

What's New in Apache Spark 2.4?

李潇

深圳 | 2018

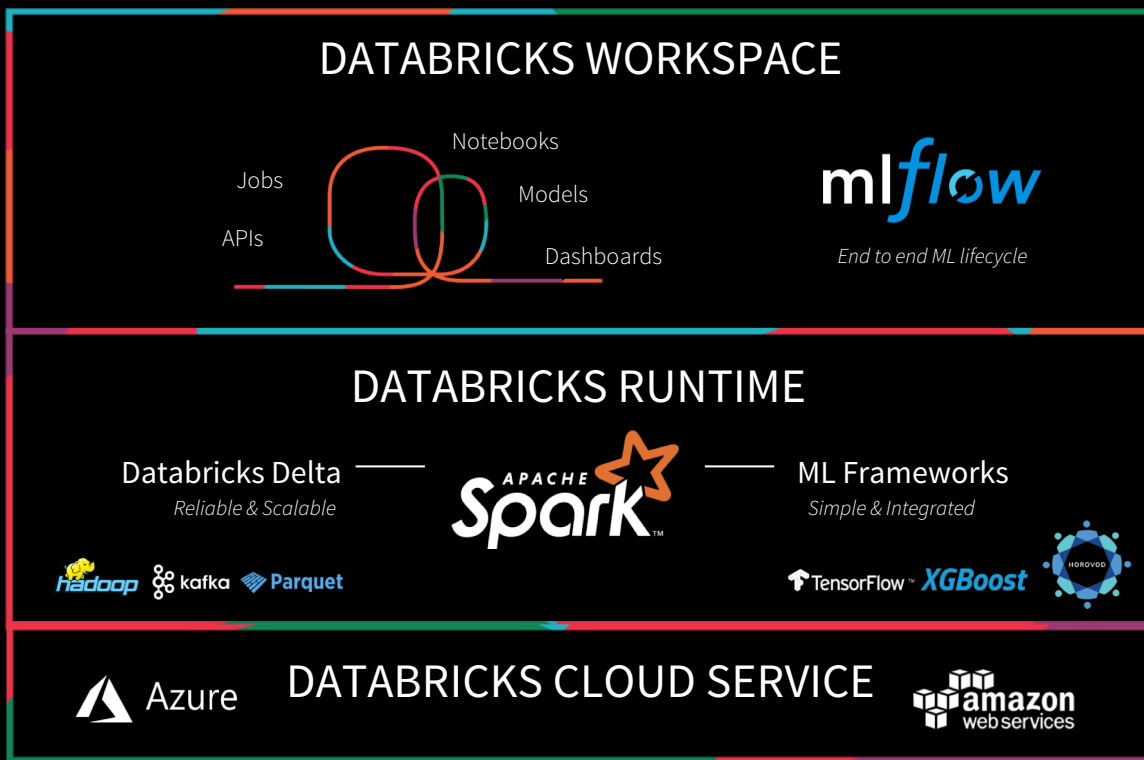


About Me

- Software Engineer at Databricks
- Apache Spark Committer and PMC Member
- Previously, IBM Master Inventor
- Spark SQL, Database Replication, Information Integration
- Ph.D. in University of Florida
- Github: [gatorsmile](https://github.com/gatorsmile)



Databricks Unified Analytics Platform



Databricks Customers Across Industries

Financial Services



Healthcare & Pharma



Media & Entertainment



Data & Analytics Services



Technology



Public Sector



Retail & CPG



Consumer Services



Marketing & AdTech



Energy & Industrial IoT



Introducing Apache Spark 2.4

Now available on Databricks Runtime 5.0



by Wenchen Fan, Xiao Li and Reynold Xin

Posted in **ENGINEERING BLOG** | November 8, 2018

UPDATED: 11/19/2018

We are excited to announce the availability of **Apache Spark 2.4** on Databricks as part of the **Databricks Runtime 5.0**. We want to thank the Apache Spark community for all their valuable contributions to the Spark 2.4 release.

