

# Spark 大数据处理

## 最佳实践

章剑锋（简锋）

# 目录

## 目录

大数据概览

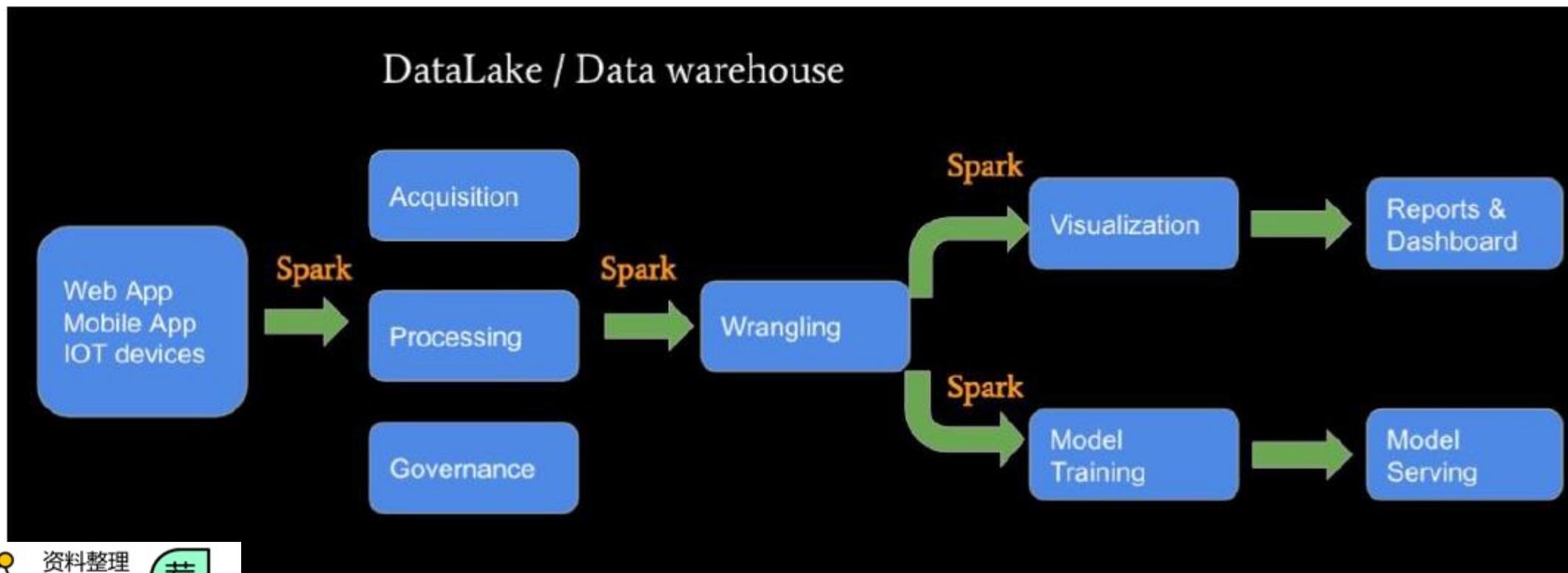
如何摆脱技术小白

Spark SQL 学习框架

EMR Studio 上的  
大数据最佳实践

# 大数据概览

- 大数据处理 ETL (Data → Data)
- 大数据分析 BI (Data → Dashboard)
- 机器学习 AI (Data → Model)



