IBM Technical Summit

Stay ahead.

IBM

Name: Dharmesh Jain and Shrinivas S Kulkarni

Title:Introduction to Hadoop and BigInsights Programming Overview

Agenda

- **Hadoop Overview**
  - Why Hadoop?
  - Hadoop distributed file system
  - Map Reduce engine
- **JAQL Overview**
  - Need for high level languages
  - Core JAQL operators
- **BigInsights programming**
  - BigInsights Apps
    - Application catalog
    - Develop your own BigInsights Applications
  - Development tools
  - BigSheets

Hardware improvements through the years...

- **CPU Speeds:**
  - 1990 - 44 MIPS at 40 MHz
  - 2000 - 3,561 MIPS at 1.2 GHz
  - 2010 - 147,600 MIPS at 3.3 GHz
- **RAM Memory**
  - 1990 – 640K conventional memory (256K extended memory recommended)
  - 2000 – 64MB memory
  - 2010 - 8-32GB (and more)
- **Disk Capacity**
  - 1990 – 20MB
  - 2000 - 1GB
  - 2010 – 1TB
- **Disk Latency (speed of reads and writes) – not much improvement in last 7-10 years, currently around 70 – 80MB / sec**

How long it will take to read 1TB of data?

- **1TB (at 80Mb / sec):**
  - 1 disk - 3.4 hours
  - 10 disks - 20 min
  - 100 disks - 2 min
  - 1000 disks - 12 sec
- **Parallel Data Processing is the answer!**

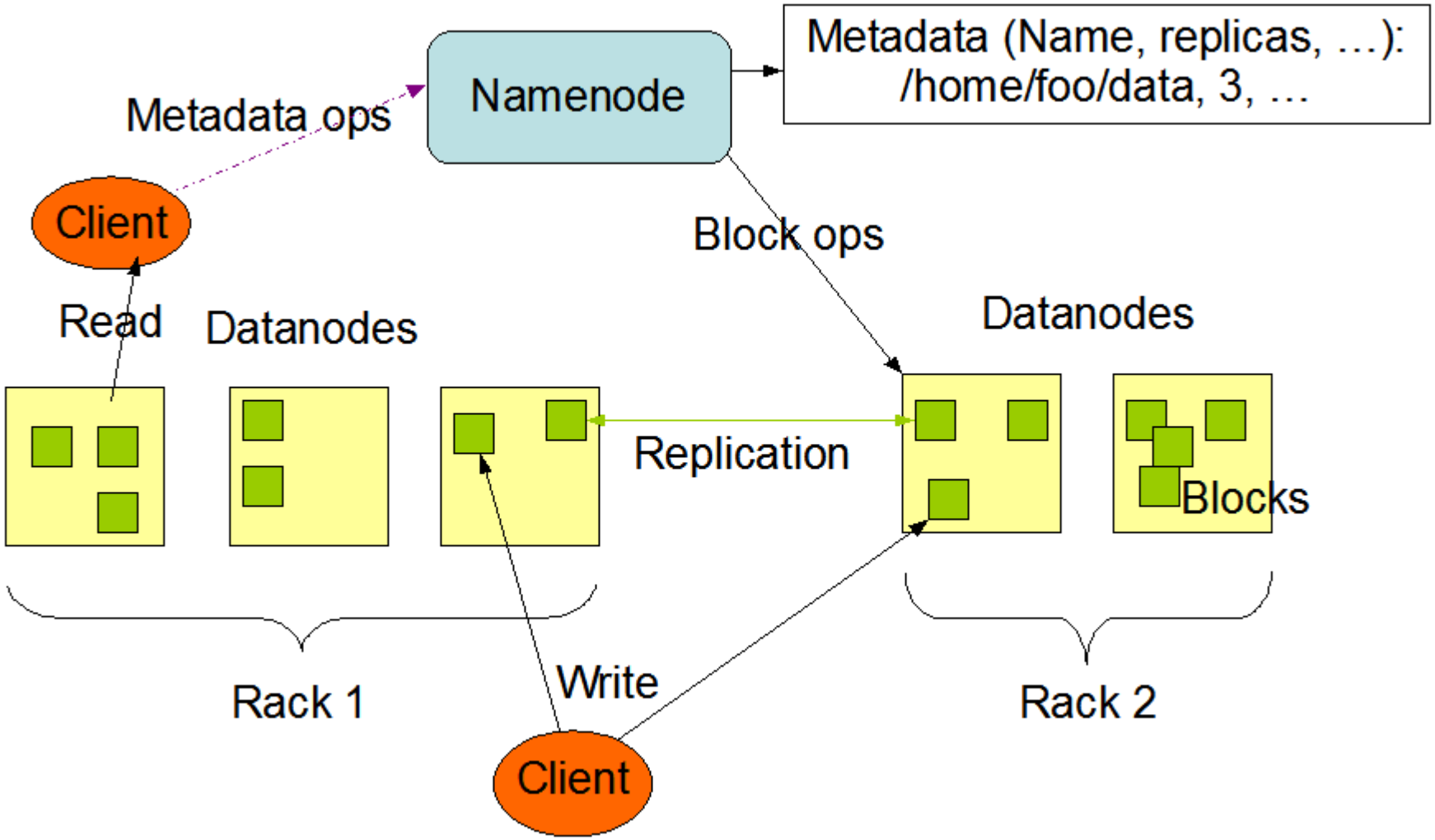
Parallel computing is not new

- **HPC and Grid computing**
  - Move data to computation- Network bandwidth becomes a bottleneck; compute nodes idle
  - Good for compute intensive jobs
  - Exchanging data requires synchronization– very tricky
  - Scalability is programmer's responsibility
    - Will require change in job implementation
- **Hadoop approach**
  - Move computation to data- conserves network bandwidth
  - Shared nothing Architecture- no dependencies between tasks
  - Communication between nodes in frameworks responsibility
  - Designed for scalability
    - Adding increased load to a system should not cause outright failure, but a graceful decline
    - Increasing resources should support a proportional increase in load capacity
    - Without modifying the job implementation

Apache 

- **A scalable fault-tolerant distributed system for data storage and processing (open source under the Apache license).**
- **Inspired by Google technologies**
  - MapReduce
  - Google file system
- **Originally built to address scalability problems of Nutch, an open source Web search technology**
  - Developed by Douglass Read *Cutting* (Doug cutting)
- **Core Hadoop has two main systems:**
  - **Hadoop Distributed File System:** self-healing high-bandwidth clustered storage.
  - **MapReduce:** distributed fault-tolerant resource management and scheduling coupled with a scalable data programming abstraction.

HDFS Architecture – Master/Slaves

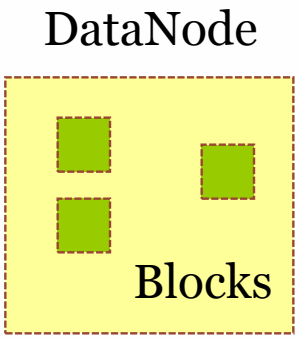


NameNode

Master node
NameNode

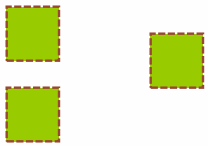
- **Manages the file system namespace**
  - Maintains file system tree and meta data for all files/directories in the tree
  - Single point of failure. Name node loss renders file system inaccessible
- **Centralizes and manages file system metadata in memory**
  - Maps blocks to DataNodes, filenames, etc
  - Metadata size limited to available RAM of NameNode.
  - Bias toward modest number of large files, not large number of small files (where metadata can grow too sizeable)
  - NameNode will crash if it runs out of RAM
- **Runs on a master node**
  - Coordinates access to DataNodes but data never goes on NameNode
  - Hadoop V1 has no built-in failover mechanism for NameNode

DataNode (Slave)



- **Files on HDFS are chopped into blocks and stored on DataNodes**
  - Size of blocks is configurable
  - Different blocks from the same file are stored on different DataNodes if possible
- **Serves read and write requests to clients**
- **Performs block creation, deletions, and replication as instructed by NameNode**
  - Replication factor is configurable
- **One instance of DataNode per slave node is recommended in real deployment**

HDFS Data Blocks



- **Much larger than traditional file system blocks**
  - 64MB by default. Increase to 128MB for very large files.
  - If chunk of file is smaller than HDFS block size, only needed space is used
- **Trade-off: block size and MapReduce parallelism**
  - Map tasks in MapReduce normally operate on one block at a time
  - so if you have too few tasks (fewer than nodes in the cluster), your jobs will run slower than they could otherwise
  - Minimize the cost of seeks
- **Advantages of HDFS's data block approach**
  - Simplifies replication, providing fault tolerance and reliability
- **Each block replicated across 3 DataNodes (by default)**
  - 1st replica placed on same node as client
  - 2nd replica placed on different rack from 1st rack
  - 3rd replica placed on same rack as 2nd rack, but on a different node

