INTRODUCTION TO SCALABLE TOOLS FOR BIG DATA

Dr. Robert Dyer
WHO IS THIS GUY?
WHAT DO I DO?
About the Boa Language and Infrastructure

Boa is a domain-specific language and infrastructure that eases mining software repositories. Boa's infrastructure leverages distributed computing techniques to execute queries against hundreds of thousands of software projects very efficiently.

Getting Started With Boa in 5 Minutes

An Example Mining Task

Consider answering a question such as "what are the average number of changed files per revision (churn rate) for all projects?"

Answering this question ordinarily requires knowledge of (at a minimum): mining project metadata, mining code repository locations, how to access those code repositories, additional filtering code, controller logic, etc.

```
1 # what are the churn rates for all projects
2 p: Project = input;
3 counts: output mean[string] of int;
```
ENOUGH ABOUT ME...

WHY ARE WE HERE TODAY?
What Happens in an Internet Minute?

- 639,800 GB of global IP data transferred
- 20 New victims of identity theft
- 47,000 App downloads
- 204 million Emails sent
- 83,000 In sales
- 1,300 New mobile users
- 100+ New LinkedIn accounts
- 320+ New Twitter accounts
- 61,141 Hours of music
- 20 million Photo views
- 300 Photo uploads
- 3,000 New tweets
- 277,000 Logins
- 6 million Facebook views
- 2+ million Search queries
- 1.3 million Video views
- 30 Hours of video uploaded
- Today, the number of networked devices = the global population
- By 2015, the number of networked devices = 2x the global population
- In 2015, it would take you 5 years to view all video crossing IP networks each second

And Future Growth is Staggering
Big Data Landscape 2016

Infrastructure
- Hadoop On-Premise
- Hadoop in the Cloud
- Spark
- Cluster Services
- Data Integration
- Data Transformation
- Cloud EDW
- MPP Databases
- Graph Databases
- NoSQL Databases

Analytics
- Analyst Platforms
- Data Science Platforms
- BI Platforms
- Statistical Computing
- Social Analytics
- Speech & NLP
- Horizontal AI

Applications
- Sales & Marketing
- Customer Service
- Human Capital
- Security
- Govt / Regulation
- Life Sciences
- Search

Open Source
- Framework
- Query / Data Flow
- Data Access
- Coordination
- Real-Time
- Stat Tools

Data Sources & APIs
- Health
- Financial & Economic Data
- Air / Space / Sea
- Location/People/Entities
- Other

Incubators & Schools
- © Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark Capital (@firstmarkcap)

Cross-Infrastructure/Analytics
- Management / Monitoring
- Security
- Storage
- App Dev
- Crowdsourcing
- Search
- Data Services
- For Business Analysts
- SMB / Comcast

Industries
- Finance
- Energy
- Aerospace
- Internet of Things
- Space

Visualization
- Solar
- Elasticsearch
- D3
- Map

Connectivity
- Legal
- Everlaw
- Everlaw
- Prenova

Text & Content
- RAVEL
- Everlaw
- Everlaw

Data
- panjiva
- Data Elite
- GA

Data Camp
- INSIGHTS
- insite
- quantopian

The Data Incubator
- DataIncubator
- DataIncubator
Big Data Landscape 2016

Infrastructure
- Hadoop
  - On-Premise: Cloudera, Hortonworks
  - In the Cloud: Amazon
- Spark
- Cluster Services
- Database
  - NoSQL: MongoDB, Couchbase
  - NewSQL: MariaDB, Citadella
- Cloud EDW
- Graph Databases
- MPP Databases

Analytics
- Analyst Platforms
  - Palantir, Ayasdi, Qualys, Enigma
- Analytics Platforms
  - IBM, Microsoft, Tableau
- Data Science Platforms
  - Continuum, DataRobot
- Log Analytics
  - Splunk, Loggly
- Social Analytics
  - NetBase, Datasync

