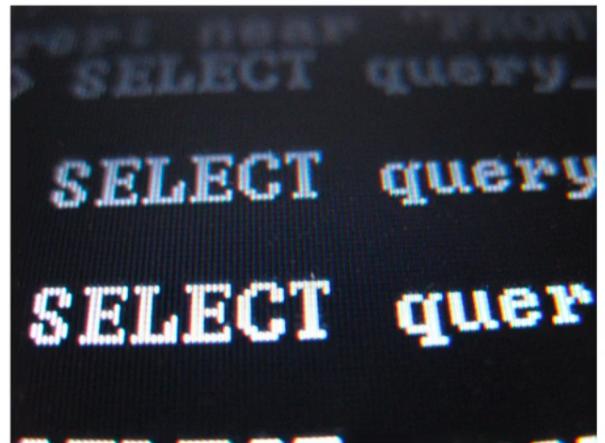


# Hive

---

## Why Hive?

- Uses familiar SQL syntax (HiveQL)
- Interactive
- Scalable – works with “big data” on a cluster
  - Really most appropriate for data warehouse applications
- Easy OLAP queries – WAY easier than writing MapReduce in Java
- Highly optimized
- Highly extensible
  - User defined functions
  - Thrift server
  - JDBC / ODBC driver



## The Hive Metastore

- Hive maintains a “metastore” that imparts a structure you define on the unstructured data that is stored on HDFS etc.

```
CREATE TABLE ratings (
    userID INT,
    movieID      INT,
    rating INT,
    time   INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '${env:HOME}/ml-100k/u.data'
OVERWRITE INTO TABLE ratings;
```

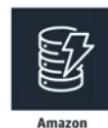
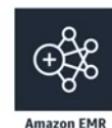
# External Hive Metastores

- Metastore is stored in MySQL on the master node by default
- External metastores offer better resiliency / integration
  - AWS Glue Data Catalog
  - Amazon RDS



## Other Hive / AWS integration points

- Load table partitions from S3
- Write tables in S3
- Load scripts from S3
- DynamoDB as an external table

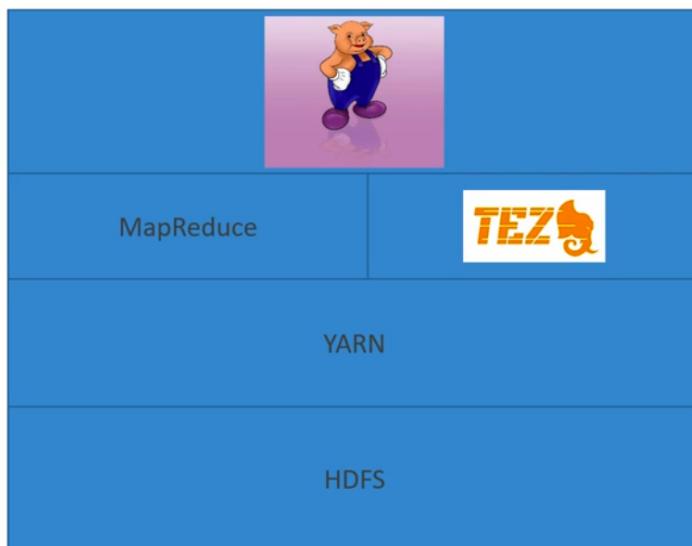


# Apache Pig

- Writing mappers and reducers by hand takes a long time.
- Pig introduces *Pig Latin*, a scripting language that lets you use SQL-like syntax to define your map and reduce steps.
- Highly extensible with user-defined functions (UDF's)



## How Pig Works



```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);
metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate:chararray, videoRelease:chararray, imbdLink:chararray);

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
    ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;

ratingsByMovie = GROUP ratings BY movieID;
avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID, AVG(ratings.rating) AS avgRating;
fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;
fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;
oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;
DUMP oldestFiveStarMovies;
```

# Pig / AWS Integration

- Ability to use multiple file systems (not just HDFS)
  - i.e., query data in S3
- Load JAR's and scripts from S3



Amazon S3

## HBase

- Non-relational, petabyte-scale database
- Based on Google's BigTable, on top of HDFS
- In-memory
- Hive integration



# Sounds a lot like DynamoDB

- Both are NoSQL databases intended for the same sorts of things
- But if you're all-in with AWS anyhow, DynamoDB has advantages
  - Fully managed (auto-scaling)
  - More integration with other AWS services
  - Glue integration
- HBase has some advantages though:
  - Efficient storage of sparse data
  - Appropriate for high frequency counters (consistent reads & writes)
  - High write & update throughput
  - More integration with Hadoop

