Beyond unit tests: Testing for Spark/Hadoop workflows

Anant Nag, LinkedIn
Shankar M, LinkedIn

#EUde12

A day in the life of data engineer

- Produce D.E.D Report by 5 AM.
- At 1 AM, an alert goes off saying the pipeline has failed
- Dev wakes up, curses his bad luck and starts backfill job at high priority
- Finds cluster busy, starts killing jobs to make way for D.E.D job
- Debugs failures, finds today is daylight savings day and data partitions has gone haywire.
- Most days it works on retry
- Some days, we are not so lucky

NO D.E.D => We are D.E.A.D

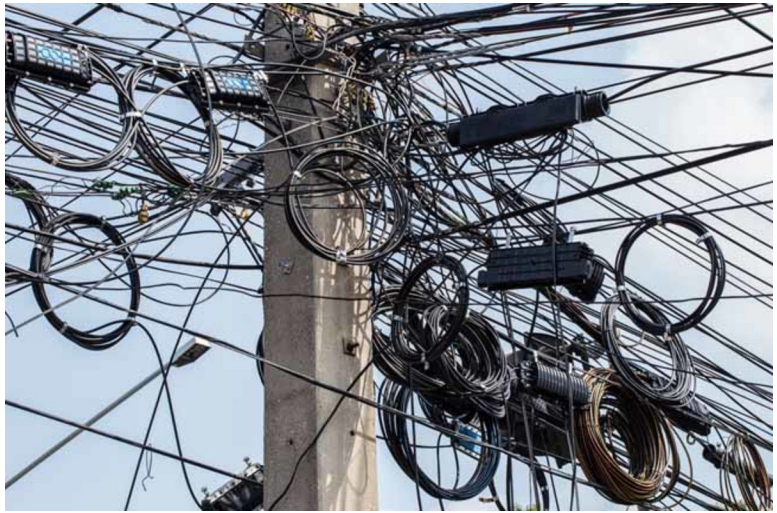
Scale @ LinkedIn

- 10s of clusters running different versions of software
- 1000s of machines in each cluster
- 1000s of users
- 100s of 1000s of Azkaban workflows running per month
- Powers key business impacting features
  - People you may know
  - Who viewed my profile

Nightmares at data street

- Cluster gets upgraded
- Data partition changes
- Code needs to be rewritten in a new technology
- Different version of a dependent jar is available
- ...

Do you know your dependencies?



- Direct dependencies
- Indirect dependencies
- Hidden dependencies
- Semantic dependencies

"Hey, I am changing column X in data P to format B. Do you foresee any issues?"

