BUILD DATA PIPELINES WITH LAKEFLOW SPARK DECLARATIVE PIPELINES

→ Spark Declarative Pipeline

(#) Spark Declarative Pipeline.

PROBLEMS WHILE BUILDING PIPELINES
- Labor intensive development
- Operational complexity
- Siloed Batch and Streaming

BENEFITS OF USING SDP
- Simplified Pipeline Authoring
- Intelligent Optimisation at scale
- Unified batch and streaming

DATA SOURCES
- Cloud Storage : S3, ADLS etc.
- Message Queues : Kafka, Pub/Sub etc.
- Databases : SQL Server, PostgreSQL etc.
- SaaS : Workday, ServiceNow etc.

→ DATASET TYPES Overview

(#) Streaming Table
- only processes new data
- exactly once processing of the file
- CREATE OR REFRESH STREAMING TABLE ... FROM STREAM read-files (...)

(#) Materialised View
- Records are processed as required to return accurate results from current data state
- Can be used anywhere in the pipeline
- Used for transformations, aggregations, pre-computing slow queries and frequently used computations
- Incrementally refresh for materialised views : Serverless compute only
- CREATE OR REFRESH MATERIALISED VIEW ... FROM ...

(#) Views
- constructs a virtual table with no physical data based on the query in your Declarative Pipelines
- registered as an object to Unity Catalog
- cannot have streaming queries / be used as a streaming source
- CREATE VIEW ...

(#) Temporary Views
- only persist across lifetime of pipeline and private to defining pipeline
- not registered as a Unity Catalog object
- useful as an intermediate queries that are not exposed to end users

→ Simplified Pipeline Development

Multi File Editors In Lakeflow SDP

- makes developing and debugging the ETL pipelines easier and more efficient

KEY FEATURES

- Pipeline asset browser
- Multi-code editor for step-by-step pipeline development
- Pipeline specific toolbar for quick access
- Interactive DAG for visualising dependencies
- Data previews to inspect intermediate results
- Execution insights panel
- Easier debugging with integrated tools
- Easier and faster validation through dry run capabilities

→ Common Pipeline Setting

Things to choose

- Compute : Choose resource and environment
- Code Assets : Manage files and code modules
- Configuration : Set pipeline parameters

Serverless Compute

- Recommended Option
- optimises costs while maintaining strong performance
- support for incremental refresh of materialised views

Classic Compute

- has autoscaling
- user need appropriate permissions to create compute resources for the Declarative Pipelines
- not recommended but good option

Code Assets

- Pipeline root folder: set to automatically, includes all relevant files within that folder for pipeline project
- Source code section : lets us specify which subfolders/individual files to include in the pipeline

Configuration

- A pipeline's configuration is a map of key value pairs that can be used to parameterize the code
- Improves code readability and maintainability
- Reuse common parameters in multiple pipeline files
- In SQL, reference the key's value using the {key} syntax e.g. {source}

→ Ensure Data Quality

CONSTRAINT cons_name ~~EXCEPT~~ EXPECT (column condition) [ON VIOLATION action]

- CONSTRAINT valid-~~notf~~ date EXPECT (timestamp>"2021-01-01") ON VIOLATION DROP ROW
- CONSTRAINT valid_~~date~~ notif EXPECT (notification IN ('Y', 'N'))
- CONSTRAINT valid-id EXPECT (custid IS NOT NULL) ON VIOLATON FAIL UPDATE

Actions

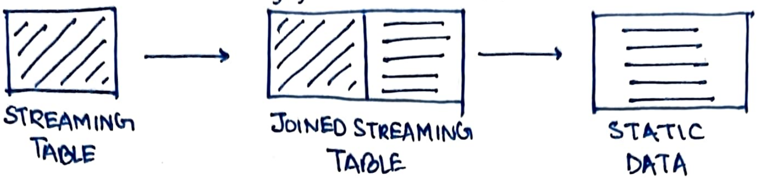
- Warn: logs a warning but still writes invalid rows to target
- Drop: drops invalid rows from output
- Fail: fails specific flow, requires manual intervention

```
CREATE OR REFRESH STREAMING TABLE silver.orders_s{
(CONSTRAINT valid_notif EXPECT (notifications IN ('Y','N')},
 CONSTRAINT valid_id EXPECT (cust_id IS NOT NULL) ON VIOLATION DROP ROW)
 AS SELECT....
```

→ Streaming Joins

Streaming Joins - Streaming w/ Static Data

- Goal is to incrementally join new data with static table



- Data is joined in real life with reference table
- Only new rows are joined with static lookup table

