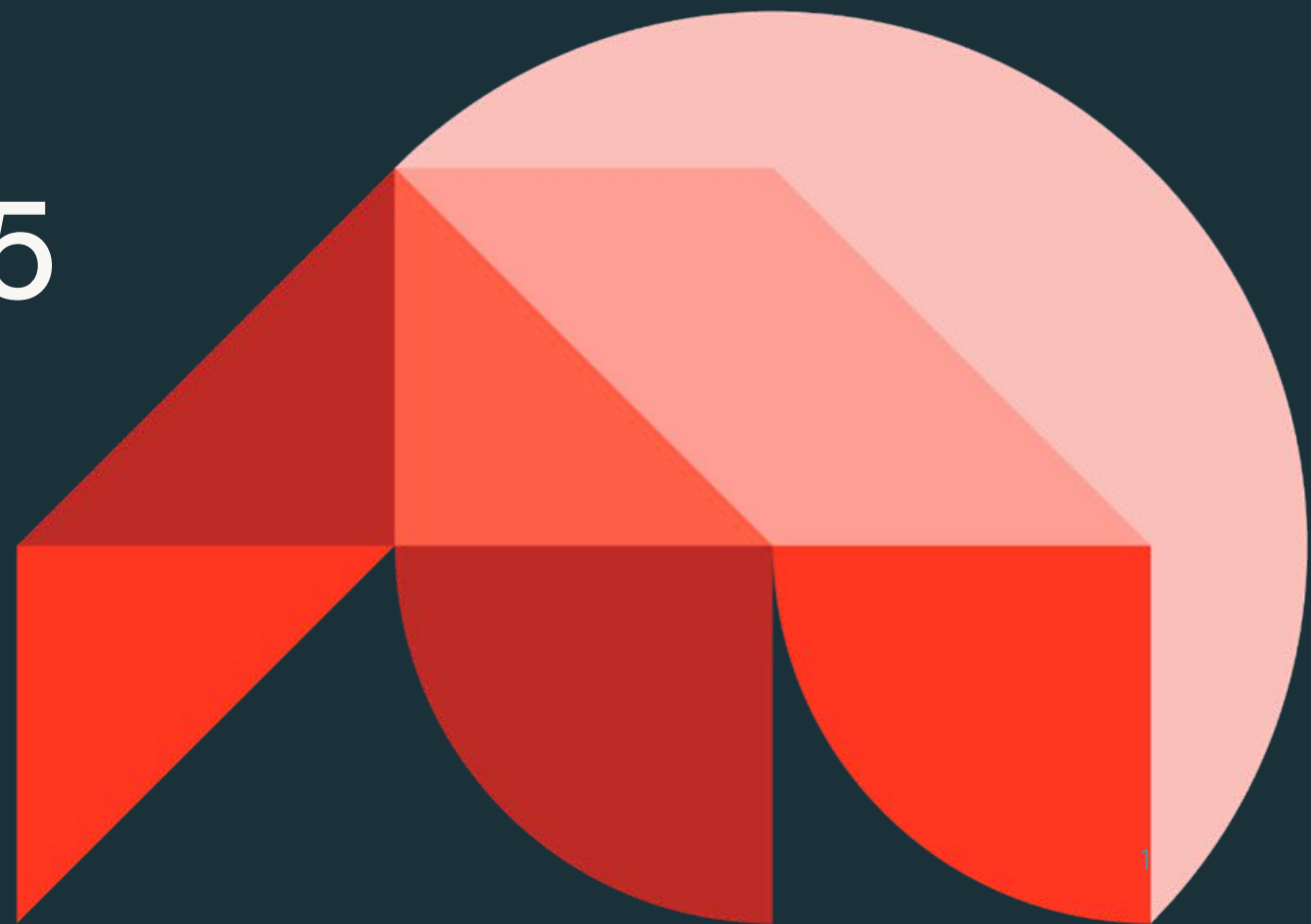




Explore the New Functionality of Apache Spark 3.5

Daniel Tenedor  dtenedor

Data + AI Summit 2024





APACHE Spark™

Transforming and Querying
Data for Everyone!





