Financial Risk Data Analytics Pipeline and Lakehouse

aws partner network

About the Client

The client is one of the world's largest ratings agencies and has recently acquired data analytics firms from whom it needs to consume massive volumes of risk analytics data to establish a new level of business providing environmental, social, governance, and climate risk products to clients downstream.

Executive Summary

The client was looking to automate manual QA processes to reduce testing cycle times.

Key Challenge/Problem Statement

When the customer engaged with VR, they were ready for their first release of their new big data pipeline and data warehouse product. It would ingest, process, and store millions of records each month and provide this data to multiple end user applications, including dashboards, data marts, and data analytics platforms. The system was expected to grow to many billions of records processed from hundreds of different data sources and transformed in as many different ways. The customer rightly decided that manual testing could not execute the thousands of tests necessary to ensure their desired level of quality before each release.

Proposed Solution and Architecture

The customer's big data pipeline ingests large environmental, social, governance, and climate risk data files and transforms them into highly normalized, partitioned, efficient AWS Glue table structures stored in S3. The output data can then be queried with various AWS tools and consumed by downstream applications. The following steps is a simplified view of the data pipeline architecture.

1. In step 1, wide (unstacked) format input CSVs are loaded into the landing S3 bucket.
2. Lambda functions invoke a parallel execution of the data pipeline using each file.
3. Data checks run against each file and passing files are loaded into the Glue Data Catalog.
4. Data is stored in S3 in the Parquet format. If all goes well, the data values match the CSV but are instead stored in long (stacked) format.
5. Downstream tools and applications consume data for analysis, visualization, and further ETL

The customer charged VR with building an automated testing framework to verify the behavior of this pipeline. The customer left the choice of testing strategy and tooling up to VR, so long as it satisfied the requirement to speed up testing cycles. VR's early analysis uncovered additional requirements for the testing framework. VR concluded that the proposed testing framework solution had to meet the following requirements:

- Improve on the speed of the current testing cycle considerably
- Be scalable, in both speed and maintainability, to thousands of tests
- Provide production readiness information to high level business stakeholders

Since the testing framework was meant to provide releasability feedback to business stakeholders, VR recognized that a unit testing framework would not be sufficient, since those types of tests inform developers rather than business leaders. The tests had to provide verification of the system as a whole, and they had to report results in a way the business could understand.
VR chose to design and implement an automated Acceptance Testing stage using Behavior Driven Development (BDD) methods.

VR built the testing framework using the Python pytest testing library and two main pytest plugins, pytest-bdd for BDD implementation and pytest-xdist for parallel test execution. Like other BDD frameworks, pytest-bdd defines test cases as scenarios in a Given, When, Then format (using a language called Gherkin) in files called feature files.

VR first implemented the test cases that would deliver the greatest benefit for the smallest development time investment. To this end, VR sought to find a quick path to checking the system's actual behavior against its expected, correct behavior. Once tests were in place to check the pipeline's reported statuses for various files, VR designed and implemented tests that exhaustively compared the integrity of input data vs output data, which precisely highlighted any piece of data that was transformed incorrectly.

Results

VR reduced the amount of time required to verify correct transformation of the data from an average of 1 hour per dataset to approximately 20 minutes per dataset. For the 27 datasets, this translated to a reduction from 27 hours of tester time to 20 minutes of compute time. Crucially, these tests were run in parallel for all 27 datasets. The automated tests run on one small Jenkins worker pod on Amazon EKS, whereas accomplishing the same volume of parallel execution with manual testers would have required the customer to pay 27 testers. When the number of tests greatly increases, pytest-xdist can support distributed test execution across more Jenkins workers.

Summary

By engaging with VR, the customer obtained a carefully planned, scalable, and maintainable testing framework that dramatically reduced testing time for their mission-critical application and enabled them to constantly test the applications releasability. VR's consultants took a business-focused approach rather than ladening the customer with technology aimed at developers only. While it will also improve developers' lives, VR's work will help improve the customer's overall business outcomes by getting them to market faster, more safely, and more confidently.