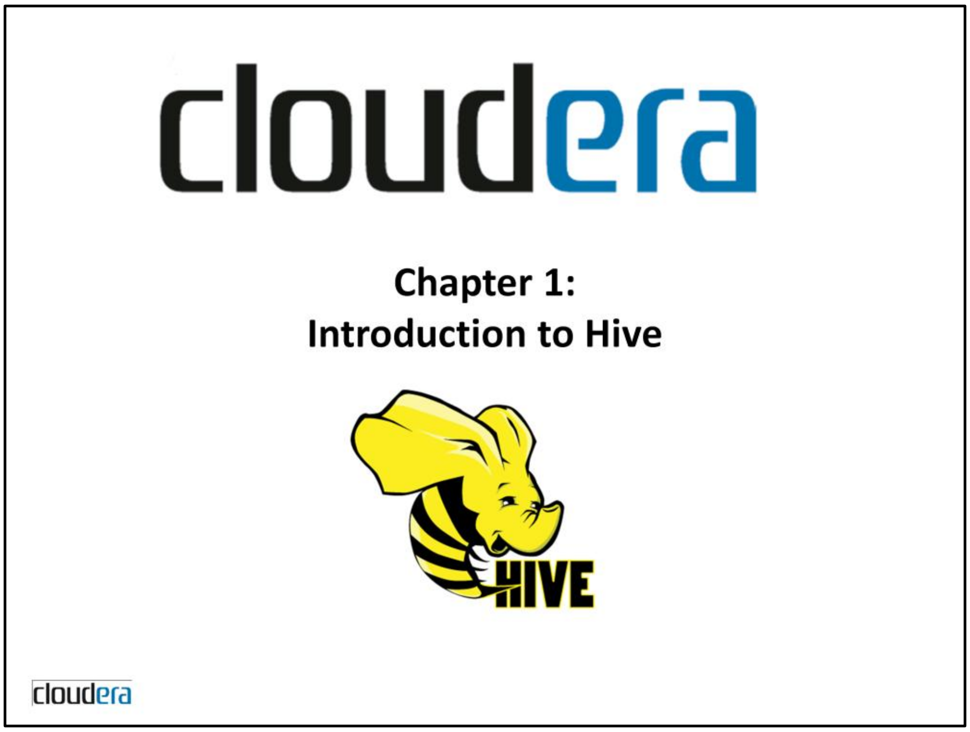
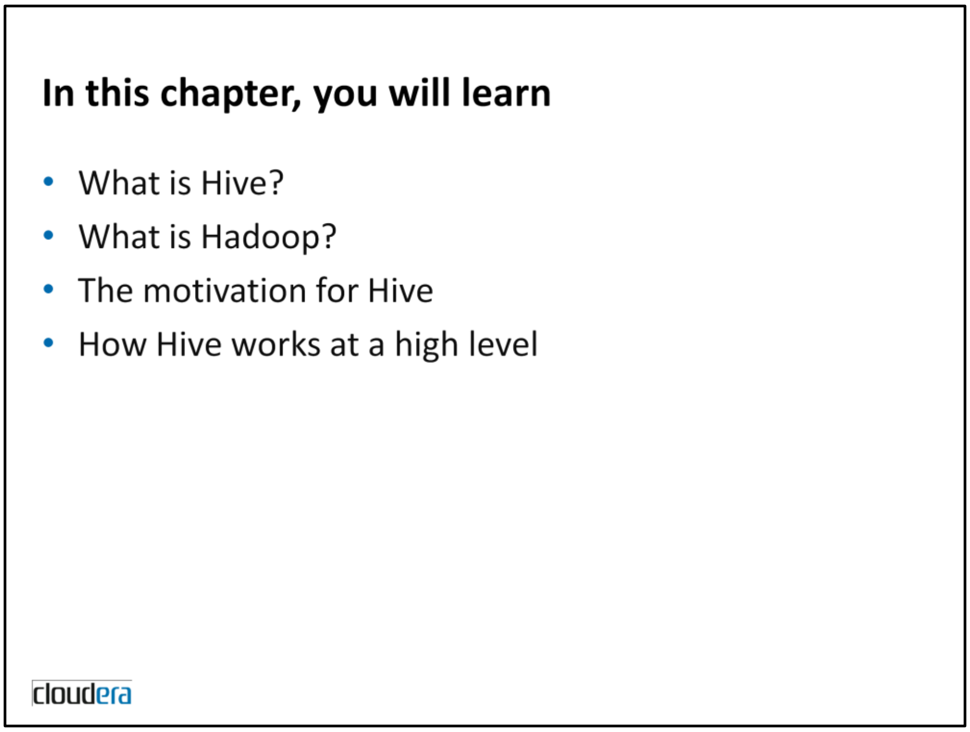
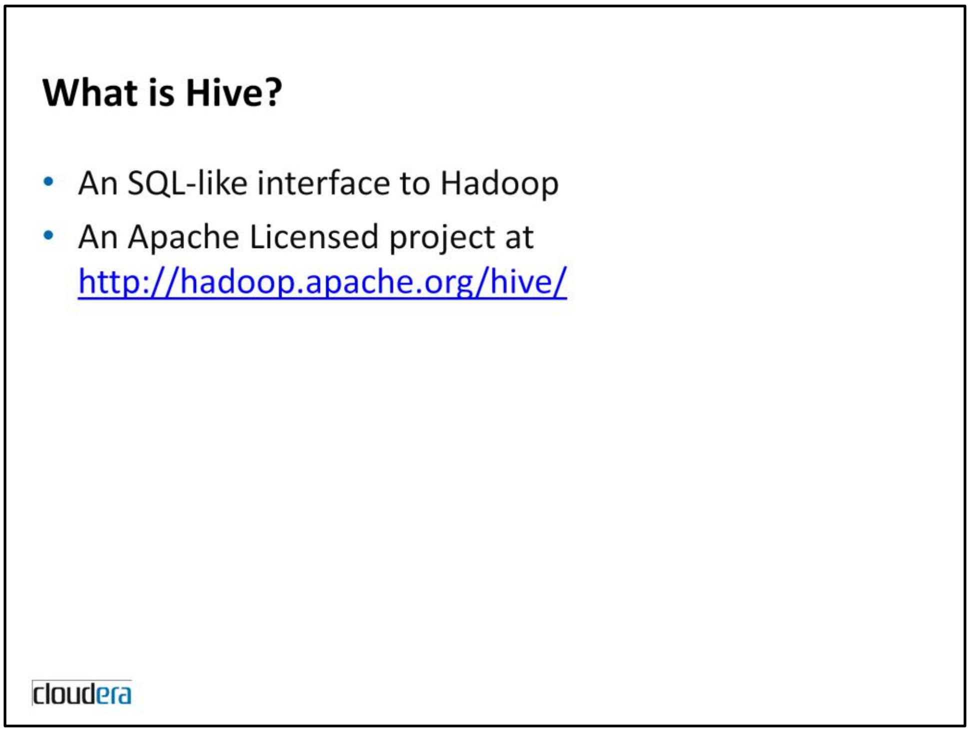


