
- Essential Points

What is Cloudera Search?

- **Cloudera Search provides full-text interactive search for data stored in Hadoop**
 - Integrates CDH with Apache Solr and other open source tools
- **Apache Solr provides high-performance indexing and search**
 - Mature platform with widespread deployment
 - Standard Solr APIs and Web UI are available in Cloudera Search
- **CDH offers scalability and reliability**
 - Distributed data storage and indexing
- **Cloudera Search is 100% open source**
 - Released under the Apache Software License



Benefits of Cloudera Search

- **Provides another way to access data stored in Hadoop**
 - More convenient and accessible for non-technical audiences
- **Particularly appropriate for semi-structured and free-form data**
 - Supports many file formats, with content in 30+ languages
- **Fast, scalable, and reliable**
 - Data and indexes can span the entire cluster
 - Not limited by the capacity or capabilities of a single machine
 - Node failure is handled gracefully, as with MapReduce
 - Can expand the cluster as additional capacity is needed

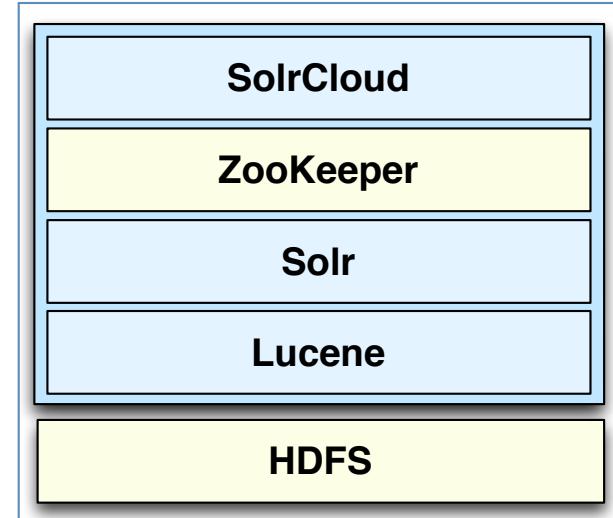
Chapter Topics

Understanding Cloudera Search

- What is Cloudera Search?
- **Search Architecture**
- Supported Document Formats
- Essential Points

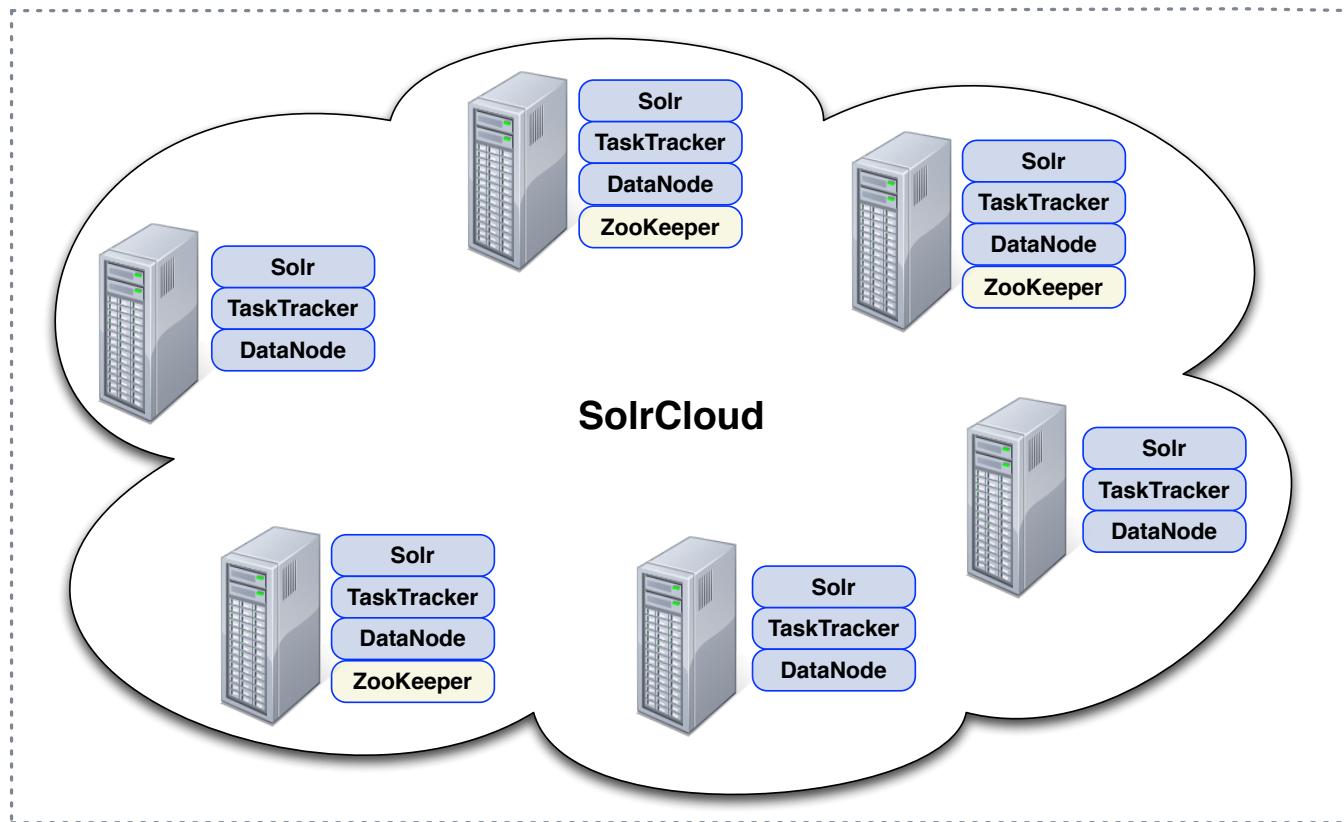
High-Level Architecture of Cloudera Search

- **HDFS is Hadoop's high-performance, reliable, distributed filesystem**
 - Used to store and serve data in Search
 - Also holds the *indexes* for data available through Search
- **Lucene is a low-level information retrieval library**
- **Solr is a standalone search platform based on Lucene**
- **SolrCloud is the distributed version of Solr**
 - Uses HDFS for storage
 - Provides scalability and fault-tolerance
 - Coordination among nodes via ZooKeeper



Daemons in a Cluster Running Cloudera Search

- Each cluster node typically runs Solr, TaskTracker, and DataNode daemons
 - There is no ‘master’ daemon for Search



Overview of Data Indexing in Cloudera Search

- Data must be indexed before it can be searched
 - Indexing requires the use of a schema
 - Schema specifies the field names and types
- A complete logical index is called a *collection*

Data to be indexed

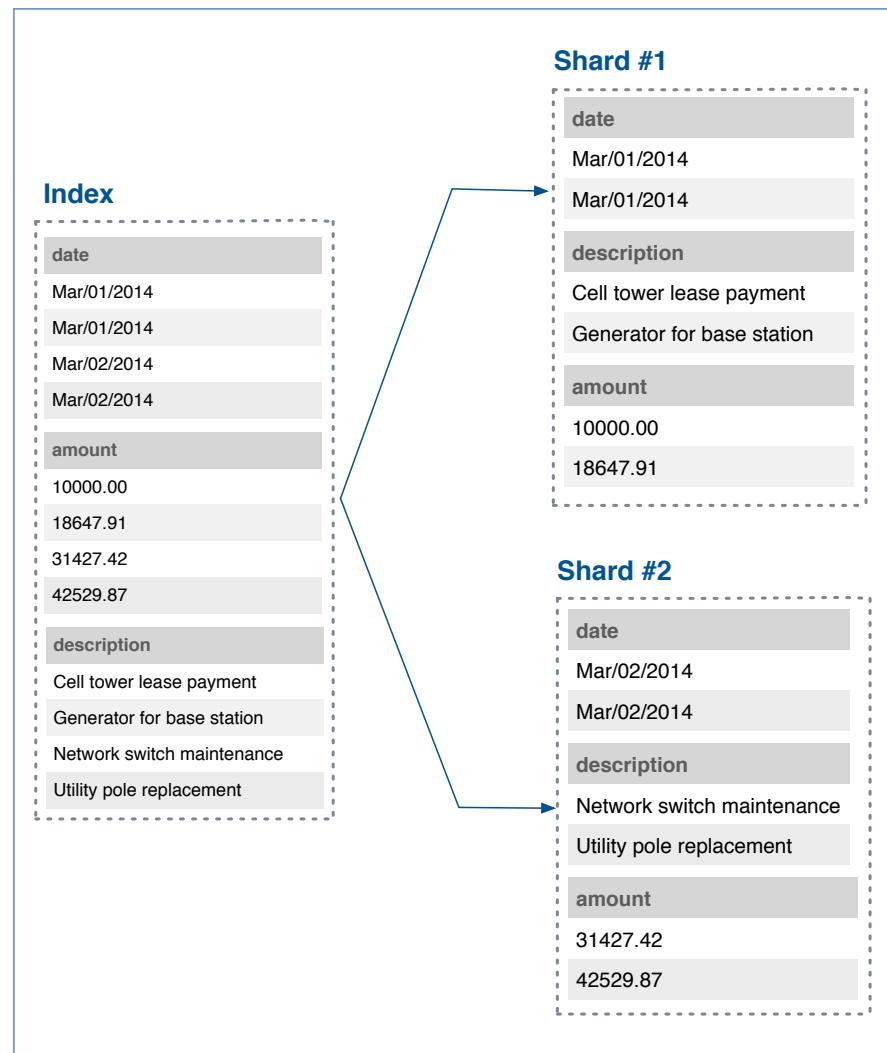
```
Mar/01/2014
    Cell tower lease payment: 10000.00
Mar/01/2014
    Generator for base station: 18647.91
Mar/02/2014
    Network switch maintenance: 31427.42
Mar/02/2014
    Utility pole replacement: 42529.87
```

Schema

Index	
date	Mar/01/2014
date	Mar/01/2014
date	Mar/02/2014
date	Mar/02/2014
amount	10000.00
amount	18647.91
amount	31427.42
amount	42529.87
description	Cell tower lease payment
description	Generator for base station
description	Network switch maintenance
description	Utility pole replacement

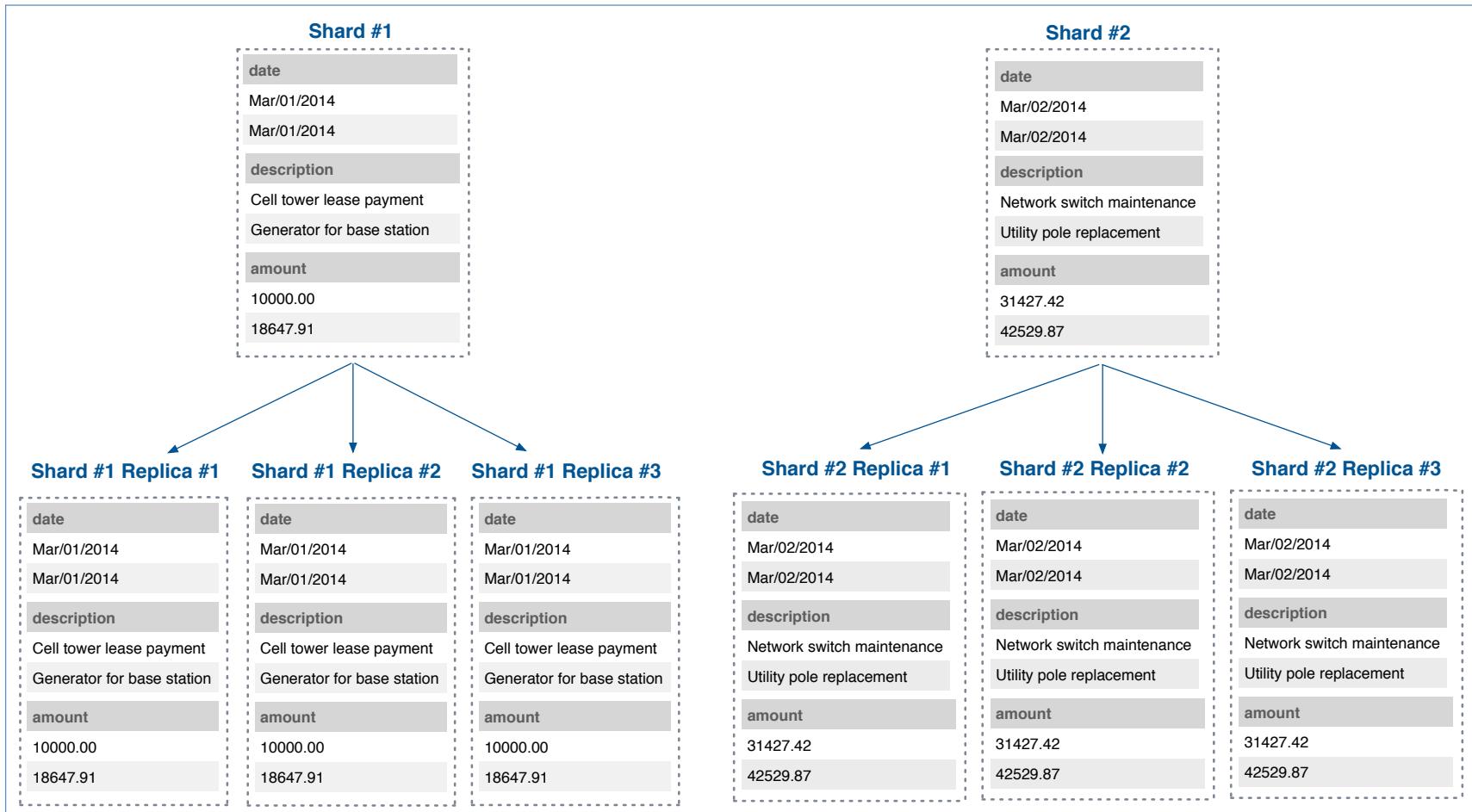
Index Sharding

- Indexes are split into *shards*
- Improves scalability
 - Shards are distributed
 - Not limited by memory or disk of a single machine
- Improves performance
 - Nodes can read in parallel
 - Results are automatically collated



