Project: Building an Azure Data Lake for Bike Share Data Analytics

Jun 21 2024

Dahi Nemutlu

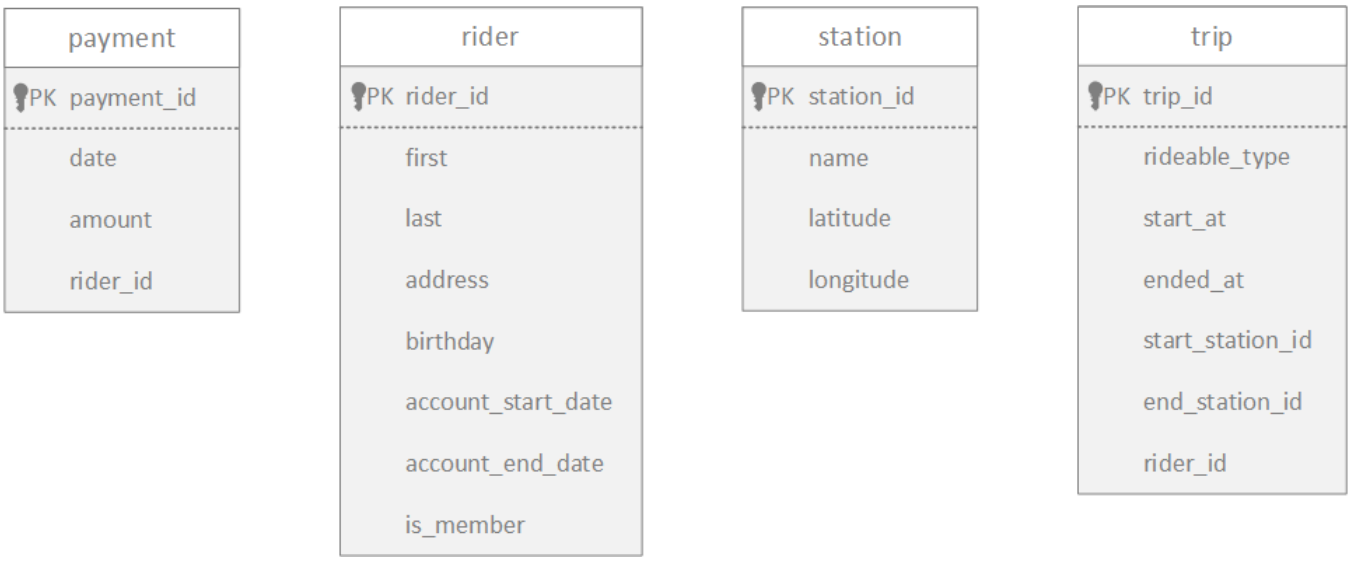
Contents

1. Project Overview

In this project, we'll build a data lake solution for Divvy bikeshare.

Divvy is a bike sharing program in Chicago, Illinois USA that allows riders to purchase a pass at a kiosk or use a mobile application to unlock a bike at stations around the city and use the bike for a specified amount of time. The bikes can be returned to the same station or to another station. The City of Chicago makes the anonymized bike trip data publicly available for projects like this where we can analyze the data.

The dataset looks like this:



The goal of this project is to develop a data lake solution using Azure Databricks using a lake house architecture. We will:

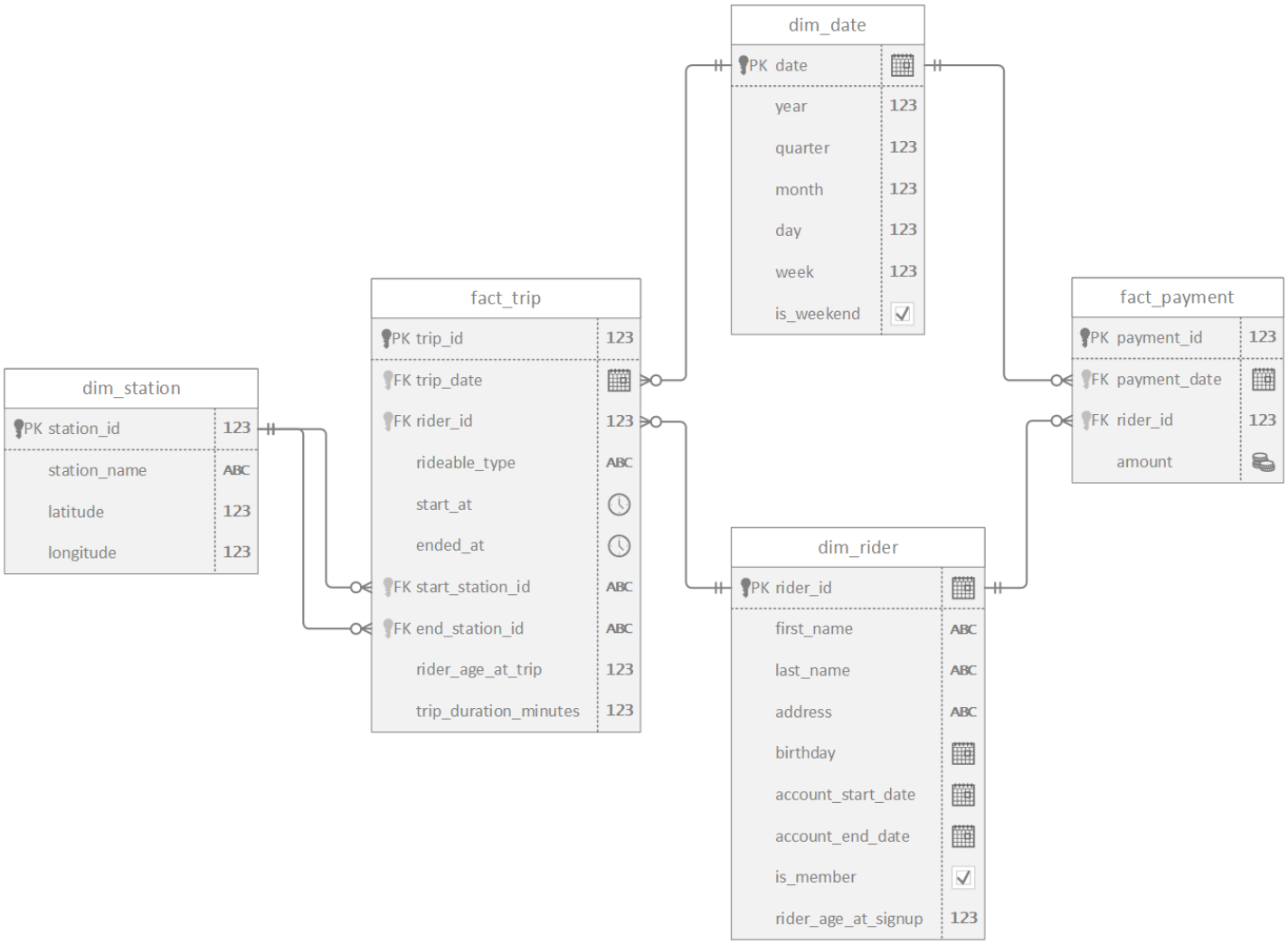
- Design a star schema based on the business outcomes listed below;
- Import the data into Azure Databricks using Delta Lake to create a Bronze data store;
- Create a silver data store in Delta Lake tables;
- Transform the data into the star schema for a Gold data store.