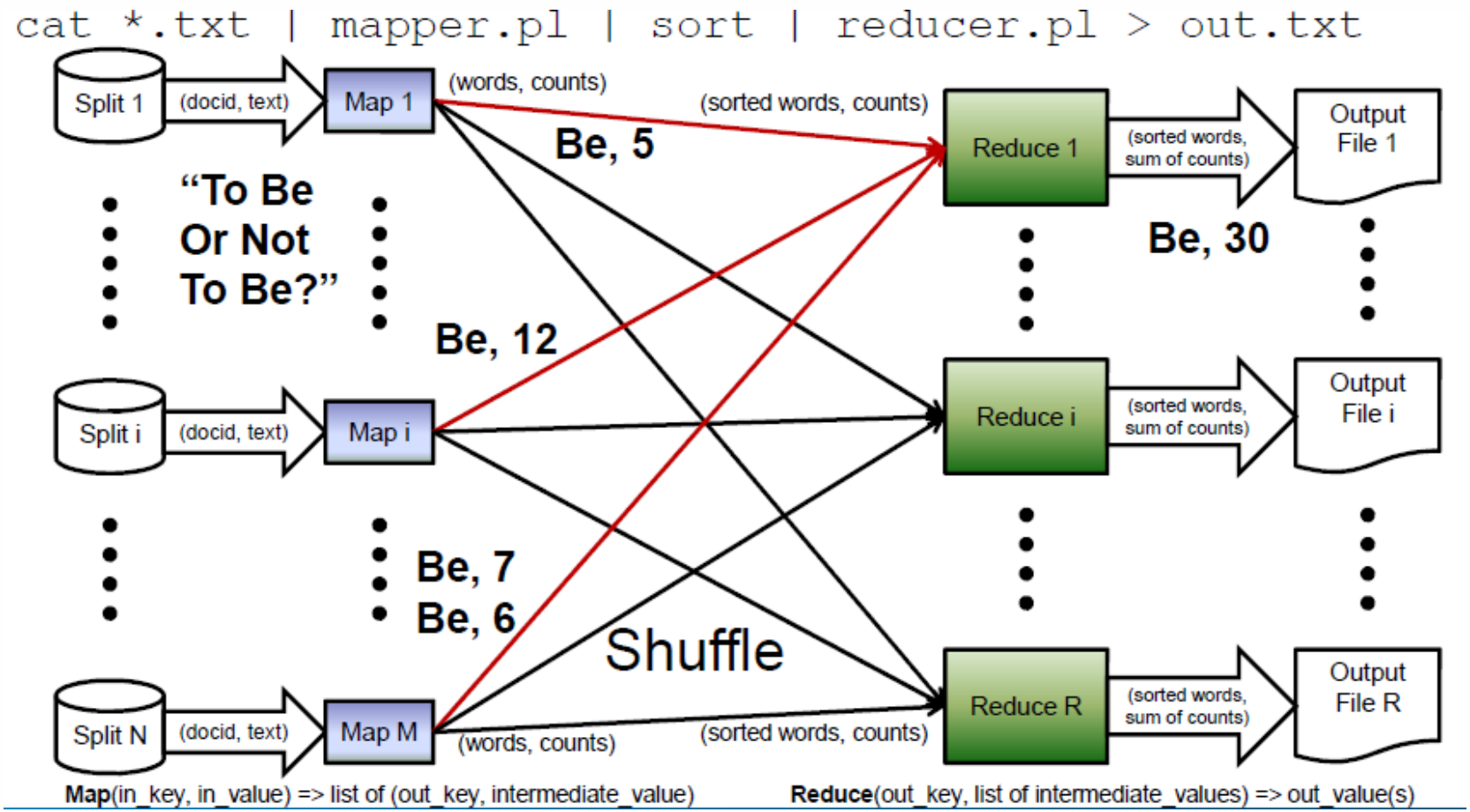
Map Reduce 101

- **Originated in functional programming but common in many languages**
- **Example of Map function:**
  - **square** x = x * x
  - map square [1,2,3,4,5] will return [1,4,9,16.25]
- **Notice that I can process "map square" in parallel:**
  - map square [1,2,3] -> [1,4,9]
  - map square [4,5] -> [16,25]
- **Example of Reduce function:**
  - MAX (1, 2, 3, 10,15, 20) -> 20
  - SUM (1, 7, 10) -> 18
- **In Hadoop REDUCE function always takes MAP function as an input**
  - REDUCE phase is optional, for some jobs no reducing is required

Hadoop MapReduce engine

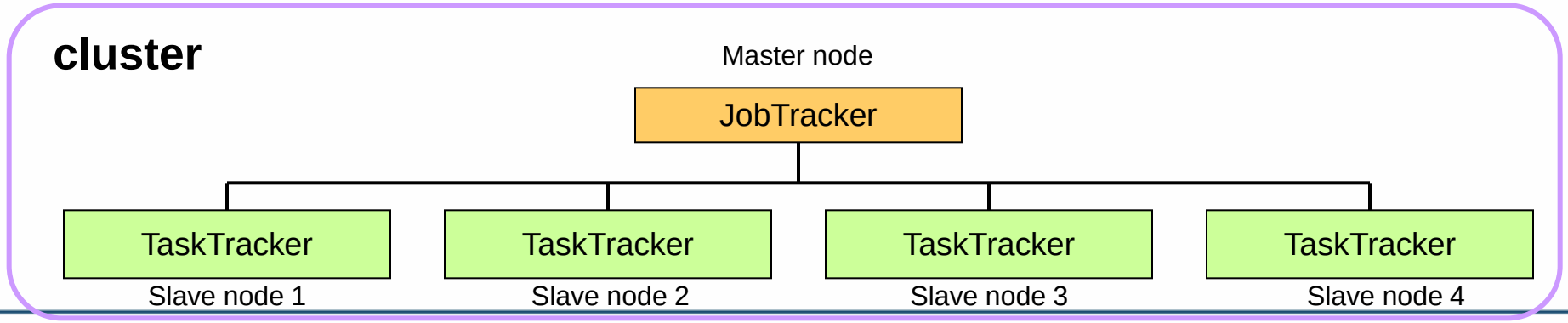
- **Framework which enables writing applications to process multi-terabyte of data in-parallel on large clusters (thousands of nodes) of commodity hardware**
- **A clean abstraction for programmers**
  - No need to deal with internals of large scale computing
  - Implement just Mapper and Reducer functions- most of the times
  - Implement in the language you comfortable with
    - Java (assembly language for Hadoop)
    - With hadoop streaming, you can run any shell utility as mapper and reducer
    - Hadoop pipes to support implementation of mapper and reducer in C++.
- **Automatic parallelization & distribution**
  - Divides the job into tasks (map and reduce task)
  - Schedules submitted jobs
  - Schedules tasks as close to data as possible
  - Monitors task progress
- **Fault-tolerance**
  - Re-execute failed or slow task instances.

Data flow in a map reduce job



MapReduce Architecture- master/slave

- **Single master (JobTracker) controls job execution on multiple slaves (TaskTrackers).**
- **JobTracker**
  - Accepts MapReduce jobs submitted by clients
  - Pushes *map* and *reduce* tasks out to TaskTracker nodes
  - Keeps the work as physically close to data as possible
  - Monitors tasks and TaskTracker status
- **TaskTracker**
  - Runs map and reduce tasks; Reports status to JobTracker
  - Manages storage and transmission of intermediate output



Word count Mapper

```
public static class WordCountMapper extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    } } }
```

Word count Reducer

```
public static class WordCountReducer extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  } }
```

Prepare and Submit job

```
public class WordCountJob {
    public static void main(String[] args)  throws Exception{
     JobConf conf = new JobConf(WordCount.class);

    // specify input and output dirs
    FileInputFormat.addInputPath(conf, new Path("input"));
    FileOutputFormat.addOutputPath(conf, new Path("output"));
    // specify output types
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    //InputFormat and OutputFormat
    conf.setInputFormat(TextInputFormat.class);

    conf.setOutputFormat(TextOutputFormat.class);

    conf.setMapperClass(WordCountMapper.class); // specify a mapper

    conf.setReducerClass(WordCountReducer.class); // specify a reducer
    conf.setCombinerClass(WordCountReducer.class);


     conf.setNumberOfReducer(2); //Number of reducer

    JobClient.runJob(conf); // Submit the job to Job Tracker

}}
```

Need for higher level languages

- **Although the Hadoop framework is implemented in Java, MapReduce applications do not need to be written in Java**
- **To abstract complexities of Hadoop programming model, a few application development languages have emerged that build on top of Hadoop:**
  - Pig
  - Hive
  - Jaql

Jaql

Word count in JAQL

- read(lines("/WordCount/input")) -> expand tokenizer($)
  -> group by w = $ into { word:w ,frequency: count($)}
  -> sort by [$.frequency]
  -> write(file("/WordCount/output/wordcount_result"));
- Anatomy of Word count
  - $doc = read(lines("/jaqlsession/data/file1"));
  - $alltokens = $doc -> expand tokenizer($);
  - $unsortedresult = $alltokens -> group by w = $ into { word:w , frequency: count($)};
  - $sortedresult = $unsortedresult -> sort by [$.frequency] ;
  - $sortedresult -> filter $.frequency > 5;
  - $stopWordList = ["the", "Hello","Work"];
  - $sortedresult -> filter not ($.word in $stopWordList);
  - $unsortedresult -> top 3 by [$.frequency desc];
  - $sortedresult -> write(file("/opt/ibm/JaqlSession/sortedWordCount"));

Core operators – TRANSFORM

- The transform operator allows you to manipulate the values in an array ("project" in DBMS vernacular)
  - An expression is applied to each element in the array
  - The result of the expression is the next element in the output array

```
jaql> recs = [ {a: 1, b: 4},  {a: 2, b: 5},  {a: -1, b: 4} ];
jaql> recs -> transform $.a + $.b;
[ 5, 7, 3 ]

jaql> recs -> transform { sum: $.a + $.b };
[ { sum: 5 }, { sum: 7 }, { sum: 3 } ]
```

- The `each` clause can be applied if you don't like `$`

```
jaql> recs -> transform each rec { sum: rec.a + rec.b };
[ { sum: 5 }, { sum: 7 }, { sum: 3 } ]
```

Other important core operators

- **EXPAND**
  - The expand operator flattens nested arrays
- **FILTER**
  - filter allows you to selectively filter out array entries
- **GROUP**
  - Performs SQL-style GROUP BY against a single input
- **JOIN**
  - The join operator joins two or more arrays
- **SORT and TOP**
  - The sort operator allows sorting of arrays
  - The top operator returns the first k rows of its input array

Why I love JAQL?

- **There are other well-known languages (Hive, Pig, etc.)** Why another?
- **Hive**
  - Good for "flat", structured data
  - Has Java UDF/UDA's
  - Familiar SQL syntax
- **Pig**
  - Better for moderately complex, nested data
  - Has Java UDF/UDA's
  - Used for simple scripts
- **JAQL: A JSON Query Language**
  - Elegantly handles deeply nested data (e.g. text analytics).
  - Has Java UDF/UDA's
  - Native Jaql functions and aggregates
  - Modules and functions allow for larger, more complex projects
  - Seamlessly integrates Jaql and SQL syntax

Developing and sharing jobs is not easy!