Databricks Runtime Version ⓘ

- 5.0 (includes Apache Spark 2.4.0, Scala 2.11) ⌵
- ✓ 5.0 (includes Apache Spark 2.4.0, Scala 2.11)
- 5.0 ML Beta (Scala 2.11)
- 5.0 (includes Apache Spark 2.4.0, GPU, Scala 2.11)
- 4.3 (includes Apache Spark 2.3.1, Scala 2.11)
- 4.3 (includes Apache Spark 2.3.1, GPU, Scala 2.11)
- 4.2 (includes Apache Spark 2.3.1, Scala 2.11)
- 4.2 (includes Apache Spark 2.3.1, GPU, Scala 2.11)
- 4.1 (includes Apache Spark 2.3.0, Scala 2.11)
- 4.1 ML Beta (includes Apache Spark 2.3.0, Scala 2.11)
- 4.1 ML Beta (includes Apache Spark 2.3.0, GPU, Scala 2.11)
- 4.1 (includes Apache Spark 2.3.0, GPU, Scala 2.11)
- 3.5 LTS (includes Apache Spark 2.2.1, Scala 2.11)
- 3.5 LTS (includes Apache Spark 2.2.1, Scala 2.10)

Release: Nov 8, 2018

Blog: <https://t.co/k7kEHrNZXp>

Above **1100** tickets.

Major Features on Spark 2.4



Barrier Execution



Spark on Kubernetes



Scala 2.12



PySpark Improvement



Structured Streaming



Image Source



Native Avro Support



Built-in source Improvement



Higher-order Functions



Various SQL Features

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



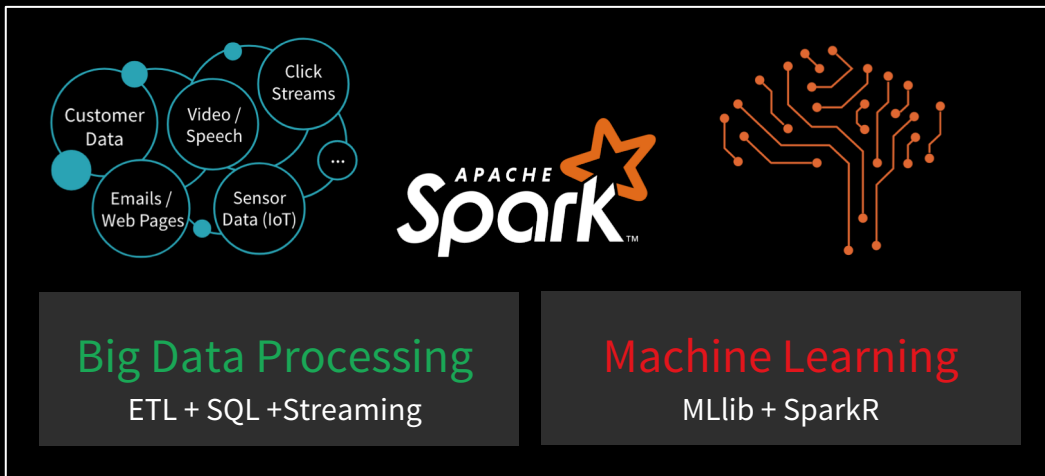
Structured
Streaming



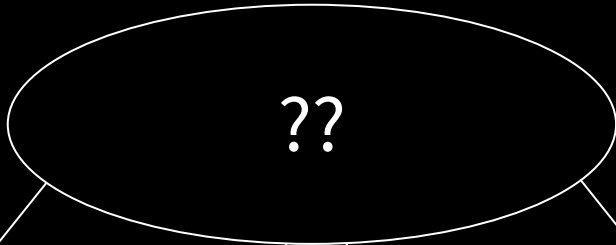
Various SQL
Features

Apache Spark: The First Unified Analytics Engine

Uniquely combines Data & AI technologies



The cross?



Map/Reduce

RDD

Python/Java/R interfaces

DataFrame-based APIs

50+ Data Sources

ML Pipelines API

Project Tungsten

Structured Streaming

Pandas UDF

Continuous Processing

CaffeOnSpark

TensorFlowOnSpark

TensorFrames

tf.data
tf.transform

Horovod

Distributed TensorFlow

Keras

Caffe/PyTorch/MXNet

xgboost

LIBLINEAR

pandas/numpy/scipy

TF XLA

TensorFlow

GraphLab

scikit-learn

AI/ML

glmnet

R

Project Hydrogen: Spark + AI

A **gang scheduling** to Apache Spark that embeds a distributed DL job as a Spark stage to simplify the distributed training workflow.