Visualization
- Tableau
- Qlik
- ChartIO

Sales & Marketing
- GainSight
- Bloomreach
- Blue Yonder
- Lattice

Customer Service
- Medallia
- Salesforce
- Claron

Human Capital
- Ravel
- Everlast
- Pendragon

Legal
- Legal Cloud
- Legal RAVEL
- Legal UCAN

Security
- Cylance
- CounterTack
- Tenable

Vertical AI Applications
- Ad Optimization
- Security
- Vertical AI

Vertical Applications
- Ad
- Optimization
- Security
- Vertical Applications

Graph Databases
- Neo4j
- OrientDB

Cloud EDW
- Amazon Redshift
- Snowflake

MPP Databases
- Teradata

Data Transformation
- Alteryx
- IBM InfoSphere DataStage

Data Integration
- Informatica

BI Platforms
- Power BI
- Tableau

Statistical Computing
- SAS
- SPSS
- Matlab

Log Analytics
- Splunk
- Loggly

Social Analytics
- NetBase
- Datasync

Data science
- H2O
- DataRobot

Cloud Analytics
- Big Data Cloud

Cloud Infra/Analytics
- IBM Watson
- Sentient

Cross-Infrastructure/Analytics
- Amazon Web Services
- Google Cloud Platform
- IBM
- Microsoft
- SAP
- SAS
- hp
- netapp
- Teradata
- Oracle
- TIBCO
- Talend

Finance
- Lending Club
- LendUp
- Kabbage

Retail
- RetailNext
- Retail

Health
- Practice Fusion
- Fitbit
- Vitalsigns

Data Sources & APIs
- Financial & Economic Data
  - Bloomberg
  - Dow Jones
  - S&P Capital IQ
- Air / Space / Sea
  - Airware
  - Ozone Display

Incubators & Schools
- DataCamp
- Insight
- Data Elite
- The Data Incubator
### Big Data Landscape 2016

#### Infrastructure
- Hadoop
  - On-Premise: Cloudera, Hortonworks, Pivotal
  - In the Cloud: Amazon Hadoop, IBM InfoSphere, Red Hat, Hortonworks
- Spark
- Cluster Services
  - Apache Mesos
  - Mesosphere
  - Mesos
- NoSQL Databases
  - MongoDB
  - Cassandra
  - Couchbase
  - Riak
  - Neo4j
- Graph Databases
  - Neo4j
  - OrientDB
  - TitanDB
- Teradata
- Amazon Redshift
- Data Warehousing
  - Snowflake
  - TIBCO Data Management
  - Qlik
  - Cloudera
- NewSQL Databases
  - MariaDB
  - MySQL
  - Postgres
- MPP Databases
  - IBM Netezza
  - SQL Data Warehouse
- Cloud EDW
  - Amazon Redshift
  - Oracle Big Data Cloud
- Data Warehouses & Marts
  - Snowflake
  - Dremio
  - Starburst
  - KSQL
- Data Lakes & Data Stores
  - Apache Hadoop
  - Apache Hadoop
  - Apache Hadoop
  - Apache Hadoop
- Data Catalogs
  - AWS
  - Azure Data Lake
  - Google Cloud
- Data Governance & Security
  - Cloudera
  - Actian
  - Talend
- API Management
  - API Management
  - API Management
  - API Management
- Machine Learning & AI
  - Google Cloud AI
  - Azure Machine Learning
  - IBM Watson
  - Amazon SageMaker

#### Analytics
- Data Science Platforms
  - Revolution
  - Anaconda
  - KNIME
- Machine Learning
  - TensorFlow
  - PyTorch
  - Scikit-learn
- Predictive Analytics
  - Tableau
  - Looker
  - Power BI
  - Splunk
- Business Intelligence
  - Qlik Sense
  - Tableau
  - Power BI
- Business Intelligence Tools
  - Tableau
  - Qlik Sense
  - Tableau
  - Power BI
  - Splunk