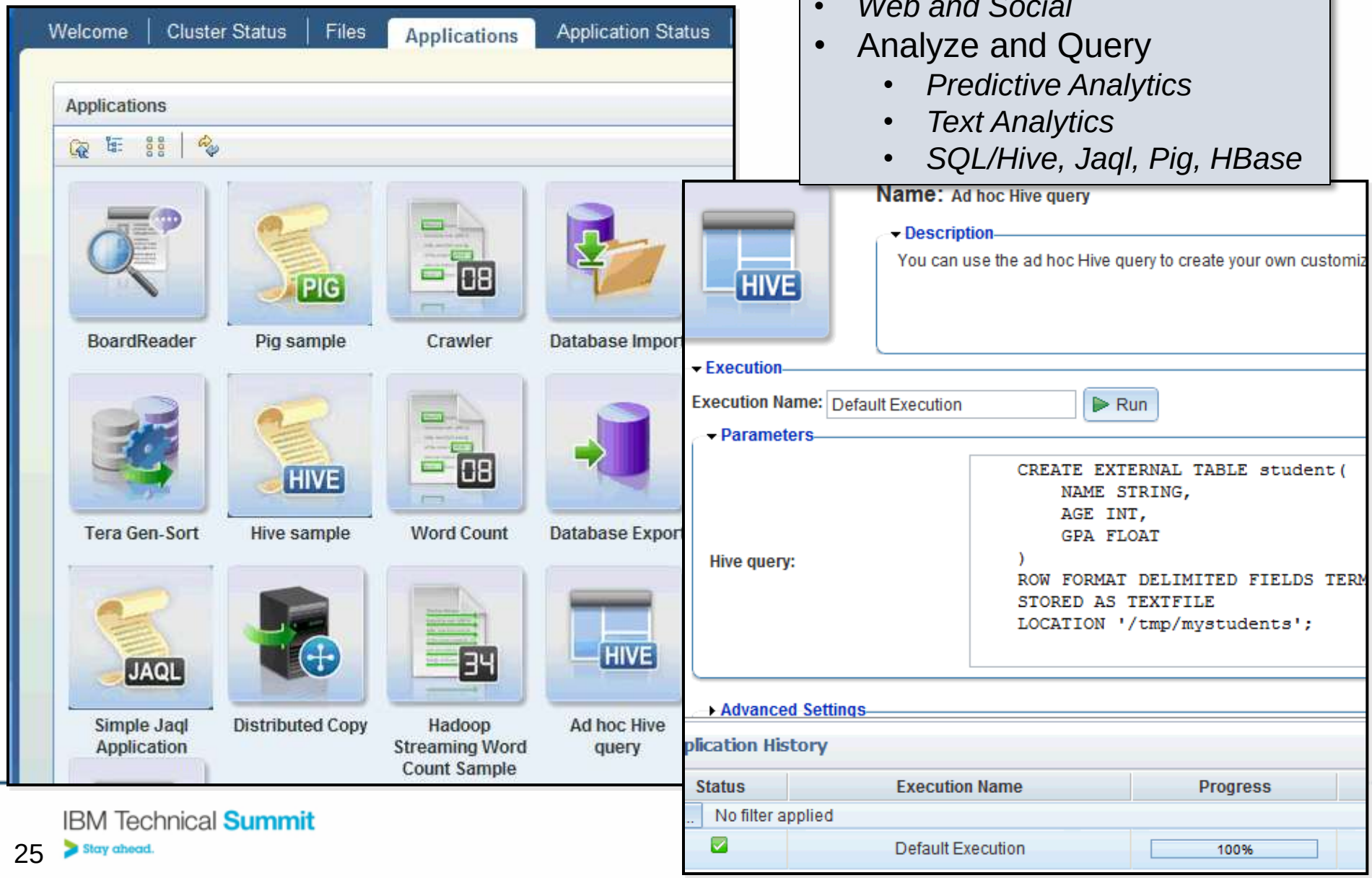
- **Typical Workflow**
  - Import data into the cluster
  - Analyze the imported data
    - Java MR jobs ; JAQL; Hive; Pig
  - Write back the analysis result in cluster
  - Optionally, export the results out
- **Some jobs needs to be run periodically**
  - Indexing
- **Big data jobs are long running**
  - You have to monitor them; to track progress
- **Packaging the workflow**
  - So that it is consumable

Applications Catalog (Web Console)

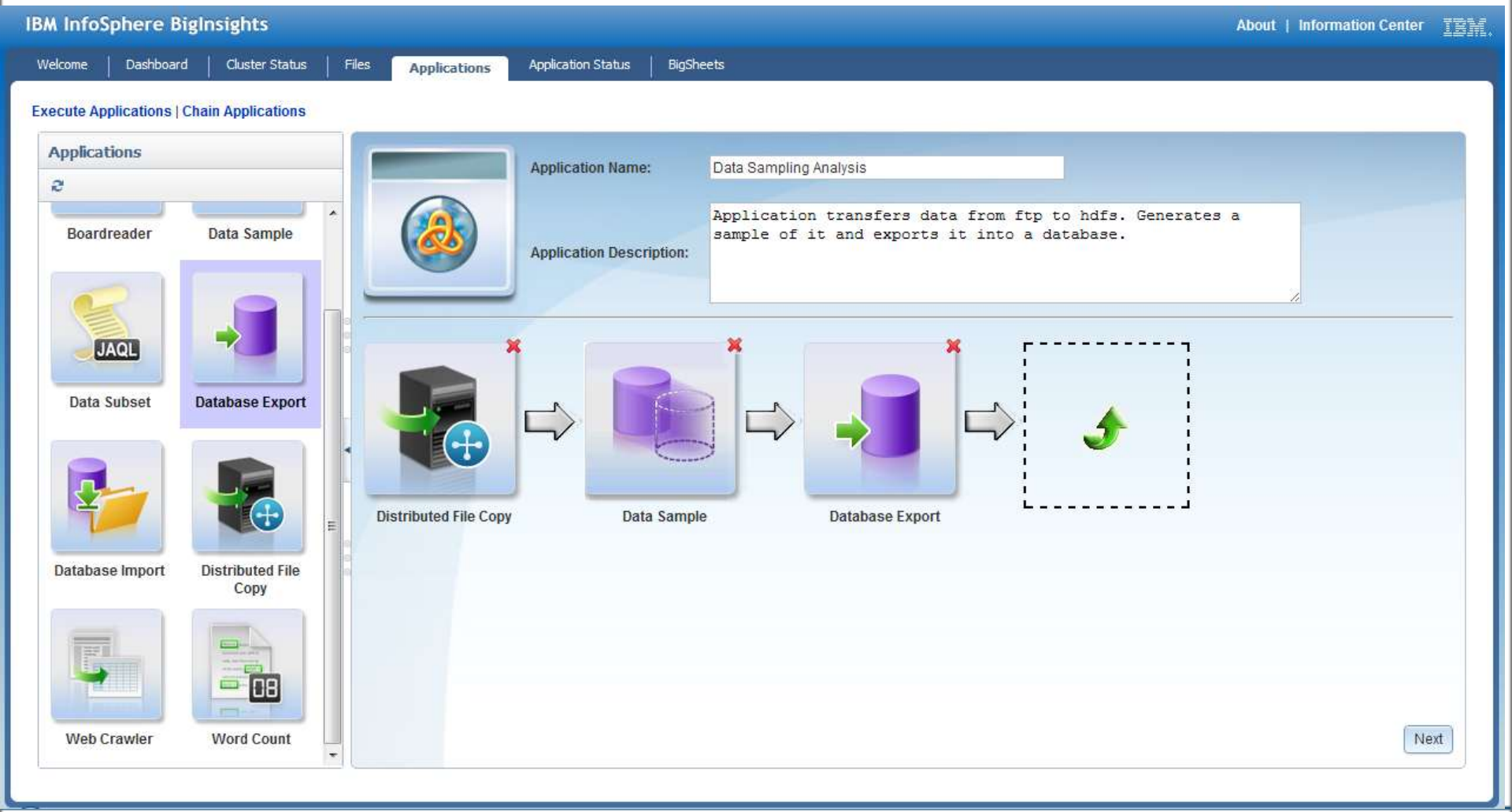
- Browse available applications
- Deploy published applications (administrators only)
- Launch (or schedule for launch) a deployed application
- Monitor job (application) execution status

Running Applications

- Import & Export Data
  - *Database & Files*
- *Web and Social*
- Analyze and Query
  - *Predictive Analytics*
  - *Text Analytics*
  - *SQL/Hive, Jaql, Pig, HBase*

