

Stranger Triumphs: Automating Spark Upgrades and Migrations at Netflix

Databricks AI Summit 2024

Holden Karau
Bobby Morck

The bottom right portion of the slide features two large, solid red geometric shapes. The first is a parallelogram tilted to the right, and the second is a vertical rectangle. They are positioned in the lower right area of the slide, partially overlapping the bottom edge.

Our Problems

We have unsupported versions of Spark in production

When things go wrong, I don't remember what we did ~5 months ago let alone ~5 years ago

They often seem to go wrong when we are trying to focus or sleep

Spark 2 is very much EOLd, Spark 4 is coming soon

Why do we have these problems?

APIs changes and code breaks

Keeping code up to date is not a lot of fun

Backporting is not fun

Candy is more fun than taxes*

Testing data pipelines well is hard

Some of our data pipelines can have real world impacts when they go wrong

How can we work around our problem?

Software:

- Automated Code Update Tools
 - (Abstract Syntax Tree (AST) transforms, or regexes both are fine)
- Generated Tests
- Automated Testing and Validation

Social:

- Increase visibility of out of date code & change incentives

Ok social first:

- People are way harder than computers
- We gave a deadline (and slipped) like a "normal" project
- Created visibility
- Found org champions

[Spark Migration Tracking](#)

[SparkSQL Workflows Stash Tracking Tab](#)

[Non-SparkSQL workflows s](#)



Spark Migration
Newsletter

Ok social first:

Spark Migrations Home Analyze SQL

Search for a workflow

Maestro Production

DSE.PAA.TC_PROCESS_COLLECTION_D

SQL Scala [View on scheduler](#)

Status as of: Mar 31, 2024, 3:23:31 AM

✓ According to our latest data, this workflow has been successfully migrated. If you believe this is not correct, please reach out to [@spark-migrations](#) for assistance.

[View pull request in Stash](#)

Spark jobs for this workflow

Job	Type	Status
DSE.PAA.Process_Collection_D	SQL	Migrated
DSE.PAA.Process_Collection_D_write	Scala	Migrated

Job detail: DSE.PAA.Process_Collection_D_write

Status as of: Dec 18, 2023, 6:48:05 PM

[Customize job verification and pull request](#)

✓ According to our latest data, this job has been successfully migrated. If you believe this is not correct, please reach out to [@spark-migrations](#) for assistance.

[Genie shadow output \(experiment\)](#) [Genie shadow output \(control\)](#) [Verification job](#)

Data correctness verification: **pass**

Metadata:

Running basic checks for the table prodhive.dse.collection_d and its snapshots 8801052831429950342 and 3897624111340605466 before validation
Both the snapshots have the same changed partitions
Both the snapshots have the same schema
Both the snapshots have the same parent snapshot
Running basic checks for the table prodhive.dse.collection_day_d and its snapshots 338255766784127082 and 5659033199068280911 before validation
Both the snapshots have the same changed partitions
Both the snapshots have the same schema
Both the snapshots have the same parent snapshot



And now onto computers:

- API changes (and updating your code) is annoying – we can automate some of that
- Testing code you inherited is a nightmare, we can sort-of-kind-of fake some of that (enough*)

Holden Karau



I'm on the Spark PMC (like tenure :p)

Worked on Spark for ~15 years

Co-author of Learning Spark (1st ed),
High Performance Spark (1st ed and
working on 2nd ed)

Twitter: @holdenkarau, bluesky
holdenkarau.com, mastodon

@holden@tech.lgbt

OOS Livestreams:

<https://youtube.com/user/holdenkarau>

Github <https://github.com/holdenk>

Outside of work: Queer, Trans,
Motorcycles, My Dog



Bobby Morck

Engineer on the Big Data Compute
Team at Netflix

Focus on Spark, Hadoop, Iceberg

Outside of Work: Half-marathons,
various athletics, learning guitar

github: <https://github.com/bmorck>



Managed Migration Tooling

Goal

Abstract and automate as much of the migration process away from our end users as possible

Managed Migration Tooling

Goal

Abstract and automate as much of the migration process away from our end users as possible

How

Build an inventory of all Spark jobs, migration control and automation plane, spark job validation, observability into migration process

Managed Migration Tooling

Goal

Abstract and automate as much of the migration process away from our end users as possible

How

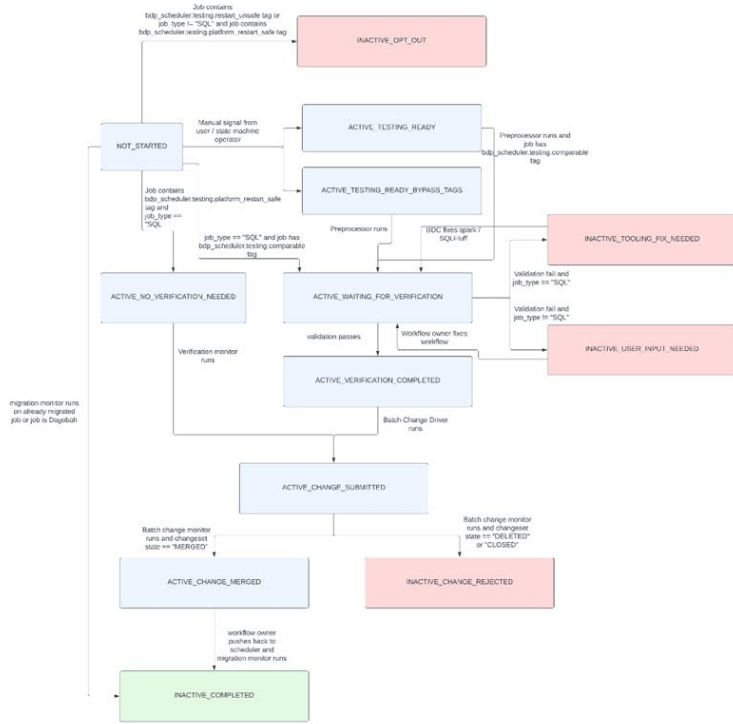
Build an inventory of all Spark jobs, migration control and automation plane, spark job validation, observability into migration process

