Lab #1 - VM setup

http://tiny.cloudera.com/StrataLab1

Lab #2 - Create a movies dataset

http://tiny.cloudera.com/StrataLab2

cloudera®

Strata+Hadoop World San Jose 2015 Building an Apache Hadoop Data Application

Ryan Blue, Joey Echeverria, Tom White

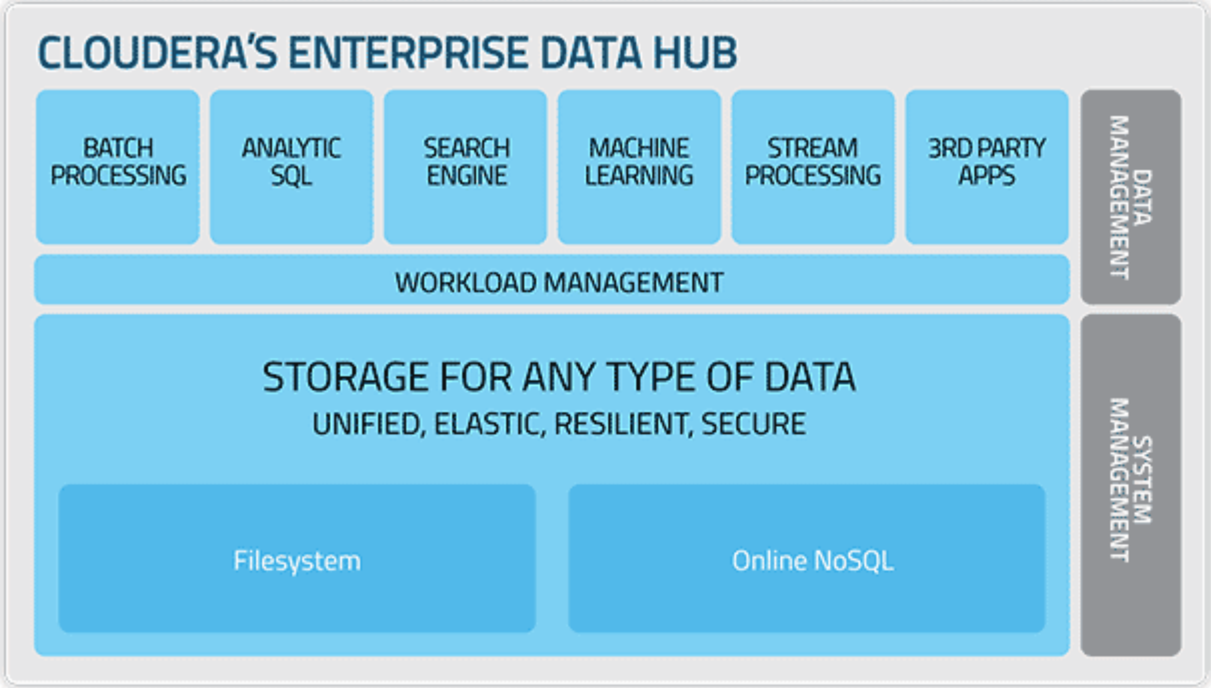
Content for today’s tutorial

- The Hadoop Ecosystem
- Storage on Hadoop
- Movie ratings app: Data ingest
- Movie ratings app: Data analysis

cloudera®

The Hadoop Ecosystem

A Hadoop Stack



cloudera

Processing frameworks

- Code: MapReduce, Crunch, Spark, Tez
- SQL: Hive, Impala, Phoenix, Trafodian, Drill, Presto
- Tuples: Cascading, Pig
- Streaming: Spark streaming (micro-batch), Storm, Samza

Coding frameworks

- **Crunch**
  - A layer around MR (or Spark) that simplifies writing pipelines
- **Spark**
  - A completely new framework for processing pipelines
  - Takes advantage of memory, runs a DAG without extra map phases
- **Tez**
  - DAG-based, like Spark’s execution engine without user-level API

