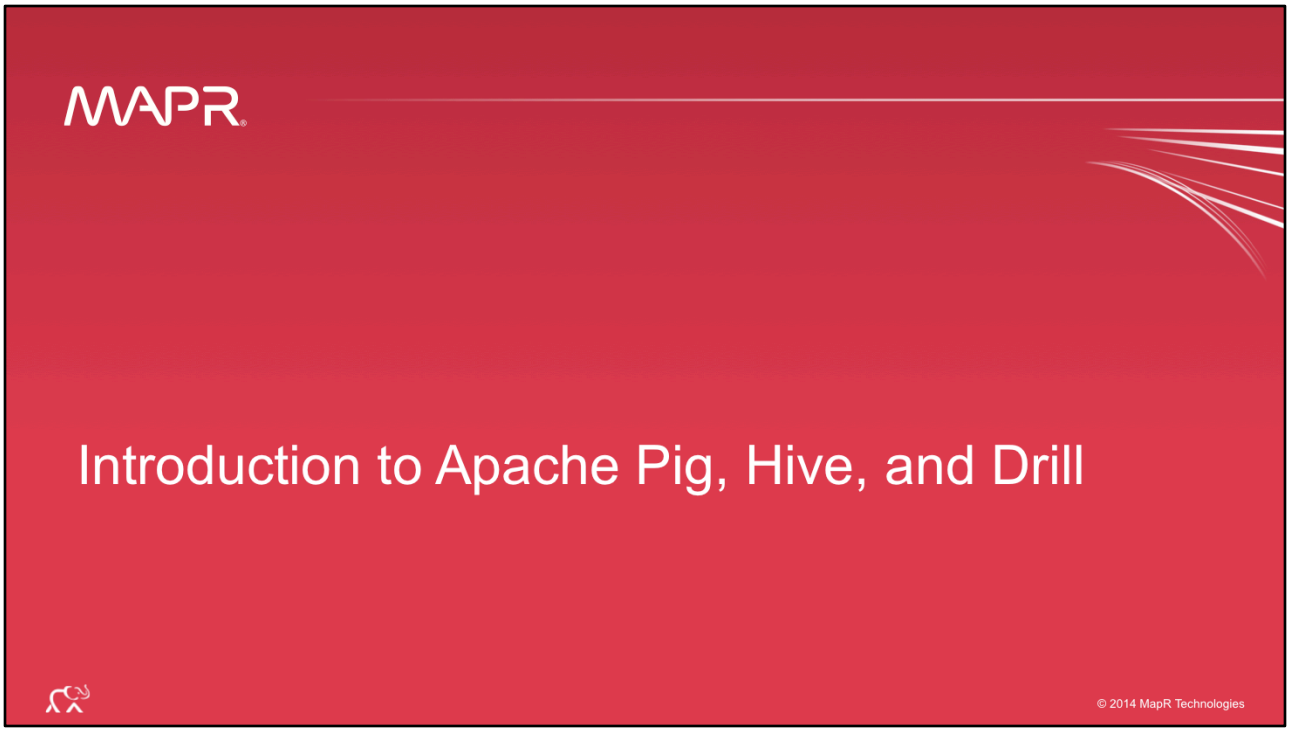
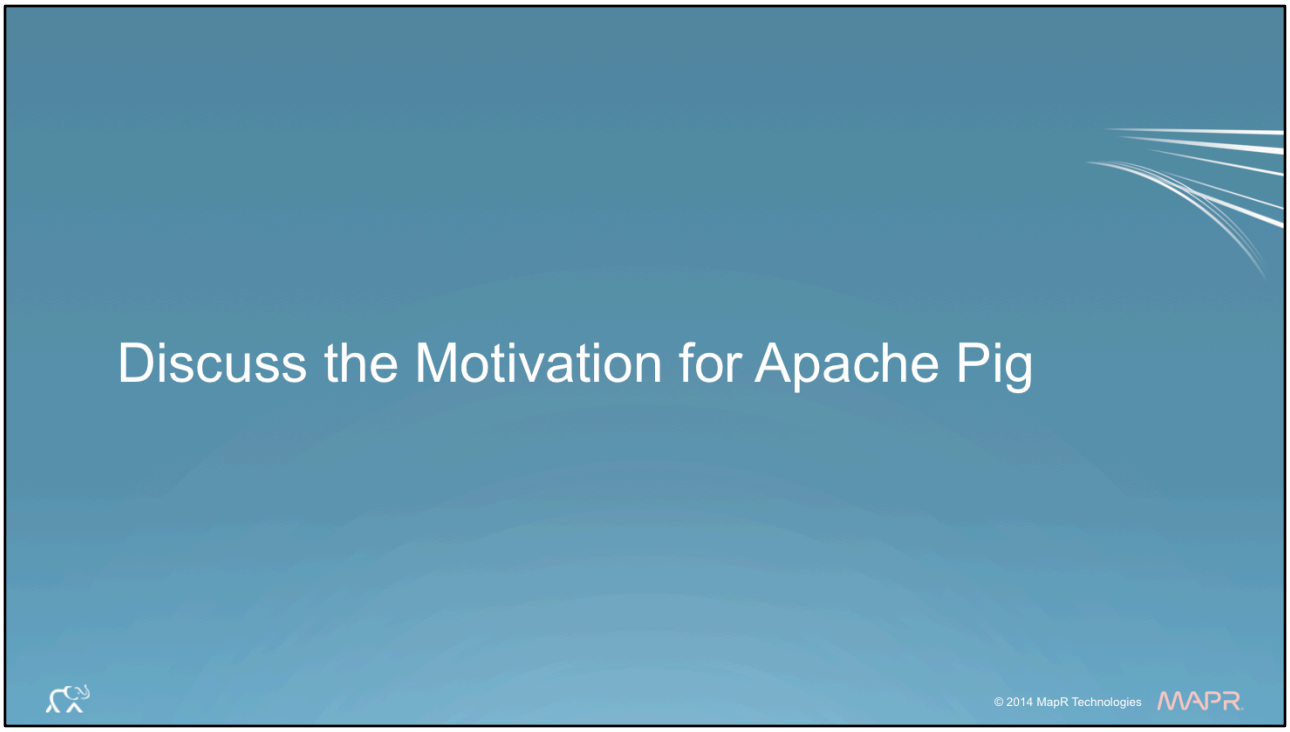
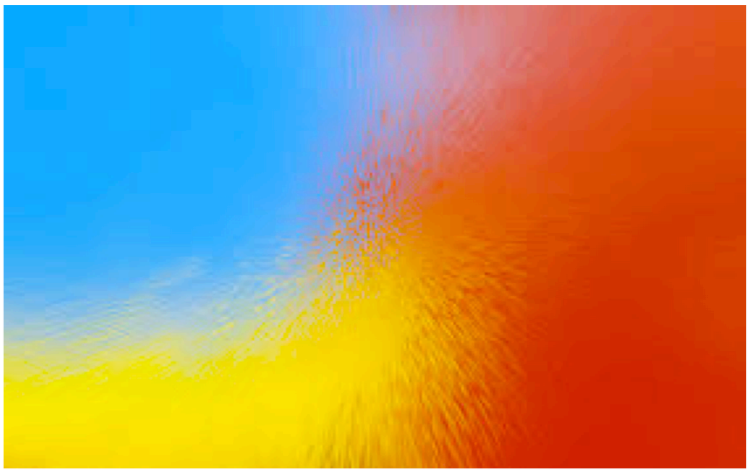


