SPARK THE HARD WAY:

Lessons from Building an On-Premise Analytics Pipeline

Damian Miraglia

Joseph de Castelnau

Nielsen MROI Solutions

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

Nielsen MROI Solutions

A Brief History

1991	2011	2015
• Marketing Analytics Founded by Ross Link	• Marketing Analytics Acquired by Nielsen	• Digital Media Consortium II • Multi-Touch Attribution

Digital Media Consortium II

Industry Collaboration

Objectives:

- Test and improve industry practices for the measurement of digital media
- Understand the best way to use newly available, granular data
- Measure and optimize return on the billions of dollars invested in marketing

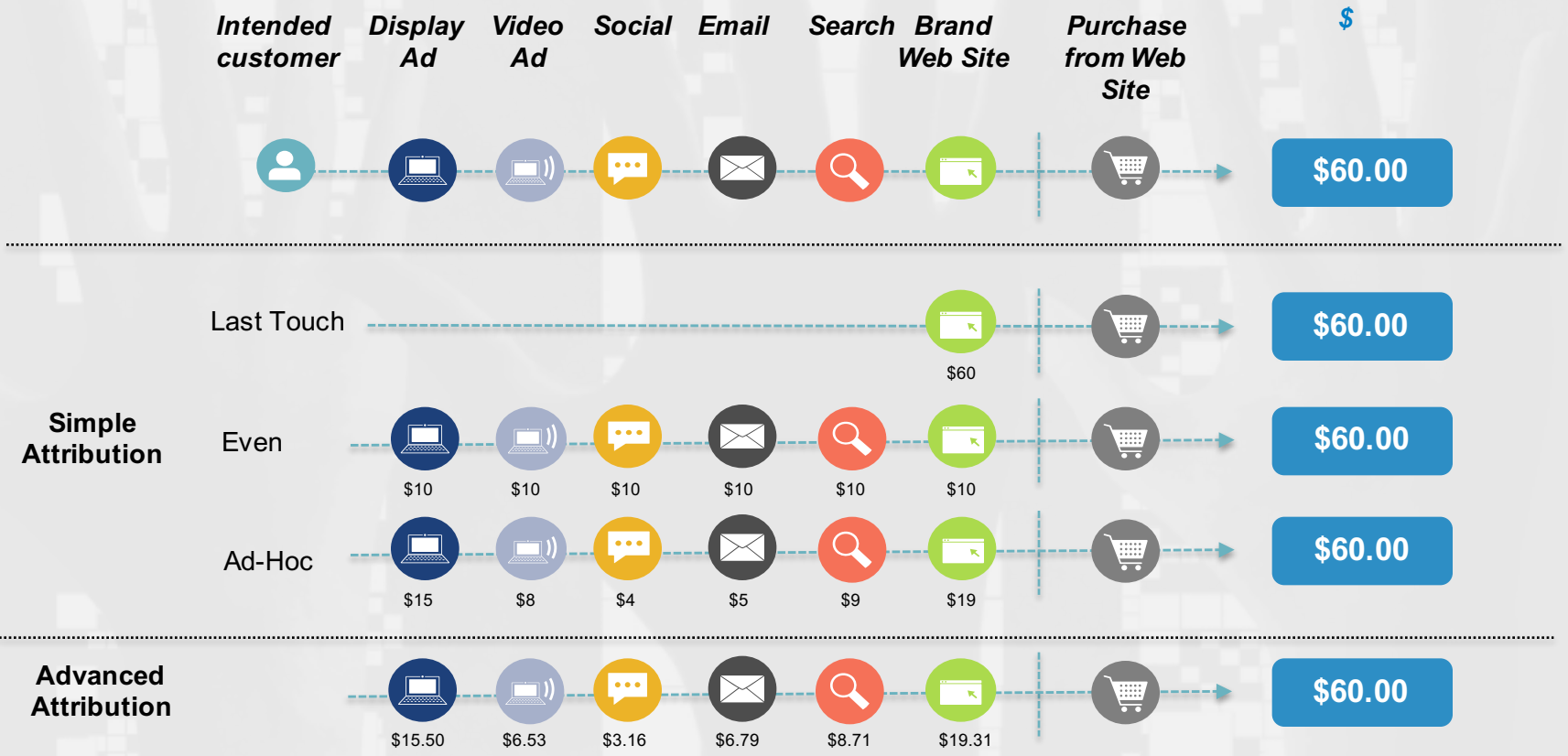
4 thousand
Advertising Campaigns

$30 billion
Worth of Sales Analyzed

4 billion
Digital Impressions Measured

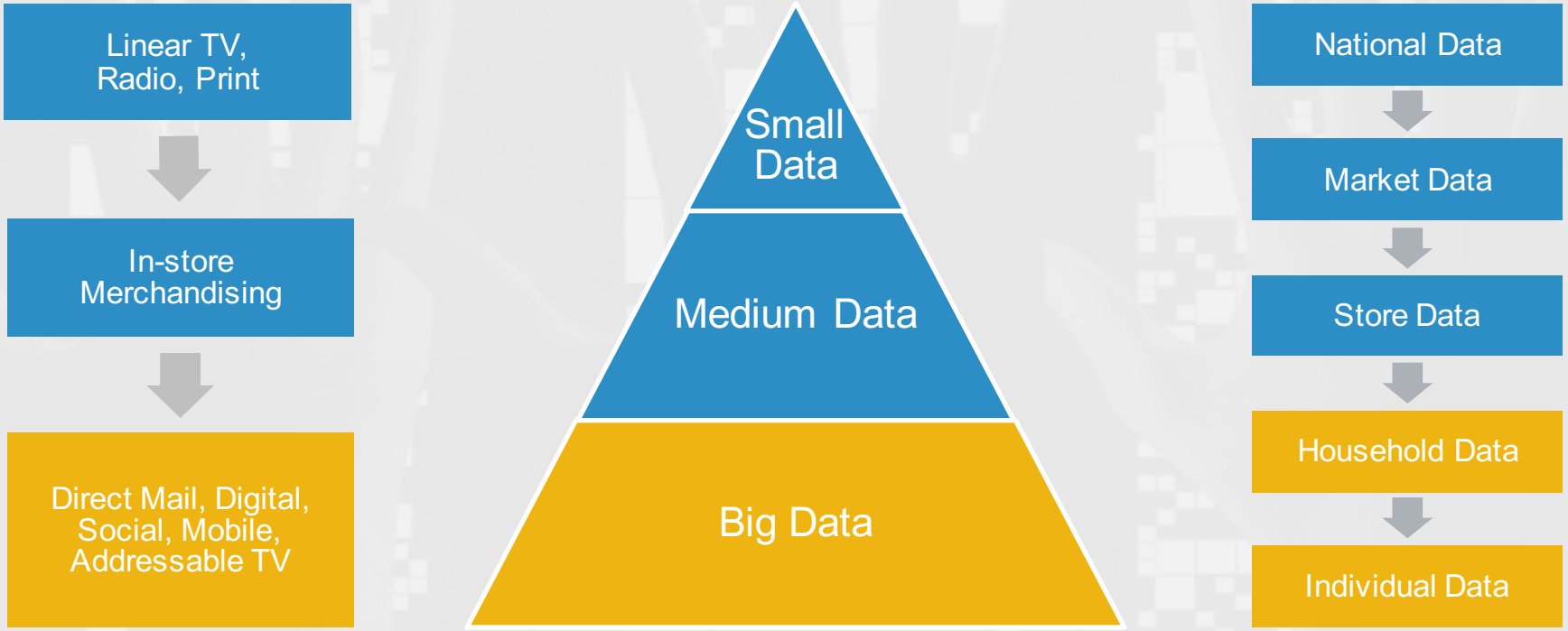