Replication of Shards

- Shards may be replicated and copied throughout the cluster

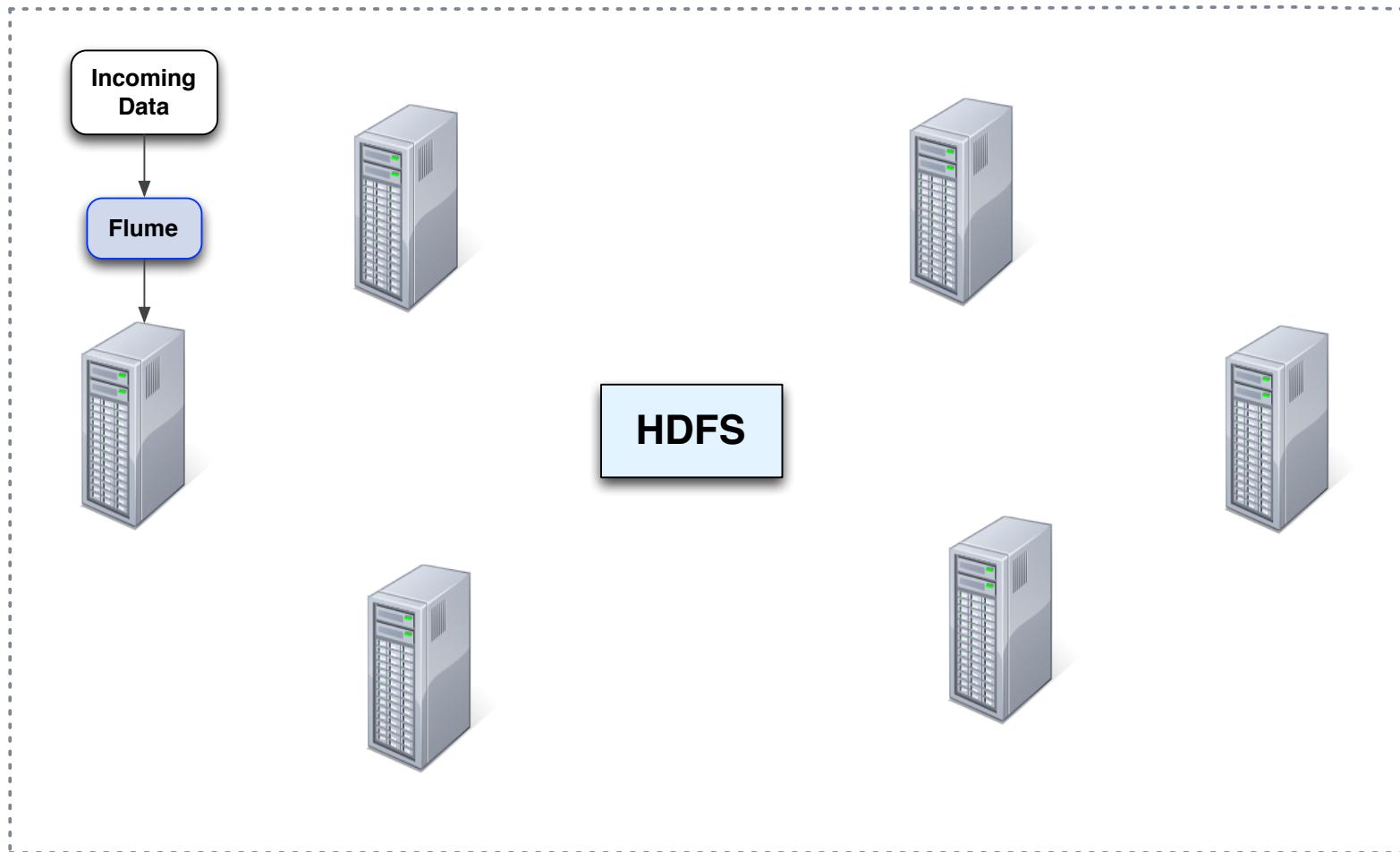


Use Case Overview

- **This course covers two ways of indexing data with Cloudera Search**
- **Near-Real-Time indexing with Flume**
 - Data is indexed immediately as it enters the cluster
- **Batch mode indexing with MapReduce**
 - Used to index data that already resides in HDFS
- **Users can interact with Search in several ways**
 - Submitting a Web form with a search term is typical

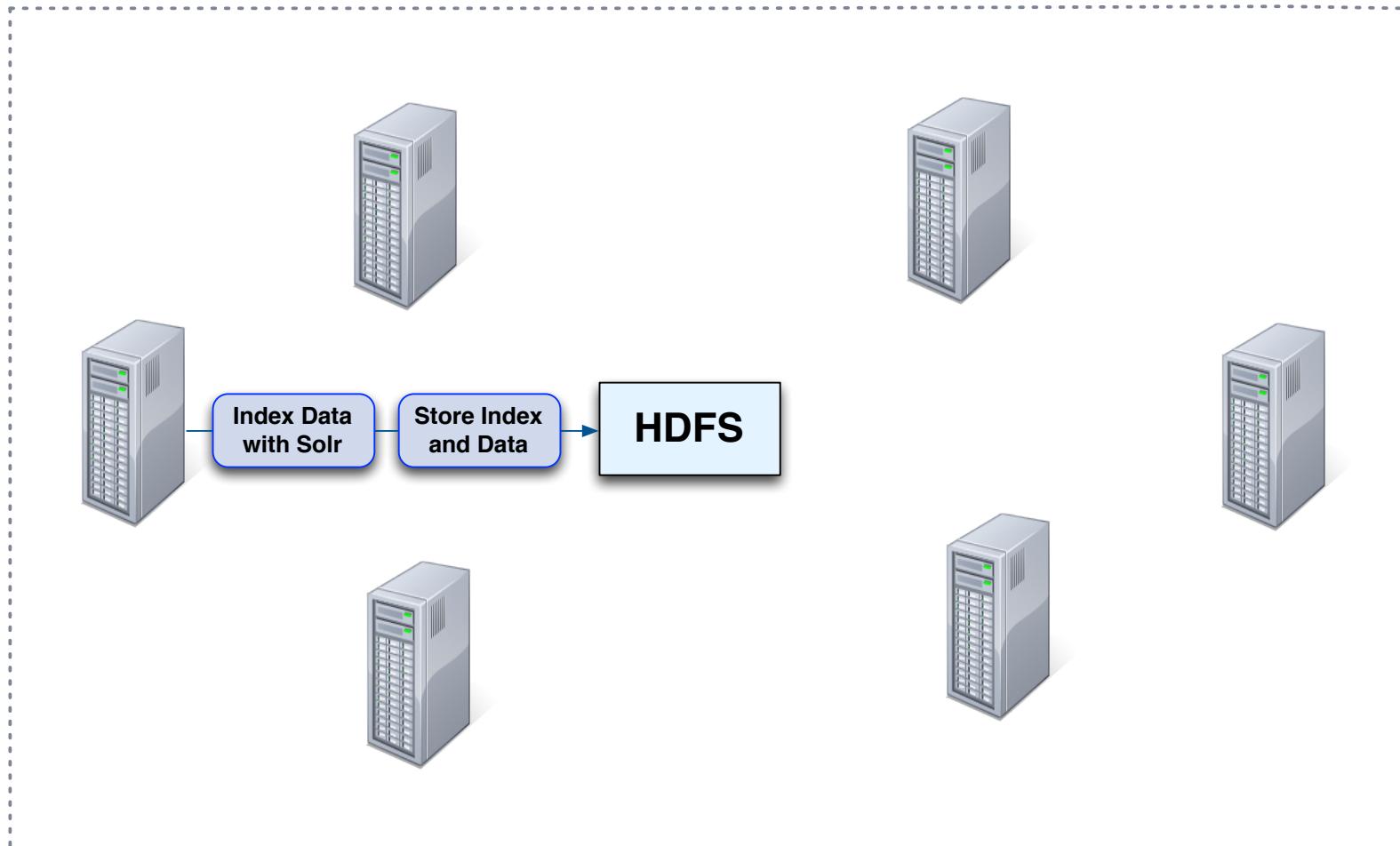
Overview of Near-Real-Time Search Use Case (1)

- Incoming data is ingested via a Flume agent



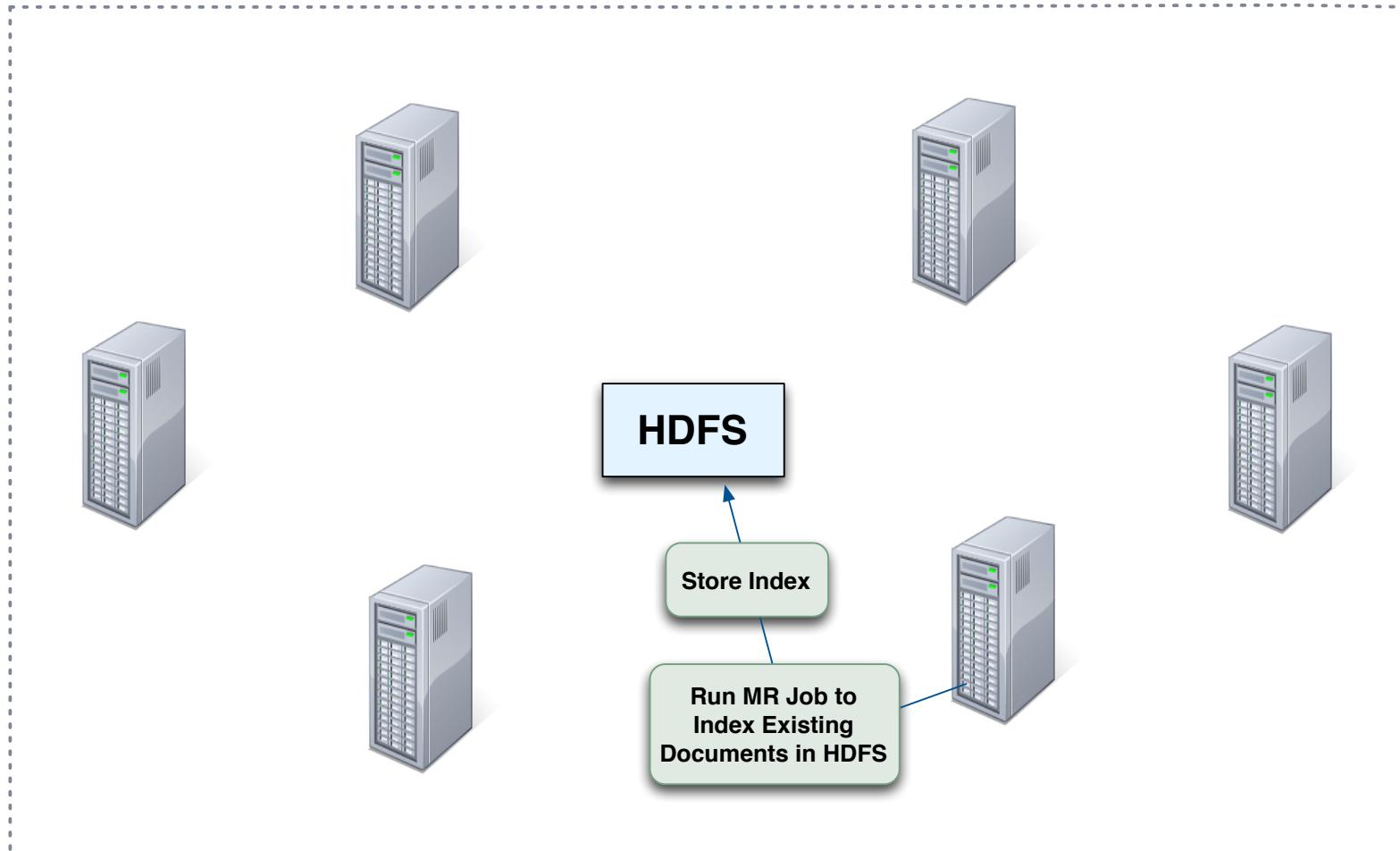
Overview of Near-Real-Time Search Use Case (2)