Caveats

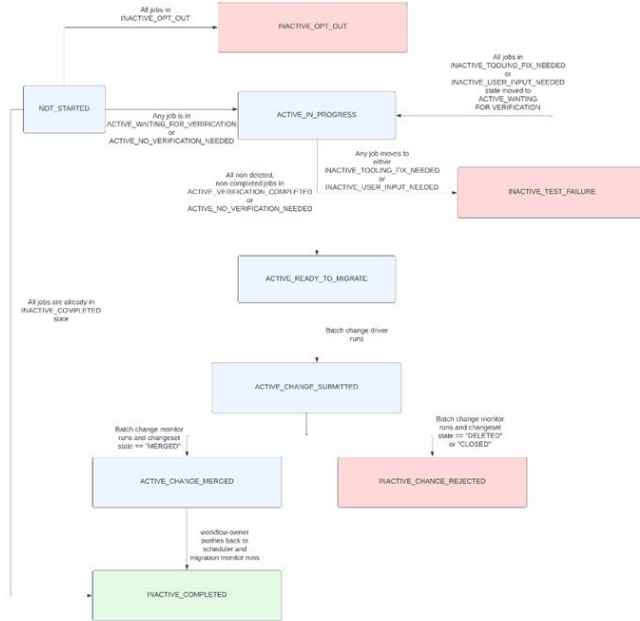
Validation tooling only compatible with Iceberg tables and non -deterministic output cannot be validated, SparkSQL / PySpark only