Multi-Touch Attribution

Attributing Sales Across Media Tactics



MTA Data Volume

Big Data Requires New Tools



Algorithm Development Process

Path from Prototype to Product

Prototype

Prototype in SAS/Netezza and test against real data

Scale

Adapt the model for Spark and optimize performance

Report

Build visualizations and navigation to explore insights from the model

Linear Path Does Not Work

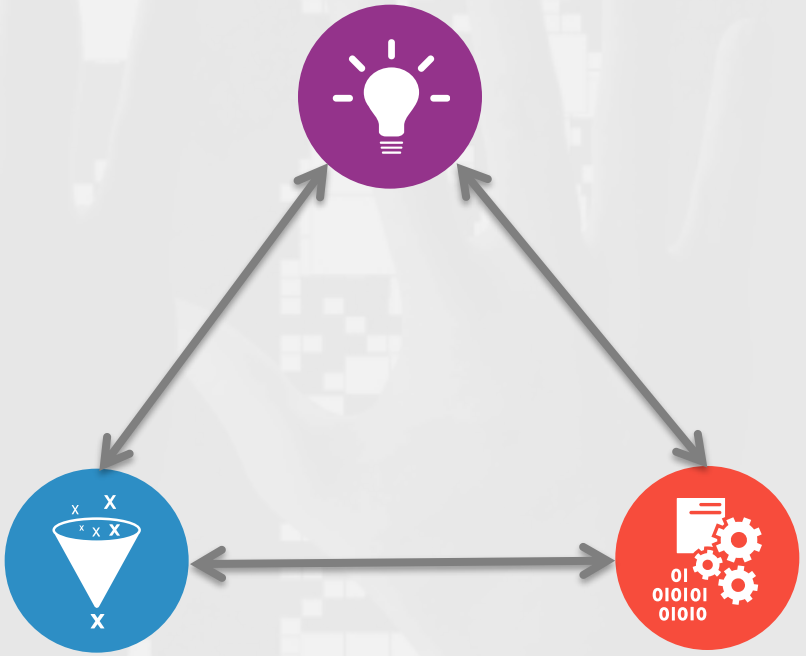
What We Learned the Hard Way

"Lift and Shift" ignores platform differences

Engineers and Statisticians should pair program

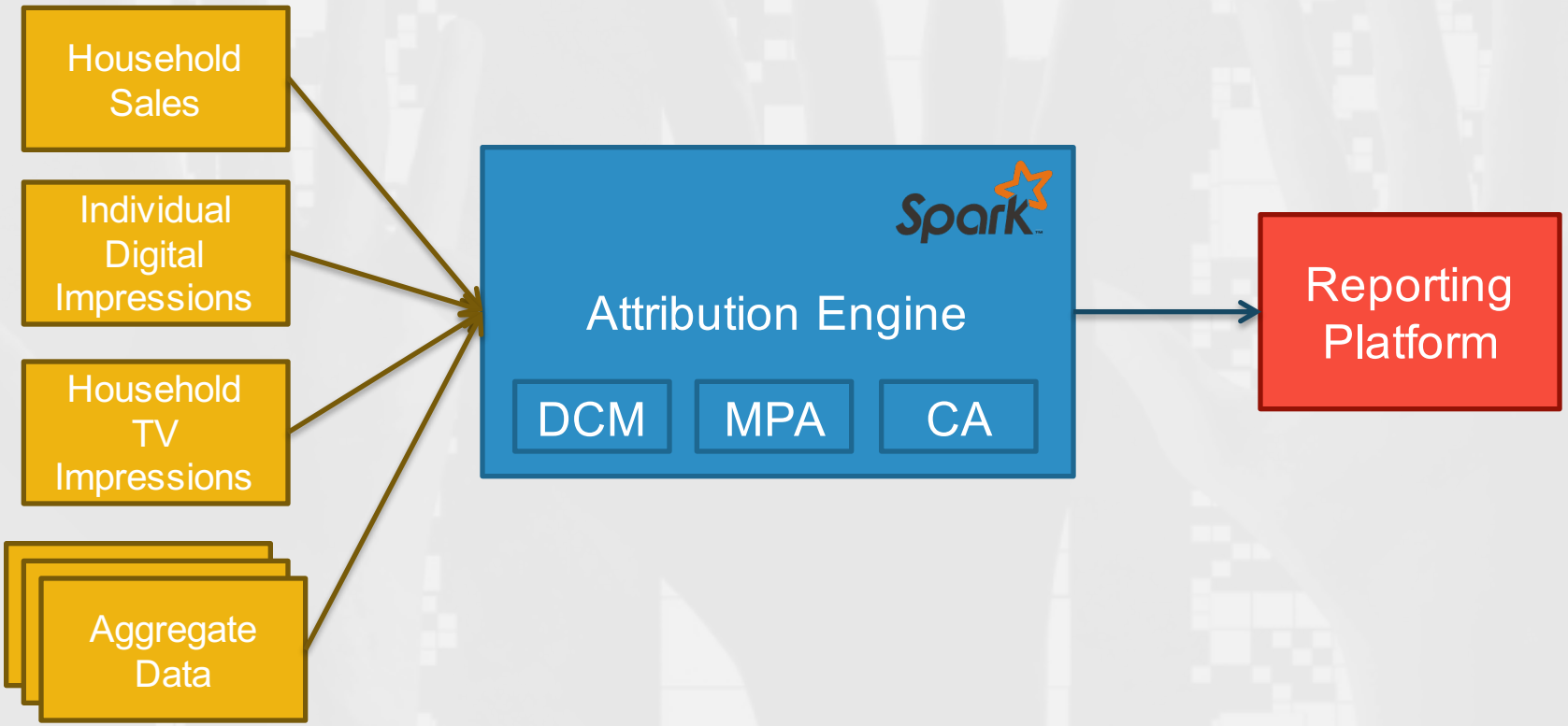
Collaboration across functions is key

Models must be built on Spark from the start



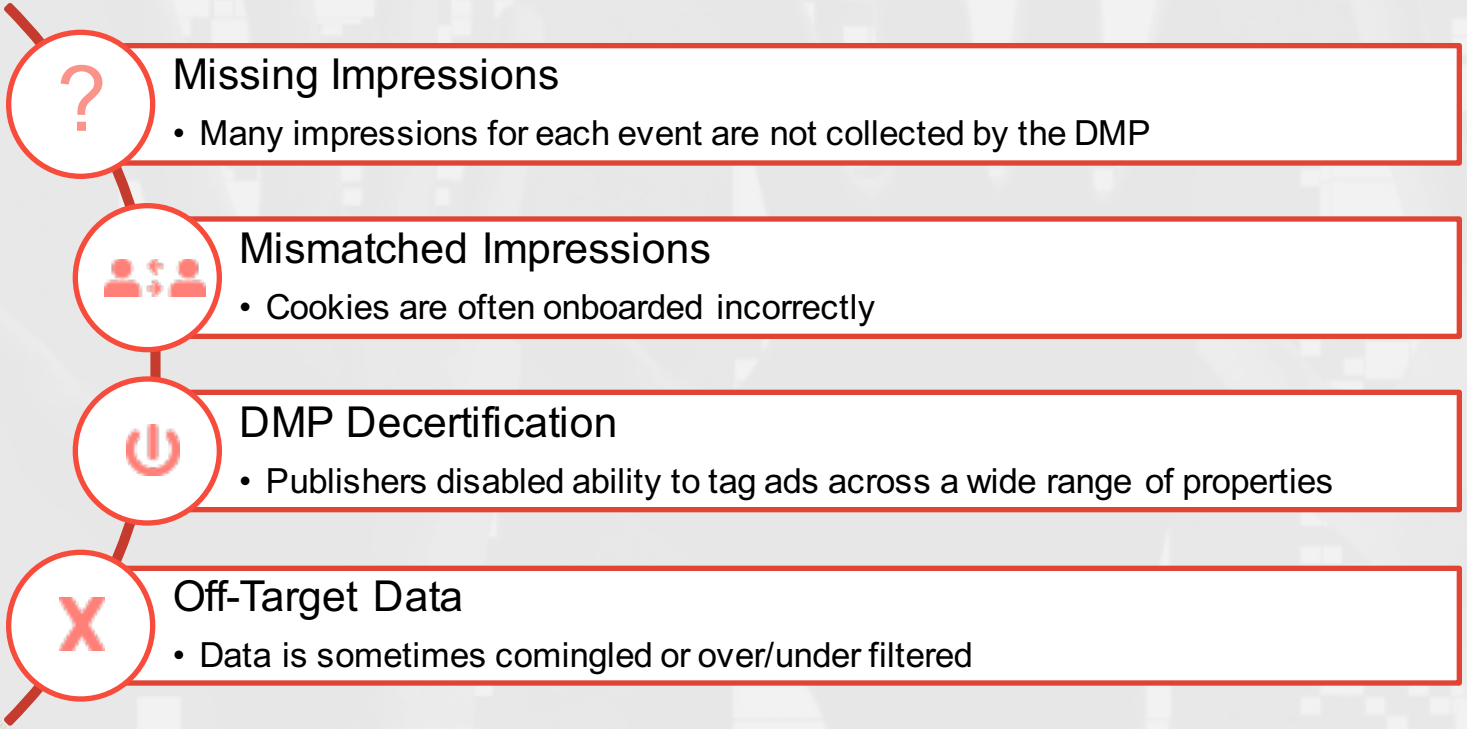
MTA Data Flow is Straightforward

A Series of Tubes



MTA Data Issues

Digital Data is Messy



Data Sensitivity and Restrictions

In agile fashion: MVP on-prem, V1.x on cloud

While we are resolving restrictions & limitations, we started with on-prem solution:

- Retailers reluctant to allow sales data on sensitive cloud system
- Privacy concerns require Nielsen to operate with all governance set to green.

</INTRO>

.

So we're stuck on premise.
Now what?

Spark on the Cloud would have been Great

- ✓ On Demand
- ✓ Elastic Scaling
- ✓ Infrastructure as a Service
- ✓ Data Science Toolchain Built-In
- ✓ Managed Services
- ✓ Isolated Environments
- ✓ Vendor Support
- ✓ Job Scheduling

Our Cluster

There are Many Like it, but this One is Ours

Red Hat Enterprise Linux

- Version 7.2
- Compliance through SELinux

Data/Compute Nodes

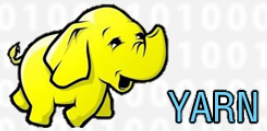
- Hortonworks Data Platform
  - Provides Zookeeper, HDFS, Yarn, Hive, Spark*
  - Cluster Management through Ambari