#### Applications
- Sales & Marketing
  - Marketo
  - Salesforce
  - Microsoft Dynamics
  - Oracle CRM
- Customer Service
  - Salesforce
  - Zendesk
  - ServiceNow
- Human Capital Management
  - Workday
  - SAP SuccessFactors
  - Oracle HCM Cloud
  - PeopleSoft
  - Workday

#### Visualization
- Tableau
- Qlik Sense
- Power BI
- Docebo
-Canvas

#### Security
- Cylance
- CrowdStrike
- FireEye
- Kaspersky
- McAfee

#### Graph Databases
- Neo4j
- OrientDB
- TitanDB
- Duraspace
- MarkLogic
- Amazon Neptune

#### Search
- Elasticsearch
- Solr
- Apache Flume
- Apache Hadoop

#### Formats & Standards
- JSON
- CSV
- XML
- RDF
- Avro
- Parquet

#### Open Source
- Apache Hadoop
- Apache Spark
- Apache Flink
- Apache Cassandra
- Apache Kafka
- Apache Druid

#### Incubators & Schools
- DataCamp
- Data Elite
- MIT
- UIUC
- Stanford

---

© Matt Turck (@mattturck), Jim Hao (@jimrhhao), & FirstMark Capital (@firstmarkcap)
HADOOP DISTRIBUTED FILE SYSTEM
BACKGROUND AND TERMINOLOGY
CLUSTER
WHAT FEATURES DOES HDFS HAVE?
FAULT TOLERANCE
SCALABILITY

2 Nodes
- 24 Cores
- 256 GB
- 3 TB
  (2 nodes min config)

16 Nodes
- 576 Cores
- 24 TB
- 194 TB
  (16 nodes max config)
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
DESIGNED FOR PROCESSING DATA
HOW DOES HDFS WORK?

HDFS ARCHITECTURE
MASTER-SLAVE ARCHITECTURE

- Master: **Namenode**
  - maintains **metadata** about files

- Slave(s): **Datanode**
  - runs on **each node in cluster**
  - stores data in **local file systems**
files split into blocks
(default 128MB)
where can I write?
Data is replicated (3 times)
Data is replicated (3 times)
Fault Recovery
Fault Recovery

DN1

DN2

DN3

DN4

NN
Fault Recovery
HOW TO USE HFS

• 3 ways

• Programmatically (Java, Python, Scala, etc)

• Command line interface (hdfs command)

• Web interfaces (Hue, etc)
Example Commands

• hdfs dfs -ls /input

• hdfs dfs -put ipsum.txt /input/

• hdfs dfs -cat /input/ipsum.txt

• hdfs dfs -rm -r /input/

• Note: all commands begin ‘hdfs dfs’
COMMON PITFALLS WITH HDFS
LOTS OF SMALL FILES
LOTS OF SMALL FILES

SequenceFile File Layout

Data

| Key | Value | Key | Value | Key | Value | Key | Value |

MapFile File Layout

Index

| Key | Key |

Data

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
TRYING TO WRITE TO FILES

HDFS files are read only
MISCONFIGURATION

• Each DataNode should have multiple physical drives
• HDFS should be using each drive
• Using hardware redundancy (RAID)
HISTORY

- Google published in 2004
- Inspired by LISP map() and reduce() functions
- Apache Hadoop (2006, Yahoo)
- Hadoop 2.x, aka YARN (~2012)
map function
map function

Input: (key, value)
map function

Input:  (key, value)

Output:  list(key, value)
map function

Input:  (key, value)

Output:  list(key, value)

reduce function
**map function**

**Input:** (key, value)

**Output:** list(key, value)

---

**reduce function**

**Input:** (key, list(value))
**map function**

**Input:** (key, value)

**Output:** list(key, value)

**reduce function**

**Input:** (key, list(value))

**Output:** list(value)
FAULT TOLERANCE

• if a task fails (or is really slow)
  • spawns same task on another node

• retries tasks 3 times
HADOOP ACTIVITY
HADOOP ACTIVITY

• Read your paragraph
• Write down how often words appear
• Send paper to instructor
HADOOP ACTIVITY