1+ Billion

Annual Downloads



100K+

Stack Overflow Questions



3700+

GitHub Contributors



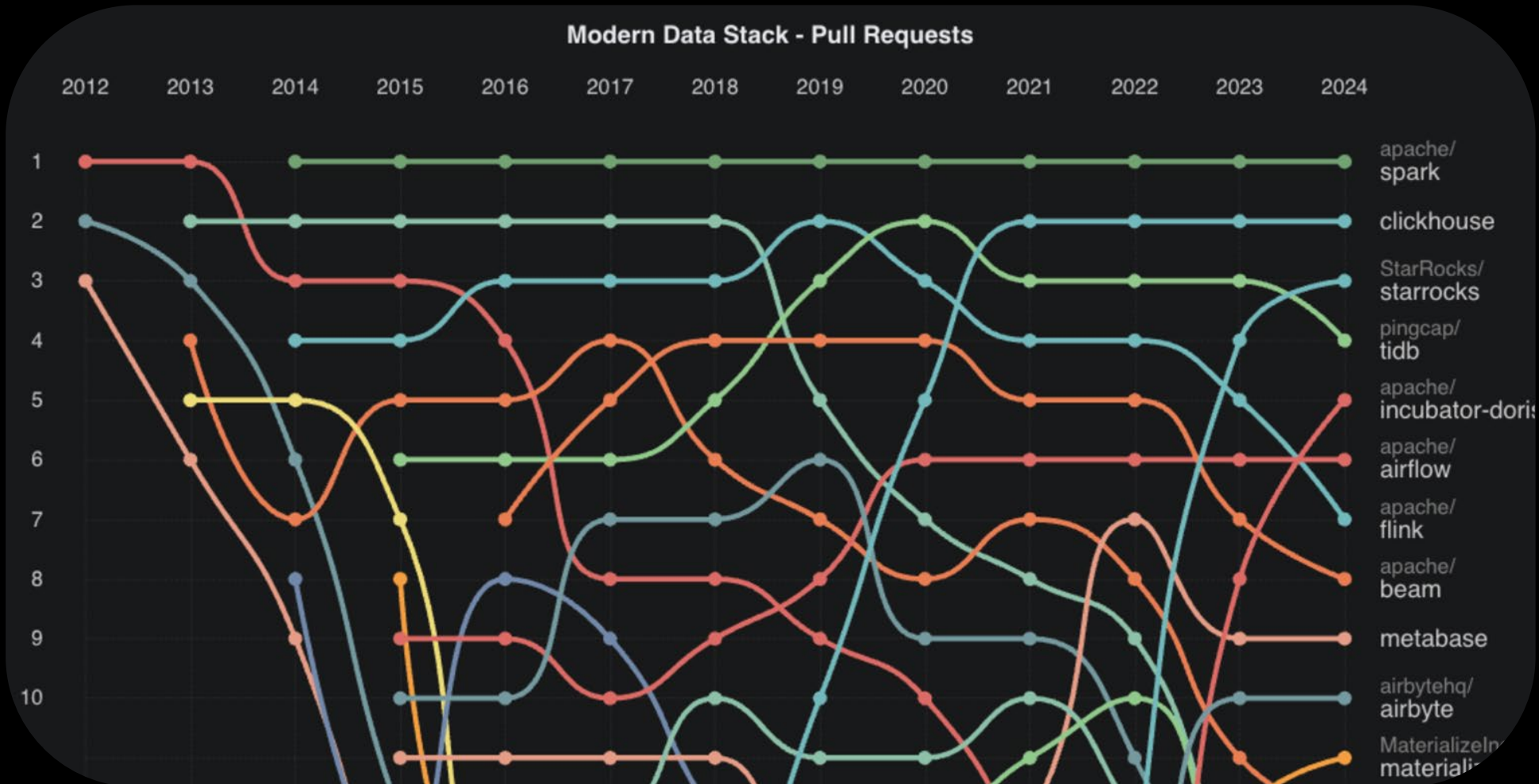
100+

Data Sources

41K+

Commits

3,700 contributors, 41,000 commits
Still #1 in developer activity for over ten years!





Berkeley

UNIVERSITY OF CALIFORNIA

Spark Connect



Scala Client



Go Client



Structured Streaming



Distributed training & inference

SQL



SQL IDENTIFIER



Built-in Functions



HyperLogLog



Named Arguments

Python



Arrow optimized Python UDF



Python UDTF



Scala/Python Functions



PySpark Testing APIs

Features



DeepSpeed Distributor



pandas APIs for Spark Connect



AQE support for SQL Cache



Decommission Enhancements

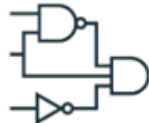
Streaming



Changelog Checkpointing



Stateful Operator Chaining



dropDuplicates WithinWatermark



Memory Management in State Store

More



Error Class Enhancements



Java 17 & Scala 2.13



Spark UI for Spark Connect



DSV2

Agenda



Spark Connect

Deploy and update Spark clusters independently from their clients



SQL Features

HyperLogLog aggregates based on Apache Dataskeches, array manipulation functions, IDENTIFIER clause, and more



PySpark Features

Arrow-optimized Python UDFs, Python UDTFs, new testing API, improved error messages, and more



Spark Streaming

Support multiple stateful operators, checkpointing for RocksDB state store, dropDuplicatesWithinWatermark