- Flume can index data with Solr and store it in HDFS as it is ingested



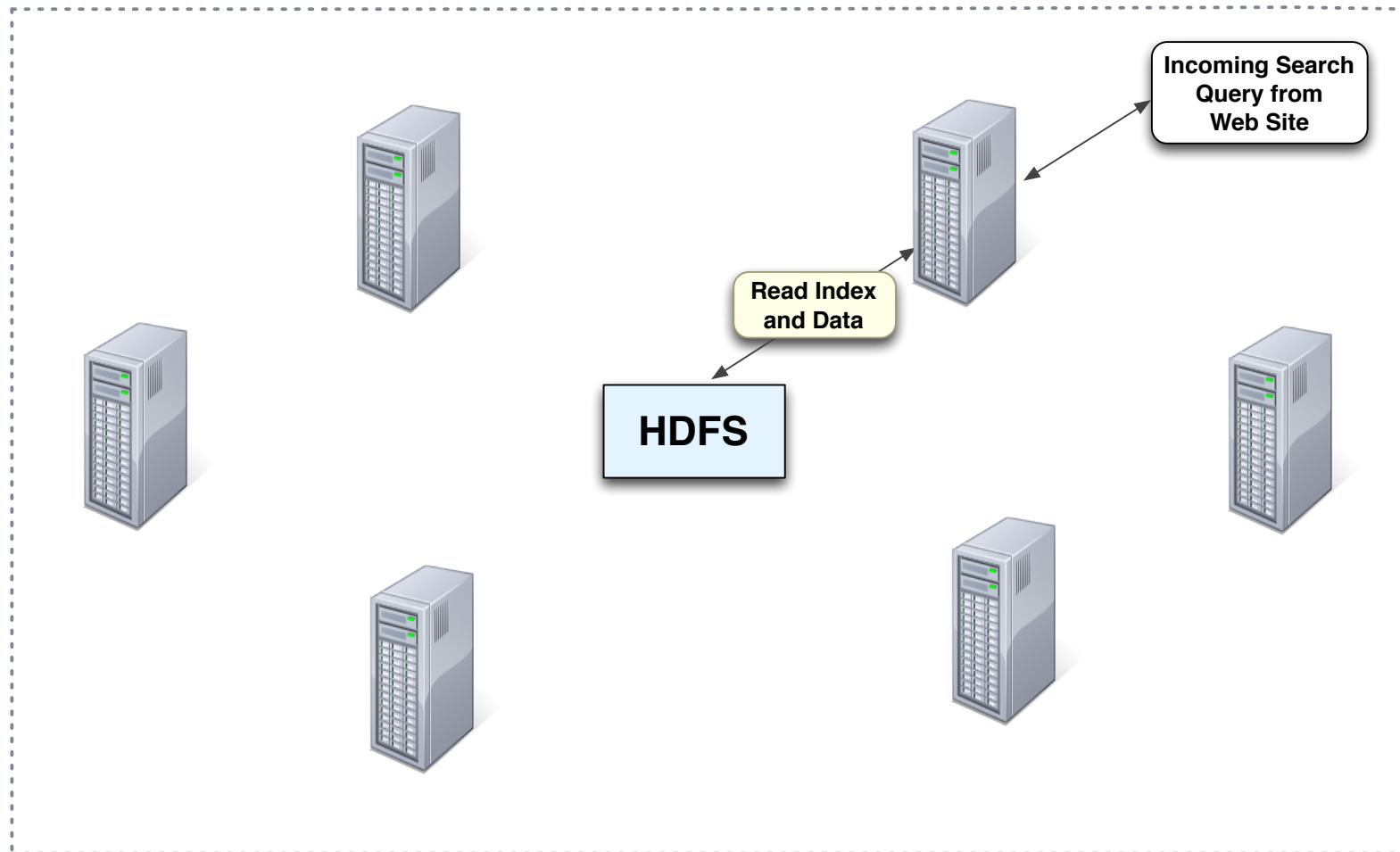
Overview of Batch Indexing Use Case

- Existing data in HDFS can also be indexed using a MapReduce job



Serving a Search Request

- Since index and data are in HDFS, any node can fulfill incoming requests



Should You Index in Batch or Near-Real-Time Mode?

- Whether you should index in batch or near-real-time mode depends on many factors
- Batch indexing is the better choice if
 - The data being indexed already exists in HDFS
 - The data has not already been indexed with Search
- Near-Real-Time (NRT) is a better choice if
 - Data must be available in search results immediately
 - The cluster has a long queue of pending MapReduce jobs
- You must use batch indexing if using Flume is not possible
 - For example, if data is ingested using Sqoop

Chapter Topics

Understanding Cloudera Search

- What is Cloudera Search?
- Search Architecture
- **Supported Document Formats**
- Essential Points

Document Support in Cloudera Search

- **Cloudera Search can index and search many document formats**
 - Relies on an open source project called Apache Tika
 - Tika extracts metadata and content from common file types
- **Supports many common document formats, such as**
 - Plain text
 - HTML and XML
 - Rich text (RTF)
 - Microsoft Office and Open Office
 - PDF and epub
- **Cloudera Search also adds support for common Hadoop file types**
 - Including SequenceFiles and Avro



Chapter Topics

Understanding Cloudera Search

- What is Cloudera Search?
- Search Architecture
- Supported Document Formats
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Doug Cutting's Blog Post Announcing Cloudera Search**
 - <http://tiny.cloudera.com/adcc13a>

Essential Points

- **Cloudera Search provides full-text interactive searching for data in Hadoop**
 - Based on Apache Solr, Apache Lucene, Apache Tika, and other tools
 - Solr is a mature high-performance open source search platform
 - CDH components provide reliability and scalability
- **Search offers an additional option for accessing data**
 - Ideal for free-form or semi-structured data in many formats
 - Does not require users to have experience with Java or SQL
 - Supports batch and near-real-time (NRT) indexing
- **A Search index is known as a collection**
 - Often split into shards and replicated across the cluster
 - Data must be indexed according to a schema before it can be searched

Indexing Data with Cloudera Search

Chapter 14



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- **Indexing Data with Cloudera Search**
- Presenting Results to Users
- Conclusion

Indexing Data with Cloudera Search

In this chapter you will learn

- **How to define the schema for a search index**
- **How Morphlines is used for ETL in Search workflows**
- **How to index data in batch and near-real-time modes**
- **How to verify successful indexing using the Solr Admin UI**

Chapter Topics

Indexing Data with Cloudera Search

- **Collection and Schema Management**
- Morphlines
- Indexing Data in Batch Mode
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- Indexing Data in Near Real Time
- Essential Points
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Collection and Schema Management Overview

- You must index data before accessing it in Search
- Before indexing the data, you will complete four prerequisite steps
 1. Generate configuration files from a template
 2. Edit the configuration files to match your data
 3. Upload the configuration files to ZooKeeper
 4. Create a collection to hold the index
- The following slides describe the steps listed above
 - These are required regardless of which method used to index data
- The example creates a collection named `employees_collection`
 - It assumes a ZooKeeper server at port 2181 on `dev.loudacre.com`
 - A similar example can be found the `demo` directory on your VM

Generating the Configuration Files

- A set of configuration files are associated with each collection
 - We can generate these from the default template
 - The directory name need not match the collection name

```
$ solrctl --generate /home/training/phones
```

- Creates a **phones** directory with a **conf** subdirectory
 - The only file we must edit is `schema.xml`
 - Schema must specify the names and types of fields to index

```
$ vi /home/training/employees/conf/schema.xml
```

Defining the Schema

- The generated schema file is heavily commented
- Root element contains two attributes

```
<schema name="employees" version="1.5">
```

- Name is used only for display purposes
 - Version defines the allowed syntax and should *not* be changed
- Two element types are typically modified in the schema

Attribute	Description
<field>	Defines a field that will be indexed
<uniqueKey>	Specifies which field uniquely identifies a record

The <field> Element: Name and Type Attributes

- As with a database schema, each field must have a name and a type
 - Field names generally have same naming restrictions as Java variables
 - The table below lists several commonly-used types

Type	Description
string	Character data that is indexed and stored verbatim
text_general	Character data that can be tokenized, lowercased, stopwords applied, etc.
boolean	Values that are true or false
int	32-bit signed integer
long	64-bit signed integer
float	32-bit floating point number
double	64-bit floating point number
date	Timestamp in ISO 8601 format (e.g., 2014-03-27T19:37:04Z)

The <field> Element: Additional Attributes

- The **field** element supports many additional attributes, including

Type	Description
indexed	True if this field should be indexed. Only indexed fields can be searched
stored	True if it should be possible to include this field in search results
default	Specifies the value used when the input document does not specify a value
required	If true, Solr will throw an error when encountering missing value
multivalued	True if this field can appear more than once per document

The <field> Element: Examples

- Here are several example field definitions
- Every schema must define a `_version_` field as shown here
 - This field is populated automatically

```
<fields>
  <field name="id" type="string"
        indexed="true" stored="true"/>
  <field name="firstname" type="string"
        indexed="true" stored="true"/>
  <field name="lastname" type="string"
        indexed="true" stored="true"/>
  <field name="salary" type="int"
        indexed="true" stored="true"/>
  <field name="skills" type="text_general"
        indexed="true" stored="true" multiValued="true"/>
  <field name="_version_" type="long"
        indexed="true" stored="true"/>
</fields>
```

Caution: Ensure Your Fields Are the Only Fields Defined

- Generated schema.xml is a template with many example fields
- You need to delete all other fields before adding your own
 - Also delete all <dynamicField> and <copyField> elements

```
<fields>
    <!-- define your fields here, and delete all others -->
    <field name="id" type="string" ... />

    <del</del><field name="name" type="text_general" ... />

    <del</del><dynamicField name="*_i" type="int" ... />
    <del</del><dynamicField name="*_s" type="int" ... />
</fields>

<del</del><copyField source="cat" dest="text"/>
<del</del><copyField source="name" dest="text"/>
```

The <uniqueKey> Element

- **This element specifies which field uniquely identifies the document**
 - We recommend that every schema define a unique key named `id`
 - This field should always be defined as type `string`

```
<uniqueKey>id</uniqueKey>
```

- **You can generate an ID if your documents do not have one**
 - Use Morphlines' `generateUUID` command during the indexing process

Creating the Instance Directory

- **The configuration files were generated locally**
- **SolrCloud uses ZooKeeper to store configuration data**
 - We must make our configuration available in ZooKeeper
 - The upload step is called “creating the *instancedir*”
- **The instancedir is created with `solrctl instancedir --create`**
 - First argument is a reference to the ZooKeeper path
 - Second argument is the name you will use for your collection
 - Third argument is the path to the *local* configuration directory

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \
--create employees_collection \
/home/training/employees
```

Creating the Collection

- Once the `instancedir` is uploaded, you may create the collection
 - The collection name must match that used in the previous command
- The `-s` option specifies the number of shards
 - If omitted, creates a collection with one shard
 - Determining the best number of shards is the result of experimentation
 - Influenced by collection size, query complexity, and expected load
 - Should generally not exceed the number of nodes in the cluster
 - We recommend you experiment with realistic data and queries
 - Can later split shards for additional scalability, if needed

```
$ solrctl --zk dev.loudacre.com:2181/solr collection \
--create employees_collection -s 5
```

- We may now index data and populate the collection

Deleting the Collection and Instance Directory

- If you encounter an error, it is often easier to just start again
 - You may also wish to start over so you can experiment with options
- You should delete items in the reverse order you created them
 - First, remove the collection

```
$ solrctl --zk dev.loudacre.com:2181/solr collection \
--delete employees_collection
```

- Afterwards, remove the instance directory

```
$ solrctl --zk dev.loudacre.com:2181/solr instancedir \
--delete employees_collection
```

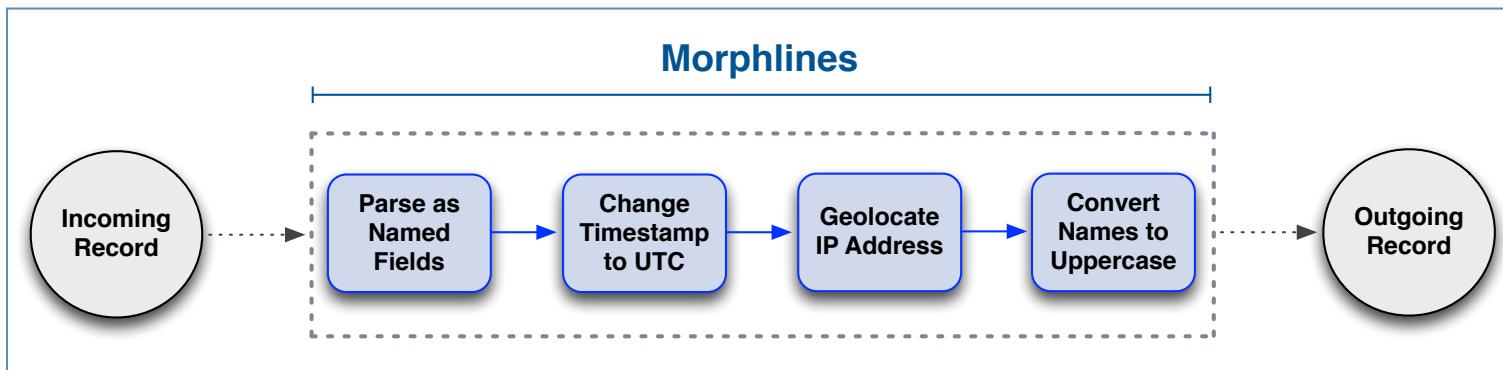
Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- **Morphlines**
- Indexing Data in Batch Mode
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- Indexing Data in Near Real Time
- Essential Points
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Morphlines

- **Morphlines is a framework for processing streams of data**
 - Originally created for Cloudera Search and later contributed to the Kite SDK
 - Many helpful features for indexing data with Search
 - A plain Java library that can be used even outside of Hadoop
- **Especially useful for Extract, Transform, and Load (ETL) processing**
 - Requires no experience using MapReduce
 - Transformations are defined declaratively in a configuration file
 - Morphlines ships with dozens of reusable commands



Morphlines Commands for Reading Data

- Incoming data arrives as a byte array, much like an e-mail attachment
 - In NRT mode, the data is the Flume event body
 - In batch mode, the data is the document in HDFS
- Before processing this data, we read it into a Morphline record
 - Table lists commonly-used Morphline commands for reading data

Command	Description
readClob	Read entire byte array into a string field named message
readLine	Read each individual line into a string field named message
readCSV	Read each line and parse into named fields using delimiter
readJson	Parse as JSON and emit a record for each contained JSON object

Morphlines Commands for Processing Data

- Once data is read, it can be processed before being indexed
- These are just a few of Morphlines' many processing commands:

Command	Description
extractJsonPaths	Extracts specific values from a JSON object
xquery	Extracts specified values from XML using an XPath expression
grok	Uses regex to extract additional fields from text content
dropRecord	Discards this record
convertTimestamp	Converts timestamp format and/or timezone
setValues	Assign values to specified fields
generateUUID	Sets value of a named field to a universally unique identifier

Example: Working With JSON Input Data in Morphlines

- Example of extracting relevant content from JSON into named fields