# 如何摆脱技术小白

---

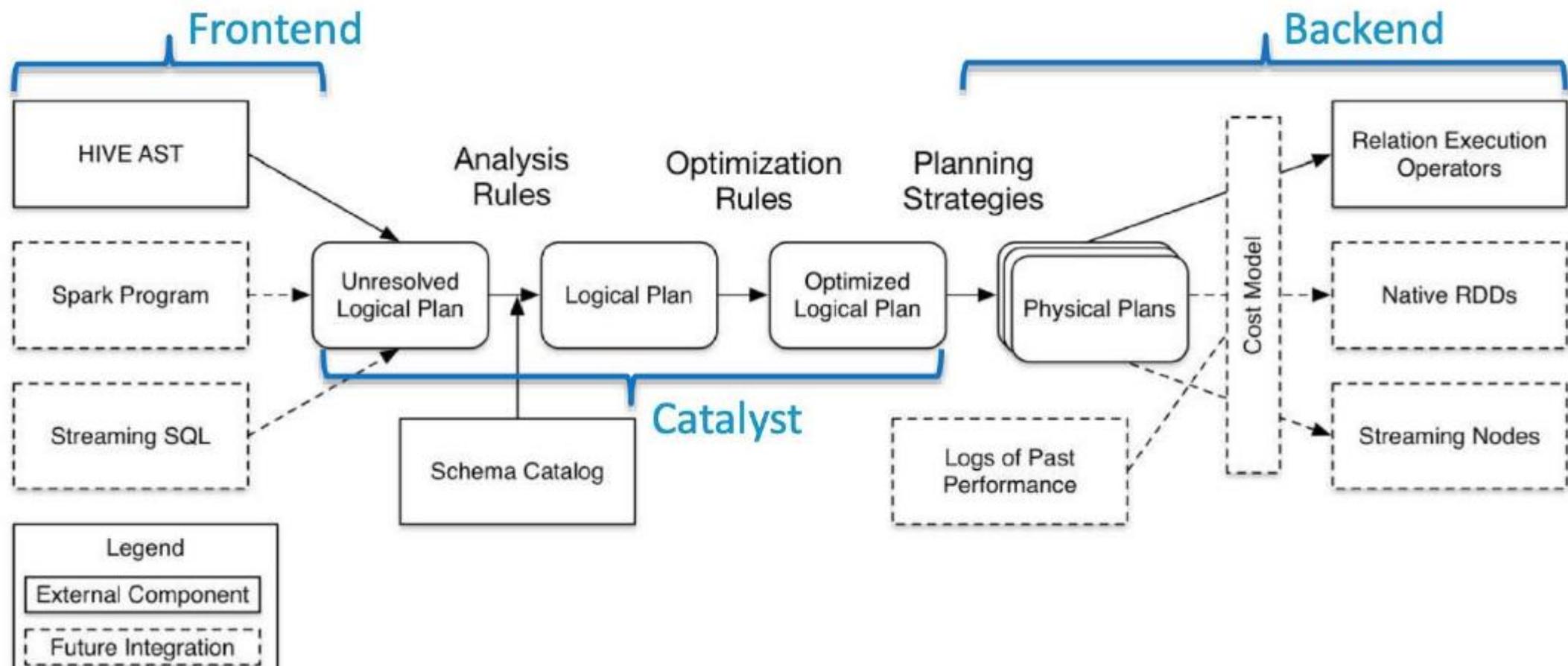
# 如何摆脱技术小白

- 只懂表面，不懂本质。
- 只懂得参考别人的Spark代码，不懂得Spark的内在机制，不懂得如何调优Spark Job

## 摆脱技术小白药方：

- 懂得运行机制
- 如何配置
- 如何看Log

# Spark SQL Architecture



# 如何配置 Spark App

- 配置 Driver
  - `spark.driver.cores`
- 配置 Executor
  - `spark.executor.cores`
- 配置 Runtime
  - `spark.jars`
- 配置 DAE
- ...

# Spark Log

- Running Spark version 2.4.7
- Submitted application: spark-C-F2B1B29D20419346-2G6BATQQW
- Changing view acls to: hadoop,user1,\*
- Changing modify acls to: hadoop,user1
- Changing view acls groups to:
- Changing modify acls groups to:
- SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop, user1, \*); groups with view permissions: Set
- Successfully started service 'sparkDriver' on port 33787.
- Registering MapOutputTracker
- Registering BlockManagerMaster
- Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
- BlockManagerMasterEndpoint up
- Created local directory at /mnt/disk2/yarn/usercache/user1/appcache/application\_1625879111378\_0501/blockmgr-ba4b6d54-0259-4a15-90fb-e2bb69702478
- Created local directory at /mnt/disk1/yarn/usercache/user1/appcache/application\_1625879111378\_0501/blockmgr-7315cdc1-dc51-473b-87b2-73707a47d017
- Created local directory at /mnt/disk4/yarn/usercache/user1/appcache/application\_1625879111378\_0501/blockmgr-2fd6a9e0-eab0-4e3f-9a14-fa2c7a4abbc6
- Created local directory at /mnt/disk3/yarn/usercache/user1/appcache/application\_1625879111378\_0501/blockmgr-34b891a1-d4a5-4ee9-aca8-f9de19cce70b
- MemoryStore started with capacity 366.3 MB
- Registering OutputCommitCoordinator
- Logging initialized @11879ms to org.spark\_project.jetty.util.log.Slf4jLog
- Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /jobs, /jobs/json, /jobs/job, /jobs/job/json, /stages, /stages/json, ,
- jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0\_252-b09
- Started @12017ms