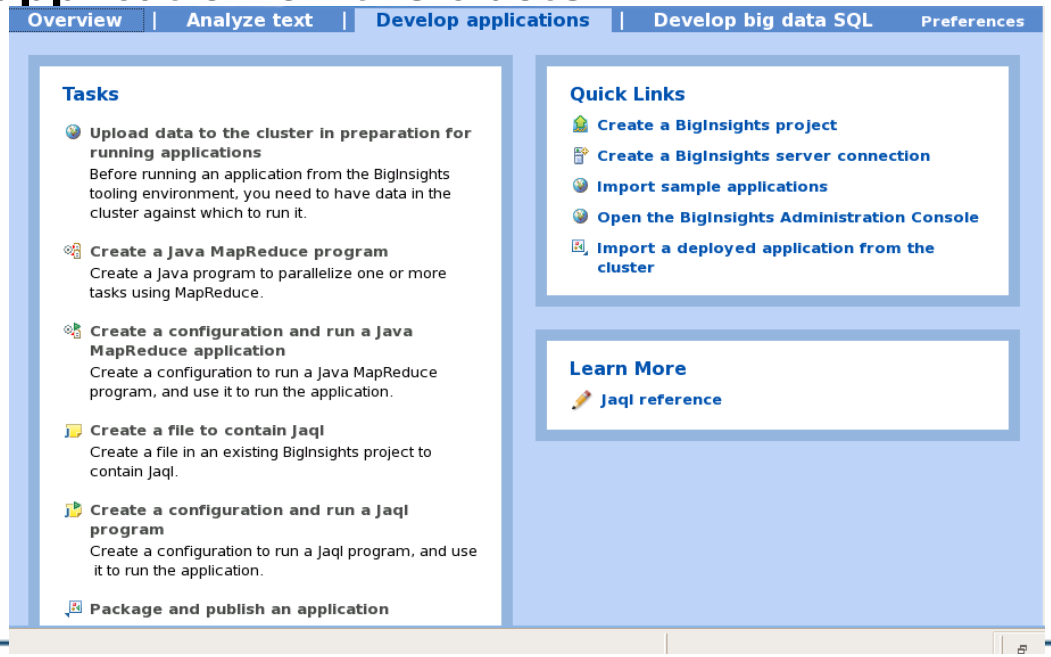


Quickly drag and drop to create new Apps



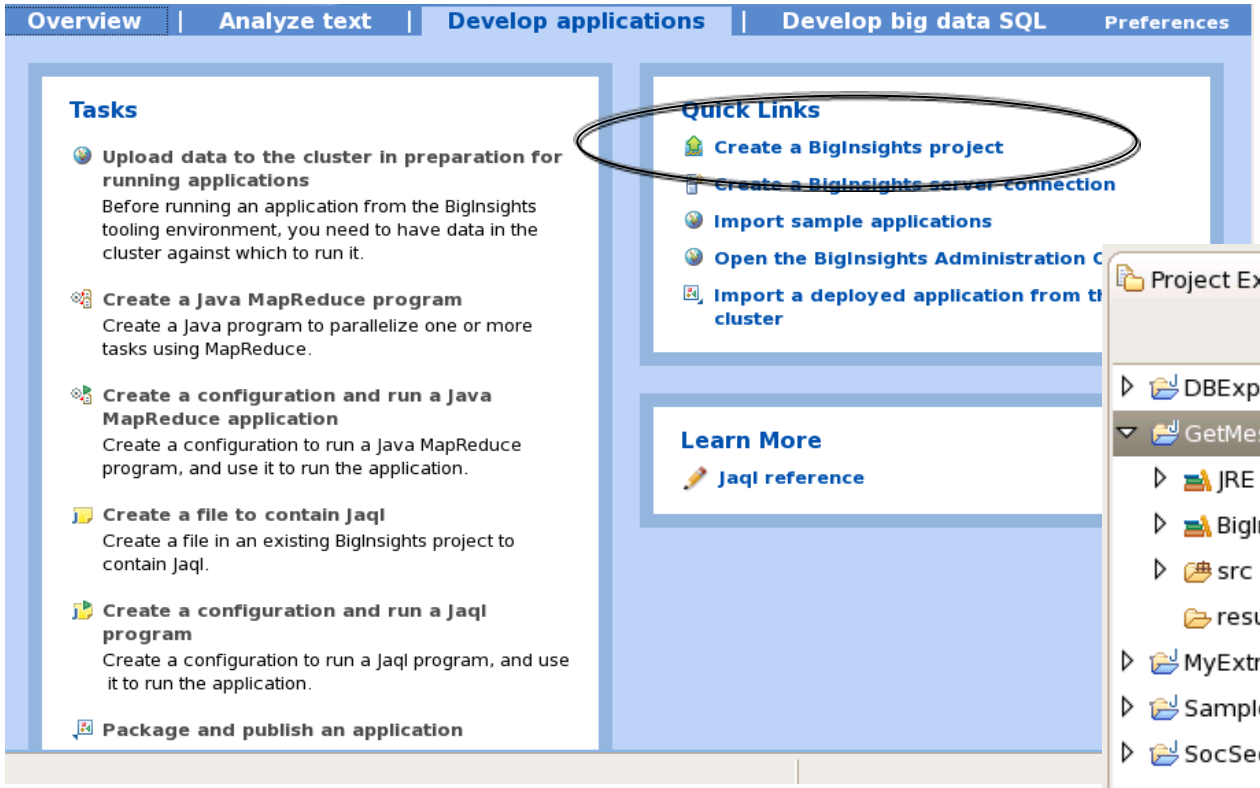
Overview of Application Development Lifecycle

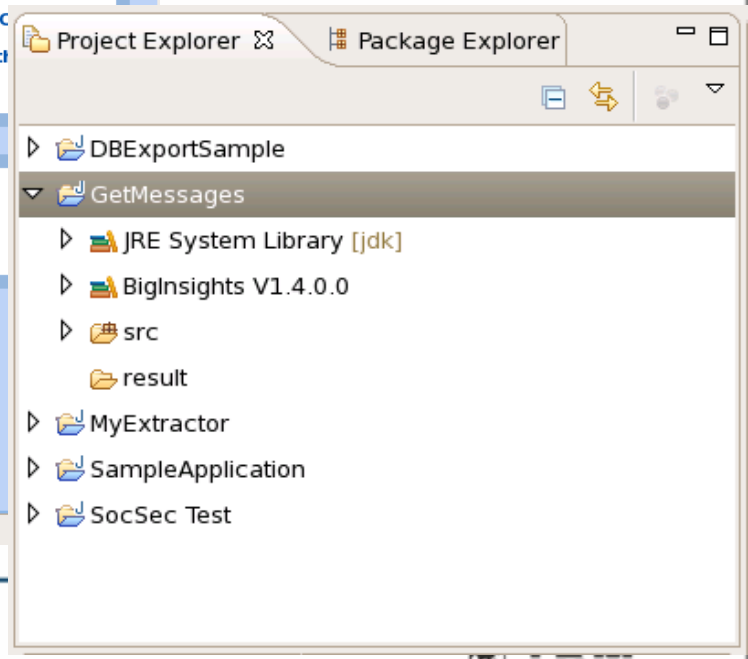
- **Configure your Eclipse environment (one-time set up)**
- **Develop your application using InfoSphere BigInsights tools**
- **Test your application**
- **Package and publish your application**
- **Deploy your application on the cluster**



Develop your application – Scripting example

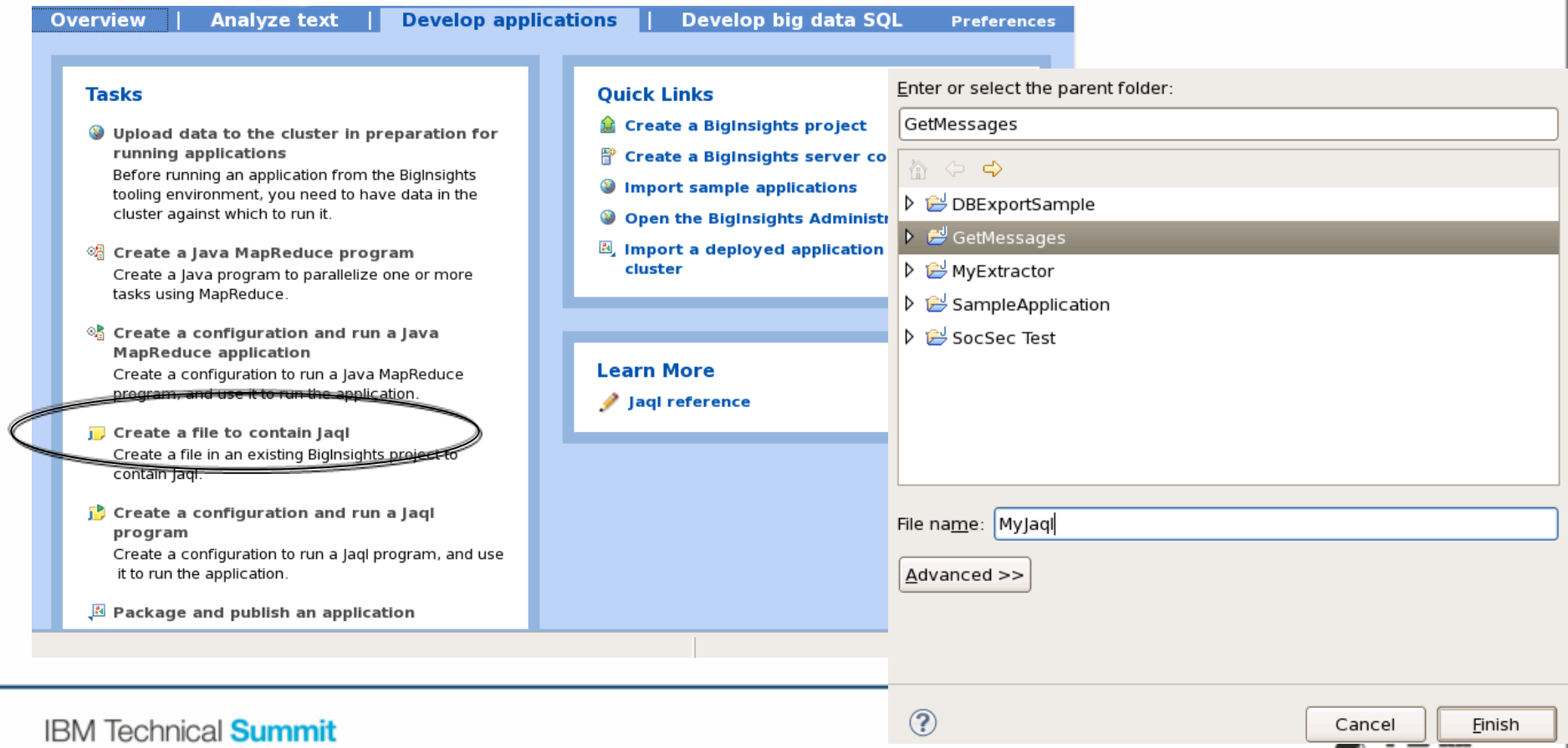
- Open the BigInsights perspective in Eclipse
- Create a BigInsights project





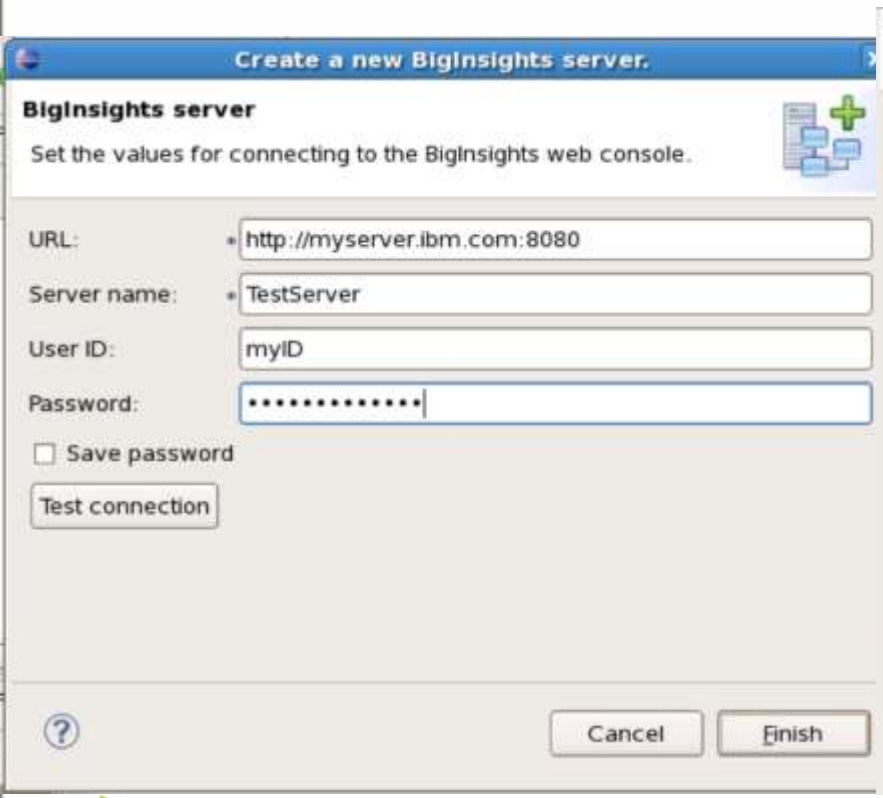
Develop your application – Scripting example (cont'd)