[\[SPARK-24374\]](#)

- Launch the tasks in a stage at the same time
- Provide enough information and tooling to embed distributed DL workloads
- Introduce a new mechanism of fault tolerance (When any task failed in the middle, Spark shall abort all the tasks and restart the stage)

Project Hydrogen: Spark + AI

Barrier
Execution
Mode

Optimized
Data
Exchange

Accelerator
Aware
Scheduling

Demo: Project Hydrogen:

<https://vimeo.com/274267107> & <https://youtu.be/hwbsWkhBeWM>

Timeline

Spark 2.4

- [[SPARK-24374](#)] barrier execution mode (basic scenarios)
- (Databricks) Horovod integration w/ barrier mode

Spark 2.5/3.0

- [[SPARK-24374](#)] barrier execution mode
- [[SPARK-24579](#)] optimized data exchange
- [[SPARK-24615](#)] accelerator-aware scheduling

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Pandas UDFs

Spark 2.3 introduced vectorized Pandas UDFs that use Pandas to process data. Faster data serialization and execution using **vectorized formats**

- Grouped Aggregate Pandas UDFs
- pandas.Series -> a scalar
- returnType: primitive data type
- [\[SPARK-22274\]](#) [\[SPARK-22239\]](#)

```
>>> from pyspark.sql.functions import pandas_udf, PandasUDFType
>>> df = spark.createDataFrame(
...     [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)],
...     ("id", "v"))
>>> @pandas_udf("double", PandasUDFType.GROUPED_AGG)
... def mean_udf(v):
...     return v.mean()
>>> df.groupby("id").agg(mean_udf(df['v'])).show()
+-----+
| id|mean_udf(v)|
+-----+
| 1|         1.5|
| 2|         6.0|
+-----+

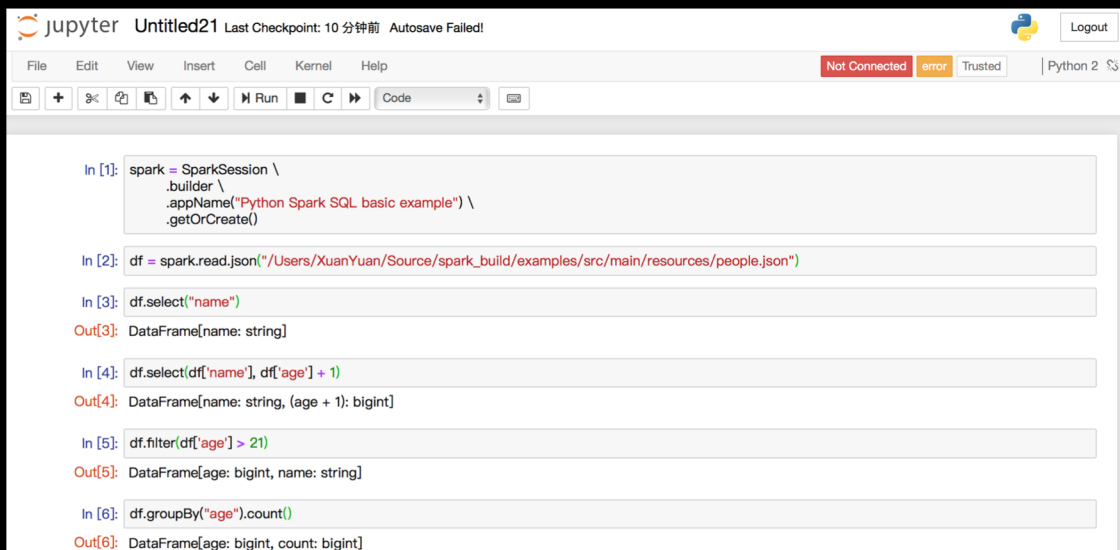
>>> w = Window.partitionBy('id')
>>> df.withColumn('mean_v', mean_udf(df['v']).over(w)).show()
+-----+
| id| v|mean_v|
+-----+
| 1| 1.0|  1.5|
| 1| 2.0|  1.5|
| 2| 3.0|  6.0|
| 2| 5.0|  6.0|
| 2|10.0|  6.0|
+-----+
```

Eager Evaluation

`spark.sql.repl.eagerEval.enabled` [[SPARK-24215](#)]

- Supports DataFrame eager evaluation for PySpark shell and Jupyter.
- When true, the top K rows of Dataset will be displayed.

