IT HOME CONFERENCE

IT Home

Amazon Deequ
vs
Apache Griffin
in
AWS EMR CloudFormation

by Gregory Kulga

EPAM 2020

IT Home

Homemade cookies are always better than Bought

Gregory Kulga

Data Quality Engineer

- Core skills: DWBI, data quality, data analysis
- 3-year experience in Big data testing, Automation testing, Data analysis
- Hobbies: quizzes, camping, snowboard
That's not me. Not yet...

IT Home

Agenda

60 mins

Big Data paradigm

Project structure and Targets

DQ Goals and opportunities

AMAZON Deequ As a rapid tool

Apache Griffin as a COMPLEX tool

Overall Estimation

IT Home

Big Data paradigm

Big Data paradigm

On what it is based

Volume

The quantity of generated
and stored data

Velocity

The speed at which the
data is generated and
processed

Variety

Type and nature of the
data.
Semi-structured and
unstructured

IT Home

IT Home

Project Structure

Our customer – Lead American health insurance provider

Business Objective is to predict and assess the possible increase or decrease diseases among specific groups of people

These Groups of people are divided by
- age
- gender
- race
- genetic diseases
- features of growth and maturation
- features of the working environment

IT Home

Machine Learning model stages

The based data split up for 3 stages

Training Set

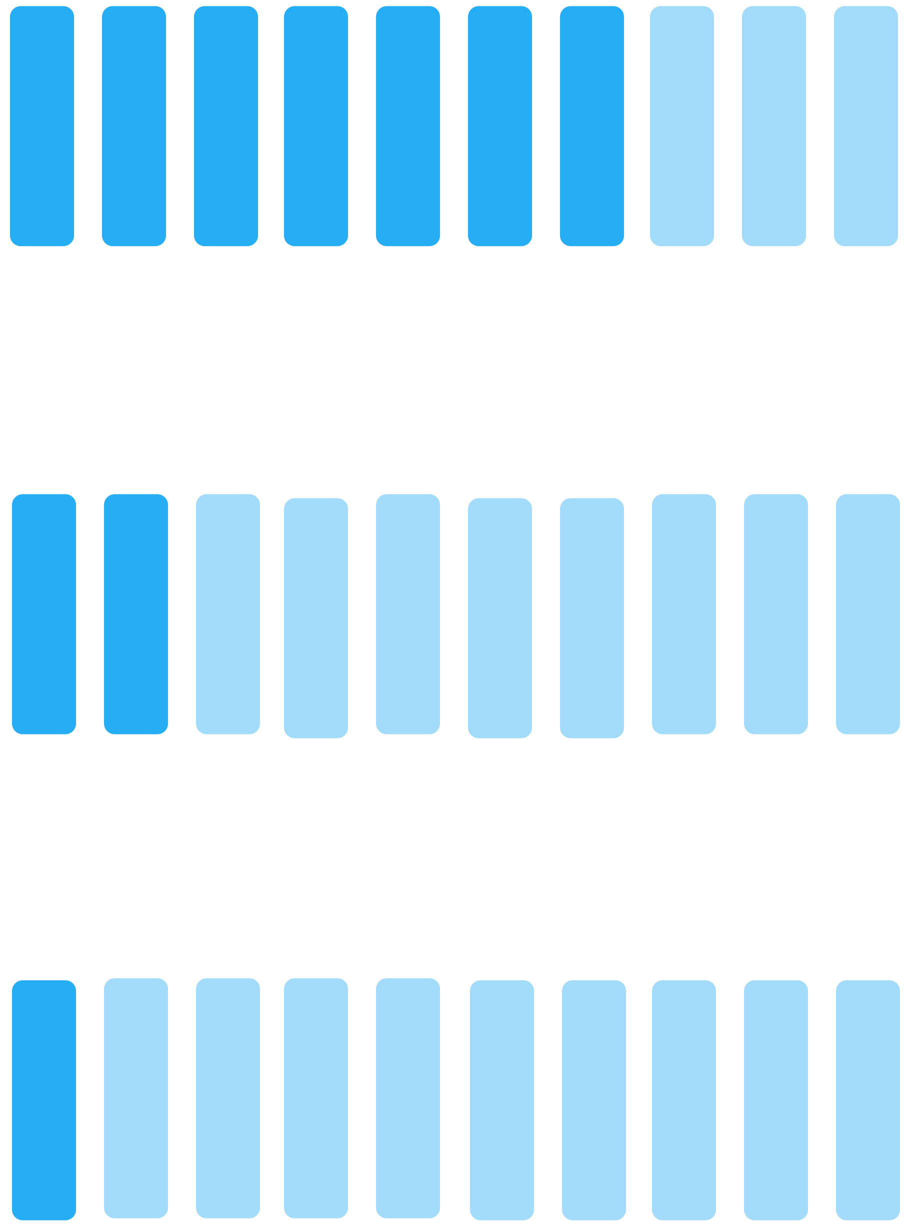
Prediction is created

Validation Set

Model check for stability

Test Set

Model check as prediction and compared with previous data



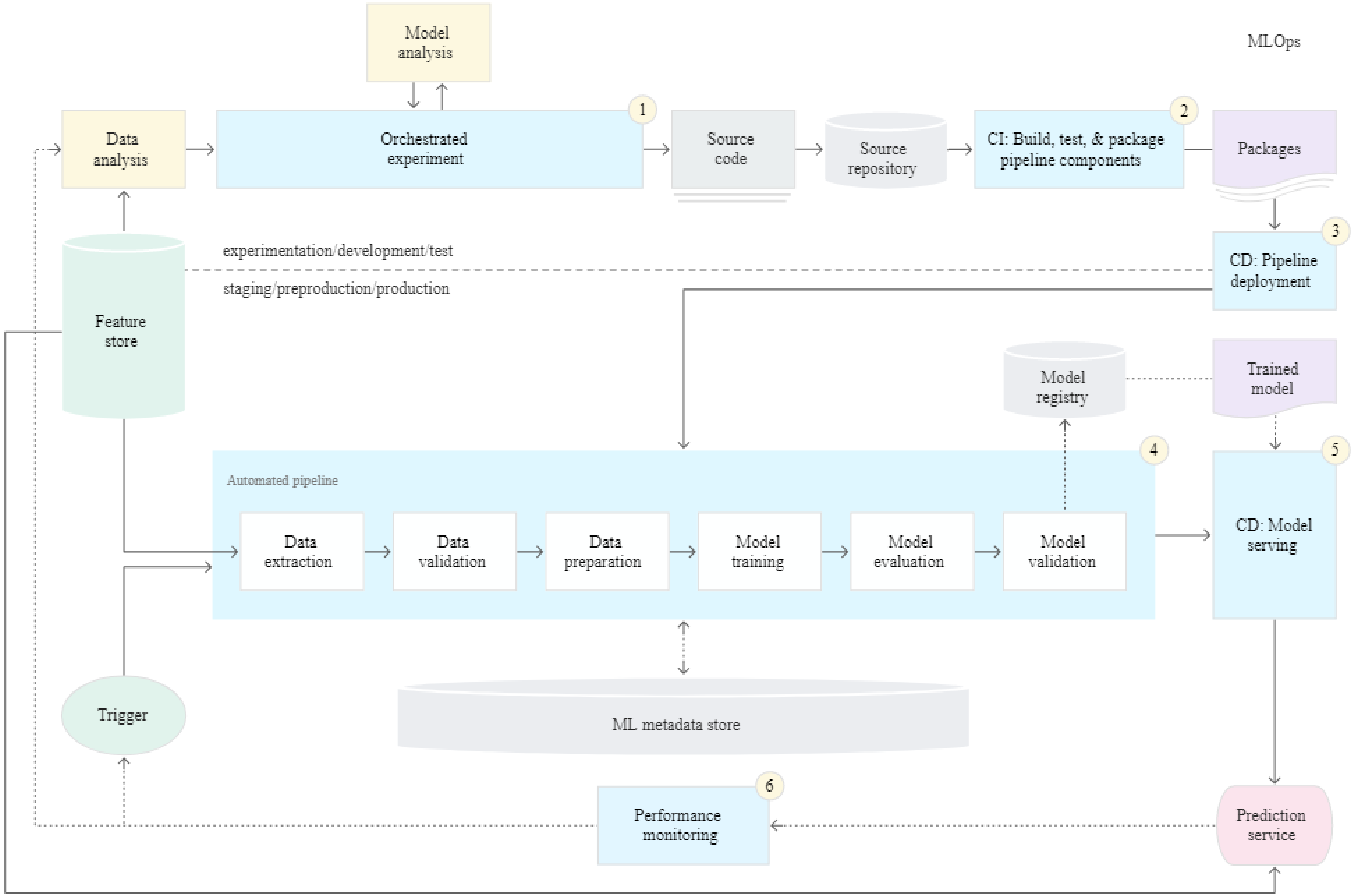
70%

20%

10%

Data train model scheme

If Something goes wrong Let's retrain!

