Iceberg\_Demo\_1 Run Book

# 1. Introduction

This document outlines the process of ingesting and analyzing NYC Yellow Taxi trip data using Apache Iceberg, detailing the setup procedures, data ingestion steps, and query performance analysis across different execution engines.

# 2. Setup and Data Ingestion Process

## 2.1 Service Initialization

To begin the process, initialize all required services using Docker Compose with the following command:

**docker compose up -d**

## 2.2 Data Source Configuration

The source data file must be added to the MinIO bucket with the following specifications:

• Storage path: s3a://source-data/yellow\_tripdata\_2016-01.csv

• File size: 1.6 GB

A screenshot of a computer

Description automatically generated

## 2.3 Iceberg Table Creation

Execute the ingestion script using Spark Submit:

**docker-compose exec spark spark-submit /opt/spark/work-dir/scripts/basic\_spark\_iceberg.py**

### Table specifications:

• Table name: taxi\_data.yellow\_trips

• Partitioning: Implemented on passenger\_count column

• Metadata management: Utilizes Hive metastore for catalog

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## 3. Query Analysis and Performance Evaluation

### 3.1 Implemented Queries

Query 1: Total Ride Count

*SELECT count(1) as total\_rides FROM taxi\_data.yellow\_trips;*

Query 2: Popular Pickup Location Analysis

*SELECT pickup\_latitude, pickup\_longitude, COUNT(\*) as total\_rides*

*FROM taxi\_data.yellow\_trips*

*GROUP BY pickup\_latitude, pickup\_longitude*

*ORDER BY total\_rides DESC*

*LIMIT 5;*

Query 3: Passenger Count Statistics

SELECT passenger\_count, COUNT(\*) as trip\_count,

AVG(trip\_distance) as avg\_distance,

AVG(total\_amount) as avg\_amount

FROM taxi\_data.yellow\_trips

GROUP BY passenger\_count

ORDER BY passenger\_count;

## 3.2 Performance Results

### 3.2.1 Spark SQL Execution and Performance Metrics

A screen shot of a computer

Description automatically generated

• Query 2 (Location Analysis): Approximately 15 seconds execution time

• Query 3 (Passenger Statistics): Approximately 2 seconds execution time

• Performance Analysis: Query 3 demonstrates significantly faster execution due to the implemented partitioning on the passenger\_count column

### 3.2.2 Trino Execution and Performance Metrics

Q1

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Q3

A screenshot of a computer

Description automatically generated

• Query 2 execution time: 4.27 seconds

• Query 3 execution time: 2.19 seconds

• Key Observation: Trino demonstrates superior query performance compared to Spark SQL

• Technical Detail: Trino utilizes Hive metastore for catalog operations

## 4. System Architecture Components

### 4.1 Core Components

• Data Storage Layer: MinIO (S3-compatible storage)

• Processing Engines: Apache Spark and Trino

• Metadata Management: Apache Hive Metastore

• Table Format: Apache Iceberg