# Spark SQL 学习框架 (结合图形/几何)

- Select Rows
- Select Columns
- Transform Column
- Group By
- Join

# Select Rows

```
val df1 = spark.createDataFrame(Seq((1, "andy", 20, "USA"),  
                                   (2, "jeff", 23, "China"),  
                                   (3, "james", 18, "USA")))  
                                   .toDF("id", "name", "age", "country")
```

```
// filter accept a Column  
val df2 = df1.filter($"age" >= 20)  
df2.show()
```





# Select Columns

```
val df1 = spark.createDataFrame(Seq((1, "andy", 20, "USA"),  
                                   (2, "jeff", 23, "China"),  
                                   (3, "james", 18, "USA")))  
    .toDF("id", "name", "age", "country")
```

// select can accept a list of string of the column names

```
val df2 = df1.select("id", "name")  
df2.show()
```





# Transform Column

```
// withColumn could be used to add/replace new Column
val df1 = spark.createDataFrame(Seq((1, "andy", 20, "USA"),
                                   (2, "jeff", 23, "China"),
                                   (3, "james", 18, "USA")))
                                   .toDF("id", "name", "age", "country")

val df2 = df1.withColumn("name", upper($"name"))
df2.show()
```





# Group By / Aggregation

```
val df1 = spark.createDataFrame(Seq((1, "andy", 20, "USA"),  
                                   (2, "jeff", 23, "China"),  
                                   (3, "james", 18, "USA")))  
                                   .toDF("id", "name", "age", "country")
```

// You can call agg function after groupBy directly, such as count/min/max/avg/sum

```
val df2 = df1.groupBy("country").count()  
df2.show()
```

// Pass a Map if you want to do multiple aggregation

```
val df3 = df1.groupBy("country").agg(Map("age" -> "avg", "id" -> "count"))  
df3.show()
```





# Join

```
val df1 = spark.createDataFrame(Seq((1, "andy", 20, 1),  
                                   (2, "jeff", 23, 2),  
                                   (3, "james", 18, 3)))  
                                   .toDF("id", "name", "age", "c_id")
```

```
val df2 = spark.createDataFrame(Seq((1, "USA"),  
                                   (2, "China")))  
                                   .toDF("c_id", "c_name")
```

```
val df4 = df1.join(df2, df1("c_id") === df2("c_id"))  
df4.show()
```





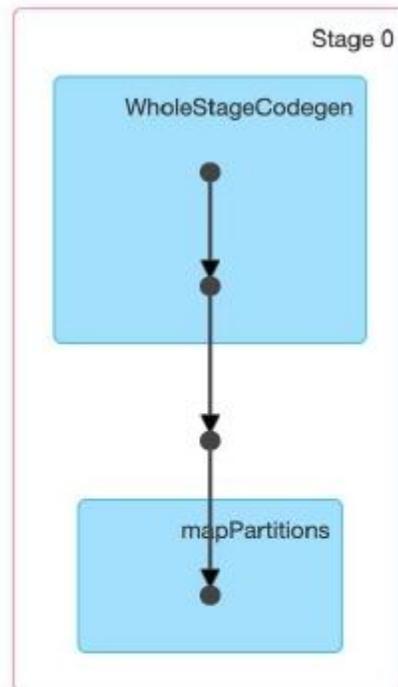

# Spark SQL 执行计划

- Where
- Group by
- Order by

# Spark SQL - Where

```
val df = spark.read.json("file:///usr/lib/spark-current/examples/src/main/resources/people.json")
df.registerTempTable("people")
```

```
select * from people where name='jeff'
```