Chapter 1: Introduction to Hive



In this chapter, you will learn

- What is Hive?
- What is Hadoop?
- The motivation for Hive
- How Hive works at a high level



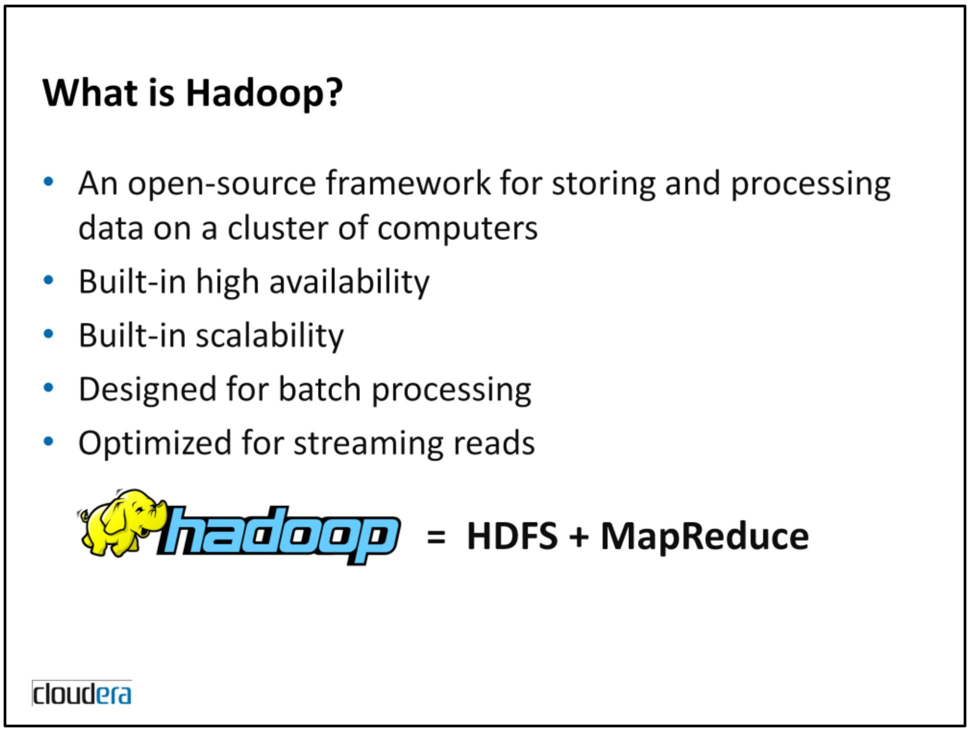
What is Hive?

- An SQL-like interface to Hadoop
- An Apache Licensed project at http://hadoop.apache.org/hive/

cloudera

Hive is an application that exposes Hadoop with an SQL-like language called HiveQL. Hive offers a command-line interface (as well as a web and JDBC/ODBC interface) that handles the complexity of converting queries to Java MapReduce programs.

Hive is an open-source subproject of the Apache Hadoop project. The official Hive project page can be found at http://hadoop.apache.org/hive. It uses an Apache License, Version 2.0.



What is Hadoop?

- An open-source framework for storing and processing data on a cluster of computers
- Built-in high availability
- Built-in scalability
- Designed for batch processing
- Optimized for streaming reads

hadoop = HDFS + MapReduce

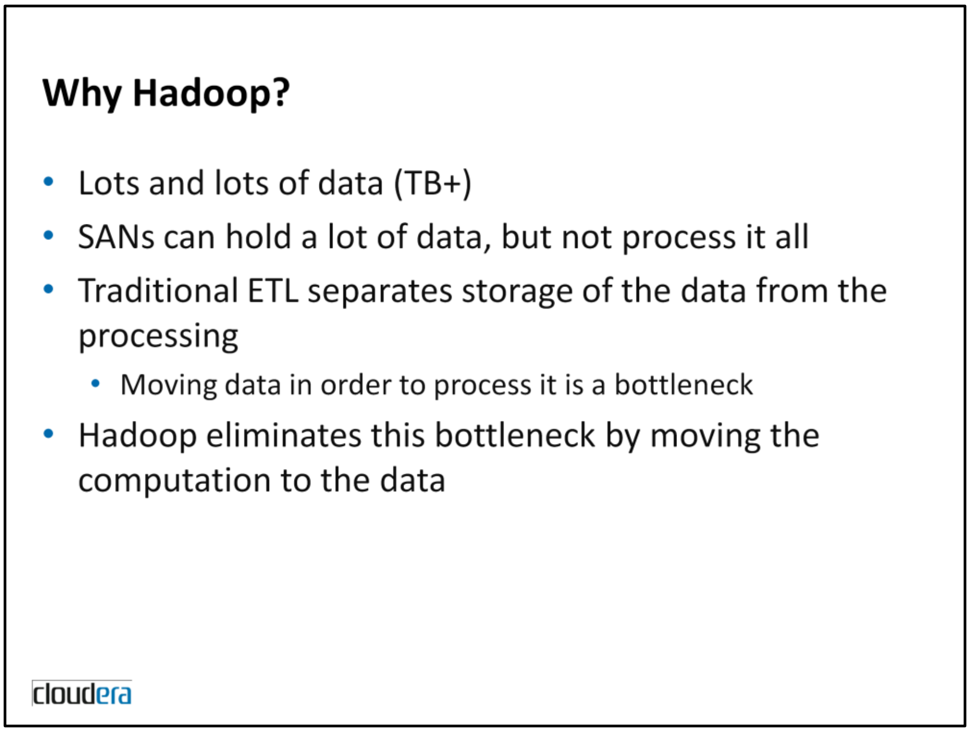
cloudera

Apache Hadoop is an open-source implementation of Google's Distributed File System (GFS) and MapReduce. It was originally created by Doug Cutting* at Yahoo!. Hadoop was named after his son's stuffed elephant.

Hadoop provides high availability and scalability by distributing data and processing across many machines. It can store and process a great deal of data, but it is designed for batch processing, not real time responses. Hadoop has been known to scale to Petabytes of data.

At its core, Hadoop is a combination of the Hadoop Distributed File System (HDFS) and the MapReduce programming paradigm.