```
{  
    readJson {}  
,  
    {  
        extractJsonPaths {  
            flatten: false  
            paths : {  
                id: "/transactionId"  
                account_num: "/accountNum"  
                time_of_sale: "/timestamp"  
                employee_id: "/salesperson"  
                product_id: "/messages[]/product"  
                item_desc: "/messages[]/description"  
                price: "/messages[]/price"  
            }  
        }  
    },  
}
```

Using Morphlines in Cloudera Search

- Morphlines also provides commands helpful for Search indexing

Command	Description
sanitizeUnknownSolrFields	Removes any field not specified in the schema.xml. Solr will throw an exception if it encounters a field that was not defined in the schema. Omit this command to “fail fast” during your indexing operation
loadSolr	Parses data using Apache Tika and loads the result into Solr

- Caution: Morphlines in CDH 4.x Search is based on a pre-Kite version
 - New features are not available
 - Package names are com.cloudera.cdk **not** org.kitesdk
 - Be careful when looking at documentation and examples!

Morphlines Configuration (1)

- Morphlines is used in both NRT and batch indexing operations
- Commands are declared in the Morphlines configuration file
 - Executed in sequence, like a UNIX pipeline
 - File name is arbitrary, but usually given a .conf extension
 - Morphlines uses HOCON syntax (variant of JSON)
- All Morphlines files used with Solr will have the following section at the top
 - Global definition of properties used elsewhere in the file
 - The collection attribute *must* match the name of your collection

```
SOLR_LOCATOR : {  
    collection : my_simple_collection  
    zkHost : "dev.loudacre.com:2181/solr"  
}
```

Note: file continues on next slide

Morphlines Configuration (2)

- This example will index everything as a single field named **contents**
 - importCommands finds commands in classpath for packages listed

```
morphlines : [
{
  id : morphline1
  importCommands : ["com.cloudera.**", "org.apache.solr.**"]
  commands : [
    {
      readClob { charset : UTF-8 }
    },
    {
      setValues { contents : "@{message}" }
    },
    {
      generateUUID { field:id }
    },
  ]
}
```

Note: file continues on next slide

Morphlines Configuration (3)

- Finally, we strip any unknown fields and load the data into Solr
 - The logTrace statement aids in debugging

```
{ logTrace { format : "output: {}", args : ["@{}"] } }

{
    sanitizeUnknownSolrFields {
        solrLocator : ${SOLR_LOCATOR}
    }
},
{
    loadSolr {
        solrLocator : ${SOLR_LOCATOR}
    }
}
]
```

Aside: Timestamp Conversion

- Solr mandates a specific timestamp format (ISO 8601) and timezone (UTC)
 - Your documents may use a different format
- The `convertTimestamp` command can perform the conversion
 - Input and output format use Java's `SimpleDateFormat` syntax

```
{  
    convertTimestamp {  
        field : timestamp  
        inputFormats : ["yyyy-MM-dd HH:mm:ss"]  
        inputTimezone : America/Los_Angeles  
        outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"  
        outputTimezone : UTC  
    }  
},
```

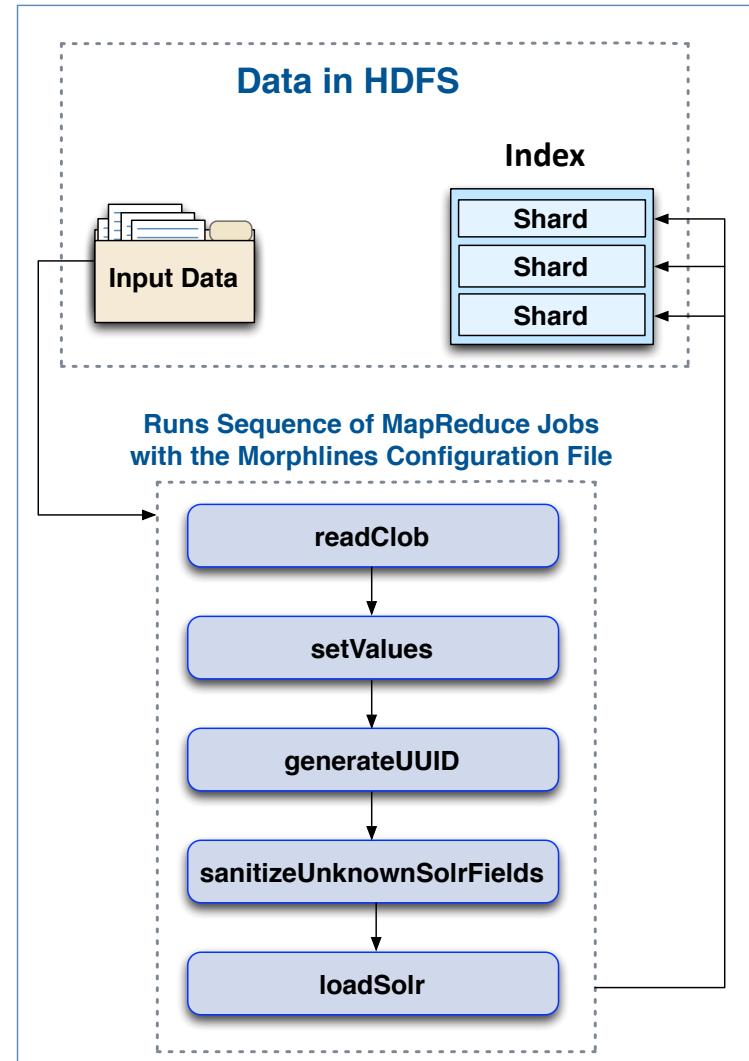
Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- Morphlines
- **Indexing Data in Batch Mode**
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- Indexing Data in Near Real Time
- Essential Points
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Overview of Batch Indexing

- Batch indexing is performed by running **MapReduceIndexerTool**
 - Runs a sequence of MapReduce jobs
 - Uses the specified Morphlines file
 - Input data must already exist in HDFS
- The “Go Live” option enables the index shards to be merged into live Solr servers
 - No downtime required



Running the MapReduceIndexerTool

- The collection must exist in Solr before running this command
- The output directory is specified using **--output-dir** option
 - Will be created if it does not exist
 - Contents will be deleted if it does exist

```
$ hadoop jar /usr/lib/solr/contrib/mr/search-mr-*-job.jar \
  org.apache.solr.hadoop.MapReduceIndexerTool \
  --zk-host dev.loudacre.com:2181/solr \
  --collection employees_collection \
  --morphline-file morphlines.conf \
  --go-live \
  --output-dir hdfs://dev.loudacre.com:8020/user/training/emp_index \
  hdfs://dev.loudacre.com:8020/user/training/employees_data
```

Debugging Failed Jobs (1)

- If a job fails, try the **--dry-run** option when re-running the job
 - Runs in local mode (faster turnaround) and does not load data into Solr
- Add **--verbose** to enable additional output
- Check the logs, typically located under **/var/log/solr**

Debugging Failed Jobs (2)

- **Use a customized Log4J properties file for even more output**
 - An example distributed with Search has a good template
 - Copy it to the current directory

```
$ cp /usr/share/doc/search-*/*examples/solr-nrt/log4j.properties .
```

- **Edit the file to change the log level as needed**
 - Setting Morphlines' level to TRACE can be *very* helpful

