Research & Development:
Enterprise Architecture (Part 2)

By Krai Chamnivikaipong

Advisor:
Dr. Akkarit Sangpetch
Dr. Orathai Sangpetch

CMKL UNIVERSITY

Table of Contents

1. Motivation

An author is working on a project named Enterprise Architecture. Given the client's current data warehouse design is aggregating all the data via Apache Kafka and storing it in PostgreSQL, the stored data is then later brought into use for several analytical use-cases (e.g., executive summary, real-time dashboard, etc.). However, for the mentioned architecture, there is still room for improvement in several aspects such as processing speed, scalability, availability, and usability.

Inspired by the opportunity, the author went through several existing solutions for data warehouse, data lake, and data lakehouse [1]. Apparently, the concept of data lakehouse [2] started to be spread around 2020 and there are several advertised tools out there in the community. However, there are still not many examples and published use-cases by the community on the available open-source tools. Moreover, **combining these tools together to bring up the most efficient** in terms of processing power, availability, fault tolerance, and other attributes from these tools are considered to be challenging or even something new to the industries. As there usually is a tradeoff between attributes for another. Hence, the author is decided to explore the tools based on the rough idea of the following metrics.

- Performance: processing speed, availability, fault tolerance, etc.
- Configuration Complexity: deployment, network configuration, etc.
- Integration Complexity: integration with other available tools in the ecosystem
- Learning Curve: documents, tutorials, syntax, knowledge to be consumed
- Community: popularity, helps, and support from the internet

The author then decided on the **solution that is able to balance all the attributes in the metrics without showing any sign of significant negative impact** (configuration failure, low fault tolerance, no support documents, etc.) **toward the user later**. Though, weights for the performance, configuration complexity, and integration complexity will be heavier than other attributes in the metrics.

A design and tools for a new Enterprise Architecture in the infrastructure layer, data ingestion layer, data storage layer, data writing formats layer, and data computation layer was already settled which are Kubernetes, Apache Kafka, Minio, Delta Lake, and Apache

Spark, respectively. Please refers to the previous report on Research & Development: Enterprise Architecture [20]

However, throughout the previous report Apache Spark has only been deployed on a single local machine instead of Kubernetes. Therefore, the author's focus remains on evaluating the data computation layer (Apache Spark deployment in Kubernetes), the data source aggregation layer, and the data analytical and visualization layer.

2. Problem

Hence, in order to **find the most efficient combinations** out of the existed tools, the following steps are required for each tool to be evaluated:

a. **Feasibility**: if the tools can be configured or are simple enough to configure given the circumstances of every party in holistic views

b. **Integration**: if the tools can be integrated with a previously selected tech stack and other tools with an acceptable configuration complexity

c. **Performance**: if the tools are capable of delivering a solution according to the use-case and performance of the tools is acceptable in terms of processing power in comparison to traditional (or current) methods

The tools will be separated into 5 layers which are the data storage layer, data computational layer, data writing formats layer, data source aggregation layer, and data analytical and visualization layer. The following diagram represents a high-level architecture of data lakehouse solutions and the tools of their category.

Data storage (file distributed storage) is one of the powerful tools that allow the user to store any type (or format) of files into it. Given such a property, it can be referred to (worked) as a lake of data where any kind of data can be stored (dumped) in it. However, the data storage itself is insufficient in handling and organizing the stored data, this is where the computational tools and writing formats came in. The writing formats help reducing the size of the CSV files by approximately 10 times. With a help of computational tools that are able to process and transform a very large size of data in a short period of time, it can easily transform data into desired writing formats in no time. Some computational tools are also able to use for analytical purposes as well.

In addition to the general writing format (Parquet), several producers have developed new writing formats which created metadata or additional schema on top of the Parquet to ensure the ACID properties and provide extra features such as time-traveling too. It helps in organizing, indexing, and handling the lake of data which is a crucial part that leads to a concept of a data lakehouse.

However, it is almost impossible that an enterprise will only have a single source of data (storage) to store the data or even a single type of data storage. Hence, managing, analyzing, and developing a solution using several data sources (storage) can be troublesome (chaotic). To solve the problem, data source aggregator tools are being introduced. The basic idea of the data source aggregator is to map (catalog) the data source and its stored data (database table, metadata) to the interface of the data source aggregator tool, where the user can instead operate (e.g., query) on that interface.

Finally, when everything looks clean and seems to work together as a single system, analytical tools came into play where they allow the user to do the analytics and visualization on them.

Despite several tools being mentioned out there in the community, there are some tools that have a strong disclaimer or are heavily mentioned in the area of their own use. Thus, the following tools are selected by the author from the previous report [20] to be a part of the system and new tools are being selected to be evaluated:

2.1 Data Storage Layer

Minio [3]: high-performance, S3 compatible object storage. Native to Kubernetes, it is the only object storage suite available on every public cloud, every Kubernetes distribution, the private cloud, and the edge. It is software-defined and is 100% open source under GNU AGPL v3.

2.2 Data Computation Layer

Apache Spark [4] a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

2.3 Data Writing Formats Layer

Delta Lake [5] an open-source project that enables building a Lakehouse architecture on top of data lakes. Delta Lake provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing on top of existing data lakes, such as S3, ADLS, GCS, and HDFS.

2.4 Data Source Aggregation Layer

Dremio [7]: an open-source data lake house platform that brings all the performance and functionality of a data warehouse to the data lake, and supports all SQL workloads from mission-critical BI dashboards to exploratory workloads. It also enables self-service data access to anyone who knows SQL and allows the user to connect with a BI tool(s), and drive interactive dashboards on fresher data as well.

Apache Hive [8]: data warehouse software facilitates reading, writing, and managing of large datasets residing in distributed storage using SQL. A structure can be projected onto data already in storage. A command-line tool and JDBC driver are provided to connect users to Hive.

Trino [9]: a highly parallel and distributed query engine, that is built from the ground up for efficient, low latency analytics. It is also an ANSI SQL compliant query engine, that works with BI tools such as R, Tableau, Power BI, Superset, and many others.

2.5 Data Analytical and Visualization Layer

Tableau [10]: a data visualization tool used in the Business Intelligence Industry. It helps in simplifying raw data in a better understandable format. Tableau helps create data that can be understood by professionals at any level in an organization. It also allows non-technical users to create customized dashboards. Data analysis is very fast with the Tableau tool and the visualizations created are in the form of dashboards and worksheets. The best features of Tableau software are Data Blending, Real-time analysis, and Collaboration of data.

Superset [11]**:** a modern data exploration and visualization platform. lightweight and highly scalable, leveraging the power of your existing data infrastructure without requiring yet another ingestion layer. It can connect to any SQL-based data source through SQLAlchemy, including modern cloud-native databases and engines at a petabyte-scale. Its visualization plug-in architecture makes it easy to build custom visualizations that drop directly into Superset.

3 Solution/Evaluation/Analysis

As is previously mentioned in the problem section that there are 3 steps for the tools to be evaluated which are **feasibility**, **integration**, and **performance**. Despite there is a single main idea for the evaluation process which is trial and error, there is still a detail for different solutions and methods for each step that can be further discussed in the section.

Note: Please refers to the ***Appendix Section A*** for how to set up an experimental Kubernetes cluster using Rancher.

3.1 Data Computation Layer

3.1.1 Apache Spark

As previous research only evaluated Apache Spark as a tool being deployed in a local machine in the cluster and its compatibility with other selected tools. The current report, now, focuses more on deploying a running Apache Spark in the Kubernetes cluster, as the majority of the integration evaluation has already been evaluated.

a. Feasibility

There are mainly 3 methods that are being used to execute (deploy) Apache Spark on the Kubernetes cluster which are briefly summarized as follows:

1. Direct Spark Image Submission [12]: Set up a Spark service account and submit a Spark image (user custom image is preferable) into a Kubernetes cluster

2. Spark-Operator [13]: Deploy a Spark-Operator and submit a Spark image to a Spark-Operator's resource in the Kubernetes cluster

3. Bitnami Spark [14]: Deploy a pre-build Spark system (master, worker, etc.) by Bitnami through helm install

Given limited time constraints after several trials and errors, the last method (Bitnami Spark) is selected by the authors as it is the only working method with the least complexity under the current circumstance.

Deploying Apache Spark via Helm Chart on Kubernetes:

```
helm install [ RELEASE_NAME ] --set image.tag=3.1.2 binami/spark
```

b. Integration

Works perfectly on the previously selected tech stack (Minio and Deltalake) with the following configurations:

Step 1: Get the Spark worker image docker ID

Step 2: Access Spark worker image

Step 3: Copy the script (python, jar, etc.) to be executed into Spark worker

Step 4: Setup the environment for Spark worker to integrate with a Deltalake

Step 5: Spark-Submit

Note: Please refers to the ***Appendix Section H*** for a detail in executing Spark-Submit.

c. Performance

Example pi.py script is being executed to prove that all the Spark workers can run a script in parallel with a proper script (code) and actually reduce computational time, which gives a positive result as presented in a table below.

Important: Local Machine Specification:

- CPU - Intel(R) Core(TM) i5-7500T CPU @2.7 GHz
- RAM - 16 GB

For a detail specification, please refers to the provided link

	Local Machine	**4 Spark Workers**	**32 Spark Workers**
pi.py	280 seconds	270 seconds	90 seconds

Table 3.1.1.c.1: Performance Comparison Table between different number of Spark worker nodes

Next, an example SQL query statement provided by ThaiBev then being tested. The following queries are being executed to compare Spark on a single local machine to a ThaiBev environment:

SQL Query Statement: TSR_RPT_VSMS_ROUTEPLAN

```
SELECT


SaleOrgId, SaleOrgName, AssignedDate, RoutePlanId, RouteId, RoutePlanDetailId,
RouteDesc, DelFlag, IsShopClose, CreatedDate, CreatedByUserId, UpdatedDate,
UpdatedByUserId, PlanType, RoamingType, Remark, CustomerCode, CustomerId ,
SOCustomerId, CustomerName, CustomerType, Latitude, Longitude, TaxNo,
BillCompanyName, BillNo, BillMoo, BillVillage, BillBuilding, BillFloor, BillRoom ,
BillSoi, BillRoad, BillSubDistrict, BillDistrict, BillProvince, BillCountry ,
BillZipCode, WorkCompanyName, WorkNo, WorkMoo, WorkVillage, WorkBuilding,
WorkFloor, WorkRoom, WorkSoi, WorkRoad, WorkSubDistrict, WorkDistrict,
WorkProvince, WorkCountry, WorkZipCode, WorkTelephone, WorkFax,
HierarchyTerritoryId, ChannelId, ChannelSAPCode, ChannelDesc, RegionGroupId,
RegionGroupCode, RegionGroupName, AreaGroupId, AreaGroupCode, AreaGroupName,
BranchGroupId, BranchGroupCode, BranchGroupName, BranchId, BranchCode, BranchName,
SalesTeamId, SalesUnitDesc, TerritoryId, TerritoryDesc, Tag


FROM delta.`s3a://tb-route-plan/tsr_rpt_vsms_routeplan`


WHERE DelFlag = 0 AND IsShopClose = 0 AND AssignedDate

BETWEEN '2021-10-01' AND '2022-01-31' and (SaleOrgId = 1 or SaleOrgId = 2)
```

SQL Query Statement: RPT_VSMS_SALES_SSC

```
SELECT


PM.PaymentId, PM.OnDate, PM.ReceiptNo, PM.DOCUMENT_NUMBER,
SO.SaleOrderId,PM.SOCompanyBranchId, SO.latitude, SO.longitude, SO.AduserId AS
UserName, so.ShelfId, so.SOCustomerId, PMSKU.SOProductId, P.ProductCode,
P.ProductName, SO.SaleType, PMSKU.ItemType, PMSKU.SKUUnitTypeId, PMSKU.Quantity
PaymentQuantity, PMSKU.Price, PMSKU.Amount, PMSKU.vat, PMSKU.VatRate,
PMSKU.NormalDiscount, PMSKU.SpecialDiscount, PMSKU.ManualDiscount,
PMSKU.SalesDiscount, PMSKU.PromotionDiscount, PM.CreatedDate, PM.UpdatedDate,
PM.CompanyCode, hst.BranchGroupCode AS BranchCode, so.UserId,
PMSKU.RefPaymentSKUId, SO.SalesTeamId SaleOrderSalesTeamId, so.Status
SaleOrderStatus, PM.IsCancel, SO.StorageCode, DO.DeliveryOnDate DO_DATE,
PM.PlantCode , PMSKU.PaymentSKUId


FROM delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_payment_ssc` PM


LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_paymentsku_ssc`

PMSKU ON PMSKU.PaymentId=PM.PaymentId and PMSKU.SaleOrgId=2


LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_saleorder_ssc` SO ON
SO.SaleOrderId=PM.SaleOrderId AND PM.DelFlag=0 AND SO.SaleOrgId=2


LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_delivery_ssc` DO ON
DO.DeliveryId=PM.DeliveryId AND DO.PaymentId = PM.PaymentId AND DO.SaleOrgId=2


LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_hierarchysalesteam` hst ON
hst.SalesteamId=PM.SalesTeamId AND hst.SaleOrgId=2


LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_product` P ON P.SOProductId =
PMSKU.SOProductId AND P.SaleOrgId=2


WHERE CAST(PM.OnDate AS date) BETWEEN '2021-10-01' AND '2022-01-31'
AND (so.Status = 3 OR so.Status = 4 or so.Status = 6) AND PM.SaleOrgId=2
```

SQL Query Statement: RPT_VSMS_SALES_CVM

```
SELECT

PM.PaymentId, PM.OnDate, PM.ReceiptNo, PM.DOCUMENT_NUMBER, SO.SaleOrderId,
PM.SOCompanyBranchId, SO.latitude, SO.longitude, SO.AduserId AS UserName,
so.ShelfId, so.SOCustomerId, PMSKU.SOProductId, P.ProductCode, P.ProductName,
SO.SaleType, PMSKU.ItemType, PMSKU.SKUUnitTypeId, PMSKU.Quantity PaymentQuantity,
PMSKU.Price, PMSKU.Amount, PMSKU.vat, PMSKU.VatRate, PMSKU.NormalDiscount,
PMSKU.SpecialDiscount, PMSKU.ManualDiscount, PMSKU.SalesDiscount,
PMSKU.PromotionDiscount, PM.CreatedDate, PM.UpdatedDate, PM.CompanyCode,
hst.BranchGroupCode AS BranchCode, so.UserId, PMSKU.RefPaymentSKUId, SO.SalesTeamId
SaleOrderSalesTeamId, so.Status SaleOrderStatus, PM.IsCancel, SO.StorageCode,
DO.DeliveryOnDate DO_DATE, PM.PlantCode, PMSKU.PaymentSKUId


FROM delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_payment` PM

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_paymentsku` PMSKU ON
PMSKU.PaymentId=PM.PaymentId AND PMSKU.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_saleorder` SO ON
SO.SaleOrderId=PM.SaleOrderId AND PM.DelFlag=0 AND SO.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_delivery`
DO ON DO.DeliveryId=PM.DeliveryId AND DO.PaymentId = PM.PaymentId AND
DO.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_hierarchysalesteam` hst ON
hst.SalesteamId=PM.SalesTeamId and hst.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_product` P ON P.SOProductId =
PMSKU.SOProductId AND P.SaleOrgId=1


WHERE CAST(PM.OnDate AS date) BETWEEN '2021-10-01' AND '2022-01-31'

AND (so.Status = 3 OR so.Status = 4 OR so.Status = 6)

AND PM.CompanyCode = '4900' AND PM.SaleOrgId=1
```

Which results in the following:

CMKL Mini Cluster - SQL Query Runs on Single Local Machine Intel(R) Core(™) i5-7500T CPU @ 2.70GHz RAM 16 GB				**ThaiBev Machine(s)** Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz 2.39GHz RAM 64 GB	
Distributed Minio (4 Nodes), Delta Lake, PySpark			PostgreSQL	PostgreSQL	
SQL Query Statement	Rows / Columns	Average Execution Time (hrs:mins:secs)	Average Execution Time (hrs:mins:secs)	Rows	Execution Time (hrs:mins:secs)
TSR_RPT_ VSMS_ ROUTEPLAN	11,624,334 / 75	00:05:23 (100%)	Crash at approximately 7 million rows	-	-
RPT_VSMS_ SALES_SSC	7,228,470 / 40	00:06:36 (100%)	Crash at approximately 7 million rows	7,493,634	00:16:49 (259%)
RPT_VSMS_ SALES_CVM	11,627,706 / 40	00:10:42 (100%)	Crash at approximately 7 million rows	13,707,236	00:23:46 (225%)

Table 3.1.1.c.2: Performance Comparison Table between Single Local Machine vs ThaiBev Machine.
Please refers to an ***Appendix Section B*** for the comparison of the deviation in SQL queries.

The single local machine then be further compared to the Spark in Kubernetes environment which result as follows:

SQL Query Statement	Rows / Columns	Local Machines (i5, 16GB)	2 Spark Worker Nodes	4 Spark Worker Nodes	8 Spark Worker Nodes	32 Spark Worker Nodes
TSR_RPT_ VSMS_ ROUTEPLAN	11,624,334 / 75	00:05:23 (100%)	00:06:06 (116%)	00:05:27 (100.8%)	00:05:24 (100.1%)	00:05:33 (102%)
RPT_VSMS_ SALES_SSC	7,228,470 / 40	00:06:36 (100%)	00:07:30 (115%)	00:06:33 (99.5%)	00:06:18 (97.2%)	00:06:51 (102.4%)
RPT_VSMS_ SALES_CVM	11,627,706 / 40	00:10:42	00:13:00 (124%)	00:10:30 (98.8%)	00:10:00 (96%)	00:10:00 (96%)

Table 3.1.1.c.3: Performance Comparison Table between SQL Queries against Different number of Nodes

Apparently, there is not much different in processing time despite adding more spark worker node. Only clear different is when there are 2 spark worker nodes which causes a processing time to be approximately 1.18 times slower.

Given the result, further investigation is required. As the stable performance might be due to SparkSQL operation, new operation has been tested (which presented in a metrics below) against a different number of a Spark worker nodes to see if the result still remains a stable trend.

Read Operation	Write Operation
Spark SQL	Spark Write NOOP
Spark Select	Spark Write CSV

Table 3.1.1.c.4: Spark operation to be tested

However, using prior SQL query statements on these operations will only break the RAM, the following SQL query statement have been used instead:

```
SELECT

SaleOrgId, SaleOrgName, AssignedDate, RoutePlanId, RouteId, RoutePlanDetailId,
RouteDesc, DelFlag, IsShopClose, CreatedDate, CreatedByUserId, UpdatedDate,
UpdatedByUserId, PlanType, RoamingType, Remark, CustomerCode, CustomerId,
SOCustomerId, CustomerName, CustomerType, Latitude, Longitude, TaxNo,
BillCompanyName, BillNo, BillMoo, BillVillage, BillBuilding, BillFloor, BillRoom,
BillSoi, BillRoad, BillSubDistrict, BillDistrict, BillProvince, BillCountry,
BillZipCode, WorkCompanyName, WorkNo, WorkMoo, WorkVillage, WorkBuilding,
WorkFloor, WorkRoom, WorkSoi, WorkRoad, WorkSubDistrict,WorkDistrict, WorkProvince,
WorkCountry, WorkZipCode, WorkTelephone, WorkFax, HierarchyTerritoryId, ChannelId,
ChannelSAPCode, ChannelDesc, RegionGroupId, RegionGroupCode,
RegionGroupName,AreaGroupId, AreaGroupCode, AreaGroupName,
BranchGroupId,BranchGroupCode, BranchGroupName, BranchId, BranchCode, BranchName,
SalesTeamId, SalesUnitDesc, TerritoryId, TerritoryDesc, Tag

FROM delta.`s3a://tb-route-plan/rpt_vsms_routeplan_ssc`
```

The result is presented in a table below.

Operation: Write NOOP

```
out_df.write.mode("overwrite").format("noop").save()
```

	Rows / Columns	**Local Machine (core i5, 16GB)**	**4 Spark Worker Nodes**	**8 Spark Worker Nodes**	**16 Spark Worker Nodes**
Spark SQL	39,182,373 / 75	00:04:47 (100%)	00:03:51 (78.5%)	00:03:24 (72.5%)	00:03:27 (73.2%)
Spark Select	39,182,373 / 75	00:04:31 (100%)	00:03:53 (81.9%)	00:03:23 (74.9%)	00:03:29 (76.3%)

Table 3.1.1.c.5: Test Result for Spark SQL and Spark Select then Write NOOP

Apparently for Write NOOP operation doesn't show a drastic change in performance (processing time) given the increasing number of the Spark worker nodes. Though, the result of using Spark worker nodes (4 nodes and more) is better than using single local machine by approximately 20%.

Operation: Write CSV

```
out_df.write.csv(...)
```

	Rows / Columns	Local Machine (core i5, 16GB)	4 Spark Worker Nodes	8 Spark Worker Nodes	16 Spark Worker Nodes
Spark SQL	39,182,373 / 75	00:10:12 (100%)	00:06:09 (60.2%)	00:04:08 (40.5%)	00:03:54 (38.2%)
Spark Select	39,182,373 / 75	00:10:02 (100%)	00:05:50 (58.1%)	00:03:51 (38.4%)	00:03:02 (30.2%)

Table 3.1.1.c.6: Test Result for Spark SQL and Spark Select then Write CSV

Apparently, it shows a significant change as the number of the Spark worker nodes are increased especially from 4 nodes to 8 nodes which is approximately 20%. The performance for Spark worker nodes (4 nodes and more) also better than using single local machine by more than 60%. The change is due to the Write CSV operation which can be done in parallel.

It can be concluded that increasing Spark worker nodes are not strongly affecting the performance (processing time) for the given simple straightforward SparkSQL operation on Delta file format in the following example. However, with a proper code design (for parallelism) and optimization, the author strongly believe that processing time can be improved more or less [15].

3.2 Data Source Aggregation Layer

3.2.1 Apache Dremio

a. Feasibility

Apparently, there are 2 versions of Dremio that are available for deploying in a local Kubernetes cluster (via helm charts). An author is using version 2, as it is recommended by the providers [16].

However, a few configurations in values.yaml are required to be adjusted to better fit with a limited resources of the experimental environment by the author as follows:

```
# Dremio Coordinator
coordinator:
 cpu: 1
 memory: 4100
 count: 0
 volumeSize: 4Gi
 storageClass: openebs-hostpath
```

```
# Dremio Executor
executor:
 cpu: 1
 memory: 4100
 engines: ["default"]
 count: 3
 volumeSize: 4Gi
 storageClass: openebs-hostpath
 cloudCache:
   enabled: true
   storageClass: openebs-hostpath
   volumes:
   - size: 4Gi
```

```
# Zookeeper
zookeeper:
 # The Zookeeper image used in the cluster.
 image: k8s.gcr.io/kubernetes-zookeeper
 imageTag: 1.0-3.4.10

 cpu: 0.5
 memory: 1024
 count: 3
```

```
 volumeSize: 4Gi
 storageClass: openebs-hostpath
```

```
# Dremio Service
service:
 type: NodePort # or LoadBalancer
```

```
# Control where uploaded files are stored for Dremio.
distStorage:
 type: "aws"

 aws:
   bucketName: "test-connect"
   path: "/folder1"
   authentication: "metadata"

   extraProperties: |
     <property>
         <name>fs.dremioS3.impl</name>
         <description>The FileSystem implementation. Must be set to
com.dremio.plugins.s3.store.S3FileSystem</description>
         <value>com.dremio.plugins.s3.store.S3FileSystem</value>
     </property>
     <property>
         <name>fs.s3a.access.key</name>
         <description>Minio server access key ID.</description>
         <value>minio</value>
     </property>
     <property>
         <name>fs.s3a.secret.key</name>
         <description>Minio server secret key.</description>
         <value>minio123</value>
     </property>
     <property>
         <name>fs.s3a.aws.credentials.provider</name>
         <description>The credential provider type.</description>
         <value>org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider</value>
     </property>
     <property>
         <name>fs.s3a.endpoint</name>
         <description>Endpoint can either be an IP or a hostname, where Minio server is running .
However, the endpoint value cannot contain the http(s) prefix. E.g. 175.1.2.3:9000 is a valid endpoint.
</description>
         <value>10.42.5.215:9000</value>
     </property>
     <property>
         <name>fs.s3a.path.style.access</name>
         <description>Value has to be set to true.</description>
         <value>true</value>
     </property>
     <property>
         <name>dremio.s3.compat</name>
         <description>Value has to be set to true.</description>
         <value>true</value>
     </property>
     <property>
```

```
       <name>fs.s3a.connection.ssl.enabled</name>
       <description>Value can either be true or false, set to true to use SSL with a secure Minio
server.</description>
       <value>false</value>
     </property>
```

Steps for Deployment

Step 1: Clone Dremio's repository into a local machine

```
git clone https://github.com/dremio/dremio-cloud-tools.git
```

Step 2: Adjust the values.yml

```
vim dremio-cloud-tools/charts/dremio_v2/values.yaml
```

Step 3: Deploy Dremio via helm using the following commands:

```
helm install <release-name> dremio_v2 -f <file>
```

b. Integration

Apparently, connecting Dremio to the previous selected tech stack which is Minio with Delta format file is working nicely. Connecting Dremio with python client using ODBC/JDBC also works. However, the simplest tool that able to connect with Dremio is undoubtably Tableau.