Edge Node

- Adjacent to, but not part of, the cluster
- Used to run drivers, submit jobs

Chef

- Ensures Consistency across Nodes
- Eases DevOps Burden



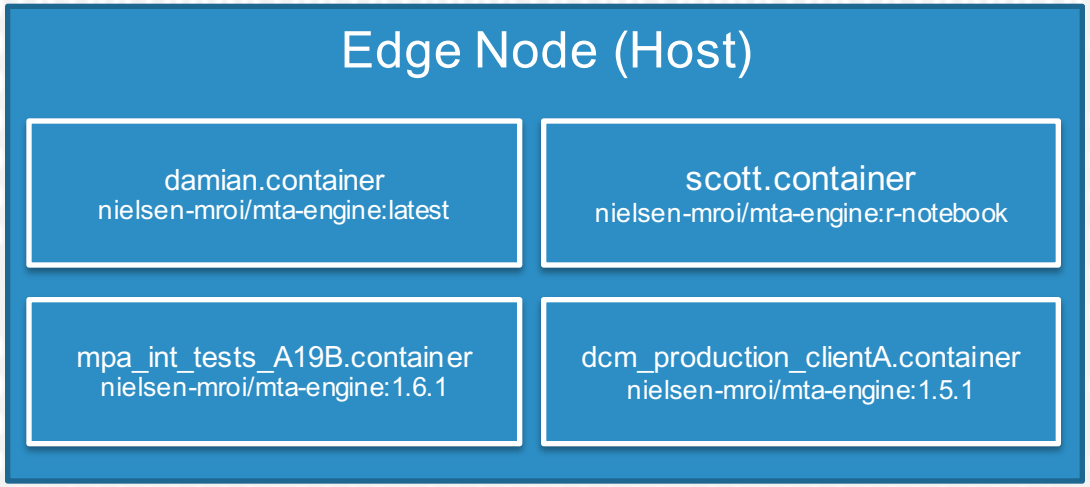
*The version of Spark included in HDP typically lags by several months

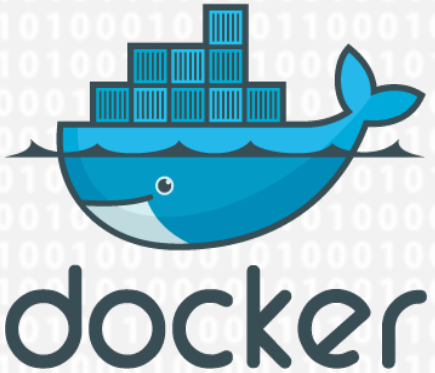
Problem: Environment Sharing

There Can Be Only One Cluster

- R&D, QA, Staging, and Production all live on the same cluster
- Different models have different dependencies and might even run on different versions of Spark
- Changing Spark configuration settings may adversely impact other developers
- Only one branch of code can be deployed at a time

Docker Creates Virtual Edge Nodes





Repository	Tag	Image ID	Created	Size
nielsen-mroi/mta-engine	latest	48218b16fe5b	4 weeks ago	3.168 GB
nielsen-mroi/mta-engine	1.6.1	48218b16fe5b	4 weeks ago	3.168 GB
nielsen-mroi/mta-engine	r-notebook	44776f55294a	4 weeks ago	4.03 GB
nielsen-mroi/mta-engine	scala-notebook	17be1ee7089f	4 weeks ago	3.247 GB
nielsen-mroi/mta-engine	1.6.0	f249c7e8729a	6 weeks ago	3.012 GB
nielsen-mroi/mta-engine	1.5.1	0ab5aafbd94b	12 weeks ago	2.988 GB
centos	latest	778a53015523	8 weeks ago	196.7 MB
centos	7	778a53015523	8 weeks ago	196.7 MB

Everything is Isolated, Consistency Assured

Models

- Spark Version via `spark.yarn.jar`
- Dependency Management
- Configuration Settings (Spark/Yarn/Hive)

Developers

- Code Branches
- Preconfigured Toolchain

Continuous Integration

- Executable Container (via bootstrap.sh)
- Run tests, Export results
- Triggered by Jenkins on Push

```
FROM centos:7
MAINTAINER nielsen-mroi

# update yum, isntall wget
RUN yum update -y \
…
# install java
RUN wget --no-cookies --no-check-certificate --header \
…
# install scala
RUN wget http://downloads.typesafe.com/scala/2.10.6/scala-2.10.6.tgz \
…
# install hadoop
RUN wget http://apache.claz.org/hadoop/common/hadoop-2.7.1/hadoop-
2.7.1.tar.gz \
…
# set environment variables needed for spark install
ENV JAVA_HOME=/opt/java \
…
# install spark
RUN wget http://apache.claz.org/spark/spark-1.6.1/spark-1.6.1.tgz \
…
# install os packages
RUN yum -y install \
…
# install python libraries
RUN pip install --upgrade pip \
…
# copy resources, and move them to their destinations
COPY files /
RUN mkdir /opt/spark_dependencies \
…
# set environment variables
COPY env_vars.sh /etc/profile.d/
ENV SPARK_HOME=/opt/spark \
…
# copy bootstrap script and set container to execute it
COPY bootstrap.sh /
RUN chown root /bootstrap.sh && chmod 700 /bootstrap.sh
ENTRYPOINT ["/bootstrap.sh"]
```

Problem: Orchestration

Step 1: Collect Data, Step 2: ???, Step 3: Profit!

Triggering long-running, multi-part jobs is risky without resumability.

Models consist of too many steps and have DAGs that are too big for Spark to handle all at once

Models developed by teams of engineers need convenient contracts between their component parts