Managed Migration Tooling continued

Spark Job State Transitions



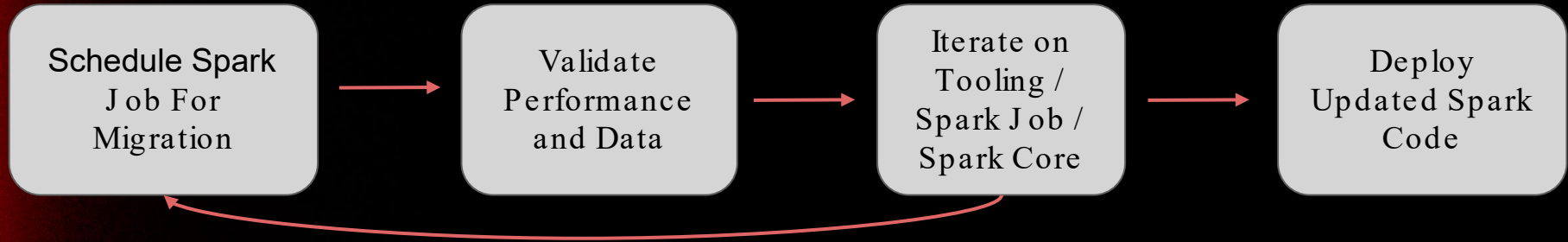
Spark Workflow State Transitions



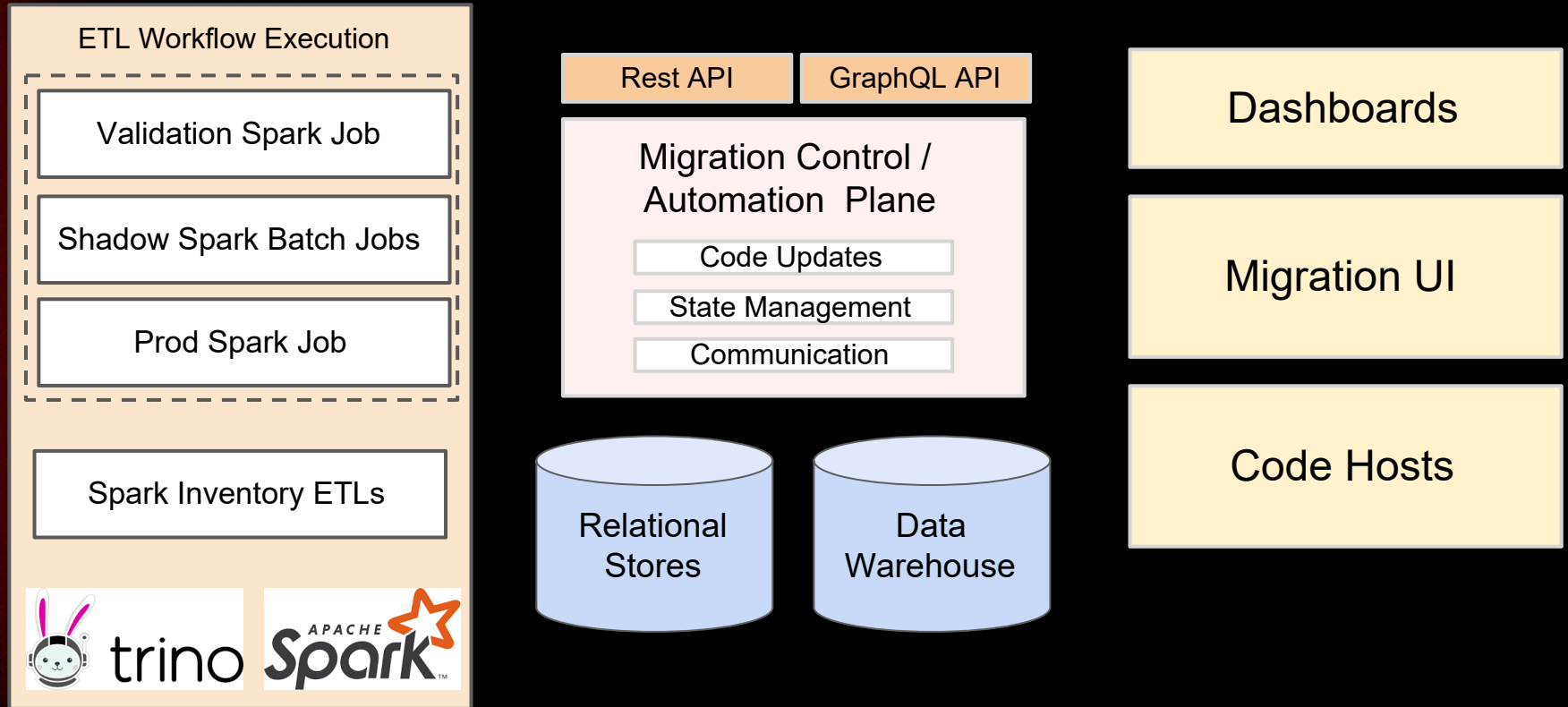
Managed Migration Tooling continued



Managed Migration Tooling continued



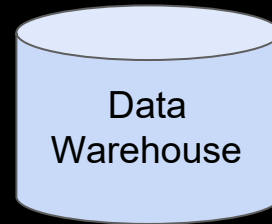
Managed Migration Tooling continued



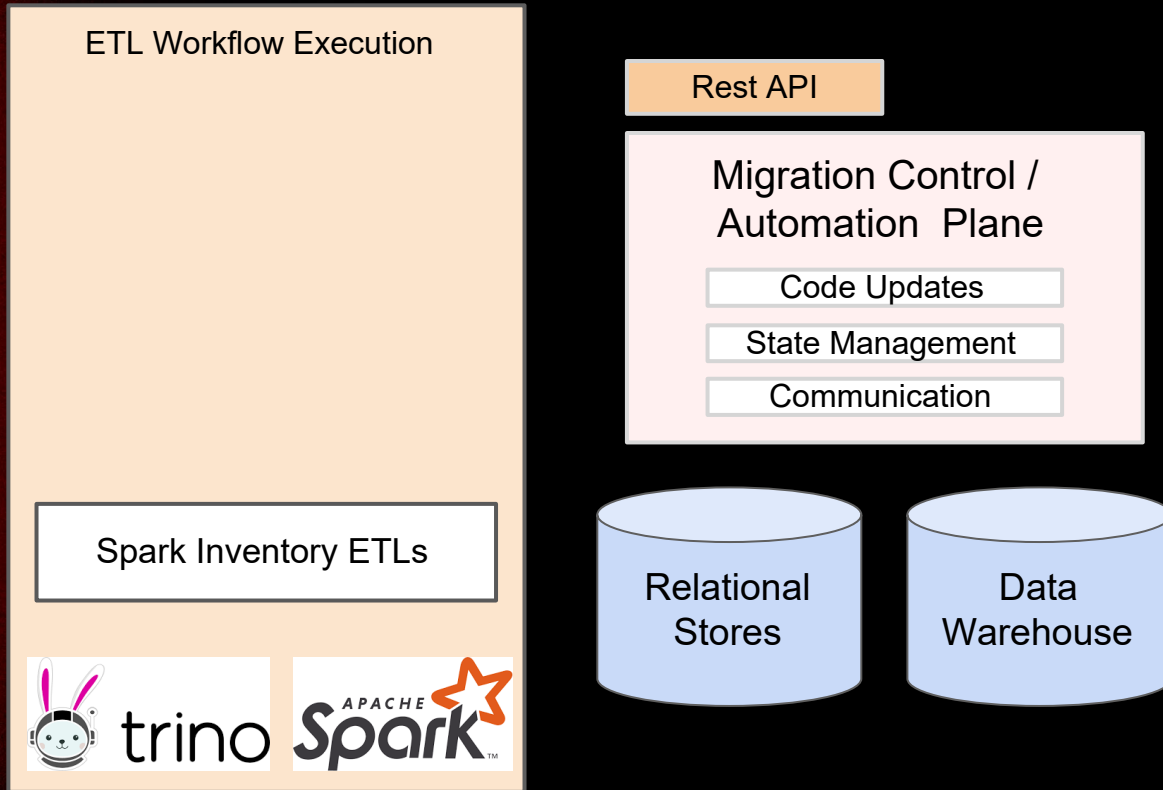