Note: Please refer to an ***Appendix Section C*** for the example python ODBC/JDBC code snippet.

PowerBI also an interesting tool that has a strong potential for integration, but currently not in the scope of evaluation. As for Apache Superset, it is expected to be able integrated with Dremio with a help of SQLAlchemy. However, SQLAlchemy isn't designed to support Delta file format.

c. Performance

A result below are examples of Dremio's Graphic User Interface.

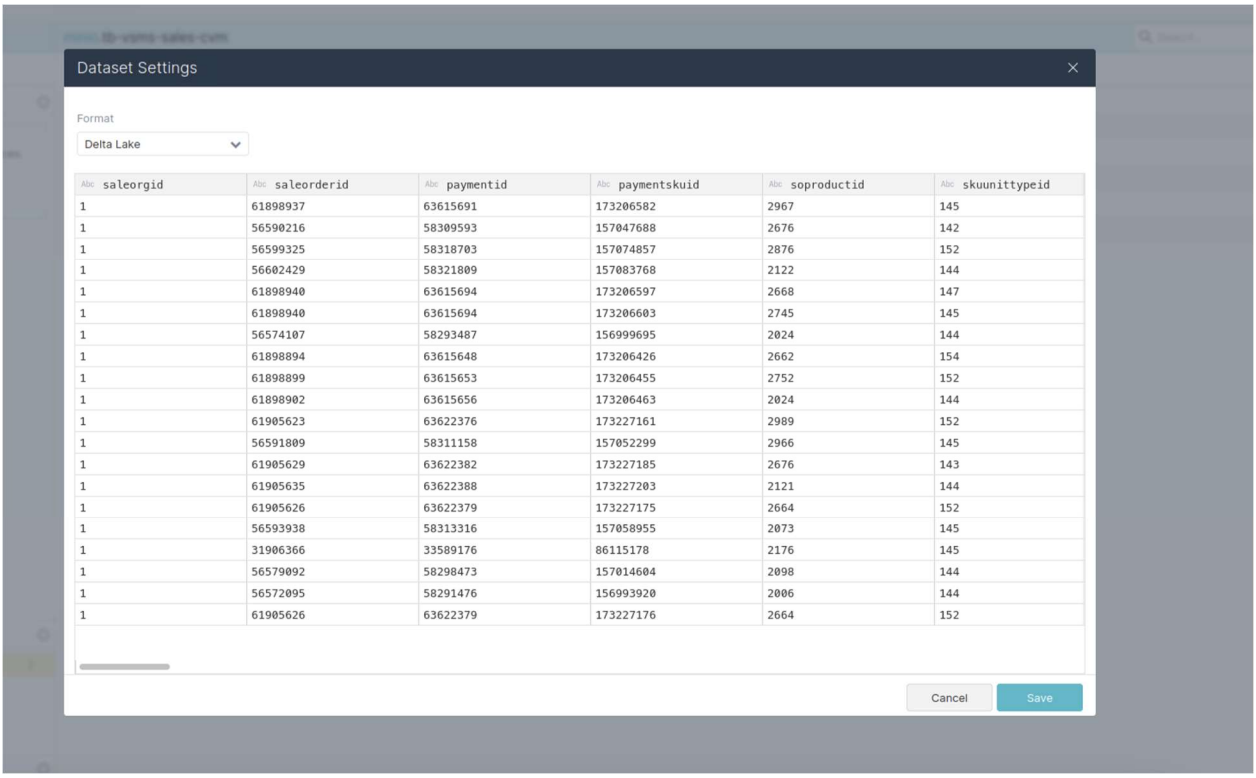


Figure 3.2.1.c.1: Retrieve Delta Table from Minio

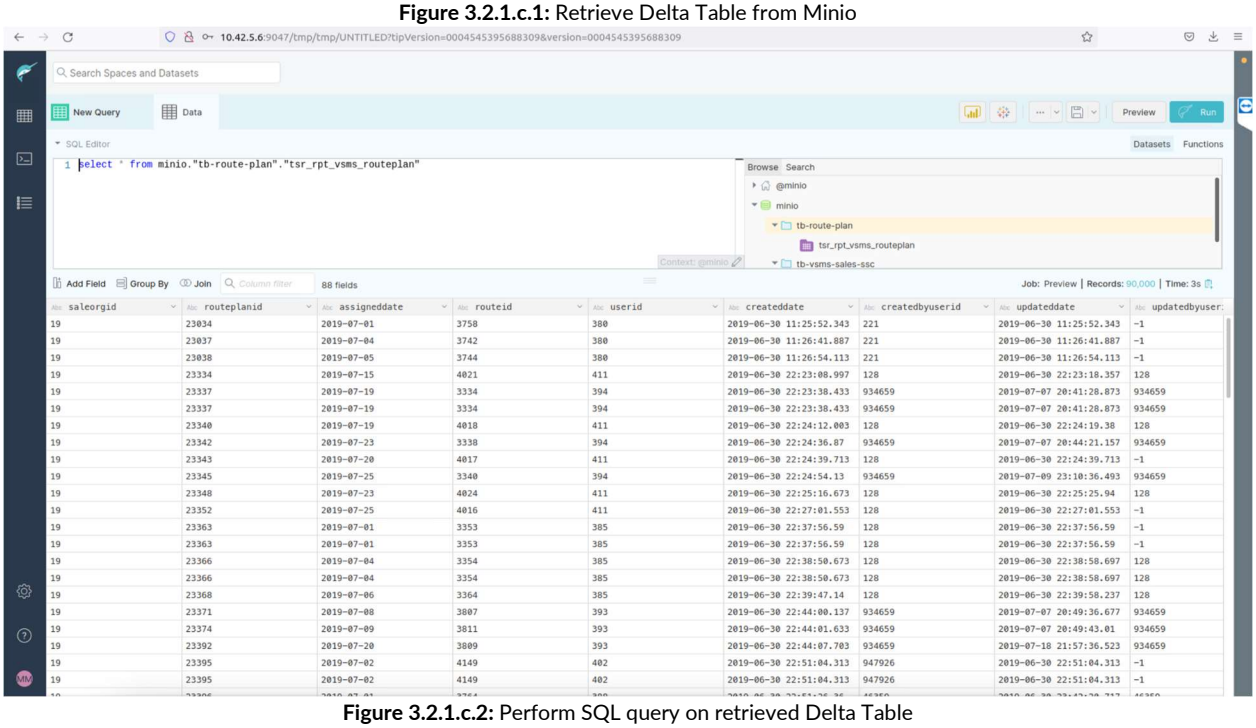


Figure 3.2.1.c.2: Perform SQL query on retrieved Delta Table

Note: Please refer to an ***Appendix Section D*** for more examples of Dremio's user interface.

Though, the SQL query result will be limited for the Dremio's graphic user interface if the result (data size) is too large, using JDBC/ODBC is a workaround for the given scenario.

The following is an example result of using **TSR_RPT_VSMS_ROUTEPLAN** query from the prior section on evaluating the Apache Spark.

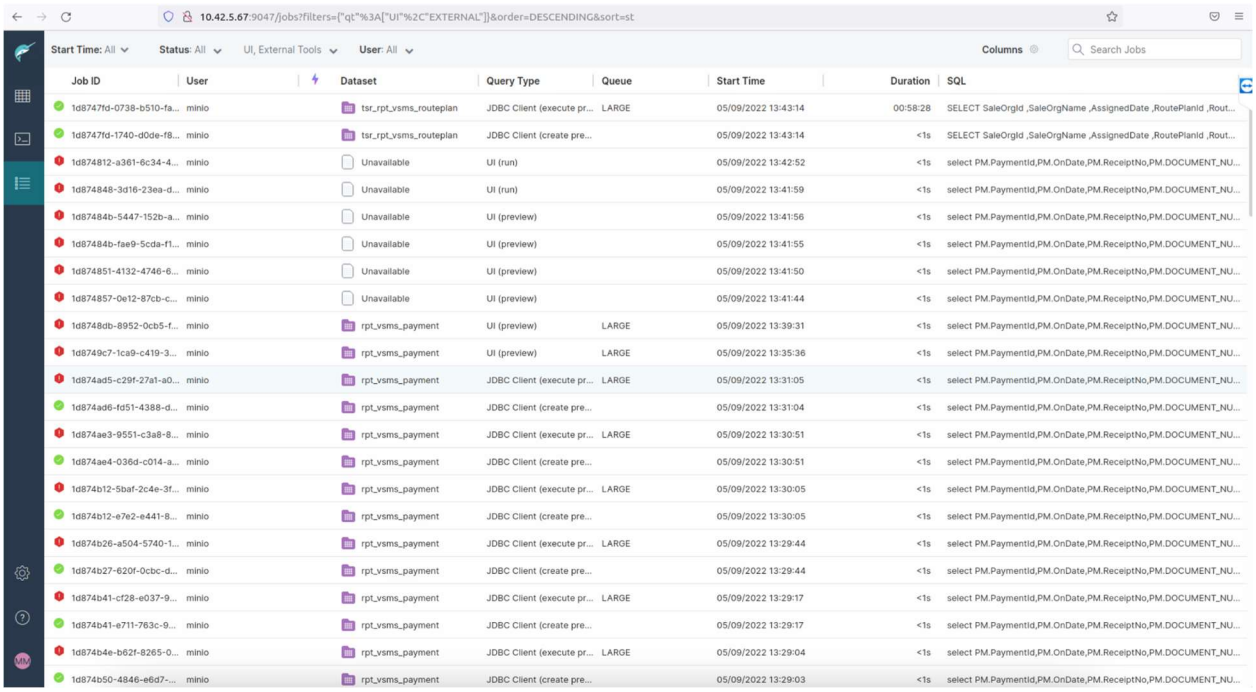


Figure 3.2.1.c.3: Result of SQL queries in Dremio

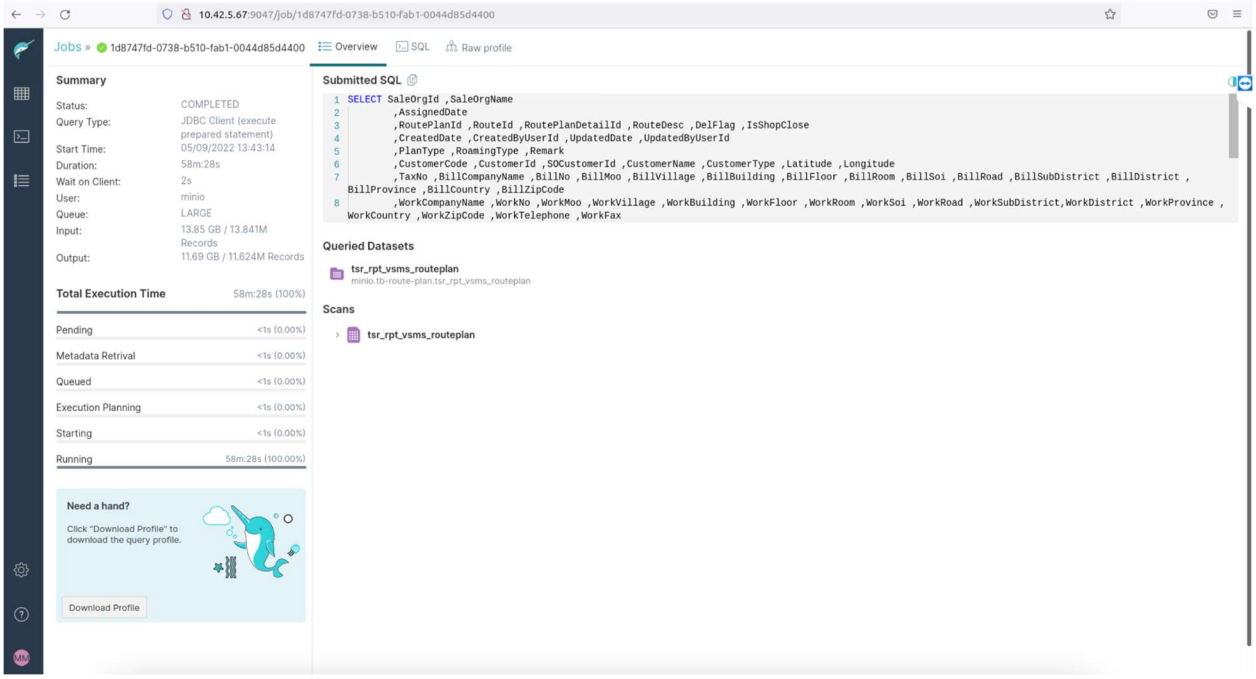


Figure 3.2.1.c.4: Detailed Result of SQL queries in Dremio

Distributed Minio (4 Nodes), Delta Lake		Python + Spark (Library: PySpark)	Python + ODBC (Library: pyodbc)	Python + JDBC (Library: jaydebeapi)
SQL Query Statement	Rows / Columns	Execution Time (hrs:mins:secs)	Execution Time (hrs:mins:secs)	Execution Time (hrs:mins:secs)
TSR_RPT_ VSMS_ ROUTEPLAN	11,624,334 / 75	00:05:23 (100%)	00:12:23 (234%)	00:58:28 (1114%)

Table 3.2.1.c.1: Comparison Table of using Python as a client on Spark from Data Source and ODBC/JDBC from Dremio

Clearly, the result is slower than the direct connection between client (PySpark) and the data source. The reason behind a slower performance is mainly due to several layers of tools (without Spark) have been applied, and complexity occurs along the way, in order to achieve this result.

Though, connecting Dremio with Python client using ODBC is still considered acceptable in term of performance (processing time) comparing to JDBC.

3.2.2 Trino

a. Feasibility

After several attempts, a quick simple deployment method on the Kubernetes for the proof-of-concept purpose that is selected by the author is via Helm Chart deployment as the followings:

```
helm repo add valeriano-manassero https://valeriano-manassero.github.io/helm-charts
helm search repo valeriano-manassero
helm install my-release valeriano-manassero/<chart>
```

b. Integration

Apparently, for a Trino to works with Minio and Deltalake, it requires Apache Hive as a bridge for an integration. Despite several attempts of trials and errors, the authors are still unable to deploy or adjust an existing Apache Hive solution to fit with the experimental environment.

Hence, due to the limited time frames and the tradeoff between pushing this evaluation through and continuing the evaluation on this area of work, building the solution using Apache Hive and Trino (or Presto) has been paused.

Despite a failure in integrating the Apache Hive into an experimental environment. The example of the existing working solution from harshavardhana repository on deploying Apache Hive and Presto together via Docker Compose [17] enabled the author to adapt the existing solution to demonstrate how the Delta table that is stored in the Minio can be integrated (created, connect) with Apache Hive.

Note: Please refers to the ***Appendix Section E*** for further details on connecting the Delta table in Minio with Apache Hive.

3.3 Data Source Aggregation Layer

3.3.1 Apache Superset

a. Feasibility

Apparently, there is a Helm Chart available for a deployment in Kubernetes provided by Apache Superset. However, a few steps are required for deploying an Apache Superset into an experimental environment as follows:

Step 1: Add Helm Chart Repository

```
helm repo add superset https://apache.github.io/superset
```

Step 2: Adjust values.yml According to User's Preference.

Note: for authors only change fill in openebs-hostpath to all the storage account tag.

Step 3: Helm install

```
helm upgrade --install --values my-values.yaml superset superset/superset
```

b. Integration

Minio and Dremio requires the help of SQLAlchemy to create a relational database table format which doesn't support delta lake either. Though, it works perfectly fine with the PostgreSQL database.

c. Performance

The followings are the example visualization by connecting Apache Superset with a PostgreSQL database.

Dashboards

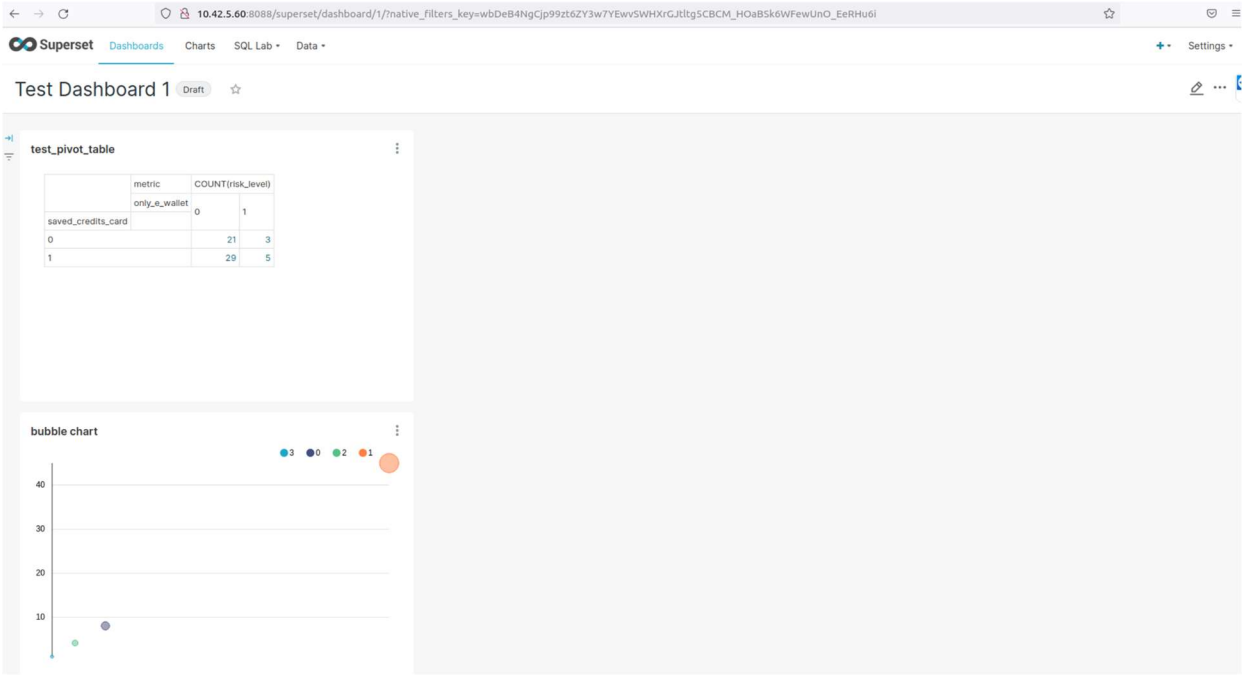


Figure 3.3.1.c.1: Apache Superset Dashboard (main page)

Chart

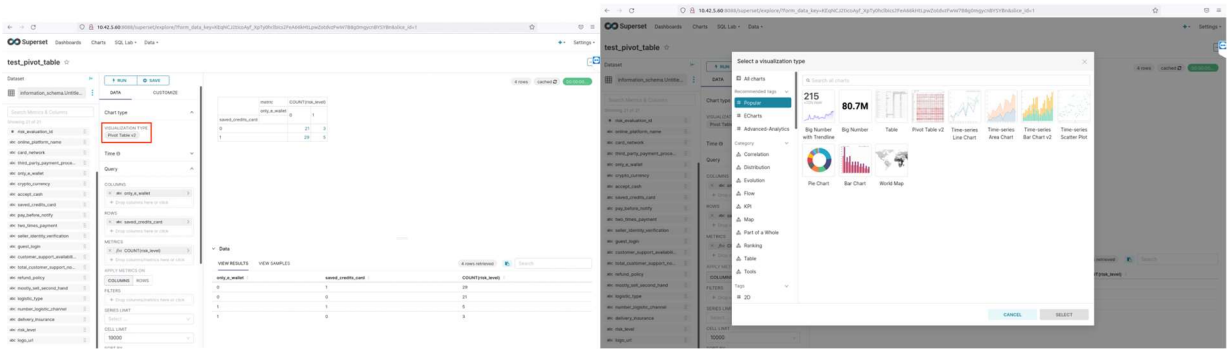


Figure 3.3.1.c.2: Apache Superset Chart

Note: Please refer to an ***Appendix Section F*** for more examples of Apache Superset's user interface.

3.3.2 Tableau

a. Feasibility

The product that the author has tried using a trial membership provided by Tableau are Tableau Desktop and Tableau Online. Tableau Desktop is only available for download in Windows and macOS versions. As for Tableau Online, it does not support internal network connections. Therefore, the data sources must be properly exposed to the internet before being able to create a connection with a Tableau Online.

b. Integration

Dremio: Tableau is able to connect to Dremio properly, and be able to present a Delta format data stored in Minio [21].

Minio: Unfortunately, there is no native connector available for Minio, it would be more complicated too as the data is stored in a Delta Lake format. However, for the workaround, there existed a use-case where users were writing a python script to retrieve the data from the Minio and published them on the Tableau server [18].

c. Performance

Results below are the visualization provided by Tableau connecting with a Dremio as a data source.