```
log4j.logger.com.cloudera.cdk.morphline=TRACE
```

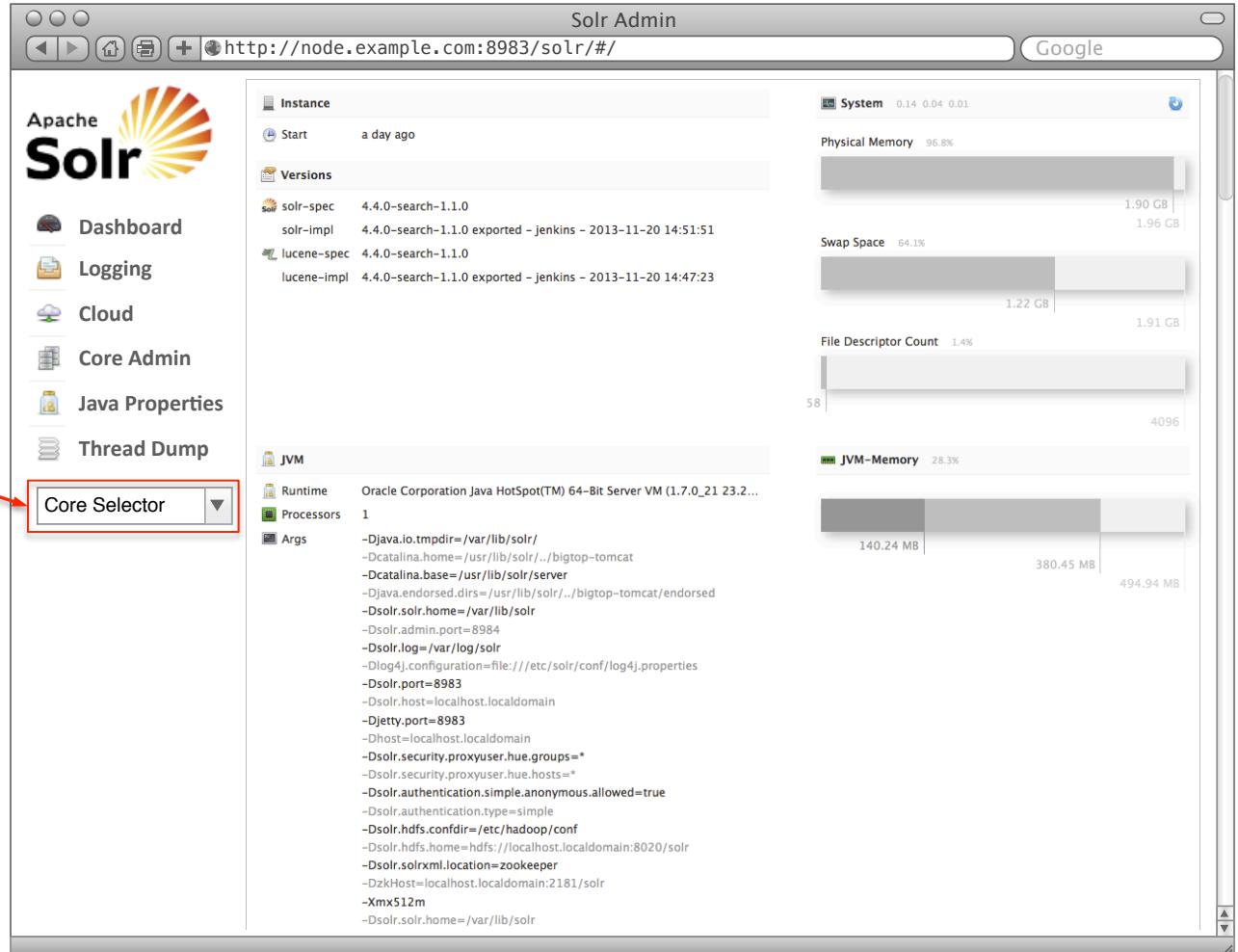
- **Specify --log4j log4j.properties option on the command line when running your batch indexing job**

Verifying Results in the Solr Admin UI (1)

- After the job completes successfully, you can issue queries
- One simple way is to use Solr's Admin UI
 - Point your Web browser to port 8983 on a host running Solr
 - Example: `http://dev.loudacre.com:8983/`
- As the name implies, the Admin UI is not meant for end users
 - It is powerful, but not intuitive
- We will discuss building custom user interfaces in the next chapter

Verifying Results in the Solr Admin UI (2)

Select your collection from this list



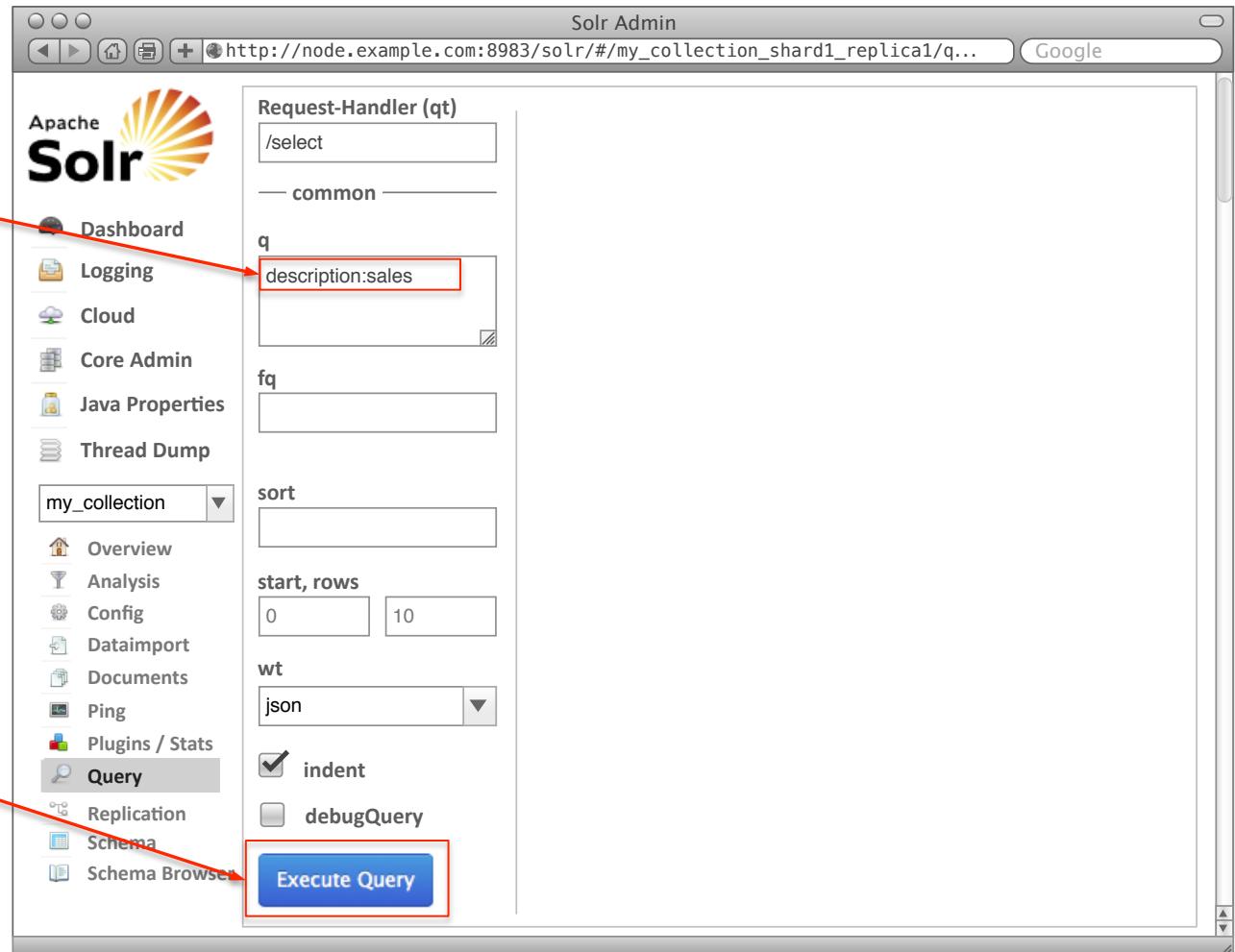
The screenshot shows the Apache Solr Admin interface at <http://node.example.com:8983/solr/#/>. On the left, there's a sidebar with links: Dashboard, Logging, Cloud, Core Admin, Java Properties, and Thread Dump. Below these is a dropdown labeled "Core Selector" with a red border around it. The main content area has several sections: "Instance" (Start: a day ago), "Versions" (listing solr-spec 4.4.0-search-1.1.0, solr-impl 4.4.0-search-1.1.0 exported - jenkins - 2013-11-20 14:51:51, lucene-spec 4.4.0-search-1.1.0, and lucene-impl 4.4.0-search-1.1.0 exported - jenkins - 2013-11-20 14:47:23), "JVM" (Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM (1.7.0_21 23.2..., Processors: 1, Args: a long list of command-line arguments including -Djava.io.tmpdir=/var/lib/solr, -Dcatalina.home=/usr/lib/solr/.., -Dcatalina.base=/usr/lib/solr/server, -Djava.endorsed.dirs=/usr/lib/solr/.., -Djava.ext.dirs=/usr/lib/solr/.., -Dsoltur.home=/var/lib/solr, -Dsoltur.admin.port=8984, -Dsoltur.log=/var/log/solr, -Dlog4j.configuration=file:///etc/solr/conf/log4j.properties, -Dsoltur.port=8983, -Dsoltur.host=localhost.localdomain, -Djetty.port=8983, -Dhost=localhost.localdomain, -Dsoltur.security.proxyuser.hue.groups=*, -Dsoltur.security.proxyuser.hue.hosts=*, -Dsoltur.authentication.simple.anonymous.allowed=true, -Dsoltur.authentication.type=simple, -Dsoltur.hdfs.confdir=/etc/hadoop/conf, -Dsoltur.hdfs.home=hdfs://localhost.localdomain:8020/solr, -Dsoltur.solrxml.location=zookeeper, -DzkHost=localhost.localdomain:2181/solr, -Xmx512m, and -Dsoltur.soltur.home=/var/lib/solr), and "System" (Physical Memory: 96.8% usage, Swap Space: 64.1% usage, File Descriptor Count: 1.4%, JVM-Memory: 28.3% usage). A note at the bottom says "Note: screenshot excerpted and reformatted to fit the screen".

Note: screenshot excerpted and reformatted to fit the screen

Verifying Results in the Solr Admin UI (3)

Optionally, type a search term in the field labeled 'q' (query)

Click the blue 'Execute Query' button



Note: screenshot excerpted and reformatted to fit the screen

Verifying Results in the Solr Admin UI (4)

Matching results
are shown in
JSON format

The screenshot shows the Apache Solr Admin interface. On the left, there's a sidebar with various navigation links like Dashboard, Logging, Cloud, Core Admin, Java Properties, Thread Dump, Overview, Analysis, Config, Dataimport, Documents, Ping, Plugins / Stats, and two sections under Query: Replication, Schema, and Schema Browser. The 'Query' section is currently selected. In the main area, there's a 'Request-Handler (qt)' form with fields for 'q' (set to 'description:sales'), 'fq', 'sort', 'start, rows' (set to 0, 10), and 'wt' (set to 'json'). Below the form are two checkboxes: 'indent' (checked) and 'debugQuery'. A large blue button at the bottom right says 'Execute Query'. To the right of the form, a red box highlights the response area. The response is a JSON object with three levels of nesting. It starts with a 'responseHeader' containing status, QTime, params (with indent set to true), and a query string. The 'response' object contains numFound (15), start (0), and docs (an array of 15 document objects). Each document object has fields like code, salary, description, total_emp, and _version_. A red arrow points from the text 'Matching results are shown in JSON format' to the 'wt' field in the request form.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 540,
    "params": {
      "indent": "true",
      "q": "sales",
      "_=": "1393389525789",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 15,
    "start": 0,
    "docs": [
      {
        "code": "11-2022",
        "salary": 106790,
        "description": "Sales managers",
        "total_emp": 322170,
        "_version_": 1461069918305779700
      },
      {
        "code": "41-3011",
        "salary": 52290,
        "description": "Advertising sales agents",
        "total_emp": 161440,
        "_version_": 1461069920111427600
      },
      {
        "code": "41-3021",
        "salary": 58580,
        "description": "Insurance sales agents",
        "total_emp": 321920,
        "_version_": 1461069920111427600
      }
    ]
  }
}
```

Note: screenshot excerpted and reformatted to fit the screen

Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- Morphlines
- Indexing Data in Batch Mode
- **Hands-On Exercise: Batch Indexing of Knowledge Base Articles**
- Indexing Data in Near Real Time
- Essential Points
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Hands-On Exercise: Batch Indexing of Knowledge Base Articles

- In this Hands-On Exercise, you will index Knowledge Base articles using the batch mode technique you just learned
 - Please refer to the Hands-On Exercise Manual for instructions

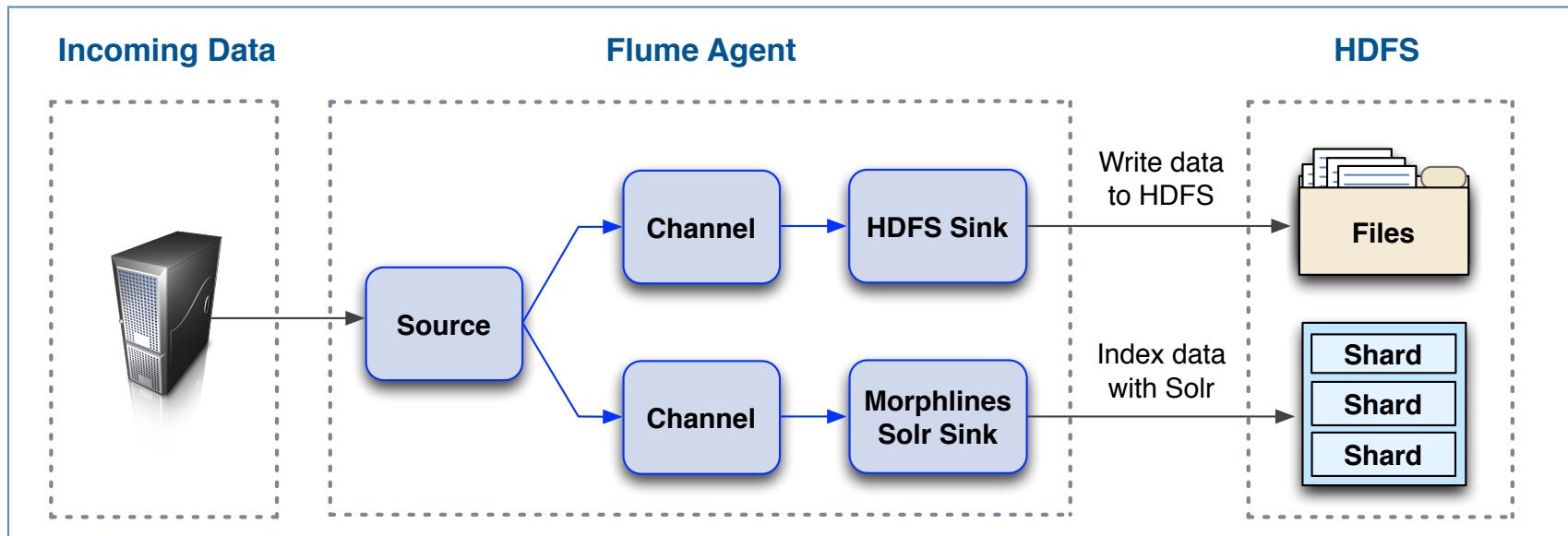
Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- Morphlines
- Indexing Data in Batch Mode
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- **Indexing Data in Near Real Time**
- Essential Points
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Overview of Near-Real-Time Indexing

- **Near-real-time mode is typically used for indexing data not already in HDFS**
 - Incoming data is captured by a Flume agent
- **Output of Flume source goes to two separate channels**
 - One channel uses HDFS sink to write the content to HDFS
 - Other channel uses MorphlineSolrSink to index the data using Solr



Flume Configuration Example (1)

- Example of the configuration file a Flume agent might use

```
# Name the components on this agent
agent1.sources = spooldirsource
agent1.sinks = docsink solrsink
agent1.channels = docchannel solrchannel

# incoming documents to be indexed are added to local directory
agent1.sources.spooldirsource.type = spooldir
agent1.sources.spooldirsource.spoolDir = /home/training/incoming_data
agent1.sources.spooldirsource.batchSize = 200
agent1.sources.spooldirsource.channels = docchannel solrchannel
```

Note: file continues on next slide

Flume Configuration Example (2)

- Here we configure the sink used to write incoming data to HDFS

```
agent1.sinks.docsink.type=hdfs  
agent1.sinks.docsink.hdfs.path = /user/training/employeedata  
agent1.sinks.docsink.channel = docchannel  
  
agent1.channels.docchannel.type = memory  
agent1.channels.docchannel.capacity = 10000  
agent1.channels.docchannel.transactionCapacity = 1000
```

Note: file continues on next slide

Flume Configuration Example (3)

- Finally, we configure the sink used to index data using Morphlines and Solr

```
agent1.sinks.solrsink.type=org.apache.flume.sink.solr.morphline.Morphline
SolrSink
agent1.sinks.solrsink.morphlineFile=/home/training/morphlines.conf
agent1.sinks.solrsink.channel = solrchannel

agent1.channels.solrchannel.type = memory
agent1.channels.solrchannel.capacity = 10000
```

Note: content of first line wraps due to limited space

Morphlines Configuration Example (1)

- This file indexes tab-delimited data to `employees_collection`
 - Collection must exist in Solr prior to starting the Flume agent

```
SOLR_LOCATOR : {
    collection : employees_collection
    zkHost : "dev.loudacre.com:2181/solr"
}

morphlines : [
{
    id : morphline1
    importCommands : ["com.cloudera.**", "org.apache.solr.**"]
    commands : [
{
        readCSV {
            separator : "\t"
            columns : ["firstname", "lastname", "salary", "city", "state", "zipcode"]
            charset : UTF-8
        }
    }
}
```