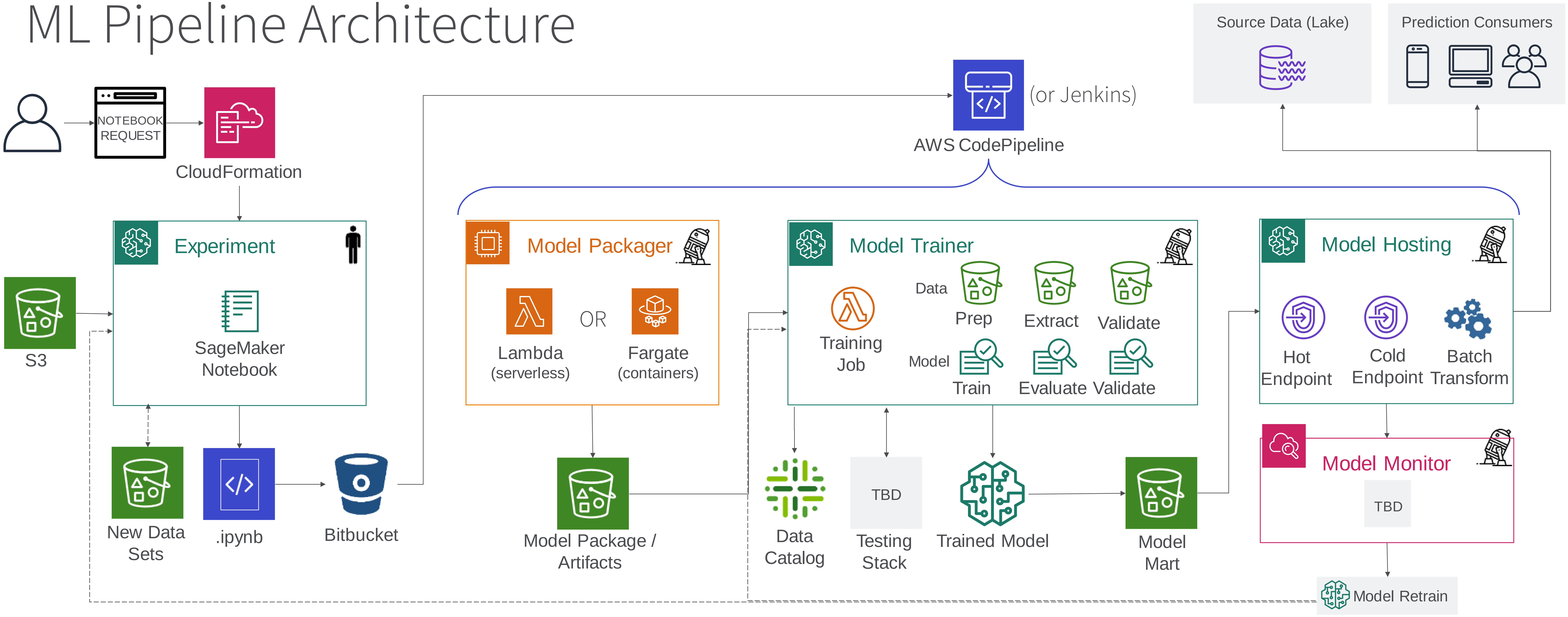


Manual Stack

How should it work?

Automation Workflow

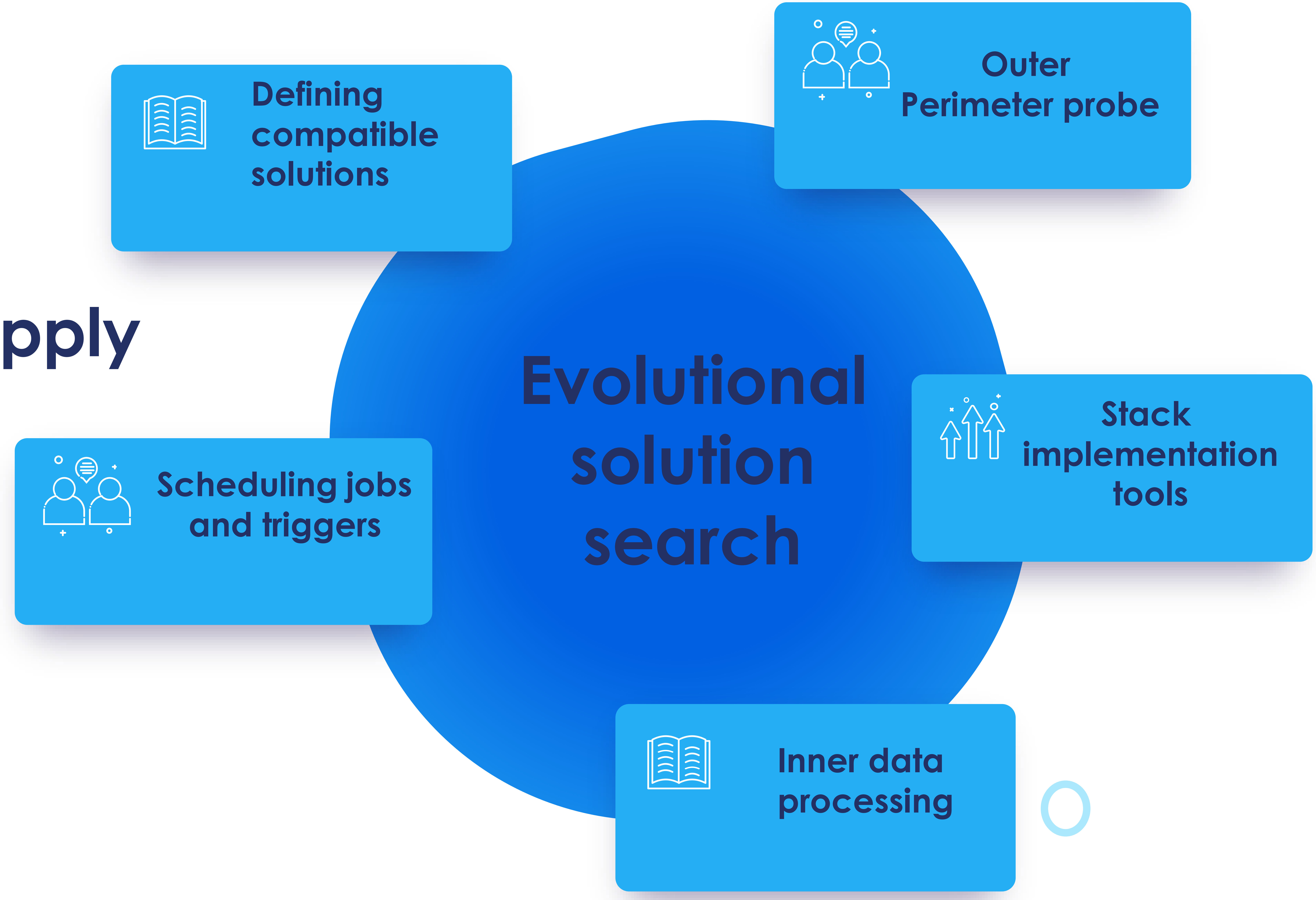
ML Pipeline Architecture



- Experiment Request
- CloudFormation Script
- S3 Data Bucket
- Experimental Environment
- Bitbucket Repository
- CodePipeline
- Model Packager
- Model Trainer
- Data Catalog
- Model Mart
- Model Hosting
- Model Monitor
- Model Retraining