Data Source(s)

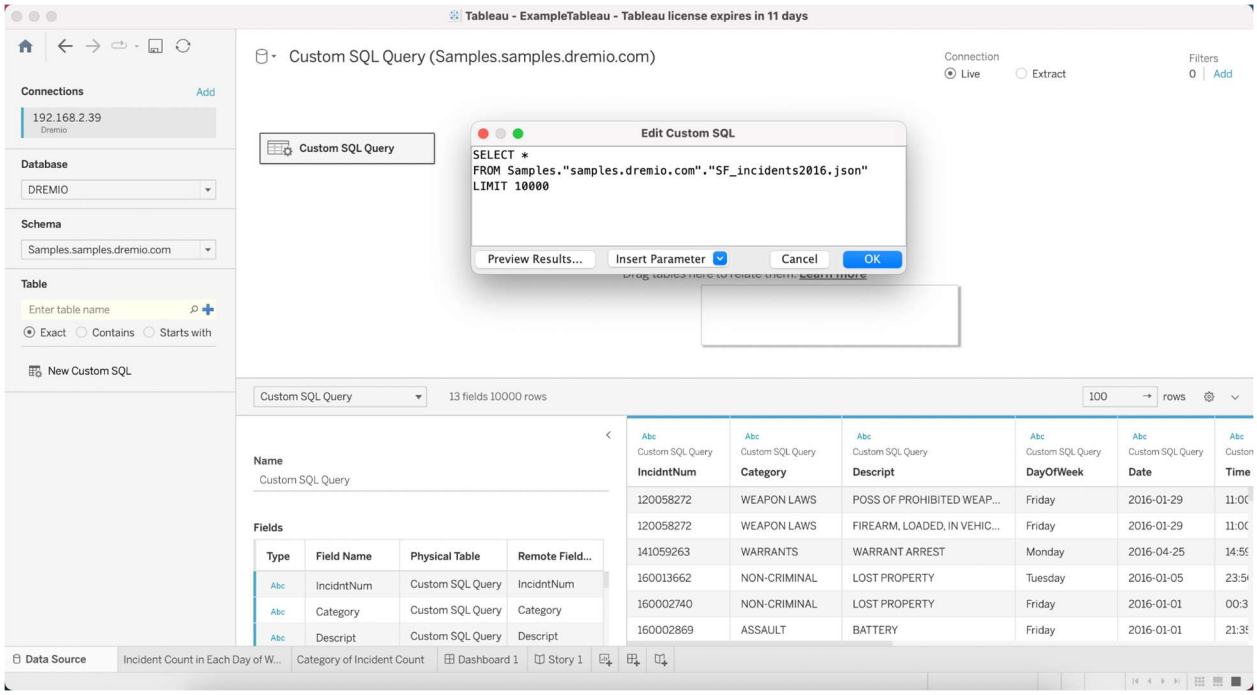


Figure 3.3.2.c.1: Tableau Interface for Connecting to Data Source

Sheet (Chart)

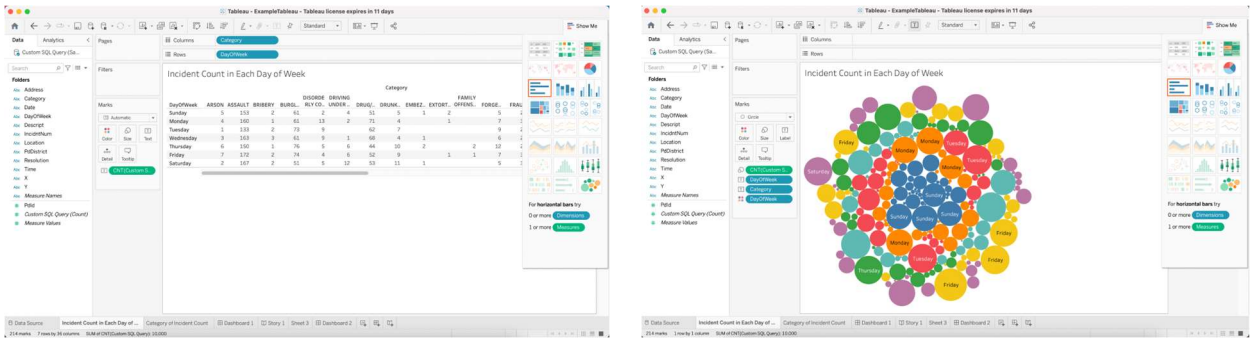


Figure 3.3.2.c.2 / 3.3.2.c.3: Tableau Visualization as Table / Bubble Chart

Dashboard (Combination of Sheets)

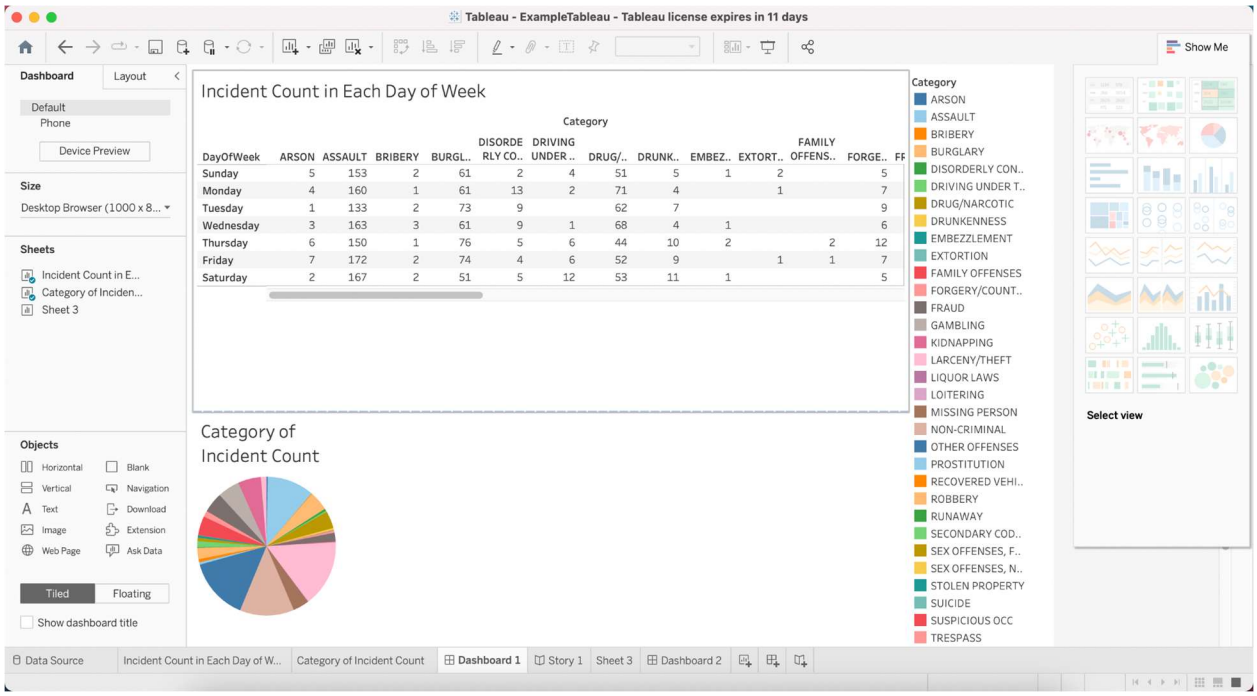


Figure 3.3.2.c.4: Tableau Visualization as Dashboard

Story (Combination of Sheets and Dashboards)

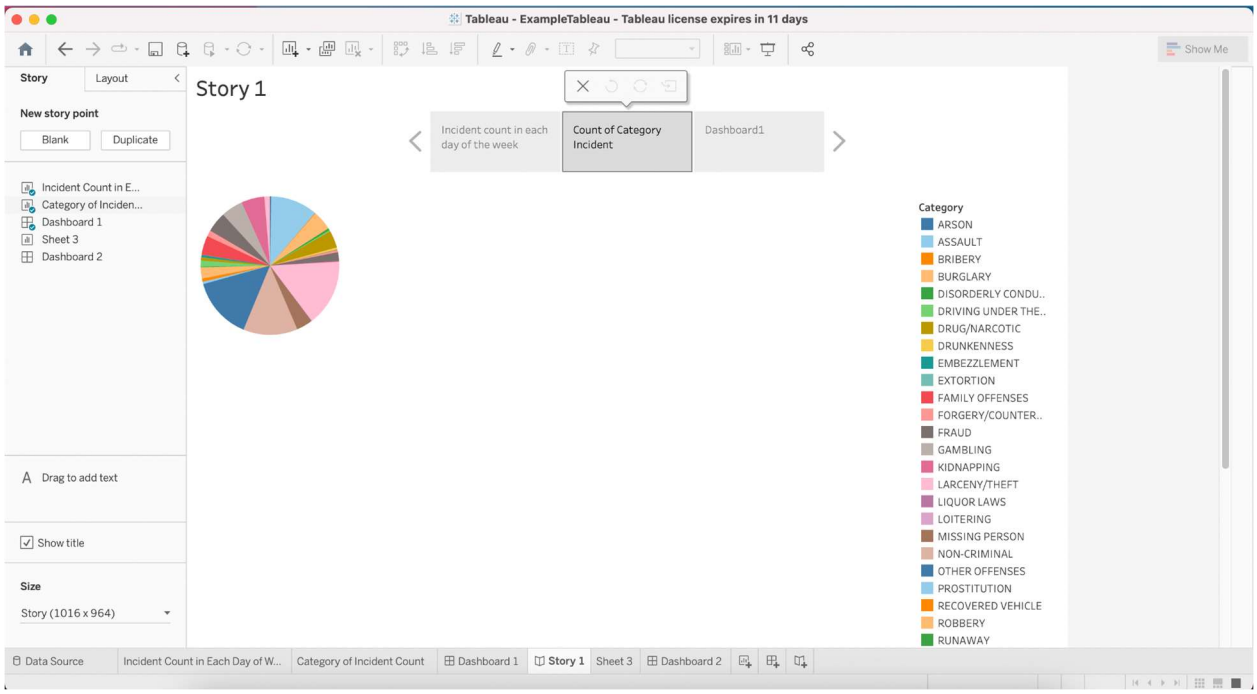


Figure 3.3.2.c.5: Tableau Visualization as Story

4. Finding / Insight

Providing an end-to-end exploration of configuring and deploying evaluated tools, testing their integration, and exploring their result. There are many points to be taken for selecting these tools and their combinations.

An author will be providing the comparison table which listed all the evaluation criteria that the author is aware of for the tools and insight based on the **experience of the author** regarding the tools.

Comparison Table: Data Source Aggregation Layer

	Dremio	***Apache Hive and Trino***
Performance	• **read** from data source only • **has** graphic user interface • limited size of the data if query from user interface • requires ODBC/JDBC connecting with the client for large size of data • query using clients (python+JDBC) is approximately 10 times slower than using SparkSQL directly on the data source (Minio)	• more flexible use-cases in an **insert**, **update**, **delete** table • **no** graphic user interface • client is a command-line interface and JDBC

Configuration Complexity	• simple and straight forward • little tricky on configuring the connection with Minio	• setup Trino for Kubernetes using Helm Chart is simple and straight forward • unable to setup Apache Hive
Integration Complexity	• can be done easily through user interface	• unable to integrate any existing Apache Hive solution with Trino
Learning Curve	• easy to moderate • user-friendly • good documents • possible for not strong (or non-) technical background users	• moderate to hard • given it requires a combination of Apache Hive and Trino, there is more information to consume • Apache Hive documents are confusing • as the tools existed for quite a period of time, there is more information to screen (old or up-to-date)
Community	• certain group of people is aware (or using) it • online course and tutorial • Dremio University [19]	• huge, everyone in the field knows (is aware of) and use it • online course and tutorial

Table 4.1: Comparison Table for Data Source Aggregation Layer

Through the exploration, it has been learned that by adding an extra layer (in this case, data source aggregation layer) to the system (more tools) will definitely slow down the processing time and increases more complexity to the system in terms of configurations, integrations, and monitoring. Hence, before adding this additional layer the factors that are being suggested to consider are size of the queried data on a daily basis, correlation of the data sources, and manageability.

Size of the queried data on a daily basis is proportional to the processing time and the larger it is the slower the processing time can be, especially when many layers of tools are being integrated into a solution.

Correlation of the Data Sources, if the data from different sources is correlated and usually required (by a use-case) for users to query them out and map them together for gaining insight then it might be necessary for users to see them together using the data source aggregator tool.

Manageability, if current data architecture solution is still manageable and be able to grow in a systematically (manageably) way because if it is manageable then the data aggregation layer might not be necessary for the system.

Hence, for the data source aggregation layer, a concern for the author will depend on the size of the queried data, correlation of the data sources, and internal data management method(s) of the client which can be briefly illustrated in the example of action metrics below:

Size of Queried Data	***Correlation of the Data Sources***	***Manageable***	***Adding Data Aggregation Layer***
small	yes	yes	yes
small	yes	no	yes
small	no	yes	depends
small	no	no	yes

large	yes	yes	no
large	yes	no	yes
large	no	yes	no
large	no	no	yes

Table 4.2: Example Evaluation Rubrics for adapting Data Source Aggregation Layer into a solution

Data Source Aggregation Layer can also be used for a proof-of-concept purpose as well. Given the new type of data source has been introduced (e.g., Minio) and needed to be tested for its performance, both old and new data sources could be connected to the data source aggregator tool(s). The client then could be able directly queries from one of these data sources with just a minor change in the query statement.

Comparison Table: Data Analytical and Visualization Layer

	Tableau	***Apache Superset***
Performance	• not free	• free
Configuration Complexity	• simple and straightforward	• complicated, especially setup for Kubernetes
Integration Complexity	• simple and straightforward, just follow through workaround URL provided in the error message,	• complicated integration, especially with Deltalake file format

Learning Curve	• easy to moderate • user-friendly • as the tools existed for quite a period of time, there is more information to consume	• easy to moderate • user-friendly • good documents
Community	• huge, widely uses in the industry • online course and tutorial • university's lecture	• certain group of people is aware (or using) it • online course and tutorial

Table 4.3: Comparison Table for Data Analytical and Visualization Layer

Tableau is a better choice in term of simplicity in deployments, configurations, integration, and maintenance. It is also more convenient to learn and find an employee who's familiar with Tableau in the job market. The only concern for the author will be regarding the price of the product, as it is a commercial product.

5. Example Use-Cases

Finally, after all the tech stack (tools) evaluation for the enterprise data architecture an author decided to create an example use-case that will give better exposure to a selected (recommended) tech stack from the author.

5.1 Example Tech Stack

- Kubernetes (Rancher, OpenEBS)
- Minio
- Apache Spark (Version 3.1.2)
- Delta Lake (Version 3.0.0)
- Dremio
- Streamlit

Example Setup Video v.1 using Google Cloud VM

5.2 Example Functions

1. Insert, View, Update, Merge, and Delete data stored in Minio in a Delta format using PySpark

2. Retrieve a data stored in Minio in a Delta format using SQL queries through Python clients which integrate with the following tools:
    a. Apache Spark (PySpark)
    b. ODBC / Dremio

5.3 Example Design

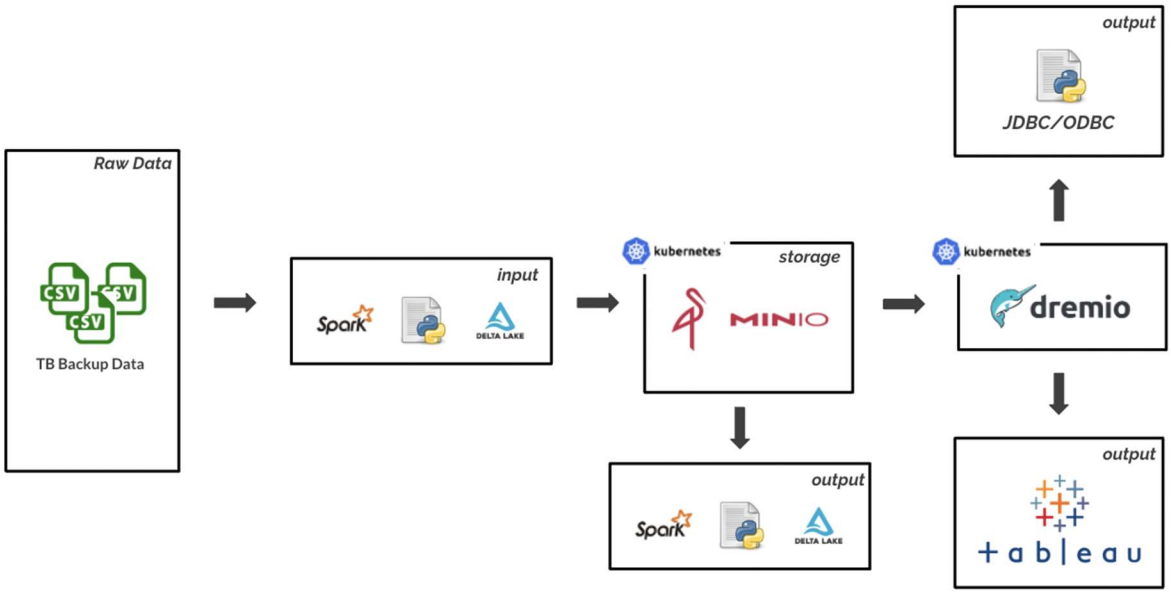


Figure 5.3.1: Example design for further use-case implementation

Raw data, in this case is stored in CSV, will be transform into a Delta format and stored in a Minio. It is done by python script which integrated with Apache Spark with a help of 2 external libraries which are PySpark and Delta-Spark. The data that is stored in the Minio, in Delta format, can be connected to Dremio or retrieved by python script as a client. As Minio is connected to Dremio as a data source, it also enables to user to retrieve the data through python script using JDBC/ODBC or Tableau.

Though, this example use-case section connecting Dremio to Tableau or JDBC as a client will not be demonstrated. Please refers the ***Appendix Section C*** and ***Section 3.3.2*** for the example for these 2 cases.

5.4 Example Use-Cases

5.4.1 Near Real-Time Dashboard

The application (website) is implemented in Python using a Streamlit. Data is stored in Minio with Delta format through a Python script which calls a local Spark though PySpark library. The application is able to auto refresh the widgets (components) every specific period of time, currently is set to every 5 seconds, to get a new input data from the data source.

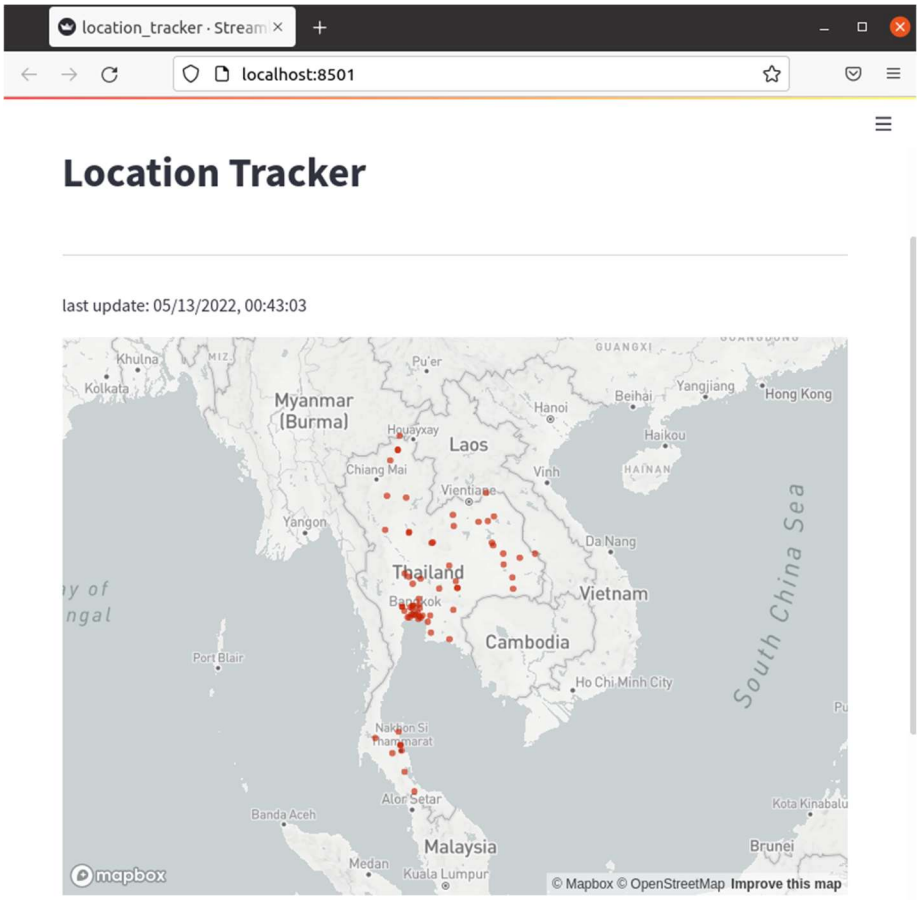


Figure 5.4.1.1: Location Tracker Web Application (demo video)

5.4.2 Data Pipeline

All of the script is implemented in Python. All the script is currently controlled by python executor script. These scripts are expected to be able to integrate further with automation tools though. The data source generator script will generate the data frame of a user and user_level data then stored in Minio in a delta format with the help of PySpark library. The raw data then will be pulled and processed through the remaining script based on the sequence in the executor script and finalize the data for the actual use.

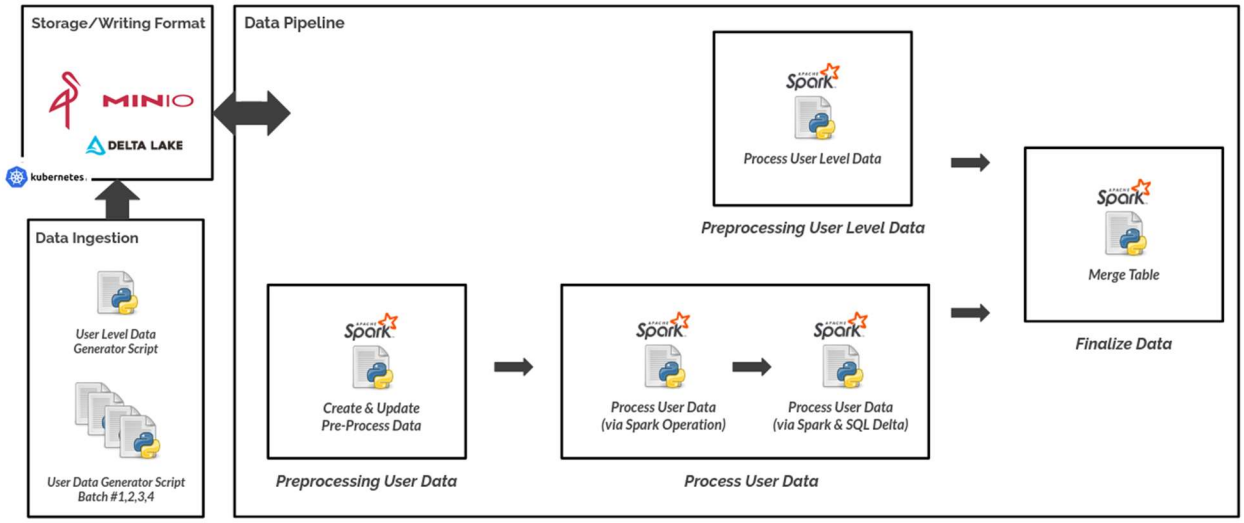


Figure 5.4.2.1: A Design for Example Data Pipeline

Note: For a further details of an example overview implementation and result please refers to the ***Appendix Section G***

5.4.3 Analytic Dashboard

Two analytics dashboard applications (website) have been implemented in Python using Streamlit and Minio with Delta format as a data source. However, for Sales Data Analytics Dashboard (using VSMS Sale CVM data) is implemented to retrieve the data from Dremio via ODBC as client. As for Location Analytics Dashboard, it uses SparkSQL operation direct query from the data sources.

Sales Data Analytics Dashboard (VSMS Sale CVM)

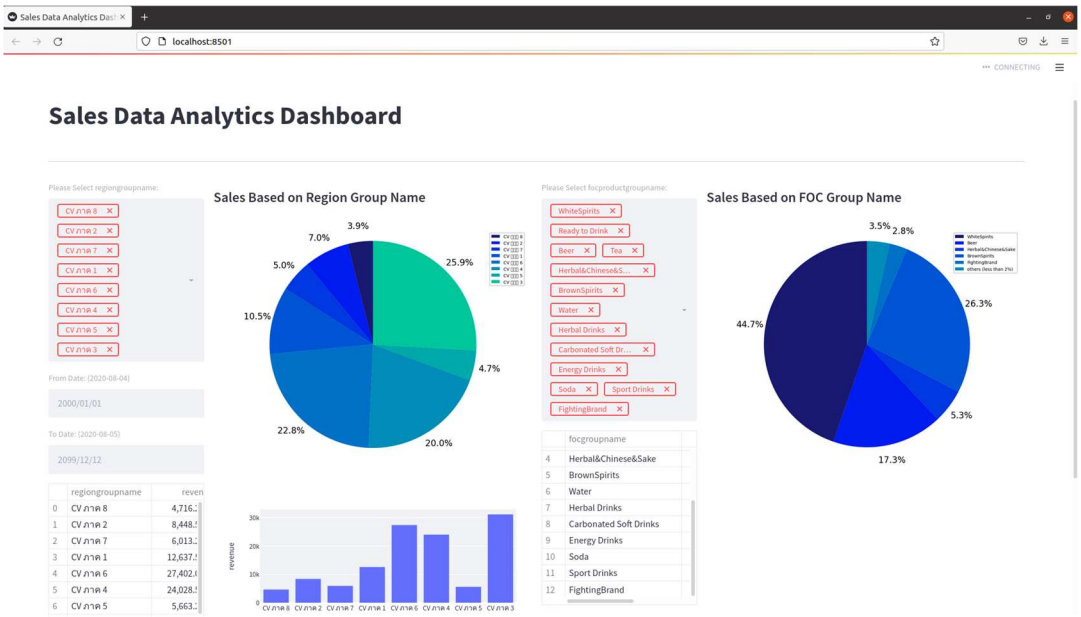


Figure 5.4.3.1: Sales Data Analytics Dashboard Web Application (demo video)

Location Analytics Dashboard (TSR Routeplan)

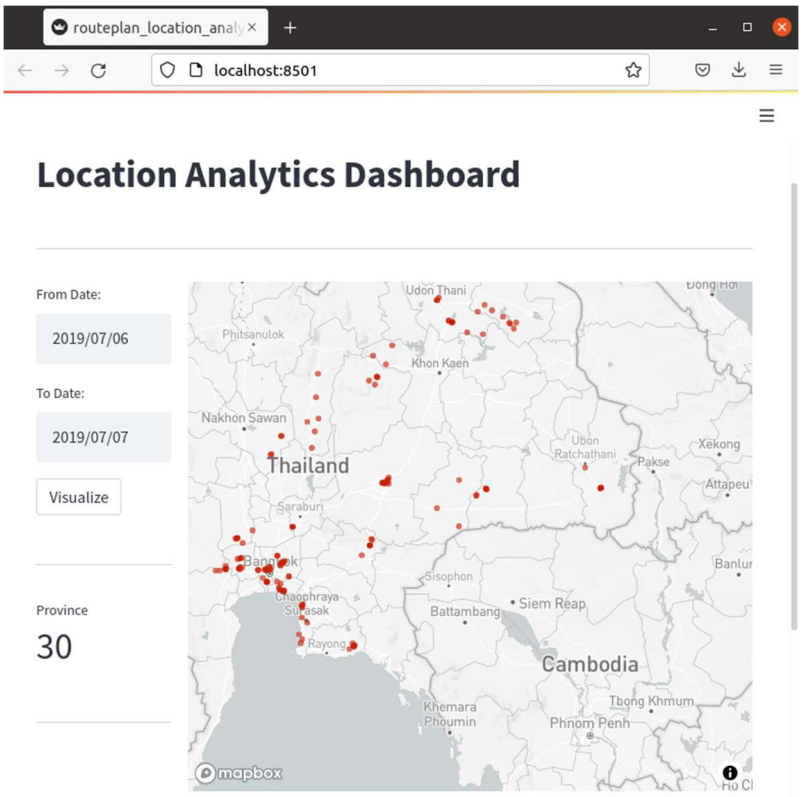


Figure 5.4.3.2: Route Plan Data Analytics Dashboard Web Application

6. Conclusion

The Data Source Aggregation layer is highly dependent on the data management methods and the size of the data given by a company. It is not necessary and might slower the processing time and create more complexity for the users if it's not handled properly.

