Apache Spark Structured Streaming: The Easy Path to Data Lake.

Dmitry Lakhvich (frostball@gmail.com) Data Engineer at Lamoda

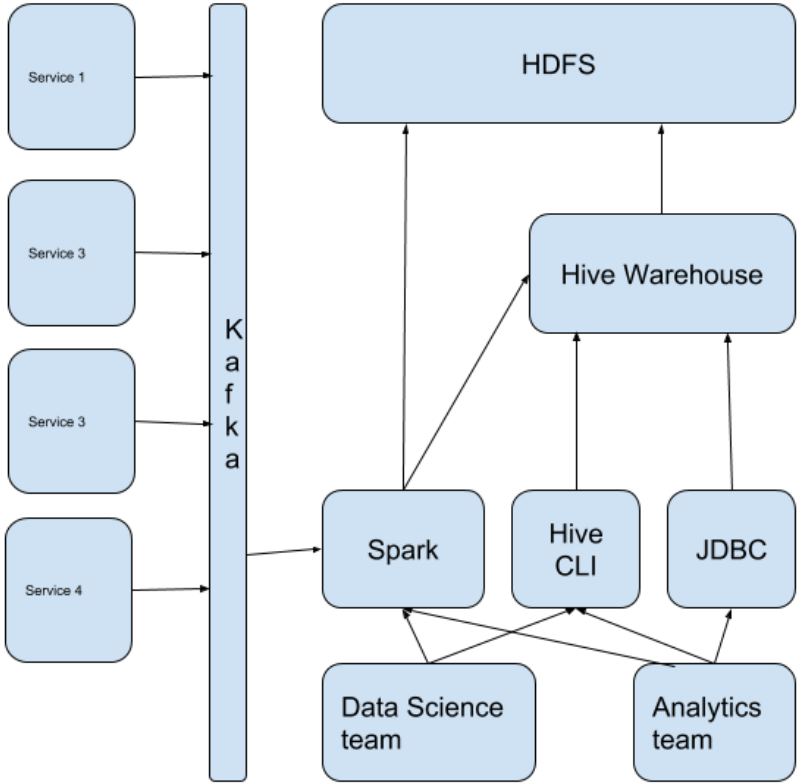
Hello, NE Scala

Dmitry Lakhvich
Data Engineer at Lamoda

Data Lake

Data Lake

- Message bus contains different streams of structured data and non-structured data
- Users want to query data as soon as possible(since some event happened)
- Persistent scalable storage
- SQL over saved data
- Programmatic access to storage
- Different data-access patterns

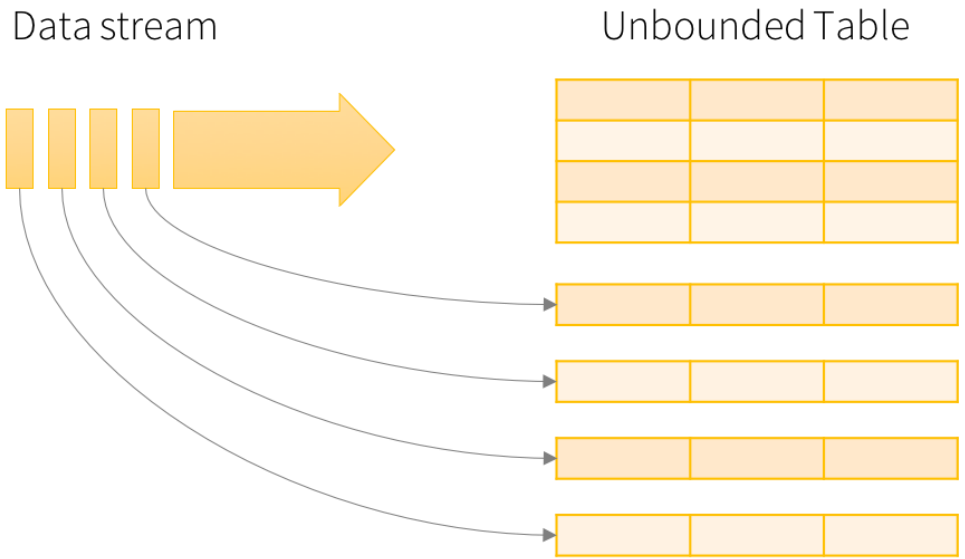


Structured streaming

Spark streaming before 2.0

Image source:
https://spark.apache.org

Spark structured streaming



new data in the
data stream
=
new rows appended
to a unbounded table

Data stream as an unbounded table

Image source:
https://spark.apache.org

Sink modes

- Complete Mode - The entire updated Result Table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.