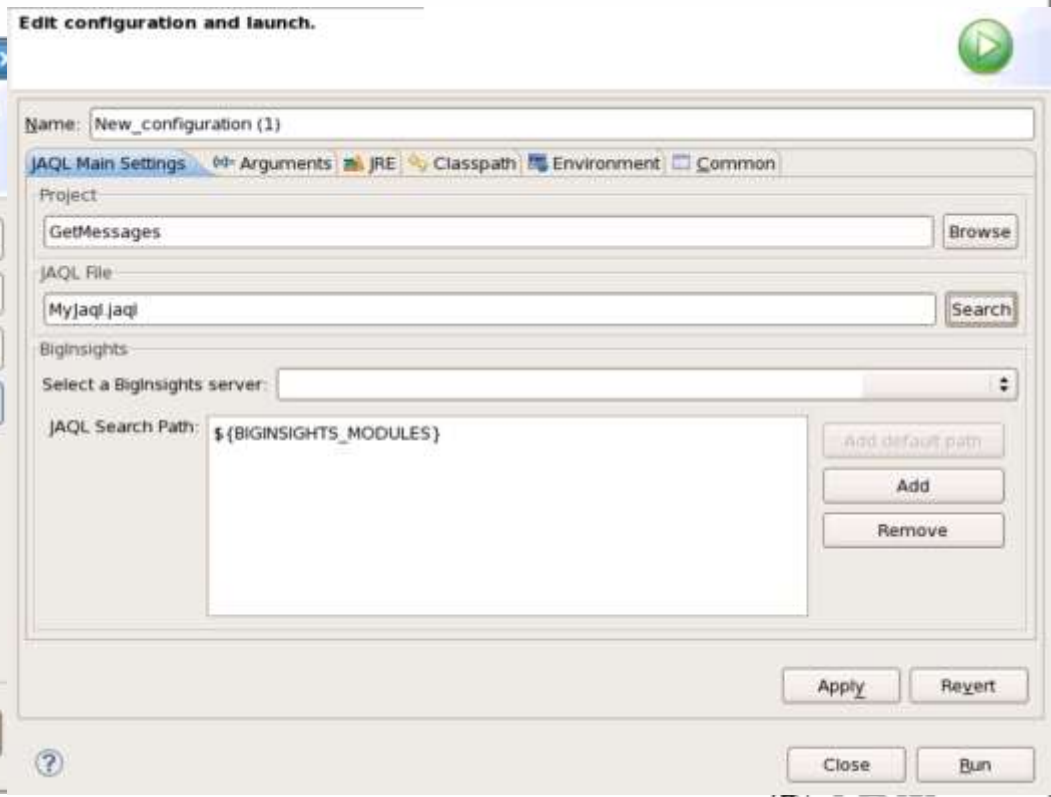
- **Create a Jaql file within your BigInsights project**



Test your application

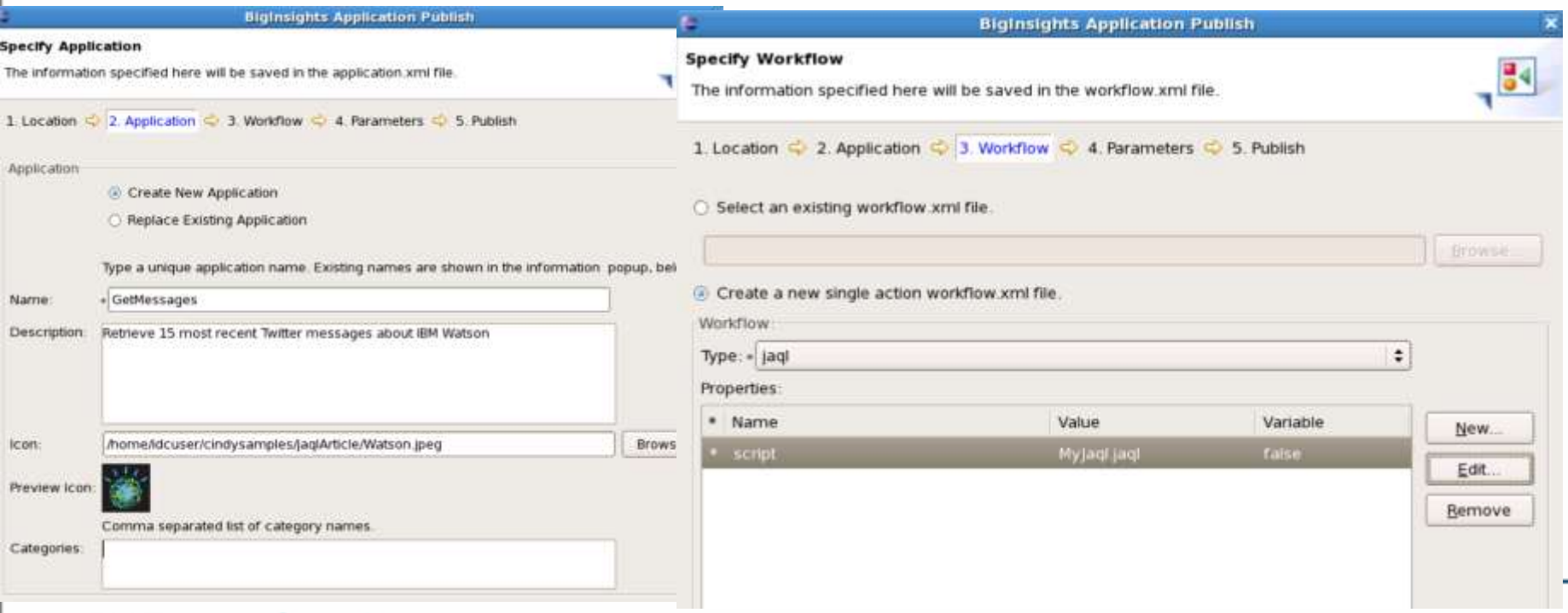
- **Create a BigInsights server connection**
- **Define a configuration for running your application**
- **Run your application from Eclipse**





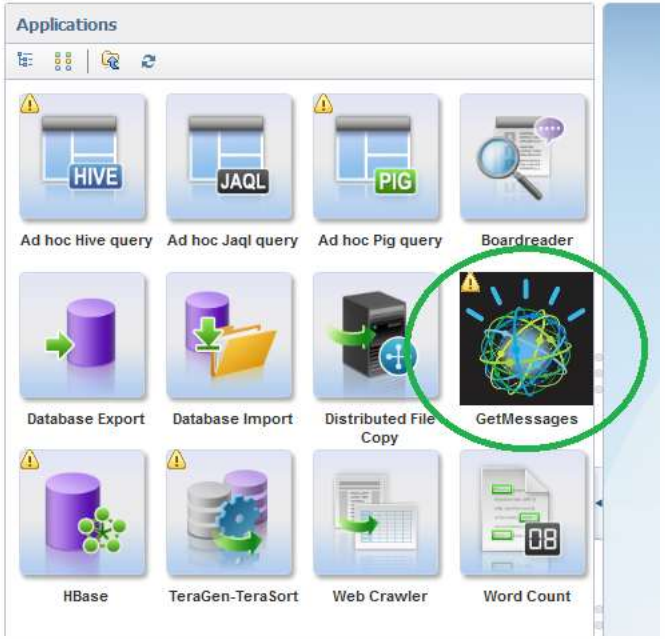
Publish your application to the console App catalog

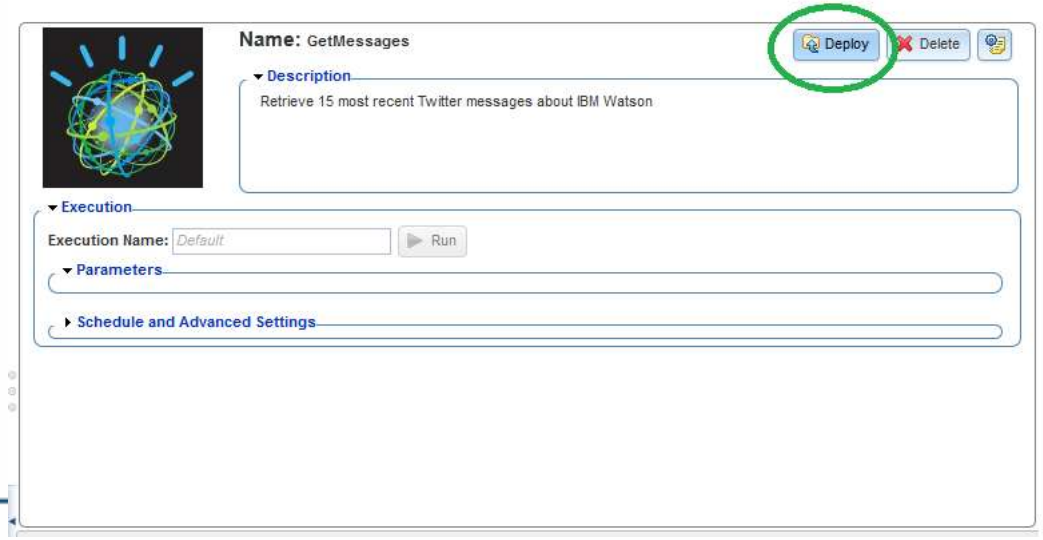
- Package and publish your application using the InfoSphere BigInsights Eclipse Task Launcher
- Specify application name, input parms, workflow requirements, etc.