Paranoia is justified

No confidence to make changes

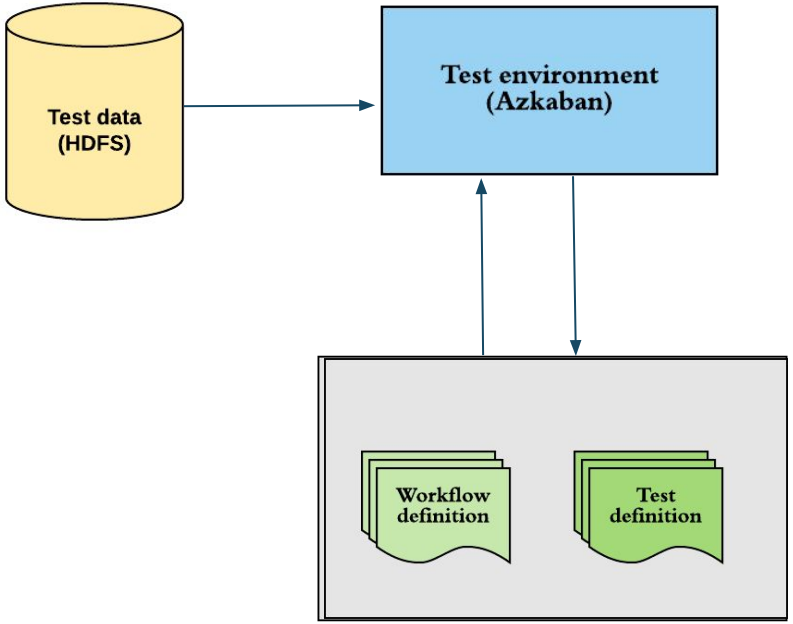
↓

Lack of agility

↓

Loss of innovation

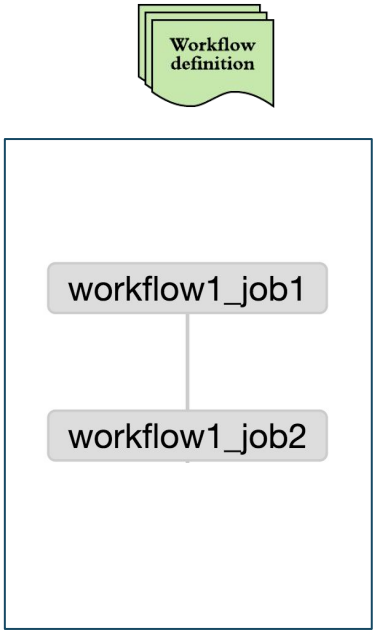
Architecture



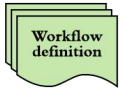
- Workflow definition
- Test definitions
- Test execution environment:
  - Local
  - Production
- Test data

Workflow definition

```
hadoop {
  workflow('workflow1') {

    sparkJob('job1') {
      uses 'com.linkedin.example.SparkJob'
      executes 'exampleSpark.jar'
      jars 'jar1.jar,jar2.jar'
      executorMemory '2G'
      numExecutors 400
    }
    sparkJob('job2') {
      uses 'com.linkedin.example.SparkJob2'
      executes 'exampleSpark.jar'
      depends 'job1'
    }
    targets 'job2'
  }
}
```



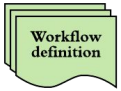
Test definition




```
hadoop {

  workflow('countByCountryFlow') {
   sparkJob('countByCountry') {
     uses 'com.linkedin.example.SparkJob'
     executes 'exampleSpark.jar'
     reads files: [
       'input_data':  "/data/input"
      ]
     writes files: [
       'output_path': "/jobs/output"
      ]
   }
    targets 'countByCountry'
  }
}
```


```
hadoop {

  workflowTestSuite("test1") {
    addWorkflow('countByCountryFlow') {
    }
  }

  workflowTestSuite("test2") {
    ...
  }
}
```

Overriding parameters




```
hadoop {

  workflow('countByCountryFlow') {
    sparkJob('countByCountry') {
      uses 'com.linkedin.example.SparkJob'
      executes 'exampleSpark.jar'
      reads files: [
        'input_data':  "/data/input"
      ]
      writes files: [
        'output_path': "/jobs/output"
      ]
    }
    targets 'countByCountry'
  }
}
```


```
hadoop {

  workflowTestSuite("test1") {
    addWorkflow('countByCountryFlow') {

      lookup('countByCountry') {

        reads files: [
          'input_data': '/path/to/test/data'
        ]
        writes files: [
          'output_path': '/path/to/test/output'
        ]
      }
    }
  }
}
```

Configuration override

- 10s of clusters
  - Multiple versions of Spark
  - Some clusters update now, some later
  - Code should run on all the versions

Configuration override

- Write multiple tests
  - One test for each version of Spark
- Override Spark version in the tests

```
workflowTestSuite("testWithSpark1_6_3") {
  addWorkflow('countByCountryFlow') {
    lookup('countByCountry') {
      set properties: [
        'spark.version': '1.6.3'
      ]
    }
  }
}
```

```
workflowTestSuite("testWithSpark2_1_0") {
  addWorkflow('countByCountryFlow') {
    lookup('countByCountry') {
      set properties: [
        'spark.version': '2.1.0'
      ]
    }
  }
}
```

```
assert(x == y)
```