- Append Mode - Only the new rows appended in the Result Table since the last trigger will be written to the external storage. This is applicable only on the queries where existing rows in the Result Table are not expected to change.

- Update Mode - Only the rows that were updated in the Result Table since the last trigger will be written to the external storage (available since Spark 2.1.1). Note that this is different from the Complete Mode in that this mode only outputs the rows that have changed since the last trigger. If the query doesn't contain aggregations, it will be equivalent to Append mode.

Source:
https://spark.apache.org

Simple problems should have simple solutions.

The easy way

Configuration – Streaming Job

```
case class StreamingJob(
    source: Source,
    sink: Sink,
    hive: Option[Hive]
)
```

Configuration - Source

```
case class Source(
    format: String,
    options: Option[Map[String, String]],
    schema: Option[String],
    columns: Option[Seq[String]],
    filter: Option[String]
)
```

Configuration – Sink

```
case class Sink(
    format: String,
    saveMode: String = "append",
    partitionBy: Option[Seq[String]],
    triggerInSeconds: Int,
    finalNumberOfPartitions: Option[Int],
    path: String,
    checkpointLocation: String,
    options: Option[Map[String, String]]
)
```

Configuration – Hive

```
case class Hive(db: String, table: String, warehousePath: String)
```

Configuration – use pureconfig

- It’s simple
- You have config validation for free
- It’s much easier to work with case classes

```
def fromConfig(path: Option[String] = None): StreamingJob =
  path.fold(loadConfig[StreamingJob])(
    x => loadConfig[StreamingJob](Paths.get(x))
  ) match {
    case Left(_)     => throw new IllegalArgumentException("Bad config")
    case Right(conf) => conf
  }
```

Configuration – application.conf

```
source {
  format = "kafka"
  filter = "location != 'scala_users'"
  schema = "user_id string, action string,platform string"
  columns = ["*"]# may contain  sql functions
  options {
    "kafka.bootstrap.servers" = "kafka-01.hosts.example.com:9092, kafka-01.hosts.example.com:9092"
    "subscribe" = "example_topic"
    "failOnDataLoss" = "false"
    "startingOffsets" = "latest"
  }
}
```

Configuration – application.conf

```
sink {
  format = "orc"
  saveMode = "append"
  partitionBy = ["platform"]
  triggerInSeconds = 600
  path = "hdfs://hadoop.hosts.example.com:8020/warehouse/exm/example_log"
  checkpointLocation = "hdfs://hadoop.hosts.example.com:8020/checkpoints/example_log"
}
hive {
  db = "exm"
  table = "example_log"
  warehousePath = "hdfs://hadoop.hosts.example.com:8020/warehouse"
}
```

Streaming application

```
object StreamApplication extends Logging with Spark {
```

Streaming application

```
def commonListener(
    sink: Sink,
    hive: Option[Hive]
): StreamingQueryListener =
```

Streaming application

```
def executeSource(sourceConfig: Source, tableName: String)(
    implicit spark: SparkSession
): DataFrame = {
```

Source execution

- Read stream from source(kafka) of data
- Create a temporary view(a table, which we can query with sql)
- Convert internal format of values to JSON(in our case kafka topic contains strings of JSONs)
- Map each JSON row to tuple according to schema

Streaming application – executeSource

```
val emptyMap: Map[String, String] = Map.empty
val queryName = "source_" + spark.conf.get("spark.app.name")
val raw = spark.readStream
  .format(format)
  .options(options.getOrElse(emptyMap))
  .option("queryName", queryName)
  .load()
raw.createOrReplaceTempView(tableName)
```