As for the Data Analytical and Visualization layer, if the company only sees technology from the perspective of cost, it is recommended to use existing commercial tools. Free tools required more effort from (technically strong) employees to support them, and they still need further proof of their capabilities in every aspect of efficiency. Instead, these efforts can be spent on improving various other areas such as infrastructure, big data architecture, security of existing system design, or automation for the end-to-end digital process.

References

[1] Ben Lorica, Michael Armbrust, Ali Ghodsi, Reynold Xin, Matei Zaharia, "What Is a Lakehouse?," databricks, 30 01 2020. [Online]. Available: https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html. [Accessed 15 12 2021].

[2] ยุทธการ อ่างมณีกุล , "what-is-data-lakehouse," sharebyheang, 03 02 2021. [Online]. Available: https://sharebyheang.com/what-is-data-lakehouse/. [Accessed 15 12 2021].

[3] Minio, "min.io," MINIO, [Online]. Available: https://min.io/. [Accessed 15 12 2021].

[4] Apache Spark, "spark.apache.org," Apache Spark, [Online]. Available: https://spark.apache.org/. [Accessed 15 12 2021].

[5] databricks, "docs.delta.io/latest/delta-intro," databricks, 2020. [Online]. Available: https://docs.delta.io/latest/delta-intro.html. [Accessed 15 12 2021].

[6] Dremio, "dremio," Dremio, 2015. [Online]. Available: https://www.dremio.com/platform/. [Accessed 15 05 2022].

[7] Dremio, "dremio," Dremio, 2015. [Online]. Available: https://www.dremio.com/platform/. [Accessed 15 05 2022].

[8] A. S. Foundation, "hive.apache," Apache Software Foundation, 01 10 2010. [Online]. Available: https://hive.apache.org/. [Accessed 15 05 2022].

[9] T. S. Foundation, "trino," Trino Software Foundation, 2012. [Online]. Available: https://trino.io/. [Accessed 15 05 2022].

[10] D. Taylor, "guru99," guru99, 16 04 2022. [Online]. Available: https://www.guru99.com/what-is-tableau.html. [Accessed 15 05 2022].

[11] A. S. Foundation, "superset.apache," Apache Software Foundation, 2016. [Online]. Available: https://superset.apache.org/. [Accessed 15 05 2022].

[12] T. A. S. Foundation, "spark.apache," The Apache Software Foundation, [Online]. Available: https://spark.apache.org/docs/latest/running-on-kubernetes.html. [Accessed 15 05 2022].

[13] J. Kremser, "databricks," Databricks, 22 10 2019. [Online]. Available: https://databricks.com/session_eu19/spark-operator-deploy-manage-and-monitor-spark-clusters-on-kubernetes. [Accessed 15 05 2022].

[14] miguelaeh, "github," Bitnami, 14 05 2022. [Online]. Available: https://github.com/bitnami/charts/tree/master/bitnami/spark/#installing-the-chart. [Accessed 15 05 2022].

[15] NNK, "sparkbyexamples," sparkbyexamples, [Online]. Available: https://sparkbyexamples.com/spark/spark-write-dataframe-single-csv-file/. [Accessed 15 05 2022].

[16] ryantse, "github," Dremio, 13 05 2022. [Online]. Available: https://github.com/dremio/dremio-cloud-tools/tree/master/charts/dremio_v2. [Accessed 15 05 2022].

[17] harshavardhana, "github," 06 07 2021. [Online]. Available: https://github.com/minio/presto-minio. [Accessed 15 05 2022].

[18] V. Sen, "community.tableau," 30 03 2020. [Online]. Available: https://community.tableau.com/s/question/0D54T00000C6o7mSAB/how-to-connect-tableau-and-minio-server. [Accessed 15 05 2022].

[19] Dremio, "university.dremio," Dremio, 2021. [Online]. Available: https://university.dremio.com/. [Accessed 15 05 2022].

[20] K. Chamnivikaipong, "Research & Development: Enterprise Architecture," CMKL, Bangkok, 2021.

[21] Dremio, "dremio," Dremio, 2021. [Online]. Available: https://www.dremio.com/blog/tableau-and-dremio-introduce-native-connector-in-tableau-to-streamline-bi-on-lakehouses/. [Accessed 15 05 2022].

GitHub Repository

https://github.com/CMKL-ThaiBev/TB-Enterprise-Arch.git (Examples for Users)

Appendices

A. Setup Kubernetes Cluster (Using Rancher and OpenEBS)

A.1 Hardware Specification

There are 6 machines connected together with a LAN network with the following specifications:

- CPU - Intel(R) Core(TM) i5-7500T CPU @ 2.70 GHz
- RAM - 16Gib

For a detailed hardware specification, please follow this link.

A.2 Kubernetes Cluster Design

A cluster is designed to deploy into the provided machines as the following:

A.3 Walkthrough Setup Using Google Cloud Virtual Machine

Video: https://youtu.be/87isSWK2-og

A.4 Linux (Ubuntu 20.04) and Network Setup

Step 1: Install Ubuntu (detailed walkthrough)

Step 2: Ensure that SSH is enabled (in case of error)

Step 3: Create SSH (private-public) key for root access to the machine

```
ssh-keygen
```

Step 4: Login as root of each machine and save the public key that previously generated

```
sudo -i
cd ~/.ssh
vim authorized_keys
```

A.5 Install Docker on Every Machine

Reference: https://phoenixnap.com/kb/how-to-install-docker-on-ubuntu-18-04

Step 1: Update Software Repository

```
sudo apt-get update
```

Step 2: Uninstall Old Version of Docker

```
sudo apt-get remove docker docker-engine docker.io
```

Step 3: Install Docker on Ubuntu

```
sudo apt install docker.io
```

A.6 Deploying Kubernetes Cluster (Using Rancher)

Reference: https://rancher.com/docs/rke/latest/en/installation/

Step 1: Installing Rancher on your workspace (your machine)

```
https://rancher.com/docs/rke/latest/en/installation/
```

Step 2: Create Cluster.yml

```
Necessary information to fill in
  • Number of the machines in the cluster
  • Machine IP Address
  • Path to SSH key
  • Port 22 (default)
  • Machine Roles (Controller, Worker, ETCD)
  • Network: Flannel (author is using)
```

Step 3: Adding OpenEBS Local Hostpath Configuration into Cluster.yml (reference)

```
services:
  kubelet:
    extra_binds:
      - /var/openebs/local:/var/openebs/local
```

Step 4: Deploying Cluster using Cluster.yml

```
rke up
```

Step 5: Merge Kubernetes config file (reference)

```
# Merge the two config files together into a new config file
KUBECONFIG=~/.kube/config:/path/to/new/config kubectl config view --flatten >
/tmp/config
```

B. SQL Queries Comparison

B.1 TSR_RPT_VSMS_ROUTEPLAN

ThaiBev Version:

```
SELECT

[SaleOrgId] ,[SaleOrgName], [AssignedDate] ,format(AssignedDate,'yyyyMM') as
[AssignedYearmonth], [RoutePlanId] ,[RouteId] ,[RoutePlanDetailId] ,[RouteDesc]
,[DelFlag] ,[IsShopClose], [CreatedDate] ,[CreatedByUserId] ,[UpdatedDate]
,[UpdatedByUserId],[PlanType] ,[RoamingType] ,[Remark], [CustomerCode]
,[CustomerId] ,[SOCustomerId] ,[CustomerName] ,[CustomerType] ,[Latitude]
,[Longitude], [TaxNo] ,[BillCompanyName] ,[BillNo] ,[BillMoo] ,[BillVillage]
,[BillBuilding] ,[BillFloor] ,[BillRoom] ,[BillSoi] ,[BillRoad] ,[BillSubDistrict]
,[BillDistrict] ,[BillProvince] ,[BillCountry] ,[BillZipCode], [WorkCompanyName]
,[WorkNo] ,[WorkMoo] ,[WorkVillage] ,[WorkBuilding] ,[WorkFloor] ,[WorkRoom]
,[WorkSoi] ,[WorkRoad] ,[WorkSubDistrict],[WorkDistrict] ,[WorkProvince]
,[WorkCountry] ,[WorkZipCode] ,[WorkTelephone] ,[WorkFax], [HierarchyTerritoryId],
[ChannelId] ,[ChannelSAPCode] ,[ChannelDesc], [RegionGroupId] ,[RegionGroupCode]
,[RegionGroupName], [AreaGroupId] ,[AreaGroupCode] ,[AreaGroupName],
[BranchGroupId] ,[BranchGroupCode] ,[BranchGroupName], [BranchId] ,[BranchCode]
,[BranchName], [SalesTeamId] ,[SalesUnitDesc] ,[TerritoryId] ,[TerritoryDesc]
,[Tag]

FROM [OTC_DB_Report].[dbo].[TSR_View_RPT_VSMS_RoutePlan]

WHERE DelFlag = 0 AND [IsShopClose] = 0 AND AssignedDate
BETWEEN '2021-10-01'and '2022-01-31 and SaleOrgId in (1,2)
```

CMKL Version:

```
SELECT

SaleOrgId, SaleOrgName, AssignedDate, RoutePlanId, RouteId, RoutePlanDetailId,
RouteDesc, DelFlag, IsShopClose, CreatedDate, CreatedByUserId, UpdatedDate,
UpdatedByUserId, PlanType, RoamingType, Remark, CustomerCode, CustomerId ,
SOCustomerId, CustomerName, CustomerType, Latitude, Longitude, TaxNo,
BillCompanyName, BillNo, BillMoo, BillVillage, BillBuilding, BillFloor, BillRoom ,
BillSoi, BillRoad, BillSubDistrict, BillDistrict, BillProvince, BillCountry ,
BillZipCode, WorkCompanyName, WorkNo, WorkMoo, WorkVillage, WorkBuilding,
WorkFloor, WorkRoom, WorkSoi, WorkRoad, WorkSubDistrict, WorkDistrict,
WorkProvince, WorkCountry, WorkZipCode, WorkTelephone, WorkFax,
HierarchyTerritoryId, ChannelId, ChannelSAPCode, ChannelDesc, RegionGroupId,
RegionGroupCode, RegionGroupName, AreaGroupId, AreaGroupCode, AreaGroupName,
BranchGroupId, BranchGroupCode, BranchGroupName, BranchId, BranchCode, BranchName,
SalesTeamId, SalesUnitDesc, TerritoryId, TerritoryDesc, Tag

FROM delta.`s3a://tb-route-plan/tsr_rpt_vsms_routeplan`

WHERE DelFlag = 0 AND IsShopClose = 0 AND AssignedDate
BETWEEN '2021-10-01' AND '2022-01-31' and (SaleOrgId = 1 or SaleOrgId = 2)
```

Deviation from ThaiBev version:

- No `format(AssignedDate,'yyyyMM') as [AssignedYearmonth]`
- No Square Bracket ([ ])
- Changes from `SaleOrgId in (1,2)` to `(SaleOrgId = 1 or SaleOrgId = 2)`

B.2 RPT_VSMS_SALES_SSC

ThaiBev Version:

```
SELECT

PM.PaymentId,PM.OnDate,PM.ReceiptNo,PM.DOCUMENT_NUMBER,SO.SaleOrderId,PM.SOCompanyBranchId,S
O.latitude,SO.longitude
,SO.AduserId AS UserName
,so.ShelfId,so.SOCustomerId,PMSKU.SOProductId,P.ProductCode,P.ProductName,SO.SaleType,PMSKU.
ItemType,PMSKU.SKUUnitTypeId,PMSKU.Quantity
PaymentQuantity,PMSKU.Price,PMSKU.Amount,PMSKU.vat,PMSKU.VatRate,PMSKU.NormalDiscount,PMSKU.
SpecialDiscount,PMSKU.ManualDiscount,PMSKU.SalesDiscount,PMSKU.PromotionDiscount
--,PM.InterfaceStatus
,convert(tinyint, case when IsTransfered=1 and PM.InterfaceStatus=0 then 1 else
PM.InterfaceStatus end) InterfaceStatus
,PM.CreatedDate,PM.UpdatedDate
,PM.CompanyCode --,PM.[SOCompanyId] CompanyCode
,hst.BranchGroupCode AS BranchCode --,convert(nvarchar(30), PM.BranchCode) BranchCode
,convert(date,PM.OnDate) PMDate
,so.UserId,PMSKU.RefPaymentSKUId,SO.SalesTeamId SaleOrderSalesTeamId,so.Status
SaleOrderStatus,PM.IsCancel,SO.StorageCode,DO.DeliveryOnDate DO_DATE
,PM.PlantCode --,socompanybranch.PlantCode
,PMSKU.PaymentSKUId

FROM RPT_VSMS_Payment_SSC PM WITH (nolock)
LEFT JOIN RPT_VSMS_PaymentSKU_SSC PMSKU WITH (nolock) ON PMSKU.PaymentId=PM.PaymentId and
PMSKU.SaleOrgId=2

LEFT JOIN RPT_VSMS_SaleOrder_SSC SO WITH (nolock) on SO.SaleOrderId=PM.SaleOrderId and
PM.DelFlag=0 and SO.SaleOrgId=2
LEFT JOIN RPT_VSMS_Delivery_SSC DO WITH (nolock) ON DO.DeliveryId=PM.DeliveryId and
DO.PaymentId = PM.PaymentId and Isnull(DO.DeliveryPlanDetailDelFlag, 0)=0 and DO.SaleOrgId=2
LEFT JOIN RPT_VSMS_HierarchySalesTeam hst WITH (nolock) on hst.SalesteamId=PM.SalesTeamId
and hst.SaleOrgId=2
LEFT JOIN RPT_VSMS_Product P WITH (nolock) on P.SOProductId = PMSKU.SOProductId and
P.SaleOrgId=2


WHERE cast(PM.OnDate as date) between '2021-10-01' and '2022-01-31'
and so.Status in (3,4,6) and PM.SaleOrgId=2
```

CMKL Version:

```
SELECT

PM.PaymentId, PM.OnDate, PM.ReceiptNo, PM.DOCUMENT_NUMBER,
SO.SaleOrderId,PM.SOCompanyBranchId, SO.latitude, SO.longitude, SO.AduserId AS UserName,
so.ShelfId, so.SOCustomerId, PMSKU.SOProductId, P.ProductCode, P.ProductName, SO.SaleType,
PMSKU.ItemType, PMSKU.SKUUnitTypeId, PMSKU.Quantity PaymentQuantity, PMSKU.Price,
PMSKU.Amount, PMSKU.vat, PMSKU.VatRate, PMSKU.NormalDiscount, PMSKU.SpecialDiscount,
PMSKU.ManualDiscount, PMSKU.SalesDiscount, PMSKU.PromotionDiscount, PM.CreatedDate,
PM.UpdatedDate, PM.CompanyCode, hst.BranchGroupCode AS BranchCode, so.UserId,
PMSKU.RefPaymentSKUId, SO.SalesTeamId SaleOrderSalesTeamId, so.Status SaleOrderStatus,
PM.IsCancel, SO.StorageCode, DO.DeliveryOnDate DO_DATE, PM.PlantCode , PMSKU.PaymentSKUId


FROM delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_payment_ssc` PM
LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_paymentsku_ssc` PMSKU ON
PMSKU.PaymentId=PM.PaymentId and PMSKU.SaleOrgId=2
LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_saleorder_ssc` SO ON
SO.SaleOrderId=PM.SaleOrderId AND PM.DelFlag=0 AND SO.SaleOrgId=2
LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_delivery_ssc` DO ON
DO.DeliveryId=PM.DeliveryId AND DO.PaymentId = PM.PaymentId AND DO.SaleOrgId=2
LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_hierarchysalesteam` hst ON
hst.SalesteamId=PM.SalesTeamId AND hst.SaleOrgId=2
LEFT JOIN delta.`s3a://tb-vsms-sales-ssc/rpt_vsms_product` P ON P.SOProductId =
PMSKU.SOProductId AND P.SaleOrgId=2


WHERE CAST(PM.OnDate AS date) BETWEEN '2021-10-01' AND '2022-01-31'
AND (so.Status = 3 OR so.Status = 4 or so.Status = 6) AND PM.SaleOrgId=2
```

Deviation from ThaiBev version:

- Remove `--,PM.InterfaceStatus`
- Remove `,convert(tinyint, case when IsTransfered=1 and PM.InterfaceStatus=0 then 1 else PM.InterfaceStatus end) InterfaceStatus`
- Remove `--,PM.[SOCompanyId] CompanyCode`
- Remove `--,convert(nvarchar(30), PM.BranchCode) BranchCode`
- Remove `convert(date,PM.OnDate) PMDate`
- Remove `--,socompanybranch.PlantCode`
- Remove `Isnull(DO.DeliveryPlanDetailDelFlag, 0)=0 and`
- Remove `WITH (nolock)`
- Changes from `so.Status in (3,4,6)` to `(so.Status = 3 OR so.Status = 4 or so.Status = 6)`

B.3 RPT_VSMS_SALES_CVM

ThaiBev Version:

```
SELECT


PM.PaymentId,PM.OnDate,PM.ReceiptNo,PM.DOCUMENT_NUMBER,SO.SaleOrderId,PM.SOCompanyBranchId,S
O.latitude,SO.longitude,SO.AduserId AS
UserName,so.ShelfId,so.SOCustomerId,PMSKU.SOProductId,P.ProductCode,P.ProductName,SO.SaleTyp
e,PMSKU.ItemType,PMSKU.SKUUnitTypeId,PMSKU.Quantity
PaymentQuantity,PMSKU.Price,PMSKU.Amount,PMSKU.vat,PMSKU.VatRate,PMSKU.NormalDiscount,PMSKU.
SpecialDiscount,PMSKU.ManualDiscount,PMSKU.SalesDiscount,PMSKU.PromotionDiscount
--,PM.InterfaceStatus
,convert(tinyint, case when IsTransfered=1 and PM.InterfaceStatus=0 then 1 else
PM.InterfaceStatus end) InterfaceStatus
,PM.CreatedDate,PM.UpdatedDate
,PM.CompanyCode --,PM.[SOCompanyId] CompanyCode
,hst.BranchGroupCode AS BranchCode --,convert(nvarchar(30), PM.BranchCode) BranchCode
,convert(date,PM.OnDate) PMDate
,so.UserId,PMSKU.RefPaymentSKUId,SO.SalesTeamId SaleOrderSalesTeamId,so.Status
SaleOrderStatus,PM.IsCancel,SO.StorageCode,DO.DeliveryOnDate DO_DATE
,PM.PlantCode --,socompanybranch.PlantCode
,PMSKU.PaymentSKUId

FROM RPT_VSMS_Payment PM WITH (nolock)
LEFT JOIN RPT_VSMS_PaymentSKU PMSKU WITH (nolock) ON PMSKU.PaymentId=PM.PaymentId and
PMSKU.SaleOrgId=1
LEFT JOIN RPT_VSMS_SaleOrder SO WITH (nolock) on SO.SaleOrderId=PM.SaleOrderId and
PM.DelFlag=0 and SO.SaleOrgId=1
LEFT JOIN RPT_VSMS_Delivery DO WITH (nolock) ON DO.DeliveryId=PM.DeliveryId and DO.PaymentId
= PM.PaymentId and Isnull(DO.DeliveryPlanDetailDelFlag, 0)=0 and DO.SaleOrgId=1
LEFT JOIN RPT_VSMS_HierarchySalesTeam hst WITH (nolock) on hst.SalesteamId=PM.SalesTeamId
and hst.SaleOrgId=1
LEFT JOIN RPT_VSMS_Product P WITH (nolock) on P.SOProductId = PMSKU.SOProductId and
P.SaleOrgId=1

WHERE cast(PM.OnDate as date) between '2021-10-01' and '2022-01-31'
and so.Status in (3,4,6) and PM.CompanyCode = '4900' and PM.SaleOrgId=1
```

CMKL Version:

```
SELECT

PM.PaymentId, PM.OnDate, PM.ReceiptNo, PM.DOCUMENT_NUMBER, SO.SaleOrderId,
PM.SOCompanyBranchId, SO.latitude, SO.longitude, SO.AduserId AS UserName, so.ShelfId,
so.SOCustomerId, PMSKU.SOProductId, P.ProductCode, P.ProductName, SO.SaleType,
PMSKU.ItemType, PMSKU.SKUUnitTypeId, PMSKU.Quantity PaymentQuantity, PMSKU.Price,
PMSKU.Amount, PMSKU.vat, PMSKU.VatRate, PMSKU.NormalDiscount, PMSKU.SpecialDiscount,
PMSKU.ManualDiscount, PMSKU.SalesDiscount, PMSKU.PromotionDiscount, PM.CreatedDate,
PM.UpdatedDate, PM.CompanyCode, hst.BranchGroupCode AS BranchCode, so.UserId,
PMSKU.RefPaymentSKUId, SO.SalesTeamId SaleOrderSalesTeamId, so.Status SaleOrderStatus,
PM.IsCancel, SO.StorageCode, DO.DeliveryOnDate DO_DATE, PM.PlantCode, PMSKU.PaymentSKUId

FROM delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_payment` PM

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_paymentsku` PMSKU ON
PMSKU.PaymentId=PM.PaymentId AND PMSKU.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_saleorder` SO ON
SO.SaleOrderId=PM.SaleOrderId AND PM.DelFlag=0 AND SO.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_delivery`
DO ON DO.DeliveryId=PM.DeliveryId AND DO.PaymentId = PM.PaymentId AND DO.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_hierarchysalesteam` hst ON
hst.SalesteamId=PM.SalesTeamId and hst.SaleOrgId=1

LEFT JOIN delta.`s3a://tb-vsms-sales-cvm/rpt_vsms_product` P ON P.SOProductId =
PMSKU.SOProductId AND P.SaleOrgId=1

WHERE CAST(PM.OnDate AS date) BETWEEN '2021-10-01' AND '2022-01-31'
AND (so.Status = 3 OR so.Status = 4 OR so.Status = 6)
AND PM.CompanyCode = '4900' AND PM.SaleOrgId=1
```

Deviation from ThaiBev version:

- Remove `--,PM.InterfaceStatus`
- Remove `,convert(tinyint, case when IsTransfered=1 and PM.InterfaceStatus=0 then 1 else PM.InterfaceStatus end) InterfaceStatus`
- Remove `--,PM.[SOCompanyId] CompanyCode`
- Remove `--,convert(nvarchar(30), PM.BranchCode) BranchCode`
- Remove `convert(date,PM.OnDate) PMDate`
- Remove `--,socompanybranch.PlantCode`
- Remove `Isnull(DO.DeliveryPlanDetailDelFlag, 0)=0 and`
- Remove `WITH (nolock)`
- Changes from `so.Status in (3,4,6)` to `(so.Status = 3 OR so.Status = 4 or so.Status = 6)`