This lesson provides a very brief introduction to Apache Pig, Hive, and Drill.



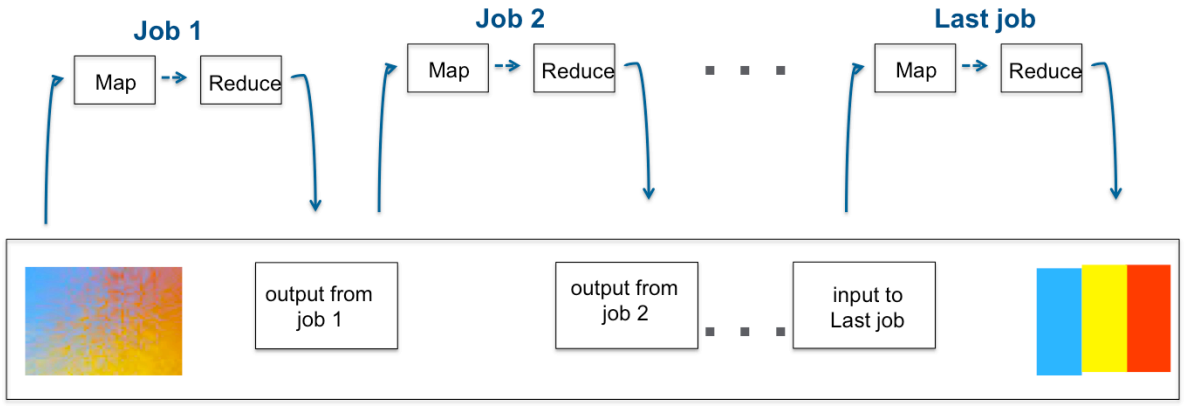
This section provides a brief motivation for Apache Pig.

Here's a Bunch of (unstructured) Data



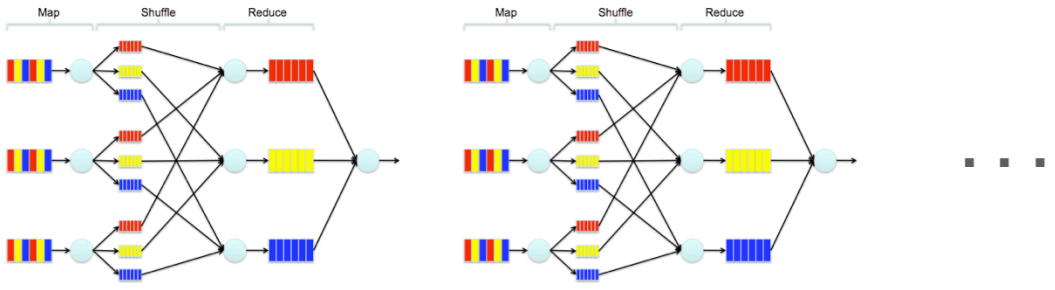
Data comes in lots of shapes and sizes. Suppose you have a lot of unstructured (non-tabular) data that you wish to analyze. You can't use any SQL-oriented tools because the data lacks a well-defined schema.

You Can Use MapReduce to Analyze the Data



You could potentially use MapReduce to analyze this data. Often time, MapReduce solutions require multiple jobs to be run in a series, where the output from the first job becomes input to the second job (and so on).

But MapReduce is Too Complex for Some



Some people find MapReduce too cumbersome and wieldy for certain problems. It's a difficult paradigm to understand in the first place.

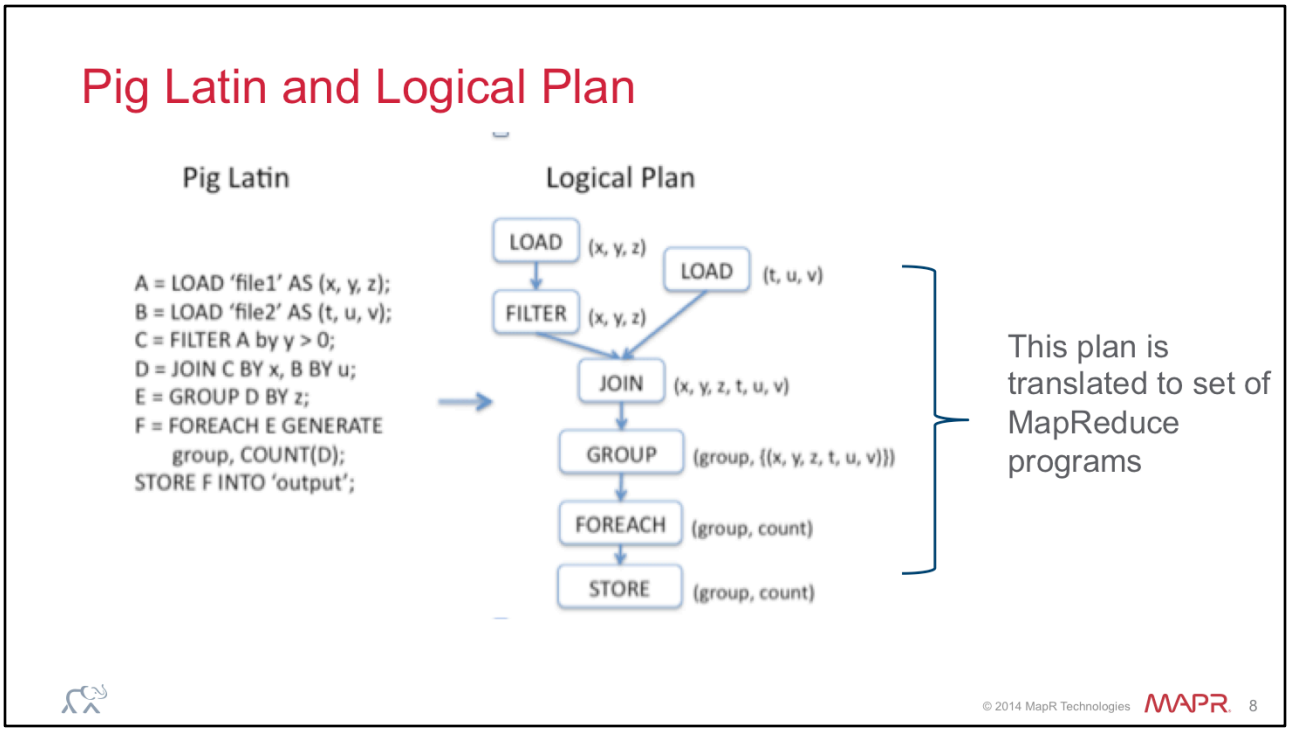
This section discusses how to use Pig.