Luigi Builds Composable Task Chains

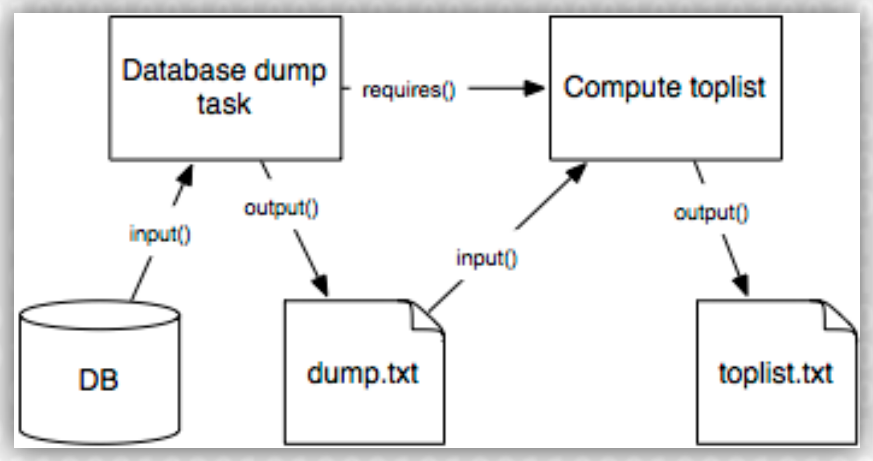
```python
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

The business logic of the task

Where it writes output

What other tasks it depends on

Parameters for this task



Source: https://luigi.readthedocs.io/

Luigi Provides Reliable Dependency Management

Dependency Resolution

- Dependency tree
- Tasks run in parallel

Resumability

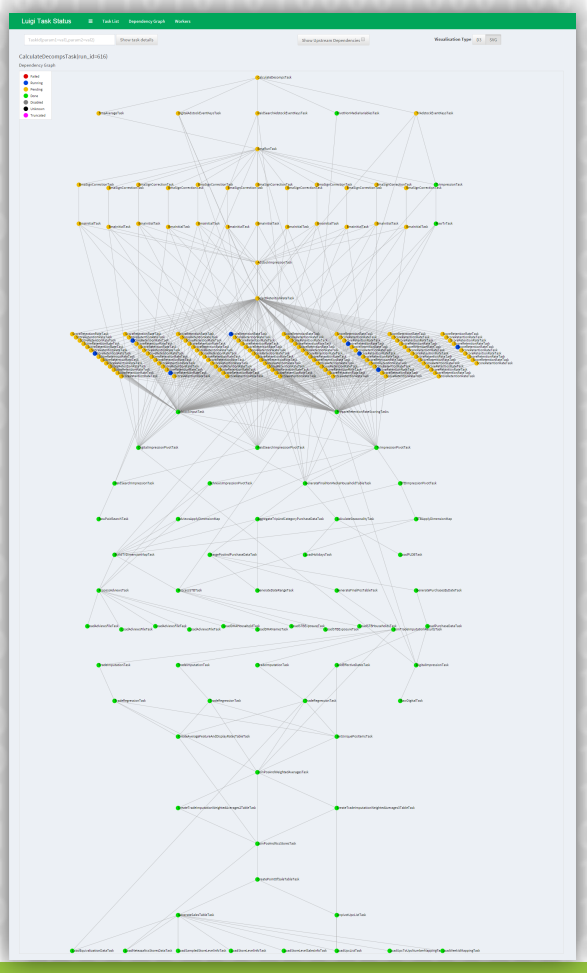
- Target discovery
- Won't rerun completed tasks

Hackable

- Python code
- Small codebase

HiveTableTarget(luigi.Target)

- Encourages checkpointing
- Solid contract between tasks
- Developer Parallelization



Problem: Performance Concerns

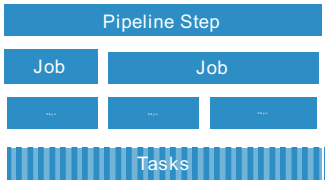
This Cluster ain't Big Enough for the Two of Us

There's only so much RAM, and only so many CPUs in the cluster.

Some jobs are resource hogs and require 50% or more of the cluster resources.

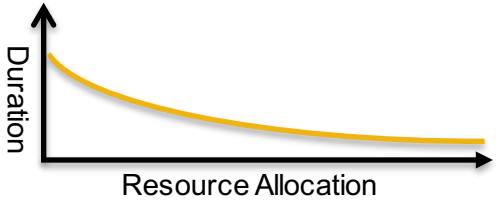
Some resources must be reserved for R&D purposes (see problem 0).

Meta-Analysis Ensures Optimal Resource Utilization



- Spark breaks a given Step into smaller pieces (**Step -> Jobs -> Stages -> Tasks**)
- Spark efficiently parallelizes these smaller pieces

- Spark's built-in parallelism improves performance as more resources are used
- Performance does not scale sub-linearly forever – **returns diminish**



Solution: Experiment with Different Resource Allocations for each Task and Optimize for Throughput

- Steps may take longer than if given all of the resources
- Steps that do not depend on each other can be run in parallel
- **Pipeline runs in less time overall**

Configuration Alleviates Resource Contention

YARN Dynamic Resource Pools

- Scheduled allocation of resources
- Unused resources can be reallocated

Spark Dynamic Allocation

- Define min & max number of executors
- Unused executors are removed and resources returned to pool

Miscellaneous Spark Configs

- spark.speculation
- spark.shuffle.io.numConnectionsPerPeer
- spark.akka.threads

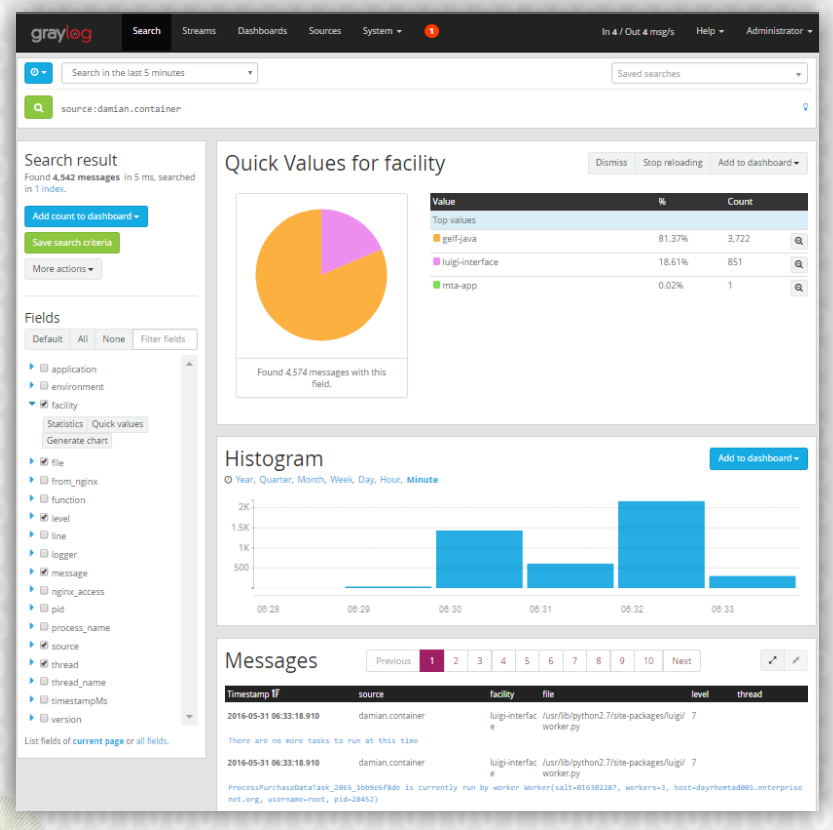
Problem: Debugging

I Hope You Like Reading Logs

Debugging Spark problems often requires digging through YARN logs.

Particularly heinous failures will sometimes prevent YARN's log aggregation from collecting everything.

Log Management Makes it Easier to Gain Insight



graylog