== Physical Plan ==

\*Project [age#6L, name#7]

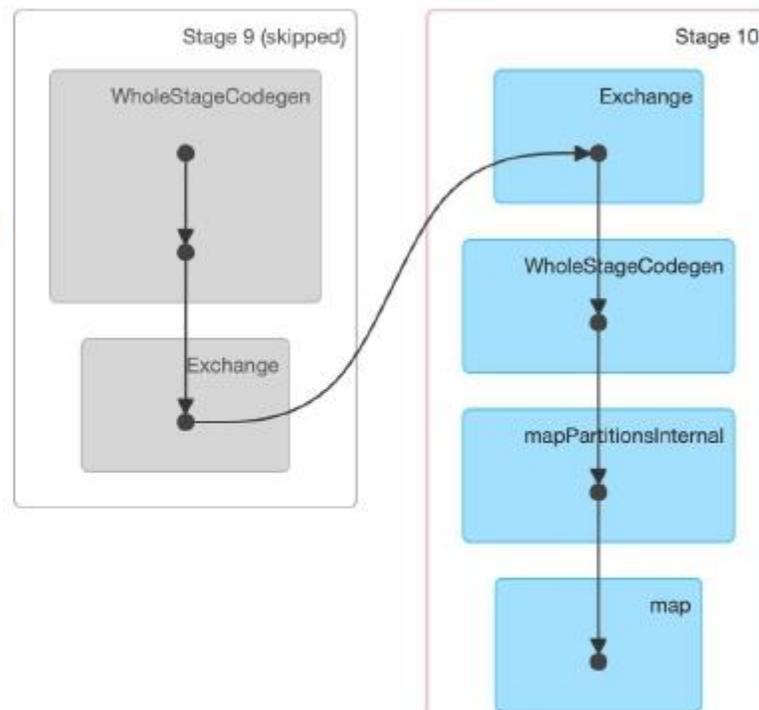
+ - \*Filter (isNotNull(name#7) && (name#7 = jeff))

+ - \*FileScan json [age#6L,name#7] Batched: false, Format: JSON, Location: InMemoryFileIndex[file:/usr/lib/spark-current/examples/src/main/resources/people.json], PartitionFilters: [], PushedFilters: [IsNotNull(name), EqualTo(name,jeff)], ReadSchema: struct<age:bigint,name:string>|

# Spark SQL - Group By

```
val df = spark.read.json("file:///usr/lib/spark-current/examples/src/main/resources/people.json")
df.registerTempTable("people")
```

```
select age, count(1) from people group by age
```



== Physical Plan ==

\* (2) HashAggregate(keys=[age#26L], functions=[count(1)])

+ Exchange hashpartitioning(age#26L, 200)

+ \* (1) HashAggregate(keys=[age#26L], functions=[partial\_count(1)])

+ \* (1) FileScan json [age#26L] Batched: false, Format: JSON, Location: InMemoryFileIndex[file:/usr/lib/spark-current/examples/src/main/resources/people.json], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<ag

# Spark SQL - Order by

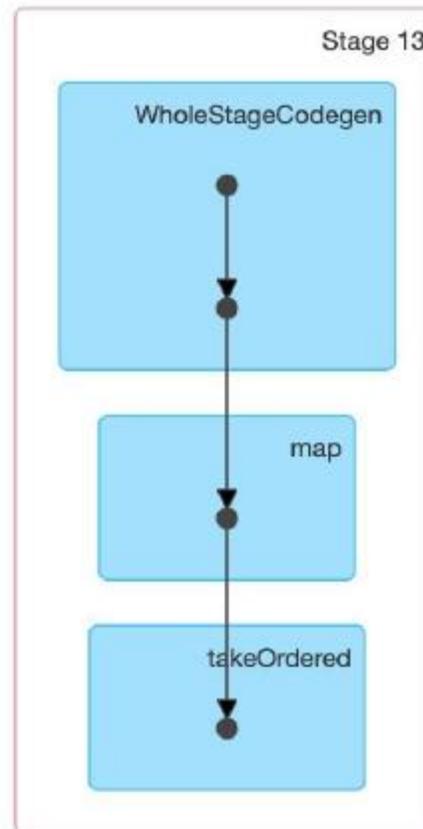
```
val df = spark.read.json("file:///usr/lib/spark-current/examples/src/main/resources/people.json")
df.registerTempTable("people")
```

```
select * from people order by age
```