Prior to 2.4



```
In [1]: spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .getOrCreate()

In [2]: df = spark.read.json("/Users/XuanYuan/Source/spark_build/examples/src/main/resources/people.json")

In [3]: df.select("name")
Out[3]: DataFrame[name: string]

In [4]: df.select(df['name'], df['age'] + 1)
Out[4]: DataFrame[name: string, (age + 1): bigint]

In [5]: df.filter(df['age'] > 21)
Out[5]: DataFrame[age: bigint, name: string]

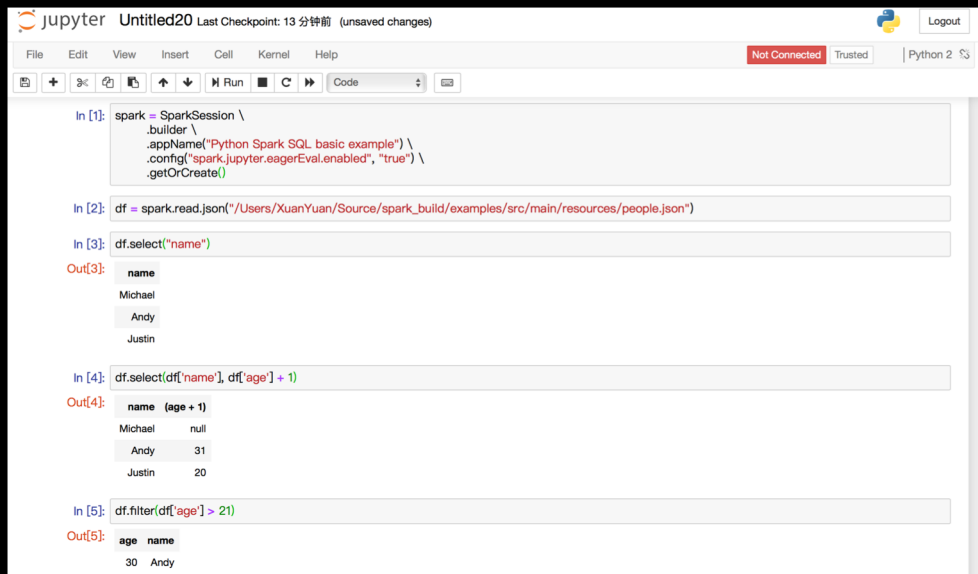
In [6]: df.groupBy("age").count()
Out[6]: DataFrame[age: bigint, count: bigint]
```

Eager Evaluation

`spark.sql.repl.eagerEval.enabled` [[SPARK-24215](#)]

- Supports DataFrame eager evaluation for PySpark shell and Jupyter.
- When true, the top K rows of Dataset will be displayed.

Since 2.4



The screenshot shows a Jupyter notebook interface with the following code and output:

```
In [1]: spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.jupyter.eagerEval.enabled", "true") \
        .getOrCreate()

In [2]: df = spark.read.json("/Users/XuanYuan/Source/spark_build/examples/src/main/resources/people.json")

In [3]: df.select("name")

Out[3]:
+-----+
| name |
+-----+
| Michael |
| Andy |
| Justin |
+-----+

In [4]: df.select(df['name'], df['age'] + 1)

Out[4]:
+-----+-----+
| name | (age + 1) |
+-----+-----+
| Michael | null |
| Andy | 31 |
| Justin | 20 |
+-----+-----+

In [5]: df.filter(df['age'] > 21)

Out[5]:
+----+-----+
| age | name |
+----+-----+
| 30 | Andy |
+----+-----+
```


Other Notable Features

[[SPARK-24396](#)] Add Structured Streaming ForeachWriter for Python

[[SPARK-23030](#)] Use Arrow stream format for creating from and collecting Pandas DataFrames

[[SPARK-24624](#)] Support mixture of Python UDF and Scalar Pandas UDF

[[SPARK-23874](#)] Upgrade Apache Arrow to 0.10.0

- Allow for adding BinaryType support [[ARROW-2141](#)]

[[SPARK-25004](#)] Add `spark.executor.pyspark.memory` limit

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Flexible Streaming Sink

[[SPARK-24565](#)] Exposing output rows of each microbatch as a DataFrame

```
foreachBatch(f: Dataset[T] => Unit)
```

- Scala/Java/Python APIs in DataStreamWriter.
- Reuse existing batch data sources
- Write to multiple locations
- Apply additional DataFrame operations