- Read your paragraph
- Write down how often words appear
- Send paper to instructor
  - facilisis
  - dictum
  - magna
EXAMPLE: COUNTING WORDS

- **Input**: lines of text
- **Output**: list of words and how often they appeared
for line in sys.stdin:

  words = [s.strip() for s in re.split('[\s]', line) if s]

for word in words:

  print '%%s\t%%s' % (word, 1)
for line in sys.stdin:
    word, count = line.strip().split('	', 1)
    if current_word == word:
        word_count += int(count)
    else:
        if current_word:
            print '%s	%s' % (current_word, word_count)
        word_count = count
        current_word = word
COMMON PITFALLS WITH MAPREDUCE
TRYING TO USE MAP REDUCE FOR EVERYTHING
NOT USING COMBINERS
NOT CHANGING NUMBER OF REDUCERS
APACHE
SPARK
HISTORY

- Research project in 2009
- Open sourced and made Apache project
- Most actively developed open source project in Hadoop ecosystem
Data Sharing in MapReduce

Input

HDFS read

iter. 1

HDFS write

iter. 2

HDFS read

HDFS write

. . .

HDFS read

query 1

result 1

query 2

result 2

query 3

result 3

. . .

HDFS read

input
Data Sharing in MapReduce

- **Input** reads from HDFS, then performs iter. 1 and iter. 2 with HDFS reads and writes.
  - Iter. 1 reads from HDFS, performs query 1, and writes to result 1.
  - Iter. 2 reads from HDFS, performs query 2, and writes to result 2.
  - Iter. 2 reads from HDFS, performs query 3, and writes to result 3.
  - Iterations continue...

- **Slow** due to replication, serialization, and disk IO.
Data Sharing in Spark

Input

iter. 1

iter. 2

... one-time processing

Distributed memory

query 1

query 2

query 3

...
Data Sharing in Spark

Input → iter. 1 → iter. 2 → ... → one-time processing → Distributed memory → query 1 → query 2 → query 3 → ... → 10-100× faster than network and disk
KEY IDEA

RESILIENT DISTRIBUTED DATASETS (RDD)
RDD

- Distributed collections
- Can be cached in memory (spill to disk if too large)
- Manipulated to generate new RDDs
- RDDs form a graph of operations
- Automatically rebuilt on failure (fault tolerant)
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(	')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns.

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
driver = spark.textFile("hdfs://...")
errors = driver.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('	')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("")"foo").count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\"\t\")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\'\t\')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('"\t"')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...

Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
## Supported Operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map</code></td>
<td><code>reduce</code></td>
<td><code>sample</code></td>
</tr>
<tr>
<td><code>filter</code></td>
<td><code>count</code></td>
<td><code>cogroup</code></td>
</tr>
<tr>
<td><code>groupBy</code></td>
<td><code>reduceByKey</code></td>
<td><code>take</code></td>
</tr>
<tr>
<td><code>sort</code></td>
<td><code>groupByKey</code></td>
<td><code>partitionBy</code></td>
</tr>
<tr>
<td><code>join</code></td>
<td><code>first</code></td>
<td><code>pipe</code></td>
</tr>
<tr>
<td><code>leftOuterJoin</code></td>
<td><code>union</code></td>
<td><code>save</code></td>
</tr>
<tr>
<td><code>rightOuterJoin</code></td>
<td><code>cross</code></td>
<td>...</td>
</tr>
</tbody>
</table>
SPARK ACTIVITY
GROUP 1
PASS ONLY THE NUMBER 2
GROUP 2

COUNT HOW MANY 2’S AND PASS THAT ALONG
GROUP 3

TAKE 2 * THE COUNT AND PASS THE VALUE TO INSTRUCTOR
INSTRUCTOR

ADD THE NUMBERS AND GIVE RESULT
WHAT DID WE JUST DO?

• We were a Spark program!