== Physical Plan ==

```
* (2) Sort [age#64L ASC NULLS FIRST], true, 0
+- Exchange rangepartitioning(age#64L ASC NULLS FIRST, 200)
   +- *(1) FileScan json [age#64L,name#65] Batched: false, Format: JSON, Location: InMemoryFileIndex[file:/usr/li
      b/spark-current/examples/src/main/resources/people.json], PartitionFilters: [], PushedFilters: [], ReadSchema: str
      uct<age:bigint,name:string>|
```



# EMR Studio 实践

---

# EMR Studio 特性

- 兼容开源组件
- 支持连接多个集群
- 适配多个计算引擎
- 通过界面化的方式进行交互式开发和作业调度
- 适用多种大数据应用场景
- 计算存储分离

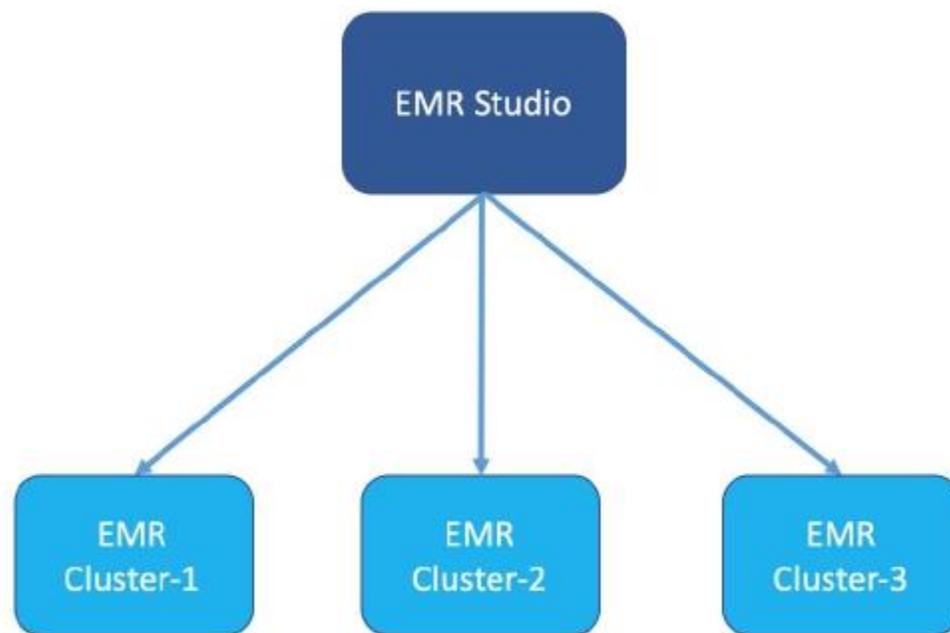
# 兼容开源组件

EMR Studio 在开源软件 Apache Zeppelin, Jupyter Notebook, Apache Airflow 的基础上优化了做了优化和增强。