Reuse existing batch data sources

```
spark.readStream.format("rate").load()
  .selectExpr("value % 10 as key")
  .groupBy("key")
  .count()
  .toDF("key", "value")
  .writeStream
  .foreachBatch { (batchDF: DataFrame, batchId: Long) =>

    batchDF.write      // Use Cassandra batch data source to write streaming out
      .cassandraFormat(tableName, keyspace)
      .option("cluster", clusterName)
      .mode("append")
      .save()
  }
  .outputMode("update")
  .start()
```

Write to multiple location

```
streamingDF.writeStream.foreachBatch { (batchDF: DataFrame, batchId: Long) =>
  batchDF.cache()
  batchDF.write.format(...).save(...) // location 1
  batchDF.write.format(...).save(...) // location 2
  batchDF.uncache()
}
```

Structured Streaming

[[SPARK-24662](#)] Support for the LIMIT operator for streams in Append and Complete output modes.

[[SPARK-24763](#)] Remove redundant key data from value in streaming aggregation

[[SPARK-24156](#)] Faster generation of output results and/or state cleanup with stateful operations (mapGroupsWithState, stream-stream join, streaming aggregation, streaming dropDuplicates) when there is no data in the input stream.

[[SPARK-24730](#)] Support for choosing either the min or max watermark when there are multiple input streams in a query.

Kafka Client 2.0.0

[[SPARK-18057](#)] Upgraded Kafka client version from 0.10.0.1 to 2.0.0

[[SPARK-25005](#)] Support “kafka.isolation.level” to read only committed records from Kafka topics that are written using a transactional producer.

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

AVRO



- Apache Avro (<https://avro.apache.org>)
 - A data serialization format
 - Widely used in the Spark and Hadoop ecosystem, especially for Kafka-based data pipelines.
- Spark-Avro package (<https://github.com/databricks/spark-avro>)
 - Spark SQL can read and write the avro data.
- Inlining Spark-Avro package [[SPARK-24768](#)]
 - Better experience for first-time users of Spark SQL and structured streaming
 - Expect further improve the adoption of structured streaming

AVRO



[[SPARK-24811](#)] `from_avro/to_avro` functions to read and write Avro data within a DataFrame instead of just files.

Example:

1. Decode the Avro data into a struct
2. Filter by column `favorite_color`
3. Encode the column `name` in Avro format

```
// Note: `from_avro` requires Avro schema in JSON string format.
val jsonFormatSchema = new String(Files.readAllBytes(
  Paths.get( first = "./examples/src/main/resources/user.avsc")))

val df = spark
  .readStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()

val output = df
  .select(from_avro( data = 'value, jsonFormatSchema) as 'user)
  .where( conditionExpr = "user.favorite_color == \"red\"")
  .select(to_avro( data = $"user.name") as 'value)

val query = output
  .writeStream
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("topic", "topic2")
  .start()
```

AVRO Performance



[[SPARK-24800](#)] Refactor Avro Serializer and Deserializer

External library

AVRO Data >> Row >> InternalRow

AVRO Data << Row << InternalRow

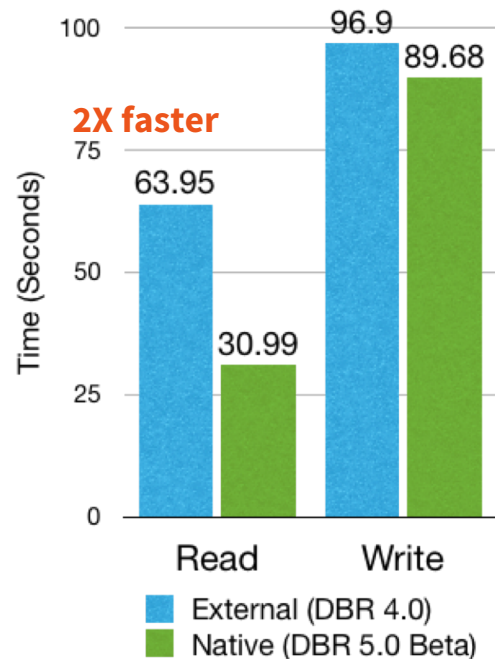
Native reader

AVRO Data >>> InternalRow

AVRO Data <<< InternalRow

Notebook: <https://dbricks.co/AvroPerf>

Runtime comparison (Lower is better)



AVRO Logical Types



Avro upgrade from 1.7.7 to 1.8.