Note: file continues on next slide

Morphlines Configuration Example (2)

- **Morphlines is configured to also generate a unique field named id**
 - Use this field as the `uniqueKey` in our schema
- **Finally, it removes any unknown fields and indexes data using Solr**

```
{ generateUUID { field: id } }

{ sanitizeUnknownSolrFields { solrLocator : ${SOLR_LOCATOR} } }

{ loadSolr { solrLocator : ${SOLR_LOCATOR} } }

]

}

]
```

Starting Flume Agent and Indexing Documents

- Final step is to start the Flume agent

```
$ sudo flume-ng agent \
  --name agent1
  --conf /etc/flume-ng/conf \
  --conf-file flume.conf
```

- Incoming data will be indexed on arrival
 - In this example, copy data to the configured spool directory
- Edit the `log4j.properties` file in Flume's `conf` directory if needed
 - Helpful for debugging problems with Morphlines
- You can use the Solr Admin UI to verify indexing, as described earlier

Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- Morphlines
- Indexing Data in Batch Mode
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- Indexing Data in Near Real Time
- **Essential Points**
- Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera Search User Guide**
 - <http://tiny.cloudera.com/adcc14a>
- **Introducing Morphlines: The Easy Way to Build and Integrate ETL Apps for Hadoop**
 - <http://tiny.cloudera.com/adcc14b>
- **Email Indexing Using Cloudera Search**
 - <http://tiny.cloudera.com/adcc14c>
- **Morphlines Reference Guide**
 - <http://tiny.cloudera.com/adcc14d>

Essential Points

- **Data must be indexed before it can be searched**
- **The `solrctl` command-line utility is used for pre-indexing steps**
 - Generating configuration templates
 - Uploading the configuration to ZooKeeper (`instancedir`)
 - Creating the collection
- **Every collection must have a schema**
 - Remove all fields in `schema.xml` template before adding your own
 - Should use the `uniqueKey` element to define an `id` field
 - Must also define a `_version_` field of type long
- **Search supports both batch and near-real-time indexing**
 - Batch is best when data already exists in HDFS
 - Near-real-time is used when the data streams in through Flume
 - Both approaches use Morphlines

Chapter Topics

Indexing Data with Cloudera Search

- Collection and Schema Management
- Morphlines
- Indexing Data in Batch Mode
- Hands-On Exercise: Batch Indexing of Knowledge Base Articles
- Indexing Data in Near Real Time
- Essential Points
- **Hands-On Exercise: Indexing Support Chat Transcripts in Near Real Time**

Hands-On Exercise: Indexing Support Chat Transcripts in NRT

- In this Hands-On Exercise, you will index transcripts of support representatives' customer chat sessions using Flume
 - Please refer to the Hands-On Exercise Manual for instructions

Presenting Results to Users

Chapter 15



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- **Presenting Results to Users**
- Conclusion

Presenting Results to Users

In this chapter you will learn

- How to create and execute queries on data indexed with Cloudera Search
- How to develop a customized UI for Search queries using Hue
- How applications can retrieve data from Impala and Search

Chapter Topics

Presenting Results to Users

- **Solr Query Syntax**
- Faceted Search
- Building a Search UI with Hue
- Accessing Impala through JDBC
- Powering a Custom Web Application with Impala and Search
- Essential Points
- Hands-On Exercise: Building the Application UI

Example Overview

- **The following slides describe basic techniques for Solr queries**
 - Performed using Solr's Admin UI
 - Intended to teach syntax
- **Based on a collection of employee data, including these fields**
 - First and last name
 - City, state, and ZIP code
 - Department
 - Date of hire
 - Salary

Recap: Searching By Field

- Use **field:value** in the q box and then execute the query
 - Response shows the query, the number of hits, and the results

The screenshot shows the Apache Solr Admin interface. On the left, there's a sidebar with various navigation links like Dashboard, Logging, Cloud, Core Admin, Java Properties, Thread Dump, Overview, Analysis, Config, Dataimport, Documents, Ping, Plugins / Stats, and Query (which is currently selected). The main area has a form titled "Request-Handler (qt)" with fields for "q" (containing "firstname:Alice"), "fq", "sort", "start", "rows", "fl", and "Raw Query Parameters" (containing "key1=val1&key2=val2"). It also has checkboxes for "indent" (checked) and "debugQuery". At the bottom is a blue "Execute Query" button. To the right, the response is displayed in JSON format. The "q" field in the query and the "firstname:Alice" entries in the results are highlighted with red boxes.

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "firstname:Alice",
      "_": "1395799133248",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 3,
    "start": 0,
    "docs": [
      {
        "hiredate": "2009-09-20T16:00:00Z",
        "department": "Sales",
        "zipcode": "94226",
        "state": "CA",
        "lastname": "Daly",
        "firstname": "Alice",
        "city": "Sacramento",
        "id": "328",
        "salary": 70500,
        "_version_": 1463597701037817900
      },
      {
        "hiredate": "2011-05-29T16:00:00Z",
        "department": "Operations",
        "zipcode": "97407",
        "state": "OR",
        "lastname": "Cardenas",
        "firstname": "Alice",
        "city": "Eugene",
        "id": "969",
        "_version_": 1463597701037817900
      }
    ]
  }
}
```

Field Searches are Precise by Default

- A partial value does not result in a match
- Values containing spaces or keywords must be quoted in the query

city:"Palo Alto"

The screenshot shows a search interface with the following fields:

- Request-Handler (qt): /select
- common:
 - q: firstname:A (highlighted with a red box)
 - fq: (empty)
 - sort: (empty)
 - start, rows: 0, 10
 - fl: (empty)
- Raw Query Parameters: key1=val1&key2=val2
- wt: json
- checkboxes:
 - indent
 - debugQuery
- Execute Query button

The JSON response on the right side of the interface is:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 6,
    "params": {
      "indent": "true",
      "q": "firstname:A",
      "_": "1395802801554",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 0,
    "start": 0,
    "docs": []
  }
}
```

Using Wildcards in Searches

- An asterisk matches zero or more characters

- `firstname:Ann*`
will match Ann, Anne, Anna, Annie, etc.
 - `* : *` matches any field with any value (i.e. all records are returned)

- A question mark matches exactly one character

- `firstname:Ann?`
matches Anne or Anna, but not Ann or Annie

The screenshot shows a search interface with the following fields:

- Request-Handler (qt): /select
- common q: `firstname:Al*` (highlighted with a red box)
- fq: (empty)
- sort: (empty)
- start, rows: 0, 10
- fl: (empty)
- Raw Query Parameters: key1=val1&key2=val2
- wt: json
- checkboxes: indent (checked), debugQuery
- button: Execute Query

The results are displayed as JSON:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "firstname:Al*",
      "_": "1395803038888",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 44,
    "start": 0,
    "docs": [
      {
        "hiredate": "2013-12-15T17:00:00Z",
        "department": "Sales",
        "zipcode": "92159",
        "state": "CA",
        "lastname": "White",
        "firstname": "Alyssa",
        "city": "San Diego",
        "id": "2629",
        "salary": 48700,
        "_version_": 1463597700373020700
      },
      {
        "hiredate": "2013-12-29T17:00:00Z",
        "department": "Quality Assurance",
        "zipcode": "93797",
        "state": "CA",
        "lastname": "Dudley",
        "firstname": "Alvin",
        "city": "Fresno",
        ...
      }
    ]
  }
}
```

Using Ranges in Searches

- Use square brackets and the TO keyword to search fields by range
- The TO keyword must be entered in uppercase
- Can use asterisks for lower or upper bounds

salary: [* TO 60000]

salary: [50000 TO *]

Request-Handler (qt)

/select

common

q salary:[50000 TO 60000]

fq

sort

start, rows
0 10

fl

Raw Query Parameters
key1=val1&key2=val2

wt json

indent

debugQuery

Execute Query

```
{
  "hiredate": "2013-12-08T17:00:00Z",
  "department": "Operations",
  "zipcode": "86047",
  "state": "AZ",
  "lastname": "Tabor",
  "firstname": "Elizabeth",
  "city": "Flagstaff",
  "id": "2539",
  "salary": 51500,
  "_version_": 1463597700219928600
},
{
  "hiredate": "2013-12-08T17:00:00Z",
  "department": "Technical Publications",
  "zipcode": "91967",
  "state": "CA",
  "lastname": "Ochoa",
  "firstname": "George",
  "city": "San Diego",
  "id": "2548",
  "salary": 57000,
  "_version_": 1463597700233560000
},
{
  "hiredate": "2013-12-08T17:00:00Z",
  "department": "Corporate Communication",
  "zipcode": "94725",
  "state": "CA",
  "lastname": "Stoddard",
  "firstname": "Harold",
  "city": "Berkeley",
  "id": "2551",
  "salary": 51000,
  "_version_": 1463597700252434400
}
```

Narrowing Results with AND

- Use AND to filter the search with additional criteria
- Matching requires that all conditions are true, just as with SQL
- The AND keyword must be entered as uppercase

Request-Handler (qt)

/select

common

q
firstname:Boris AND city:Sacramento

fq

sort

start, rows
0 10

fl

Raw Query Parameters
key1=val1&key2=val2

wt
json

indent

debugQuery

Execute Query

```
{  
  "responseHeader": {  
    "status": 0,  
    "QTime": 18,  
    "params": {  
      "indent": "true",  
      "q": "firstname:Boris AND city:Sacramento",  
      "_": "1395804221349",  
      "wt": "json"  
    }  
  },  
  "response": {  
    "numFound": 2,  
    "start": 0,  
    "docs": [  
      {  
        "hiredate": "2011-11-07T17:00:00Z",  
        "department": "Legal",  
        "zipcode": "95854",  
        "state": "CA",  
        "lastname": "Wilcox",  
        "firstname": "Boris",  
        "city": "Sacramento",  
        "id": "1169",  
        "salary": 119900,  
        "_version_": 1463597696526844000  
      },  
      {  
        "hiredate": "2013-07-21T16:00:00Z",  
        "department": "Marketing",  
        "zipcode": "95751",  
        "state": "CA",  
        "lastname": "Badenov",  
        "firstname": "Boris",  
        "city": "Sacramento",  
      }  
    ]  
  }  
}
```

Expanding Results with OR

- Use OR to expand the match criteria
- Matching requires that *any* condition is true, just as with SQL
- The OR keyword must be entered as uppercase
- Keywords must be quoted if used as literals in query

state: "OR"

The screenshot shows a 'Request-Handler (qt)' interface with the following parameters:

- /select
- common
- q: firstname:Ron OR city:Berkeley
- fq: (empty)
- sort: (empty)
- start, rows: 0, 10
- fl: (empty)
- Raw Query Parameters: key1=val1&key2=val2
- wt: json
- checkbox: indent (checked)
- checkbox: debugQuery
- Execute Query button

The resulting JSON response is:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "firstname:Ron OR city:Berkeley",
      "_": "1395805568472",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 58,
    "start": 0,
    "docs": [
      {
        "hiredate": "2013-12-01T17:00:00Z",
        "department": "Technical Publications",
        "zipcode": "94233",
        "state": "CA",
        "lastname": "McClure",
        "firstname": "Ron",
        "city": "Sacramento",
        "id": "2483",
        "salary": 56600,
        "_version_": 1463597700140236800
      },
      {
        "hiredate": "2013-12-08T17:00:00Z",
        "department": "Corporate Communication",
        "zipcode": "94725",
        "state": "CA",
        "lastname": "Stoddard",
        "firstname": "Harold",
        "city": "Berkeley"
      }
    ]
}
```

Sorting Results

- The **sort** field is used to specify how results should be sorted
- Use the **ASC** modifier for ascending order and **DESC** for descending order
 - There is no default!
- Use commas to specify sort on multiple fields

salary DESC, lastname ASC

Request-Handler (qt)

/select

common

q city:Berkeley

fq

sort salary DESC

start, rows 0 10

fl

Raw Query Parameters key1=val1&key2=val2

wt json

indent

debugQuery

Execute Query

```
{
  "hiredate": "2008-07-20T16:00:00Z",
  "department": "Executive",
  "zipcode": "94783",
  "state": "CA",
  "lastname": "Kimson",
  "firstname": "Leo",
  "city": "Berkeley",
  "id": "1",
  "salary": 191800,
  "_version_": 1463597695886164000
},
{
  "hiredate": "2008-09-10T16:00:00Z",
  "department": "Executive",
  "zipcode": "94761",
  "state": "CA",
  "lastname": "Delvalle",
  "firstname": "Noe",
  "city": "Berkeley",
  "id": "35",
  "salary": 179000,
  "_version_": 1463597701060886500
},
{
  "hiredate": "2010-12-19T17:00:00Z",
  "department": "Legal",
  "zipcode": "94762",
  "state": "CA",
  "lastname": "Fleming",
  "firstname": "Dorothy",
  "city": "Berkeley",
  "id": "783",
  "salary": 170300,
}
```

Result Paging (1)

- A search may match many thousands or even millions of items
 - By default, only the first ten are returned

Request-Handler (qt)