Streaming application – executeSource

```
  val sch: String = schema.getOrElse(
    throw new IllegalArgumentException("Schema not specified")
  )
  raw
    .selectExpr("cast (value as string) as json") //not only json
    .select(from_json($"json", sch, emptyMap) as "data")
    .select("data.*")
    .createOrReplaceTempView(tableName)
}

val cols = columns.getOrElse(Seq("*")).mkString(", ")
val where = filter.fold("")(x => s"where $x")
spark.sql(s"select $cols from $tableName $where")
```

Sink execution

- Read result DataFrame from Source
- Detect it’s schema
- Save it to hdfs with partitioning
- Create Hive table if needed according to result DataFrame from result

Streaming application - executeSink

```
def executeSink(df: DataFrame, sinkConfig: Sink)(
    implicit spark: SparkSession
): StreamingQuery = {
```

Streaming application - executeSink

```
val numPartitions = finalNumberOfPartitions.getOrElse(1)
val queryName = "sink_" + spark.conf.get("spark.app.name")
val stream = df
  .coalesce(numPartitions)
  .writeStream
  .format(format)
  .outputMode(saveMode)
  .option("checkpointLocation", checkpointLocation)
  .option("path", path)
  .option("queryName", queryName)
  .options(options.getOrElse(Map.empty[String, String]))
  .trigger(Trigger.ProcessingTime(triggerInSeconds.seconds))
  .partitionBy(partitionBy.getOrElse(Seq.empty[String]): _*)
  .start()
stream
```

Streaming application – run

```
spark.streams.addListener(commonListener(job.sink, hive))
val source = executeSource(job.source, tableName)
hive.foreach { conf =>
  import conf._
  HiveTools.recreateDbAndTableIfNotExists(
    warehousePath,
    db,
    table,
    sink.partitionBy,
    sink.format,
    source.schema
  )
}
executeSink(source, job.sink)
spark.streams.awaitAnyTermination()
```

Streaming application – run(typed)

```
def run[A <: Product: ClassTag: TypeTag, B <: Product: ClassTag: TypeTag](
    convert: A => Option[B]
): Unit = {
```

Streaming application

```
spark.streams.addListener(commonListener(job.sink, hive))
val source =
  executeSource(job.source, tableName).as[A].flatMap(x => convert(x)).toDF()
hive.foreach { conf =>
  import conf._
  HiveTools.recreateDbAndTableIfNotExists(
    warehousePath,
    db,
    table,
    sink.partitionBy,
    sink.format,
    source.schema
  )
}
executeSink(source, job.sink)
spark.streams.awaitAnyTermination()
```

Hive tools - repairTable

```
def repairTable(db: String, table: String)(
    implicit spark: SparkSession
): Unit = {
  spark.sql(s"msck repair table $db.$table")
  ()
}
```

Hive tools - createHiveDb

```
def createHiveDbIfNotExists(db: String, dbLocation: String)(
    implicit spark: SparkSession
): Unit = {
  val dbQuery = s"create database if not exists $db location '$dbLocation'"
  spark.sql(dbQuery)
  ()
}
```

HiveTools - createHiveTable

```
def recreateDbAndTableIfNotExists(
    warehousePath: String,
    db: String,
    table: String,
    partitionBy: Option[Seq[String]],
    format: String,
    schema: StructType
)(implicit spark: SparkSession): Unit = {
```

HiveTools - createHiveTable

```
val dbPath = s"$warehousePath/$db"
val tablePath = s"$dbPath/$table"
val allCols =
  schema.map(x => x.name.concat(" ").concat(x.dataType.simpleString))
val parts = partitionBy
  .getOrElse(Seq.empty[String])
  .map(x => allCols.dropWhile(a => !a.contains(x)))
  .filter(x => x.nonEmpty)
  .map(x => x.head)
val cols = allCols diff parts
val partsQuery =
  if (parts.isEmpty) "" else s"partitioned by (${parts.mkString(", ")})"
```