SQL on Hadoop

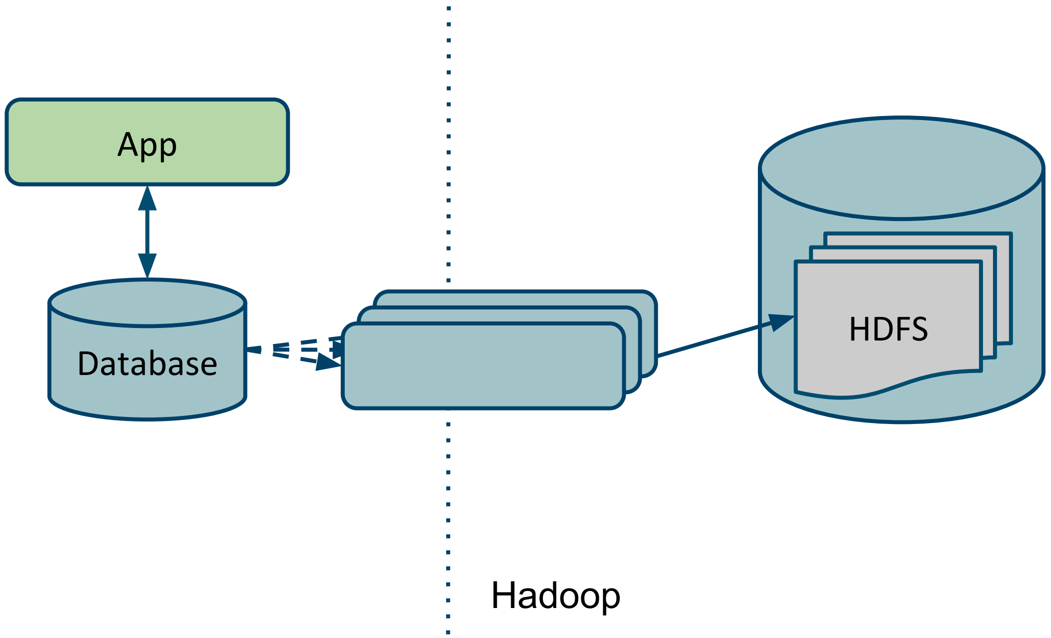
- **Hive** for batch processing
- **Impala** for low-latency queries
- **Phoenix** and **Trafodion** for transactional queries on HBase

Ingest tools

- Relational: Sqoop, Sqoop2
- Record channel: Kafka, Flume
- Files: NiFi
- Numerous commercial options

Ingest tools

- **Relational**: Sqoop, Sqoop2
- Record channel: Kafka, Flume
- Files: NiFi

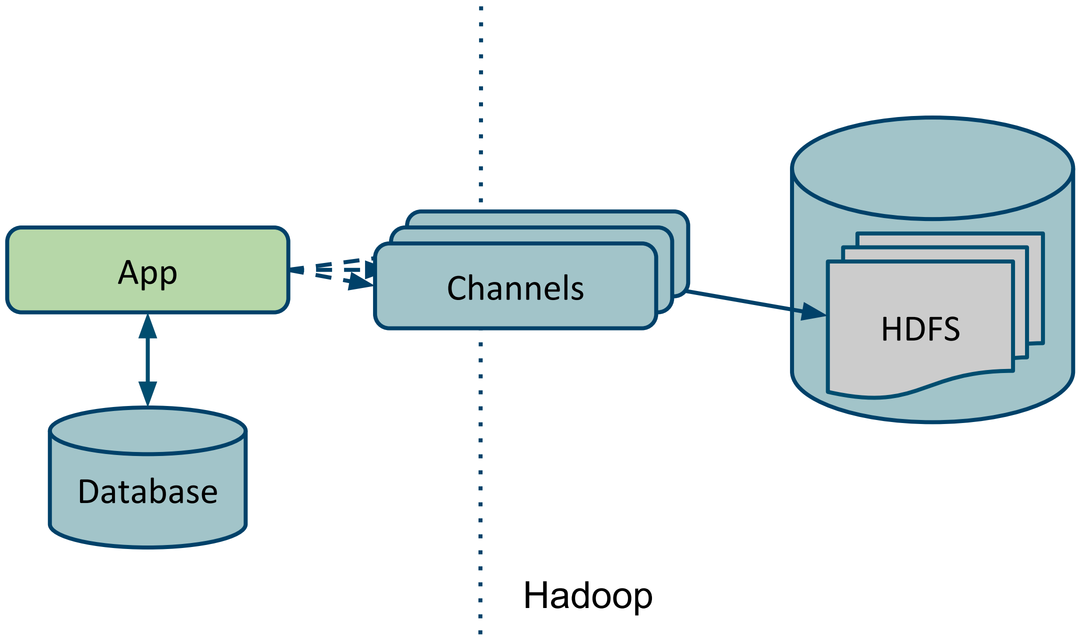


Relational DB to Hadoop

- **Sqoop**
  - CLI to run MR-based import jobs
- **Sqoop2**
  - Fixes configuration problems with Sqoop with credentials service
  - More flexible to run on non-MR frameworks
  - New and under active development