C. Example Code Snippet for Connecting Python with Dremio

C.1 Connecting Python with Dremio via JDBC

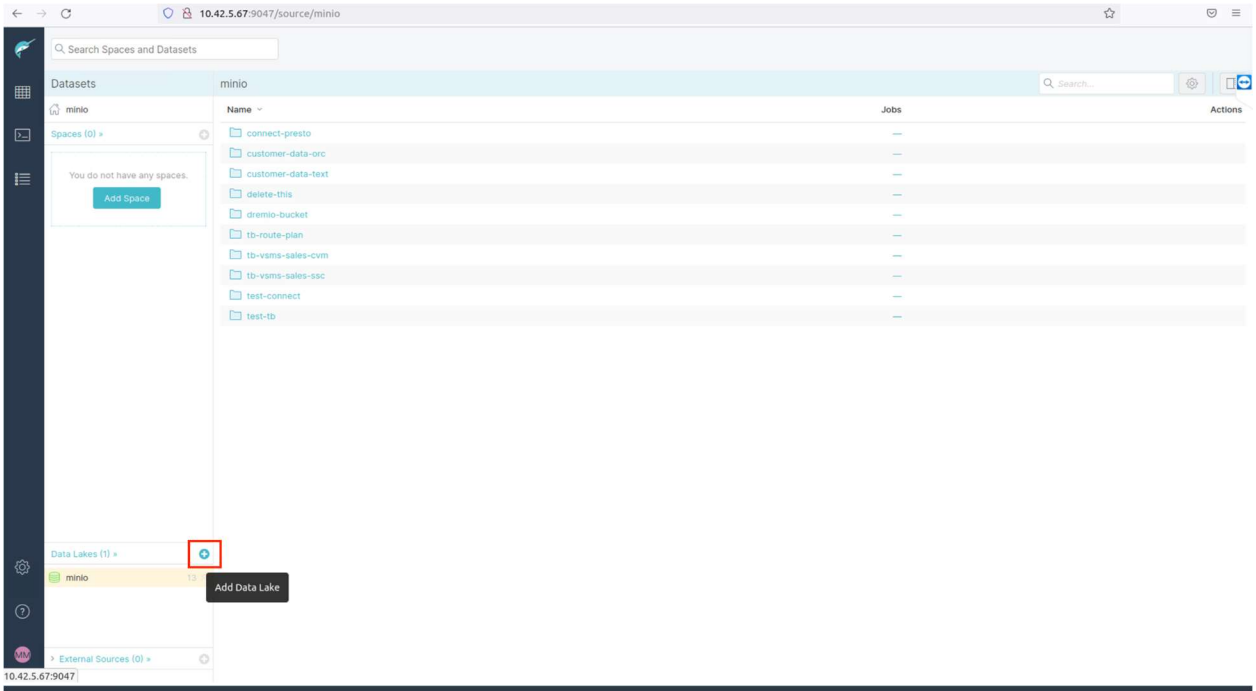
```
# https://gist.github.com/narendrans/550860550fcebb3da1040aeb5ff31458

from __future__ import generators
import jaydebeapi
import time



def ResultIterator(cursor, arraysize=10000):
    'iterator using fetchmany and consumes less memory'
    while True:
        results = cursor.fetchmany(arraysize)
        if not results:
            break
        for result in results:
            yield result


start_time = time.time()


# conn is a DB-API database connection
conn = jaydebeapi.connect("com.dremio.jdbc.Driver",
"jdbc:dremio:direct=localhost:31010",
["minio", "minio123"],
'/home/kraikrai/Documents/dremio-jdbc-driver-3.1.9-201904051346520183-a35b753.jar',)
curs = conn.cursor()
sql = '''SELECT * FROM minio."tb-route-plan"."tsr_rpt_vsms_routeplan"'''
curs.execute(sql)


count = 0
for row in ResultIterator(curs):
    print(row)
    count+=1
    if count % 1000000 == 0:
        print(count)
        lap_time = time.time()
        print("lap:", lap_time - start_time)

stop_time = time.time()

print("ELAPSED TIME:", stop_time - start_time)
```

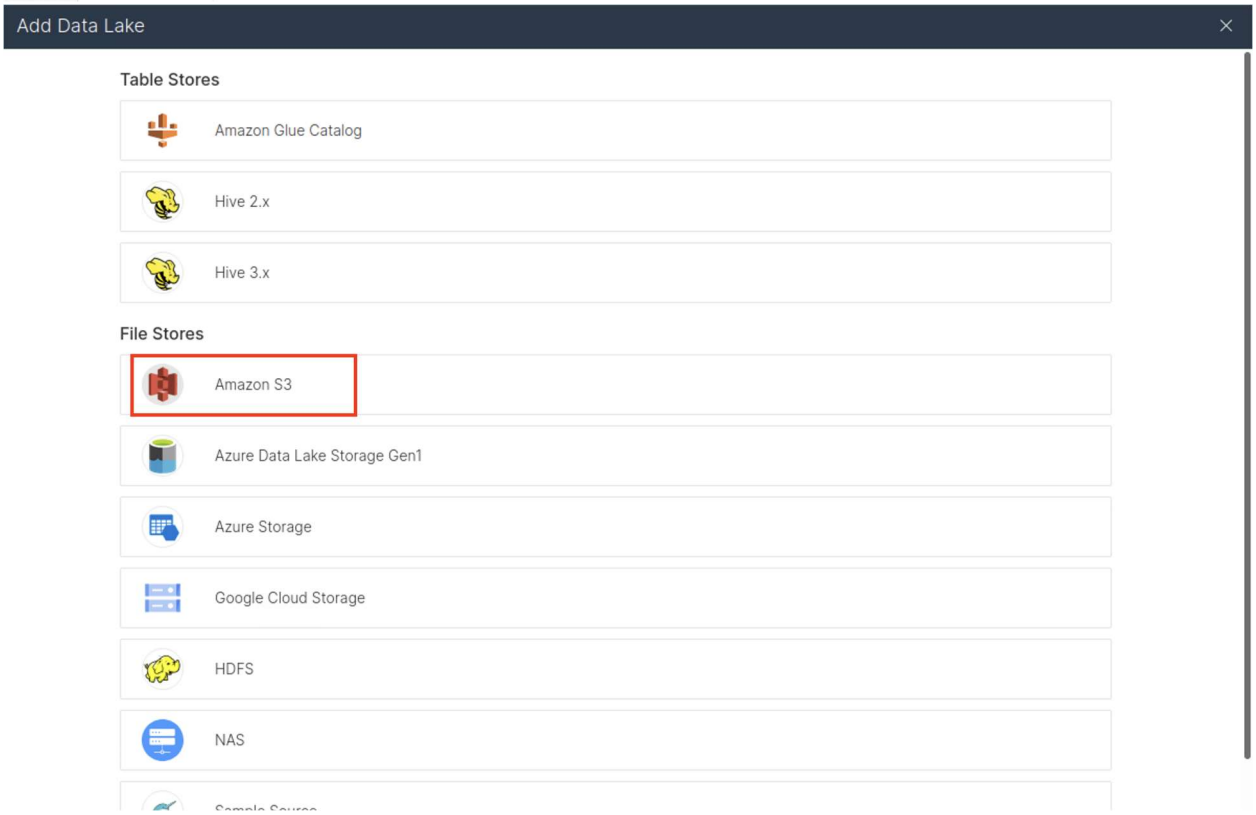
C.2 Connecting Python with Dremio via ODBC

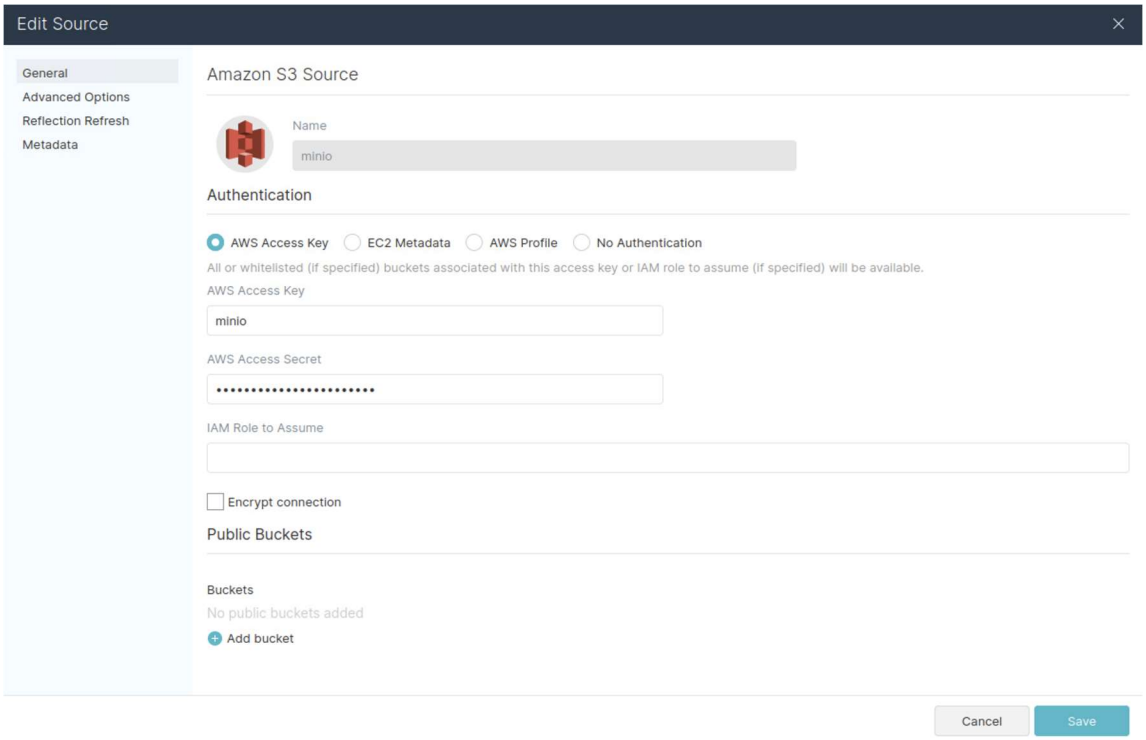
```
import pyodbc, pandas, time

start_time = time.time()

host = 'localhost'
port = 31010
uid = 'minio'
pwd = 'minio123'
driver = '/opt/dremio-odbc/lib64/libdrillodbc_sb64.so'

cnxn =
pyodbc.connect("Driver={};ConnectionType=Direct;HOST={};PORT={};AuthenticationType=Plain;UID
={};PWD={}".format(driver, host,port,uid,pwd),autocommit=True)
cur = cnxn.cursor()

sql = '''SELECT * FROM minio."tb-route-plan"."tsr_rpt_vsms_routeplan"'''

cur.execute(sql)

count = 0
for row in cur:
    print(row)
    count+=1
    if count % 1000000 == 0:
        print(count)
        lap_time = time.time()
        print("lap:", lap_time - start_time)

stop_time = time.time()

print("ELAPSED TIME:", stop_time - start_time)
```

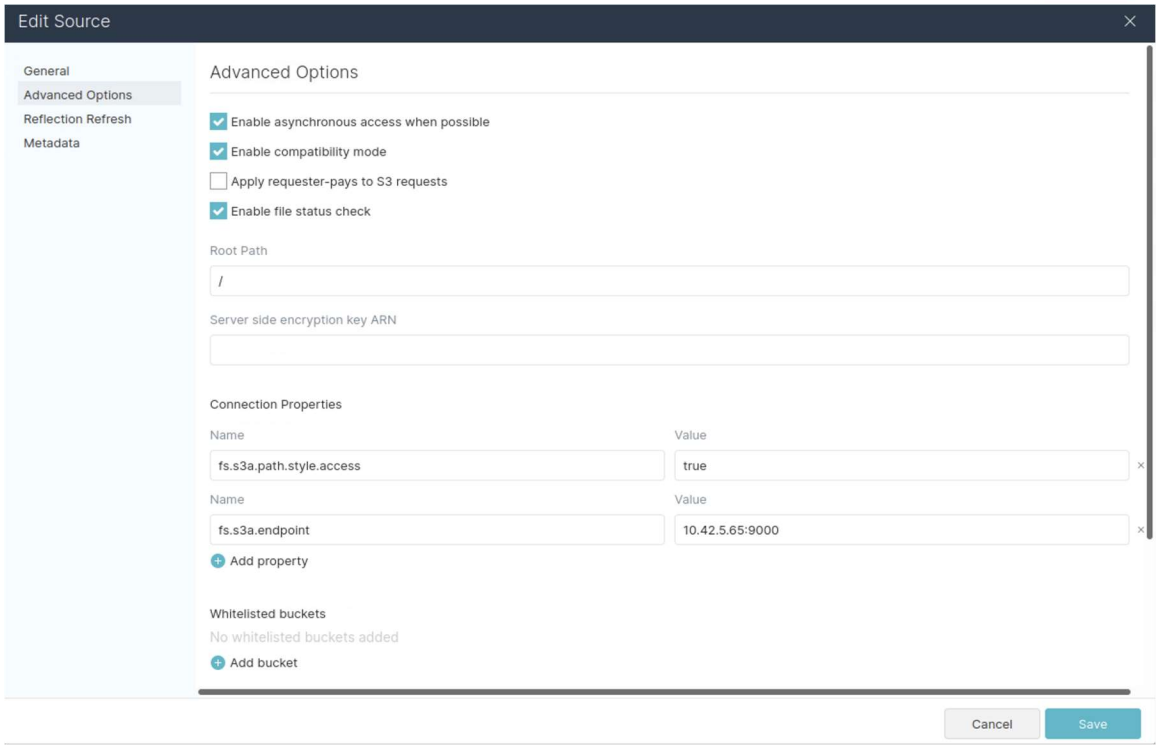
D. Dremio Example Graphic User Interface

D.1 Connecting to S3 Data Source (Minio)





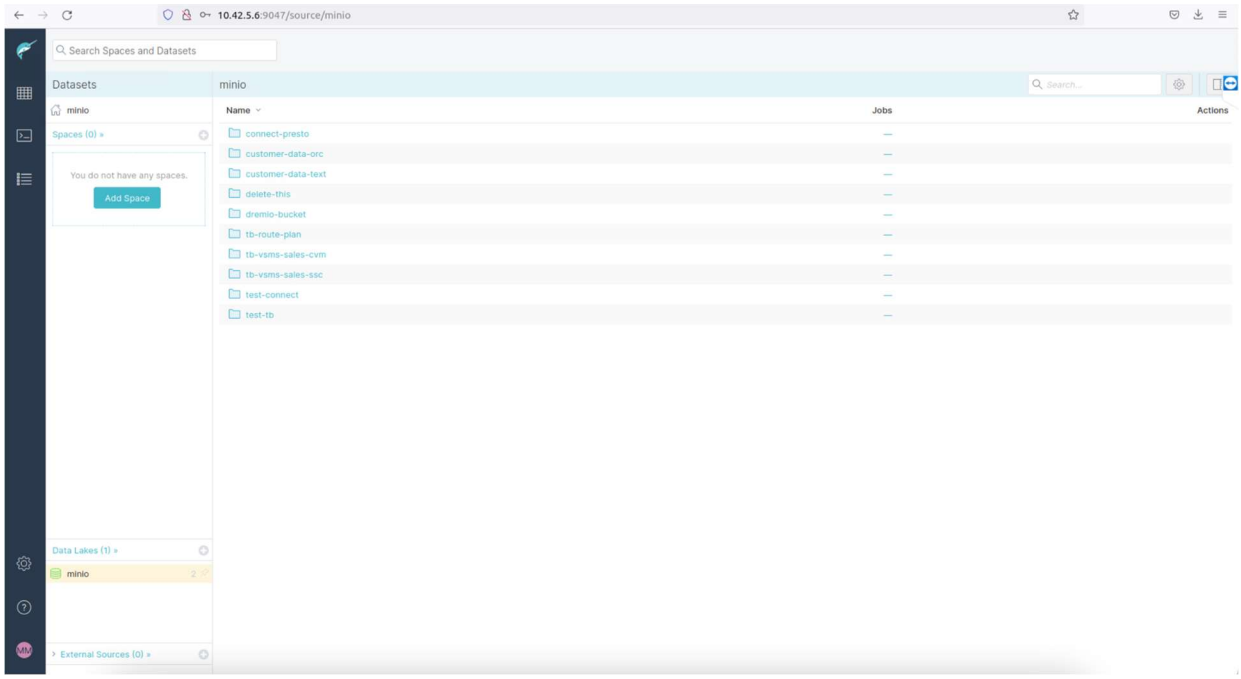


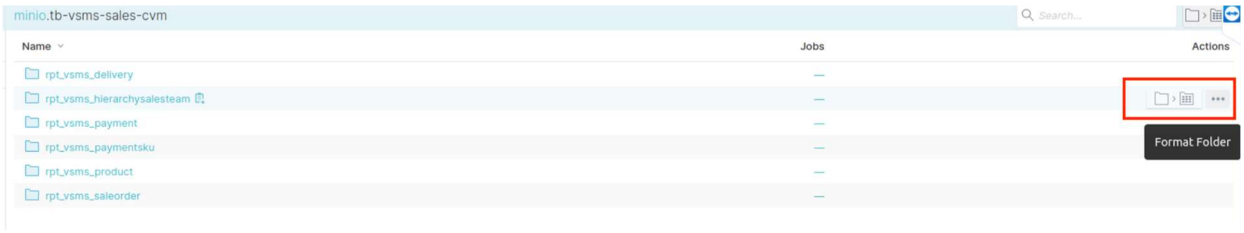


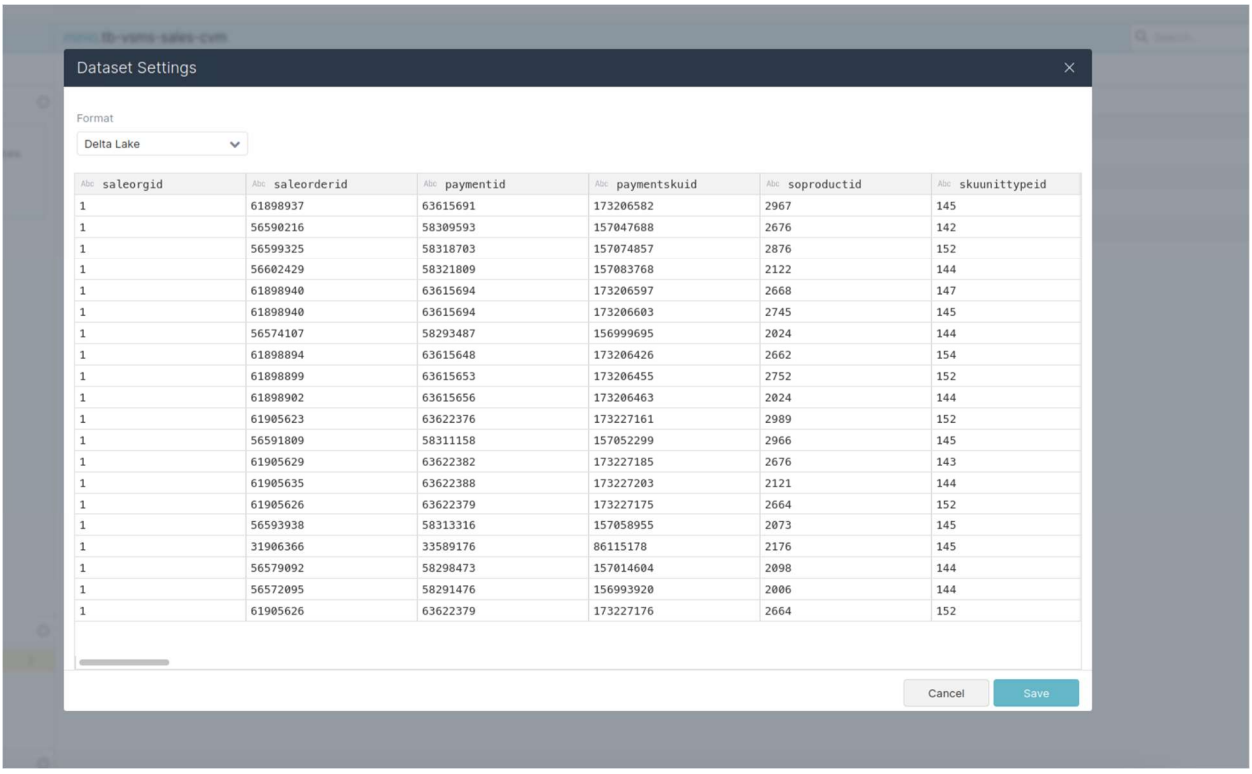
Reference: https://docs.dremio.com/software/data-sources/s3/#configuring-s3-for-minio

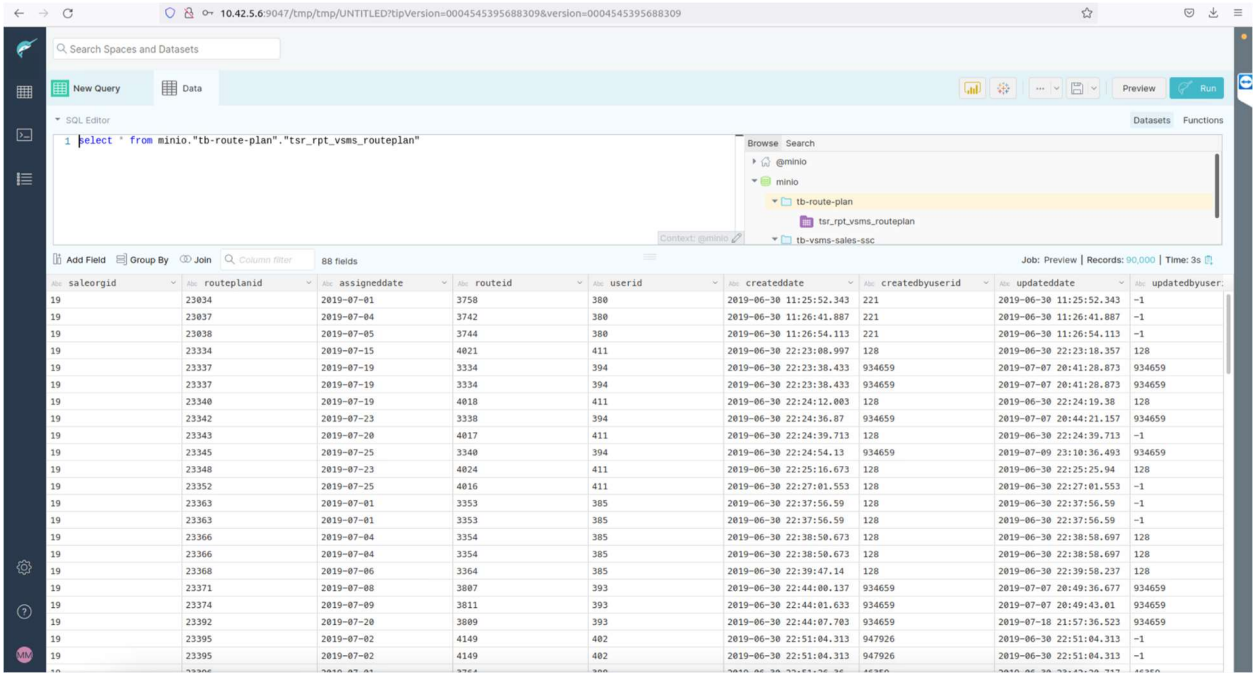
Reference Configuration: https://docs.dremio.com/software/deployment/dist-store-config/

D.2 Read Delta Table From Data Source (Minio)