* Doug is now an employee at Cloudera



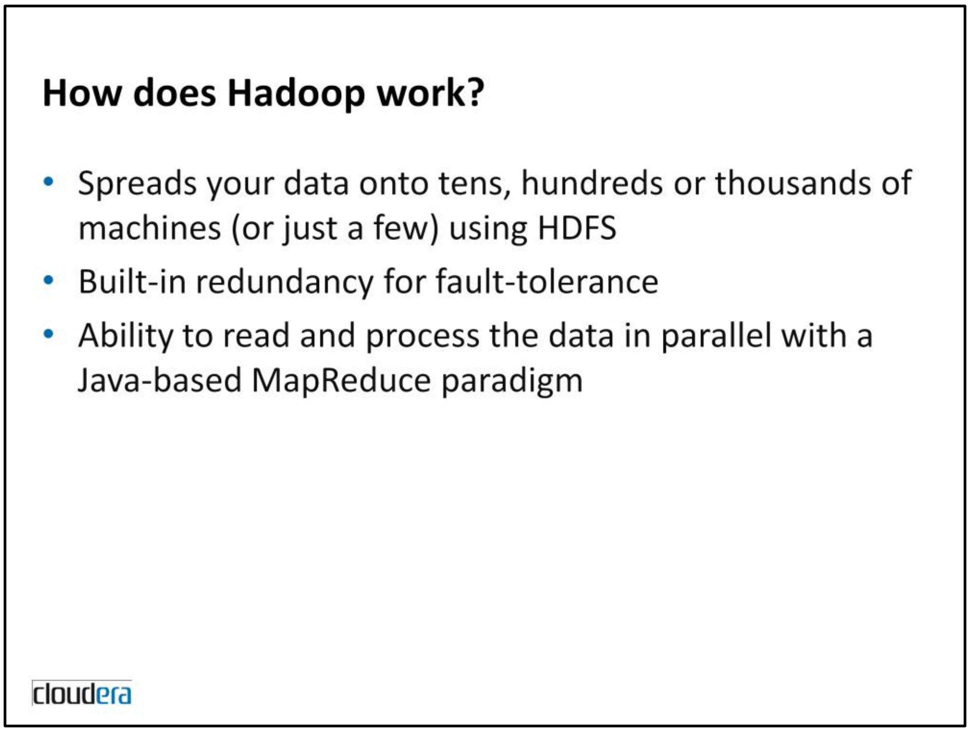
Why Hadoop?

- Lots and lots of data (TB+)
- SANs can hold a lot of data, but not process it all
- Traditional ETL separates storage of the data from the processing
  - Moving data in order to process it is a bottleneck
- Hadoop eliminates this bottleneck by moving the computation to the data

cloudera

The amount of data that companies have collected is tremendous and grows daily. It is often more data than a single disk or array of disks can hold. Storage Area Networks (SANs) are a potential solution for storing massive amounts of data, but they are poorly suited for processing all of the data (e.g., performing Extract-Transform-Load operations). In order to read the data, they have to transfer the bytes from the SAN to a computer. The limiting factor is the transfer rate.

Hadoop takes a different approach to scaling: move the processing to the data, not the data to the processing. By spreading the data onto many machines, Hadoop can take advantage of the CPUs on each of these machines without transferring data around.

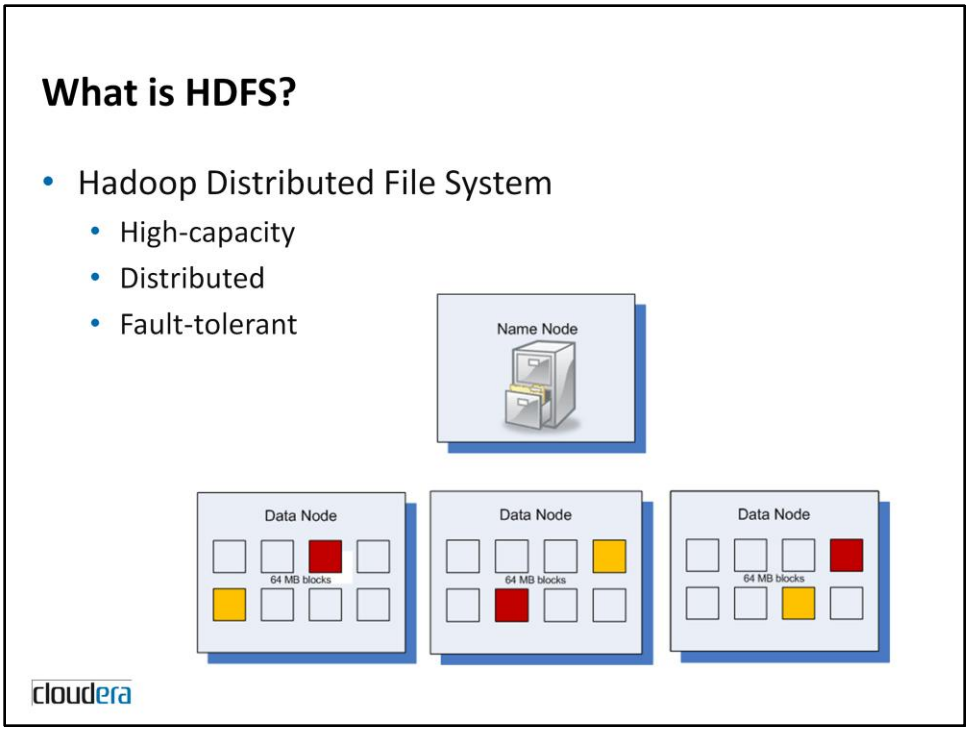

How does Hadoop work?

- Spreads your data onto tens, hundreds or thousands of machines (or just a few) using HDFS
- Built-in redundancy for fault-tolerance
- Ability to read and process the data in parallel with a Java-based MapReduce paradigm

cloudera

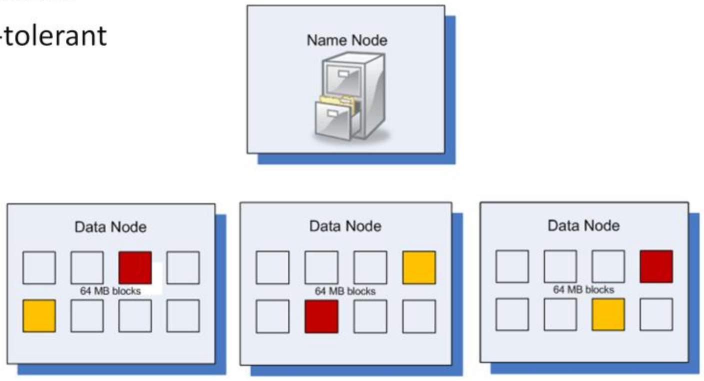
A typical Hadoop cluster consists of dozens (or more) of commodity machines. The more computers in the cluster, the higher the likelihood that a node in your cluster will fail. Hard drives fail, network cards go bad, RAM can have bad bits. Hadoop needs to isolate the user from these issues. It does this with hardware redundancy and software fault-tolerance. Each file loaded into Hadoop is replicated automatically.