DQ Goals and opportunities

DQ apply



Evolutional solution search

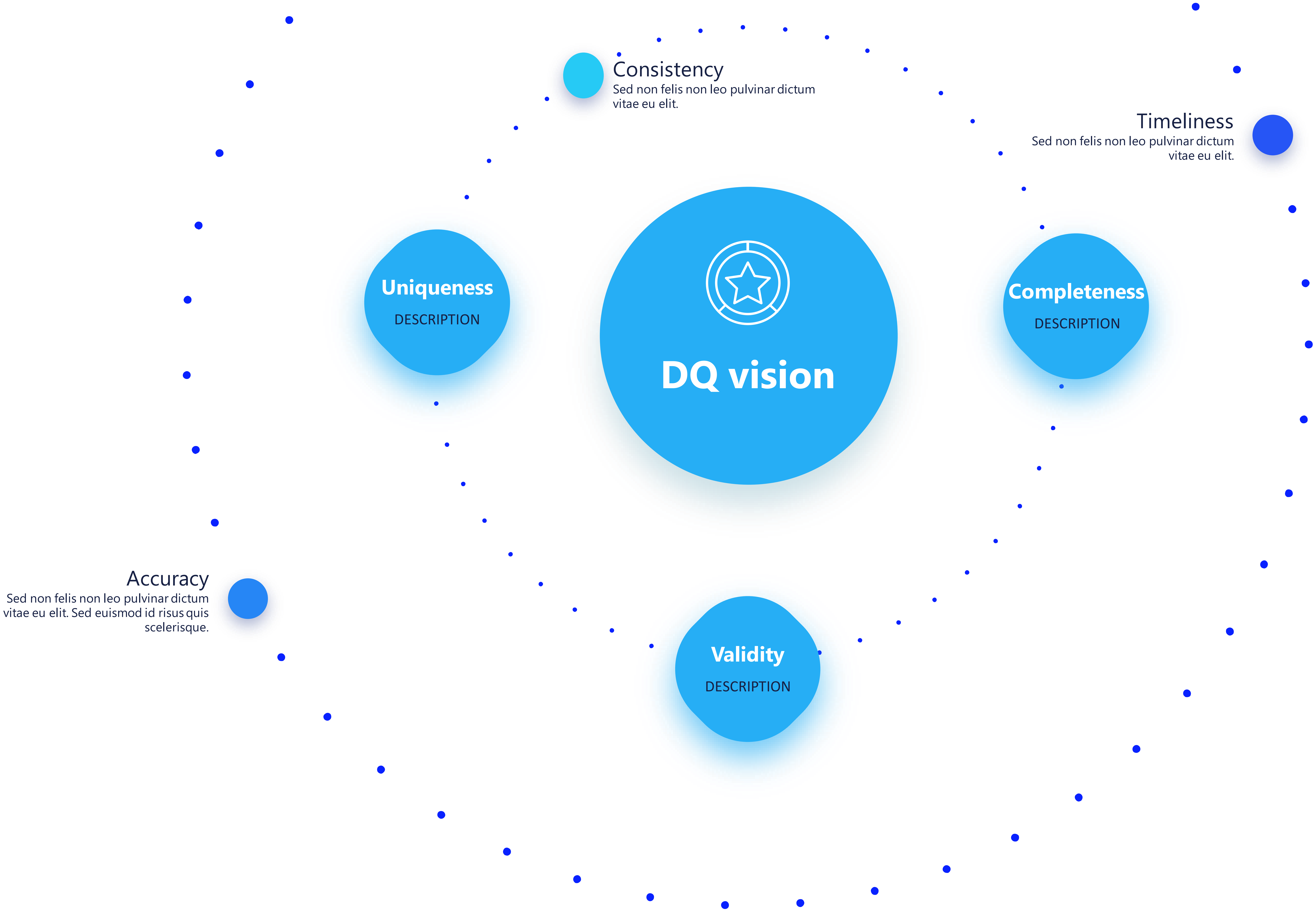
Defining compatible solutions

Outer Perimeter probe

Scheduling jobs and triggers

Stack implementation tools

Inner data processing



Consistency

Sed non felis non leo pulvinar dictum vitae eu elit.

Timeliness

Sed non felis non leo pulvinar dictum vitae eu elit.

Uniqueness

DESCRIPTION

DQ vision

Completeness

DESCRIPTION

Accuracy

Sed non felis non leo pulvinar dictum vitae eu elit. Sed euismod id risus quis scelerisque.

Validity

DESCRIPTION

Amazon Deequ

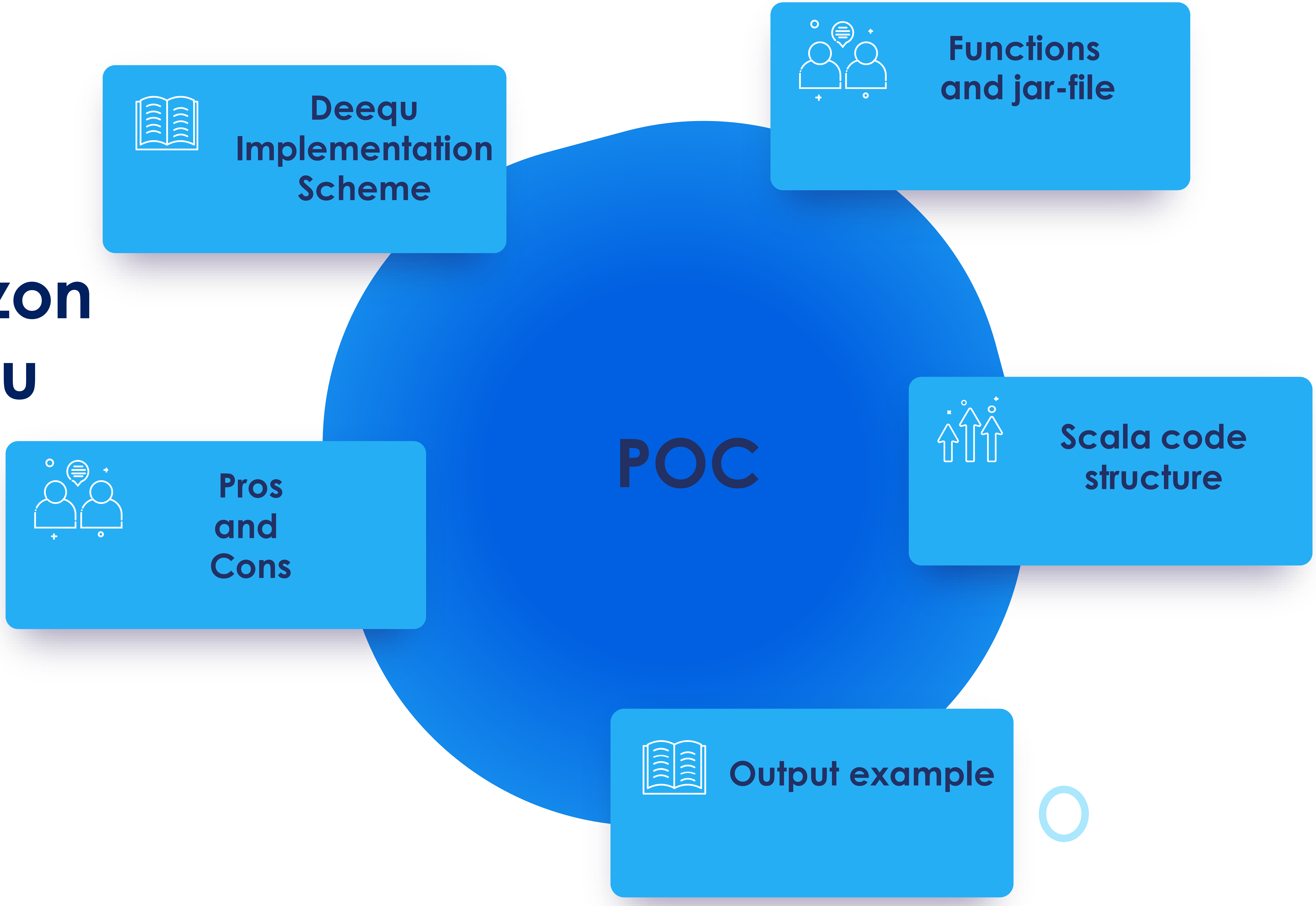
IT Home

What could be better than Amazon (Deequ)?