Scala Client



Go Client



Structured
Streaming

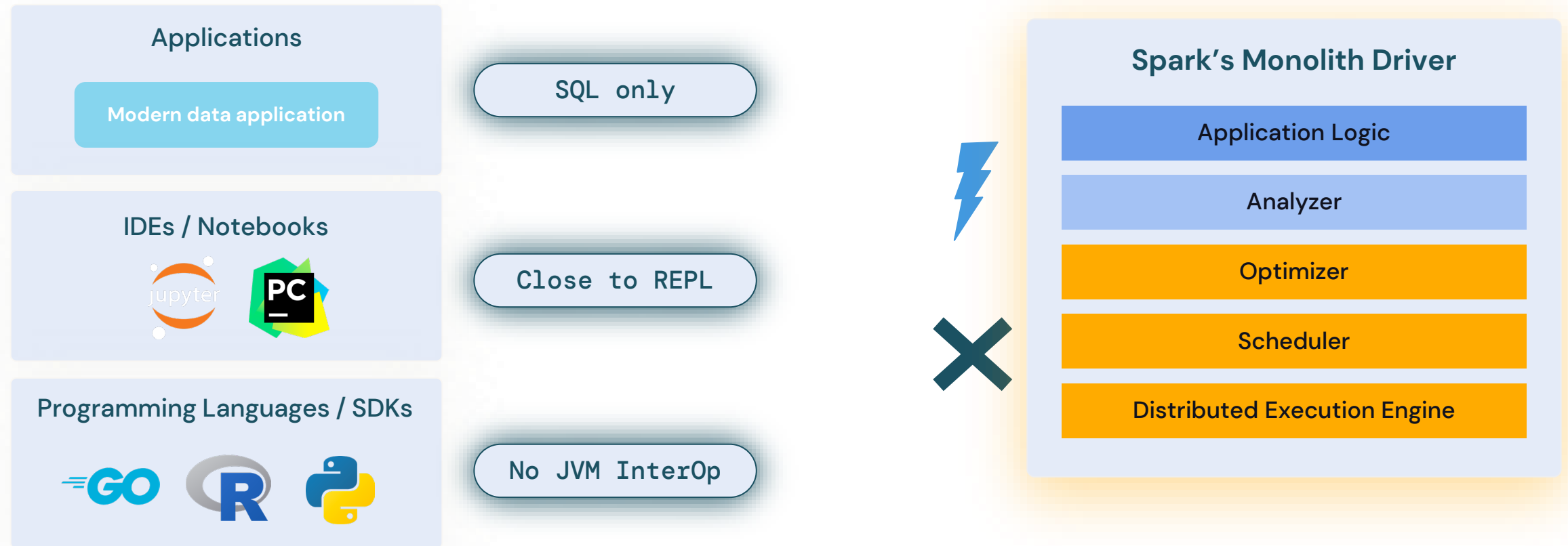


Distributed training
& inference

Spark Connect

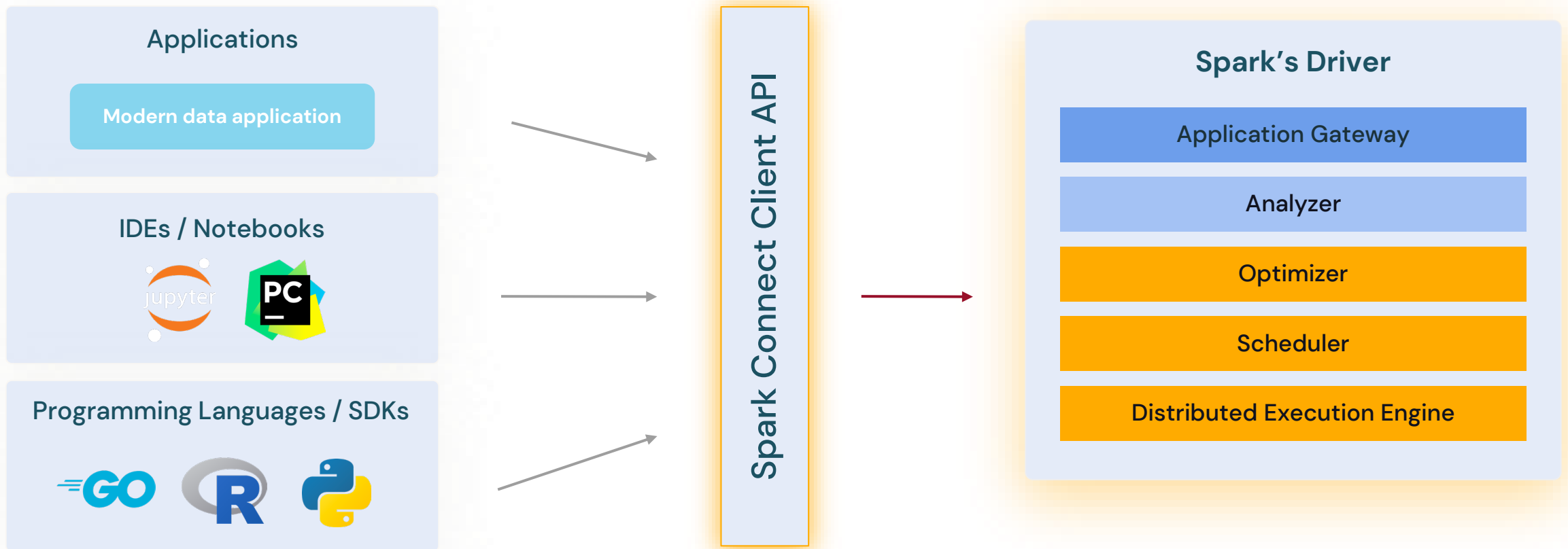
How to embed Spark in applications?

Up until Spark Connect: Hard to support today's developer experience requirements



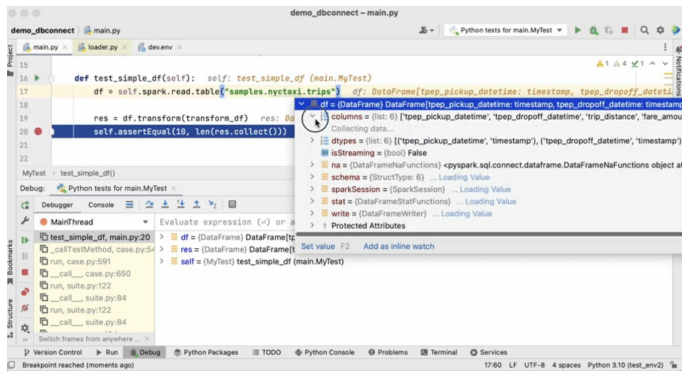
Spark Connect General Availability

Thin client, with full power of Apache Spark



Connect to Spark from Any Application

Interactively develop & debug from your IDE



`pip install pyspark>=3.5.0`

in your favorite IDE!

©2024 Databricks Inc. — All rights reserved

New Connectors and SDKs in any language!



Databricks
Connect



Check out [Databricks Connect](#)
use & contribute the [Go](#) client

Build interactive Data Applications

NYC Taxi Cockpit: Plotly x Databricks Demo

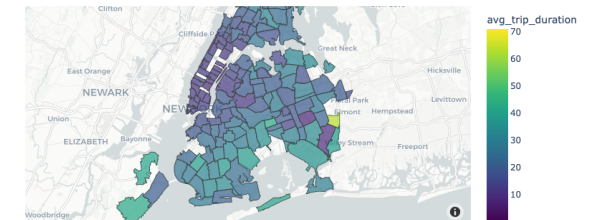
This is a sample application to show-case how easy it is to get started with Databricks Connect and build interactive Python applications.

NYC Taxi analysis (data processing on Databricks)

The below visualization uses a heatmap display based on geocoordinates for either the pickup or dropoff dimension and a second dimension is used for coloring.

Dimension 1
Dimension 2

dropoff_zip
avg_trip_duration



Get started with our [GitHub example!](#)



New Spark Connect Scala Client Features!

[SPARK-42554](#)

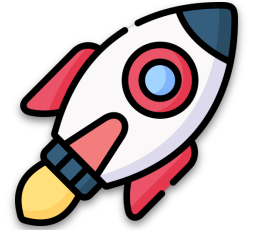
- The Scala client now supports more features in Spark 3.5!
- Part of this work was a major refactoring to split the `sql` submodule into `client` (`sql-api`) and `server-compatible` (`sql`) modules to reduce the set of dependencies needed on the client for classpath isolation ([SPARK-44273](#)).