Streaming Joins — Streaming w/ materialized view

- Goal is to take all rows from two streaming tables and join them together each time pipeline is run
- Materialised view efficiently computes all data in streaming tables and joins the row

Streaming Joins - Streaming w/ Incremental Stream

- Goal is to incrementally join new data from two tables
- Past data is not used

→ Deploying Production to Pipeline

Scheduling

- Two types : triggered and continuously

TRIGGERED·

- Manually / run on a schedule
- Refreshes selected tables using data available at the start & stops once complete

CONTINUOUSLY

- continuously processes new data to keep streaming tables and materialised views up to date in real time
- Monitors dependencies and updates only when source data changes

Email Notification

- we can configure email notification when scheduling the pipeline to keep stake-holders informed

Monitor Pipelines

- Captures all key information about pipeline, including audit logs, data quality checks, pipeline progress and data lineage

Querying the DP Event Log

- Publish event log as Delta table
- Event log is written as hidden Delta tables

→ CDC Overview

CDC

- Capture Data Change
- technique used to track and capture data changes ~~as~~ in a data source

Slowly Changing Dimensions

SCD TYPE 1

- when records update, previous record is simply overwritten by its key with the new value
- when record is deleted by its key, the ~~remove~~ record is removed
- No tracking of old keys
- Used when only current data matters and historical data is irrelevant

SCD TYPE 2

- when records change, old record is preserved with an additional column indicating its validity period
- New record is inserted with updated information
- when record is deleted, record is kept and a column indicates that record is inactive
- Used when historical data is important, and system needs to track how attributes change over time

BASIC DIFFERENCE

- Type 1 overwrites existing data (no tracking) while Type 2 tracks historical changes by storing previous version of records

USING AUTO CDC INTO in SDP

```
CREATE OR REFRESH STREAMING TABLE customers;
CREATE FLOW flow1 AS AUTO CDC INTO customers FROM ~~sh~~ STREAM updates
KEYS (customerID) APPLY AS DELETE WHEN operation = 'DELETE'
SEQUENCE BY processDate COLUMNS * EXCEPT (operations) STORED AS SCDTYPE1;
```

DevOps Essential For Data Engineering...

→ Introduction

Best coding practices

CODE READABILITY
- Write code that is easy to understand and maintain

NAMING CONVENTIONS
- Use descriptive consistent names for variable functions and classes

MODULAR DESIGN
- Break down software into functions

Document Code

- Improves Code Maintainability
- Enhances collaboration
- Facilitates Knowledge Transfer

Automated Testing

UNIT TESTS
- Verifies functionability of a single unit in isolation

INTEGRATION TESTS
- Tests how different components/systems work together

Version Control and Code Review

VERSION CONTROL
- Use Git and collaborating tools

CODE REVIEW
- Helps catch bugs easily and early

CI/CD

CONTINUOUS INTEGRATION
- A practice where developers frequently commit, build, test and release code to a shared repository

CONTINUOUS DELIVERY
- Automatically release the code to production after passing automated tests

Workspaces

- Utilizing multiple workspaces, one for each environment (like testing and staging and development)

Catalogs

- Utilize multiple catalogs, one for each environment

Databricks Tools Overview

- Develop code and run unit tests in Databricks Workspace or locally using the Notebooks or Files (SQL/Python)
- Utilise Databricks Git folders to provide version control and signifantly improve the workflow
- Focus on using Unity Catalog with a single or multiple Workspaces to isolate the environment securely
- Git code tested and deployed via CI/CD pipelines using Databricks deployment tools

→ Modularising PySpark code

Non Modular Code

- Everything is in one block, making it harder to modify or test specific parts
- Code duplication could occur if same operations are needed elsewhere in the project

Modular Code

- Converting snippets of code into its function
- Easier maintainence by updating specific functions
- Reuse functions in different projects
- Testing individual functions through unit tests

→ DevOps Fundamental

DevOps

- practice that combines Software Engineering Best Practices with IT Operations to deliver software more rapidly, efficiently and with higher quality

BENEFITS

- Faster development cycles
- Improved collaboration between teams
- Enhanced system reliability
- Better scalability and efficiency

KEY STEPS OF DEVOPS LIFECYCLE

(i) Planning - define project goals, gather requirements and do proper planning with them

(ii) Coding - developers write app's source code, building source code, building features and functionality

(iii) Building - Compile the code into executable files, ensuring all dependencies are correct integrated

(iv) Testing - Run automated tests to ensure the code works as expected, catching any bug before releasing

(v) Release - Package the app, ensuring its production ready and prepare for a controlled rollout

(vi) Deploying - Push the app to production environment and make it available to users