Managed Migration Tooling continued

ETL Workflow Execution

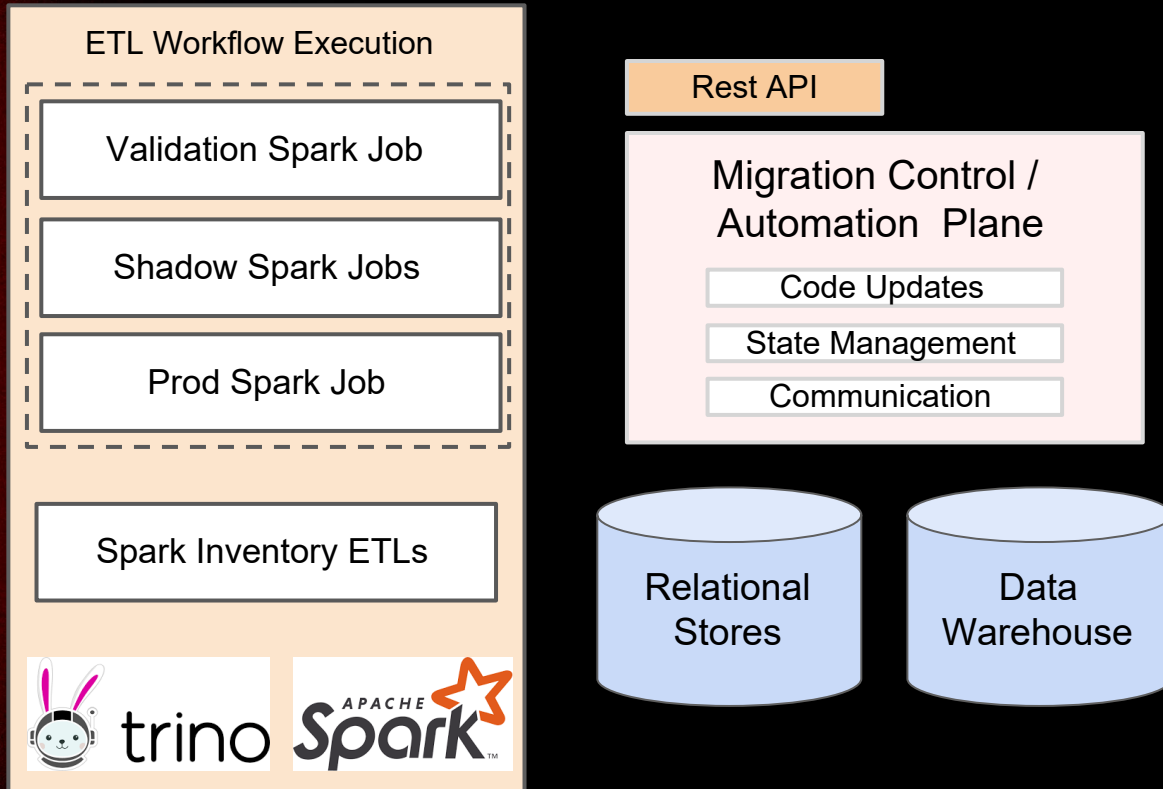
Spark Inventory ETLs



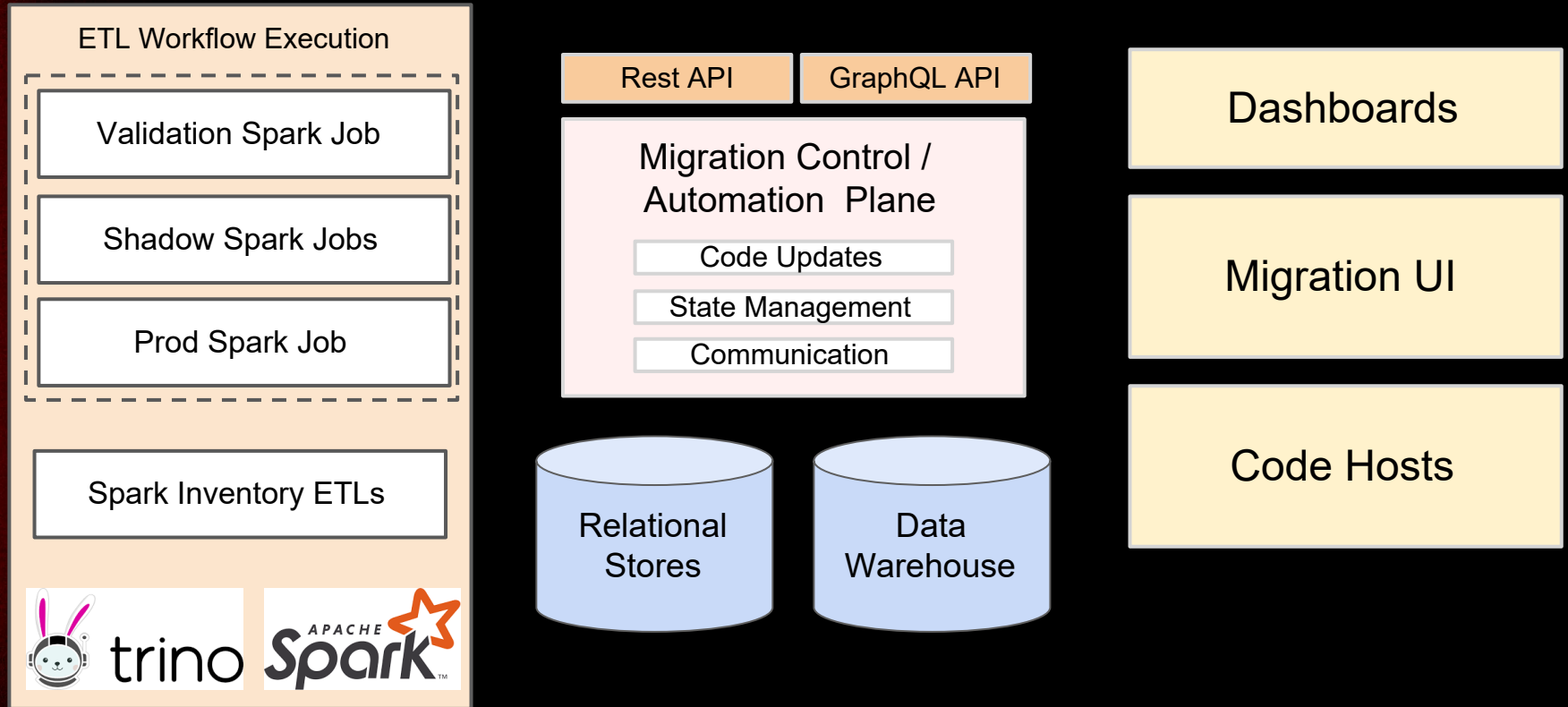
Managed Migration Tooling continued



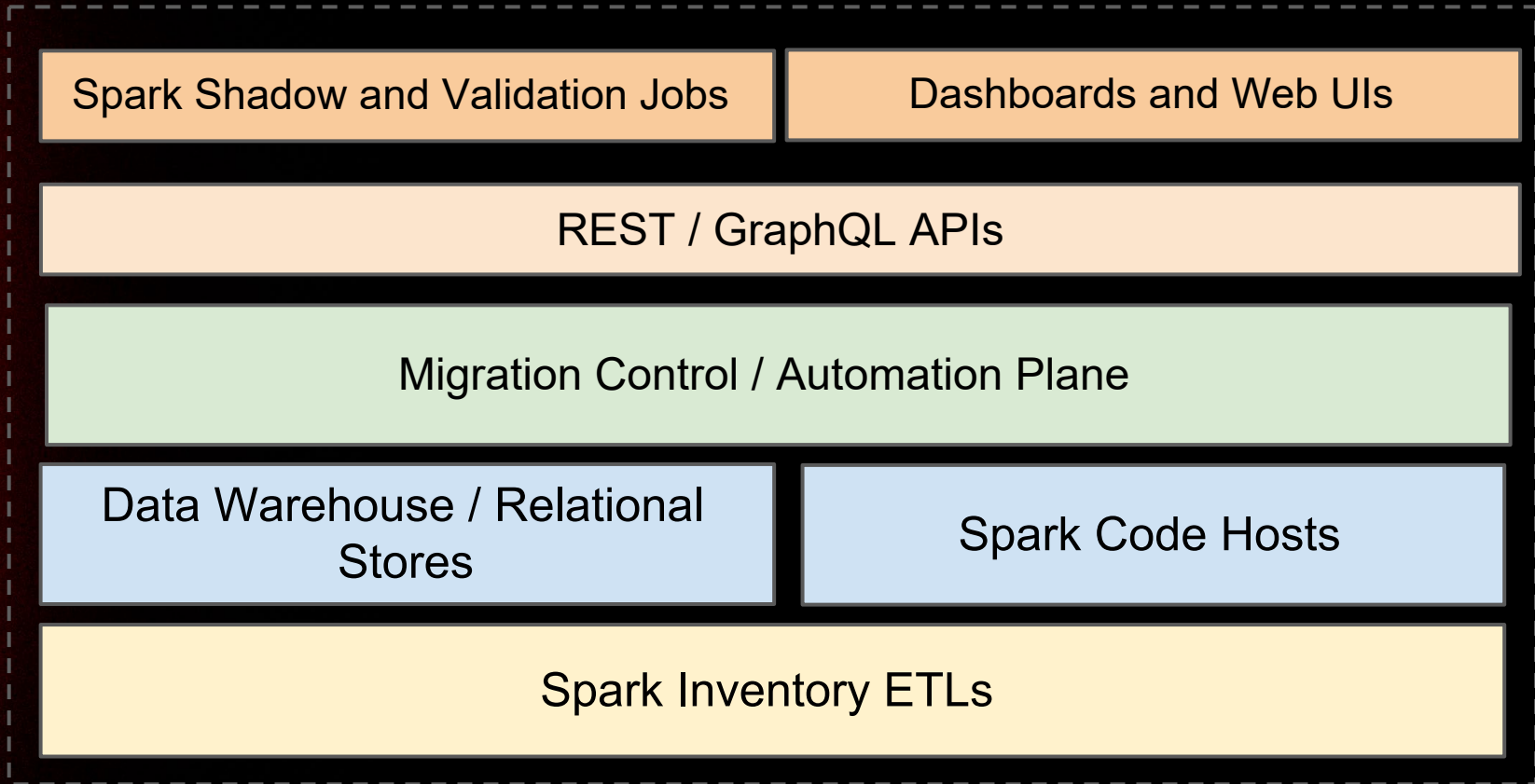
Managed Migration Tooling continued



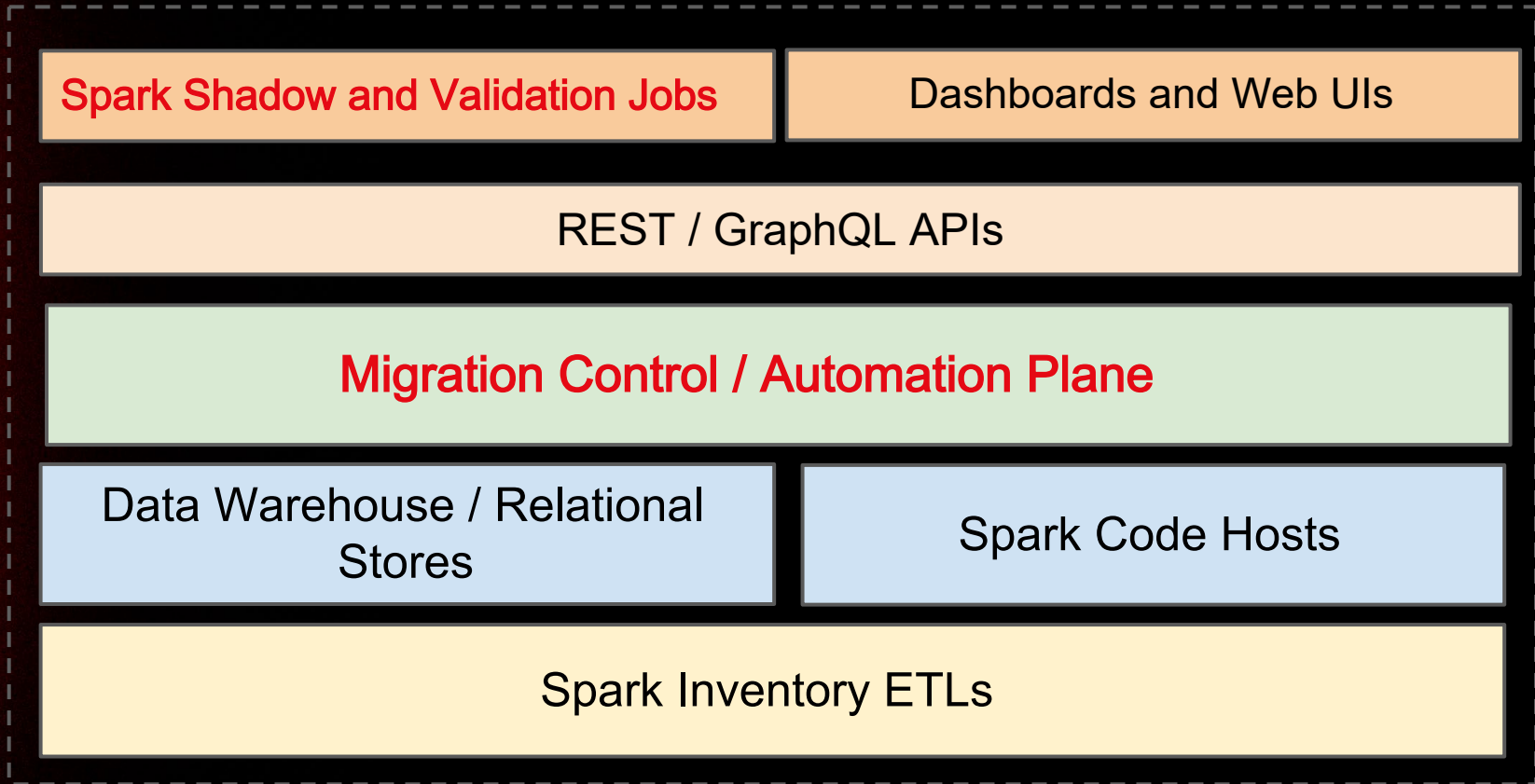
Managed Migration Tooling continued



Managed Migration Tooling continued



Managed Migration Tooling continued



Code Update Tools

- Generally not regular expressions.
- Scala: Spark Auto Upgrade (ScalaFix)
- Python: PySparkler (libcst)
- SQL: SQLFluff
- Java: (skipped, we didn't have that many)
- Check them out at
<https://github.com/holdenk/spark-upgrade>













How do you figure out the rules to make?

