SHARE Atlanta 2016

SHARE
EDUCATE • NETWORK • INFLUENCE

z/OS Platform for Apache Spark Install and Usage (Session 19406)

Michael Gildein
megildei@us.ibm.com
@gildmi

Joe Bostian
jbostian@us.ibm.com

Learning Objectives

1. Learn what is Apache Spark and why have Spark on z/OS
2. Learn use cases of z/OS Platform for Apache Spark
3. Learn about the installation and configuration
4. Learn about the ecosystem around Spark

What is Apache Spark?

Apache Spark

Fast

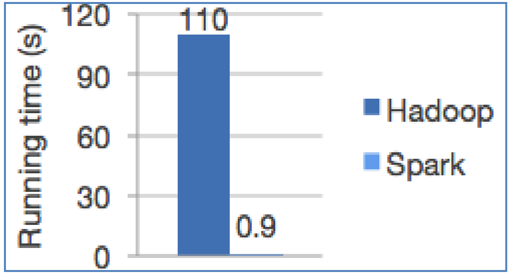
- Leverages aggressively cached in-memory distributed computing
- JVM threads
- Faster than MapReduce for some workloads

Ease of use (for programmers)

- Written in Scala, an object-oriented, functional programming language
- Scala, Python, R, and Java APIs
- Interactive shells: Scala and Python
- Runs on Hadoop, Mesos, standalone, or cloud

General purpose

- Covers a wide range of workloads
- Provides SQL, streaming, and complex analytics



Logistic regression in Hadoop and Spark

Apache Spark is not…

- A data cache or store
- An SQL query optimizer
- A magic analytics solution

Apache Spark

Spark Core

- APIs for working with raw data collections
- Map reduce functions to transform and evaluate – Filter, aggregation, sorting, …

Spark SQL

- APIs for working with structured and semi-structured data
- Provide for relational queries expressed in SQL, HiveQL, and Scala
- Seamlessly mix SQL queries with Spark programs via DataFrames

Spark Streaming

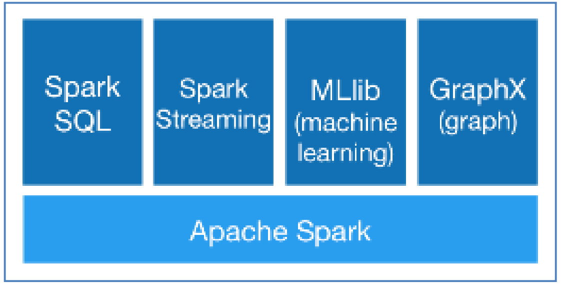
- Streams of data arriving over time
- Fault tolerant and long running tasks
- Integrates with batch processing of data

Machine Learning (MLlib)

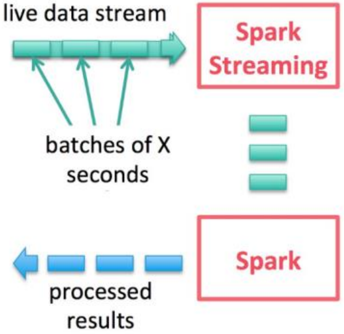
- Efficient, iterative algorithms across distributed datasets
- Focus on parallel algorithms that run well on clusters
- Relatively low-level (e.g. K-means, alternating least squares)

Graph Computation (GraphX)

- View the same data as graph or collection-based
- Transform and join graphs to manipulate data sets
- Graph analysis - PageRank, strongly connected components, ...



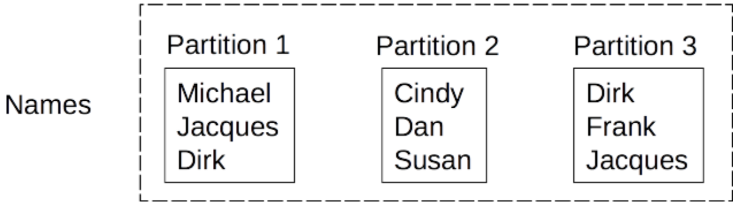
Source: http://spark.apache.org



Spark Processing

Resilient Distributed Datasets (RDDs)