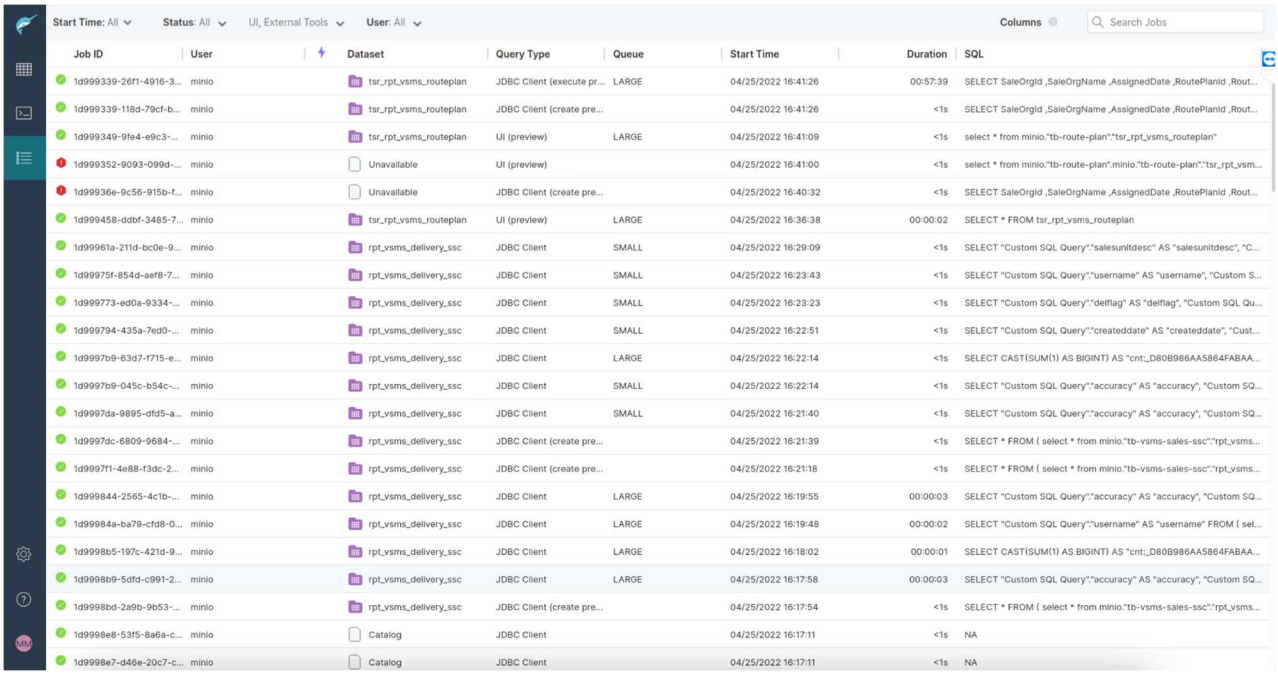


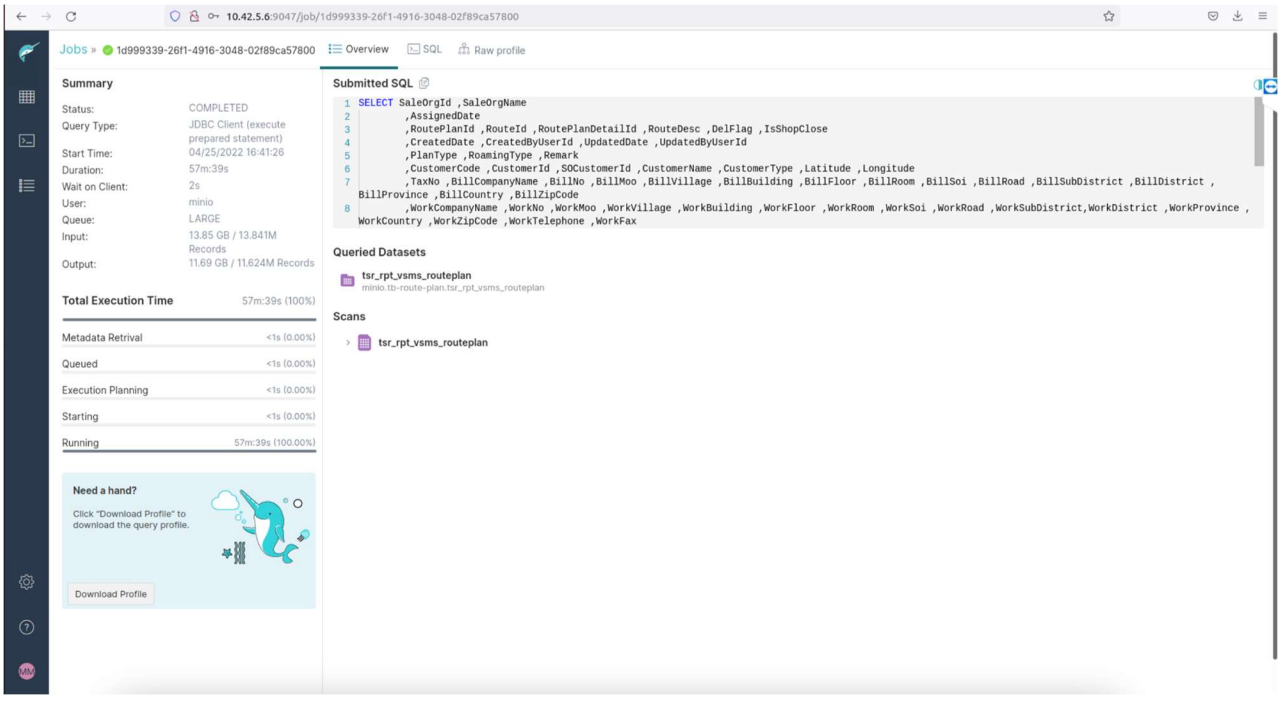


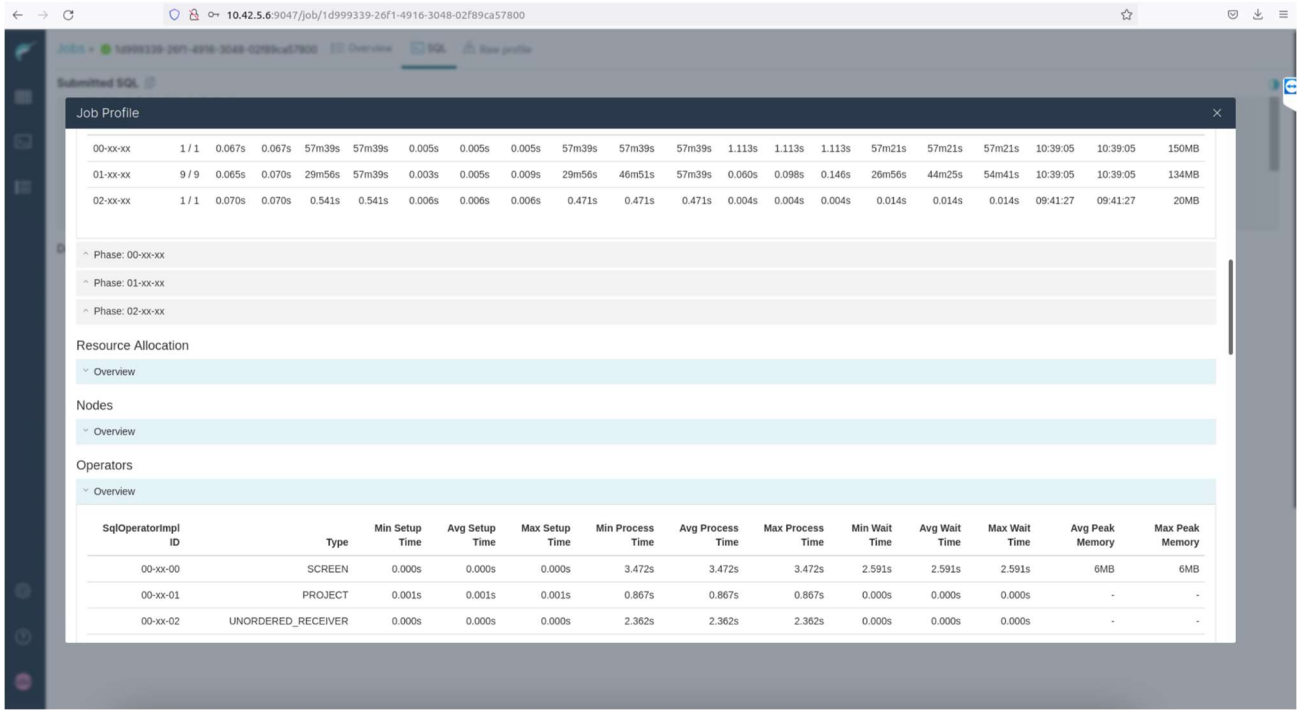

Dataset Settings
Format
Delta Lake
saleorgid saleorderid paymentid paymentskuid soproductid skuunittypeid
1 61898937 63615691 173206582 2967 145
1 56590216 58309593 157047688 2676 142
1 56599325 58318703 157074857 2876 152
1 56602429 58321809 157083768 2122 144
1 61898940 63615694 173206597 2668 147
1 61898940 63615694 173206603 2745 145
1 56574107 58293487 156999695 2024 144
1 61898894 63615648 173206426 2662 154
1 61898899 63615653 173206455 2752 152
1 61898902 63615656 173206463 2024 144
1 61905623 63622376 173227161 2989 152
1 56591809 58311158 157052299 2966 145
1 61905629 63622382 173227185 2676 143
1 61905635 63622388 173227203 2121 144
1 61905626 63622379 173227175 2664 152
1 56593938 58313316 157058955 2073 145
1 31906366 33589176 86115178 2176 145
1 56579092 58298473 157014604 2098 144
1 56572095 58291476 156993920 2006 144
1 61905626 63622379 173227176 2664 152
Cancel
Save

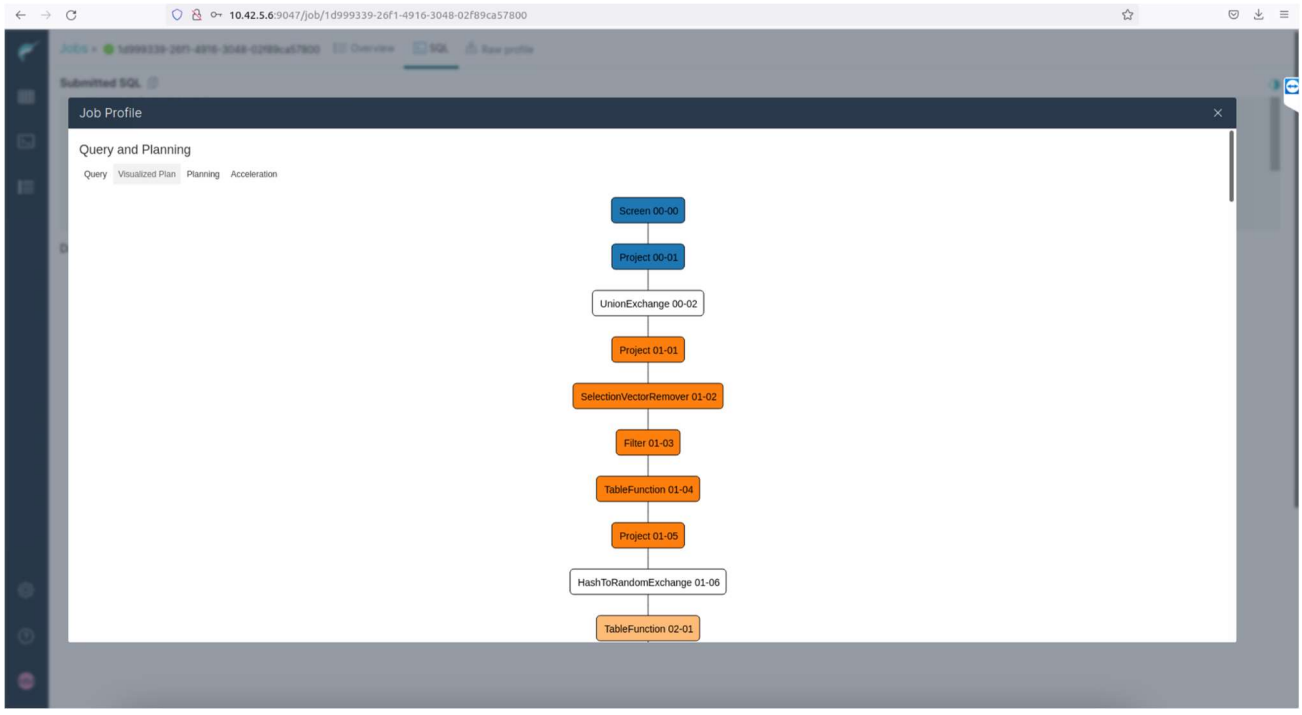

10.42.5.6:9047/tmp/tmp/UNTITLED?tipVersion=0004545395688309&version=0004545395688309
Search Spaces and Datasets
New Query
Data
Preview
Run
SQL Editor
Datasets
Functions
select * from minio."tb-route-plan"."tsr_rpt_vsms_routeplan"
Browse
Search
@minio
minio
tb-route-plan
tsr_rpt_vsms_routeplan
tb-vsms-sales-ssc
Add Field
Group By
Join
Column filter
88 fields
Job: Preview | Records: 90,000 | Time: 3s
saleorgid routeplanid assigneddate routeid userid createddate createdbyuserid updateddate updatedbyuser
19 23034 2019-07-01 3758 380 2019-06-30 11:25:52.343 221 2019-06-30 11:25:52.343 -1
19 23037 2019-07-04 3742 380 2019-06-30 11:26:41.887 221 2019-06-30 11:26:41.887 -1
19 23038 2019-07-05 3744 380 2019-06-30 11:26:54.113 221 2019-06-30 11:26:54.113 -1
19 23334 2019-07-15 4021 411 2019-06-30 22:23:08.997 128 2019-06-30 22:23:18.357 128
19 23337 2019-07-19 3334 394 2019-06-30 22:23:38.433 934659 2019-07-07 20:41:28.873 934659
19 23337 2019-07-19 3334 394 2019-06-30 22:23:38.433 934659 2019-07-07 20:41:28.873 934659
19 23340 2019-07-19 4018 411 2019-06-30 22:24:12.003 128 2019-06-30 22:24:19.38 128
19 23342 2019-07-23 3338 394 2019-06-30 22:24:36.87 934659 2019-07-07 20:44:21.157 934659
19 23343 2019-07-20 4017 411 2019-06-30 22:24:39.713 128 2019-06-30 22:24:39.713 -1
19 23345 2019-07-25 3340 394 2019-06-30 22:24:54.13 934659 2019-07-09 23:10:36.493 934659
19 23348 2019-07-23 4024 411 2019-06-30 22:25:16.673 128 2019-06-30 22:25:25.94 128
19 23352 2019-07-25 4016 411 2019-06-30 22:27:01.553 128 2019-06-30 22:27:01.553 -1
19 23363 2019-07-01 3353 385 2019-06-30 22:37:56.59 128 2019-06-30 22:37:56.59 -1
19 23363 2019-07-01 3353 385 2019-06-30 22:37:56.59 128 2019-06-30 22:37:56.59 -1
19 23366 2019-07-04 3354 385 2019-06-30 22:38:50.673 128 2019-06-30 22:38:58.697 128
19 23366 2019-07-04 3354 385 2019-06-30 22:38:50.673 128 2019-06-30 22:38:58.697 128
19 23368 2019-07-06 3364 385 2019-06-30 22:39:47.14 128 2019-06-30 22:39:58.237 128
19 23371 2019-07-08 3807 393 2019-06-30 22:44:00.137 934659 2019-07-07 20:49:36.677 934659
19 23374 2019-07-09 3811 393 2019-06-30 22:44:01.633 934659 2019-07-07 20:49:43.01 934659
19 23392 2019-07-20 3809 393 2019-06-30 22:44:07.703 934659 2019-07-18 21:57:36.523 934659
19 23395 2019-07-02 4149 402 2019-06-30 22:51:04.313 947926 2019-06-30 22:51:04.313 -1
19 23395 2019-07-02 4149 402 2019-06-30 22:51:04.313 947926 2019-06-30 22:51:04.313 -1

D.3 Query Result

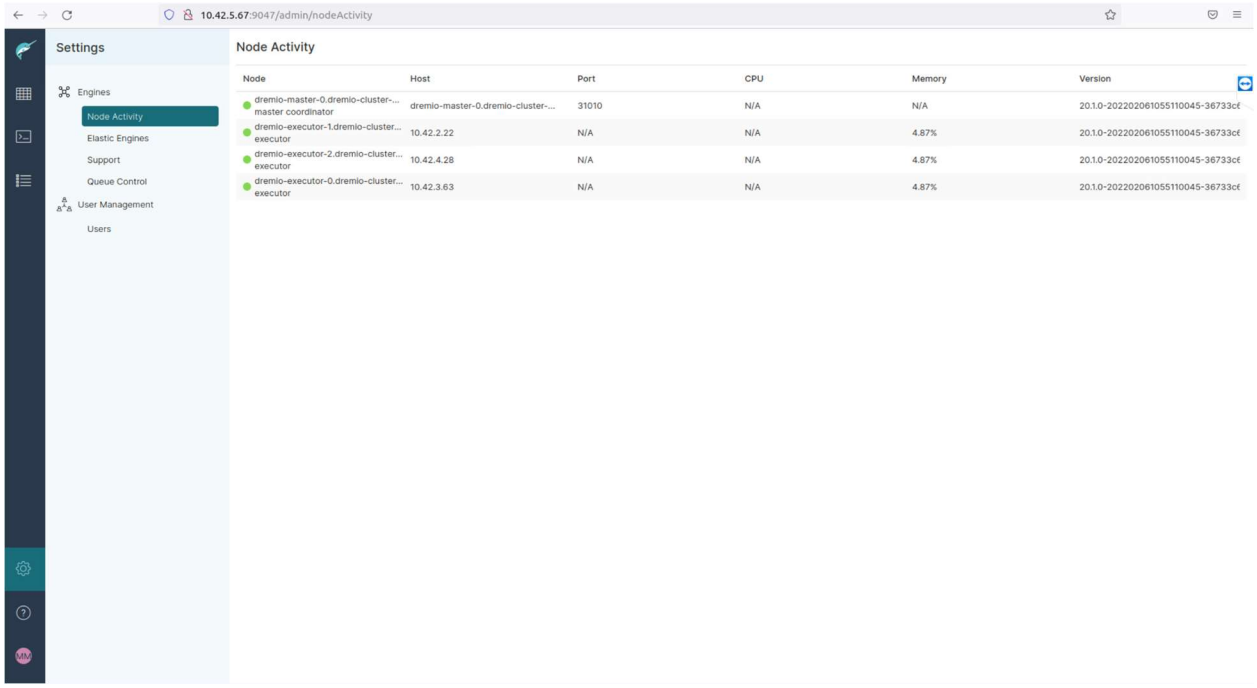





10.42.5.6:9047/job/1d999339-26f1-4916-3048-02f89ca57800
Job Profile
00-xx-xx 1 / 1 0.067s 0.067s 57m39s 57m39s 0.005s 0.005s 0.005s 57m39s 57m39s 57m39s 1.113s 1.113s 1.113s 57m21s 57m21s 57m21s 10:39:05 10:39:05 150MB
01-xx-xx 9 / 9 0.065s 0.070s 29m56s 57m39s 0.003s 0.005s 0.009s 29m56s 46m51s 57m39s 0.060s 0.098s 0.146s 26m56s 44m25s 54m41s 10:39:05 10:39:05 134MB
02-xx-xx 1 / 1 0.070s 0.070s 0.541s 0.541s 0.006s 0.006s 0.006s 0.471s 0.471s 0.471s 0.004s 0.004s 0.004s 0.014s 0.014s 0.014s 09:41:27 09:41:27 20MB
Phase: 00-xx-xx
Phase: 01-xx-xx
Phase: 02-xx-xx
Resource Allocation
Overview
Nodes
Overview
Operators
Overview
SqlOperatorImpl ID Type Min Setup Time Avg Setup Time Max Setup Time Min Process Time Avg Process Time Max Process Time Min Wait Time Avg Wait Time Max Wait Time Avg Peak Memory Max Peak Memory
00-xx-00 SCREEN 0.000s 0.000s 0.000s 3.472s 3.472s 3.472s 2.591s 2.591s 2.591s 6MB 6MB
00-xx-01 PROJECT 0.001s 0.001s 0.001s 0.867s 0.867s 0.867s 0.000s 0.000s 0.000s - -
00-xx-02 UNORDERED_RECEIVER 0.000s 0.000s 0.000s 2.362s 2.362s 2.362s 0.000s 0.000s 0.000s - -


10.42.5.6:9047/job/1d999339-26f1-4916-3048-02f89ca57800
Job Profile
Query and Planning
Query
Visualized Plan
Planning
Acceleration
Screen 00-00
Project 00-01
UnionExchange 00-02
Project 01-01
SelectionVectorRemover 01-02
Filter 01-03
TableFunction 01-04
Project 01-05
HashToRandomExchange 01-06
TableFunction 02-01

D.4 Settings


10.42.5.67:9047/admin/nodeActivity
Settings
Engines
Node Activity
Elastic Engines
Support
Queue Control
User Management
Users
Node Activity
Node
Host
Port
CPU
Memory
Version
dremio-master-0.dremio-cluster-...
master coordinator
dremio-master-0.dremio-cluster-...
31010
N/A
N/A
20.1.0-202202061055110045-36733c6
dremio-executor-1.dremio-cluster...
executor
10.42.2.22
N/A
N/A
4.87%
20.1.0-202202061055110045-36733c6
dremio-executor-2.dremio-cluster...
executor
10.42.4.28
N/A
N/A
4.87%
20.1.0-202202061055110045-36733c6
dremio-executor-0.dremio-cluster...
executor
10.42.3.63
N/A
N/A
4.87%
20.1.0-202202061055110045-36733c6

E. Connecting Delta Table In Minio With Apache Hive

Step 1: Create Delta Table in Minio

```
from pyspark import SparkContext
from pyspark.sql import SparkSession
from delta import *
from minio import Minio


def load_config(spark_context: SparkContext):
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.access.key','minio')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.secret.key','minio123')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.path.style.access','true')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.impl','org.apache.hadoop.fs.s3a.S3AFileSystem')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.endpoint','10.42.5.215:9000')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.connection.ssl.enabled','false')

builder = SparkSession.builder \
               .appName('Pyspark Delta Minio') \
               .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
               .config("spark.sql.catalog.spark_catalog",
"org.apache.spark.sql.delta.catalog.DeltaCatalog")

spark = spark = configure_spark_with_delta_pip(builder).getOrCreate()
load_config(spark.sparkContext)


# configure and create SparkSession
spark = SparkSession.builder.appName('Pyspark Delta Minio').getOrCreate()
load_config(spark.sparkContext)

# input
in_df = spark.read.option("header","true").csv("/home/saturday/TB-Enterprise-
Arch/test_connect_presto/student.csv")

# minio config
minio_config_dict = {"access_key":"minio", "secret_key":"minio123", "endpoint":"10.42.5.215:9000"}
minio_path = 's3a://connect-presto/'


# write dataframe as a delta
in_df.write.format('delta').save(minio_path + "example_pyspark_delta_minio")

# read delta as a dataframe
out_df =  spark.read.format("delta").load(minio_path + "example_pyspark_delta_minio")
out_df.count()
out_df.show()
```

Step 2: Fix hadoopo/core-site.xml to connect with user's Minio's endpoint

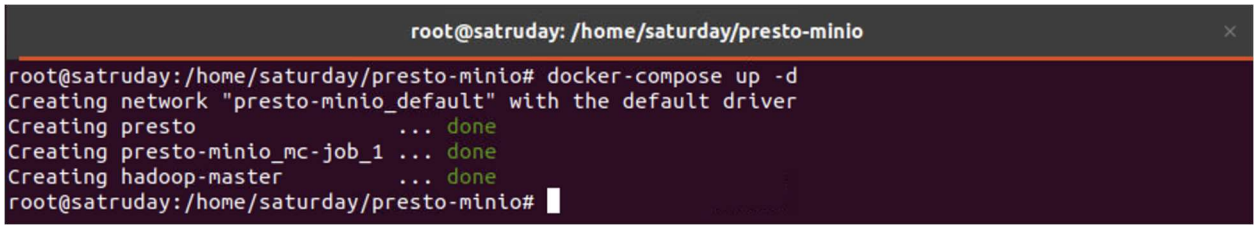
```
<property>
   <name>fs.s3a.access.key</name>
   <value>minio</value>
</property>
<property>
   <name>fs.s3a.secret.key</name>
   <value>minio123</value>
</property>
<property>
   <name>fs.s3a.connection.ssl.enabled</name>
   <value>false</value>
</property>
<property>
   <name>fs.s3a.path.style.access</name>
   <value>true</value>
</property>
<property>
   <name>fs.s3a.endpoint</name>
   <value>[ USER MINIO ENDPOINT ]:9000</value>
</property>
```

Step 3: Fix presto/minio.properties to connect with user's Minio's endpoint

```
connector.name=hive-hadoop2
hive.metastore.uri=thrift://hadoop-master:9083
hive.s3.path-style-access=true
hive.s3.endpoint=[ USER MINIO ENDPOINT ]:9000
hive.s3.aws-access-key=minio
hive.s3.aws-secret-key=minio123
hive.non-managed-table-writes-enabled=true
hive.storage-format=ORC
```

Step 4: Setup Environment via Docker Compose

```
docker-docker-compose up -d
```



