Hadoop Inspector

Ken Farmer
William Farmer

August 14, 2015

Abstract

1 Background

Data quality problems have plagued analytical systems for twenty years: continually appearing in the top four reasons for project failure. In this space data quality problems loom large - a small defect that could be safely ignored or forgotten in the transactional world hamper queries and cause users to question our credibility for months.

The advent and innovation in Big Data and Data Science has not diminished this challenge. On Hadoop specifically:

- Our data generally lacks any enforced constraints to ensure data validity
- We are adding data faster than ever, with less time to research upstream and ETL pipeline issues
- We are building vast systems, sometimes with hundreds of thousands of tasks being defined
- We often have democratized access to our clusters - with dozens of different people adding data.

Additionally, in these large clusters most teams struggle to comply with policies and other requirements, whether regulatory, corporate or defined by their own teams. These might define general data retention requirements, or specific requirements for individual tables. They might define table naming conventions, security requirements, or stats aging & collection requirements.

2 Objective

Hadoop-Inspector is being built because we believe that the complexity of a large, constantly loaded cluster defies an unmanaged approach or QA testing in the development process. It requires something more like an automobile assembly line: continuous quality control (QC) that can take into account undocumented changes from upstream sources, accidental changes to production, changes that bypass QA, etc. And it shouldn't be limited to traditional quality tests, but should be able to test for compliance with policies as well.

3 Technical Details

Users determine the schedule on which they wish to test their cluster, and every allotted time increment, every check is run. These results are kept in a local database which is then referenced when users examine the frontend.

Again, given the intricacies of a hadoop cluster, most checks should be written by the cluster maintainers and operators. Every environment and system is different, and every cluster will have different requirements. We've strived to provide some basic checks, as well as suggestions for checks that may apply to cluster-specific requirements, however for the most part, these checks are the user's responsibility.

3.1 Checks

Every check is simply a script that lives in a type-specific folder. Rules and Warnings exist separately. The directory structure is up to the user, however we recommend using our standard layout:

```
checks
    |--Rules
    |--Warnings
```

The final directory structure is configurable through either JSON config, or command-line arguments.

Each script returns a JSON-encoded object to `stdout` that meets the following specification. Every check that is added to the system must abide by this specification in order to be added to the record. If a check does not meet this specification its output will be ignored.

```
{
    "name":"Name of test",
    "violations":9000,
    "Output":"Test Specific output. Either JSON or String"
}
```

3.1.1 Rules

Rules are strict rules about the cluster. They should never be ignored, or disobeyed, and any violation of a set rule results in a violation. Ideally, a “healthy” cluster should have no rule violations.

An example rule would be that a specific column only contains integers.

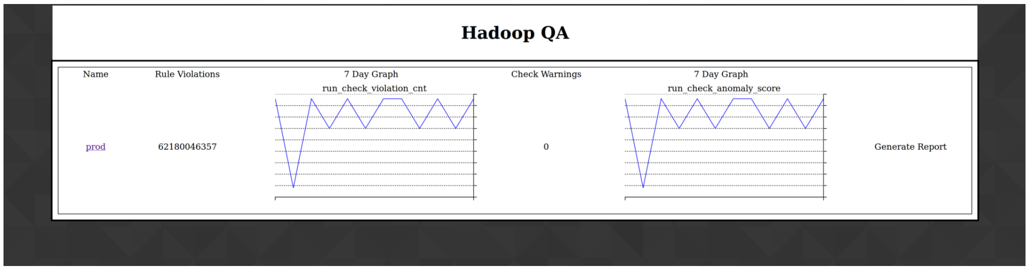
3.1.2 Warnings

Checks are suggestions about specific tables, or databases. These are a lot more fluid, and a warning from a check does not necessarily indicate an issue with the database, but rather that something new may have happened.

For example, a check that examines the average number of a column may throw a warning if an entry is too far from the mean. This isn't an indication that the data is bad, just that it *may* be bad.