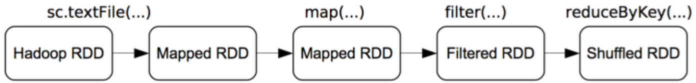
- Spark's basic unit of data
- Immutable
- Fault tolerance
  - If data in memory is lost it will be recreated from lineage
- Caching, persistence (in-memory or disk) – intermediate results or explicit call
- Backed by original data
  - File system, database, etc
- An RDD is physically distributed across the cluster, but manipulated as one logical entity
  - Spark will "distribute" any required processing to all partitions where the RDD exists and perform necessary redistributions and aggregations



Names

Partition 1	Partition 2	Partition 3
Michael	Cindy	Dirk
Jacques	Dan	Frank
Dirk	Susan	Jacques

Spark Processing

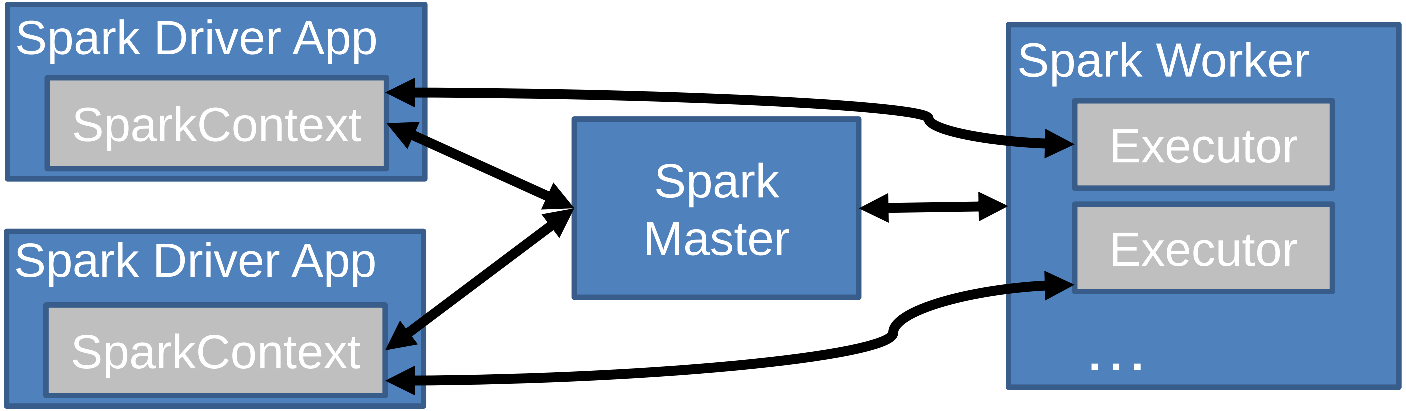


- A Spark application is a sequence of operations on RDDs
  - Transformation: takes an RDD and returns a new RDD (Memory usage grows!)
  - Action: takes an RDD and returns a result

http://creativecommons.org/licenses/by-nc-nd/3.0/

Spark Processing

- A **Driver Program** contains the code that will be executed, for example a Java program
  - This code will establish a **SparkContext**
- Communication is via TCP/IP between the **Driver**, **Master**, and **Worker**
- **Spark Master** manages resources and Driver jobs
- **Executor** JVM is spawned with several threads