Step 5: Generating a Delta Table Creation Script

```
from pyspark import SparkContext
from pyspark.sql import SparkSession
from delta import *
from minio import Minio


# Quick Setup
# setup openEBS: https://docs.openebs.io/docs/next/uglocalpv-hostpath.html
# setup Minio: helm install krai-distributed-minio --set
mode=distributed,accessKey=minio,secretKey=minio123,persistence.storageClass=openebs-
hostpath,service.type=NodePort,persistence.enabled=true,replicas=4 minio/minio
# delta lake library: pip install delta-spark
# *** make sure to create the bucket before running the script ***
# manage minio directory: pip install minio / minio-py3



def load_config(spark_context: SparkContext):
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.access.key','minio')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.secret.key','minio123')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.path.style.access','true')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.impl','org.apache.hadoop.fs.s3a.S3AFileSystem')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.endpoint','10.42.5.215:9000')
   spark_context._jsc.hadoopConfiguration().set('fs.s3a.connection.ssl.enabled','false')

builder = SparkSession.builder \
               .appName('Pyspark Delta Minio') \
               .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
               .config("spark.sql.catalog.spark_catalog",
"org.apache.spark.sql.delta.catalog.DeltaCatalog")

spark = spark = configure_spark_with_delta_pip(builder).getOrCreate()
load_config(spark.sparkContext)


# configure and create SparkSession
spark = SparkSession.builder.appName('Pyspark Delta Minio').getOrCreate()
load_config(spark.sparkContext)

deltaTable = DeltaTable.forPath(spark,'s3a://connect-presto/example_pyspark_delta_minio')
```

```
deltaTable.generate("symlink_format_manifest")
```

Reference: https://docs.delta.io/latest/presto-integration.html

Step 6: Login to Apache Hive Image

```
docker exec -it hadoop-master /bin/bash
```

```
root@satruday:/home/saturday/presto-minio# docker exec -it hadoop-master /bin/bash
[root@hadoop-master /]#
```

Step 7: Create Delta Table Catalog

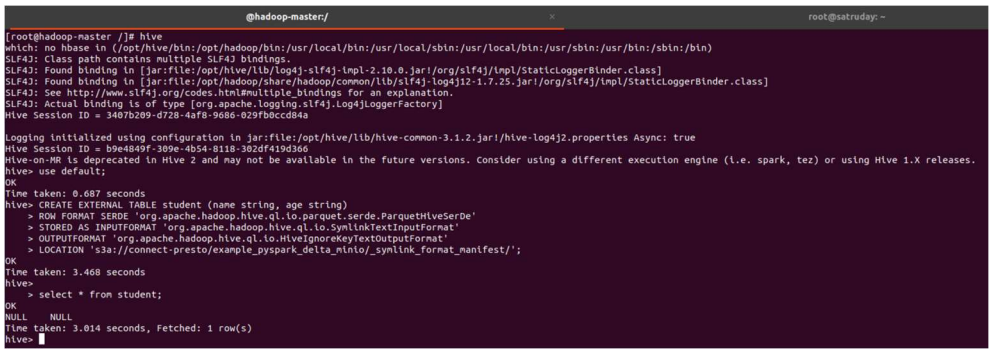
Example Generated Delta Table Creation Command

```
CREATE EXTERNAL TABLE student (name string, age string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3a://connect-presto/example_pyspark_delta_minio/_symlink_format_manifest/';
```

Example Command

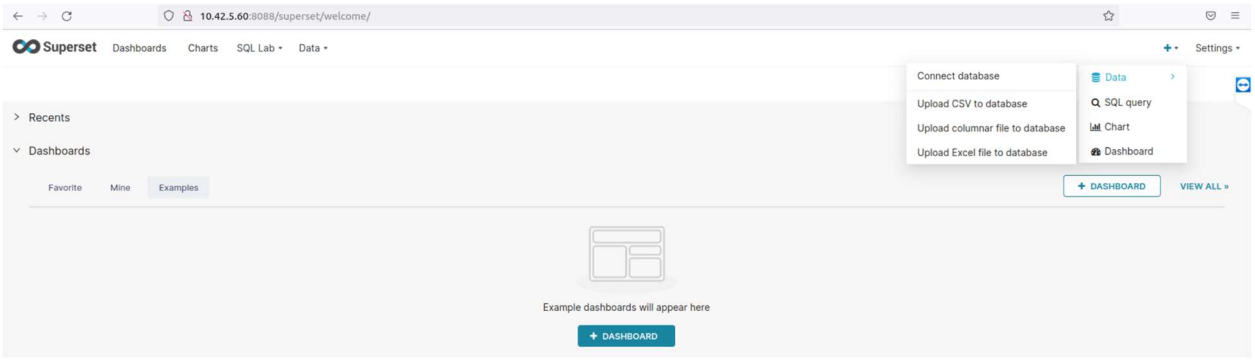
```
[root@hadoop-master /]# hive
hive> use default;
hive> [ Example Generated Delta Table Creation Command ]
hive> select * from customer_text;
```

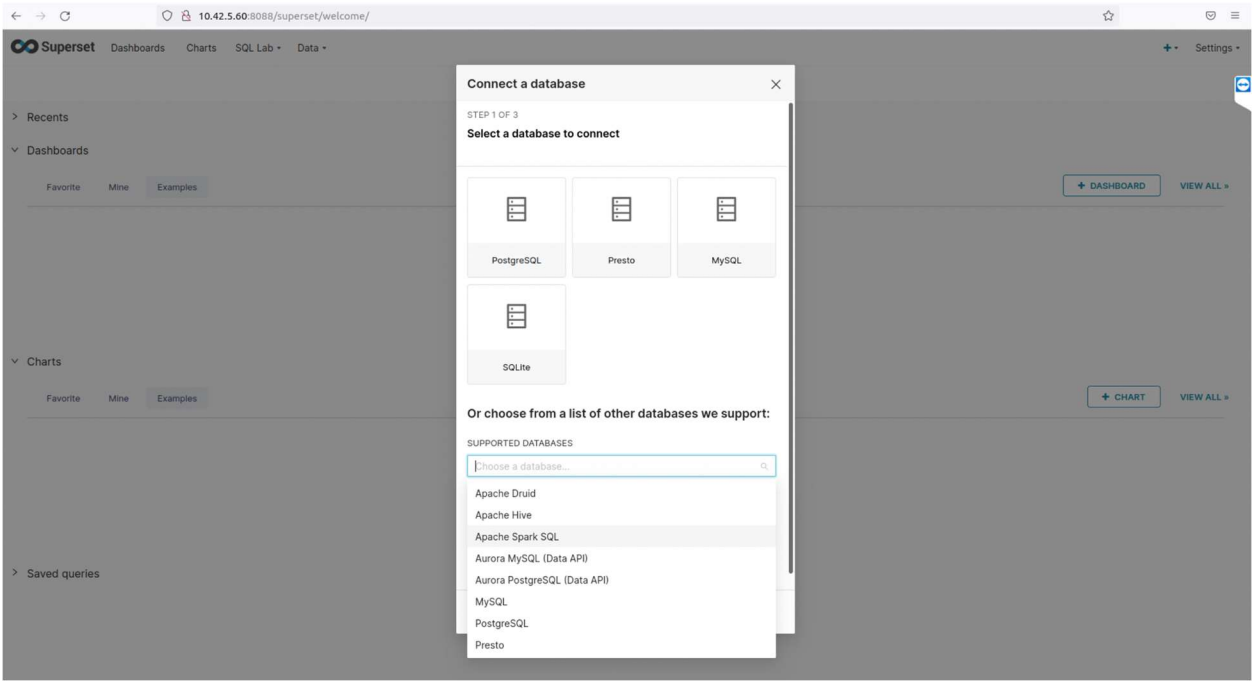
Example Output

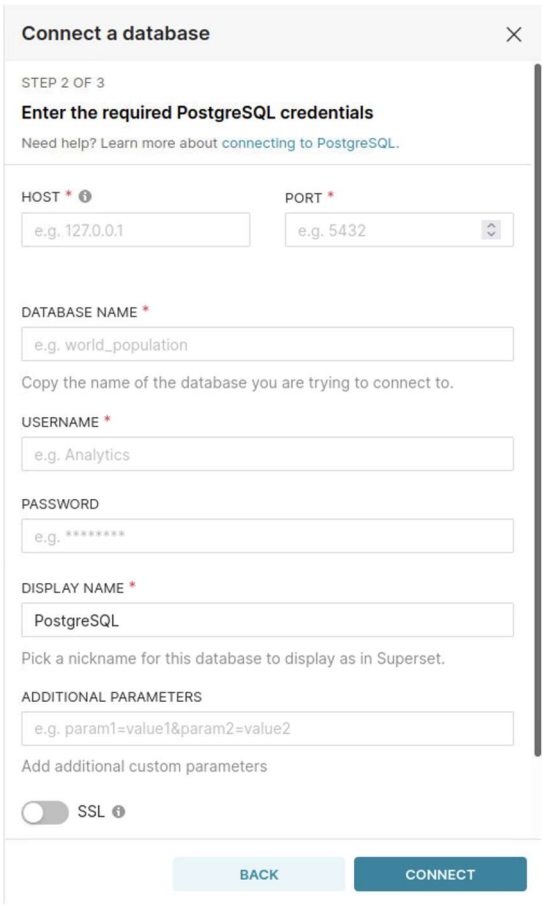


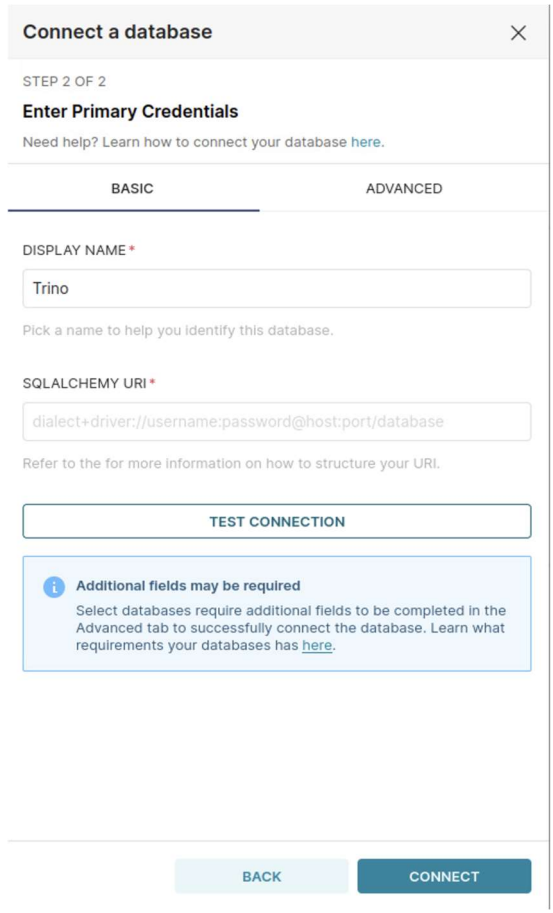
F. Apache Superset Example Graphic User Interface

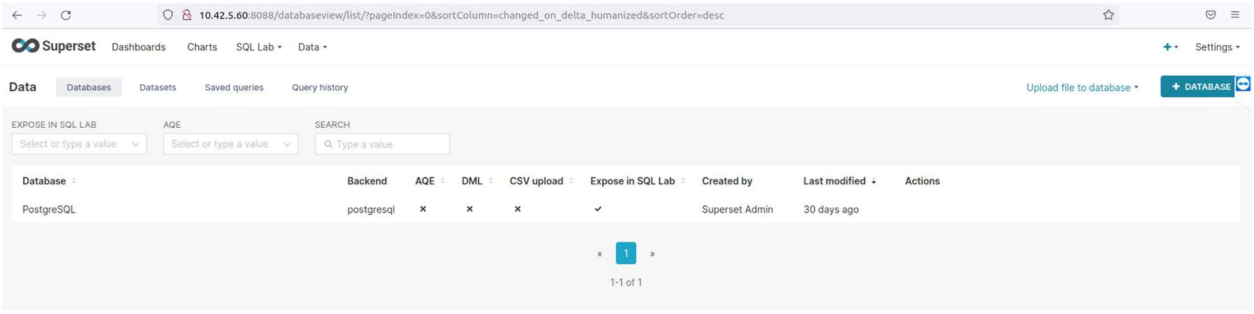
F.1 Data Sources (how to connect to data sources)



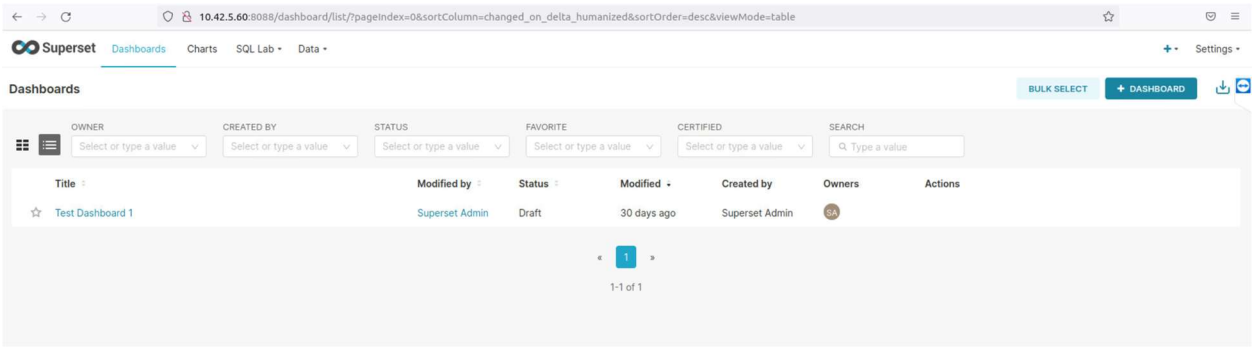


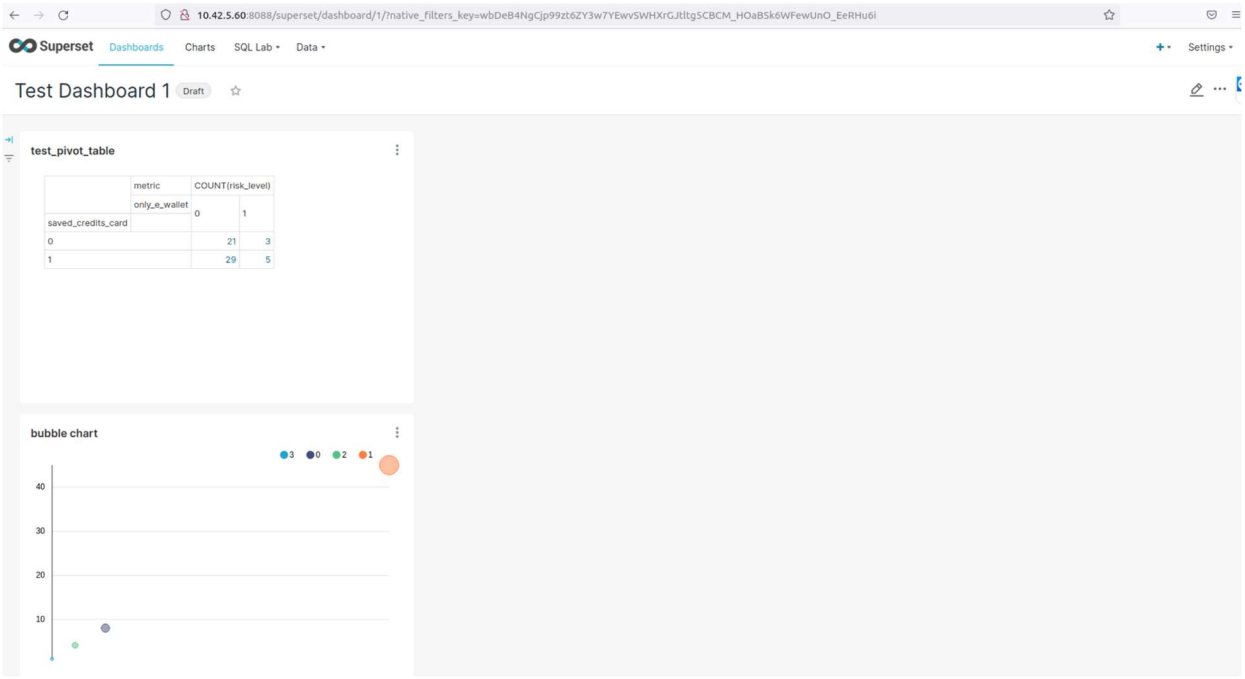

Connect a database
STEP 2 OF 3
Enter the required PostgreSQL credentials
Need help? Learn more about connecting to PostgreSQL.
HOST *
e.g. 127.0.0.1
PORT *
e.g. 5432
DATABASE NAME *
e.g. world_population
Copy the name of the database you are trying to connect to.
USERNAME *
e.g. Analytics
PASSWORD
e.g. ********
DISPLAY NAME *
PostgreSQL
Pick a nickname for this database to display as in Superset.
ADDITIONAL PARAMETERS
e.g. param1=value1&param2=value2
Add additional custom parameters
SSL
BACK
CONNECT


Connect a database
STEP 2 OF 2
Enter Primary Credentials
Need help? Learn how to connect your database here.
BASIC
ADVANCED
DISPLAY NAME *
Trino
Pick a name to help you identify this database.
SQLALCHEMY URI *
dialect+driver://username:password@host:port/database
Refer to the for more information on how to structure your URI.
TEST CONNECTION
Additional fields may be required
Select databases require additional fields to be completed in the Advanced tab to successfully connect the database. Learn what requirements your databases has here.
BACK
CONNECT

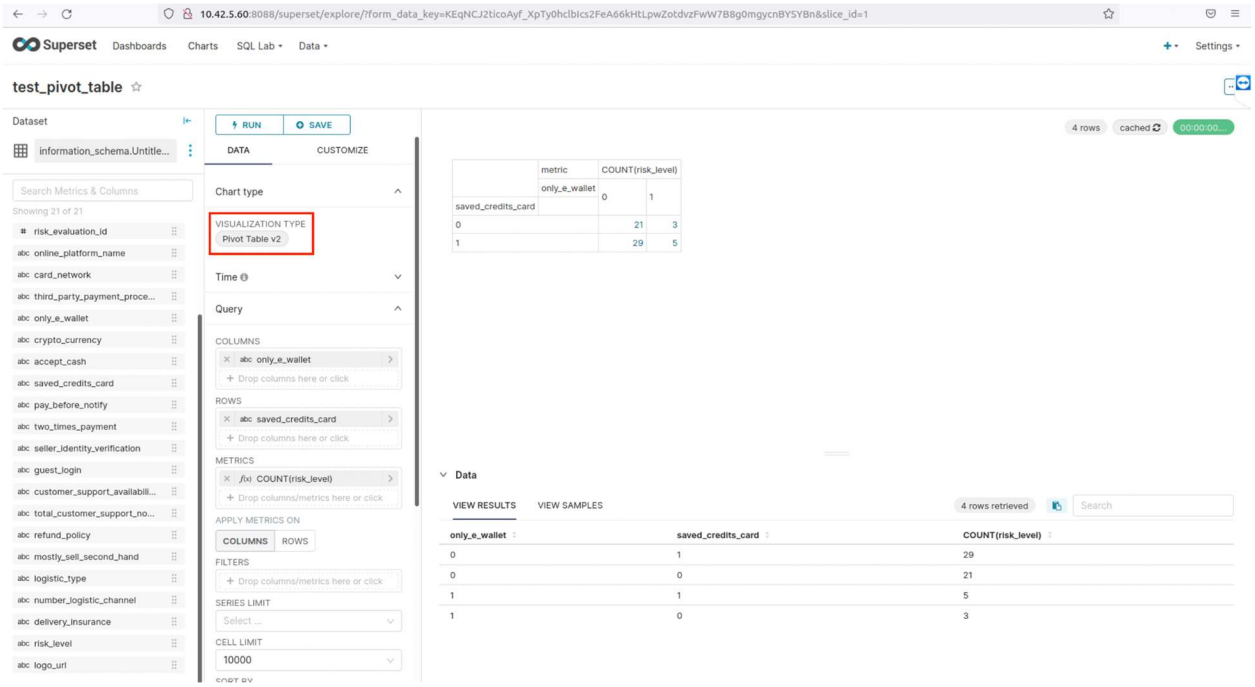

10.42.5.60:8088/databaseview/list/?pageIndex=0&sortColumn=changed_on_delta_humanized&sortOrder=desc
Superset
Dashboards
Charts
SQL Lab
Data
Settings
Data
Databases
Datasets
Saved queries
Query history
Upload file to database
+ DATABASE
EXPOSE IN SQL LAB
AQE
SEARCH
Database
Backend
AQE
DML
CSV upload
Expose in SQL Lab
Created by
Last modified
Actions
PostgreSQL
postgresql
Superset Admin
30 days ago
1-1 of 1

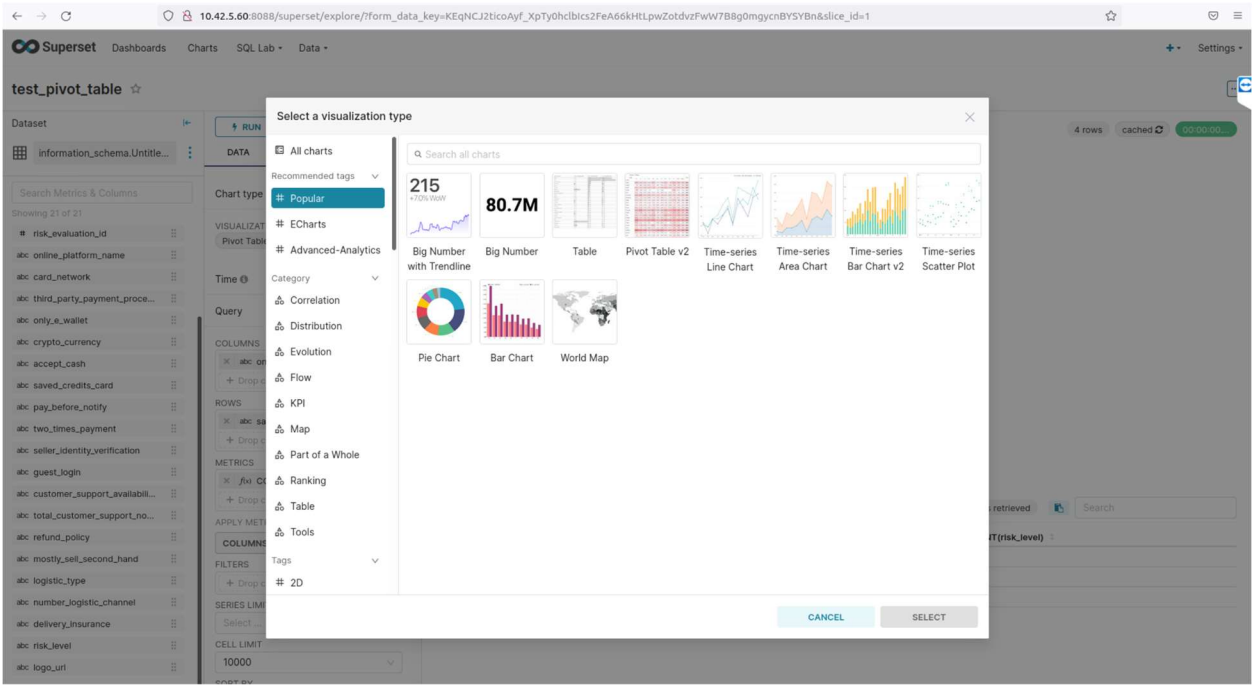
F.2 Dashboards





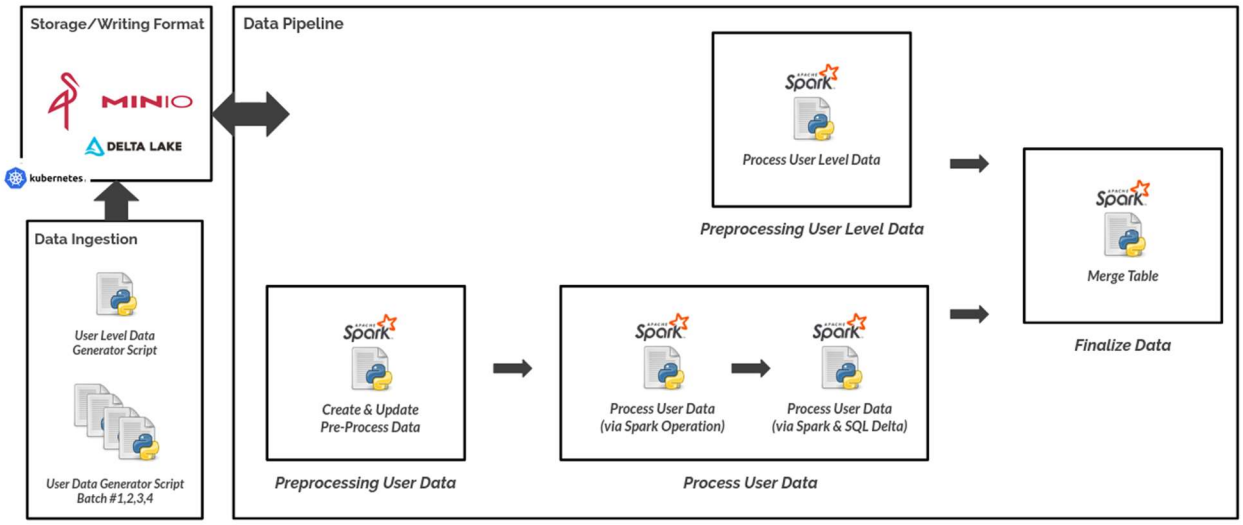
F.3 Chart


Superset
Dashboards
Charts
SQL Lab
Data
Settings
test_pivot_table
Dataset
information_schema.Untitle...
RUN
SAVE
DATA
CUSTOMIZE
Chart type
VISUALIZATION TYPE
Pivot Table v2
Time
Query
COLUMNS
only_e_wallet
ROWS
saved_credits_card
METRICS
COUNT(risk_level)
APPLY METRICS ON
COLUMNS
ROWS
FILTERS
SERIES LIMIT
CELL LIMIT
10000
4 rows
cached
Data
VIEW RESULTS
VIEW SAMPLES
4 rows retrieved
Search
only_e_wallet
saved_credits_card
COUNT(risk_level)
0 1 29
0 0 21
1 1 5
1 0 3