New Spark Connect Scala Client Features!

[SPARK-42554](#)






















- It is now possible to use MLlib directly with Spark Connect to do distributed training and inference ([design doc](#)).
- This supports logistic regression classifiers, basic feature transformers, basic model evaluators, and more!
- This also integrates with Spark's vectorized Python UDF framework.



New Spark Connect Scala Client Features!

SPARK-42554

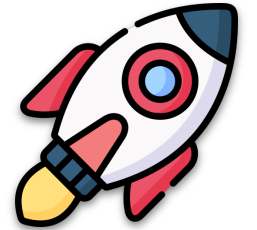
▼ Sub-Tasks

1.	Make spark connect supporting canceling job group		OPEN	Unassigned
2.	 High level design doc for Distributed ML <> spark connect		RESOLVED	Weichen Xu
3.	 Initial prototype implementation for PySparkML		RESOLVED	Weichen Xu
4.	 Extract the common .ml classes to `mllib-common`		RESOLVED	Ruifeng Zheng
5.	 Make LiteralExpression support array		RESOLVED	Ruifeng Zheng
6.	 Factor literal value conversion out to connect-common		RESOLVED	Ruifeng Zheng
7.	 Helper function to convert proto literal to value in Python Client		RESOLVED	Ruifeng Zheng
8.	 Implement ml function {array_to_vector, vector_to_array}		RESOLVED	Ruifeng Zheng
9.	 Move `toCatalystValue` to connect-common		RESOLVED	Ruifeng Zheng
10.	 Make Torch Distributor compatible with Spark Connect		RESOLVED	Ruifeng Zheng
11.	 Torch Distributor support Local Mode		RESOLVED	Ruifeng Zheng

New Spark Connect Scala Client Features!

[SPARK-42497](#)

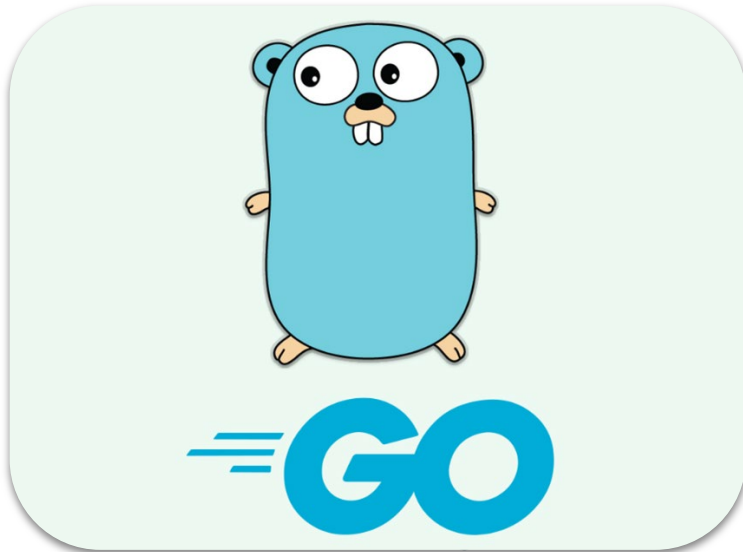
- Parity of the Pandas API on Spark using Spark Connect has improved.
- The Spark Connect client for structured streaming workloads both in Python and Scala now also supports all available features.



New Spark Connect Scala Client Features!

[SPARK-43351](#)

- The community also started a client for Spark Connect in Golang in a separate repository here: github.com/apache/spark-connect-go.

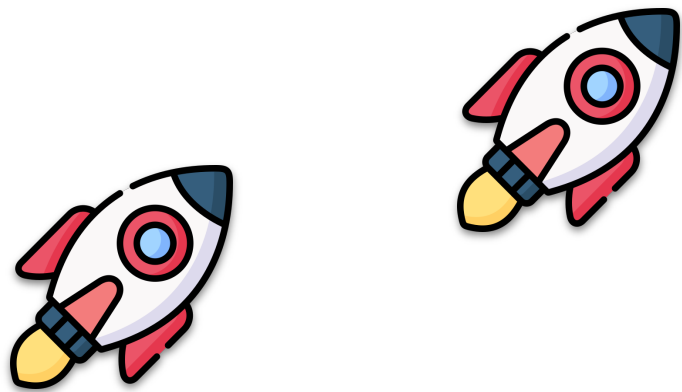


Pandas API Support for Spark Connect

[SPARK-42497](#)



- Spark Connect now includes the capability to execute Pandas functions and logic as needed in your PySpark programs.



Agenda



Spark Connect

Deploy and update Spark clusters independently from their clients



SQL Features

HyperLogLog aggregates based on Apache Dataskeches, array manipulation functions, IDENTIFIER clause, and more



PySpark Features

Arrow-optimized Python UDFs, Python UDTFs, new testing API, improved error messages, and more



Spark Streaming

Support multiple stateful operators, checkpointing for RocksDB state store, dropDuplicatesWithinWatermark



SQL
IDENTIFIER



Built-in
Functions



HyperLogLog



Named
Arguments

SQL Features



The IDENTIFIER Clause

SPARK-41231

- The new IDENTIFIER clause provides flexibility to avoid risk of SQL injection attacks.
- Using this feature to specify table/column/function names is powerful when paired with the query parameter feature added in the previous Spark release.