Here we go!

You’re exceptionally awesome but we at least exist in this reality

Amazon Deequ



POC

Deequ Implementation Scheme

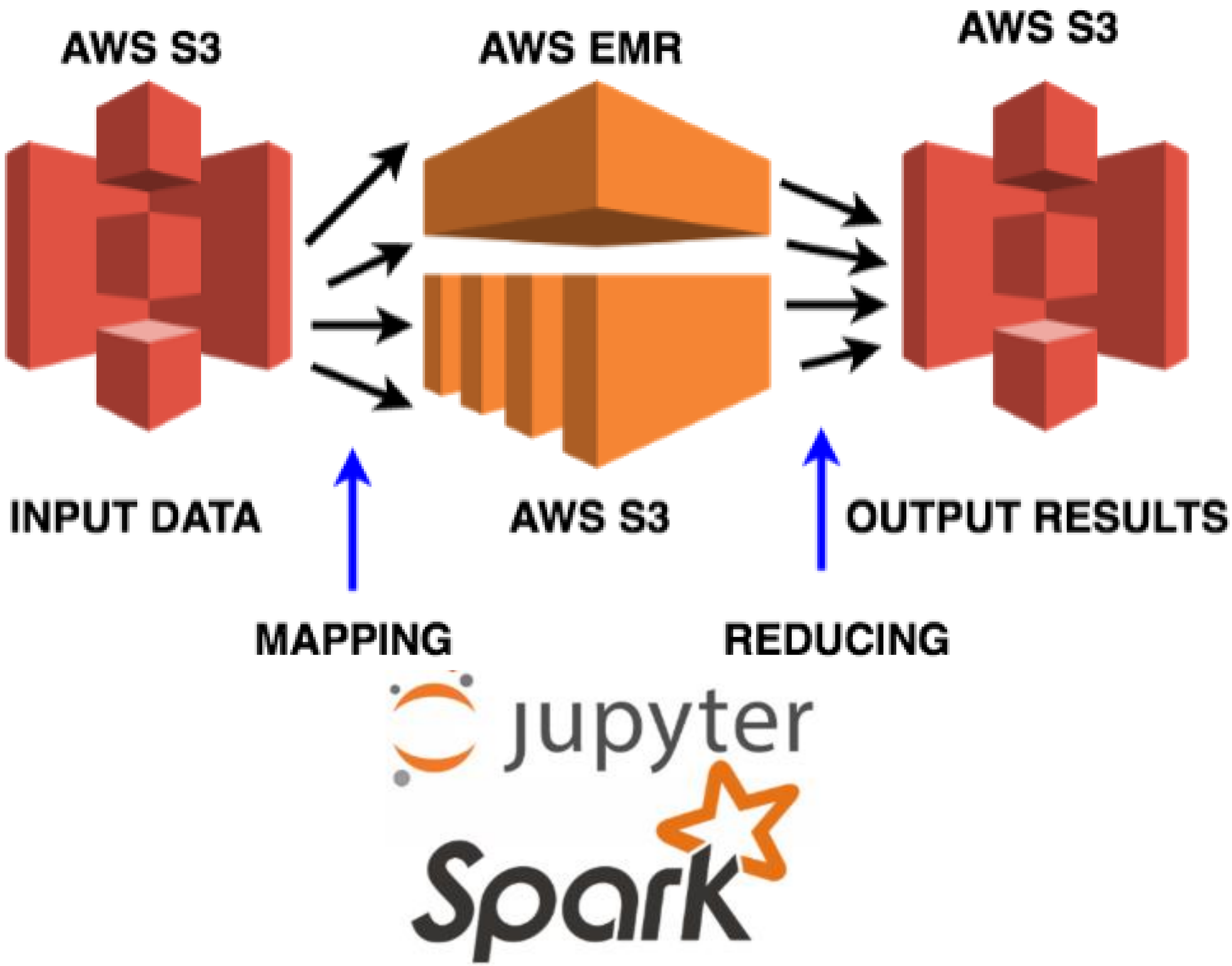
Functions and jar-file

Scala code structure

Output example

Pros and Cons

Deequ Implementation Scheme

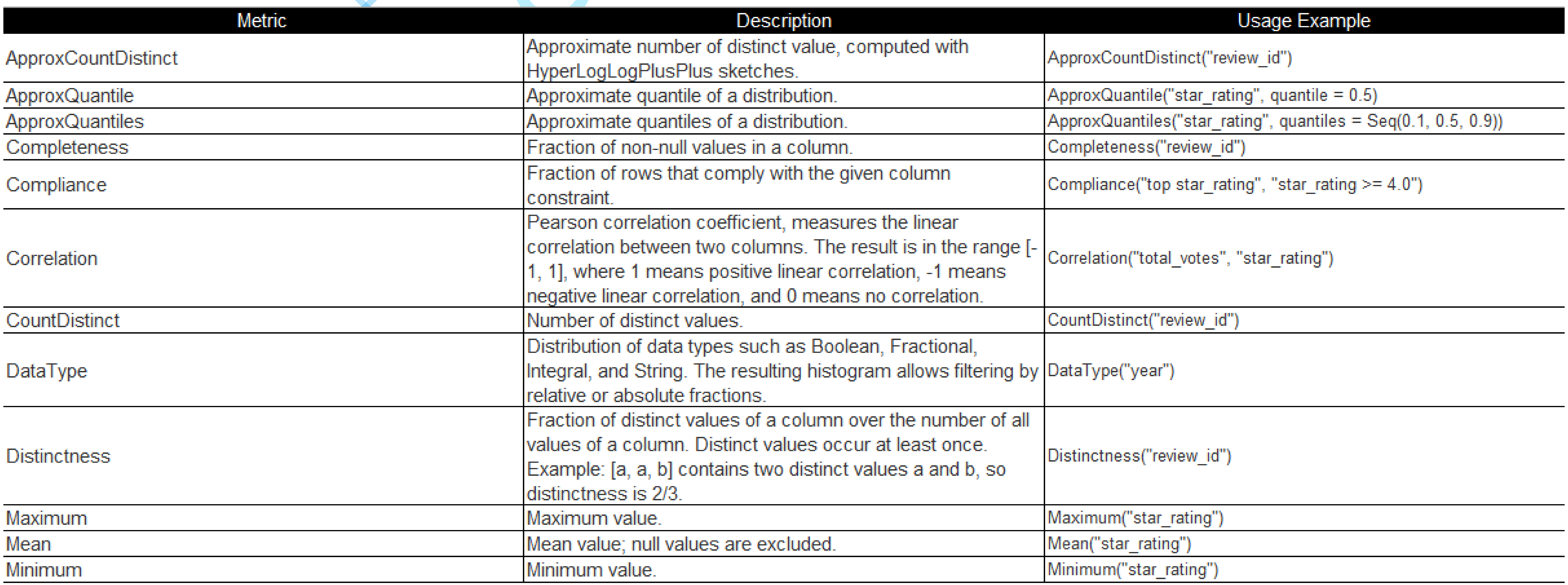


Sequence execution

- Source data is stored in AWS Data Lake S3
- Load and install .jar-files for Deequ libs
- Save scala code in S3 via Jupyter Notebook cells executed on spark-shell
- Use EMR Terminal via Spark-shell to execute stored .scala
- Get the result into EMR Terminal