```
# Set everything to be logged to the console
log4j.rootCategory=ERROR, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n

# Settings to quiet third party logs that are too verbose
log4j.logger.org.eclipse.jetty=WARN
log4j.logger.org.eclipse.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO

# Send all INFO logs to graylog2
log4j.rootLogger=INFO, graylog2

# Define the graylog2 destination
log4j.appender.graylog2=org.graylog2.log.GelfAppender
log4j.appender.graylog2.graylogHost=dayrhemtad005
log4j.appender.graylog2.facility=gelf-java
log4j.appender.graylog2.layout=org.apache.log4j.PatternLayout
log4j.appender.graylog2.extractStacktrace=true
log4j.appender.graylog2.addExtendedInformation=true
log4j.appender.graylog2.originHost=damian.container
log4j.appender.graylog2.additionalFields={'environment': 'DEV', 'application': 'Spark'}
log4j.appender.graylog2.originHost=damian.container
log4j.appender.graylog2.additionalFields={'environment': 'DEV', 'application': 'Spark'}
```

Graylog is Easy to Wire Up with Spark

Log4J Appender

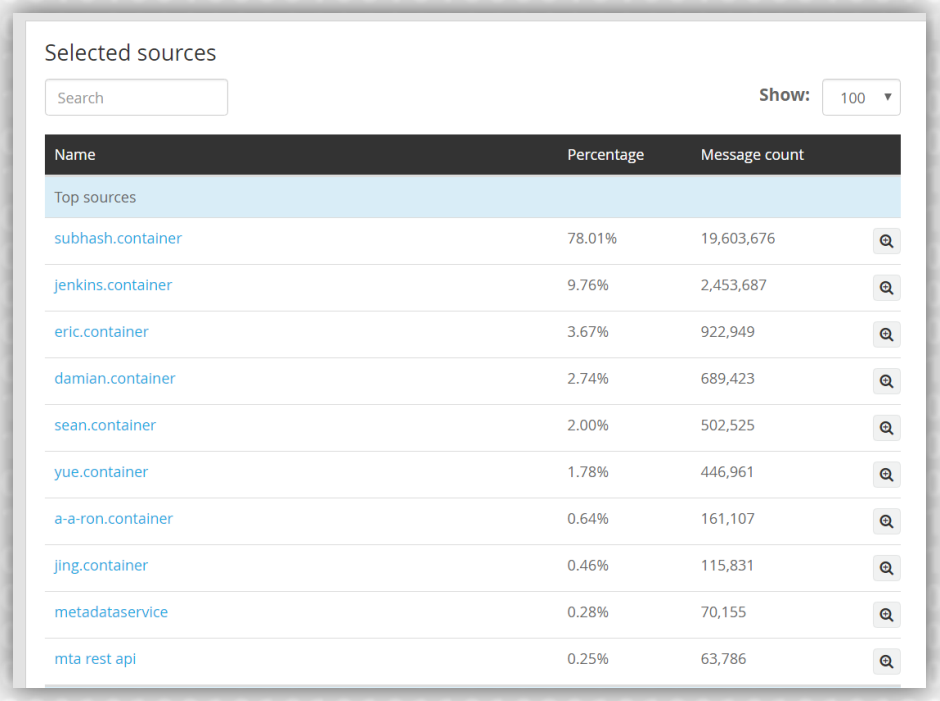
- Easy setup to forward Spark logs
- Works even if YARN log aggregation fails

Containerized Installation

- Can be run from Docker containers
- Low setup cost

Powerful

- Search across many fields