Complete your session evaluations online at SHARE.org/Evaluation

z/OS Platform for Apache Spark

z/OS Platform for Apache Spark 1.1.0

Apache Spark

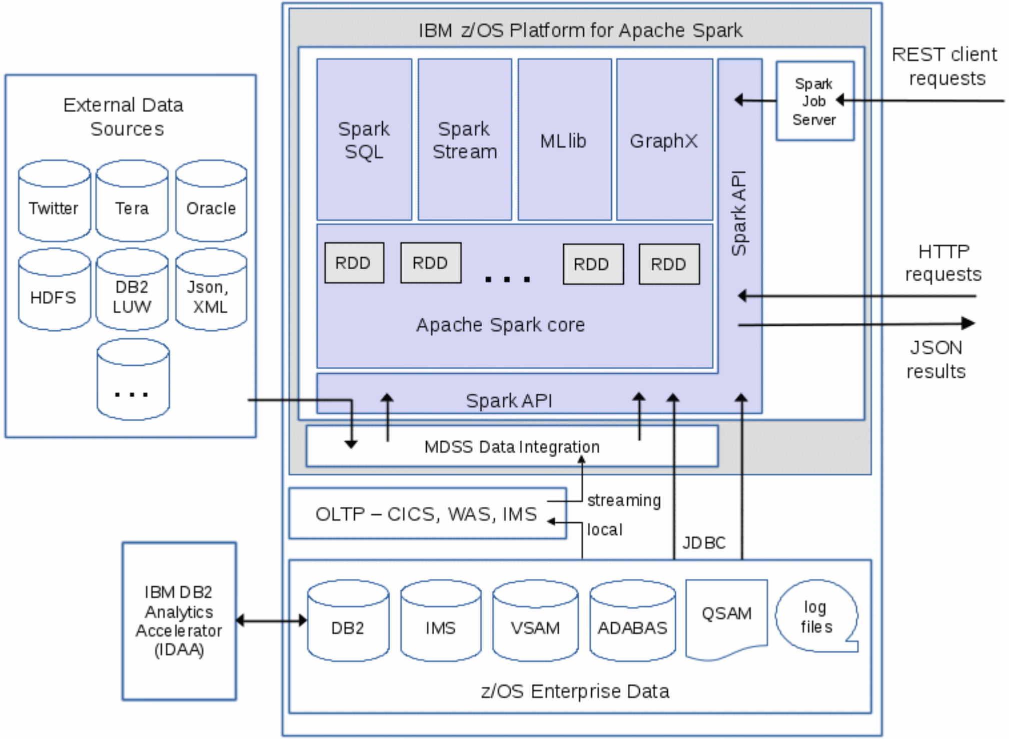
- Apache Spark 1.5.2
  - Spark Core
  - SparkSQL
  - Spark Streaming
  - MLlib
  - GraphX
- Java & Scala driver applications supported
- JVMs
- zIIP-elegible

Mainframe Data Service for Apache Spark (MDSS)

- Single optimized view of heterogeneous data sources
- Intra-SQL and intra-partition parallelism for optimal data access
- Eclipse based Data Studio used to the define virtual data tables and sources
- Mostly zIIP-elegible
- z/OS native and distributed data access
  - DB2
  - SMF (>2000 virtual views)
  - IMS
  - VSAM
  - Physical Sequential
  - Oracle
  - Derby
  - SQL Server
  - MongoDB
  - …

http://creativecommons.org/licenses/by-nc-nd/3.0/

z/OS Platform for Apache Spark



Complete your session evaluations online at SHARE.org/Evaluation

Why on z/OS?

- **Real-time** - Efficient real-time access to current and historical transactions
- **Platform of Record** - Where a majority of data is z/OS resident
- **Security** - When data contains sensitive information
- **Auditing** - Data is not sent through the network to be scattered across several distributed nodes to be held in memory for some unknown period of time
- **Traffic** - Less network traffic moving data around
- **Common** - Implementing common analytic interfaces that are shared with users on distributed platforms
- **Optimization** - IDAA optimization with z/OS DB2 can be integrated with this environment
- **Federate** - Almost any data source from any location can be processed in the Spark environment on z/OS

Why on z/OS?

- **Sysplex** enabled Spark clusters for world class availability
- **SMT2** for added thread performance
- **SIMD** enhances performance on select operations
- **zIIP** eligible to reduce CPU cost
- z/OS's superior **memory** management:
  - RDMA capabilities
  - Large page support
  - Off-heap memory
  - DRAM integration with Flash for scalable elastic memory

10TB

http://creativecommons.org/licenses/by-nc-nd/3.0/

Use Cases

Use Cases

Real Time Data Access

- Any extract, transform, or load (ETL) process takes time
- OLTP transactions happen on z/OS
- On and off platform data accessibility
- Examples
  - Fraud detection
  - Approval predictions

Federated Upsale & Cross Sale Demo
https://youtu.be/sDmWcuO5Rk8

Use Cases

IT Analytics

- z/OS produces a significant volume of operational data
- Analyzing operational data be used to
  - Optimize performance
  - Detect malicious use
  - Detect activity anomalies
- Example data
  - SMF data
  - Logs
  - RMF data