Processing data in Hadoop is done with a programming paradigm called MapReduce (based on Google's paper).
http://labs.google.com/papers/mapreduce.html



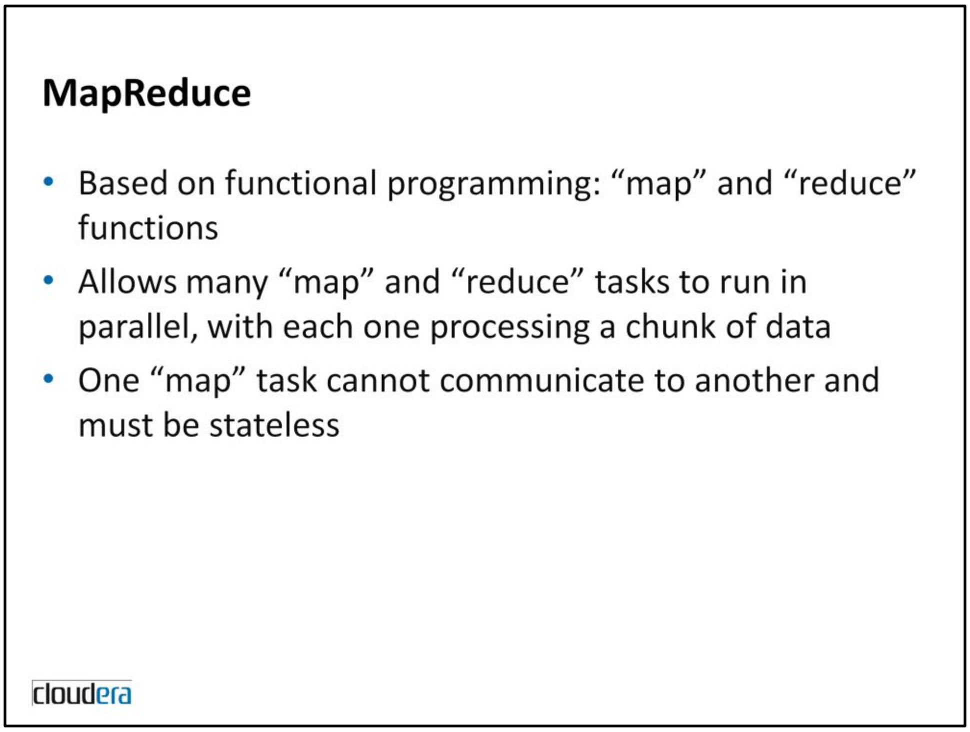
What is HDFS?

- Hadoop Distributed File System
  - High-capacity
  - Distributed
  - Fault-tolerant



Hive stores its data in the Hadoop Distributed File System (HDFS). HDFS is a distributed, fault-tolerant file system that can store large data sets (e.g., petabytes). Data is distributed across tens, hundreds or even thousands of machines. HDFS separates data into blocks (typically 64 or 128 megabytes) and spreads those blocks across the machines in the cluster. It is fault-tolerant by replicating the data (by default each block will be stored 3 times). This allows machines to fail without affecting the user or losing data.

It is possible to use HDFS outside of Hive. For example, as a archiving location for data or for Java MapReduce programs. Hive uses a specific directory for its data, typically /user/hive/warehouse. This location is configurable with a parameter named `hive.metastore.warehouse.dir`. Each table becomes a subdirectory such as /user/hive/warehouse/*tablename*.

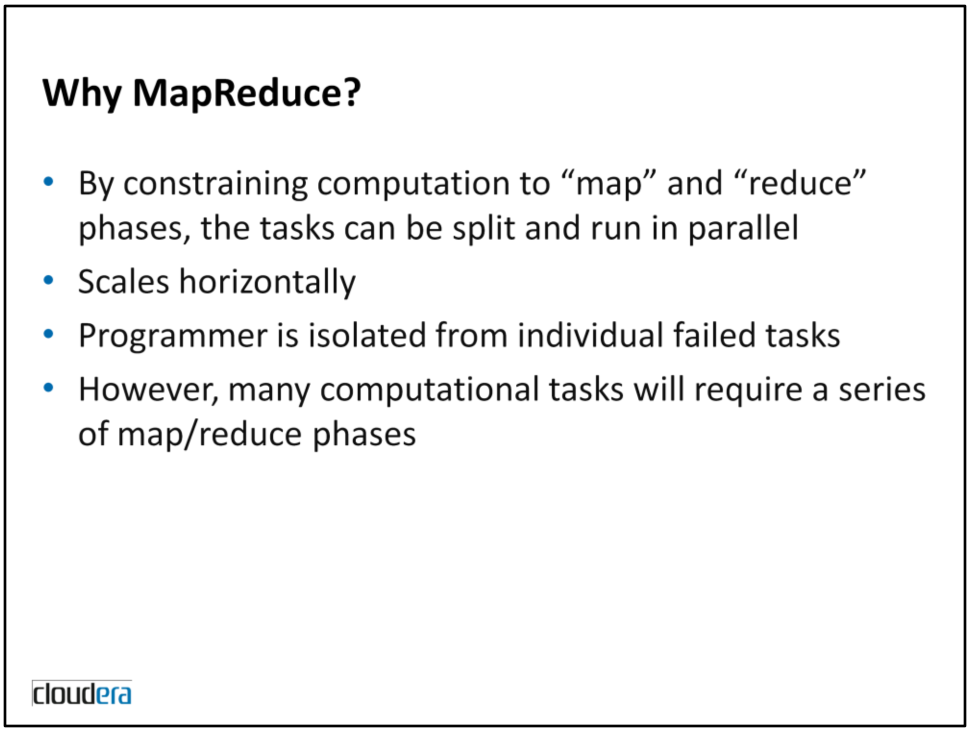


MapReduce

- Based on functional programming: “map” and “reduce” functions
- Allows many “map” and “reduce” tasks to run in parallel, with each one processing a chunk of data
- One “map” task cannot communicate to another and must be stateless

cloudera

MapReduce is a simple programming model consisting of a “map” function and a “reduce” function. The map is applied to every input (e.g., line of a file) in the data set. It reads the line and outputs a (key, value) pair. These (key, value) pairs are collected, sorted by key, and sent to the reducers. The map (and reduce) functions may run many times in parallel on different sets of inputs. Also a given map or reduce function can fail and it will automatically be re-executed. Given this distributed model, one must write the map and reduce functions without sharing state across other map or reduce invocations.

