Data Ingestion Engine

Design Overview for Big Data Meetup

Agenda

- Introduction
- Data Ingestion Engine
  - Why do this? Isn’t data ingestion just ETL?
  - Overview of Engine design
  - Advantages of this approach
- Future enhancements
- Q/A & Discussion (remainder)

Who is this guy?

- Steven Frobish, Senior Consultant at Daugherty
- 17 years in Data and Analytics
  - 3 years doing big data on mainframe before it was called “Big Data”
  - 4 years Informatica support
  - 4 years Data Integration Architect
  - 6 years in the Hadoop space
    - Ingestion, Cataloging, and Provisioning
    - Operational Analytics
    - Machine Learning

Motivation

- Analysts – Just give me the raw data!
- Auditors – Where is your sensitive information? (including free text)
- Records Retention – How long is your data going to stick around?
- Project Managers – I need 1800 tables moved in the next 2 weeks!
- DBA’s – Your schema is going to change (and you won’t know when)

Motivation - contd

- What did I learn:
  - As “Big Iron” analyst
    - Disk I/O is painfully slow, but ’parallelizable’
    - Joins are expensive
  - As Informatica Support Engineer:
    - Roughly 30% of mappings were ‘pass-thru’
    - Another 50% were similarly “doing the same thing over and over”
      - …but on slightly different data
  - As Data Integration Architect:
    - Most consumers can make sense of data on their own
    - They just want everything, all the time
    - Even highly modeled stores have common features that can be generalized

Motivation - contd

- What did I learn:
    - As Data Lake Builder
        - Lines of Business almost always need change data if available
        - They are happy with schema on read
            - but don’t break their stuff
        - They will inevitably have their favorite format
            - which they will discard immediately in favor of consistency
        - Audits happen
        - Security will always tell you that you are doing it wrong
            - unless you can plug in any level of encryption they can think of…
                - (then literally anything other than plaintext will do)

Why not ETL?

- Connecting to data is the easy part
  - JDBC
  - SFTP
  - REST API's
  - Loading to Hadoop from edge node
- ETL tool's main product is 'T' (no 'T' = no ETL)
- 1 table = 1 mapping
- Schema definition is part and parcel to the mapping
- Resource based Scheduling is difficult (impossible?)

Why not Spark?

- Spark is a parallel processing framework
  - It is not free performance or instant throughput
  - “Accessing the power of the cluster” is not needed here
- Problem of ingesting data is inherently not parallel
  - Read a single source of data and write to a single target
  - Any parallelism added in the middle is only adding complexity, not improving throughput

What about NiFi?

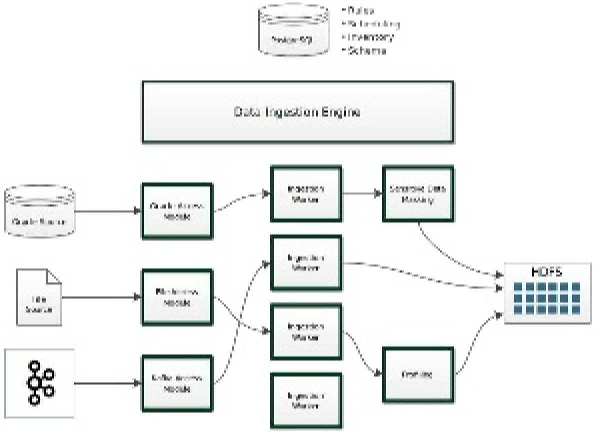
- NiFi is cool
  - Has many features Engine does not explore
    - Back-pressure, fine-grained access, write-ahead logs
  - An event driven architecture
    - Typically requires more “plumbing” to set up
- Flows are still 1:1 to sources and actions are not metadata driven
- Costly to setup many flows at once

What about SQOOP?

- Actually SQOOP is in the current Engine as an Access Module
- It has many features we want:
  - Incremental extracts
  - Captures schema information
- And several shortcomings:
  - No resource based scheduling
  - Doesn’t track inventory
  - Field level masking is very difficult