What is Apache Pig?

- High-level data flow scripting language (Pig Latin)
- Interpreted and compiled into MapReduce (runtime engine)
- Easy, optimizable, and extensible (user-defined functions)
- Great for unstructured data (optional schema on read)

Apache Pig is a data flow scripting language called “Pig Latin”. That is to say, each statement in a Pig script transforms data in some way. Pig Latin is compiled and run as Java MapReduce programs. The language provides a rich set of operations, but you can write and consume your own user-defined functions.

Apache Pig is ideal for unstructured data.



The graphic above depicts how a Pig script gets translated into a logical plan and then ultimately executed as a series of MapReduce programs. There's a 1:1 relationship between lines of Pig script that perform data transformation and the MapReduce program that is built to execute that logic.

Use Cases for Pig

Use Case	Example
ETL data pipeline	Import log files, clean/join/transform data, precompute aggregates and load into EDW
Data analysis	Ad-hoc query support on unstructured and/or nested data
Incremental processing	Join data sets once daily, then join incremental changes in data over the course of the day

Most frequent use case for Pig is data pipeline. A common example is a Web company importing logs from Web servers, cleaning/transforming/joining data before uploading into data warehouse.

SQL is used traditionally to support ad-hoc queries to quickly form a question of the data. Raw data lacks structure (and therefore schema) which precludes using SQL tools to do these queries. Researchers who can use Python or Perl on single machines of smaller data sets enjoy using Pig on large hadoop clusters to support their ad-hoc queries.

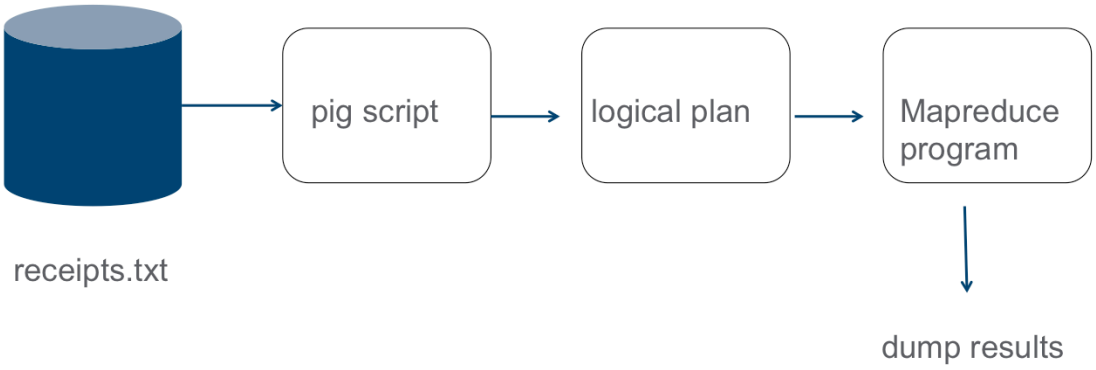
References for Pig

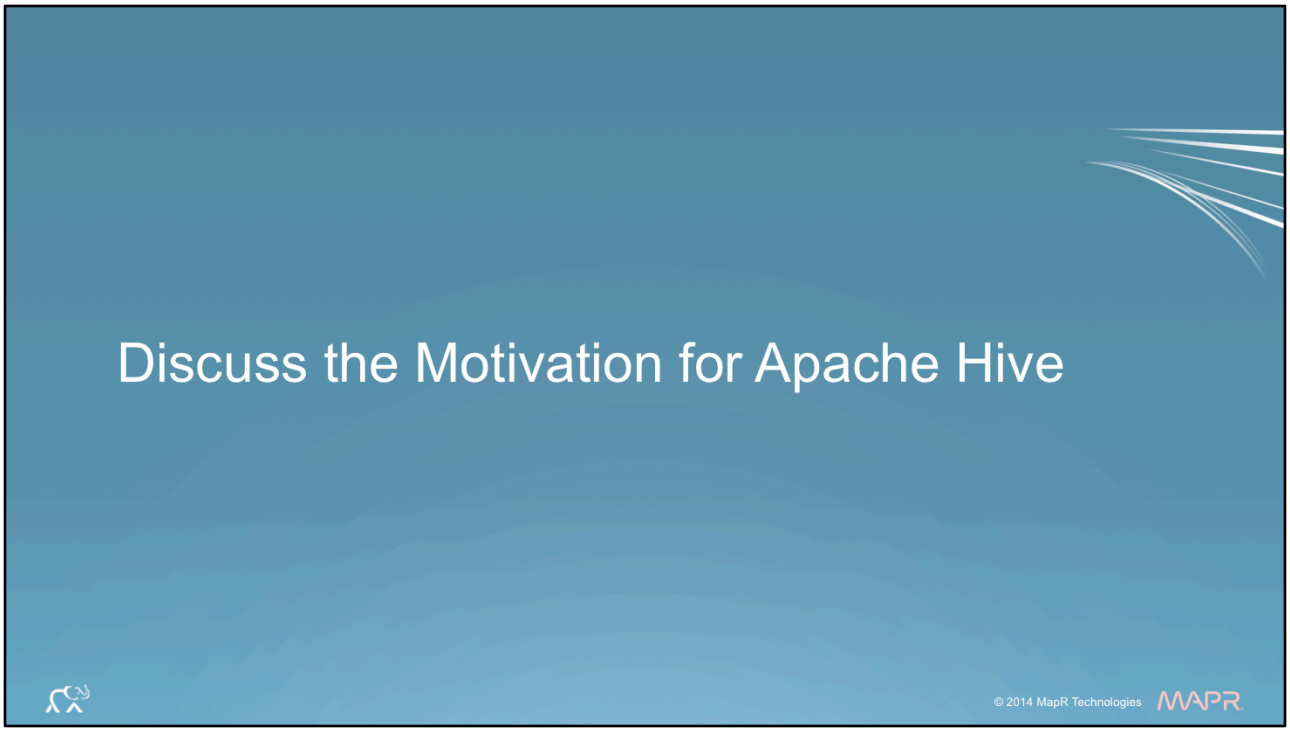
- http://doc.mapr.com/display/MapR/Pig
- http://pig.apache.org
- Programming Pig (O'Reilly)

The slide above identifies some references to get you started using Apache Pig. There are many more resources available for you on Pig, though. Just google it!