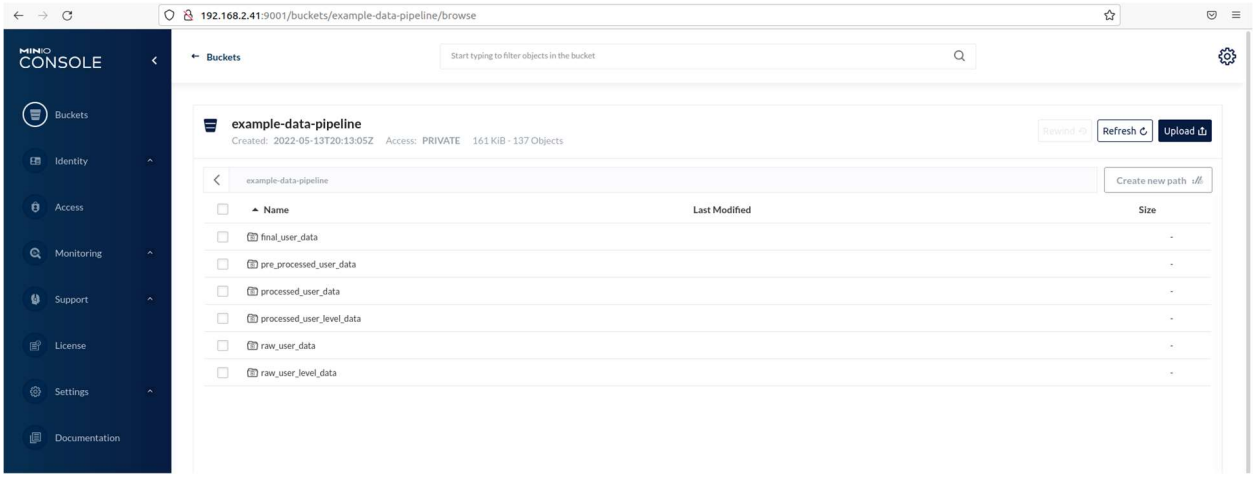

Select a visualization type
All charts
Recommended tags
Popular
ECharts
Advanced-Analytics
Category
Correlation
Distribution
Evolution
Flow
KPI
Map
Part of a Whole
Ranking
Table
Tools
Tags
2D
Search all charts
Big Number with Trendline
Big Number
Table
Pivot Table v2
Time-series Line Chart
Time-series Area Chart
Time-series Bar Chart v2
Time-series Scatter Plot
Pie Chart
Bar Chart
World Map
CANCEL
SELECT

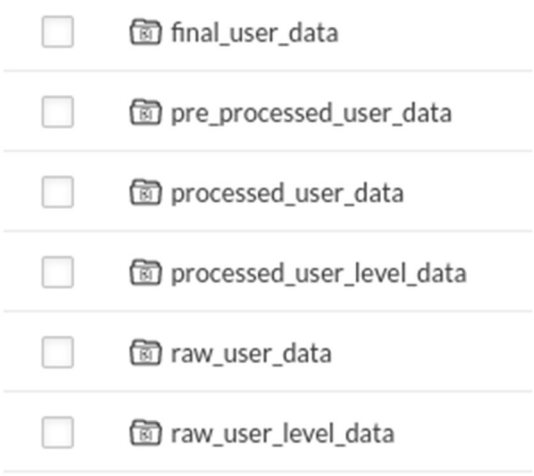
G. Example Data Pipeline

All of the script is implemented in Python. All the script is currently controlled by python executor script. These scripts are expected to be able to integrate further with automation tools though. The data source generator script will generate the data frame of a user and user_level data then stored in Minio in a delta format with the help of PySpark library. The raw data then will be pulled and processed through the remaining script based on the sequence in the executor script and finalize the data for the actual use.



G.1 Delta Table Created in example-data-pipeline Bucket in Minio





G.2 Overview Implementation and Output

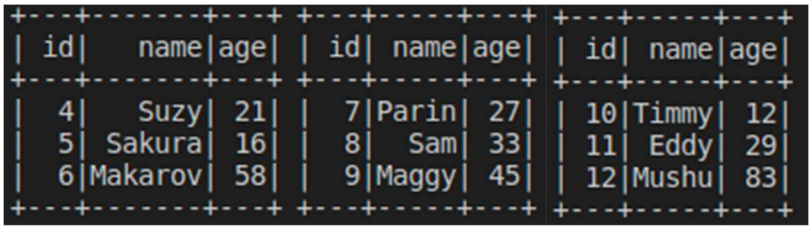
1_1_create_raw_user_data.py: generate (hard-coded) dataframe and create ***raw_user_data*** Delta table from the created dataframe

```
+---+------+---+
| id|  name|age|
+---+------+---+
|  2|Naruto| 18|
|  3| Happy| 35|
|  1|  Krai| 24|
+---+------+---+
```

1_2_create_raw_user_level_data.py: generate (hard-coded) dataframe and create ***raw_user_level_data*** Delta table from the created dataframe.

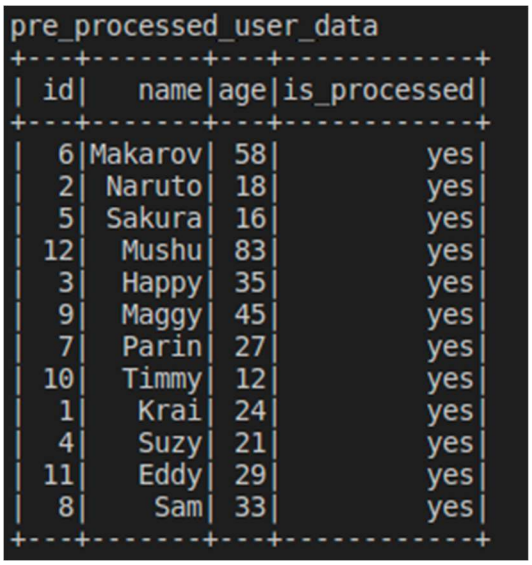
```
raw_user_level_data
+-------+-------+
|user_id|  level|
+-------+-------+
|      5|level C|
|      6|level C|
|      8|level A|
|      9|level A|
|     10|level B|
|      1|level A|
|      4|level A|
|      7|level B|
|     11|level C|
|     12|level A|
|      2|level B|
|      3|level A|
+-------+-------+
```

2_1(2,3)_insert_raw_data.py: generate (hard-coded) dataframe and add new data to the existing ***raw_user_data*** Delta table from the created dataframe.



```
+---+-------+---+ +---+-----+---+ +---+-----+---+
| id|   name|age| | id| name|age| | id| name|age|
+---+-------+---+ +---+-----+---+ +---+-----+---+
|  4|   Suzy| 21| |  7|Parin| 27| | 10|Timmy| 12|
|  5| Sakura| 16| |  8|  Sam| 33| | 11| Eddy| 29|
|  6|Makarov| 58| |  9|Maggy| 45| | 12|Mushu| 83|
+---+-------+---+ +---+-----+---+ +---+-----+---+
```

3_create_preprocess_data.py: create ***pre_process_user_data*** Delta table to insert extra column 'is_processed' for processing step, so that data that has already been processed won't be processed.



```
pre_processed_user_data
+---+-------+---+------------+
| id|   name|age|is_processed|
+---+-------+---+------------+
|  6|Makarov| 58|         yes|
|  2| Naruto| 18|         yes|
|  5| Sakura| 16|         yes|
| 12|  Mushu| 83|         yes|
|  3|  Happy| 35|         yes|
|  9|  Maggy| 45|         yes|
|  7|  Parin| 27|         yes|
| 10|  Timmy| 12|         yes|
|  1|   Krai| 24|         yes|
|  4|   Suzy| 21|         yes|
| 11|   Eddy| 29|         yes|
|  8|    Sam| 33|         yes|
+---+-------+---+------------+
```

4_1_process_user_data.py: get data from ***pre_process_user_data*** Delta table to process using Spark operations and create/update ***processed_user_data*** Delta table

4_2_process_user_data.py: get data from ***pre_process_user_data*** Delta table to process using Spark SQL operation to update ***processed_user_data*** Delta table

```
processed_user_data
+---+-------+---+---------+--------------------+
| id|   name|age|age_group|           age_range|
+---+-------+---+---------+--------------------+
|  6|makarov| 58|      old|older or equal to 40|
|  3|  happy| 35|    adult|        less than 40|
|  5| sakura| 16|      kid|        less than 20|
| 11|   eddy| 29|    adult|        less than 40|
|  2| naruto| 18|      kid|        less than 20|
|  1|   krai| 24|    adult|        less than 40|
|  4|   suzy| 21|    adult|        less than 40|
|  8|    sam| 33|    adult|        less than 40|
|  9|  maggy| 45|      old|older or equal to 40|
|  7|  parin| 27|    adult|        less than 40|
| 10|  timmy| 12|      kid|        less than 20|
| 12|  mushu| 83|      old|older or equal to 40|
+---+-------+---+---------+--------------------+
```

4_3_process_user_level_data.py: get data from ***raw_user_level_data*** Delta table to process using Spark operations and create ***processed_user_level_data*** Delta table

```
processed_user_level_data
+-------+-------+
|user_id|  level|
+-------+-------+
|      8|level a|
|      9|level a|
|      5|level c|
|      6|level c|
|      1|level a|
|     10|level b|
|      7|level b|
|      4|level a|
|     11|level c|
|     12|level a|
|      2|level b|
|      3|level a|
+-------+-------+
```

5_merge_table: merge data from ***processed_user_data*** and data from ***processed_user_level_data*** through Spark operations and create new Delta table ***final_user_data***

final_user_data

id	name	age	age_group	age_range	level
6	makarov	58	old	older or equal to 40	level c
1	krai	24	adult	less than 40	level a
4	suzy	21	adult	less than 40	level a
4	suzy	21	adult	less than 40	level a
8	sam	33	adult	less than 40	level a
9	maggy	45	old	older or equal to 40	level a
7	parin	27	adult	less than 40	level b
10	timmy	12	kid	less than 20	level b
12	mushu	83	old	older or equal to 40	level a
8	sam	33	adult	less than 40	level a
9	maggy	45	old	older or equal to 40	level a
7	parin	27	adult	less than 40	level b
5	sakura	16	kid	less than 20	level c
2	naruto	18	kid	less than 20	level b
3	happy	35	adult	less than 40	level a
5	sakura	16	kid	less than 20	level c
2	naruto	18	kid	less than 20	level b
1	krai	24	adult	less than 40	level a
4	suzy	21	adult	less than 40	level a
1	krai	24	adult	less than 40	level a

only showing top 20 rows

6_get_all_table: present all tables and their data from ***example-data-pipeline*** bucket

7_bonus_describe_table_history: shows the history of the selected Delta Table, currently author is using ***raw_user_data*** Delta table

version	timestamp	userId	userName	operation	operationParameters	job	notebook	clusterId	readVersion	isolationLevel	isBlindAppend	operationMetrics	userMetadata
11	2022-05-14 06:59:18	null	null	WRITE	{mode -> Append, partitionBy -> []}	null	null	null	10	null	true	{numFiles -> 4, numOutputBytes -> 3184, numOutputRows -> 3}	null
10	2022-05-14 06:59:03	null	null	WRITE	{mode -> Append, partitionBy -> []}	null	null	null	9	null	true	{numFiles -> 4, numOutputBytes -> 3175, numOutputRows -> 3}	null
9	2022-05-14 06:58:49	null	null	WRITE	{mode -> Append, partitionBy -> []}	null	null	null	8	null	true	{numFiles -> 4, numOutputBytes -> 3211, numOutputRows -> 3}	null
8	2022-05-14 06:58:35	null	null	WRITE	{mode -> Append, partitionBy -> []}	null	null	null	7	null	true	{numFiles -> 4, numOutputBytes -> 3193, numOutputRows -> 3}	null
7	2022-05-14 05:41:15	null	null	DELETE	{predicate -> []}	null	null	null	6	null	false	{numRemovedFiles -> 4, executionTimeMs -> 196, scanTimeMs -> 195, rewriteTimeMs -> 0}	null
6	2022-05-14 05:40:37	null	null	WRITE	{mode -> Append, partitionBy -> []}	null	null	null	5	null	true	{numFiles -> 4, numOutputBytes -> 3184, numOutputRows -> 3}	null
5	2022-05-14 05:39:50	null	null	DELETE	{predicate -> []}	null	null	null	4	null	false	{numRemovedFiles -> 4, executionTimeMs -> 183, scanTimeMs -> 182, rewriteTimeMs -> 0}	null

Run_this_execution_script: executed all the scripts in sequence according to the following pseudocode

```
import os


os.system("python3 1_2_create_raw_user_level_data")
os.system("python3 1_2_create_raw_user_level_data")

script_to_run =
["1_1_create_raw_user_data.py","2_1_insert_raw_data.py",
"2_3_insert_raw_data.py", "2_3_insert_raw_data.py"]

for script in script_to_run:
    os.system("python3 " + script)
    os.system("python3 3_create_preprocess_data.py")
    os.system("python3 4_1_process_user_data.py")
    os.system("python3 4_2_process_user_data.py")
    os.system("python3 5_merge_table.py")

os.system("python3 6_get_all_table")
```

H. Running Apache Spark (on Bitnami Spark)

Step 1: Get the Spark worker image docker ID

```
kubectl describe pod [ SPARK WORKER NAME ]
```

```
root@satruday:/home/saturday/Desktop/dremio-cloud-tools/charts# kubectl describe pod j-spark-four-worker-0
Name:         j-spark-four-worker-0
Namespace:    default
Priority:     0
Node:         worker-2/192.168.0.5
Start Time:   Tue, 29 Mar 2022 20:09:19 +0700
Labels:       app.kubernetes.io/component=worker
              app.kubernetes.io/instance=j-spark-four
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/name=spark
              controller-revision-hash=j-spark-four-worker-58d5f77746
              helm.sh/chart=spark-5.9.2
              statefulset.kubernetes.io/pod-name=j-spark-four-worker-0
Annotations:  cni.projectcalico.org/containerID: 31756b425f9a77df77ad5be517f2c84db65506252a2d19097d3a1c8a5ed472f2
              cni.projectcalico.org/podIP: 10.42.5.54/32
              cni.projectcalico.org/podIPs: 10.42.5.54/32
Status:       Running
IP:           10.42.5.54
IPs:
  IP:           10.42.5.54
Controlled By:  StatefulSet/j-spark-four-worker
Containers:
  spark-worker:
    Container ID:   docker://fff7af3f6a9fac761cbcd64cacd354ebf75dff6a963e3f0e8f09b42480591964
    Image:          docker.io/bitnami/spark:3.1.2
    Image ID:       docker-pullable://bitnami/spark@sha256:6fe1b9dc053bb1ad4cb05941bed455fa1d256577e4fae5d130b5aee8db57cf22
    Port:           8081/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 07 May 2022 13:28:58 +0700
    Last State:     Terminated
      Reason:       Error
      Exit Code:    255
      Started:      Sat, 07 May 2022 07:46:20 +0700
      Finished:     Sat, 07 May 2022 13:27:14 +0700
```

Step 2: Access Spark worker image

```
kubectl exec -u root -ti [ DOCKER CONTAINER ID ] /bin/bash
```

```
root@tuesday:~# docker exec -u root -ti fff7af3f6a9fac761cbcd64cacd354ebf75dff6a963e3f0e8f09b42480591964 /bin/bash
root@j-spark-four-worker-0:/opt/bitnami/spark#
```

Step 3: Copy the script (python, jar, etc.) to be executed into Spark worker

```
kubectl cp [ YOUR SCRIPT ].py [ SPARK WORKER NAME ]:/tmp/[ YOUR SCRIPT ].py
```

Step 4: Setup the environment for Spark worker to integrate with a Deltalake

```
# update
apt update
yes Y | apt upgrade
```

```
# install necessary package
yes Y | apt install vim
yes Y | apt install wget
```

```
# get delta-core.jar
yes Y | wget https://repo1.maven.org/maven2/io/delta/delta-
core_2.12/1.0.0/delta-core_2.12-1.0.0.jar
```

```
# move delta-core.jar to Apache Spark's classpath
cp delta-core_2.12-1.0.0.jar ../../
cp delta-core_2.12-1.0.0.jar ../../jars/
```

```
# install deltalake library for python
pip install delta-spark==1.0.0
```

```
# copy script to run to Apache Spark execute directory
cp tmp/[ YOUR SCRIPT ].py /opt/bitnami/spark/examples/src/main/python/
```

Step 5: Spark-Submit

```
# get Spark worker name
WORKER=$(kubectl get pods --namespace default | awk '/[SPARK WORKER NAME]/ { print $1 }')
export WORKER


# get Spark master name
MASTER_IP=$(kubectl get svc --namespace default | awk '/[SPARK NAME]-master-svc/ { print $3 }')
export MASTER_IP


# execute spark-submit
kubectl exec -ti $WORKER -- spark-submit --master spark://$MASTER_IP:7077 \
  --class org.apache.spark.examples.SparkPi \
  examples/src/main/python/select_route_plan.py
```

Example

input:

```
root@satruday:~# WORKER=$(kubectl get pods --namespace default | awk '/j-spark-eight-worker-0/ { print $1 }')
root@satruday:~# export WORKER
root@satruday:~#
root@satruday:~# MASTER_IP=$(kubectl get svc --namespace default | awk '/j-spark-eight-master-svc/ { print $3 }')
root@satruday:~# export MASTER_IP
root@satruday:~#
root@satruday:~# kubectl exec -ti $WORKER -- spark-submit --master spark://$MASTER_IP:7077 \
>   --class org.apache.spark.examples.SparkPi \
>   examples/src/main/python/pi.py 100
```

output:

```
22/05/10 06:38:00 INFO TaskSetManager: Finished task 86.0 in stage 0.0 (TID 86) in 150 ms on 10.42.5.51 (executor 2) (87/100)
22/05/10 06:38:00 INFO TaskSetManager: Starting task 95.0 in stage 0.0 (TID 95) (10.42.5.49, executor 0, partition 95, PROCESS_LOCAL, 4465 bytes) taskResourceAssignments Map()
22/05/10 06:38:00 INFO TaskSetManager: Finished task 88.0 in stage 0.0 (TID 88) in 105 ms on 10.42.5.49 (executor 0) (88/100)
22/05/10 06:38:00 INFO TaskSetManager: Starting task 96.0 in stage 0.0 (TID 96) (10.42.4.17, executor 1, partition 96, PROCESS_LOCAL, 4465 bytes) taskResourceAssignments Map()
22/05/10 06:38:00 INFO TaskSetManager: Finished task 90.0 in stage 0.0 (TID 90) in 117 ms on 10.42.4.17 (executor 1) (89/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 89.0 in stage 0.0 (TID 89) in 135 ms on 10.42.4.18 (executor 7) (90/100)
22/05/10 06:38:00 INFO TaskSetManager: Starting task 97.0 in stage 0.0 (TID 97) (10.42.4.18, executor 7, partition 97, PROCESS_LOCAL, 4465 bytes) taskResourceAssignments Map()
22/05/10 06:38:00 INFO TaskSetManager: Starting task 98.0 in stage 0.0 (TID 98) (10.42.3.59, executor 4, partition 98, PROCESS_LOCAL, 4465 bytes) taskResourceAssignments Map()
22/05/10 06:38:00 INFO TaskSetManager: Finished task 87.0 in stage 0.0 (TID 87) in 152 ms on 10.42.3.59 (executor 4) (91/100)
22/05/10 06:38:00 INFO TaskSetManager: Starting task 99.0 in stage 0.0 (TID 99) (10.42.2.10, executor 5, partition 99, PROCESS_LOCAL, 4465 bytes) taskResourceAssignments Map()
22/05/10 06:38:00 INFO TaskSetManager: Finished task 92.0 in stage 0.0 (TID 92) in 107 ms on 10.42.2.10 (executor 5) (92/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 91.0 in stage 0.0 (TID 91) in 113 ms on 10.42.2.17 (executor 3) (93/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 93.0 in stage 0.0 (TID 93) in 116 ms on 10.42.3.55 (executor 6) (94/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 94.0 in stage 0.0 (TID 94) in 104 ms on 10.42.5.51 (executor 2) (95/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 95.0 in stage 0.0 (TID 95) in 104 ms on 10.42.5.49 (executor 0) (96/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 96.0 in stage 0.0 (TID 96) in 106 ms on 10.42.4.17 (executor 1) (97/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 97.0 in stage 0.0 (TID 97) in 105 ms on 10.42.4.18 (executor 7) (98/100)
22/05/10 06:38:00 INFO TaskSetManager: Finished task 98.0 in stage 0.0 (TID 98) in 108 ms on 10.42.3.59 (executor 4) (99/100)
22/05/10 06:38:01 INFO TaskSetManager: Finished task 99.0 in stage 0.0 (TID 99) in 172 ms on 10.42.2.10 (executor 5) (100/100)
22/05/10 06:38:01 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
22/05/10 06:38:01 INFO DAGScheduler: ResultStage 0 (reduce at /opt/bitnami/spark/examples/src/main/python/pi.py:42) finished in 4.129 s
22/05/10 06:38:01 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
22/05/10 06:38:01 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
22/05/10 06:38:01 INFO DAGScheduler: Job 0 finished: reduce at /opt/bitnami/spark/examples/src/main/python/pi.py:42, took 4.227746 s
Pi is roughly 3.144520
22/05/10 06:38:01 INFO SparkUI: Stopped Spark web UI at http://j-spark-eight-worker-0.j-spark-eight-headless.default.svc.cluster.local:4040
22/05/10 06:38:01 INFO StandaloneSchedulerBackend: Shutting down all executors
22/05/10 06:38:01 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
22/05/10 06:38:01 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
22/05/10 06:38:01 INFO MemoryStore: MemoryStore cleared
22/05/10 06:38:01 INFO BlockManager: BlockManager stopped
22/05/10 06:38:01 INFO BlockManagerMaster: BlockManagerMaster stopped
22/05/10 06:38:01 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
22/05/10 06:38:01 INFO SparkContext: Successfully stopped SparkContext
22/05/10 06:38:02 INFO ShutdownHookManager: Shutdown hook called
22/05/10 06:38:02 INFO ShutdownHookManager: Deleting directory /tmp/spark-8e8ae44d-9355-45e7-b882-d71fd938f6b5/pyspark-b00e0618-7500-4cd3-b8dc-ea9b7b869d22
22/05/10 06:38:02 INFO ShutdownHookManager: Deleting directory /tmp/spark-0070d823-92c5-4180-868e-b2ce1df00b27
22/05/10 06:38:02 INFO ShutdownHookManager: Deleting directory /tmp/spark-8e8ae44d-9355-45e7-b882-d71fd938f6b5
root@satruday:~#
```