- Dashboards for admins
- Alerts for failures

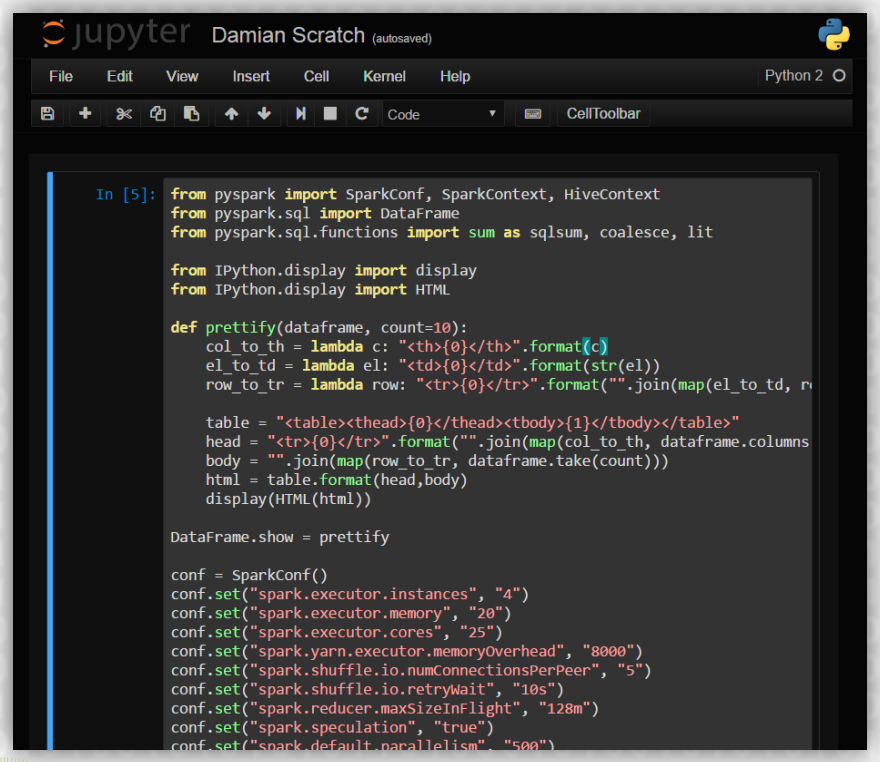


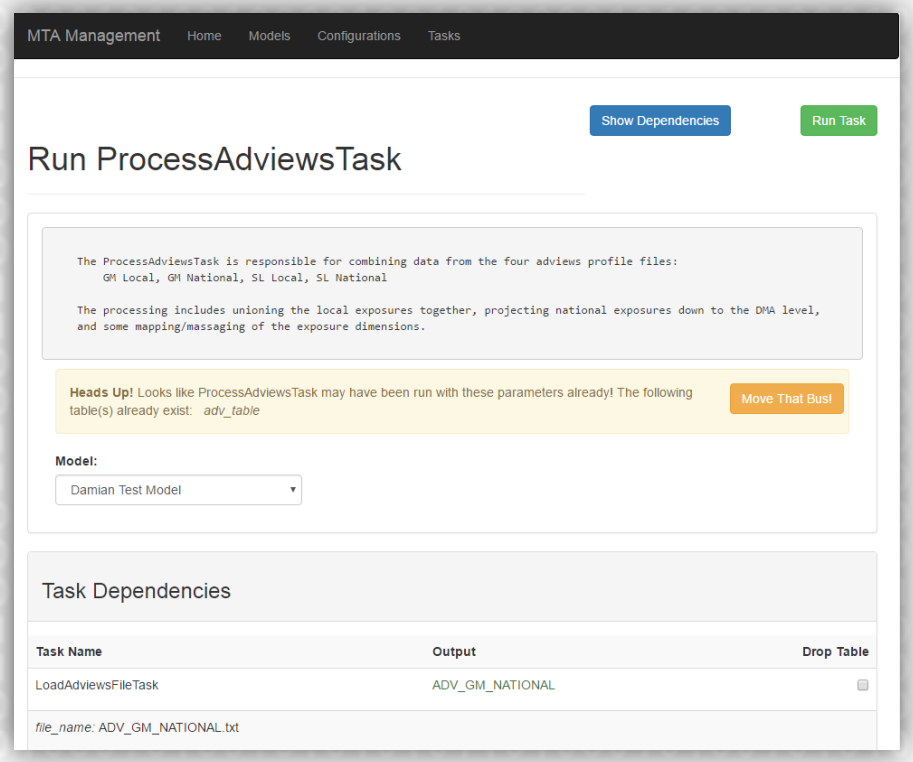
Problem: Development Tooling

Developing with Spark-Submit alone is inefficient

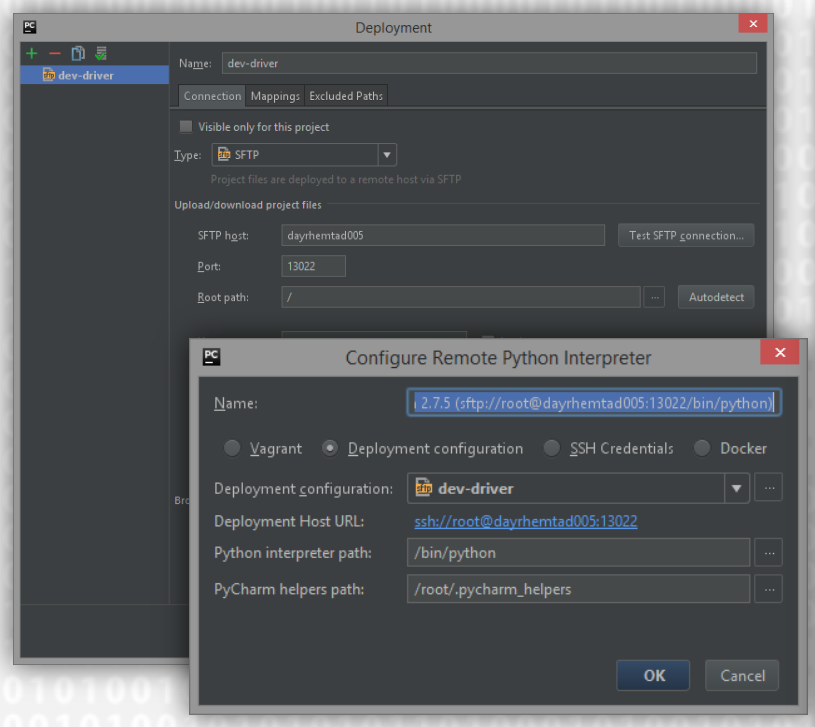
Not all users have the requisite skills to handle the entire development toolchain

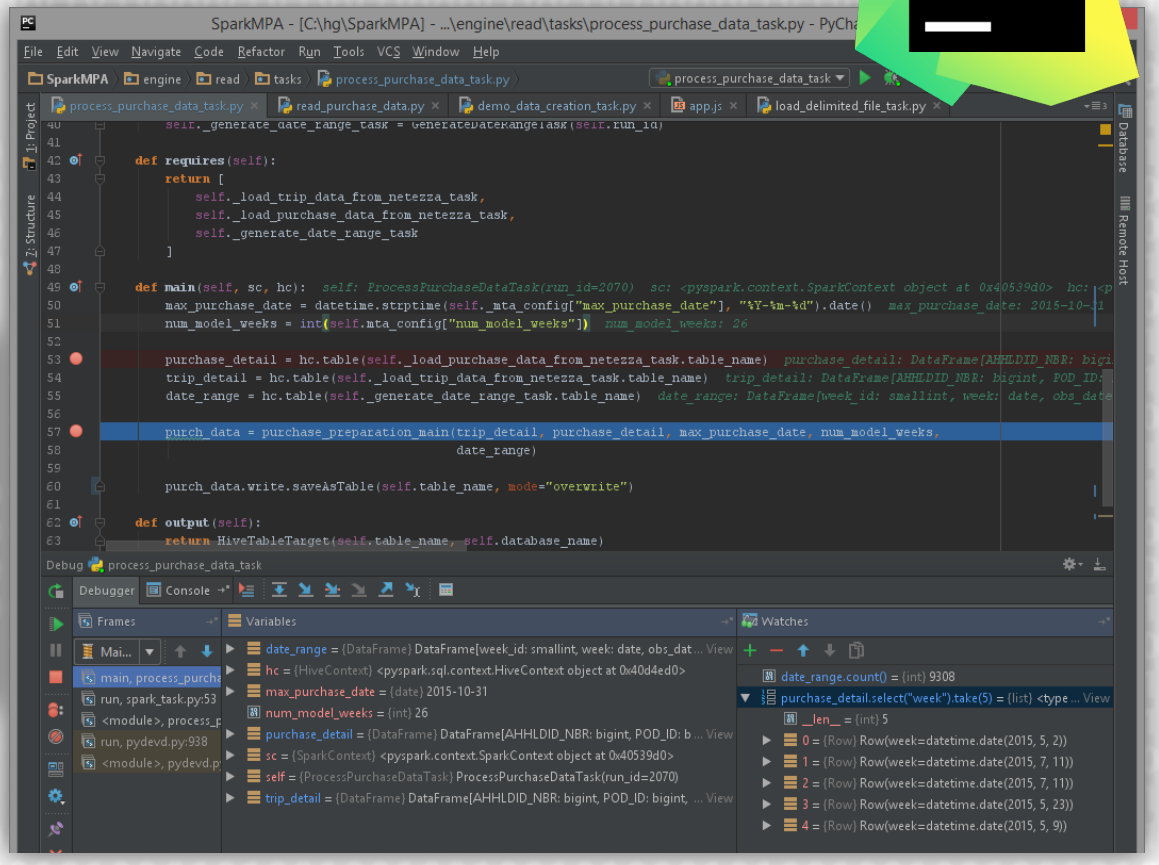
Roll Your Own Tooling



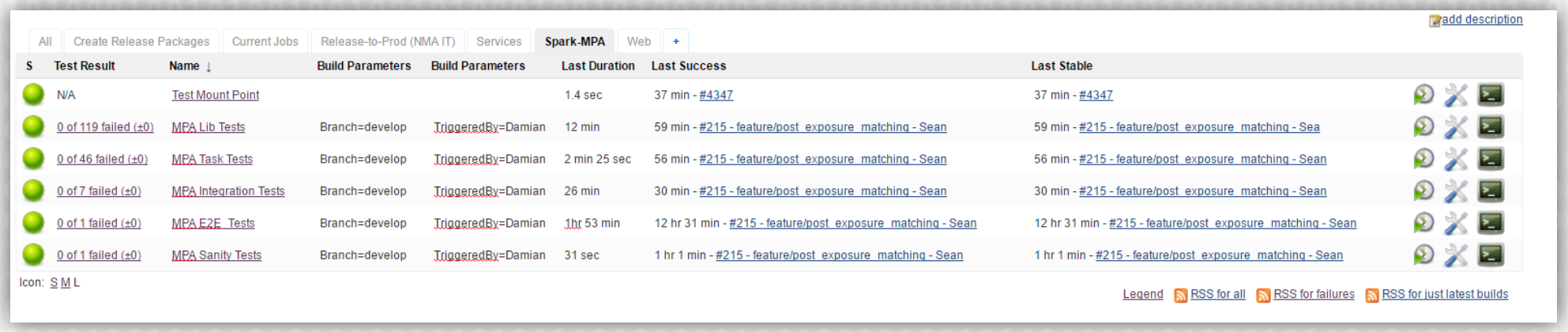


Get Creative with Your Tools





CI for Spark Development



add description

All | Create Release Packages | Current Jobs | Release-to-Prod (NMA IT) | Services | Spark-MPA | Web | +

S	Test Result	Name ↓	Build Parameters	Build Parameters	Last Duration	Last Success	Last Stable
	N/A	Test Mount Point			1.4 sec	37 min - #4347	37 min - #4347
	0 of 119 failed (±0)	MPA Lib Tests	Branch=develop	TriggeredBy=Damian	12 min	59 min - #215 - feature/post_exposure_matching - Sean	59 min - #215 - feature/post_exposure_matching - Sea
	0 of 46 failed (±0)	MPA Task Tests	Branch=develop	TriggeredBy=Damian	2 min 25 sec	56 min - #215 - feature/post_exposure_matching - Sean	56 min - #215 - feature/post_exposure_matching - Sean
	0 of 7 failed (±0)	MPA Integration Tests	Branch=develop	TriggeredBy=Damian	26 min	30 min - #215 - feature/post_exposure_matching - Sean	30 min - #215 - feature/post_exposure_matching - Sean