Perform a Demonstration for Pig

Demo: Using Pig for Aggregating Data





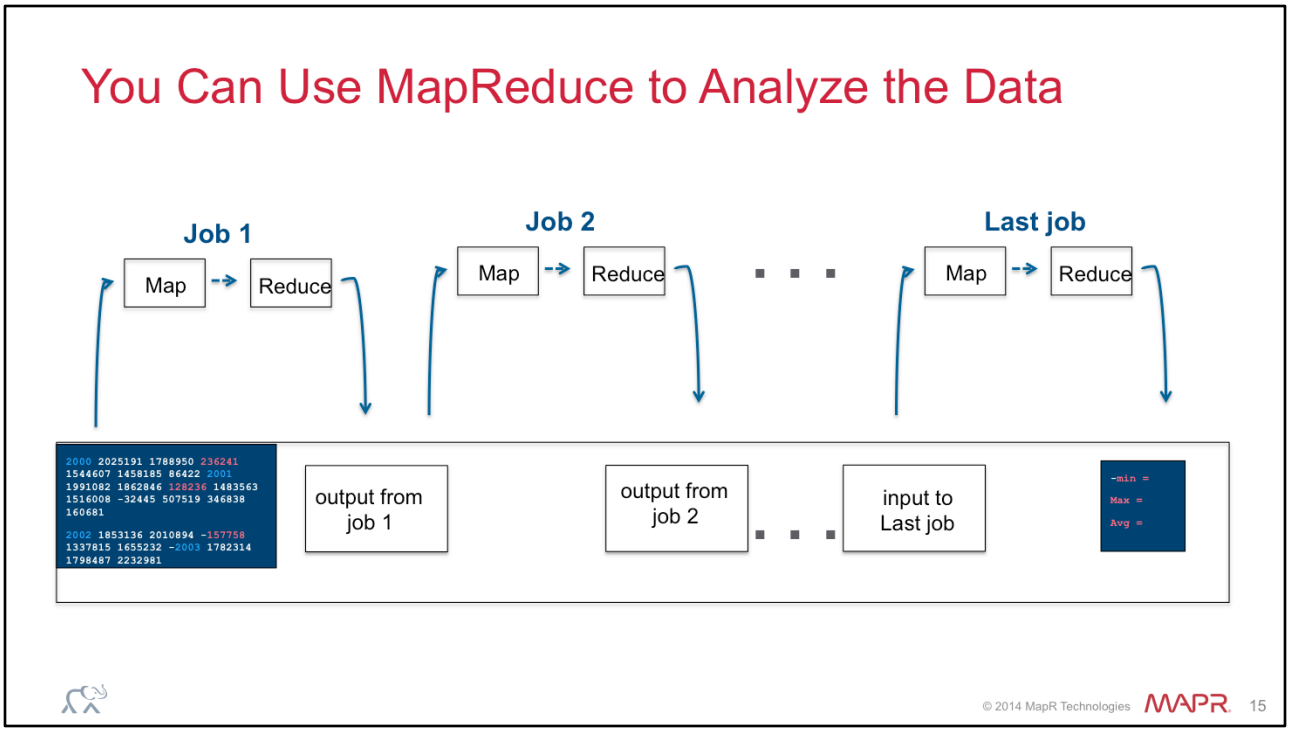
This section discusses a brief motivation for Apache Hive.

Here's a Bunch of (Structured) Data

```
2000 2025191 1788950 236241 1544607 1458185 86422 480584 330765 149819
2001 1991082 1862846 128236 1483563 1516008 -32445 507519 346838 160681
2002 1853136 2010894 -157758 1337815 1655232 -317417 515321 355662 159659
2003 1782314 2159899 -377585 1258472 1796890 -538418 523842 363009 160833
2004 1880114 2292841 -412727 1345369 1913330 -567961 534745 379511 155234
2005 2153611 2471957 -318346 1576135 2069746 -493611 577476 402211 175265
2006 2406869 2655050 -248181 1798487 2232981 -434494 608382 422069 186313
2007 2567985 2728686 -160701 1932896 2275049 -342153 635089 453637 181452
2008 2523991 2982544 -458553 1865945 2507793 -641848 658046 474751 183295
2009 2104989 3517677 -1412688 1450980 3000661 -1549681 654009 517016 136993
2010 2162706 3457079 -1294373 1531019 2902397 -1371378 631687 554682 77005
2011 2303466 3603059 -1299593 1737678 3104453 -1366775 565788 498606 67182
2012 2450164 3537127 -1086963 1880663 3029539 -1148876 569501 507588 61913
```

 MAPR 14

Data comes in lots of shapes and sizes. The data shown in the slide above appears structured, which means there may be an explicit schema associated with the data when it is both read and written. You wish to analyze this data on your Hadoop cluster.



The core mechanism for analyzing data is using MapReduce. Usually, MapReduce jobs must be written as a series of jobs (where each job performs a small set of transformations or calculations). The output from the first job becomes input to the second job, and so on.

But MapReduce is Too Complex for Some

```
 public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
for (Text value: values) {
      compositeString = value.toString();
      compositeStringArray = compositeString.split("_");
      tempYear = new Text(compositeStringArray[0]);
      tempValue = new Long(compositeStringArray[1]).longValue();
      if(tempValue < min) {
         min=tempValue;
         minYear=tempYear;
      }
   }
   Text keyText = new Text("min" + "(" + minYear.toString() + "): ");
   context.write(keyText, new FloatWritable(min));
 }
```

 MAPR 16

MapReduce is too complex for some people who lack programming skills. Also, it may be overkill, especially if there is inherent structure (schema) within the data.

This section briefly describes how to use Apache Hive.