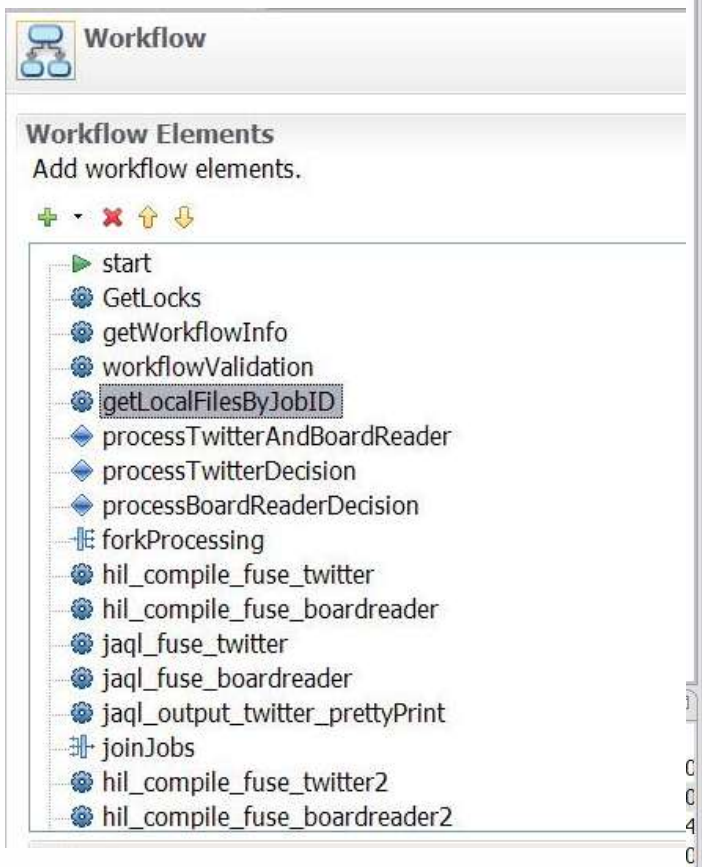
Deploy your application on the cluster

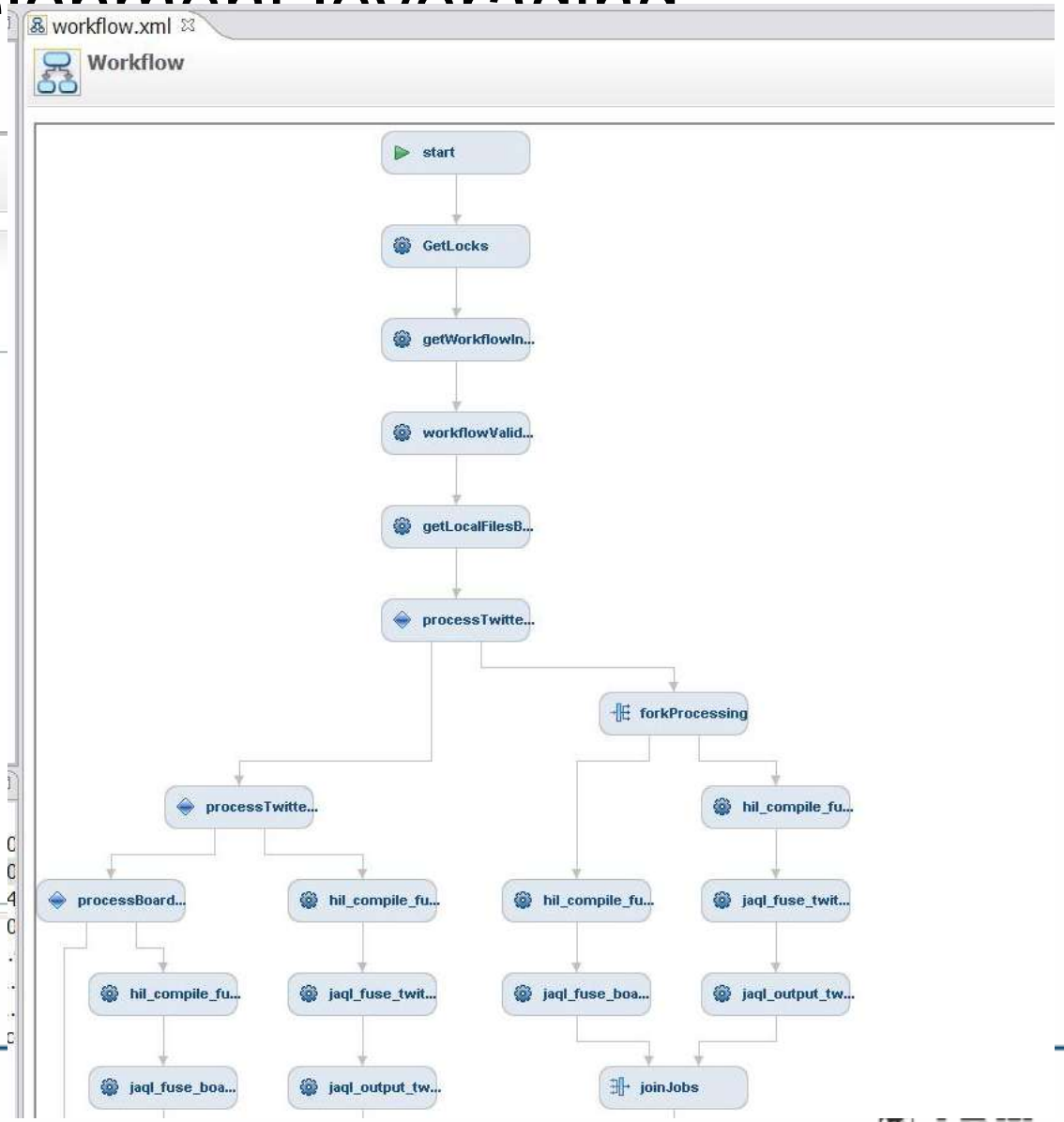
- Access the Applications tab of the Web console
- Locate your new app and click “deploy”
- Optionally, “run” the application after it’s been deployed