[\[SPARK-24771\]](#)

Logical type support:

- Date [\[SPARK-24772\]](#)
- Decimal [\[SPARK-24774\]](#)
- Timestamp [\[SPARK-24773\]](#)

Options:

- compression
- ignoreExtension
- recordNamespace
- recordName
- avroSchema

Blog: Apache Avro as a Built-in Data Source in Apache Spark 2.4.

<https://t.co/jks7j27PxJ>

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Image schema data source

[\[SPARK-22666\]](#) Spark datasource for image format

- Partition discovery [**new**]
- Loading recursively from directory [**new**]
- dropImageFailures path wildcard matching
- Path wildcard matching

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Analytic Database



- 30+ PB data
- 5,000 tables
- 20,000 temporary tables
- 50,000 jobs



- 1,800+ nodes per day
- 15,000 jobs



Parquet



Update from 1.8.2 to 1.10.0 [[SPARK-23972](#)].

- [PARQUET-1025](#) - Support new min-max statistics in parquet-mr
- [PARQUET-225](#) - INT64 support for delta encoding
- [PARQUET-1142](#) Enable `parquet.filter.dictionary.enabled` by default.

Predicate pushdown

- STRING [[SPARK-23972](#)] [**20x faster**]
- Decimal [[SPARK-24549](#)]
- Timestamp [[SPARK-24718](#)]
- Date [[SPARK-23727](#)]
- Byte/Short [[SPARK-24706](#)]
- StringStartsWith [[SPARK-24638](#)]
- IN [[SPARK-17091](#)]

ORC



Native vectorized ORC reader is GAed!

- Native ORC reader is on by default [[SPARK-23456](#)]
- Update ORC from 1.4.1 to 1.5.2 [[SPARK-24576](#)]
- Turn on ORC filter push-down by default [[SPARK-21783](#)]
- Use native ORC reader to read Hive serde tables by default [[SPARK-22279](#)]
- Avoid creating reader for all ORC files [[SPARK-25126](#)]

CSV

- Option `samplingRatio` for schema inference [[SPARK-23846](#)]
- Option `enforceSchema` for throwing an exception when user-specified schema doesn't match the CSV header [[SPARK-23786](#)]
- Option `encoding` for specifying the encoding of outputs. [[SPARK-19018](#)]

Performance:

- Parsing only required columns to the CSV parser [[SPARK-24244](#)]
- Speed up `count()` for JSON and CSV [[SPARK-24959](#)]
- Better performance by switching to uniVocity 2.7.3 [[SPARK-24945](#)]

JSON

- Option **encoding** for specifying the encoding of inputs and outputs. [[SPARK-23723](#)]
- Option **dropFieldIfAllNull** for ignoring column of all null values or empty array/struct during JSON schema inference [[SPARK-23772](#)]
- Option **lineSep** for defining the line separator that should be used for parsing [[SPARK-23765](#)]
- Speed up count() for JSON and CSV [[SPARK-24959](#)]

JDBC

- Option `queryTimeout` for the number of seconds the the driver will wait for a Statement object to execute. [[SPARK-23856](#)]
- Option `query` for specifying the query to read from JDBC [[SPARK-24423](#)]
- Option `pushDownFilters` for specifying whether the filter pushdown is allowed [[SPARK-24288](#)]
- Auto-correction of partition column names [[SPARK-24327](#)]
- Support `Date/Timestamp` in a JDBC partition column when reading in parallel from multiple workers. [[SPARK-22814](#)]
- Add `cascadeTruncate` option to JDBC datasource [[SPARK-22880](#)]

Major Features on Upcoming Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Higher-order Functions

Transformation on complex objects like **arrays**, **maps** and **structures** inside of columns.

```
tbl_nested
```

```
|-- key: long (nullable = false)
```

```
|-- values: array (nullable = false)
```

```
|   |-- element: long (containsNull = false)
```

UDF ?