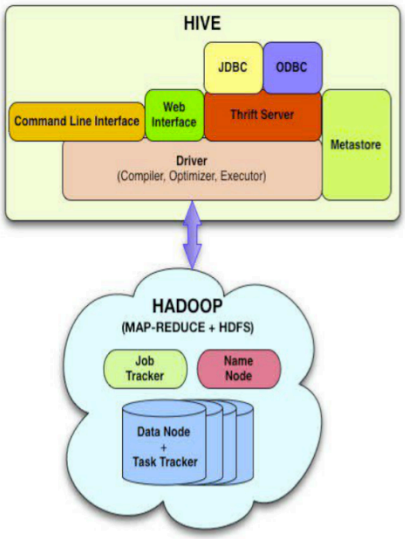
What is Hive?

- Client-server abstraction
- SQL-like language (HiveQL)
- Interpreted and compiled into MapReduce (runtime engine)
- Easy, optimizable, and extensible (user-defined functions)
- Great for structured data (schemas stored in metastore)

Hive is a client-server abstraction for MapReduce that provides SQL-like syntax to analysts on Hadoop data. The language (called HiveQL) provides a small set of SQL (mostly DDL) operations that get transformed into MapReduce programs. The language may be extended to include user-defined functionality.

Hive is ideal for structured data.

Hive Architecture



The graphic above depicts the Hive architecture. Note the rich set of client interfaces to Hive (ODBC, JDBC, REST, CLI), making Hadoop data extremely accessible to external applications for analysis. The driver compiles, optimizes, and executes the HiveQL queries as MapReduce programs on Hadoop. Schema data applied to HDFS files is stored in the metastore.

Deployment Types for Hive Metastore

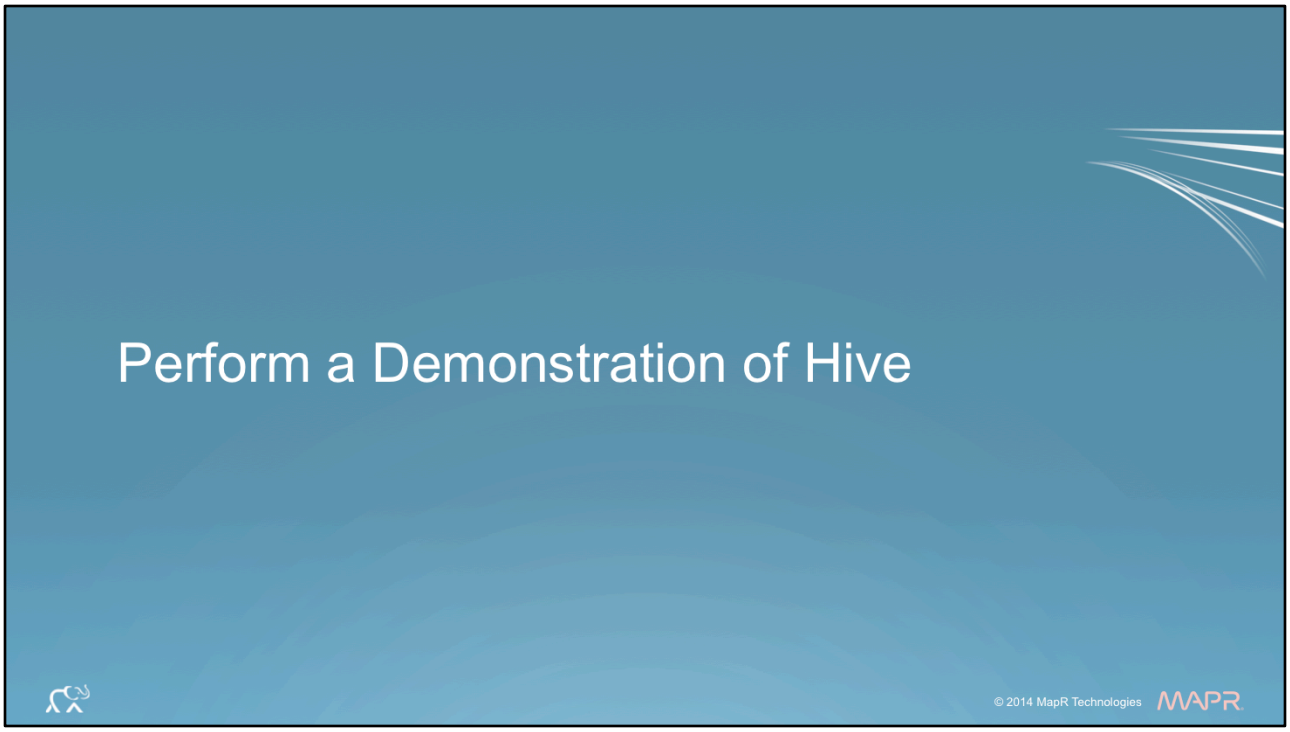
Use Case	Example
Embedded	Metadata stored in Apache Derby database embedded in metastore server
Local	Metadata stored in a RDBMS database (e.g. MySQL) running within the cluster
Remote	Metadata stored in a RDBMS database (e.g. Oracle) running outside the cluster

There are 3 ways to deploy the Hive metastore: embedded, local, and remote. The embedded method uses an Apache Derby database that the metastore accesses using Derby application libraries. The local method uses a RDBMS system running within the cluster and uses a connector to read to and write from it. For example, if you use MySQL in the cluster, you can use the JDBC connector for MySQL to read from and write to the metastore. The remote method uses a RDBMS system running outside the cluster and uses a connector to read to and write from it. For example, if you Oracle outside the cluster, you can use the ODBC connector for Oracle to read from and write to the metastore.

References for Hive

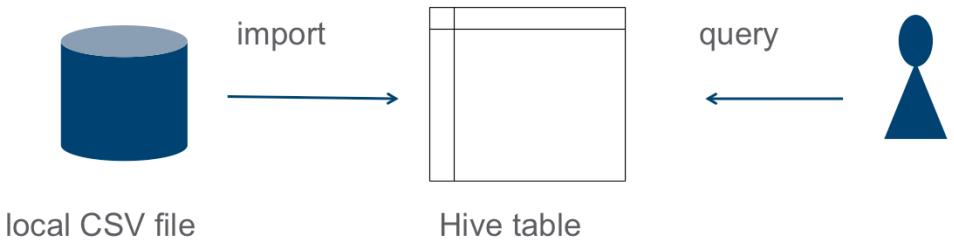
- http://doc.mapr.com/display/MapR/Hive
- http://hive.apache.org
- Programming Hive (O'Reilly)

The slide above identifies some references to get you started using Apache Hive. Of course, there are many more resources available to you on Hive. Just google it!