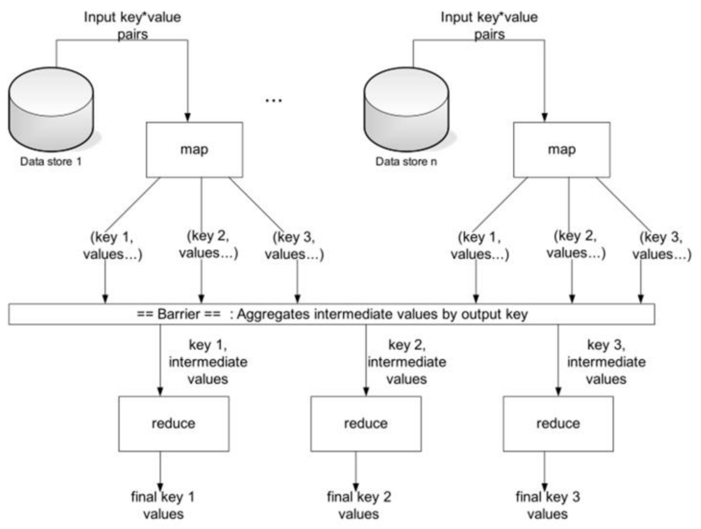


Why MapReduce?

- By constraining computation to “map” and “reduce” phases, the tasks can be split and run in parallel
- Scales horizontally
- Programmer is isolated from individual failed tasks
- However, many computational tasks will require a series of map/reduce phases

MapReduce has a lot of benefits, such as enabling parallelism and software fault-tolerance. This allows the system to scale by adding more nodes. Also, the programmer does not need to handle the case where a task fails. The framework can automatically re-execute that task. However, these benefits come at the cost of framing all problems in terms of map/reduce phases. A higher-level abstraction is required to allow programmers to write in terms of filters, joins, unions, aggregates, etc.

MapReduce

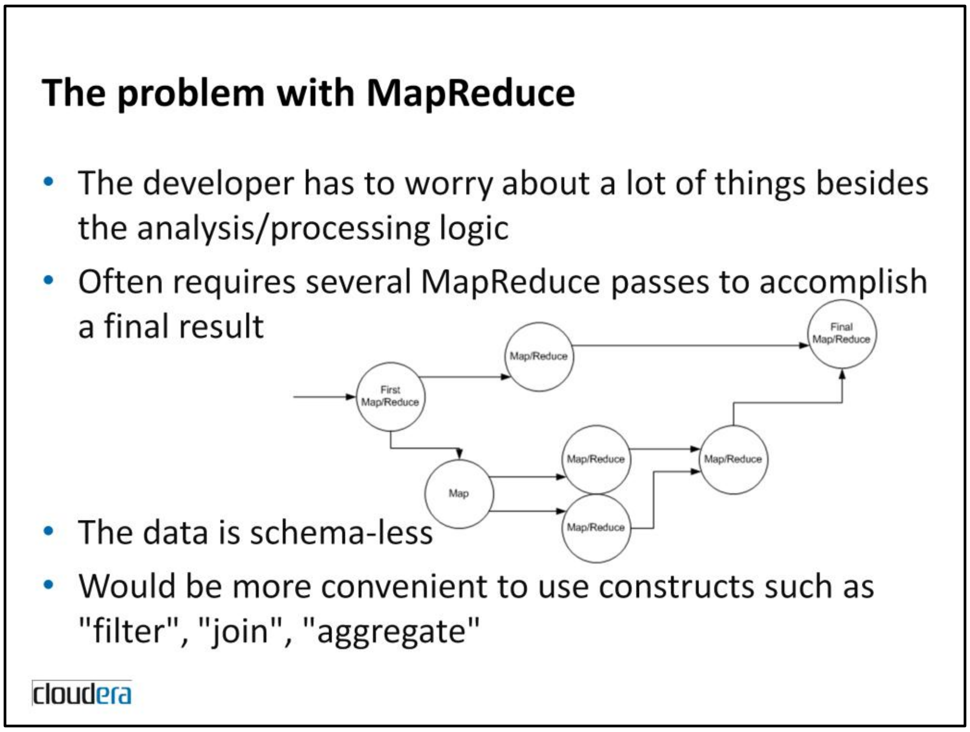


cloudera

Hadoop's MapReduce framework essentially has 2 phases: the map phase and the reduce phase.

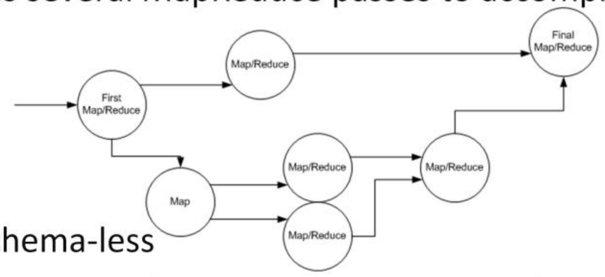
Map phase: The Hadoop framework will execute as many map tasks as required for the input data. These will run in parallel across the nodes of the cluster. The map tasks receive the input data (e.g., lines from a file), execute some code on that input, and emit zero or more key/value pairs.

Reduce phase: A specified number of reduce tasks will receive the key/value pairs that the maps emitted. These key/value pairs will be grouped by key and distributed amongst the reducers. A particular reducer will process its keys in sorted order. The reducers then output key/value pairs which will be stored in files.



The problem with MapReduce

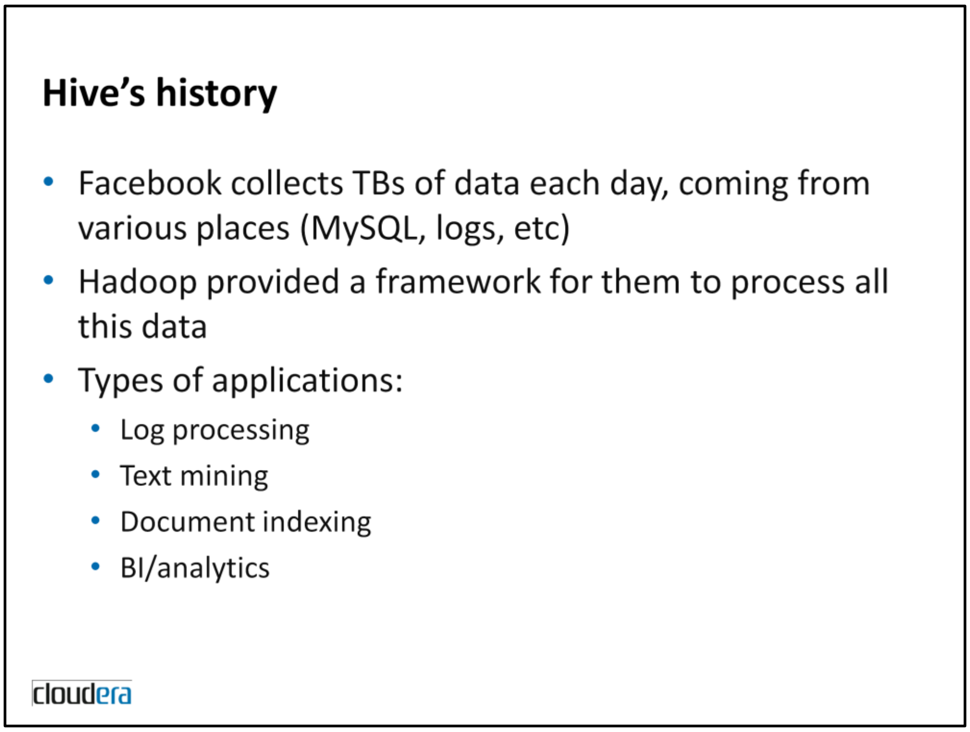
- The developer has to worry about a lot of things besides the analysis/processing logic
- Often requires several MapReduce passes to accomplish a final result



- The data is schema-less
- Would be more convenient to use constructs such as "filter", "join", "aggregate"

cloudera

Hadoop’s MapReduce is very powerful for processing massive amounts of data. But it also has some issues. In addition, there is a fair amount of setup and extra code involved that is unrelated to your analysis/processing logic. Most data processing jobs will actually require mulitple map/reduce jobs. Furthermore, MapReduce acts on files in HDFS, which do not have a schema or any self-describing metadata. This causes developers to embed parsing logic in their analysis code. Finally, it would be more convenient to use common constructs such as filters, joins, aggregates, etc.