• **Group 1**: `rdd1 = rawRdd.filter(lambda x: x == 2)`

• **Group 2**: `rdd2 = rdd1.count()`

• **Group 3**: `rdd3 = rdd2.map(lambda x: x * 2)`

• **Instructor**: `rdd3.reduce(lambda a, b: a + b).collect()`
Example: Counting Words

- **Input**: lines of text
- **Output**: list of words and how often they appeared
READ THE INPUT FILE

rawRdd = sc.textFile('hdfs://master:9000/ipsunm.txt')
SPLIT INTO WORDS

words = rawRdd.flatMap(
    lambda line: [s.strip() for s in re.split('\\s', line) if s]
)

counts = words.map(lambda x: (x, 1))

totals = counts.reduceByKey(lambda a, b: a + b)
SHOW THE RESULT

counts.collect()

or

counts.saveAsTextFile('hdfs://master:9000/out.txt')
COMMON PITFALLS WITH SPARK
MAP() VS FLATMAP()

• Important if your map generates a sequence/array

• map() will return an RDD of arrays
  • e.g., [[1, 2], [2, 3], [4], [5, 6, 7]]

• flatMap() returns an RDD of values
  • e.g., [1, 2, 2, 3, 4, 5, 6, 7]
RUNNING ON YARN
NOT CONTROLLING # OF EXECUTORS
NOT CONTROLLING EXECUTOR MEMORY
Big Data Landscape 2016

Infrastructure
- Hadoop
  - On-Premise
  - In the Cloud
- Spark
- Cluster Services
- NoSQL Databases
  - DynamoDB
  - MongoDB
  - OrientDB
  - Neo4j
- NewSQL Databases
  - MariaDB
  - VoltDB
  - Citrix
  - Infinispan
- Graph Databases
  - Neo4j
  - OrientDB
  - Neo
- MPP Databases
  - Teradata
- Cloud EDW
  - Amazon Redshift
- Data Transformation
  - Alteryx
  - Talend
  - Informatica
- Data Integration
  - Talend
  - Informatica
  - IBM DB2
  - Microsoft SQL Server
- Real-Time
  - Storm
  - Kafka
  - Apache BookKeeper

Analytics
- Analyst Platforms
  - Tableau
  - Palantir
  - Ayasdi
  - Quid
- Analytics Platforms
  - Microsoft
  - Tableau
  - Qlik

Data Science Platforms
- R
- Python
- SAS
- SPSS
- MATLAB
- R
- Python

BI Platforms
- Power BI
- Tableau
- Qlik
- Microsoft
- Microsoft
- Microsoft

Statistical Computing
- R
- Python
- MATLAB
- S-PLUS
- R
- Python

Log Analytics
- Splunk
- loggly
- loggly
- loggly

Social Analytics
- Hootsuite
- Hootsuite
- Hootsuite

Cross-Infrastructure/Analytics

Open Source
- Apache
- Hadoop
- Spark
- Storm
- Kafka
- Finagle
- Scala
- Python

Query/Data Flow
- SQL
- Hive
- Impala
- Cassandra
- MongoDB
- HBase
- Redis

Data Sources & APIs
- Amazon S3
- Google Cloud Storage
- MongoDB
- Apache Cassandra
- Redis
- Kafka

Framework
- Apache
- Hadoop
- Spark
- Kafka
- Akka

Data Access
- Apache
- Cassandra
- MongoDB
- HBase
- Redis

Search
- Google
- Bing
- Yahoo!
- Yandex
- Baidu

Security
- SSL
- TLS
- Kerberos
- OpenID
- SAML

Social Networking
- Twitter
- Facebook
- LinkedIn
- Instagram

Healthcare
- EHR
- EMR
- Patient
- Health

Financial & Economic Data
- Bloomberg
- Dow Jones
- FactSet

Air/Sea/Space
- Airbus
- Boeing
- SpaceX

Location/People/Entities
- Google Maps
- GPS
- GIS

Other
- IoT
- IOT
- Internet
- Internet

Incubators & Schools
- DataCamp
- Data Elite
- FirstMark Incubator

© Matt Turck (@mattturck), Jim Hao (@jimhao), & FirstMark Capital (@firstmarkcap)