## HBase / AWS integration

- Can store data (StoreFiles and metadata) on S3 via EMRFS
- Can back up to S3



presto sql query about all servers are peta byte of storage

# Presto

- It can connect to many different “big data” databases and data stores at once, and query across them
- **Interactive queries at petabyte scale**
- Familiar SQL syntax
- Optimized for OLAP – analytical queries, data warehousing
- Developed, and still partially maintained by Facebook
- This is what Amazon Athena uses under the hood
- Exposes JDBC, Command-Line, and Tableau interfaces



## Presto connectors

- HDFS
- S3
- Cassandra
- MongoDB
- HBase
- SQL
- Redshift
- Teradata



- If you're familiar with iPython notebooks – it's like that
  - Lets you interactively run scripts / code against your data
  - Can interleave with nicely formatted notes
  - Can share notebooks with others on your cluster
- Spark, Python, JDBC, HBase, Elasticsearch + more

## Zeppelin + Spark

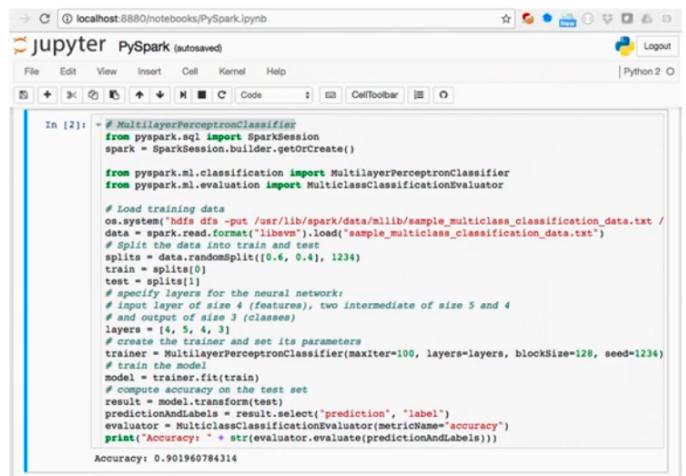
- Can run Spark code interactively (like you can in the Spark shell)
  - This speeds up your development cycle
  - And allows easy experimentation and exploration of your big data
- Can execute SQL queries directly against SparkSQL
- Query results may be visualized in charts and graphs
- Makes Spark feel more like a data science tool!



amazon offers emr notebook

# EMR Notebook

- Similar concept to Zeppelin, with more AWS integration
- Notebooks backed up to S3
- Provision clusters from the notebook!
- Hosted inside a VPC
- Accessed only via AWS console



A screenshot of a Jupyter Notebook interface titled "PySpark (autosaved)". The code in the cell (In [2]) is as follows:

```
# MultilayerPerceptronClassifier
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Load training data
os.system('hdfs dfs -put /usr/lib/spark/data/mllib/sample_multiclass_classification_data.txt /')
data = spark.read.format("libsvm").load("sample_multiclass_classification_data.txt")

# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# specify layers for the neural network
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [4, 5, 4, 3]
# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

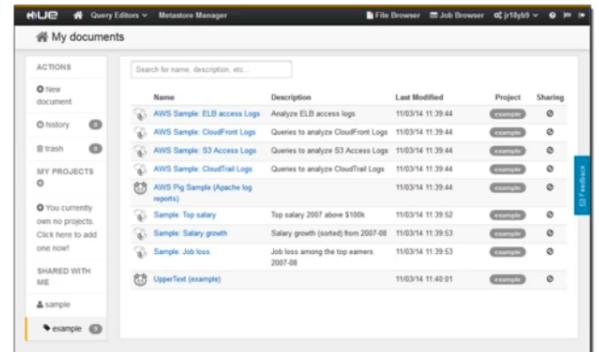
# train the model: model = trainer.fit(train)
model = trainer.fit(train)

# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Accuracy: " + str(evaluator.evaluate(predictionAndLabels)))
```

The output shows the accuracy: Accuracy: 0.901960784314

## Hue

- Hadoop User Experience
- Graphical front-end for applications on your EMR cluster
- IAM integration: Hue Super-users inherit IAM roles
- S3: Can browse & move data between HDFS and S3