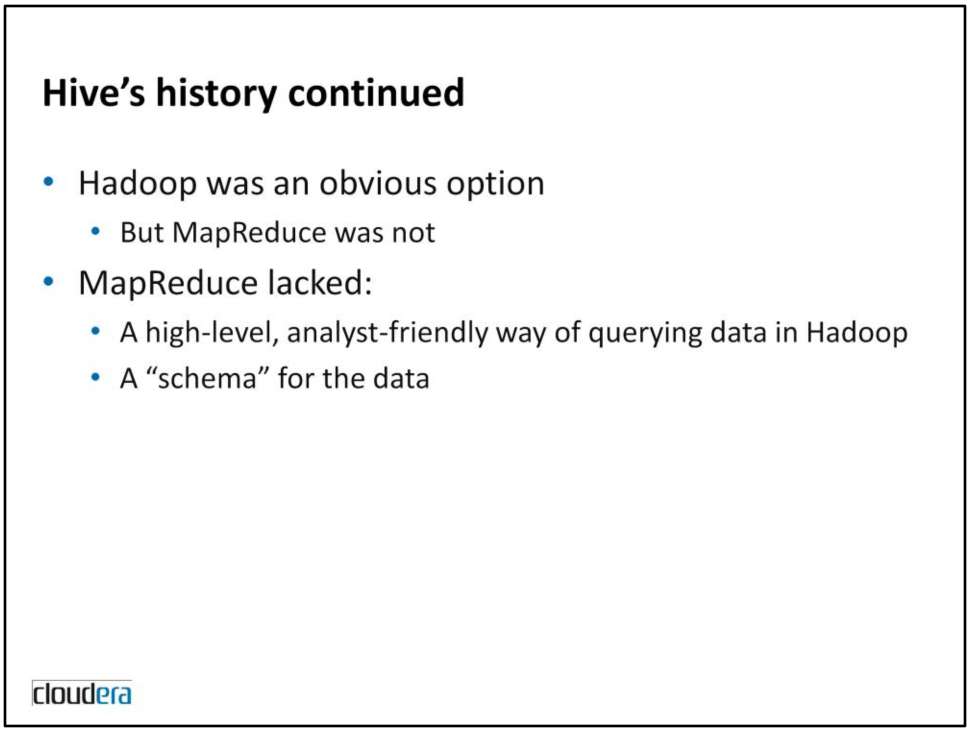


Hive's history

- Facebook collects TBs of data each day, coming from various places (MySQL, logs, etc)
- Hadoop provided a framework for them to process all this data
- Types of applications:
    - Log processing
    - Text mining
    - Document indexing
    - BI/analytics

Facebook aggregates data from many sources into Hadoop clusters. This allows them to do various kinds of processing on that data. But Facebook wanted many of their employees to be able to analyze the data using tools that were familiar to them. Some of the data processing applications at Facebook include:
• Log processing
• Text mining
• Document indexing
• Business intelligences and analytics

For example, in order to recommend possible friends that you should connect with, Facebook must analyze the pattern of connections amongst Facebook users.

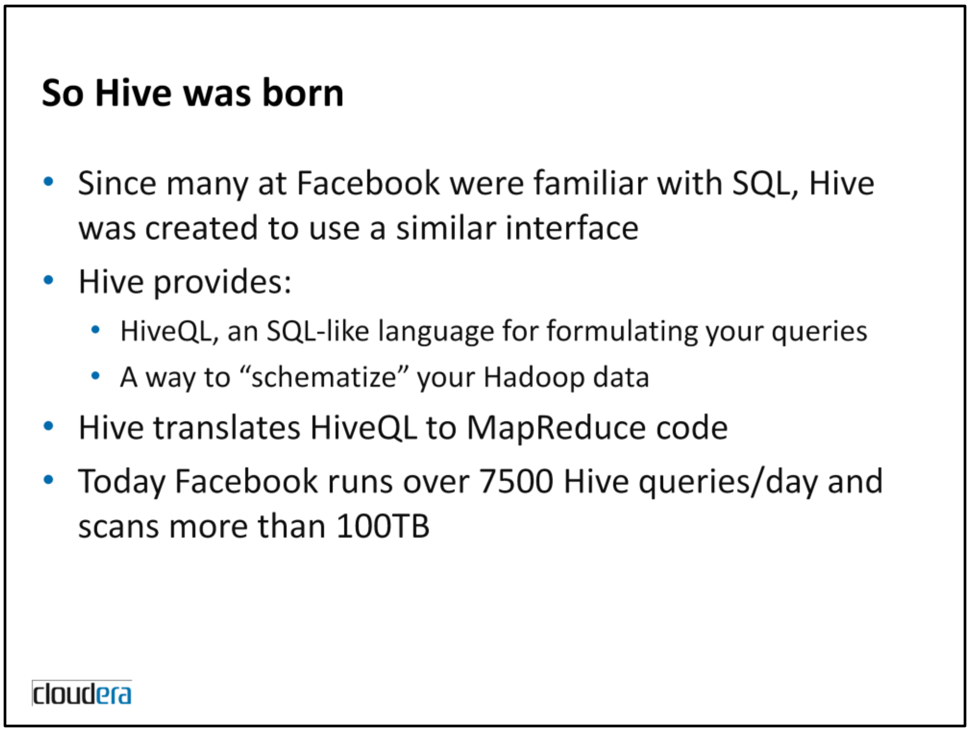


Hive's history continued

- Hadoop was an obvious option
  - But MapReduce was not
- MapReduce lacked:
  - A high-level, analyst-friendly way of querying data in Hadoop
  - A "schema" for the data

Facebook decided to create a tool that allowed people to quickly and easily leverage Hadoop without the effort of writing Java MapReduce (or streaming).

Furthermore, HDFS provides a facility for storing data, but does not provide any mechanisms for defining the structure of that data (e.g., what do the rows look like?). Facebook wanted a way of separating the parsing logic from the analysis logic.