The IDENTIFIER Clause

SPARK-41231

- The new IDENTIFIER clause provides flexibility to avoid risk of SQL injection attacks.

```
spark.sql(  
  "CREATE TABLE IDENTIFIER(:tbl)(col INT)",  
  args = {  
    "tbl": "my_schema.my_tbl"  
  }  
)
```



The IDENTIFIER Clause

SPARK-41231

- The new IDENTIFIER clause provides flexibility to avoid risk of SQL injection attacks.

```
spark.sql(  
  "SELECT IDENTIFIER(:col) FROM IDENTIFIER(:tbl)",  
  args = {  
    "col": "col",  
    "tbl": "my_schema.my_tbl"  
  }  
)
```



Named Argument Syntax for Function Calls

SPARK-44059

- Spark SQL now lets users call functions with parameter names preceding their values.

```
SELECT mask(  
  'AbCD123-@$',  
  lowerChar => 'q',  
  upperChar => 'Q',  
  digitChar => 'd');
```



HyperLogLog Approx. Aggregate Functions

SPARK-16484

- New SQL functions count unique values within groups with precision and efficiency, including storing the result of intermediate computations to sketch buffers which can be persistent into storage and loaded back later.



HyperLogLog Approx. Aggregate Functions

[SPARK-16484](#)

- These implementations use the Apache Datasketches library for consistency with the open-source community and easy integration.



HyperLogLog Approx. Aggregate Functions

SPARK-16484

- These implementations use the Apache Datasketches library for consistency with the open-source community and easy integration.

```
SELECT hll_sketch_estimate(  
  hll_sketch_agg(col))  
FROM VALUES  
  ("abc"), ("def"), ("abc"), ("ghi"), ("abc") tab(col);  
> 4
```



New Functions for Manipulating Arrays

SPARK-41231

```
SELECT array_append(array(1, 2, 3), "HELLO");  
> [ 1, 2, 3, "HELLO" ]
```

```
SELECT array_prepend(array(1, 2, 3), 99);  
> [ 99, 1, 2, 3 ]
```

```
SELECT array_insert(array(1, 2, 3), 0, 4);  
> [ 4, 1, 2, 3 ]
```

```
SELECT array_compact(array(1, NULL, 3));  
> [ 1, 3 ]
```



SQL Functions ⇒ Scala, Python, R APIs

[SPARK-43907](#)

- Before Spark 3.5, there were many SQL functions that were not available in the Scala, Python, or R DataFrame APIs.



SQL Functions ⇒ Scala, Python, R APIs

SPARK-43907

- Before Spark 3.5, there were many SQL functions that were not available in the Scala, Python, or R DataFrame APIs.
- This presented difficulties invoking the functions within DataFrames as users found it necessary to type the function name in string literals without any help from auto-completion.



SQL Functions ⇒ Scala, Python, R APIs

SPARK-43907

- Spark 3.5 removes this problem by making 150+ SQL functions available in the DataFrame APIs.

```
/**
 * Returns the number of days since 1970-01-01.
 *
 * @group datetime_funcs
 * @since 3.5.0
 */
def unix_date(e: Column): Column = Column.fn("unix_date", e)

/**
 * Returns the number of microseconds since 1970-01-01 00:00:00 UTC.
 *
 * @group datetime_funcs
 * @since 3.5.0
 */
def unix_micros(e: Column): Column = Column.fn("unix_micros", e)
```



SQL Functions ⇒ Scala, Python, R APIs

SPARK-43907

- Spark 3.5 removes this problem by making 150+ SQL functions available in the DataFrame APIs.

```
spark.conf.set(  
  "spark.sql.session.timeZone",  
  "America/Los_Angeles")  
df = spark.createDataFrame(  
  [ ("2015-07-22 10:00:00", ) ],  
  ["t"])
```



SQL Functions ⇒ Scala, Python, R APIs

SPARK-43907

- Spark 3.5 removes this problem by making 150+ SQL functions available in the DataFrame APIs.



New!

```
df.select(  
  unix_micros(  
    to_timestamp(df.t))  
  .alias("n")  
  .collect()  
  
> [Row(n=14375844000000000)]
```





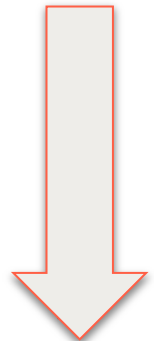
SQL Functions ⇒ Scala, Python, R APIs

SPARK-43907

Sub-Tasks

...

1.	✓	Add percentile like functions to Scala and Python API		RESOLVED	Jiaan Geng
2.	✓	Add misc functions to Scala and Python		RESOLVED	BingKun Par
3.	✓	Add some, bool_or, bool_and, every to Scala and Python		RESOLVED	Jiaan Geng
4.	✓	Add array_agg, array_size, cardinality, count_min_sketch, mask, named_struct, json_* to Scala and Python		RESOLVED	Tengfei Hua
5.	✓	Add cast alias to Scala and Python		RESOLVED	Unassigned
6.	✓	Add bit operations to Scala and Python		RESOLVED	Jiaan Geng
7.	✓	Add date time functions to Scala and Python - part 1		RESOLVED	Jiaan Geng
8.	✓	Add unix_* functions to Scala and Python		RESOLVED	BingKun Par
9.	✓	Add make_* functions to Scala and Python		RESOLVED	BingKun Par
10.	✓	Add current_* functions to Scala and Python		RESOLVED	Ruifeng Zhe
11.	✓	Add linear regression aggregate functions to Scala and Python		RESOLVED	Jiaan Geng



Agenda



Spark Connect

Deploy and update Spark clusters independently from their clients



SQL Features

HyperLogLog aggregates based on Apache Dataskeches, array manipulation functions, IDENTIFIER clause, and more



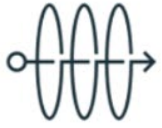
PySpark Features

Arrow-optimized Python UDFs, Python UDTFs, new testing API, improved error messages, and more



Spark Streaming

Support multiple stateful operators, checkpointing for RocksDB state store, dropDuplicatesWithinWatermark



Arrow optimized
Python UDF



Python
UDTF



Scala/Python
Functions



PySpark Testing
APIs

PySpark



Arrow-Optimized Python UDFs

[SPARK-40307](#)

- Python UDFs run 2X faster on modern CPU architectures, thanks to vectorized I/O!

```
spark.conf.set(
    "spark.sql.execution.pythonUDF.arrow.enabled",
    True)

@udf("integer")
def my_len_udf(s: str) -> int:
    return len(s)
```