Expensive data serialization

Higher-order Functions

1) Check for element existence

```
SELECT EXISTS(values, e -> e > 30) AS v  
FROM tbl_nested;
```

tbl_nested

|-- key: long (nullable = false)

|-- values: array (nullable = false)

| |-- element: long (containsNull = false)

2) Transform an array

```
SELECT TRANSFORM(values, e -> e * e) AS v  
FROM tbl_nested;
```


Higher-order Functions

3) Filter an array

```
SELECT FILTER(values, e -> e > 30) AS v  
FROM tbl_nested;
```

tbl_nested

|-- key: long (nullable = false)

|-- values: array (nullable = false)

| |-- element: long (containsNull = false)

4) Aggregate an array

```
SELECT REDUCE(values, 0, (value, acc) -> value + acc) AS sum  
FROM tbl_nested;
```

Ref Databricks Blog: <http://dbricks.co/2rUKQ1A>

Built-in Functions

[[SPARK-23899](#)] New or extended built-in functions for ArrayTypes and MapTypes

- 26 functions for ArrayTypes
transform, filter, reduce, array_distinct, array_intersect, array_union, array_except, array_join, array_max, array_min, ...
- 3 functions for MapTypes
map_from_arrays, map_from_entries, map_concat

Blog: Introducing New Built-in and Higher-Order Functions for Complex Data Types in Apache Spark 2.4. <https://t.co/p1TRRtabJJ>

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

When having many columns/functions

[[SPARK-16406](#)] Analyzer: Improve performance of LogicalPlan.resolve

Add an indexing structure to resolve(...) in order to find potential matches quicker.

[[SPARK-23963](#)] Properly handle large number of columns in query on text-based Hive table

Turns a list to array, makes a hive table scan 10 times faster when there are a lot of columns.

[[SPARK-23486](#)] Analyzer: Cache the function name from the external catalog for lookupFunctions

Optimizer/Planner

[[SPARK-23803](#)] Support Bucket Pruning

Prune buckets that cannot satisfy equal-to predicates, to reduce the number of files to scan.

[[SPARK-24802](#)] Optimization Rule Exclusion

Disable a list of optimization rules in the optimizer

[[SPARK-4502](#)] Nested schema pruning for Parquet tables.

Column pruning on nested fields.

More: [[SPARK-24339](#)] [[SPARK-23877](#)] [[SPARK-23957](#)] [[SPARK-25212](#)] ...

SQL API Enhancement

[[SPARK-24940](#)] Coalesce and Repartition Hint for SQL Queries

```
INSERT OVERWRITE TABLE targetTable  
SELECT /*+ REPARTITION(10) */ *  
FROM sourceTable
```

[[SPARK-19602](#)] Support column resolution of fully qualified column name (3 part name). (i.e., \$DBNAME.\$TABLENAME.\$COLUMNNAME)

```
SELECT * FROM db1.t3  
WHERE c1 IN (SELECT db1.t4.c2 FROM  
db1.t4 WHERE db1.t4.c3 = db1.t3.c2)
```

More: INTERSECT ALL,
EXCEPT ALL, Pivot,
GROUPING SET,
precedence rules for set
operations and ...

[[SPARK-21274](#)] [[SPARK-24035](#)] [[SPARK-24424](#)]
[[SPARK-24966](#)]

Blog: SQL Pivot: Converting
Rows to Columns in Apache
Spark 2.4

<https://t.co/AgGKOcl2N4>

Other Notable Changes in SQL

[\[SPARK-24596\]](#) Non-cascading Cache Invalidation

- Non-cascading mode for temporary views and `DataSet.unpersist()`
- Cascading mode for the rest

[\[SPARK-23880\]](#) Do not trigger any job for caching data

[\[SPARK-23510\]](#)[\[SPARK-24312\]](#) Support Hive 2.2 and Hive 2.3 metastore

[\[SPARK-23711\]](#) Add fallback generator for UnsafeProjection

[\[SPARK-24626\]](#) Parallelize location size calculation in Analyze Table command