HiveTools - createHiveTable

```
val tableQuery =
  s"""create external table if not exists $db.$table
     |(${cols.mkString(", \n")})
     |$partsQuery
     |stored as $format location '$tablePath' tblproperties ('$format.compress'='SNAPPY')
   """.stripMargin
createHiveDbIfNotExists(db, dbPath)
spark.sql(s"drop table if exists $db.$table")
spark.sql(tableQuery)
if (partitionBy.getOrElse(Seq.empty[String]).nonEmpty)
  repairTable(db, table)
```

Example

```
object Example extends App {
  StreamApplication.run()
}
```

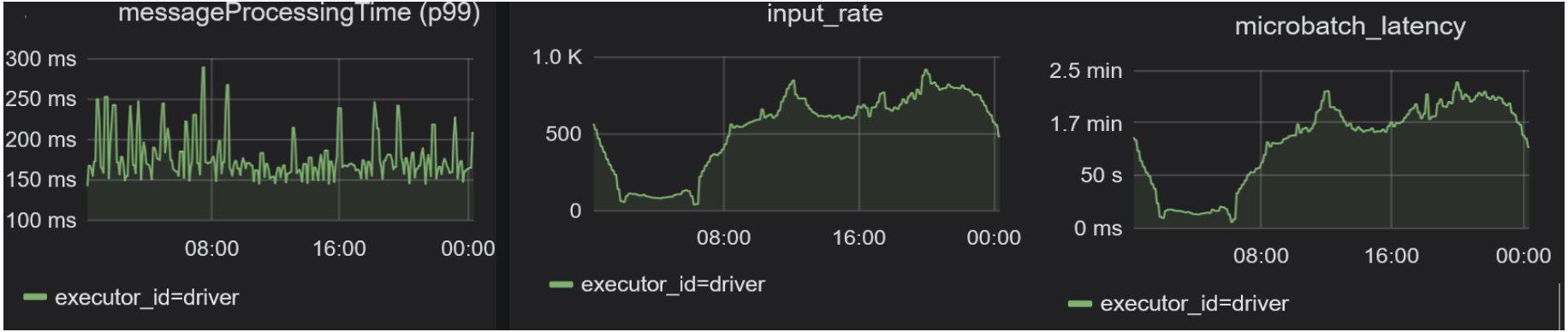
Typed Example

```
object TypedExample {
  StreamApplication.run[User, TaggedUser](convert)

  def convert(user: User): Option[TaggedUser] = {
    import user._
    Some(TaggedUser(Some("tag"), name, action, dt, platform))
  }
}
```

Metrics – Stats metrics.properties

```
master.source.jvm.class=org.apache.spark.metrics.source.JvmSource
worker.source.jvm.class=org.apache.spark.metrics.source.JvmSource
driver.source.jvm.class=org.apache.spark.metrics.source.JvmSource
executor.source.jvm.class=org.apache.spark.metrics.source.JvmSource

*.sink.statsd.class=org.apache.spark.metrics.sink.StatsdSink
*.sink.statsd.prefix="spark"
*.sink.statsd.period=10
*.sink.statsd.unit=seconds
*.sink.statsd.host="example.com"
*.sink.statsd.port=228
```

Metrics


messageProcessingTime (p99)
300 ms
250 ms
200 ms
150 ms
100 ms
08:00
16:00
00:00
executor_id=driver
input_rate
1.0 K
500
0
08:00
16:00
00:00
executor_id=driver
microbatch_latency
2.5 min
1.7 min
50 s
0 ms
08:00
16:00
00:00
executor_id=driver

Tips

- Name your queries
- You need understand how “hot” is your data
- Do not forget about compaction(small files make your NameNode cry)
- Spark can’t work properly with managed transactional Hive tables
- You can derive *schema* from case classes and have more guaranties(of course you need recompile your project)
- It’s just an example, you may implement in another way

Thank you, NE Scala

Dmitry Lakhvich
Data Engineer at Lamoda
LinkedIn: https://www.linkedin.com/in/dmitry-lakhvich-a610706b/
Twitter: @KrivdaTheTriewe
Telegram: @KrivdaTheTriewe
Email: frostball@gmail.com
Example: https://github.com/ReiReiRei/spark_easy_datalake

Special thanks to Andrey Sutugin