Use Cases

Security

- z/OS is home to confidential and sensitive data
- Data security
  - Data in flight
  - Data at rest
  - Data in use
  - User access
- Tidy data
  - Anonymize
  - Cleanse
  - Summarize
- Examples
  - Patient records
  - Total sales

http://creativecommons.org/licenses/by-nc-nd/3.0/

Use Cases

Volume

- z/OS touches massive volumes of data
- Transporting all of the data off platform takes time
- Transform data
  - Cleanse
  - Filter
- Examples
  - Filter data first with Spark, then write to tidy data cache
    - Sensors (IoT)
    - Entity analysis

z/OS Platform for Apache Spark 1.1.0 Installation

Installation Dependencies

- z/OS V2R1 or above
- IBM 64-bit Java SDK for z/OS Version 8 Service Refresh 2 Fix Pack 10 or above
- Bourne Again Shell (bash) version 4.2.53 or above
  http://www.rocketsoftware.com/ported-tools
- Scala 2.10.4 (For compiling Scala applications on z/OS)

Installation Materials

- Web - Available via try It before you buy! http://www-03.ibm.com/systems/z/os/zos/apache-spark.html
- ShopZ
  - Apache Spark – ZFS file system
  - Mainframe Data Service
- FREE - No license fee
- Support and Service available for purchase

Spark Directory Structure

- **$SPARK_HOME/bin** - Startup scripts for submitting Spark applications
- **$SPARK_HOME/sbin** - Startup shell scripts for Spark processes (master, worker)
- **$SPARK_HOME/conf** - Configuration files for startup scripts to use
- **$SPARK_HOME/logs** - Logs directory
- **$SPARK_HOME/work** - Directory to run applications in, which will include both logs and scratch space
- **metastore_db** - Created where called from; Central repository to store metadata for SparkSQL uses Hive
- **/tmp** - Used for temporary files

**User to start Spark will need read/write access to $SPARK_HOME

.profile

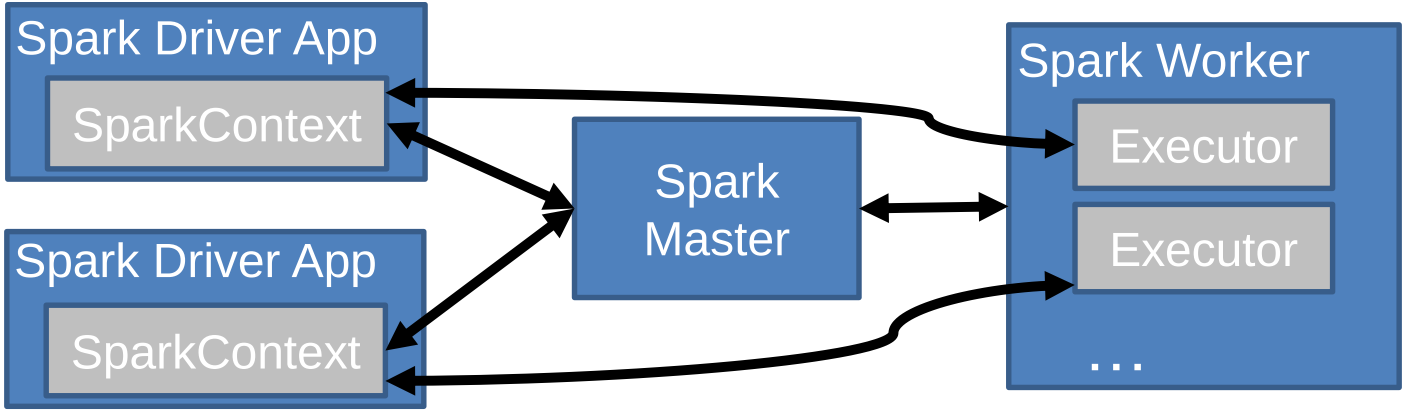
```
JAVA_HOME=/java/J8.0_64
SHELL=/bash/bin
SPARK_HOME= /usr/lpp/IBM/Spark/IBM
PATH=$JAVA_HOME/bin:$SHELL:$PATH:$HOME:$SPARK_HOME/bin
IBM_JAVA_OPTIONS="-Dfile.encoding=ISO8859-1"
_BPXK_AUTOCVT=ON

#EXPORT SETTINGS
export  JAVA_HOME SHELL PATH IBM_JAVA_OPTIONS _BPXK_AUTOCVT SPARK_HOME
```

http://creativecommons.org/licenses/by-nc-nd/3.0/

Spark Processing on z/OS

- $SPARK_HOME/sbin/start-master.sh –h 0.0.0.0 –p 7077
- $SPARK_HOME/sbin/start-slave.sh spark://0.0.0.0:7077
- Address spaces
  - Master; Worker; Each executor; Each Driver (on the system)