So Hive was born

- Since many at Facebook were familiar with SQL, Hive was created to use a similar interface
- Hive provides:
  - HiveQL, an SQL-like language for formulating your queries
  - A way to "schematize" your Hadoop data
- Hive translates HiveQL to MapReduce code
- Today Facebook runs over 7500 Hive queries/day and scans more than 100TB

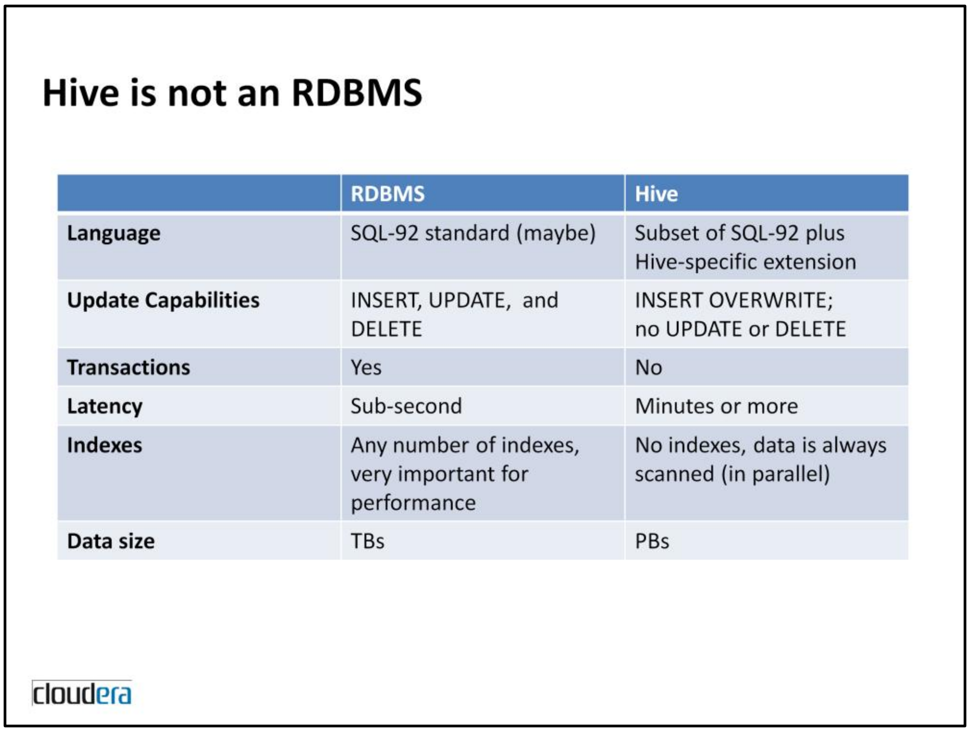
cloudera

Facebook had many employees that were familiar with SQL, so a logical choice was an SQL-like interface to query Hadoop. SQL provides many constructs such as filters (WHERE), aggregates (GROUP BY) and sorting (ORDER BY). Facebook wanted to use this capability to answer questions and analyze their data.

Hive translates this Hive query language (HiveQL) into a collection of Java MapReduce jobs. Developers can therefore focus on their data analysis problem and not the code to implement a MapReduce job.

According to Facebook engineers, in October of 2009 they had more than 7,500 Hive jobs executed each day. Over 100TB of data is scanned every day*.

* Source: Presentation by Dhruba Borthakur and Zheng Shao at Hadoop World October 2, 2009.



Hive is not an RDBMS

	RDBMS	Hive
Language	SQL-92 standard (maybe)	Subset of SQL-92 plus Hive-specific extension
Update Capabilities	INSERT, UPDATE, and DELETE	INSERT OVERWRITE; no UPDATE or DELETE
Transactions	Yes	No
Latency	Sub-second	Minutes or more
Indexes	Any number of indexes, very important for performance	No indexes, data is always scanned (in parallel)
Data size	TBs	PBs

cloudera

Although Hive was modeled after MySQL's shell and SQL variant, do not confuse Hive with a relational database. As stated earlier, Hive is providing an SQL translator to Hadoop's MapReduce paradigm. Therefore the design characteristics of Hadoop still apply. Some differences include:

Language: Hive provides a subset of the SQL-92 language plus several Hive-specific extensions designed to leverage Hadoop.

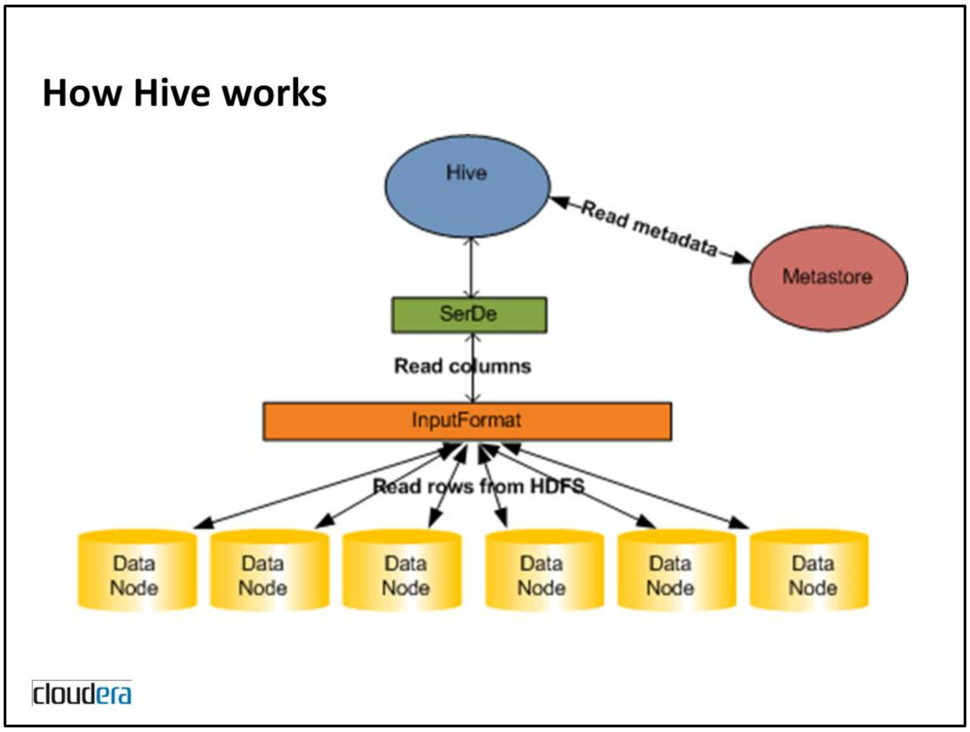
Update Capabilities: Hive is a higher-level language on top of HDFS, which does not provide random write capabilities. At this time, it is only possible to add data or to remove whole files. In order to selectively update or delete parts of a file, the data must be read, transformed and stored into a new table.