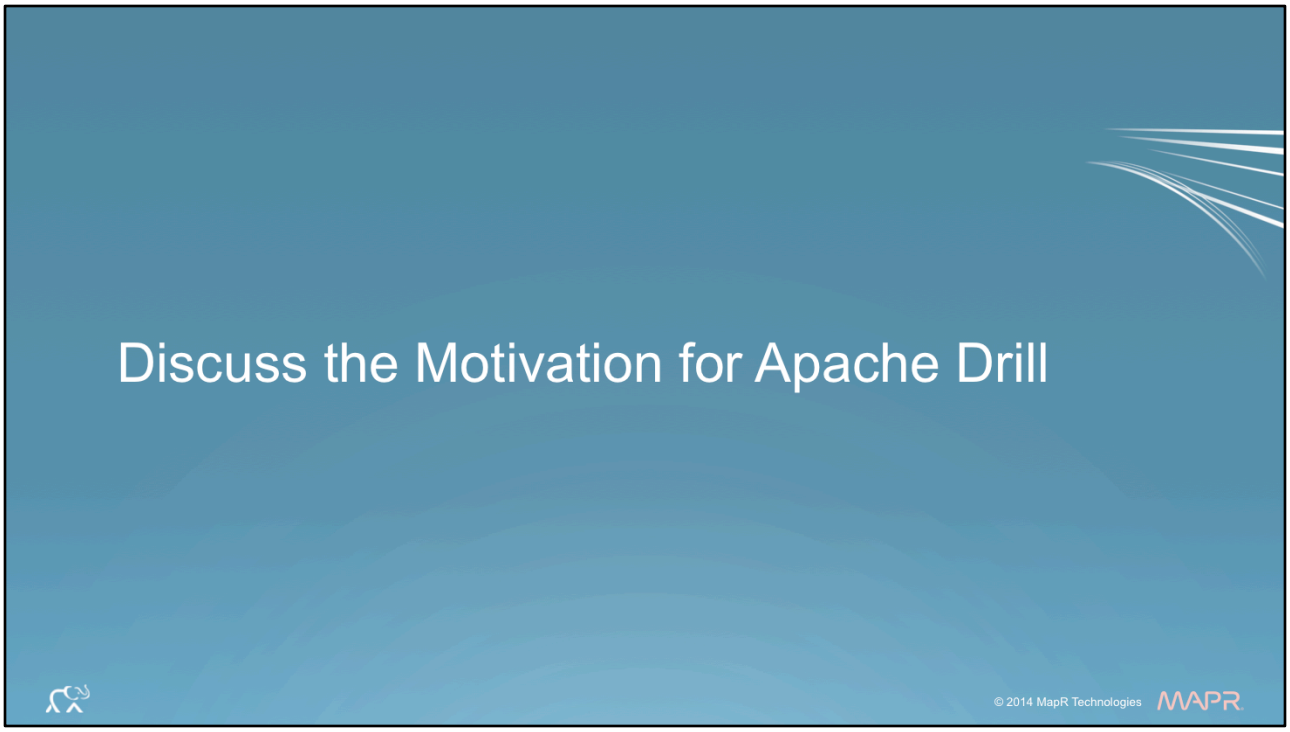
This section performs a brief demonstration of Hive.

Demo: Hive Script for Calculating Min/Max/Mean

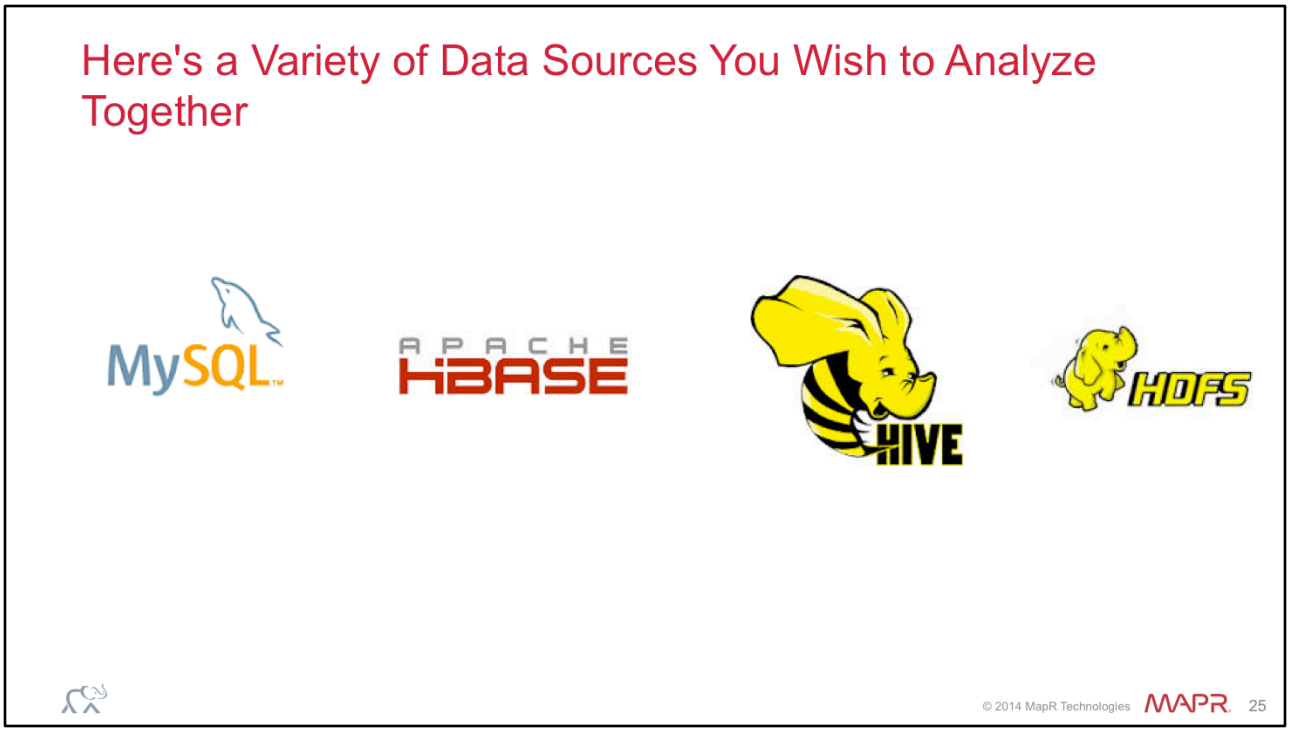


The script in the slide above is written in HiveQL. The script performs the following:

1. Creates a table in the metastore with a specific schema
2. Loads data from local file system into HDFS applying the schema
3. Find maximum value and year associated for “delta”
4. Find minimum value and year associated for “delta”
5. Find average (mean) “delta”



This section motivates the need for Apache Drill.