Arrow-Optimized Python UDFs

[SPARK-40307](#)

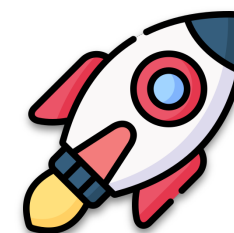
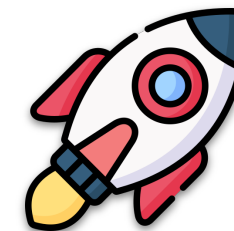
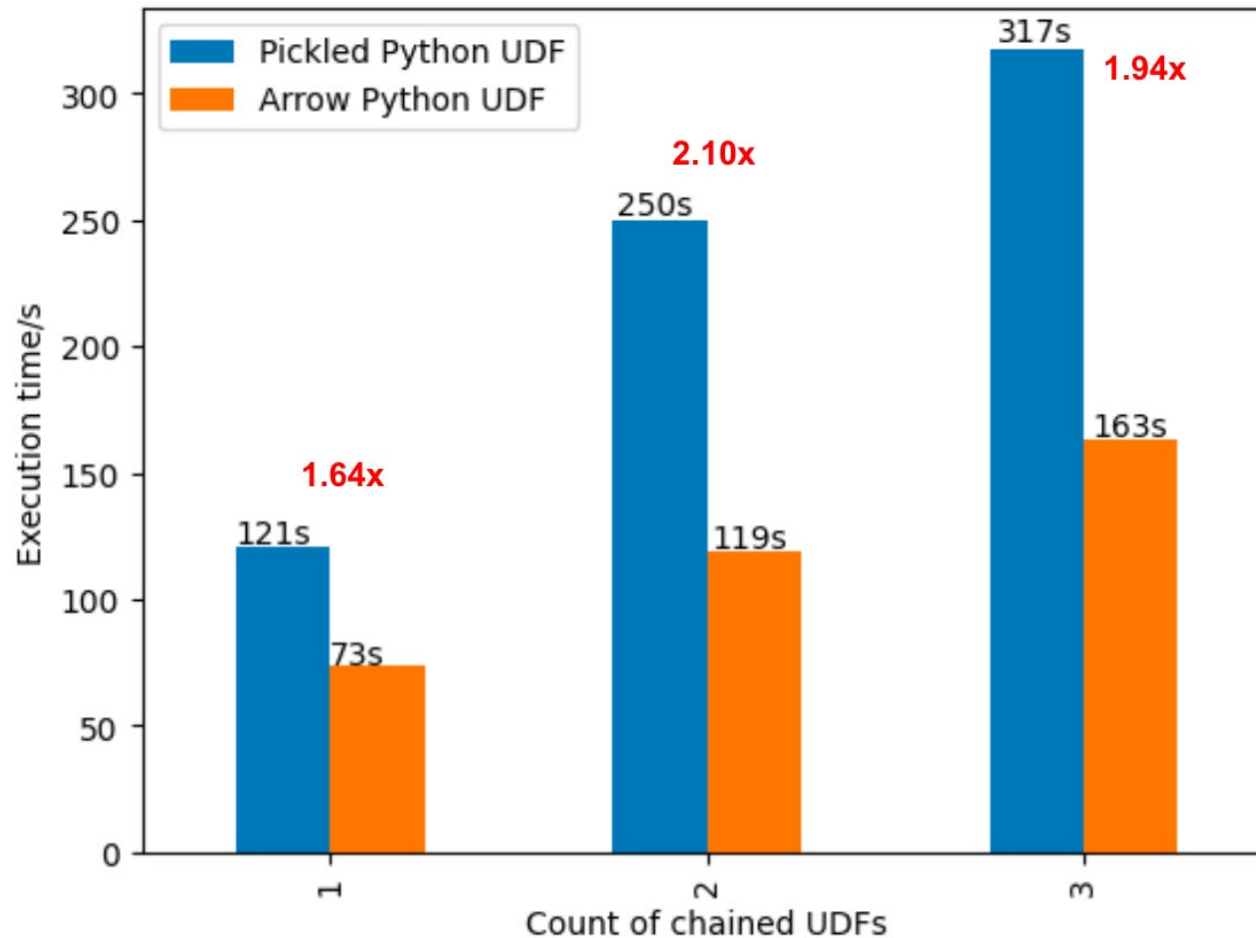
- You can also specify `useArrow=True` at registration time instead of using the config.

```
@udf("integer", useArrow=True)
def my_len_udf(s: str) -> int:
    return len(s)
```



Arrow-Optimized Python UDFs

SPARK-40307



Python User Defined Table Functions

Spark
3.5

This is a new kind of function that returns an *entire table* as output instead of a single scalar result value

- Once registered, they can appear in the FROM clause of a SQL query



Python User Defined Table Functions


Spark
3.5

This is a new kind of function that returns an *entire table* as output instead of a single scalar result value

- Once registered, they can appear in the FROM clause of a SQL query
- Or use the DataFrame API to call them



Python User Defined Table Functions



Spark
3.5

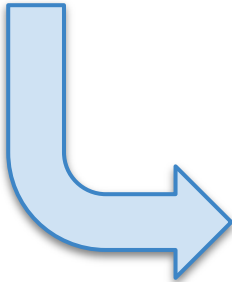
```
from pyspark.sql.functions import udtf

@udtf(returnType="num: int, squared: int")
class SquareNumbers:
    def eval(self, start: int, end: int):
        for num in range(start, end + 1):
            yield (num, num * num)
```