/select

— common —————

q
state:AZ

fq

sort

start, rows

fl

Raw Query Parameters
key1=val1&key2=val2

wt
json

indent

debugQuery

Execute Query

```
{  
  "responseHeader": {  
    "status": 0,  
    "QTime": 0,  
    "params": {  
      "sort": "salary desc",  
      "indent": "true",  
      "q": "state:AZ",  
      "_": "1395806887200",  
      "wt": "json"  
    }  
  },  
  "response": {  
    "numFound": 254,  
    "start": 0,  
    "docs": [  
      {  
        "hiredate": "2013-11-03T17:00:00Z",  
        "department": "Engineering",  
        "zipcode": "85553",  
        "state": "AZ",  
        "lastname": "Bradshaw",  
        "firstname": "Jerome",  
        "city": "Globe",  
        "id": "2123",  
        "salary": 101500,  
        "_version_": 1463597699578200000  
      },  
      {  
        "hiredate": "2012-06-03T16:00:00Z",  
        "department": "Engineering",  
        "zipcode": "85079",  
        "state": "AZ",  
        "lastname": "Wentworth",  
        "firstname": "Lorraine",  
        "city": "Payson",  
        "id": "2124",  
        "salary": 101500,  
        "_version_": 1463597699578200001  
      }  
    ]  
  }  
}
```

Result Paging (2)

- Use the `start` and `rows` fields for paging through the entire set of results

Request-Handler (qt)

/select

— common —

q
state:AZ

fq

sort

start, rows

fl

Raw Query Parameters
key1=val1&key2=val2

wt
json

indent

debugQuery

Execute Query

```
{  
  "responseHeader": {  
    "status": 0,  
    "QTime": 1,  
    "params": {  
      "sort": "salary desc",  
      "indent": "true",  
      "start": "0",  
      "q": "state:AZ",  
      "_": "1395807051587",  
      "wt": "json",  
      "rows": "5"  
    }  
  },  
  "response": {  
    "numFound": 254,  
    "start": 0,  
    "docs": [  
      {  
        "hiredate": "2013-11-03T17:00:00Z",  
        "department": "Engineering",  
        "zipcode": "85553",  
        "state": "AZ",  
        "lastname": "Bradshaw",  
        "firstname": "Jerome",  
        "city": "Globe",  
        "id": "2123",  
        "salary": 101500,  
        "_version_": 1463597699578200000  
      },  
      {  
        "hiredate": "2012-06-03T16:00:00Z",  
        "department": "Engineering",  
        "zipcode": "85079",  
        "state": "AZ",  
        "lastname": "Fitzpatrick",  
        "firstname": "Cathy",  
        "city": "Phoenix",  
        "id": "1234",  
        "salary": 101500,  
        "_version_": 1463597699578200000  
      }  
    ]  
  }  
}
```

Field Selection

- Specify a comma-delimited list of fields in the `f1` parameter to control which fields are returned

Request-Handler (qt)

/select

common

q

```
*:*
```

fq

sort

start, rows

0 10

f1

```
firstname,lastname,salary
```

Raw Query Parameters

key1=val1&key2=val2

wt

json

indent

debugQuery

Execute Query

```
{  
  "responseHeader": {  
    "status": 0,  
    "QTime": 1,  
    "params": {  
      "f1": "firstname,lastname,salary",  
      "indent": "true",  
      "q": "*:*",  
      "_": "1395807671401",  
      "wt": "json"  
    }  
  },  
  "response": {  
    "numFound": 3148,  
    "start": 0,  
    "docs": [  
      {  
        "lastname": "Rangel",  
        "firstname": "Earl",  
        "salary": 64300  
      },  
      {  
        "lastname": "Gallant",  
        "firstname": "Elizabeth",  
        "salary": 68300  
      },  
      {  
        "lastname": "Tabor",  
        "firstname": "Elizabeth",  
        "salary": 51500  
      },  
      {  
        "lastname": "Shannon",  
        "firstname": "Michael",  
        "salary": 51500  
      }  
    ]  
  }  
}
```

Result Formats

- Results are returned in JSON format by default
- Use the **wt** (writer type) field to request results in another format
- Many formats supported
 - JSON
 - XML
 - CSV
 - Python
 - Ruby
 - PHP

Request-Handler (qt)

/select

common

q
state:CA

fq

sort

start, rows
0 10

fl
firstname,lastname,salary

Raw Query Parameters
key1=val1&key2=val2

wt
 ▼

indent

debugQuery

Execute Query

```
<?xml version="1.0" encoding="UTF-8"?>
<response>

<lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">0</int>
<lst name="params">
  <str name="fl">firstname,lastname,salary</str>
  <str name="indent">true</str>
  <str name="q">state:CA</str>
  <str name="_>1395808183141</str>
  <str name="wt">xml</str>
</lst>
</lst>
<result name="response" numFound="2345" start="0">
  <doc>
    <str name="lastname">Rangel</str>
    <str name="firstname">Earl</str>
    <int name="salary">64300</int></doc>
  <doc>
    <str name="lastname">Gallant</str>
    <str name="firstname">Elizabeth</str>
    <int name="salary">68300</int></doc>
  <doc>
    <str name="lastname">Shannon</str>
    <str name="firstname">Michael</str>
    <int name="salary">112100</int></doc>
  <doc>
    <str name="lastname">Vinson</str>
    <str name="firstname">Elsa</str>
    <int name="salary">43100</int></doc>
  <doc>
    <str name="lastname">Strait</str>
    <str name="firstname">Emanuel</str>
```

HTTP API

- Each result page displays the URL used to request the results from Solr
- The Solr Admin UI is a convenient way to build queries that will be used in other applications to access and display the results.

The screenshot shows the Solr Admin UI interface for the Request-Handler (qt). On the left, there are several input fields and checkboxes:

- Request-Handler (qt): /select
- common:
 - q: state:AZ
 - fq: (empty)
 - sort: (empty)
 - start, rows: 0, 5
 - fl: (empty)
- Raw Query Parameters: key1=val1&key2=val2
- wt: json
- checkboxes: indent (checked), debugQuery

At the bottom is a blue "Execute Query" button. To the right of the interface is the resulting JSON response:

```
{  
  "responseHeader": {  
    "status": 0,  
    "QTime": 1,  
    "params": {  
      "sort": "salary desc",  
      "indent": "true",  
      "start": "0",  
      "q": "state:AZ",  
      "_": "1395807051587",  
      "wt": "json",  
      "rows": "5"  
    }  
  },  
  "response": {  
    "numFound": 254,  
    "start": 0,  
    "docs": [  
      {  
        "hiredate": "2013-11-03T17:00:00Z",  
        "department": "Engineering",  
        "zipcode": "85553",  
        "state": "AZ",  
        "lastname": "Bradshaw",  
        "firstname": "Jerome",  
        "city": "Globe",  
        "id": "2123",  
        "salary": 101500,  
        "_version_": 1463597699578200000  
      },  
      {  
        "hiredate": "2012-06-03T16:00:00Z",  
        "department": "Marketing",  
        "zipcode": "85553",  
        "state": "AZ",  
        "lastname": "Fitzgerald",  
        "firstname": "Patty",  
        "city": "Globe",  
        "id": "2124",  
        "salary": 101500,  
        "_version_": 1463597699578200000  
      }  
    ]  
  }  
}
```

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- **Faceted Search**
- Building a Search UI with Hue
- Accessing Impala through JDBC
- Powering a Custom Web Application with Impala and Search
- Essential Points
- Hands-On Exercise: Building the Application UI

Faceted Search

- A *facet* is an attribute that can be used to classify data
 - Often used to filter search results
- There are three common types of facets in Solr

Type	Filters Results Based On...	Examples
Field	Discrete field value	<ul style="list-style-type: none">• State of residence• Department name
Range	Numbers in a continuous range of values	<ul style="list-style-type: none">• Salary between \$50,000 - \$60,000• Rating between 4.0 and 5.0
Date	Dates in a continuous range of values	<ul style="list-style-type: none">• Hired during 2013• Promoted during last 30 days

Common Examples of Faceted Search (1)

Range Facet



Field Facet



Range Facet



amazon Try Prime Your Amazon.com Today's Deals Gift Cards Sell Help

Shop by Department Search USB Cables Go Hello, Sign in Your Account Try Prime Cart Wish List

Computers Brands Best Sellers Laptops & Tablets Desktops & Monitors Hard Drives & Storage Computer Accessories Tablet Accessories PC Components

Department Electronics Computers & Accessories Cables & Accessories Cables & Interconnects USB Cables

Length Under 4 Feet (2,497) 4 to 5.9 Feet (5,936) 6 to 7.9 Feet (12,362) 8 to 9.9 Feet (371) 10 to 14.9 Feet (7,813) 15 to 24.9 Feet (1,868) 25 Feet & Above (66)

Condition New (57,908) Used (1,254) Refurbished (177)

Price Under \$25 (52,120) \$25 to \$50 (3,308) \$50 to \$100 (1,450) \$100 to \$200 (668) \$200 & Above (687)

\$ to \$ GO

Electronics > Computers & Accessories > Cables & Accessories > Cables & Interconnects > USB Cables

Showing 1 - 24 of 58,172 Results Detail Image Sort by Price: Low to High

SANOXY® Right Angle Micro USB OTG to USB 2.0 Adapter
\$0.99 **\$1.60**
In Stock
More Buying Choices
\$0.01 new (50 offers)
\$0.99 used (1 offer)

USB Printer Cord NEW !! 2.0 A - B Cable 6'
\$4.15 Add-on Item
Add it to a qualifying order within 26 hours to get it by Monday, Mar 3
FREE Shipping on orders over \$35
More Buying Choices
\$0.01 new (34 offers)
\$0.98 used (3 offers)

C&E CNE13028 USB 2.0 Female to Micro USB Male OTG On-The-Go Cable Adapter
\$3.47 **\$1.37**
In Stock
More Buying Choices
\$0.01 new (50 offers)
\$0.49 used (2 offers)

See Pattern Options

Common Examples of Faceted Search (2)

Field Facet →

Range Facet →

The screenshot shows the Best Buy website interface. At the top, there's a navigation bar with the Best Buy logo, a search bar, "WEEKLY DEALS", and links for "Stores", "Orders", "Help", "Credit Cards", and "Español". Below the navigation is a blue header bar with "PRODUCTS", "SERVICES", "SHOPS & DEALS", "GIFTS", and a "my BEST BUY" link. To the right of the header is a sign-in link and a shopping cart icon with a '0'.

In the center, a banner says "FREE SHIPPING on orders \$25 and up. See details >". Below it, a breadcrumb trail shows "Best Buy > Appliances > Small Appliances".

Toaster Ovens & Pizza Ovens

You've Selected

Brand: Hamilton Beach
[Remove]
Price: \$50 - \$99.99
[Remove]
Clear all selections

Narrow Your Results

Brand

- Hamilton Beach (18)
- Cuisinart (8)
- DeLonghi (5)
- T-Fal (4)
- Black & Decker (3)
- Coffee Pro (3)
- Elite Gourmet (3)
- Elite Platinum (3)

See All (24)

Price

- Less than \$50 (12)
- \$50 - \$99.99 (18)
- \$100 - \$149.99 (2)

Compare

Hamilton Beach - Convection Toaster Oven - Stainless-Steel

Model: 31333 SKU: 6001452

Accommodates an up to 12" pizza or 6 slices of bread; convection cooking; toast, bake and broil settings; 30-minute timer; stay-on setting; curved glass door; auto-advance rack; slide-out crumb tray

Customer Reviews: ★★★★☆ 4.3 of 5 (9 reviews)

Check Shipping & Availability

Hamilton Beach - Convection Toaster/Pizza Oven - Black/Stainless-Steel

Model: 31331 SKU: 6582697

Fits up to a 12" pizza or 6 slices of bread; convection, toast, bake and broil settings; 30-minute timer with bell; curved glass door; auto advance rack; slide-out crumb tray

Customer Reviews: ★★★★☆ 4.3 of 5 (9 reviews)

Check Shipping & Availability

Hamilton Beach - Set & Forget Programmable Convection Oven - Black

Model: 31230 SKU: 3254355

Customer Reviews: ★★★★☆ 4.4 of 5 (5 reviews)

Check Shipping & Availability

Sale:

See price in cart >
Reg. Price: \$59.99

- Free Shipping on Orders \$25 and Up
- Get up to 6% Back in Rewards: See How

Add to Cart

\$59.99

- Free Shipping on Orders \$25 and Up
- Get up to 6% Back in Rewards: See How

Add to Cart

Sale:

See price in cart >
Reg. Price: \$99.99

- Free Shipping on Orders \$25 and Up

Add to Cart

Common Examples of Faceted Search (3)

The screenshot shows the Yahoo News search results for the query "banking". On the left, there are two vertical menus with red arrows pointing to them from the left side of the slide:

- Field Facet** (points to the "Filter by Source" section)
- Date Facet** (points to the "Filter by Time" section)

The search bar at the top contains "banking". To the right of the search bar are "Sort by Relevance | Time" buttons. The main content area displays five news articles:

- Banking Ikea Style Uses Buffer Gap to Give Credit to Swedes**
Bloomberg Feb 28 02:40am
As Sweden's biggest banks wait for regulators to tighten Europe's toughest capital rules, a less fettered form of credit is gaining traction.
- 6 Signs of a New Age of Consumer Banking**
FOX Business Feb 27 03:05pm
The banking landscape is changing rapidly. How will you and your money be affected?
- D3 Banking to Demonstrate Omnichannel, Data Driven Digital Banking at FinovateSpring 2014**
Business Wire via Yahoo! Finance Feb 27 08:00am
D3 Banking, an innovator in omnichannel, data driven digital™ banking, has been selected to demonstrate its D3 Banking solution at FinovateSpring, April 29-30, 2014 in San Jose. Fi
- Mobile banking apps hit by glitches**
BBC News 2 hours, 40 minutes ago
The mobile banking apps of several big banks are returning to normal after suffering glitches.
- Malauzai Software Introduces Business Mobile Banking**
Business Wire via Yahoo! Finance Feb 27 07:09am
Malauzai Software Inc., a provider of mobile banking SmartApps for community financial institutions, has developed a mobile banking app for banks' commercial account holders. The B
- ECB's Lautenschlaeger: European banking needs one mechanism for failure**
Reuters via Yahoo! Finance 1 hour, 48 minutes ago
The new banking supervision housed in the European Central Bank must be accompanied by a single mechanism to deal with failing banks, ECB Executive Board member Sabine

Aside: Field Types for Faceted Search

- Solr's schema supports special field types optimized for range queries
 - Only relevant for numeric and date types
 - These are called "Trie" fields

Standard Type	Trie Type
int	tint
long	tlong
float	tfloat
double	tdouble
date	tdate

- If you expect to use range or date facets on a field, use one of these types
 - This will provide better performance

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- Faceted Search
- **Building a Search UI with Hue**
- Accessing Impala through JDBC
- Powering a Custom Web Application with Impala and Search
- Essential Points
- Hands-On Exercise: Building the Application UI

Hue Search App UI

- Hue ships with an application which makes it easy to build a Search UI

The screenshot shows the Hue Welcome Home page. At the top is a blue header bar with various application icons: HDFS, File Browser, Hive, Impala, Sqoop, MapReduce, Job Tracker, YARN, Solr, User, and Help. The 'Solr' icon is highlighted with a red box and has a callout bubble labeled 'Solr Search'. Below the header is a large 'Welcome Home.' heading. A sub-header below it reads 'Hue is a Web UI for Apache Hadoop. Select an application below.' Two main application sections are displayed: 'Query' on the left and 'Hadoop' on the right. Each section has a title bar with a grid icon and the section name, followed by a list of links.

Query	Hadoop
» Hive	» Files
» Impala	» Jobs

Hue Search App UI: Instructor-Led Demo

- **The instructor will now demonstrate how to build a Search UI using Hue**
 - This is based on a data set of Loudacre employee information
 - The indexing process is fully scripted and available on your VM at
`~/training_materials/bigdata/demo/hue_search_ui`

Drawbacks of the Hue Search App

- **End user requires access to the Hue server**
 - Hue is typically available only on an internal network
 - Requires users to have login credentials
- **Cannot easily integrate with an external application**
 - Support for this will be improved in a future version
- **Loudacre's application is an external application**
 - Combines data accessed from both Search and Impala
 - Requires more extensive customization than is supported in Hue

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- Faceted Search
- Building a Search UI with Hue
- **Accessing Impala through JDBC**
- Powering a Custom Web Application with Impala and Search
- Essential Points
- Hands-On Exercise: Building the Application UI

Accessing Impala through JDBC

- **Impala was designed for fast interactive queries**
 - It is accessible via both ODBC and JDBC
 - Many Business Intelligence (BI) applications support Impala
 - Your applications can use Impala too
- **Queries work as they would in any other database**
 - Porting can be as easy as changing a connection string
 - Any query that works in the Impala shell should work in JDBC
- **However, Impala does not support some database features**
 - Updating or deleting individual records
 - Transactions, stored procedures, or triggers
 - Impala is intended for read-only data

Adding the JDBC Driver to Your Project's Classpath

- Impala uses the same JDBC driver as HiveServer2
- Add the following to your `pom.xml`
 - The value of `hive.version` varies by CDH release

```
<dependency>
  <artifactId>hive-jdbc</artifactId>
  <groupId>org.apache.hive</groupId>
  <version>${hive.version}</version>
</dependency>
```

Querying Impala through JDBC (1)

- The following class finds the top three states by customer count

```
public class CountAccountsByState {  
  
    String url = "jdbc:hive2://dev.loudacre.com:21050/;auth=noSasl";  
  
    public static void main(String[] args) throws Exception {  
        // load the driver class  
        Class.forName("org.apache.hive.jdbc.HiveDriver");  
  
        Connection conn = null;  
        try {  
            conn = DriverManager.getConnection(url, "alice", "abc123");  
            Statement stmt = conn.createStatement();  
  
            String sql = "SELECT state, COUNT(acct_num) AS customers "  
                + "FROM accounts "  
                + "GROUP BY state "  
                + "ORDER BY customers DESC "  
                + "LIMIT 3";  
            ResultSet res = stmt.executeQuery(sql);  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {  
            if (conn != null) {  
                try {  
                    conn.close();  
                } catch (SQLException e) {  
                    e.printStackTrace();  
                }  
            }  
        }  
    }  
}
```

Cont'd...

Querying Impala through JDBC (2)

```
        while (res.next()) {
            // JDBC indexes start at 1 like SQL, not 0 like Java
            String state = res.getString(1);
            int customers = res.getInt(2);
            System.out.println(state + "\t" + customers);
        }
    } finally {
    if (conn != null) {
        conn.close(); // important
    }
}
}
```

- **It is important to close your connection from a `finally` block**
 - This ensures that memory associated with the query is freed

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- Faceted Search
- Building a Search UI with Hue
- Accessing Impala through JDBC
- **Powering a Custom Web Application with Impala and Search**
- Essential Points
- Hands-On Exercise: Building the Application UI

How to Get Results from Impala

- **Applications can use JDBC or ODBC to query Impala**
 - JDBC approach described earlier
- **Web applications should use a container-managed data source**
 - Makes access convenient from Java
 - Avoids hardcoding credentials and connection details in business logic
 - Better for security
 - May also improve performance
 - Ask your system administrator for assistance
- **Can use standard tag libraries to query and display results in JSP**

Using JSTL to Display Query Results in JSP

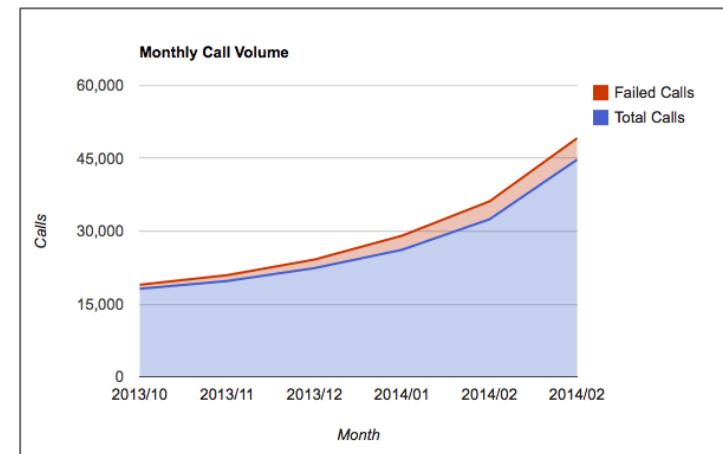
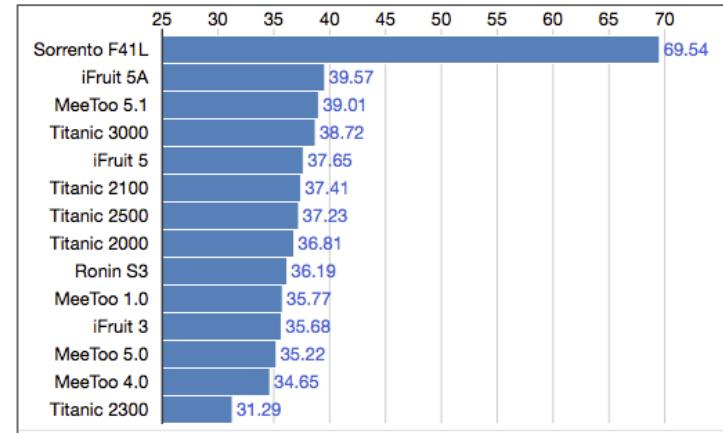
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>

<sql:query var="accounts" dataSource="jdbc/impala">
    SELECT acct_num, first_name, last_name
    FROM accounts WHERE state = 'CA' LIMIT 10
</sql:query>

<table>
    <tr>
        <th>Account Number</th>
        <th>First Name</th>
        <th>Last Name</th>
    </tr>
    <c:forEach var="row" items="${accounts.rows}">
        <tr>
            <td><c:out value="${row.acct_num}" /></td>
            <td><c:out value="${row.first_name}" /></td>
            <td><c:out value="${row.last_name}" /></td>
        </tr>
    </c:forEach>
</table>
```

Visualization

- You can power charts and graphs with Impala
- Many Javascript charting libraries support data access from HTTP requests
 - D3.js
 - Dimple
 - Google Chart API
- Write a servlet or JSP to produce data in the required format (CSV, TSV, JSON, etc.)
 - Configure the charting library to use this as its source



Displaying Results from Search in JSP

- Use JSTL to request XML results and extract content with XPath

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<c:import url="SOLR_QUERY_URL_THAT_RETURNS_XML" var="xml" />
<x:parse doc="${xml}" var="employees" />



| Department | Last Name | Salary |
|------------|-----------|--------|
|            |           |        |


```

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- Faceted Search
- Building a Search UI with Hue
- Accessing Impala through JDBC
- Powering a Custom Web Application with Impala and Search
- **Essential Points**
- Hands-On Exercise: Building the Application UI

Bibliography

The following offer more information on topics discussed in this chapter

- **Secrets of Cloudera Support: Impala and Search Make the Customer Experience Even Better**
 - <http://tiny.cloudera.com/adcc15a>
- **Secrets of Cloudera Support: Inside Our Own Enterprise Data Hub**
 - <http://tiny.cloudera.com/adcc15b>
- **Tutorial: Search Hadoop in Hue**
 - <http://tiny.cloudera.com/adcc15c>
- **Apache Solr Reference Guide (PDF)**
 - <http://tiny.cloudera.com/adcc15d>

Essential Points

- **Cloudera Search uses Solr's powerful syntax to query indexed data**
- **The Solr Admin UI is helpful for testing queries**
 - The URL it displays can be used in custom applications
- **Hue makes it easy to create a user-friendly Search UI**
 - Customizable using a Web-based editor
 - Supports advanced features like facets and highlighting
- **More robust applications may combine results from Impala and Search**
 - Not easily supported by Hue
 - Use Search URL to request results in XML (or other format) over HTTP
 - Use JDBC or ODBC to execute queries with Impala

Chapter Topics

Presenting Results to Users

- Solr Query Syntax
- Faceted Search
- Building a Search UI with Hue
- Accessing Impala through JDBC
- Powering a Custom Web Application with Impala and Search
- Essential Points
- **Hands-On Exercise: Building the Application UI**

Hands-On Exercise: Building the Application UI

- In this Hands-On Exercise, you will put the finishing touches to the prototype Web application that will revolutionize how Loudacre support representatives identify and solve problems for customers
 - Please refer to the Hands-On Exercise Manual for instructions

Conclusion

Chapter 16



Course Chapters

- Introduction
- Application Architecture
- Designing and Using Data Sets
- Using the Kite SDK Data Module
- Importing Relational Data with Apache Sqoop
- Capturing Data with Apache Flume
- Developing Custom Flume Components
- Managing Workflows with Apache Oozie
- Processing Data Pipelines with Apache Crunch
- Working with Tables in Apache Hive
- Developing User-Defined Functions
- Executing Interactive Queries with Impala
- Understanding Cloudera Search
- Indexing Data with Cloudera Search
- Presenting Results to Users
- Conclusion

Conclusion (1)

During this course, you have learned how to

- **Determine which Hadoop-related tools are appropriate for specific tasks**
- **Understand how file formats, serialization, and data compression affect application compatibility and performance**
- **Design and evolve schemas in Apache Avro**
- **Create, populate, and access data sets with the Kite SDK**
- **Integrate with external systems using Apache Sqoop and Apache Flume**
- **Integrate Apache Flume with existing applications and develop custom components to extend Flume's capabilities**
- **Create, package, and deploy Oozie jobs to manage processing workflows**
- **Develop Java-based data processing pipelines with Apache Crunch**

Conclusion (2)

- **Implement user-defined functions for use in Apache Hive and Impala**
- **Index both static and streaming data sets with Cloudera Search**
- **Use Hue to build a Web-based interface for Search queries**
- **Integrate results from Impala and Cloudera Search into your applications**

Which Course to Take Next?

Cloudera offers a range of training courses for you and your team

- **For developers**
 - *Cloudera Developer Training for Apache Hadoop*
 - *Cloudera Training for Apache HBase*
- **For system administrators**
 - *Cloudera Administrator Training for Apache Hadoop*
- **For data analysts and data scientists**
 - *Cloudera Data Analyst Training: Using Pig, Hive, and Impala with Hadoop*
 - *Introduction to Data Science: Building Recommender Systems*
- **For architects, managers, CIOs, and CTOs**
 - *Cloudera Essentials for Apache Hadoop*

The background of the image features a vibrant, multi-colored powder explosion against a teal gradient background. The colors transition from white and light blue on the left, through yellow and orange, to red and purple on the right. The powder is depicted with a mix of fine dots and larger, more defined shapes.

cloudera®
Ask Bigger Questions