Functions Variety

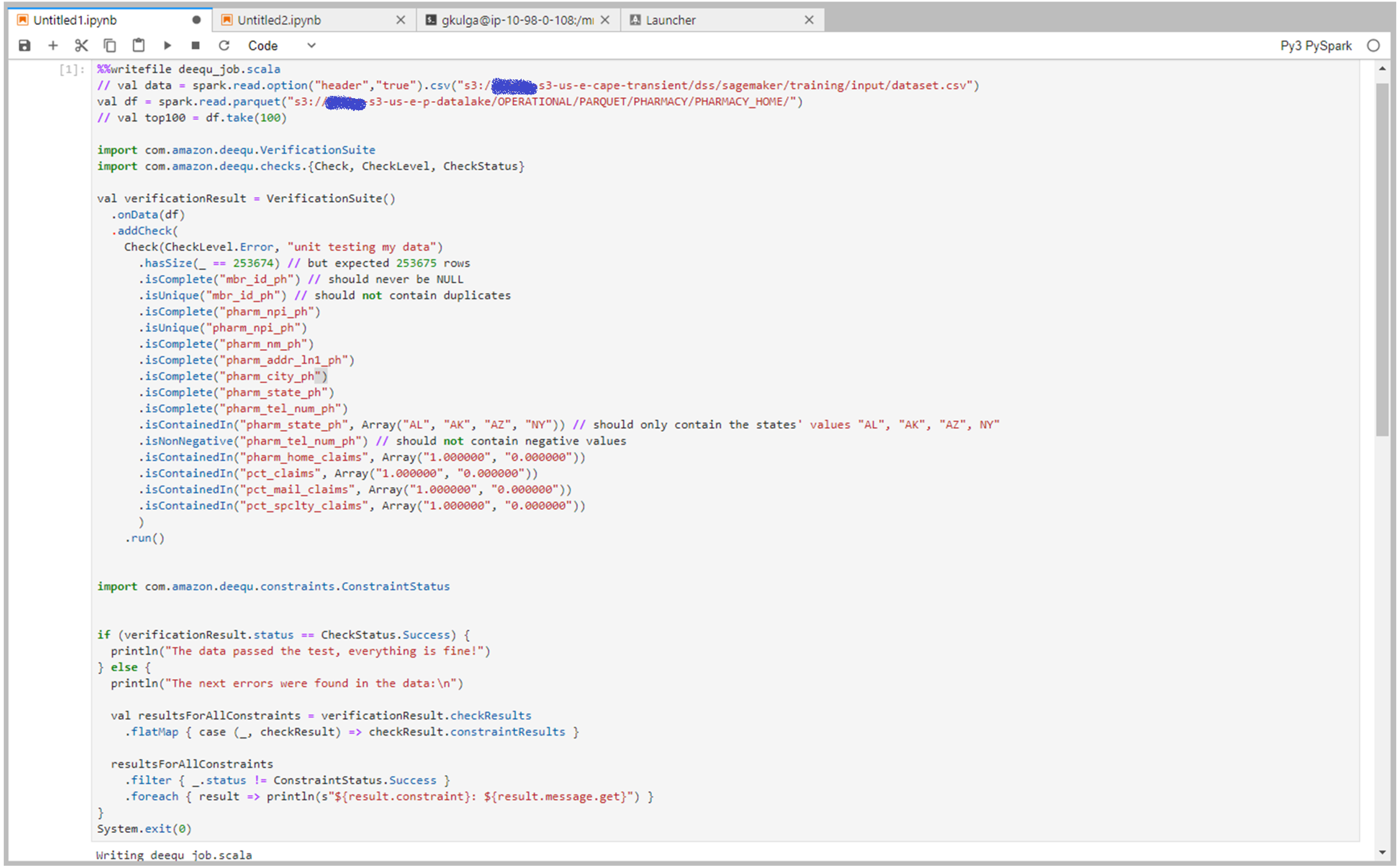
IT Home



Metric	Description	Usage Example
ApproxCountDistinct	Approximate number of distinct value, computed with HyperLogLogPlusPlus sketches.	ApproxCountDistinct("review_id")
ApproxQuantile	Approximate quantile of a distribution.	ApproxQuantile("star_rating", quantile = 0.5)
ApproxQuantiles	Approximate quantiles of a distribution.	ApproxQuantiles("star_rating", quantiles = Seq(0.1, 0.5, 0.9))
Completeness	Fraction of non-null values in a column.	Completeness("review_id")
Compliance	Fraction of rows that comply with the given column constraint.	Compliance("top star_rating", "star_rating >= 4.0")
Correlation	Pearson correlation coefficient, measures the linear correlation between two columns. The result is in the range [-1, 1], where 1 means positive linear correlation, -1 means negative linear correlation, and 0 means no correlation.	Correlation("total_votes", "star_rating")
CountDistinct	Number of distinct values.	CountDistinct("review_id")
DataType	Distribution of data types such as Boolean, Fractional, Integral, and String. The resulting histogram allows filtering by relative or absolute fractions.	DataType("year")
Distinctness	Fraction of distinct values of a column over the number of all values of a column. Distinct values occur at least once. Example: [a, a, b] contains two distinct values a and b, so distinctness is 2/3.	Distinctness("review_id")
Maximum	Maximum value.	Maximum("star_rating")
Mean	Mean value; null values are excluded.	Mean("star_rating")
Minimum	Minimum value.	Minimum("star_rating")

Deequ scala code

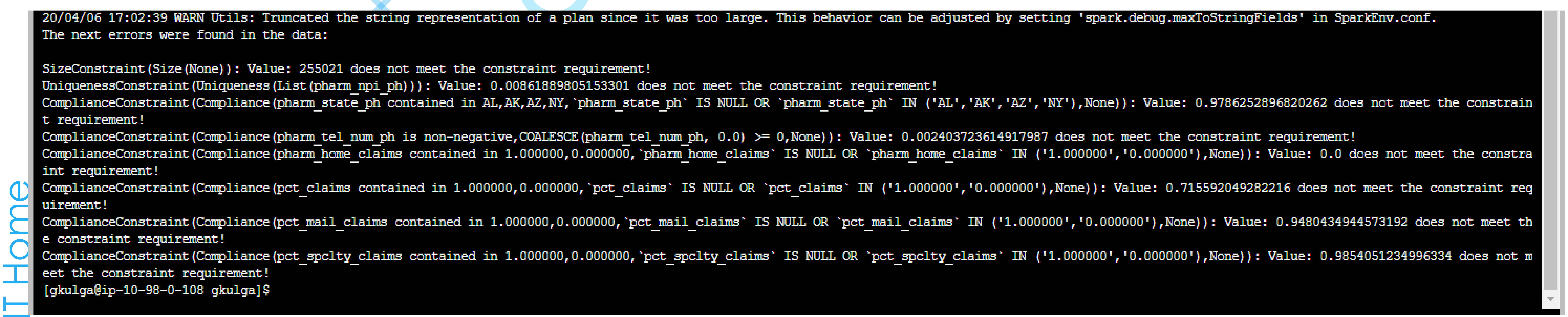
IT Home



```
%%writefile deequ_job.scala
// val data = spark.read.option("header","true").csv("s3:/[redacted]s3-us-e-cape-transient/dss/sagemaker/training/input/dataset.csv")
val df = spark.read.parquet("s3://[redacted]s3-us-e-p-datalake/OPERATIONAL/PARQUET/PHARMACY/PHARMACY_HOME/")
// val top100 = df.take(100)

import com.amazon.deequ.VerificationSuite
import com.amazon.deequ.checks.{Check, CheckLevel, CheckStatus}

val verificationResult = VerificationSuite()
  .onData(df)
  .addCheck(
    Check(CheckLevel.Error, "unit testing my data")
      .hasSize(_ == 253674) // but expected 253675 rows
      .isComplete("mbr_id_ph") // should never be NULL
      .isUnique("mbr_id_ph") // should not contain duplicates
      .isComplete("pharm_npi_ph")
      .isUnique("pharm_npi_ph")
      .isComplete("pharm_nm_ph")
      .isComplete("pharm_addr_ln1_ph")
      .isComplete("pharm_city_ph")
      .isComplete("pharm_state_ph")
      .isComplete("pharm_tel_num_ph")
      .isContainedIn("pharm_state_ph", Array("AL", "AK", "AZ", "NY")) // should only contain the states' values "AL", "AK", "AZ", NY"
      .isNonNegative("pharm_tel_num_ph") // should not contain negative values
      .isContainedIn("pharm_home_claims", Array("1.000000", "0.000000"))
      .isContainedIn("pct_claims", Array("1.000000", "0.000000"))
      .isContainedIn("pct_mail_claims", Array("1.000000", "0.000000"))
      .isContainedIn("pct_spclty_claims", Array("1.000000", "0.000000"))
      )
    .run()


import com.amazon.deequ.constraints.ConstraintStatus


if (verificationResult.status == CheckStatus.Success) {
  println("The data passed the test, everything is fine!")
} else {
  println("The next errors were found in the data:\n")

  val resultsForAllConstraints = verificationResult.checkResults
    .flatMap { case (_, checkResult) => checkResult.constraintResults }

  resultsForAllConstraints
    .filter { _.status != ConstraintStatus.Success }
    .foreach { result => println(s"${result.constraint}: ${result.message.get}") }
}
System.exit(0)
```

```
Writing deequ_job.scala
```