Complete your session evaluations online at SHARE.org/Evaluation

z/OS Platform for Apache Spark 1.1.0 Configuration

Configuration

- Apache Spark is **highly** configurable
  https://spark.apache.org/docs/1.5.2/configuration.html
- Application(s) and workload will dictate the required and optimal configurations
- Data volume will dictate the required and optimal configurations
- Configurations will need to change over time as data and apps grow
- Every configuration and instance will be different

Default Configuration

Ports

- Master Web UI:8080
- Worker Web UI:8081+
- Driver Program Web UI:4040+
- REST:6066
- SPARK_WORKER_PORT:8888
- spark.shuffle.service.port:7337

Memory

- Master – 1 GB heap
- Worker – (System Memory - 1GB)
- Executor – 1GB each

CPs

- Worker – All available

Logging

- Settings - $SPARK_HOME/conf/log4j.properties
- Default is **INFO**
- http://logging.apache.org/log4j/1.2/

Folders

- Mostly under $SPARK_HOME
- /tmp
- metastore_db in folder the call was made from

Compression

- spark.io.compression.codec
- LZ4 codec

http://creativecommons.org/licenses/by-nc-nd/3.0/

How much memory do I need?

- As much as possible; Buy more!
- Depends upon
  - Volume of data
  - Number of transformations (new RDDs made)
  - What the app driver does
- RDD persistence
  - Least-recently-used (LRU) to manage cache usage
  - RDD.unpersist() (Effective at next garbage collection?)
- **Test & monitor** closely

- **Example**
  - LPAR 6/30 GB Real Free
  - Data: 4 GB
  - App Driver: 1 GB
  - MDS: 2 GB MEMLIMIT
  - Spark:
    - 1 GB Master
    - 1 GB Worker (w/ 1 GB Executor)

  - *Result == Paging*

http://creativecommons.org/licenses/by-nc-nd/3.0/

Configuration

Ports

- Same as default

Memory

- SPARK_WORKER_MEMORY=#G
  (Desired Total Spark Worker Memory)
  Executors * spark.executor.memory

CPs

- SPARK_WORKER_CORES=#
  (Desired Total Spark Cores)
  Executors * spark.executor.cores

Logging

- $SPARK_HOME/conf/log4j.properties
- Level=WARNING

Folders

- New $SPARK_CONF_DIR

Compression

- Same as default

.profile

```
JAVA_HOME=/java/J8.0_64
SHELL=/bash/bin
SPARK_HOME= /usr/lpp/IBM/Spark/IBM
PATH=$JAVA_HOME/bin:$SHELL:$PATH:$HOME:$SPARK_HOME/bin:$SPARK_HOME/sbin
IBM_JAVA_OPTIONS="-Dfile.encoding=ISO8859-1"
_BPXK_AUTOCVT=ON
_BPX_SHAREAS=NO

#EXPORT SETTINGS
export  JAVA_HOME SHELL PATH IBM_JAVA_OPTIONS _BPXK_AUTOCVT SPARK_HOME _BPX_SHAREAS

#SPARK SETTINGS
export SPARK_CONF_DIR=/home/spark/conf
export HIVE_CONF_DIR=/home/spark/conf
```

http://creativecommons.org/licenses/by-nc-nd/3.0/

$SPARK_CONF_DIR/conf Files

spark-env.sh

```
#!/usr/bin/env bash
SPARK_LOCAL_DIRS= $SPARK_CONF_DIR/tmp
SPARK_LOG_DIR= $SPARK_CONF_DIR/logs
SPARK_PID_DIR= $SPARK_CONF_DIR/tmp
SPARK_WORKER_DIR= $SPARK_CONF_DIR/work
```