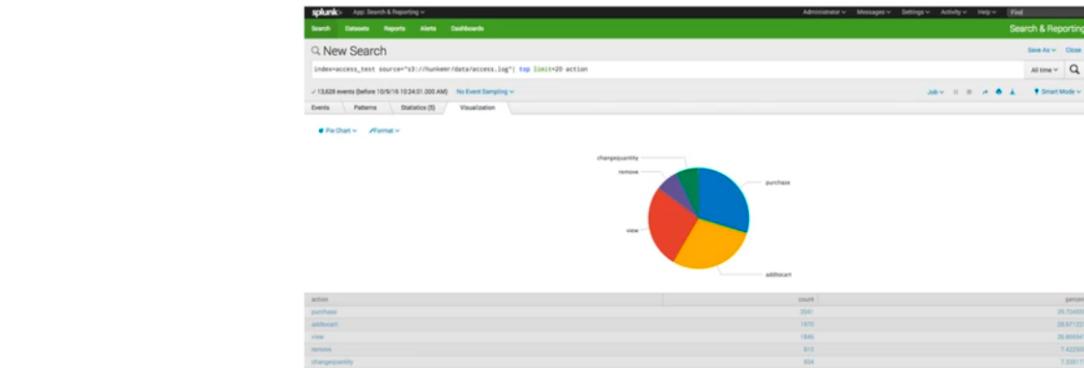


The screenshot shows the Hue interface with the title bar "HUE" and "Query Editors" and "Metastore Manager". The main area is titled "My documents" and contains a table of HDFS files:

ACTIONS	Name	Description	Last Modified	Project	Sharing
New document	AWS Sample: ELB access Logs	Analyze ELB access logs	11/03/14 11:39:44	example	
History	AWS Sample: CloudFront Logs	Queries to analyze CloudFront Logs	11/03/14 11:39:44	example	
Train	AWS Sample: S3 Access Logs	Queries to analyze S3 Access Logs	11/03/14 11:39:44	example	
	AWS Sample: CloudTrail Logs	Queries to analyze CloudTrail Logs	11/03/14 11:39:44	example	
	AWS Pig Sample (Apache log reports)	Top salary 2007 above \$100k	11/03/14 11:39:44	example	
	Sample: Top salary	Top salary 2007 above \$100k	11/03/14 11:39:52	example	
	Sample: Salary growth	Salary growth (surface) from 2007-08	11/03/14 11:39:53	example	
	Sample: Job loss	Job loss among the top earners	2007-08	example	
	UpperText (example)		11/03/14 11:40:01	example	

# Splunk

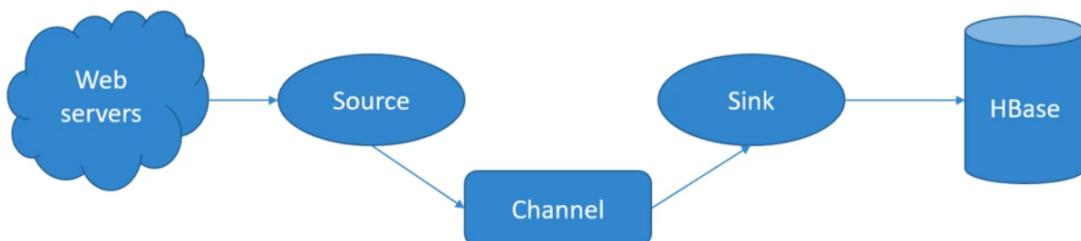
- Splunk / Hunk “makes machine data accessible, usable, and valuable to everyone”
- Operational tool – can be used to visualize EMR and S3 data using your EMR Hadoop cluster.



# Flume



- Another way to stream data into your cluster
- Made from the start with Hadoop in mind
  - Built-in sinks for HDFS and HBase
- Originally made to handle log aggregation



other libraries

# Other EMR / Hadoop Tools

- **Ganglia** (monitoring)
- **Mahout** (machine learning)
- **Accumulo** (another NoSQL database)
- **Sqoop** (relational database connector)
- **HCatalog** (table and storage management for Hive metastore)
- **Kinesis Connector** (directly access Kinesis streams in your scripts)
- **Tachyon** (accelerator for Spark)
- **Derby** (open-source relational DB in Java)
- **Ranger** (data security manager for Hadoop)
- Install whatever you want

## EMR Security

- IAM policies
  - Grant or deny permissions
  - Allow user actions
  - Combine with tagging to control access per cluster
- Kerberos
  - Secure user authentication
- SSH
  - Secure connection to command line
  - Tunneling for web interfaces
  - Can use Kerberos or EC2 key pairs
- IAM roles
  - Control access to EMRFS data based on user, group, location of data
  - Each cluster must have a service role and a role for the EC2 instance profile
  - IAM policies attached to roles
  - Auto-scaling role
  - Service-linked roles



# EMR Security

- “Block public access”
  - Easy way to prevent public access to data stored on your EMR cluster
  - Can set at the account level before creating the cluster.



## EMR: Choosing Instance Types

- Master node:
  - m5.xlarge if < 50 nodes, larger if > 50 nodes
- Core & task nodes:
  - m5.xlarge is usually good
  - If cluster waits a lot on external dependencies (i.e. a web crawler), t2.medium
  - Improved performance: m4.xlarge
  - Computation-intensive applications: high CPU instances
  - Database, memory-caching applications: high memory instances
  - Network / CPU-intensive (NLP, ML) – cluster computer instances
- Spot instances
  - Good choice for task nodes
  - Only use on core & master if you’re testing or very cost-sensitive; you’re risking partial data loss