Results via Amazon Deequ



```
20/04/06 17:02:39 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.
The next errors were found in the data:

SizeConstraint(Size(None)): Value: 255021 does not meet the constraint requirement!
UniquenessConstraint(Uniqueness(List(pharm_npi_ph))): Value: 0.00861889805153301 does not meet the constraint requirement!
ComplianceConstraint(Compliance(pharm_state_ph contained in AL,AK,AZ,NY,`pharm_state_ph` IS NULL OR `pharm_state_ph` IN ('AL','AK','AZ','NY'),None)): Value: 0.9786252896820262 does not meet the constrain
t requirement!
ComplianceConstraint(Compliance(pharm_tel_num_ph is non-negative,COALESCE(pharm_tel_num_ph, 0.0) >= 0,None)): Value: 0.002403723614917987 does not meet the constraint requirement!
ComplianceConstraint(Compliance(pharm_home_claims contained in 1.000000,0.000000,`pharm_home_claims` IS NULL OR `pharm_home_claims` IN ('1.000000','0.000000'),None)): Value: 0.0 does not meet the constra
int requirement!
ComplianceConstraint(Compliance(pct_claims contained in 1.000000,0.000000,`pct_claims` IS NULL OR `pct_claims` IN ('1.000000','0.000000'),None)): Value: 0.715592049282216 does not meet the constraint req
uirement!
ComplianceConstraint(Compliance(pct_mail_claims contained in 1.000000,0.000000,`pct_mail_claims` IS NULL OR `pct_mail_claims` IN ('1.000000','0.000000'),None)): Value: 0.9480434944573192 does not meet th
e constraint requirement!
ComplianceConstraint(Compliance(pct_spclty_claims contained in 1.000000,0.000000,`pct_spclty_claims` IS NULL OR `pct_spclty_claims` IN ('1.000000','0.000000'),None)): Value: 0.9854051234996334 does not m
eet the constraint requirement!
[gkulga@ip-10-98-0-108 gkulga]$
```

IT Home

Results could be stored in S3

C

b

c

Amazon Deequ Pros and Cons

IT Home

Pros

- Small and easy-supportable library
- SQL-like functions supported via scala syntax
- Fast deliverable code:

  one script – one file – one run

Cons

- Console output
- No documentation with alerts, time-postponed functions
- Poorly configurable options

Random squirrel

- If you tired - look right

- If you distracted - look right

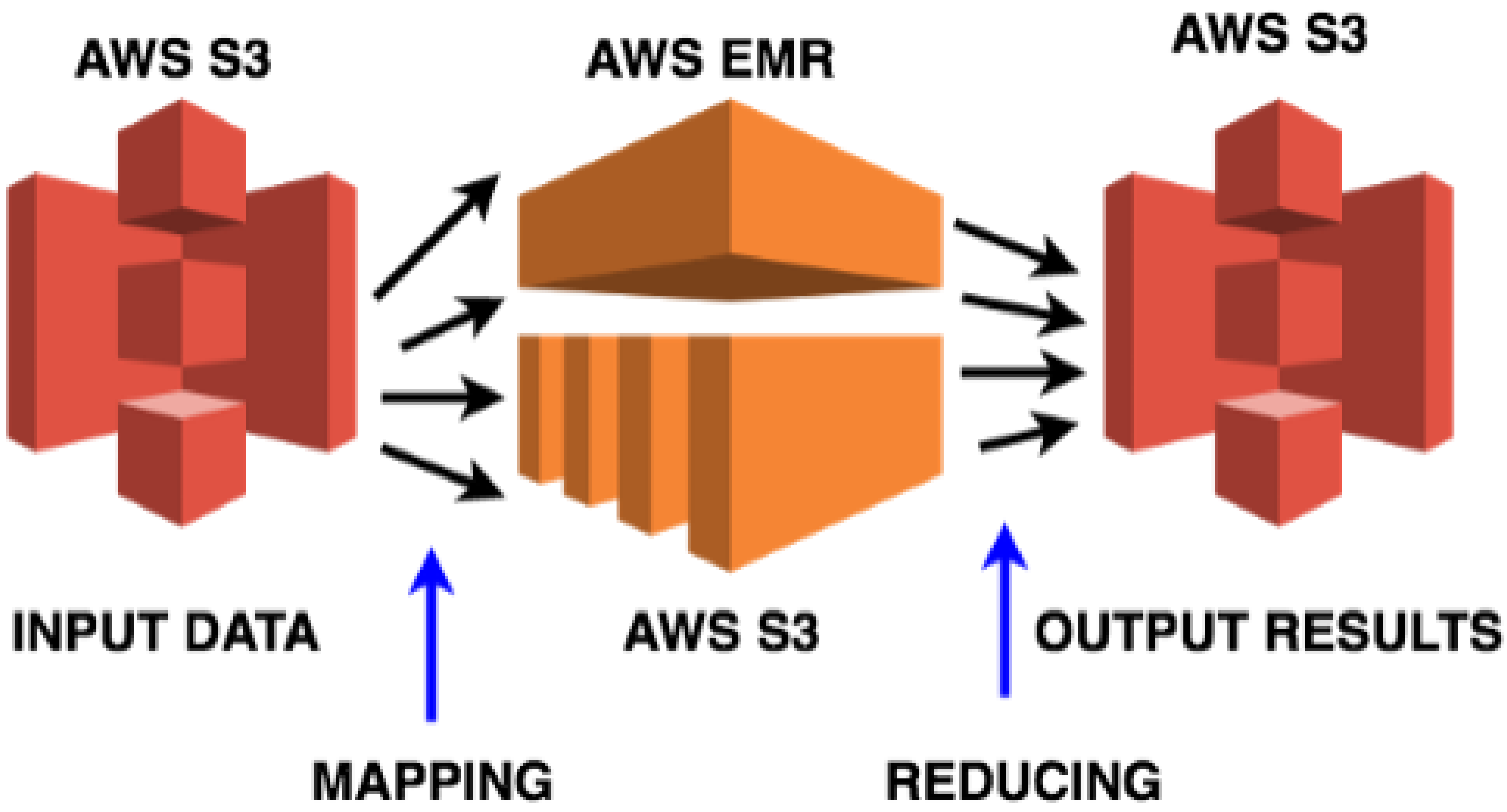
Don’t be rude – wave back!