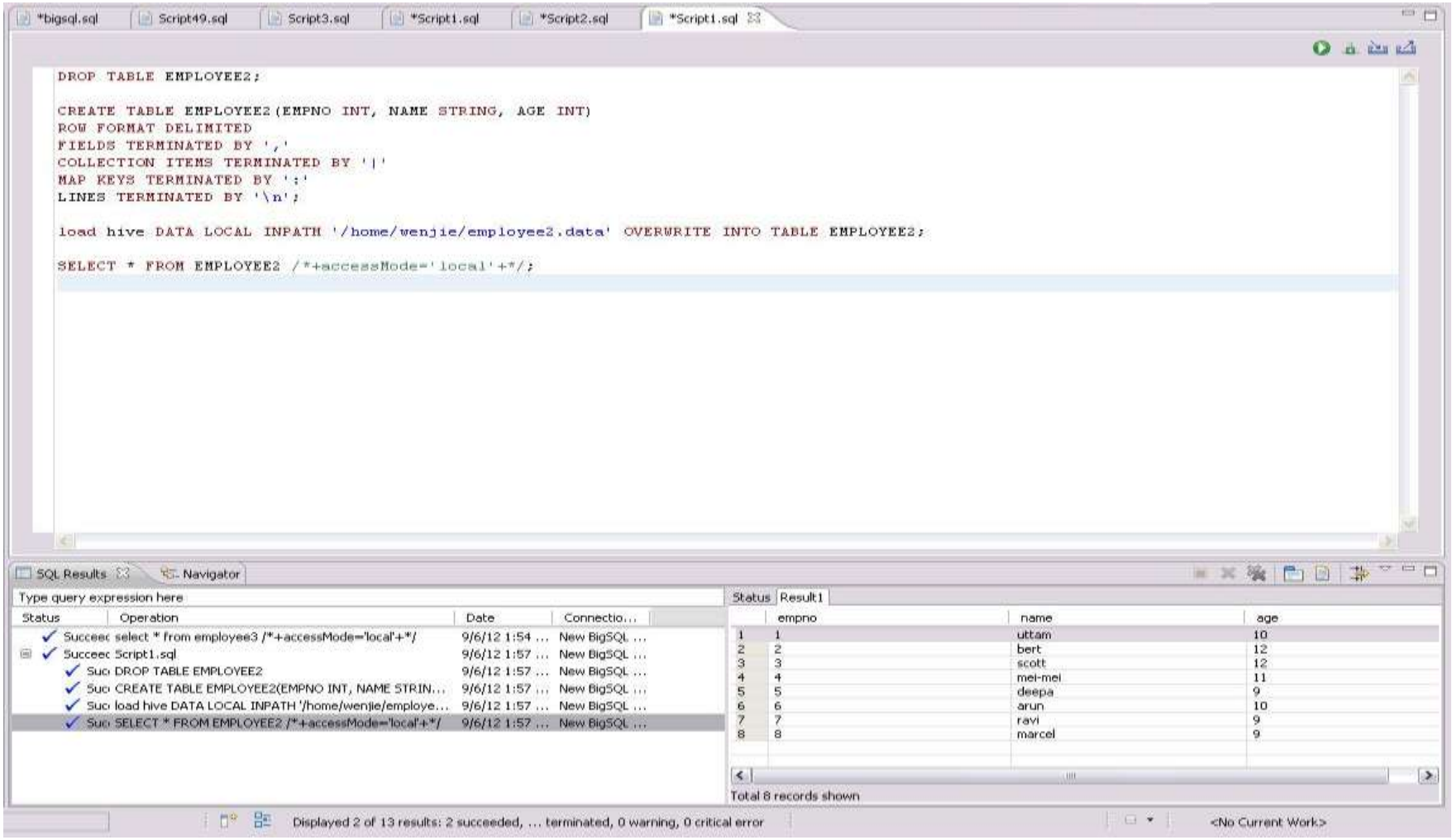


Workflow development leveraging

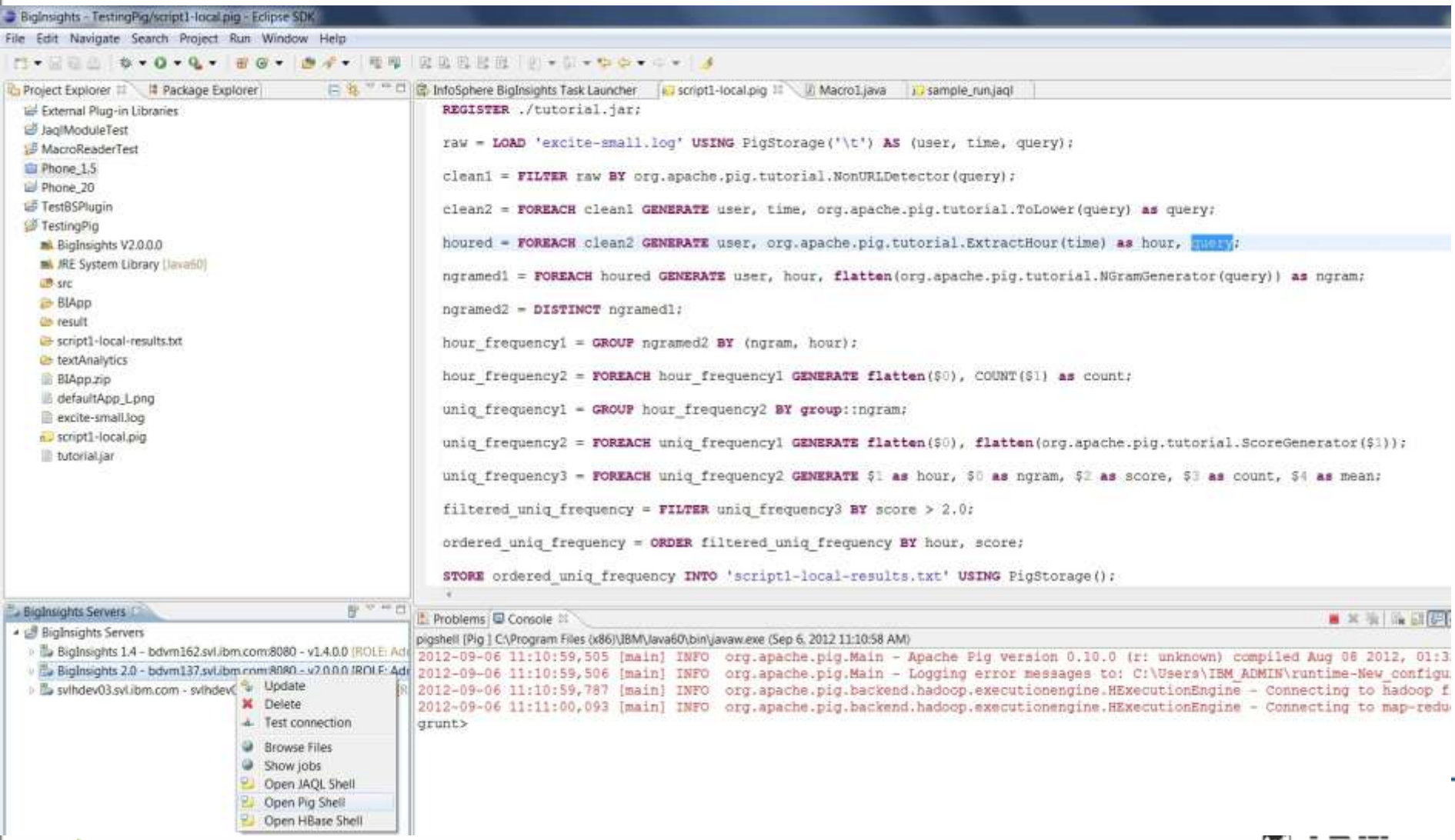




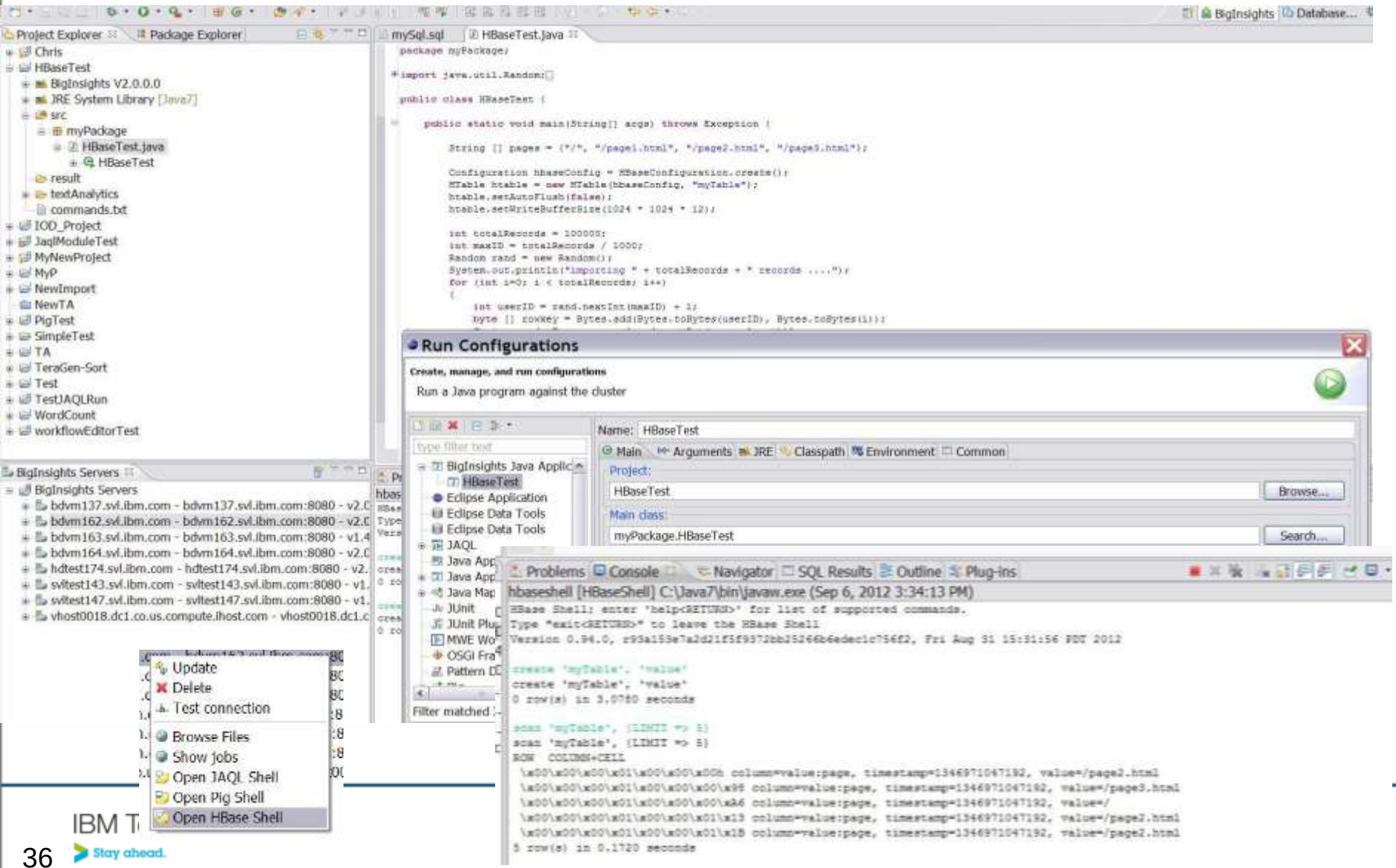
Develop, run, and test SQL



Develop, run, and test Pig



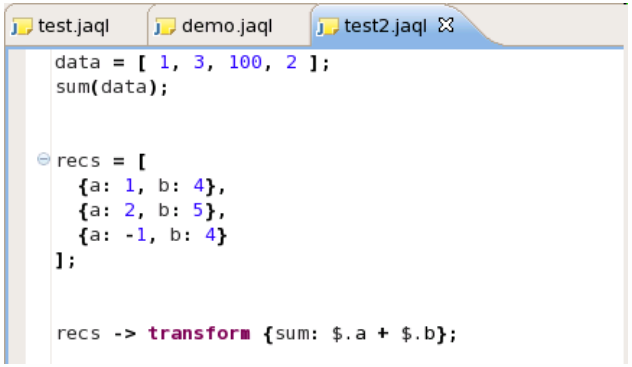
Develop, run, and test HBase

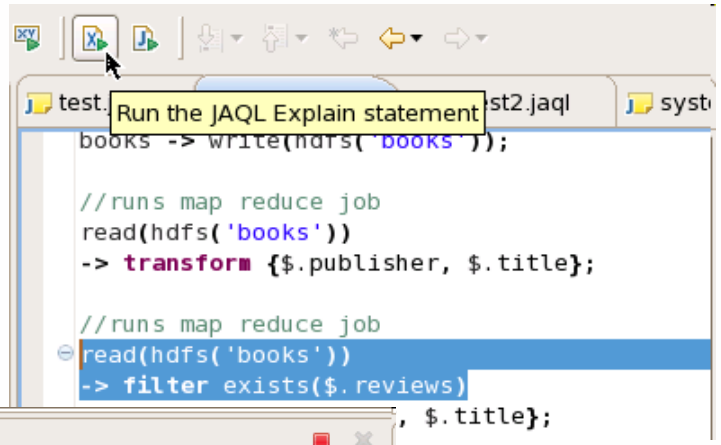


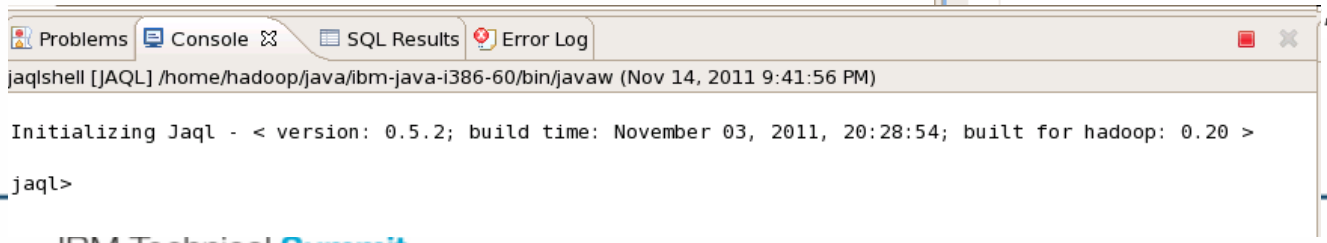
Develop, run, and test Jaql

- Features
  - Syntax highlighting
  - Execute all or selected pieces
  - Explain output
  - Launch Jaqlshell
  - Publish as Application to BI server