- Release notes (incomplete)
- Migration Manager (MIMA) changes (soooo many)
- Try and see what's broken :p (aka YOLO)

Upgrading from Spark SQL 3.0 to 3.1

- In Spark 3.1, statistical aggregation function includes `std`, `stddev`, `stddev_samp`, `variance`, `var_samp`, `skewness`, `kurtosis`, `covar_samp`, `corr` will return `NULL` instead of `Double.NaN` when `DivideByZero` occurs during expression evaluation, for example, when `stddev_samp` applied on a single element set. In Spark version 3.0 and earlier, it will return `Double.NaN` in such case. To restore the behavior before Spark 3.1, you can set `spark.sql.legacy.statisticalAggregate` to `true`.
- In Spark 3.1, `grouping_id()` returns long values. In Spark version 3.0 and earlier, this function returns int values. To restore the behavior before Spark 3.1, you can set `spark.sql.legacy.integerGroupId` to `true`.
- In Spark 3.1, SQL UI data adopts the formatted mode for the query plan explain results. To restore the behavior before Spark 3.1, you can set `spark.sql.ui.explainMode` to `extended`.
- In Spark 3.1, `from_unixtime`, `unix_timestamp`, `to_unix_timestamp`, `to_timestamp` and `to_date` will fail if the specified datetime pattern is invalid. In Spark 3.0 or earlier, they result `NULL`.
- In Spark 3.1, the Parquet, ORC, Avro and JSON datasources throw the exception `org.apache.spark.sql.AnalysisException: Found duplicate column(s) in the data schema in read` if they detect duplicate names in top-level columns as well in nested structures. The datasources take into account the SQL config `spark.sql.caseSensitive` while detecting column name duplicates.

Code Update Tools

 SparkSQL: Fix Group By Clause ●	6
#4732 by bmorck was merged on Apr 13, 2023 · Approved	
 SparkSQL: Improvements to lateral view, hints, sort by ●	5
#4731 by bmorck was merged on Apr 14, 2023 · Approved	
 SparkSQL: Improve window frame bounds ●	3
#4722 by bmorck was merged on Apr 12, 2023 · Approved 1 task done	
 SparkSQL: Allow for any ordering of create table clauses ●	10
#4721 by bmorck was merged on Apr 14, 2023 · Approved 1 task done	
 SparkSQL: Add distinct to comparison operator ●	1
#4719 by bmorck was merged on Apr 11, 2023 · Approved	
 SparkSQL: Fix file literal lexing ●	6
#4718 by bmorck was merged on Apr 11, 2023 · Approved	
 SparkSQL: Create external table support ●	4
#4692 by bmorck was merged on Apr 11, 2023 · Approved	
 SparkSQL: Add using and options clause to create view statement ●	2
#4691 by bmorck was merged on Apr 8, 2023 · Approved	
 Support Spark Iceberg DDL ●	9
#4690 by bmorck was merged on Apr 12, 2023 · Approved	
 Remove TIME as reserved keyword in SparkSQL ●	2
#4662 by bmorck was merged on Apr 4, 2023 · Approved	
 Add support for non-quoted file paths in SparkSQL ●	5
#4650 by bmorck was merged on Apr 3, 2023 · Approved 1 task done	
 Add SparkSQL support for LONG primitive type ●	3
#4639 by bmorck was merged on Mar 30, 2023 · Approved	

What do some rules look like?

- Let's just look at SQL & Scala

SQL rules: It's like an AST transform but... eh

```
SELECT approx_percentile(col, array(0.5, 0.4, 0.1), 100.0) FROM VALUES (0), (1), (2), (10)  
AS tab(col);
```

```
SELECT approx_percentile(col, 0.5, 100.0) FROM VALUES (0), (6), (7), (9), (10) AS  
tab(col);
```

```
SELECT approx_percentile(col, 0.5, 100.0) FROM VALUES (INTERVAL '0' MONTH), (INTERVAL '1'  
MONTH) AS tab(col);
```

```
SELECT approx_percentile(col, array(0.5, 0.7), 100.0) FROM VALUES (INTERVAL '0' SECOND),  
(INTERVAL '0 00:00:01.000000000'), (INTERVAL '0 00:00:02.000000000');
```

SQL rules: It's like an AST transform but... eh

```
SELECT approx_percentile(col, array(0.5, 0.4, 0.1), CAST(100.0 AS INT)) FROM VALUES (0),  
(1), (2), (10) AS tab(col);
```

```
SELECT approx_percentile(col, 0.5, CAST(100.0 AS INT)) FROM VALUES (0), (6), (7), (9),  
(10) AS tab(col);
```

```
SELECT approx_percentile(col, 0.5, CAST(100.0 AS INT)) FROM VALUES (INTERVAL '0' MONTH),  
(INTERVAL '1' MONTH) AS tab(col);
```

```
SELECT approx_percentile(col, array(0.5, 0.7), CAST(100.0 AS INT)) FROM VALUES (INTERVAL  
'0' SECOND), (INTERVAL '0 00:00:01.000000000'), (INTERVAL '0 00:00:02.000000000');
```

SQL rules: It's like an AST transform but... eh

```
def _eval(self, context: RuleContext) -> Optional[LintResult]:  
    functional_context = FunctionalContext(context)  
    children = functional_context.segment.children()  
    function_name_id_seg = (  
        children.first(sp.is_type("function_name"))  
        .children()  
        .first(sp.is_type("function_name_identifier"))[0]  
    )
```

SQL rules: It's like an AST transform but... eh

```
raw_function_name = function_name_id_seg.raw.upper().strip()

function_name = raw_function_name.upper().strip()

bracketed_segments = children.first(sp.is_type("bracketed"))

if function_name == "APPROX_PERCENTILE" or function_name == "PERCENTILE_APPROX":

    expression_count = 0

    expression_segment = None

    # Find "middle" of the approx_percentile(bloop) (e.g. bloop)

    for segment in bracketed_segments.children().iterate_segments(

        sp.is_type("expression")

    ):
```