Python User Defined Table Functions

Spark
3.5



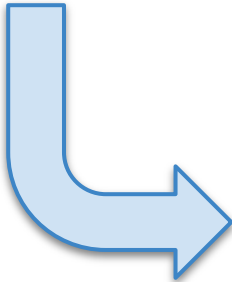
```
SELECT *  
FROM SquareNumbers(  
  start => 1,  
  end => 3);
```

num	squared
1	1
2	4
3	9



Python User Defined Table Functions

Spark
3.5



```
SquareNumbers(  
  lit(1), lit(3))  
  .show()
```

num	squared
1	1
2	4
3	9



Python User Defined Table Functions

Polymorphic Analysis

Compute the output schema for each call depending on arguments, using `analyze`

```
class ReadFromConfigFile:
    @staticmethod
    def analyze(filename: AnalyzeArgument):
        with open(os.path.join(
            SparkFiles.getRootDirectory(),
            filename.value), "r") as f:
            # Compute the UDTF output schema
            # based on the contents of the file.
            return AnalyzeResult(
                from_file(f.read()))
    ...
```



Python User Defined Table Functions

Polymorphic Analysis

Compute the output schema for each call depending on arguments, using `analyze`

```
ReadFromConfigFile(lit("config.txt")).show()
```

```
+-----+-----+
| start_date | other_field |
+-----+-----+
| 2024-04-02 |           1 |
+-----+-----+
```



Python User Defined Table Functions

Input Table Partitioning

Split input rows among instances: `eval` runs once per row, then `terminate` runs last

```
class CountAndMax:
    def __init__(self):
        self._count = 0
        self._max = 0
    def eval(self, row: Row):
        self._count += 1
        self._max = max(self._max, row[0])
    def terminate(self):
        yield self._count, self._max
```



Python User Defined Table Functions

Input Table Partitioning

Split input rows among instances: `eval` runs once per row, then `terminate` runs last

```
WITH t AS (SELECT id FROM RANGE(0, 100))  
SELECT * FROM CountAndMax(  
  TABLE(t) PARTITION BY id / 10 ORDER BY id);
```

count	max
10	0
10	1
...	



Python User Defined Table Functions

Variable Keyword Arguments

The `analyze` and `eval` methods may accept `*args` or `**kwargs`

```
class VarArgs:
    @staticmethod
    def analyze(**kwargs: AnalyzeArgument):
        return AnalyzeResult(StructType(
            [StructField(key, arg.dataType)
             for key, arg in sorted(
                 kwargs.items())]))

    def eval(self, **kwargs):
        yield tuple(value for _, value
                    in sorted(kwargs.items()))
```



Python User Defined Table Functions

Variable Keyword Arguments

The `analyze` and `eval` methods may accept `*args` or `**kwargs`

```
SELECT * FROM VarArgs(a => 10, b => 'x');
```

a	b
10	"x"



Python User Defined Table Functions

Custom Initialization

Create a subclass of `AnalyzeResult` and consume it in each subsequent `__init__`

```
class SplitWords:
    @dataclass
    class MyAnalyzeResult(AnalyzeResult):
        numWords: int
        numArticles: int

    def __init__(self, r: MyAnalyzeResult):
        ...
```



Python User Defined Table Functions

Custom Initialization

Create a subclass of `AnalyzeResult` and consume it in each subsequent `__init__`

```
@staticmethod
def analyze(text: str):
    words = text.split(" ")
    return MyAnalyzeResult(
        schema=StructType()
            .add("word", StringType())
            .add("total", IntegerType()),
        withSinglePartition=True,
        numWords=len(words)
        numArticles=len(
            word for word in words
            if word in ("a", "an", "the")))
```



Enhanced error messages in PySpark

SPARK-42986

- Previously, the set of exceptions thrown from the Python Spark driver did not leverage the error classes introduced in Apache Spark™ 3.3.
- All of the errors from DataFrame and SQL have been migrated, and contain the appropriate error classes and codes.



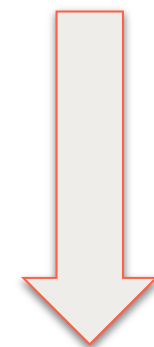
Enhanced error messages in PySpark

SPARK-42986

▼ Sub-Tasks

...

1.	✔	Introduce PySparkRuntimeError	📄	RESOLVED	Haejoon Lee
2.	✔	Migrate Spark Connect DataFrame errors into error class	📄	RESOLVED	Haejoon Lee
3.	✔	Migrate Column errors into error class	📄	RESOLVED	Haejoon Lee
4.	✔	Migrate ValueError from DataFrame into PySparkValueError.	📄	RESOLVED	Haejoon Lee
5.	✔	Refactoring similar error classes such as `NOT_XXX`.	📄	RESOLVED	Unassigned
6.	✔	Automate error class documentation	📄	RESOLVED	Unassigned
7.	✔	Testing JVM-captured exceptions from Python side.	📄	RESOLVED	Unassigned
8.	✔	Migrate Spark Connect Column errors into error class	📄	RESOLVED	Haejoon Lee
9.	✔	Migrate TypeError from DataFrame(Reader Writer) into error class	📄	RESOLVED	Haejoon Lee
10.	✔	Migrate UDF errors into error class	📄	RESOLVED	Haejoon Lee
11.	✔	Migrate Expression errors into error class	📄	RESOLVED	Haejoon Lee



DataFrame Equality Testing API

[SPARK-44042](#)

- New DataFrame equality test utility functions including detailed, color-coded test error messages, which clearly indicate differences between DataFrame schemas and data within DataFrames.



DataFrame Equality Testing API

[SPARK-44042](#)

- This lets developers easily add equality tests that produce actionable results for their applications to enhance productivity.



DataFrame Equality Testing API

[SPARK-44042](#)

- `pyspark.testing.assertDataFrameEqual`
- `pyspark.testing.assertPandasOnSparkEqual`
- `pyspark.testing.assertSchemaEqual`



DataFrame Equality Testing API

SPARK-44042

```
pyspark.errors.exceptions.base.PySparkAssertError: [DIFFERENT_ROWS]
Results do not match: ( 33.33333 % )
*** actual ***
  Row(name='Amy', languages=['C++', 'Rust'])
! Row(name='Jane', languages=['Scala', 'SQL', 'Java'])
  Row(name='John', languages=['Python', 'Java'])

*** expected ***
  Row(name='Amy', languages=['C++', 'Rust'])
! Row(name='Jane', languages=['Scala', 'Java'])
  Row(name='John', languages=['Python', 'Java'])
```



PySpark DeepSpeed Distributor

[SPARK-44264](#)

- This makes it easier for Pyspark users to run distributed training and inference with DeepSpeed on Spark clusters.