The Data Ingestion Engine

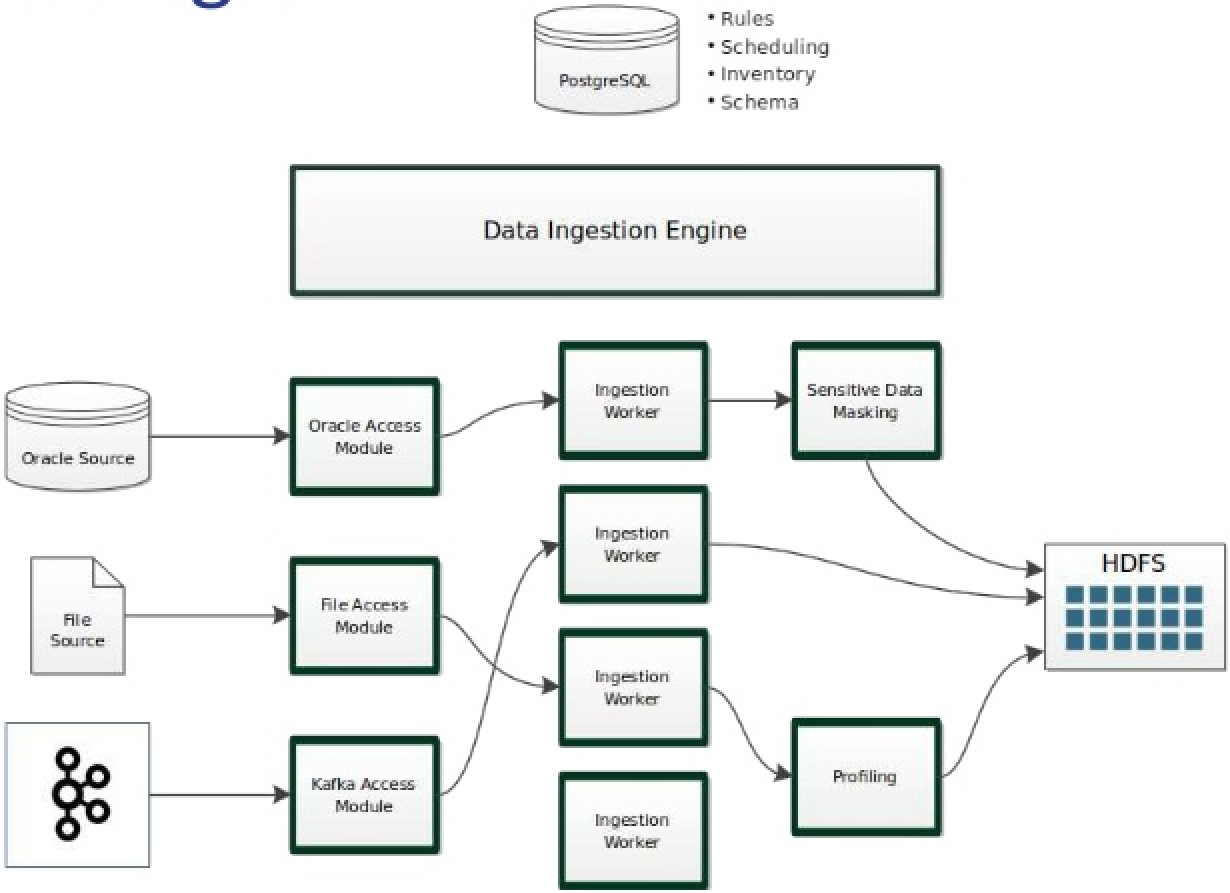
- Is really just a driver program that:
    - Queries a repository
    - Decides to run jobs
        - Time based
        - ...but conscious of existing work
    - Launches worker threads