	0 of 1 failed (±0)	MPA E2E Tests	Branch=develop	TriggeredBy=Damian	1hr 53 min	12 hr 31 min - #215 - feature/post_exposure_matching - Sean	12 hr 31 min - #215 - feature/post_exposure_matching - Sean
	0 of 1 failed (±0)	MPA Sanity Tests	Branch=develop	TriggeredBy=Damian	31 sec	1 hr 1 min - #215 - feature/post_exposure_matching - Sean	1 hr 1 min - #215 - feature/post_exposure_matching - Sean

Icon: S M L

Legend | RSS for all | RSS for failures | RSS for just latest builds

Jenkins

docker

ORGANIZATIONAL SOLUTIONS

Risk Mitigation for the Cloud

Multi-Cloud

- Keep sensitive data with preferred cloud vendor
- Prevents “all-in” bets
- Pushes workloads toward most innovative technologies
- <> Cloud agnostic

Encryption

- Data classification first step.
- Encrypting data at rest and in transit is solvable .
- Can be opportunity to couple to an advanced hash attribute matching algo.
- Vendor space can shorten your cycle time based on your IP.

Next Steps for our Platform

- Cloud (!)
- Much easier end to end analytics development model
- Significantly simpler data wrangling
- Enable the right kind of “Citizen Data Scientist” insights.
- Syndication/Automation when warranted

THANK YOU.

We are hiring talent.

Email: damian.miraglia@nielsen.com