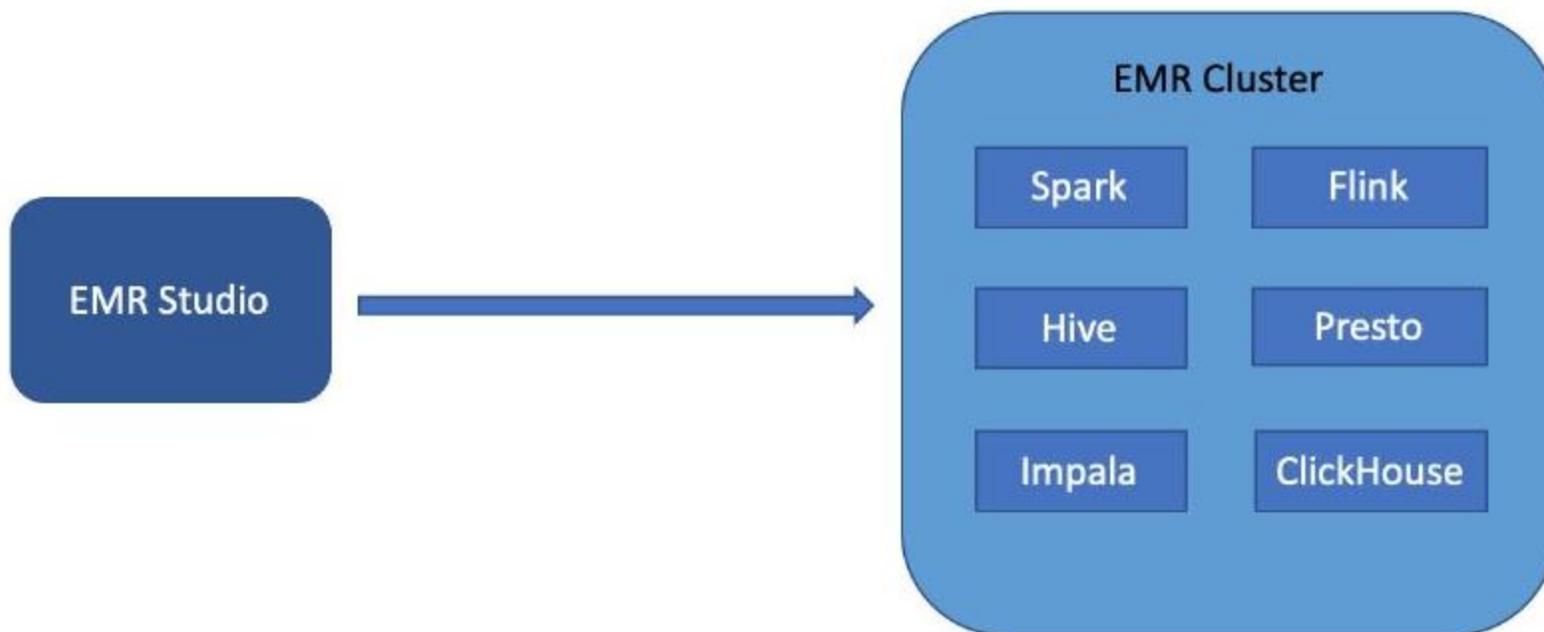
# 支持连接多个集群

一个 EMR Studio 可以连接多个EMR计算集群，您可以很方便地切换计算集群，提交作业到不同的计算集群上运行。



# 适配多个计算引擎

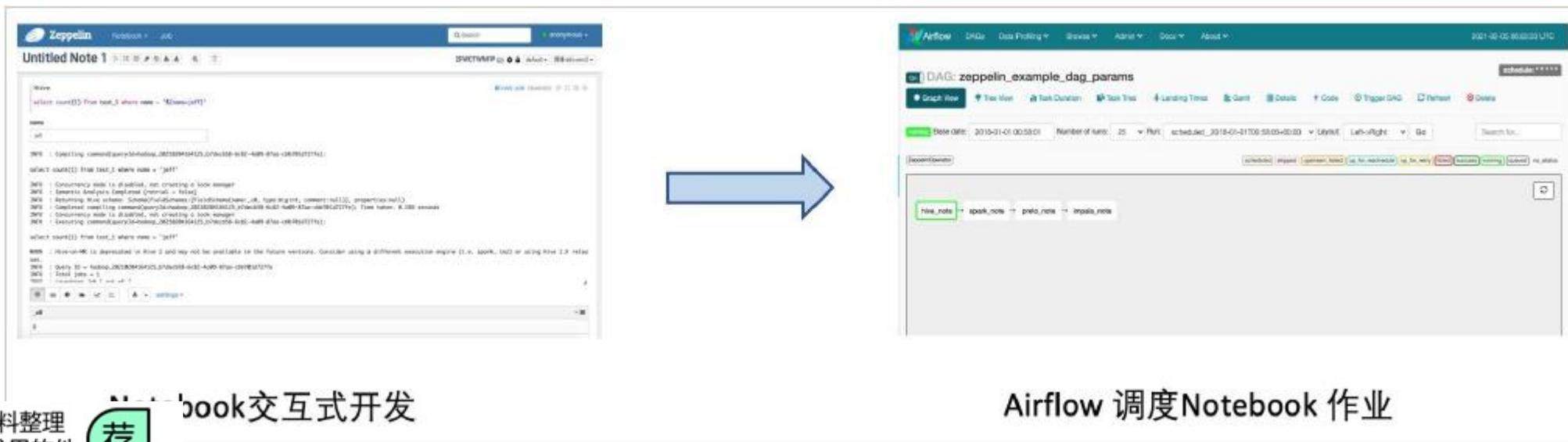
自动适配Hive、Spark、Flink、Presto、Impala和Shell等多个计算引擎，无需复杂配置，多个计算引擎间协同工作