3.2 Viewing Results

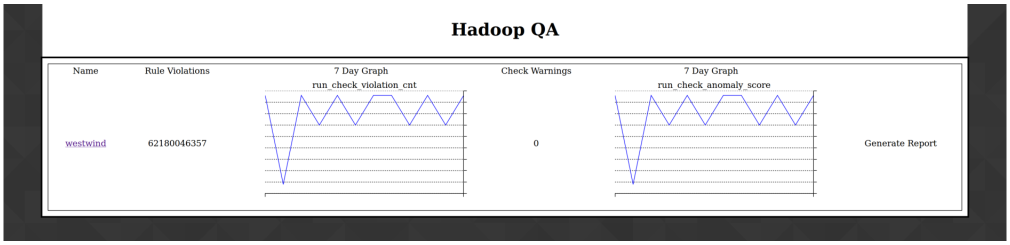
All results can be viewed through the front end. This provides a way to analyze your cluster's health, as well as its current status.



Hadoop QA

Name	Rule Violations	7 Day Graph	Check Warnings	7 Day Graph	
prod	62180046357	run_check_violation_cnt	0	run_check_anomaly_score	Generate Report

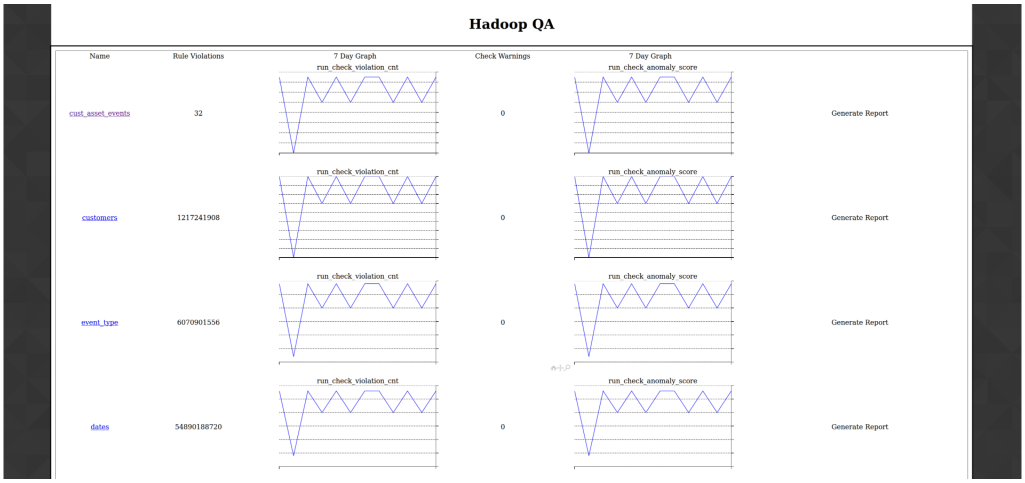
Figure 1: Index



Hadoop QA

Name	Rule Violations	7 Day Graph run_check_violation_cnt	Check Warnings	7 Day Graph run_check_anomaly_score	
westwind	62180046357		0		Generate Report

Figure 2: Environment View



Hadoop QA

Name	Rule Violations	7 Day Graph	Check Warnings	7 Day Graph	
cust_asset_events	32	run_check_violation_cnt	0	run_check_anomaly_score	Generate Report
customers	1217241908	run_check_violation_cnt	0	run_check_anomaly_score	Generate Report
event_type	6070901556	run_check_violation_cnt	0	run_check_anomaly_score	Generate Report
dates	54890188720	run_check_violation_cnt	0	run_check_anomaly_score	Generate Report

Figure 3: Database View

3.3 Configuration

Most aspects of this framework can be configured using the JSON files located in `/config`. As of August 14, 2015, the files are located in `/config`.

4 Current Status

Our initial focus has been on building a demo to help us validate ideas, and build some of our UIs. This includes:

- **hadoopinspector-demogen -** Can generate 50, 000+ check results against a hypothetical user hadoop environment
- **server -** Runs a website that allows the user to analyze these demo results
- **report -** Produces a pdf check result summary report