Agenda



Spark Connect

Deploy and update Spark clusters independently from their clients



SQL Features

HyperLogLog aggregates based on Apache Dataskeches, array manipulation functions, IDENTIFIER clause, and more



PySpark Features

Arrow-optimized Python UDFs, Python UDTFs, new testing API, improved error messages, and more



Spark Streaming

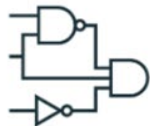
Support multiple stateful operators, checkpointing for RocksDB state store, dropDuplicatesWithinWatermark



Changelog
Checkpointing



Stateful Operator
Chaining



dropDuplicates
WithinWatermark



Memory
Management in
State Store

Streaming

Full Support for Multiple Stateful Operators

[SPARK-42376](#)

- Time interval joins between streams are now supported, possibly followed by other stateful operators.
- For example, workloads can now join streams of ads and clicks, then aggregate over time windows.



Full Support for Multiple Stateful Operators

SPARK-42376

For example, here's an example of time window aggregation in both streams followed by stream-stream join with event time window:

Scala

Java

Python

```
clicksWindow = clicksWithWatermark.groupBy(
    clicksWithWatermark.clickAdId,
    window(clicksWithWatermark.clickTime, "1 hour")
).count()

impressionsWindow = impressionsWithWatermark.groupBy(
    impressionsWithWatermark.impressionAdId,
    window(impressionsWithWatermark.impressionTime, "1 hour")
).count()

clicksWindow.join(impressionsWindow, "window", "inner")
```

Full Support for Multiple Stateful Operators

SPARK-42376

Here's another example of stream-stream join with time range join condition followed by time window aggregation:

Scala

Java

Python

```
val joined = impressionsWithWatermark.join(
  clicksWithWatermark,
  expr("""
    clickAdId = impressionAdId AND
    clickTime >= impressionTime AND
    clickTime <= impressionTime + interval 1 hour
  """),
  joinType = "leftOuter"      // can be "inner", "leftOuter", "rightOuter",
                             "fullOuter", "leftSemi"
)

joined
  .groupBy($"clickAdId", window($"clickTime", "1 hour"))
  .count()
```

Changelog Checkpointing for RocksDB State Store Providers

[SPARK-43421](#)

- This new checkpoint mechanism for the RocksDB state store provider persists the changelog (updates) of the state.
- This reduces the commit latency significantly which also reduces end to end latency significantly.
- To enable, set this config to true:
`spark.sql.streaming.stateStore.rocksdb.changelogCheckpointing.enabled`



Changelog Checkpointing for RocksDB State Store Providers

SPARK-43421

Here are the configs regarding to RocksDB instance of the state store provider:

Config Name	Description
<code>spark.sql.streaming.stateStore.rocksdb.compactOnCommit</code>	Whether we perform a range compaction of RocksDB instance for commit operation
<code>spark.sql.streaming.stateStore.rocksdb.changelogCheckpointing.enabled</code>	Whether to upload changelog instead of snapshot during RocksDB StateStore commit

RocksDB State Store Provider Memory Management Enhancements

[SPARK-43311](#)

- New fine-grained memory management lets users cap the total memory usage across RocksDB instances in the same executor process.
- Users can now reason about and configure the memory usage per executor process.



Introducing `dropDuplicatesWithinWatermark`

[SPARK-42931](#)

- This new API deduplicates events without requiring the timestamp for event time to be the same, as long as the timestamp for these events are close enough to fit within the watermark delay.
- With this new feature, users can avoid errors like “Timestamp for event time could differ even for events to be considered as duplicates.”



Introducing dropDuplicatesWithinWatermark

[SPARK-42931](#)

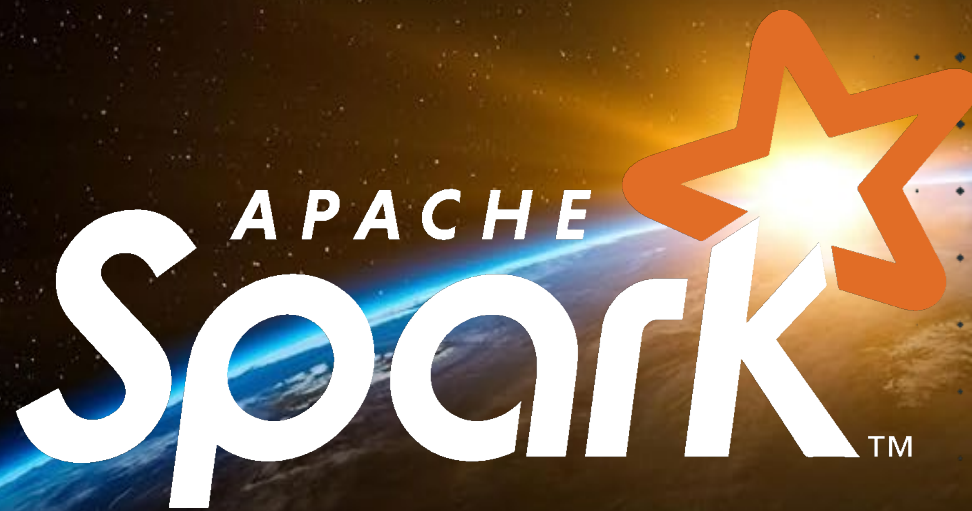
Python

Scala

Java

```
streamingDf = spark.readStream. ...  
  
# deduplicate using guid column with watermark based on eventTime column  
streamingDf \  
  .withWatermark("eventTime", "10 hours") \  
  .dropDuplicatesWithinWatermark("guid")
```





Thank you for your
contributions!



Daniel Tenedorio (daniel.tenedorio @ databricks)