# 交互式开发 + 作业调度无缝衔接

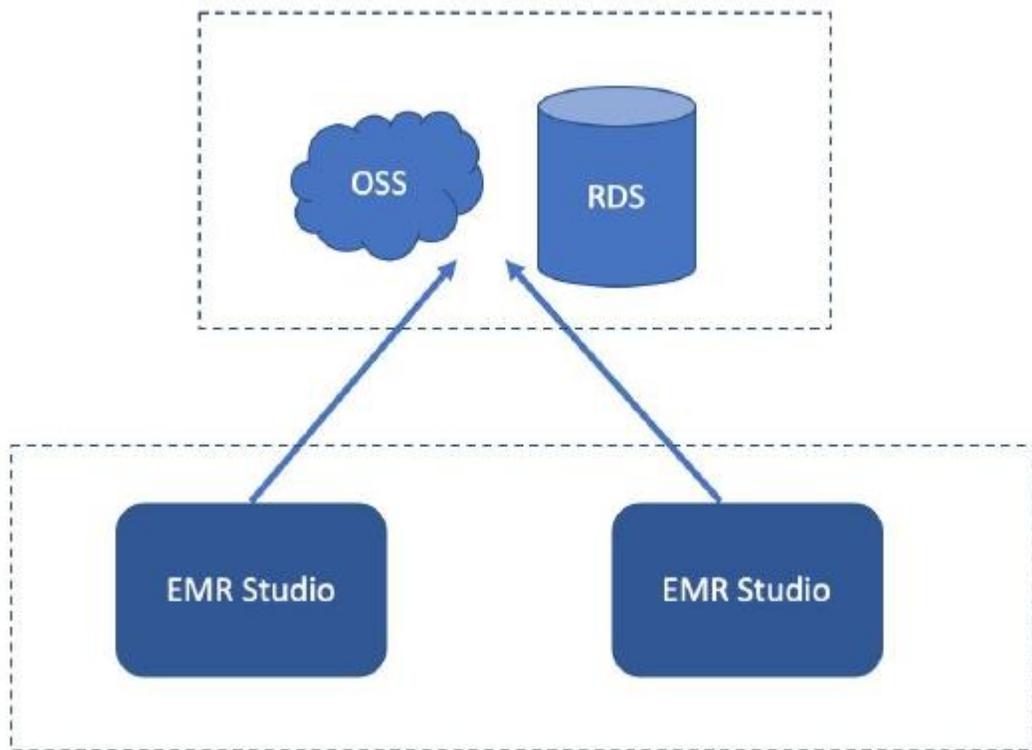
## Notebook + Airflow: 无缝衔接开发环节和生产调度环节

- 利用交互式开发模式可以快速验证作业的正确性。
- 在Airflow里调度Notebook作业，最大程度得保证开发环境和生产环境的一致性，防止由于开发阶段和生产阶段环境不一致而导致的问题。



# 计算存储分离

- 所有数据都保存在OOS上，包括：
  - 用户Notebook代码
  - 调度作业Log
- 即使集群销毁，也可以重建集群轻松恢复数据



# 适用多种大数据应用场景

- 大数据处理 ETL
- 交互式数据分析
- 机器学习
- 实时计算