Apache Griffin

Apache Griffin

- Production implementation schema
- Configuration files structure
- Pros and Cons

Griffin Implementation Scheme

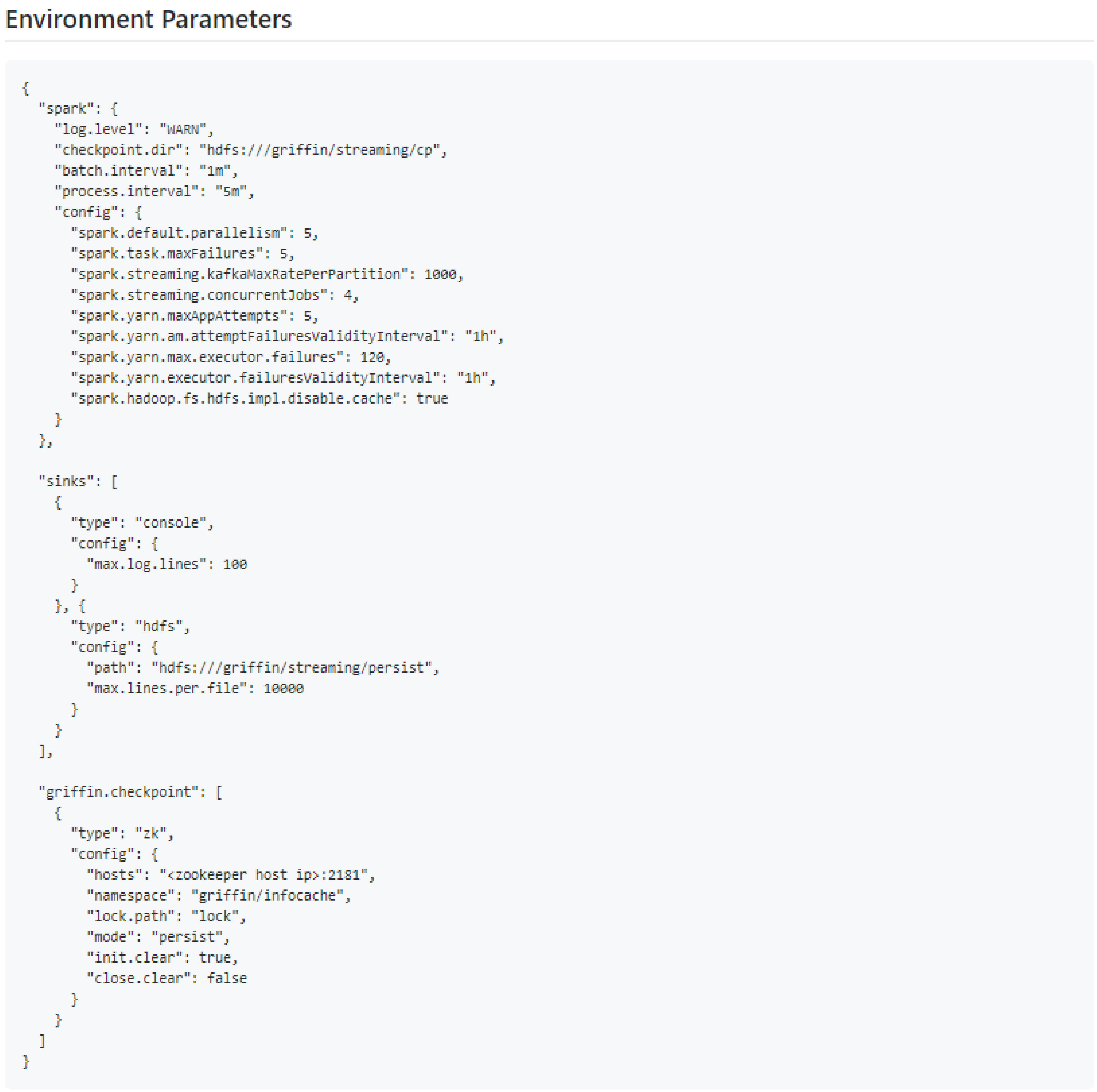


Sequence execution

- Source data is stored in AWS Data Lake S3
- Load and install .jar-files for Griffin libs
- Save json-files in S3 via Jupyter Notebook cells executed on spark-shell
- Use EMR Terminal via Spark-shell to execute stored jsons
- Get the result into S3 storage or console output

Possible Features

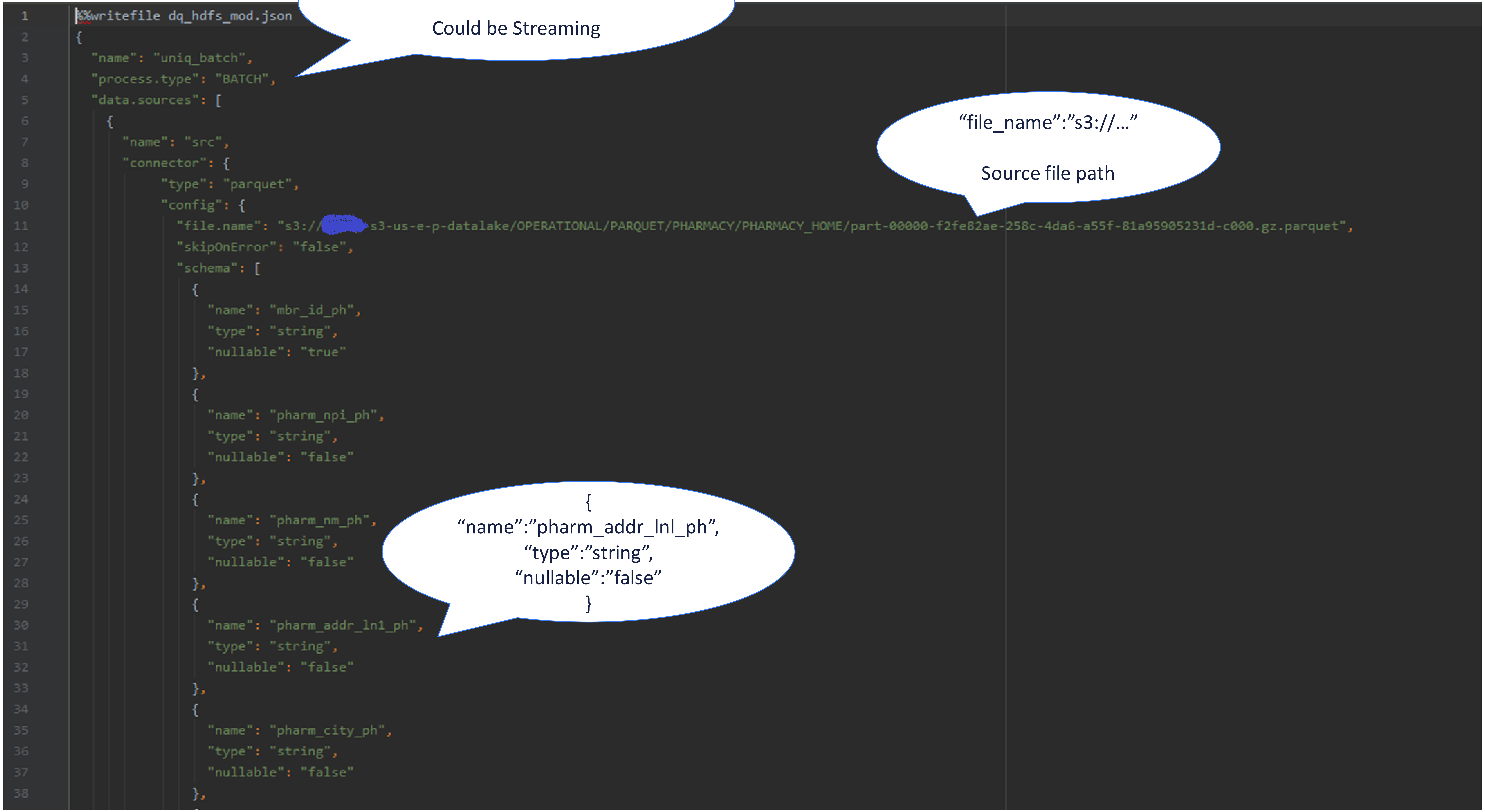
- Batch and Streaming are included both
- HDFS, Cloud, Console, Elasticsearch ways as options for output
- Combining functions rule created
- Scheme must be described: in mismatching case run will fail with relevant error