Ingest tools

- Relational: Sqoop, Sqoop2
- **Record channel**: Kafka, Flume
- Files: NiFi



Record streams to Hadoop

- **Flume** - source, channel, sink architecture
  - Well-established and integrated with other tools
  - No order guarantee, duplicates are possible

- **Kafka** - pub-sub model for low latencies
  - Partitioned, provides ordering guarantees, easier to eliminate duplicates
  - More resilient to node failure with consumer groups

Files to Hadoop

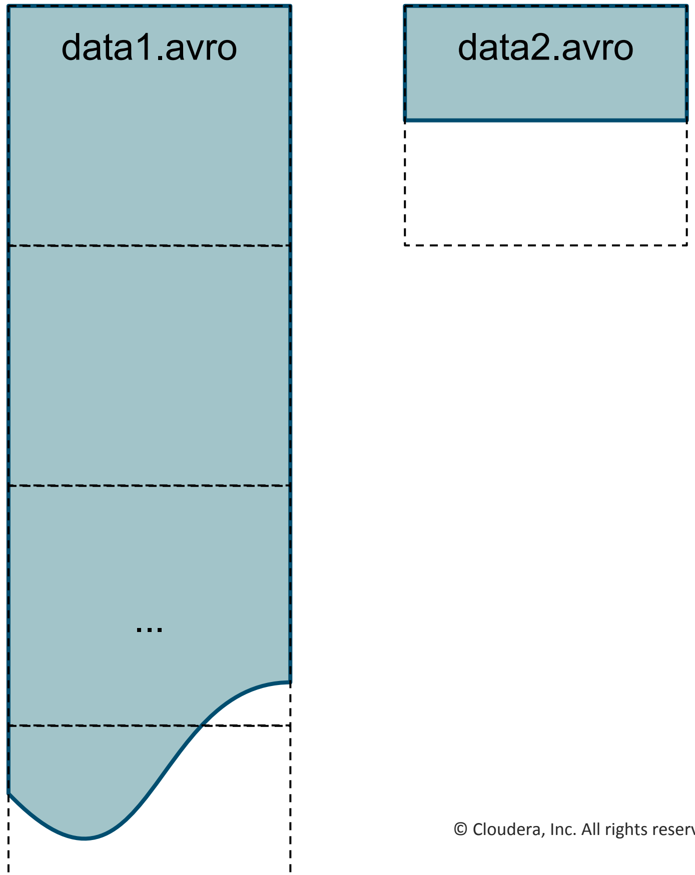
- **NiFi**
  - Web GUI for drag & drop configuration of a data flow
  - Enterprise features: back-pressure, monitoring, provenance, etc.
  - Integration to and from spool directory, HTTP, FTP, SFTP, and HDFS
  - New to the Apache Incubator (but widely deployed privately)
  - First Apache release in January

cloudera®

Data storage in Hadoop

HDFS Blocks

- Blocks
  - Increase parallelism
  - Balance work
  - Replicated
- Configured by `dfs.blocksize`
  - Client-side setting



Splittable File Formats

- Splittable: Able to process part of a file
  - Process blocks in parallel

- Avro is splittable
- Gzipped content is not splittable
- CSV is *effectively* not splittable

File formats

- Existing formats: XML, JSON, Protobuf, Thrift
- Designed for Hadoop: SequenceFile, RCFile, ORC
- Makes me sad: Delimited text
- **Recommended**: Avro or Parquet

Avro

- Recommended row-oriented format
  - Broken into blocks with sync markers for splitting
  - Binary encoding with block-level compression
- Avro schema
  - Required to read any binary-encoded data!
  - Written in the file header
- Flexible object models