SQL rules: It's like an AST transform but... eh

```
expression_count += 1

if expression_count == 3:
    expression_segment = segment

if expression_segment is not None:
    expression_child = expression_segment.children().first()

    # cast can either be a keyword or a function depending on if we're iterating on
    # parsed or updated code.

    if expression_child[0].type == "keyword":
        if expression_child[0].raw == "cast":
            return None
```


SQL rules: It's like an AST transform but.... eh

```
elif expression_child[0].type == "function":  
    function_name_id_seg = (  
        expression_child.children()  
        .first(sp.is_type("function_name"))  
        .children()  
        .first(sp.is_type("function_name_identifier"))[0]  
    )
```

SQL rules: It's like an AST transform but.... eh

```
raw_function_name = function_name_id_seg.raw.upper().strip()

function_name = raw_function_name.upper().strip()

# If we see a cast then we know this was already fixed.

if function_name == "CAST":

    return None

expression_child = expression_child[0]
```

SQL rules: It's like an AST transform but... eh

```
    edits = [  
        KeywordSegment("cast"),  
        SymbolSegment("(", type="start_bracket"),  
        expression_child,  
        WhitespaceSegment(),  
        KeywordSegment("as"),  
        WhitespaceSegment(),  
        KeywordSegment("int"),  
        SymbolSegment(")", type="end_bracket"),  
    ]  
    return LintResult(  
        anchor=context.segment,  
        fixes=[  
            LintFix.replace(expression_child, edits),  
        ],  
    ),  
)
```

What do they look like [Scala]



```
override def fix(implicit doc: SemanticDocument): Patch = {
  val readerMatcher =
    SymbolMatcher.normalized("org.apache.spark.sql.DataFrameReader")
  val jsonReaderMatcher =
    SymbolMatcher.normalized("org.apache.spark.sql.DataFrameReader.json")
  val utils = new Utils()

  def matchOnTree(e: Tree): Patch = {
    e match {
      case ns @ Term.Apply(jsonReaderMatcher(reader), List(param)) =>
```

What do they look like [Scala] continued

```
param match {  
    case utils.rddMatcher(rdd) =>  
        (Patch.addLeft(rdd, "session.createDataset(") +  
Patch.addRight(rdd, ")(Encoders.STRING)") +  
  
utils.addImportIfNotPresent(importer"org.apache.spark.sql.Encoders"))  
    case _ =>  
        Patch.empty  
}
```

What do they look like [Scala] continued

```
case elem @ _ =>
  elem.children match {
    case Nil => Patch.empty
    case _ => elem.children.map(matchOnTree).asPatch
  }
}
}
matchOnTree(doc.tree)
}
```

How do we know if it worked?

- Hope is not a plan
- Tests? (See <https://github.com/holdenk/spark-testing-base>)
- lakeFS, Iceberg, Delta, etc. + side by side runs
<https://github.com/holdenk/spark-upgrade/tree/main/pipelinecompare>
- Validation queries
 - [SodaCL](#)
 - <https://datatest.readthedocs.io/en/latest/intro/pipeline-validation.html>
 - [spark-expectations](#)

WAP to MAD

- Write Audit Publish (WAP) v.s. Migrate Audit Discard (MAD)
 - Write Audit Publish – popularized by Michelle Winters from Netflix in her talk "Whoops the Numbers are Wrong."
- Different meaning of "Audit"

Is that expensive? Does it catch everything?

- Yes
 - Beyond quadrupling the cost for shadow jobs comparisons themselves took substantial compute resources.
- No
 - Jobs with side effects
 - Non-deterministic jobs (we catch this with a double run)
 - etc.

Demo

Let's hope it doesn't crash

Note: this is a demo of the OSS version of the tool, our internal version depends on some extra internals “features” to go faster (insert engine noises)

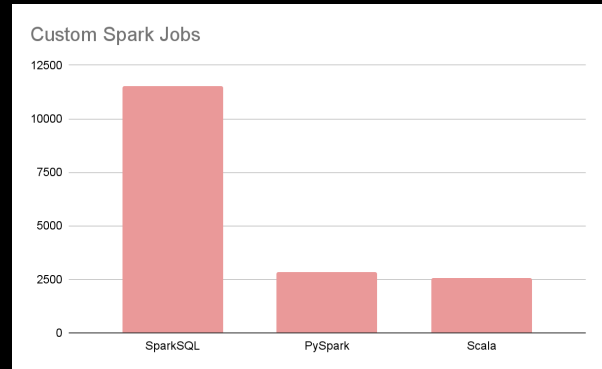
Results

Total Spark Jobs Running on Spark
3.3*

29.8K

Total Custom
Spark Jobs
Running on Spark
3.3*

16.9K



Results

SparkSQL

Avg Spark 2.x Runtime:
811s

Avg Spark 3.3 Runtime:
651s

Scala

Avg Spark 2.x Runtime:
1371s

Avg Spark 3.3 Runtime:
669s

PySpark

Avg Spark 2.x Runtime:
747s

Avg Spark 3.3 Runtime:
699s

Limitations

- Code Update Tooling
 - Not being able to infer types in code mod tooling
- Validation Tooling
 - Not freezing snapshots
 - Slow python UDFs
- Manual Support in case of Validation Failure

Limitations

```
186 @udf(returnType=DecimalType(38, 0))
187 def compute_hash_using_deephash(val):
188     h = DeepHash(val, number_to_string_func=safe_number_to_string)[val]
189     # Only take the first 22 digits in the integer and cast to a Decimal. We take 22 digits
190     # to ensure that we have sufficient extra digits to store the summation of column
191     # hashes. With 16 extra digits, we will be able to sum roughly a quadrillion hashes, which
192     # we do not expect to exceed.
193     return Decimal(str(int(h, 16))[:22])
194
195
196 @udf(returnType=StringType())
197 def compute_row_hash_using_deephash(*cols):
198     precision = cols[-1]
199     cols = cols[:-1]
200     return DeepHash(cols, number_to_string_func=safe_number_to_string, significant_digits=precision)[cols]
```