Environment Parameters

```
{
  "spark": {
    "log.level": "WARN",
    "checkpoint.dir": "hdfs:///griffin/streaming/cp",
    "batch.interval": "1m",
    "process.interval": "5m",
    "config": {
      "spark.default.parallelism": 5,
      "spark.task.maxFailures": 5,
      "spark.streaming.kafkaMaxRatePerPartition": 1000,
      "spark.streaming.concurrentJobs": 4,
      "spark.yarn.maxAppAttempts": 5,
      "spark.yarn.am.attemptFailuresValidityInterval": "1h",
      "spark.yarn.max.executor.failures": 120,
      "spark.yarn.executor.failuresValidityInterval": "1h",
      "spark.hadoop.fs.hdfs.impl.disable.cache": true
    }
  },

  "sinks": [
    {
      "type": "console",
      "config": {
        "max.log.lines": 100
      }
    }, {
      "type": "hdfs",
      "config": {
        "path": "hdfs:///griffin/streaming/persist",
        "max.lines.per.file": 10000
      }
    }
  ],

  "griffin.checkpoint": [
    {
      "type": "zk",
      "config": {
        "hosts": "<zookeeper host ip>:2181",
        "namespace": "griffin/infocache",
        "lock.path": "lock",
        "mode": "persist",
        "init.clear": true,
        "close.clear": false
      }
    }
  ]
}
```

:Data.json



```
%%writefile dq_hdfs_mod.json
{
  "name": "uniq_batch",
  "process.type": "BATCH",
  "data.sources": [
    {
      "name": "src",
      "connector": {
           "type": "parquet",
           "config": {
             "file.name": "s3://[redacted]s3-us-e-p-datalake/OPERATIONAL/PARQUET/PHARMACY/PHARMACY_HOME/part-00000-f2fe82ae-258c-4da6-a55f-81a95905231d-c000.gz.parquet",
             "skipOnError": "false",
             "schema": [
               {
                 "name": "mbr_id_ph",
                 "type": "string",
                 "nullable": "true"
               },
               {
                 "name": "pharm_npi_ph",
                 "type": "string",
                 "nullable": "false"
               },
               {
                 "name": "pharm_nm_ph",
                 "type": "string",
                 "nullable": "false"
               },
               {
                 "name": "pharm_addr_ln1_ph",
                 "type": "string",
                 "nullable": "false"
               },
               {
                 "name": "pharm_city_ph",
                 "type": "string",
                 "nullable": "false"
               },
```

“process.type”:”BATCH”

Could be Streaming

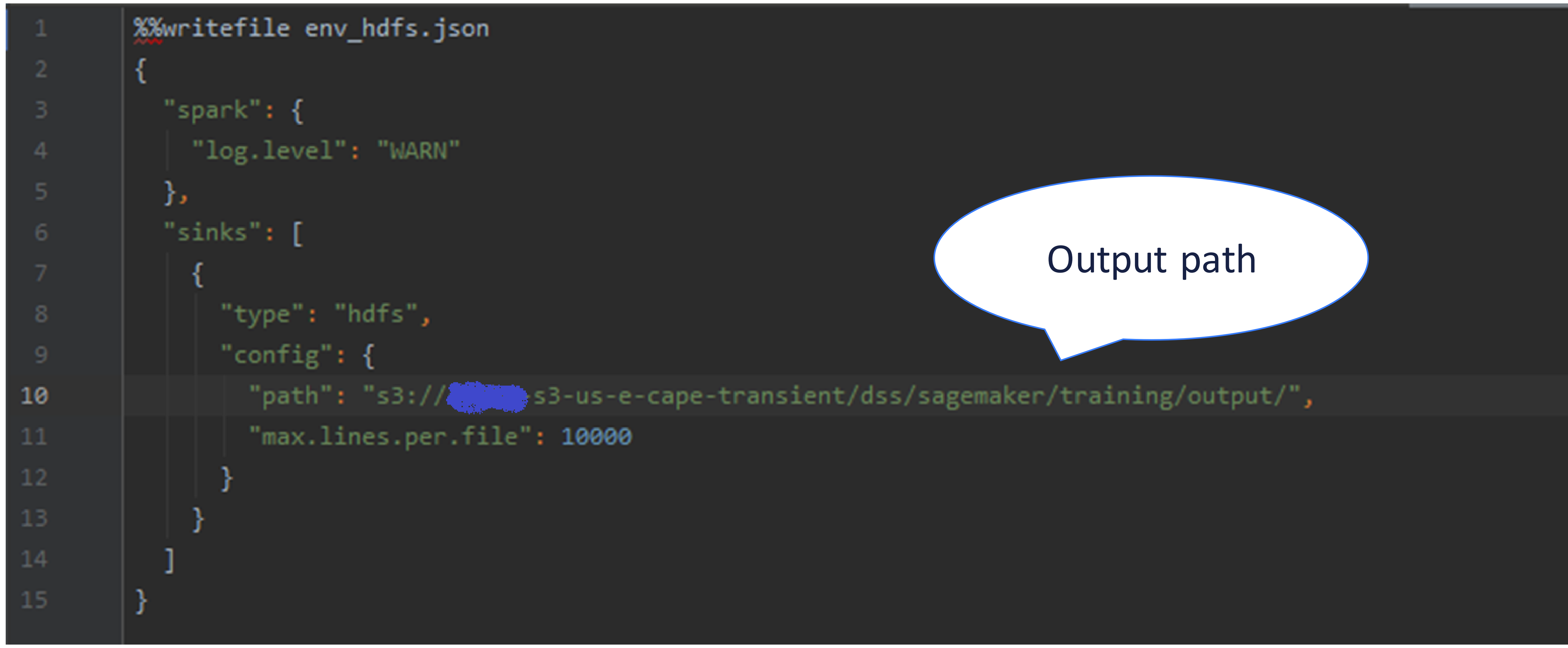
“file_name”:”s3://…”

Source file path

{
“name”:”pharm_addr_lnl_ph”,
“type”:”string”,
“nullable”:”false”
}

Environment.json

- Structure “env” defined in main file separated to another file and described in it;
- Technology and path according to it should be written;
- Output file size in lines must be defined in structure.



```
%%writefile env_hdfs.json
{
  "spark": {
    "log.level": "WARN"
  },
  "sinks": [
    {
      "type": "hdfs",
      "config": {
        "path": "s3://[redacted]s3-us-e-cape-transient/dss/sagemaker/training/output/",
        "max.lines.per.file": 10000
      }
    }
  ]
}
```

Output path

Apache Griffin Pros and Cons

Pros

- Wide functionality
- Data-driven tool
- Clear mapping
- Ability to create alerts and notifications
- Two deeply configurable jsons

Cons

- Open-source tool
- Documentation isn’t fully updated
- Difficulties with manage and support

Overall Estimation

Amazon Deequ vs Apache Griffin

Amazon Deequ

- Small and easy-supportable library
- SQL-like functions syntax via scala
- Fast deliverable code:

  one script – one file – one run
- No alerts and notifications at start
- No mapping

Apache Griffin

- Complex library
- Structured functions
- Two deeply configurable .json-files:

  first - for data, second - for environment
- Ability to create alerts and notifications
- Clear mapping at start

Amazon Deequ vs Apache Griffin

- Amazon Deequ

- Apache Griffin

WARNING

THIS STORY CONTAINS
GRAPHIC IMAGERY
READ AT YOUR OWN RISK

IT Home

Thank you for attention