Avro in-memory object models

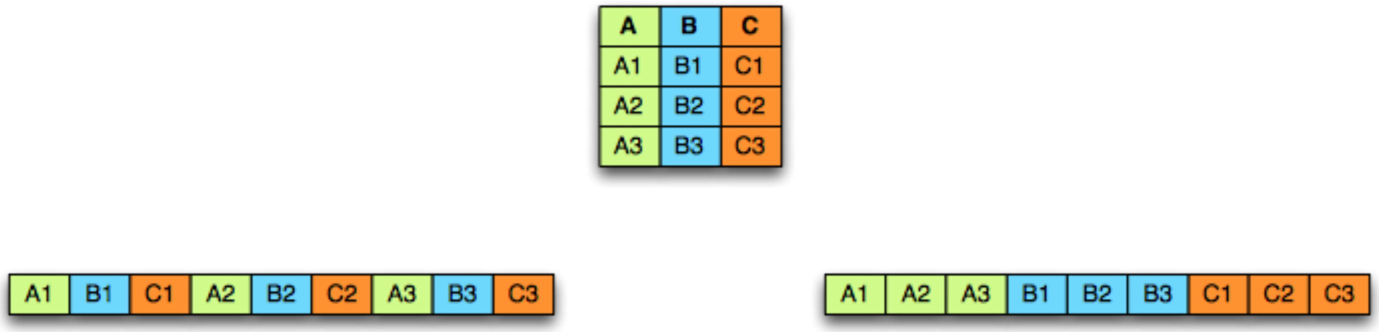
- **generic**
  - Object model that can be used with any schema
- **specific** - compile schema to java object
  - Generates type-safe runtime objects
- **reflect** - java object to schema
  - Uses existing classes and objects

Lab #3 - Using avro-tools

http://tiny.cloudera.com/StrataLab3

cloudera

Row- and column-oriented formats



- Able to reduce I/O when projecting columns
- Better encoding and compression

Images © Twitter, Inc.
https://blog.twitter.com/2013/dremel-made-simple-with-parquet

Parquet

- Recommended column-oriented format
  - Splittable by organizing into row groups
  - Efficient binary encoding, supports compression
- Uses other object models
  - Record construction API rather than object model
  - **parquet-avro** - Use Avro schemas with generic or specific records
  - parquet-protobuf, parquet-thrift, parquet-hive, etc.

cloudera

Parquet trade-offs

- Rows are buffered into groups that target a final size
- Row group size
  - Memory consumption grows with row group size
  - Larger groups get more I/O benefit and better encoding
- Memory consumption grows for *each open file*

cloudera

Lab #4 - Using parquet-tools

http://tiny.cloudera.com/StrataLab4

cloudera

Partitioning

- Splittable file formats aren’t enough
- Not processing data is better than processing in parallel
- Organize data to avoid processing: **Partitioning**
- Use HDFS paths for a coarse index: `data/y=2015/m=03/d=14/`

Partitioning Caution

- Partitioning in HDFS is the primary index to data
  - Should reflect the most common access pattern
  - Test partition strategies for multiple workloads
- Should balance file size with workload
  - Lots of small files are bad for HDFS - partitioning should be more coarse
  - Larger files take longer to find data - partitioning should be more specific

cloudera

Implementing partitioning

- Build your own - *not recommended*
- Hive and Impala managed
  - Partitions are treated as data columns
  - Insert statements must include partition calculations
- Kite managed
  - Partition strategy configuration file
  - Compatible with Hive and Impala

Kite

- High-level data API for Hadoop
  - Built around datasets, not files
  - Tasks like partitioning are done internally
- Tools built around the data API
  - Command-line
  - Integration in Flume, Sqoop, NiFi, etc.