The slide above depicts various data sources that you may wish to analyze at the same time. Each of these data sources has their own interface and set of tools that you can use independently, but how would you analyze all these data sources together in the same query?

And Here's Your Favorite Tool

The de facto tool for data analysts is SQL (standard query language). It's an official language published by ANSII. There are countless data analysis and business intelligence tools built using SQL.

But SQL Doesn't Work on All Those Data Sources

- SQL doesn't work on Hbase
- SQL doesn't work on HDFS
- Hive is not ANSI SQL compliant
- SQL can't join RDBMS with Hadoop data sources

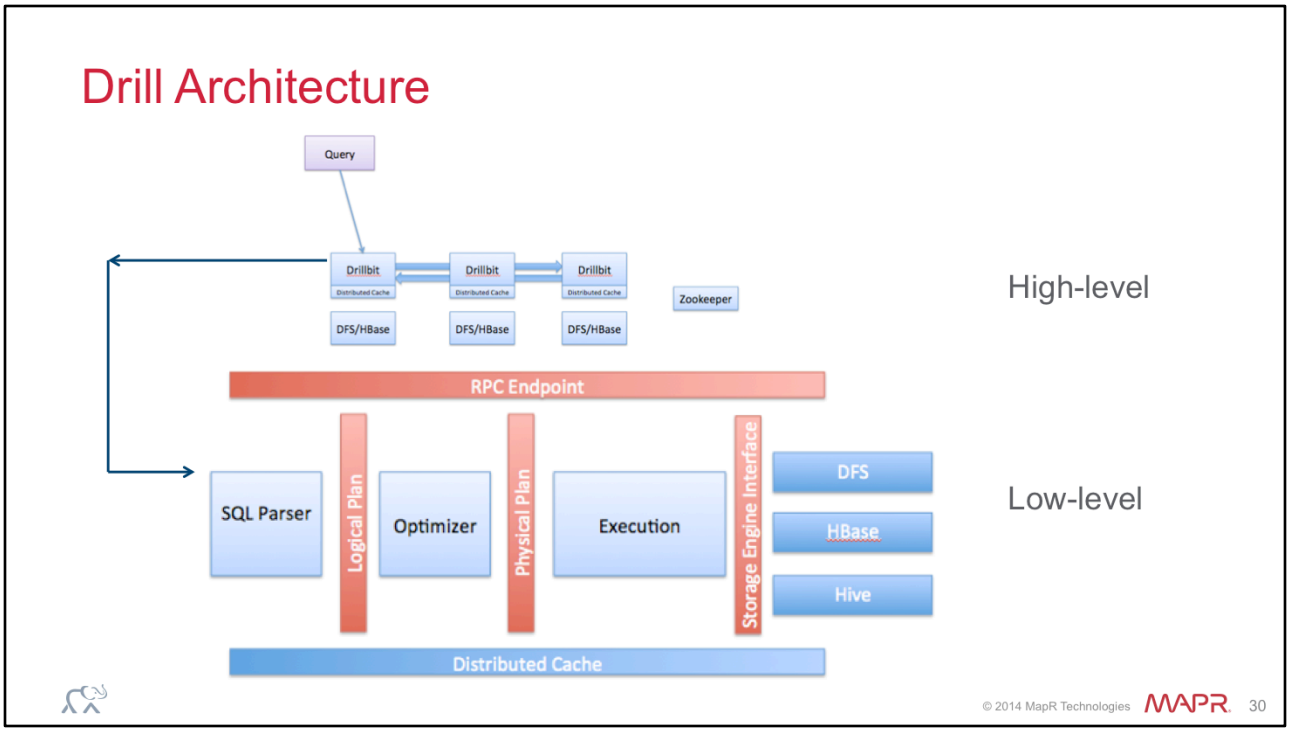
The problem with using SQL to analyze all this data is that it doesn't work across all these data sources. SQL doesn't work on Hbase or HDFS. HiveQL is only a proper subset of functionality provided by ANSII SQL. Even if SQL was supported on all your data sources, you can't join RDBMS with Hadoop data sources (or even Hadoop data sources with each other).

This section briefly describes how to use Apache Drill.

What is Drill?

- Low-latency distributed query engine for large-scale data sets
- Supports structured, unstructured, and nested data sources
- Apache open-source / extensible code
- Does NOT require Hadoop / only requires Zookeeper
- Inspired by Google Dremel

Drill is a low-latency query engine since all the data is loaded into RAM. It distributes queries across and outside the Hadoop cluster to external data sources. Drill supports structured data (e.g. RDBMS), semi-structured data sources (e.g. Hbase), and unstructured data sources (e.g. HDFS). Drill was inpired by Google Dremel which was created for the same purpose. Drill does not even require Hadoop. The only requirement for Drill as a zookeeper service.



The Drill high-level and low-level architectures are described in the graphic above. Shown in the high-level architecture, the user issues a query to drill which is interpreted by a “drill bit”. Drill bits run across a cluster and share their data using a distributed cache mechanism. Shown in the low-level architecture is that the drill bit will forward the query to the SQL parser to parse out the query and make a logical plan. The optimizer takes the logical plan and converts it to a physical plan. The execution engine executes the physical plan, accessing the appropriate data sources invoked in the query.