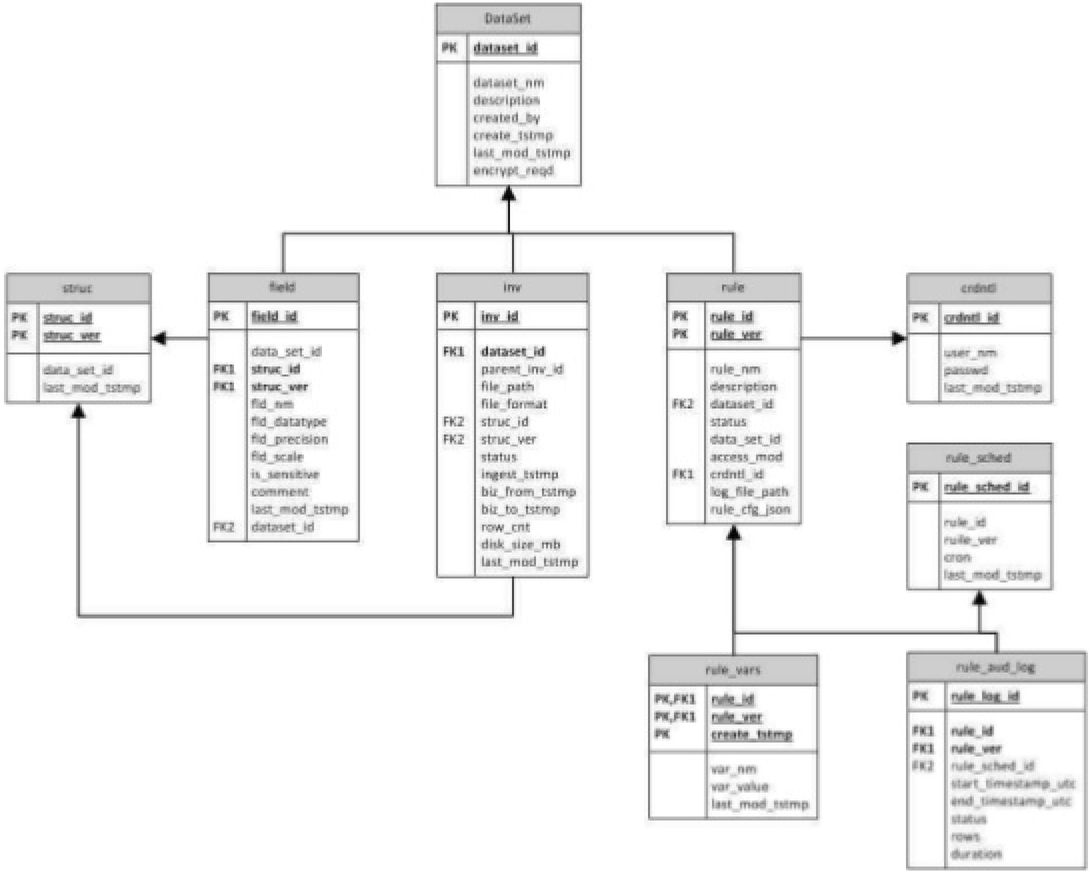
Workers

- Find work to do from the queue
- Load up appropriate access module
- Track:
  - Logging
  - Runtime metadata
  - Inventory metadata
- Occasionally:
  - Transform sensitive data
  - Capture profile stats
- Report errors
- Find *more* work to do from queue

Engine Design



Metadata Repository



DataSet	
PK	dataset_id
	dataset_nm description created_by create_tstmp last_mod_tstmp encrypt_reqd

struc	
PK	struc_id
PK	struc_ver
	data_set_id last_mod_tstmp

field	
PK	field_id
	data_set_id
FK1	struc_id
FK1	struc_ver
	fld_nm fld_datatype fld_precision fld_scale is_sensitive comment last_mod_tstmp
FK2	dataset_id

inv	
PK	inv_id
FK1	dataset_id
	parent_inv_id file_path file_format
FK2	struc_id
FK2	struc_ver
	status ingest_tstmp biz_from_tstmp biz_to_tstmp row_cnt disk_size_mb last_mod_tstmp

rule	
PK	rule_id
PK	rule_ver
	rule_nm description
FK2	dataset_id
	status data_set_id access_mod
FK1	crdntl_id
	log_file_path rule_cfg_json

crdntl	
PK	crdntl_id
	user_nm passwd last_mod_tstmp

rule_sched	
PK	rule_sched_id
	rule_id ruile_ver cron last_mod_tstmp

rule_vars	
PK,FK1	rule_id
PK,FK1	rule_ver
PK	create_tstmp
	var_nm var_value last_mod_tstmp

rule_aud_log	
PK	rule_log_id
FK1	rule_id
FK1	rule_ver
FK2	rule_sched_id
	start_timestamp_utc end_timestamp_utc status rows duration

Access Modules

- The code necessary to access a given source
  - Oracle
  - SFTP
  - Kafka
- Each module is responsible for:
  - Pulling data
  - Acquiring schema information
  - Recording basic statistics - row/record count, byte size, time taken

Transform Modules

- Use schema information to:
  - Mask data
  - Transform data to a given format
  - Optionally capture profile stats
- Did someone mention sensitive data?
  - HIPAA, PCI, PII, etc.
  - Would some column tags help?
    - Explicitly tag column name (e.g. ‘colA’ tagged as “is_sensitive”)
    - Regex on column name (e.g. like ‘.*phone.*’, or ‘b(irth)?_?date’)
  - Have some regexes for free text
    - Regex on content (e.g. ‘\d{3}-?\d{2}-?\d{4}’)

Metadata Driven Approach Benefits

- Generate many rules quickly
- Imagine an Access Module for which the output is more rules metadata
  - Scan Oracle Catalog for a given schema
  - Generate rules/datasets for every table
  - Let the access module do the heavy lifting
  - You just need to tell it where to go and provide credentials

Metadata Driven Approach Benefits

- Capture schema changes immediately
- Use inventory data to trigger consumer processes
- Full lineage & Audit logs accessible in a single place
- Resource based scheduling and connection throttling
- Information Retention driven by metadata instead of scanning data
- … many more that become apparent only after you have built it

Some other stuff we are working on…

- Using KSQL to enrich data from Kafka
  - Engine will start (if necessary) KSQL server
  - With metadata generated KSQL***
- Provision data to an outside location***
- Launch Oozie workflow

*** Future feature

Questions?

Contact

steven.frobish@daugherty.com

Linked In