Other Notable Features in Core

[[SPARK-23243](#)] Fix RDD.repartition() data correctness issue

[[SPARK-24296](#)] Support replicating blocks larger than 2 GB

[[SPARK-24307](#)] Support sending messages over 2GB from memory

Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

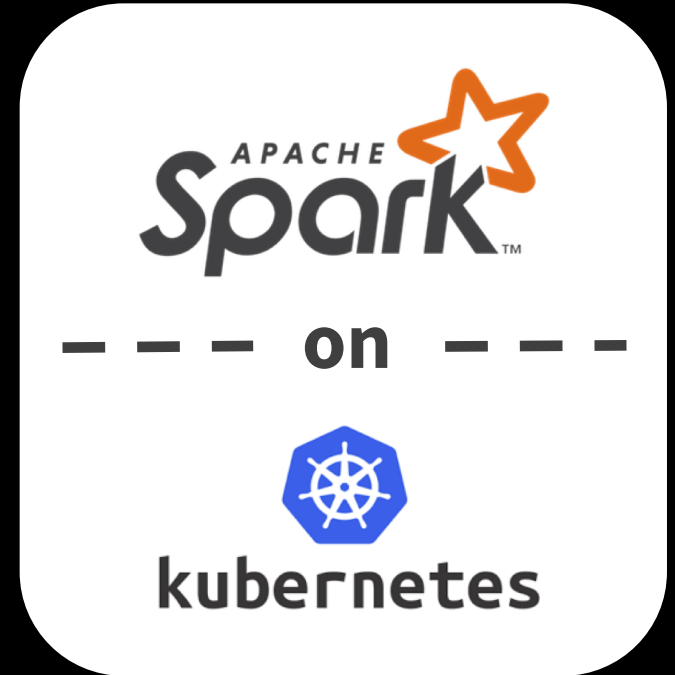
Native Spark App in K8S

New Spark scheduler backend

- PySpark support [[SPARK-23984](#)]
- SparkR support [[SPARK-24433](#)]
- Client-mode support [[SPARK-23146](#)]
- Support for mounting K8S volumes [[SPARK-23529](#)]

Blog: What's New for Apache Spark on Kubernetes in the Upcoming Apache Spark 2.4 Release

<https://t.co/uUpdUj2Z4B>



Major Features on Spark 2.4



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Scala 2.12 Beta Support

[[SPARK-14220](#)] Build Spark against Scala 2.12

All the tests PASS! <https://dbricks.co/Scala212Jenkins>



Databricks Runtime 5.0



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
Improvement



PySpark
Improvement



Higher-order
Functions



Scala
2.12



Structured
Streaming



Various SQL
Features

Databricks Runtime 5.0



Barrier
Execution



Spark on
Kubernetes



Native Avro
Support



Image
Source



Built-in source
improvement



PySpark
Improvement



Higher-order
functions



Scala
2.12



Structured
Streaming



Various SQL
Features

AVAILABLE NOW

Try Databricks Runtime 5.0 For Free



<https://databricks.com/try-databricks>





SPARK+AI SUMMIT 2019

APRIL 23 - 25 | SAN FRANCISCO

ORGANIZED BY  databricks

Save \$700 with special
year end pricing

**REGISTER TODAY!
OFFER ENDS DEC 31ST.**

Nike: Enabling Data Scientists to bring their Models to Market

Facebook: Vectorized Query Execution in Apache Spark at Facebook

Tencent: Large-scale Malicious Domain Detection with Spark AI

IBM: In-memory storage Evolution in Apache Spark

Capital One: Apache Spark and Sights at Speed: Streaming, Feature management and Execution

Apple: Making Nested Columns as First Citizen in Apache Spark SQL

EBay: Managing Apache Spark workload and automatic optimizing.

Google: Validating Spark ML Jobs

HP: Apache Spark for Cyber Security in big company

Microsoft: Apache Spark Serving: Unifying Batch, Streaming and RESTful Serving

ABSA Group: A Mainframe Data Source for Spark SQL and Streaming

Facebook: an efficient Facebook-scale shuffle service

IBM: Make your PySpark Data Fly with Arrow!

Facebook: Distributed Scheduling Framework for Apache Spark

Zynga: Automating Predictive Modeling at Zynga with PySpark

World Bank: Using Crowdsourced Images to Create Image Recognition Models and NLP to Augment Global Trade indicator

JD.com: Optimizing Performance and Computing Resource.

Microsoft: Azure Databricks with R: Deep Dive

ICL: Cooperative Task Execution for Apache Spark



Airbnb: Apache Spark at Airbnb

Netflix: Migrating to Apache Spark at Netflix

Microsoft: Infrastructure for Deep Learning in Apache Spark

Intel: Game playing using AI on Apache Spark

Facebook: Scaling Apache Spark @ Facebook

Lyft: Scaling Apache Spark on K8S at Lyft

Uber: Using Spark Mllib Models in a Production Training and Serving Platform

Apple: Bridging the gap between Datasets and DataFrames

Salesforce: The Rule of 10,000 Spark Jobs

Target: Lessons in Linear Algebra at Scale with Apache Spark

Workday: Lesson Learned Using Apache Spark