Transactions: Hive does not offer transactions, though loading data into a table is an atomic operation.

Latency: In a relational database, queries often return in under a second. However, Hadoop's distributed nature, reliability and scalability lend it to be high latency. A typical MapReduce job will run in no less than 30 seconds and could take minutes or hours.

Indexes: Hive does not support indexes. Instead, all data from a table or partition is scanned in parallel and processed.

Data size: However, compared to an RDBMS, Hadoop can process much larger volumes of data.

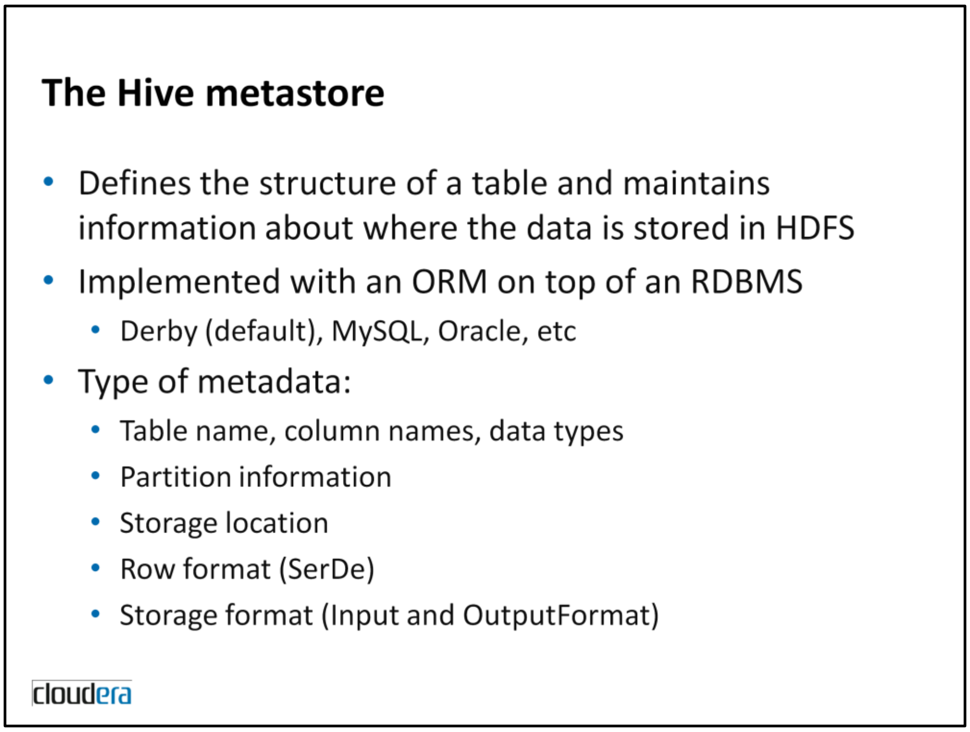


How Hive works

HDFS just stores files, with no concept of what the data looks like inside the files. Hadoop uses a class called an InputFormat to specify how to read rows from a file. For example, the data could be stored as text with newlines separating each line in the file. Or, the data may be key/value pairs of binary data. Hive uses an appropriate InputFormat, such as TextInputFormat or SequenceFileInputFormat, to read records out of the HDFS files.

In addition to reading rows, Hive needs to reads the columns. Hive uses a SerDe for this. SerDe stands for Serializer/Deserializer. For example, the column values may be comma separated within each row.

Finally, Hive also needs to store the metadata for its tables somewhere: the Hive metastore.



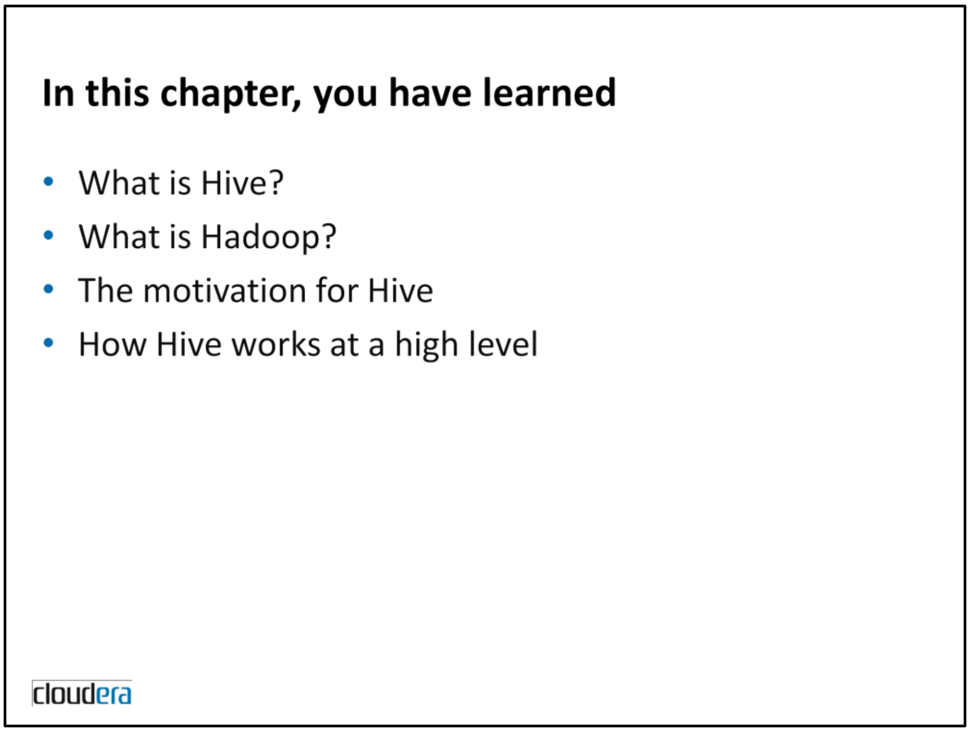
The Hive metastore

- Defines the structure of a table and maintains information about where the data is stored in HDFS
- Implemented with an ORM on top of an RDBMS
  - Derby (default), MySQL, Oracle, etc
- Type of metadata:
  - Table name, column names, data types
  - Partition information
  - Storage location
  - Row format (SerDe)
  - Storage format (Input and OutputFormat)

cloudera

The Hive metastore contains metadata about the Hive tables. This includes table and column names, column types, partition information, the storage location of the data in HDFS, the row format (SerDe) and storage format (Input and OutputFormat).

Hive uses an ORM (object-relational mapping) on top an RDBMS (relational database management system) such as Derby, MySQL, PostgreSQL or Oracle. By default this metastore is an embedded Derby database running in the same process as the Hive client. This is useful for testing, but not practical when multiple users are using Hive. In that case it is necessary to configure a centralized database server. This is covered later in the course.



In this chapter, you have learned

- What is Hive?
- What is Hadoop?
- The motivation for Hive
- How Hive works at a high level