**Folders need to exist
& sparkid needs r/w access

hive-site.xml

```
<configuration>
  <property>
     <name>hive.metastore.schema.verification</name>
     <value>false</value>
     <description>
        Turn off version check when creating Derby DB for 1st
time
     </description>
  </property>
  <property>
     <name>javax.jdo.option.ConnectionURL</name>
     <value>jdbc:derby:;databaseName=/home/spark/work/meta
store_db;create=true</value>
     <description>JDBC connect string for a JDBC
metastore</description>
  </property>
</configuration>
```

http://creativecommons.org/licenses/by-nc-nd/3.0/

Tips

- Possible to start multiple stand alone (master w/ 1 worker) instances
  - Define the Spark filesystem as read only
  - Define own CONF, LOCAL, WORK, and LOG dirs
  - Define explicit ports (so that its not default+1) ie. 7**1**77
- Can spawn new instance in the driver
  - $SPARK_HOME/bin/spark-submit --master local[*] --class com.ibm.test ~/sparkapps/fat.jar
- User MEMLIMIT is enforced
  - If the user submits a job and ask for a 2GB JVM but have a 1GB limit, it will fail to instantiate
- MDS has its own MEMLIMIT setting
  - Defaults to 32GB
  - If data exceeds MEMLIMIT size, data is sent in iterations
- Jobs will FIFO queue in Spark if resources are not available
- File system sizes for local dirs, work, and tmp need to be large as data may be written to disk

http://creativecommons.org/licenses/by-nc-nd/3.0/

Ecosystem

The Ecosystem

A set of tools and infrastructure that make analysis from the Spark Cluster accessible to consumers

- Generally run off-platform
- Includes:
  - Jupyter notebook interface for application development
  - Data Studio framework for defining a common data source view
  - Tidy data repository for intermediate results
  - Standalone Scala application development environment
  - User-written REST applications to consume results

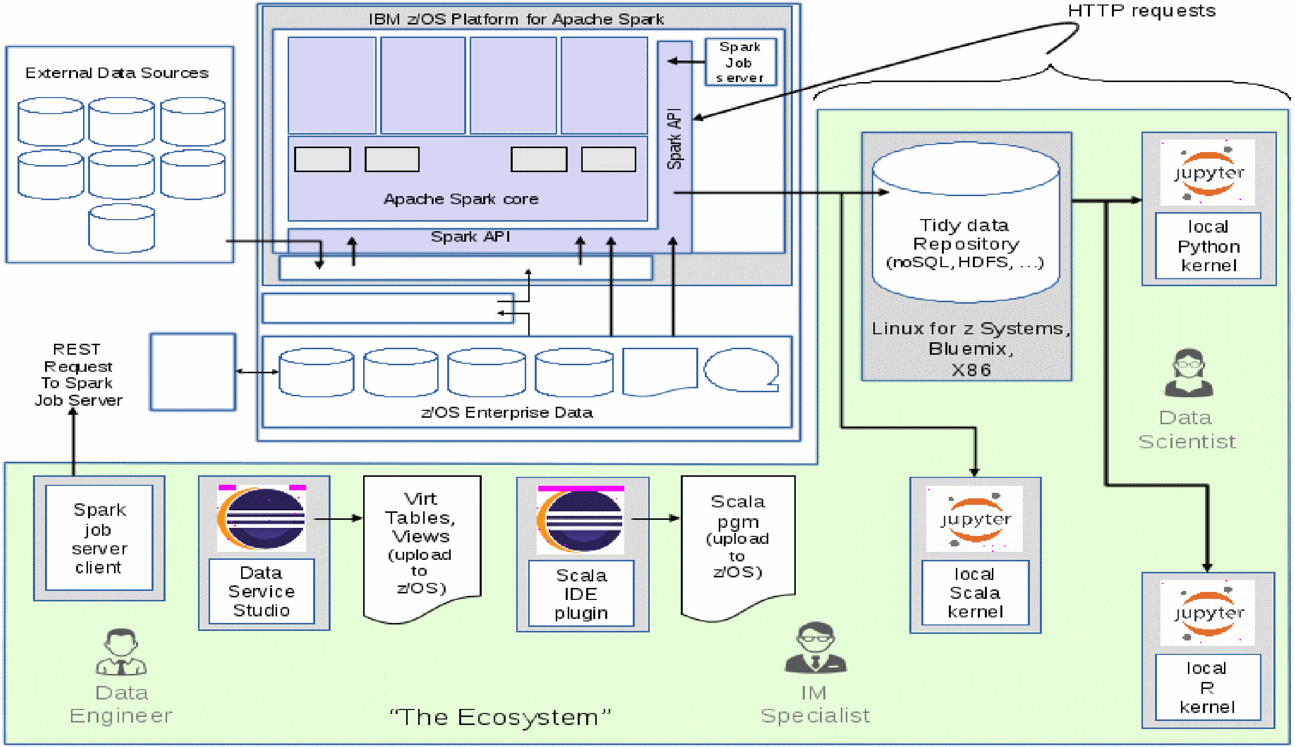
Allows for highly customized use of results