True story

- Complex pipeline with 10s of Jobs
- Most of them are Spark Jobs

- DataFrames API
- Rewrite all spark jobs to use DataFrames
- Is my new code ready for production??
- Write tests
    - Assertions on output
- All tests succeed after changes (^_^)

Data validation and assertion

- Types of checks
  - Record level checks
  - Aggregate level
    - Data transformation
    - Data aggregation
    - Data distribution
- Assert against Expectation

Record level validation

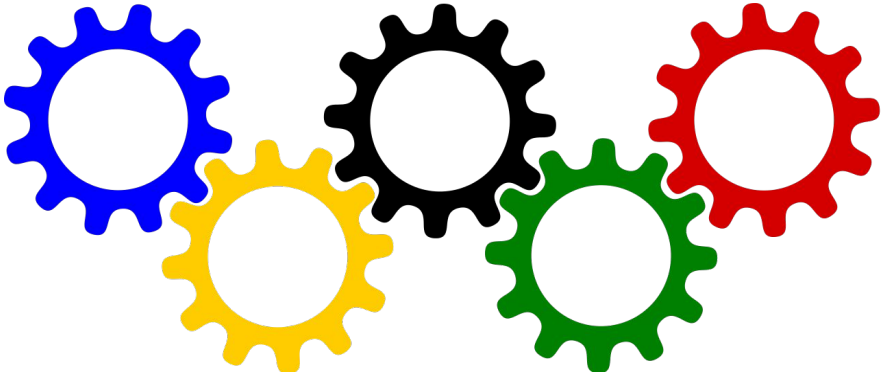
```
us 148083
in 46074
cn 34332
br 30836
gb 24387
fr 14983
...
```

```
hadoop {
  workflowTestSuite(‘test1’) {
    addWorkflow(‘countFlow’,’testCountFlow’){}
    assertionWorkflow('assertNonNegativeCount')
{
      sparkJob("assertNonNegativeCount") {
      }
      targets 'assertNonNegativeCount'
    }
}
```

```
val count = spark.read.avro(input)

require(count.
     map(r => r.getAs[Long](“count”)).
         where(_ < 0 )).count() == 0)
```

Aggregated validation

- Binary spam classifier C1
- C1 classifies 30% of test input as spam
- Wrote a new classifier C2
- C2 can deviate at most 10%
- Write aggregated validations



Execution

Test execution

$> gradle azkabanTest -Ptestname=test1

- Test results on the terminal
- Reports for the passed and failed tests

```
Tests [1/1]:=> Flow TEST-countByCountry completed with status SUCCEEDED

Summary of the completed tests

Job Statistics for individual test
-----------------------------------------------------------------------------------------------------------------------------------------------------------
Flow Name                | Latest Exec ID | Status        | Running  | Succeeded | Failed   | Ready    | Cancelled | Disabled  | Total
-----------------------------------------------------------------------------------------------------------------------------------------------------------
TEST-countByCountry      | 2997645        | SUCCEEDED | 0        | 2         | 0        | 0        | 0         | 0         | 2
```

Test automation

- Auto deployment process for Azkaban artifacts
- Tests run as part of the deployment
- Tests fail => Artifact deployment fails
- No un-tested code can go to production

Test Data

Test data

- Real data
  - Very large data
  - Tests run very slow
- Randomly generated data
  - Not equivalent to real data
  - Real issues can never be catched
- Manually created data
  - Covers all the cases
  - Too much effort
- Samples

Requirements of sample data

- Representative of real data
- Smaller in size
- Automated generation
- Sharable
- Discoverable

Road ahead

Flexible execution environment

- Sandboxed
- Replicate production settings
- Save and Restore
- Ability to run in a single box

Data validation Framework

- Validation logic in schema
- Automated validation on read and write
- Serves as contract between producers and consumers

Azkaban DSL: https://github.com/linkedin/linkedin-gradle-plugin-for-apache-hadoop

Azkaban: https://github.com/azkaban/azkaban