Limitations: Incidental Fix for an Iceberg bug

```
89 89     }
90 90
91 91     @Override
92 92     public int read() throws IOException {
93 93 +     private int readWithRetry(final int retryCount) throws IOException {
94 94 +         if (retryCount > awsProperties.getS3ReadRetries()) {
95 95 +             throw new IOException("Failed to read from S3 after " + awsProperties.getS3ReadRetries() + " retries");
96 96 +         }
97 97 +
98 98 +     }
99 99
100 100     Preconditions.checkState(!closed, "Cannot read: already closed");
101 101     positionStream();
102 102
103 103     pos += 1;
104 104     next += 1;
105 105     readBytes.increment();
106 106     readOperations.increment();
107 107
108 108     return stream.read();
109 109
110 110     try {
111 111 +         final int byteRead = stream.read();
112 112 +         pos += 1;
113 113 +         next += 1;
114 114 +         readBytes.increment();
115 115 +         readOperations.increment();
116 116 +         return byteRead;
117 117 +     } catch (IOException e) {
118 118 +         LOG.warn("IOException while reading from S3. Attempting retry #{}, retryCount + 1, e);
119 119 +         // Retry connection reset. Prior call to positionStream() ensures
120 120 +         // we will start at the correct offset.
121 121 +         openStream();
122 122 +         return readWithRetry(retryCount + 1);
123 123 +     }
124 124
125 125 }
126 126
```

Ok, but where doesn't this work well?

- Dependencies
- Programming language version change
 - The reality is there's a lot of Scala 2.11 code out there, OSS resources are focused on 2.12->2.13 migration's but folks are further back
 - Scala version change was the #2 reason blocking Spark upgrades for folks



In conclusion:

- Great success! No* more Spark 2.X! Yay!
- If you want to upgrade Spark and are lazy – <https://github.com/holdenk/spark-upgrade>
- Thanks to our employer (Netflix) and they are hiring
- The good news is we haven't made a system so powerful we can change APIs without caring
- The bad news is the same
- The excellent news is: Holden's dog is cute AF



Thank You.



N

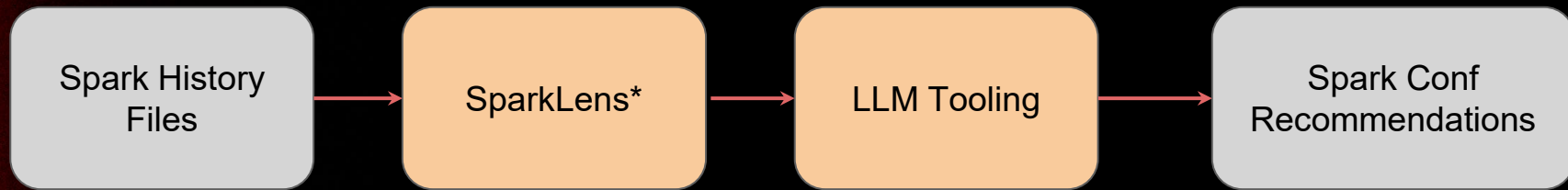
Looking to future migrations

Leverage OSS dataframe comparisons (available in Spark 3.5)

Leverage GenAI to automate spark config tuning

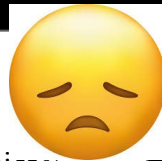
Smaller micro migrations to stay closer to the open source

Leverage GenAI to automate spark config tuning



* <https://github.com/Netflix-Skunkworks/sparklens/tree/oss-main>

Tips and Tricks



```
Caused by: java.lang.OutOfMemoryError: Unable to acquire 16384 bytes of memory, got 0
    at org.apache.spark.memory.MemoryConsumer.allocateArray(MemoryConsumer.java:100)
    at org.apache.spark.util.collection.unsafe.sort.UnsafeInMemorySorter.<init>(UnsafeInMemorySorter.java:117)
    at org.apache.spark.util.collection.unsafe.sort.UnsafeExternalSorter.<init>(UnsafeExternalSorter.java:154)
    at org.apache.spark.util.collection.unsafe.sort.UnsafeExternalSorter.create(UnsafeExternalSorter.java:121)
    at org.apache.spark.sql.execution.window.WindowExec$$anonfun$14$$anon$1.fetchNextPartition(WindowExec.scala:340)
    at org.apache.spark.sql.execution.window.WindowExec$$anonfun$14$$anon$1.next(WindowExec.scala:391)
    at org.apache.spark.sql.execution.window.WindowExec$$anonfun$14$$anon$1.next(WindowExec.scala:290)
    at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator.agg_doAggregateWithKeys1$(Unknown
Source)
    at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator.agg_doAggregateWithKeys$(Unknown
Source)
    at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator.processNext(Unknown Source)
    at org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterator.java:43)
    at
org.apache.spark.sql.execution.WholeStageCodegenExec$$anonfun$8$$anon$1.hasNext(WholeStageCodegenExec.scala:379)
    at org.apache.spark.sql.hive.SparkHiveWriterContainer.writeToFile(hiveWriterContainers.scala:297)
    at org.apache.spark.sql.hive.execution.InsertIntoHiveTable$$anonfun$1.apply(InsertIntoHiveTable.scala:218)
    at org.apache.spark.sql.hive.execution.InsertIntoHiveTable$$anonfun$1.apply(InsertIntoHiveTable.scala:218)
    at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
    at org.apache.spark.scheduler.Task.run(Task.scala:100)
    at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:336)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:750)
```

Short Term Stop Gaps (e.g. Spark Conf Changes)

Change Broadcast Join Threshold (or disable in the case of high driver memory): `spark.sql.autoBroadcastJoinThreshold`

Disable AQE: `spark.sql.adaptive.enabled`

Driver Memory: `spark.driver.memory`

Executor Memory: `spark.executor.memory`

Executor Memory Fraction: `spark.memory.fraction`

Driver Memory Fraction: `spark.driver.memoryOverhead`

Adjust Number of Partitions: `spark.sql.shuffle.partitions`, `spark.default.parallelism`

Regressions

- Caching SQL UNION of different column data types does not work inside `Dataset.union`
 - Fixed in Spark 3.5 (backported internally)
- Evaluate subquery before filter push down
 - <https://github.com/apache/spark/pull/43471>
- Filter pushdown through project results in double evaluation
 - <https://github.com/apache/spark/pull/45802>

Managed Migration Tooling continued