The Audience for Spark on z/OS

Several consumers / customers for the analysis performed on z/OS:

- The **Data Scientist** - the primary customer
  - Creates the spark application(s) that produce insights with business value
  - Probably doesn't know or care where all of the Spark resources come from
- The **Information Management Specialist**
  - Helps the Data scientist assemble and clean the data, write applications
  - Probably better awareness about resource details, but still is primarily concerned with the problem to solve, not the platform
- The **Data Engineer**
  - Also called the “data wrangler”
  - Close to the platform, probably a Z-based person
  - Works with the IM specialist to associate a view of the data with the actual on-platform assets

Ecosystem – Big Picture



Complete your session evaluations online at SHARE.org/Evaluation

Key Parts of the Ecosystem

- Scala Workbench (Jupyter Notebook) interface
  - The primary user (Data Scientist) works here
  - Used to develop applications and illustrate results
  - Once developed, applications can be moved to z/OS & run via spark-submit

- Tidy data repository
  - Catches results reduced from the original data
  - Allows access to results for a large number of consumers
  - Keeps the results in a noSQL form that consumers already recognize

- Data Service Studio
  - Data engineer creates virtual views of heterogeneous data sources
  - Upload maps to z/OS for use by MDS

Key Parts of the Ecosystem ...

- Scala IDE can be used to create standalone Scala applications:
  - Transfer to z/OS for execution via spark-submit
  - Eclipse-based interface
  - Downloadable from the Eclipse marketplace

- Developers may also create REST applications to access Spark through jobserver

- Support for this ecosystem is available at our github site
  - http://zos-spark.github.io/
  - Contains information and installable code at no charge

Jupyter Demo

Brunel Visualization In Jupyter
https://www.youtube.com/watch?v=ceYCrAB6cro&index=5&list=PLBNWh4hu0geOAy9szDzKKHsXwlcq4v8f7

Thank you!

Michael Gildein

megildei@us.ibm.com
@gildmi

Joseph Bostian

jbostian@us.ibm.com

- **z/OS Platform for Apache Spark** http://www-03.ibm.com/systems/z/os/zos/apache-spark.html
- **hackZone2.0 – Spark w/ SMF Challenge!** http://ibm.biz/hackzone
- **Use Case Demo** https://youtu.be/sDmWcuO5Rk8
- **Brunel Visualziation In Jupyter Demo** https://www.youtube.com/watch?v=ceYCrAB6cro&index=5&list=PLBNWh4hu0geOAy9szDzKKHsXwlcq4v8f7
- **Ecosystem of Tools for z/OS Platform for Apache Spark** http://zos-spark.github.io/
- **Apache Spark 1.5.2** http://spark.apache.org/docs/1.5.2/

Visit the SHARE Booth (#303)

Educate

- Learn more about **hot topics** from peers in the **Tech Talk Corner**
- Take part in the **Hack-a-Thon** with IBM & Rocket

Network

- **Engage** with fellow attendees
- Connect with the **zNextGen**® community
- Join the **#SHAREatl** social media conversation

Influence

- Discover ways to **get involved** with SHARE
- Meet **SHARE Volunteers**

http://creativecommons.org/licenses/by-nc-nd/3.0/

Thank You for Attending!
Please remember to complete your evaluation of this session in the SHARE mobile app.

z/OS Platform for Apache Spark
Install and Usage (Session 19406)