Lab #5 - Create a partitioned dataset

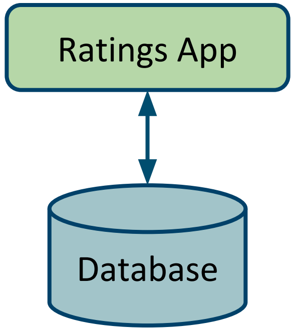
http://tiny.cloudera.com/StrataLab5

cloudera

Movie ratings app: Data ingest pipeline

Movie ratings scenario

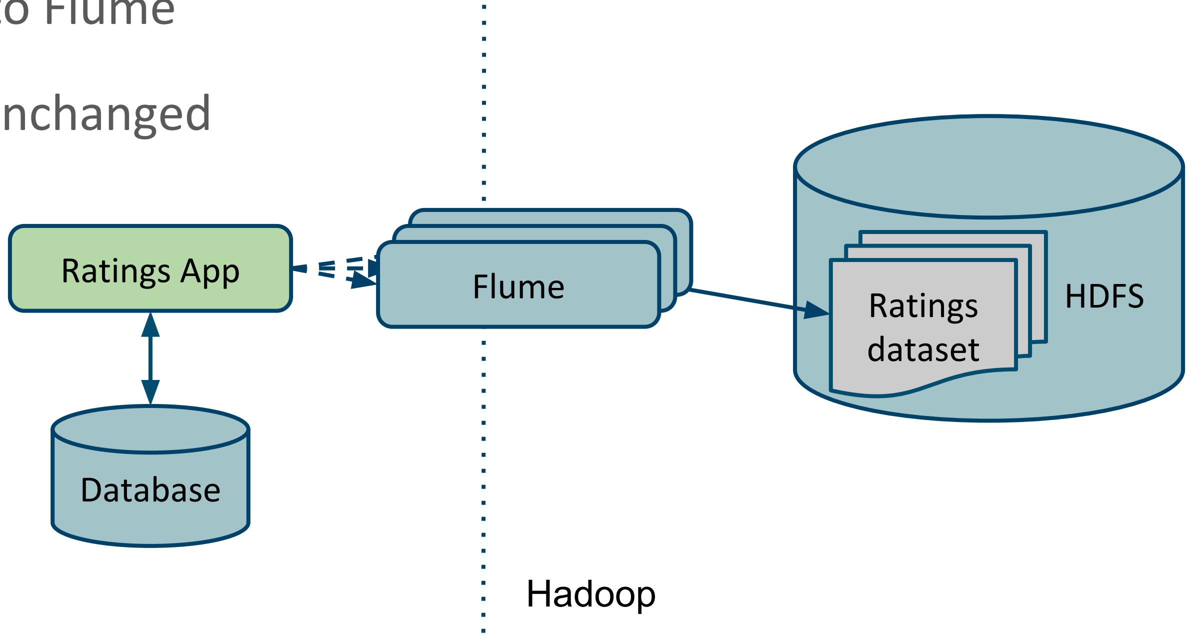
- Your company runs a web application where users can rate movies
- You want to use Hadoop to analyze ratings over time
  - Avoid scraping the production database for changes
  - Instead, you want to log every rating submitted



cloudera

Movie ratings app

- Log ratings to Flume
- Otherwise unchanged



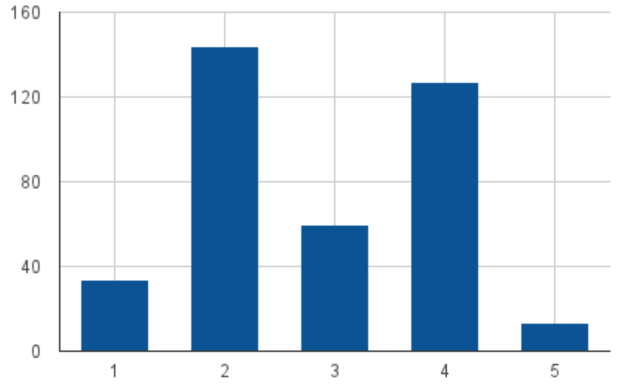
Lab #6 - Create a Flume pipeline

http://tiny.cloudera.com/StrataLab6

Movie ratings app: Analyzing ratings data

Movie ratings analysis

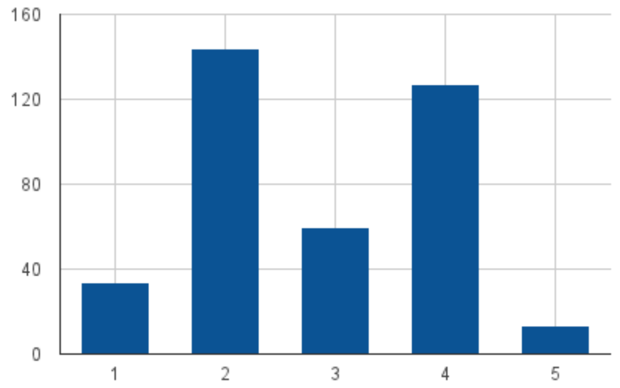
- Now you have several months of data
- You can query it in Hive and Impala for most cases
- Some questions are difficult to formulate as SQL
  - Are there any movies that people either love or hate?



cloudera

Analyzing ratings

- Map
  - Extract key, movie_id, and value, rating
- Reduce:
  - Reduce groups all of the ratings by movie_id
  - Count the number of ratings for each movie
  - If there are two peaks, output the movie_id and counts
  - Peak detection: difference between counts goes from negative to positive



cloudera

Crunch background

- Stack up functions until a group-by operation to make a map phase
- Similarly, stack up functions after a group-by to make a reduce phase
- Additional group-by operations set up more MR rounds automatically

```java
PTable<Long, Double> table = collection
    .by(new GetMovieID(), Avros.longs())
    .mapValues(new GetRating(), Avros.ints())
    .groupByKey()
    .mapValues(new AverageRating(), Avros.doubles());
```

Lab #7 - Analyze ratings with Crunch

http://tiny.cloudera.com/StrataLab7

cloudera

Thank you

blue@cloudera.com
joey@scalingdata.com
tom@cloudera.com