Use Cases for Drill

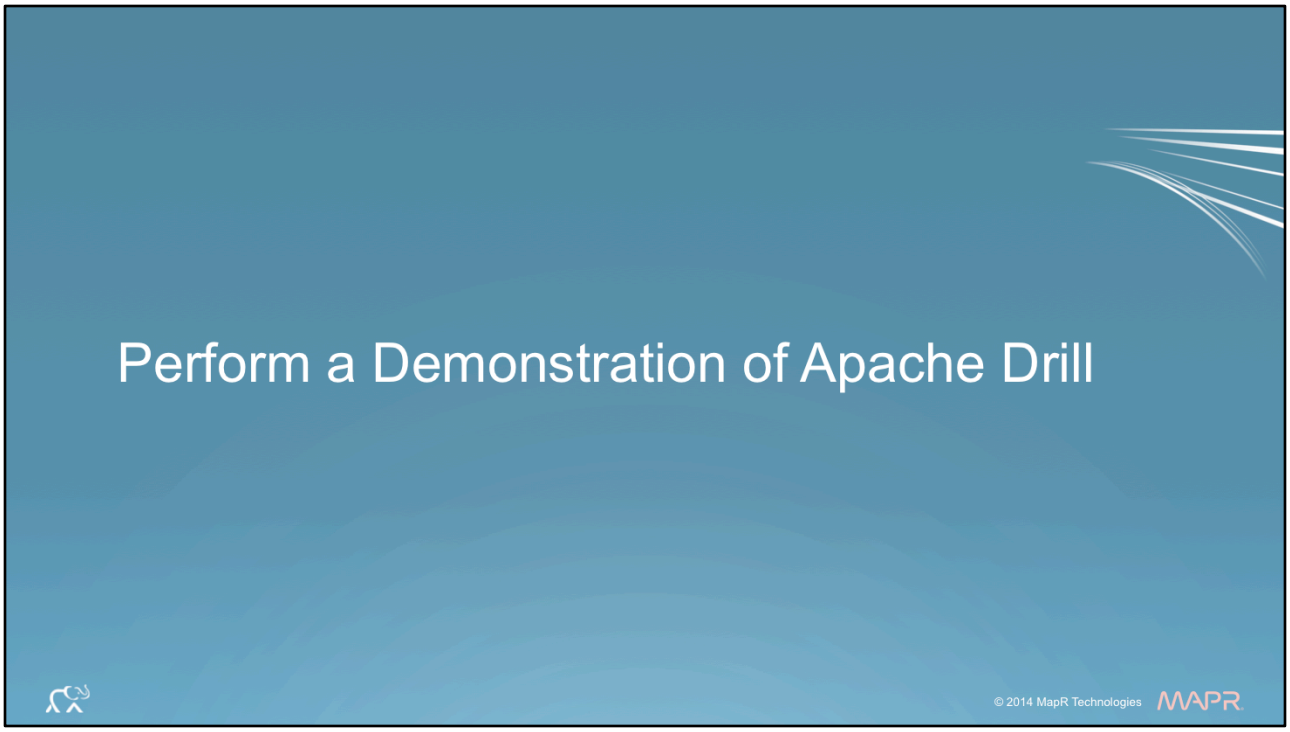
Use Case	Example
Data warehouse consolidation	Break down silos of data to enable streamlined access
Query embedded data	Convert / interpret fields within a data source on the fly
Join Hadoop and RDBMS	Interface with Hadoop (HDFS, Hbase, and Hive) and RDBMS data sources in same SQL query

The table above summarizes the common use cases anticipated for Apache Drill. The primary use case is consolidating multiple data warehouses to enable and streamline access. These data sources can include, but aren't limited to, HDFS, Hbase, Hive, and external data sources such as an RDBMS. Drill can also be used to interpret data embedded in data source fields on the fly.

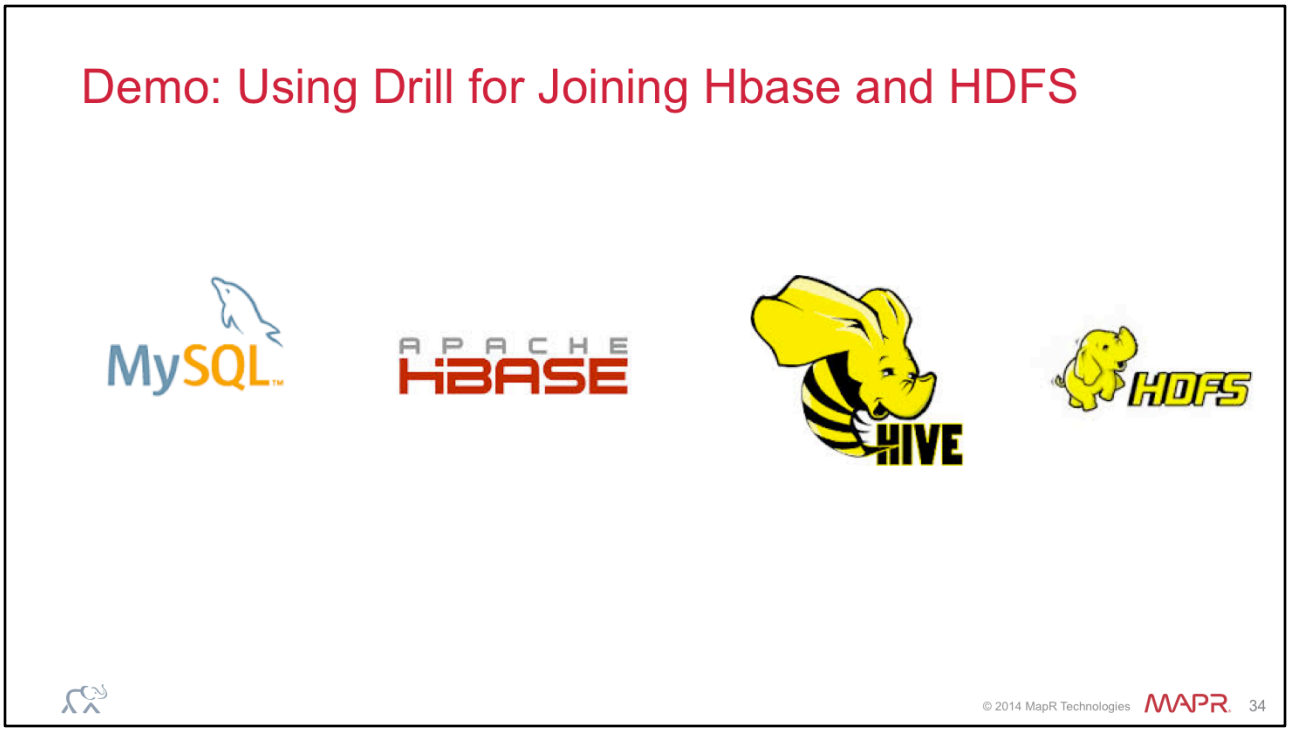
References for Drill

- http://doc.mapr.com/display/MapR/Drill
- https://cwiki.apache.org/confluence/display/DRILL/Apache+Drill+Wiki
- http://incubator.apache.org/drill/

The slide above identifies available resources for you to start using Apache Drill. There aren't many more resources available on Drill since it is a new Apache project. However, you are still encouraged to google it!



This section performs a simple demonstration of Apache Drill.



This demonstration will join Hbase and HDFS data sources in a single query.