Jaql execution/explain is not yet supported on Windows





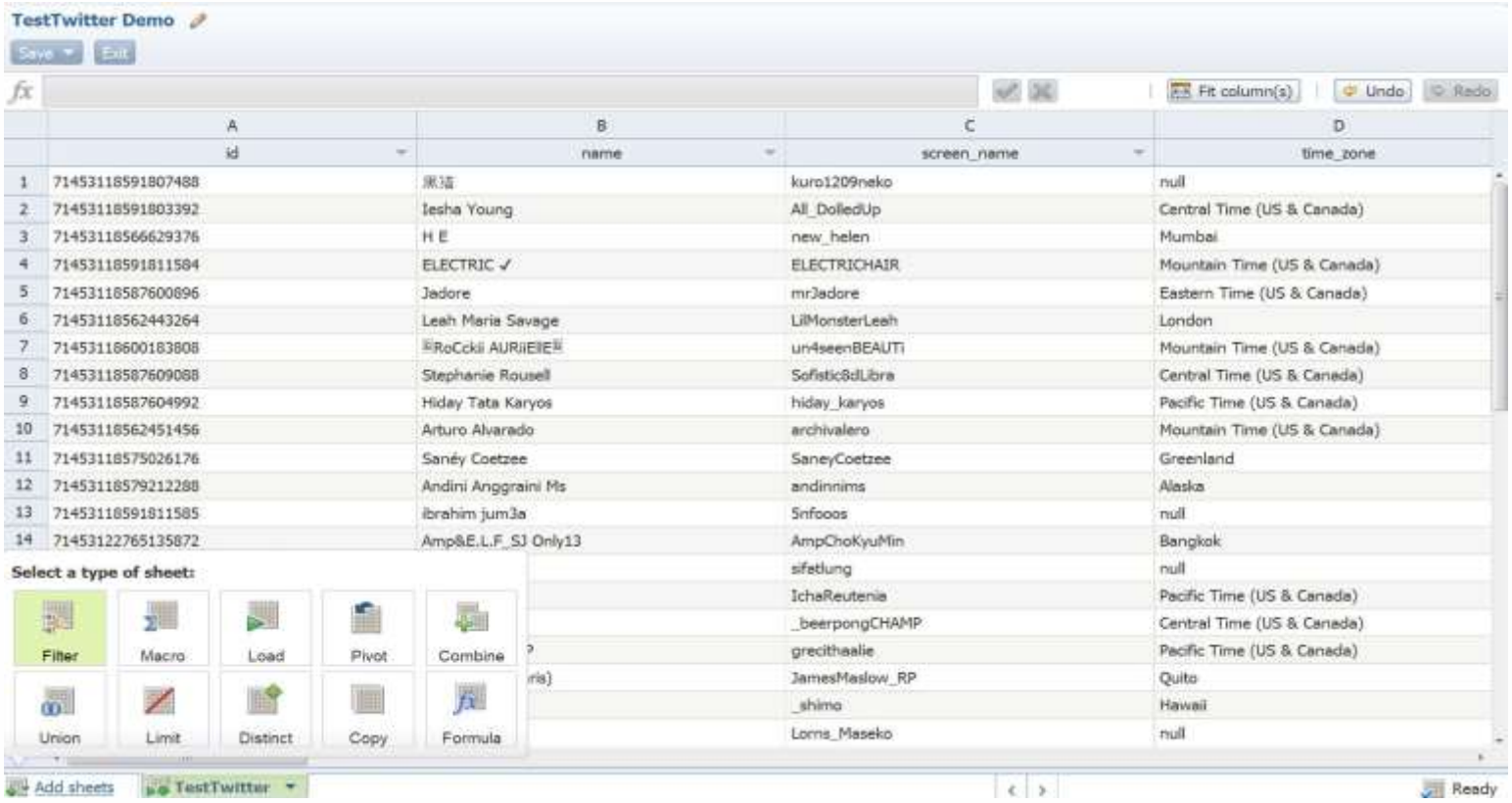


BigSheets - Spreadsheet-style Analytic Tool

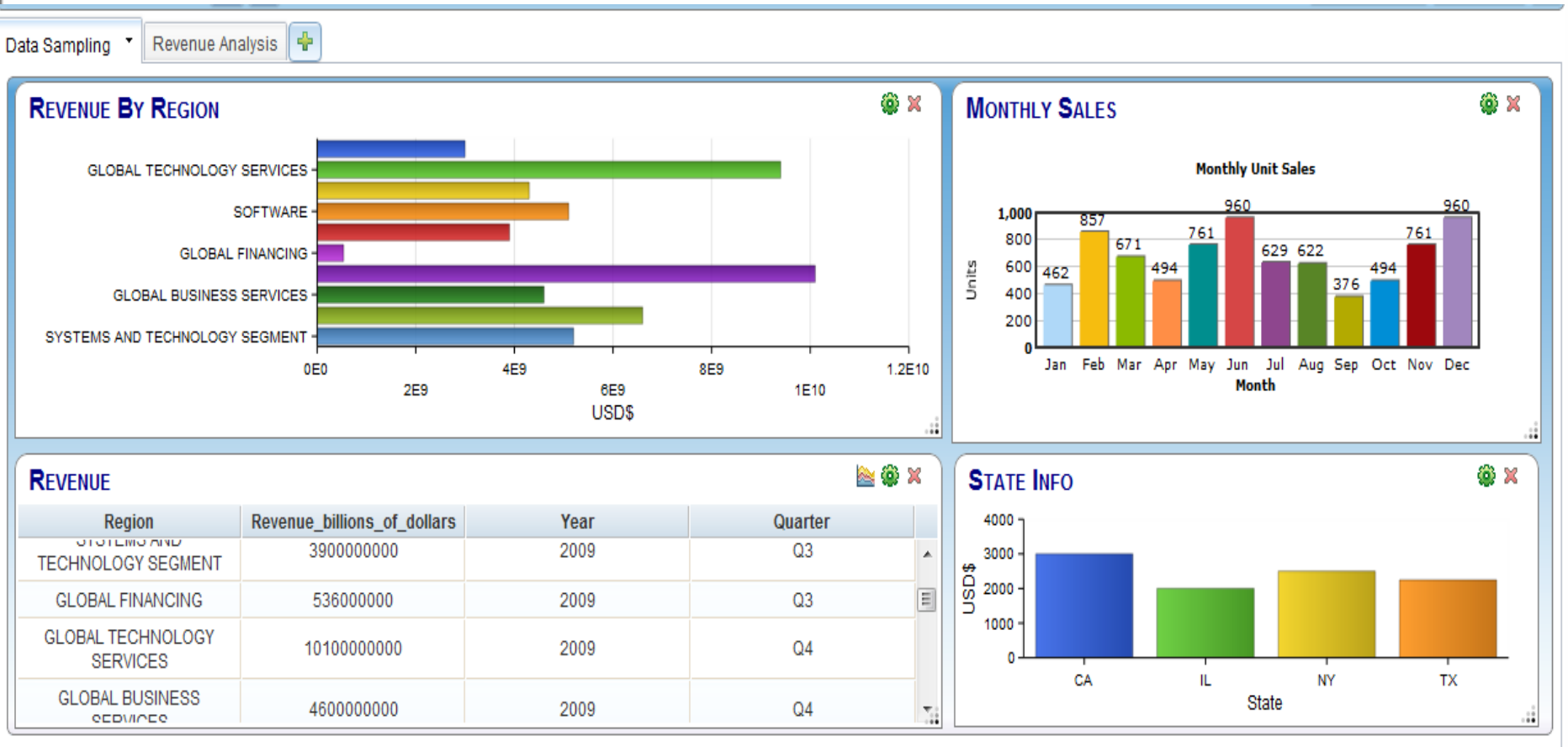
- Model “big data” collected from various sources as *collections* (tabular structures)
- Filter and enrich content with built-in functions
- Combine data in different collections
- Visualize results through spreadsheets, charts
- Export data into common formats (if desired)

No programming knowledge needed!

Spreadsheet-style Data Analysis and Discovery



Visualize results in the customizable dashboard

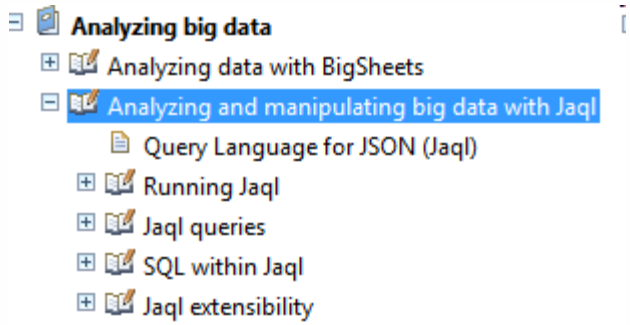


Getting Started

- In the Class Room
  - IBM Education
  - Big Data University
  - https://bigdatauniversity.com
- On Your Cluster
  - BigInsights 2.1 QuickStart Edition VM
  - BigInsights 2.1 Basic Edition
- Stay current
  - BigData on developerWorks
  - http://ibm.co/bigdatadev
  - http://tinyurl.com/biginsights
    - Links to demos, papers, forum, downloads, etc.
- Stay connected with IBM Big Data
  - http://ibmbigdatahub.com

JAQL Resources

- IBM Big Data Youtbje Channel – JAQL playlist
  - Youtube.com/ibmBigData
- developerWorks articles
  - "Query social media and structured data with InfoSphere BigInsights"
  - "Developing, publishing, and deploying your first Big Data application with InfoSphere BigInsights"
- BigInsights 2.1 QuickStart Edition VM
  - JAQL Tutorail
- Big Data University
  - Learn core Hadoop
  - Let me know if you are interested in helping?
- Information Center JAQL Docs




IBM big data • IBM big data • IBM big data
THINK
BIG
IBM big data • IBM big data • IBM big data
IBM big data • IBM big data
IBM big data • IBM big data