(vii) Operate - Monitoring the performance, managing resources and quickly addressing any issue

(viii) Monitoring - We track application's performance, gather feedback and continuously work on improvements

Data Ops

- subset of DevOps that applies DevOps to Data Engineering
- automates management of data pipelines, ensuring smooth, reliable data flows from collection to processing

ML Ops

- subset of DevOps that applies DevOps to Machine Learning
- streamlines process of deploying and managing ML models, ensuring models move from development to production quickly and monitored for performance

DEV OPS

(i) Automate CI/CD

(ii) Enable continuous code testing

(iii) Version control

(iv) Establish Production grade Lakeflow Jobs

(v) Orchestration and Automation

(vi) Monitor system performance

DATA Ops

(i) Optimise data processing

(ii) ~~Contraw~~ Centralised data discovery management and governance

(iii) Establish traceable data lineage and monitoring

(iv) Enhance collaboration across teams

(v) Monitor data quality

MLOps

(i) Treating model code as softwar

(ii) Treating models as data

(iii) Manage the model lifecycle

(iv) Monitor model performance

→ Role of CI/CD in DevOps

(#) CI/CD Overview

- Automates and streamlines software development process
- Improves code quality, speed and reliability
- Code is deployed to production through an automated process
- Enable development and delivery of software in short cycles
- Use automated pipelines to ensure faster development and consistency
- Growing importance in Data Engineering and Data Science

(#) Continuous Integration

- CI involves regularly merging code changes from multiple contributors into a central repository and running automated tests to ensure data code quality

BENEFITS

- Early detection of issues
- Faster Development cycles
- Improved collaboration and code quality
- Automated Testing and Verification

HIGH LEVEL TESTING STEPS

↑ Slow | Fast ↓

System Tests: Test entire app, ensuring all parts work together in a real world scenario

Integration Tests : Test the interaction between different components or system

Unit Tests : Test the individual functions or methods in isolation.
Fast, low cost, high coverage and automated

e.g. System Tests → end to end data pipeline
Integration Tests → notebooks
Unit Tests → Custom PySpark function

(#) Continuous Delivery / Deployment

- automating process of pushing changes to staging or pre-production environments
- Once a change passes all tests, its automatically deployed to production

→ Project Planning

(#) Dev Data

(i) Often a small static subset of production data

(ii) Can be anonymised

(iii) Supports rapid testing and development

(#) Stage Data

(i) Staging Data Mirrors Production Structure and Volume

(ii) Can be anonymised and scrubbed sensitive info

(iii) Ensure realistic testing and validation

(#) Production Data

(i) Live and Fully operational

(ii) Contains real user data

(iii) Continuously updated

(iv) Requires high security, privac
privacy and compliance standards

(*) You can isolate your dev, stage and prod environments at the Workspace & storage level

⊛ You can also use Unity Catalog Isolation for isolating environment
- With this method, we create a catalog for dev, stage and production
- Can also utilise Unity Catalog ~~across~~ access control for developers, only giving them the required permissions for each

→ Unit Tests

Unit Tests

- Tests only one specific function on small amount of data

PYSPARK.TESTING.UTILS
- assertDataFrameEqual ({actual}, {expected}, ...)
- assertSchemaEqual ({actual}, {expected})

PYTEST
- is popular testing framework for Python that makes it easy to write simple & scalable test cases
- uses simple syntax
- provides assertions
- automatic discovery
- rich ecosystem

→ Executing Integration Tests

Spark Declarative Pipelines

- SDP reads the data from corresponding target environment
- Same SDP transformation logic which includes custom functions is applied in each environments
- we then do tests on all environments to validate

Tasks

- Performing units tests to confirm all created functions work correctly
- Executing a SDP to create the necessary tables without using expectations
- Add notebooks or files within a task to perform integration tests within a job

→ Git
- Complementary concept to CI/CD
- Enables effective CI

→ Continuous Development

Deployment Options

RESTAPI — Postman, Databricks

DATABRICKS CLI
- wraps the Rest API
- Ideal for one-off tasks and shell scripting

DATABRICKS SDK
- Python, Go, R, Java supported
- Most developer friendly
- Best for embedding Databricks functionality in applications

Databricks Asset Bundles

- Version Control
- Code Review
- Testing
- Continuous Integration

⊛ DABs provide a structured approach to managing Databricks projects while adhering to the Software Engineering best practices

⊛ YAML files ~~that~~ specify the artifact, resources and configurations of a Databricks project. This leads to easy configuration of complex notebook and pipeline interactions and reproducibility

⊛ Bundles provide an exact definition of Databricks resources that are to be used within a project with support for validation and deployment instructions