The business outcomes we are designing for are as follows:

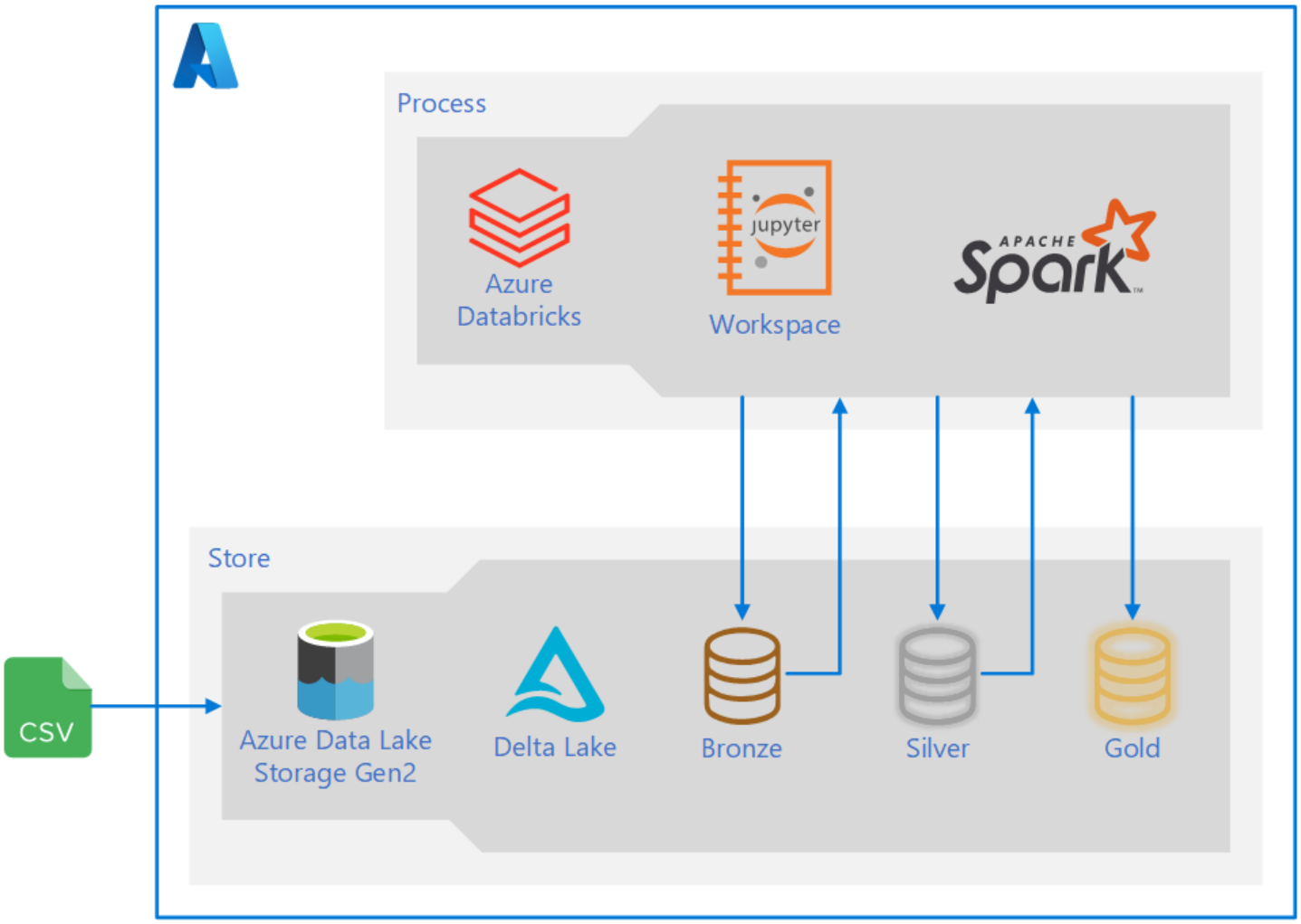
1. Analyze how much time is spent per ride
    - Based on date and time factors such as day of week and time of day
    - Based on which station is the starting and / or ending station
    - Based on age of the rider at time of the ride
    - Based on whether the rider is a member or a casual rider
2. Analyze how much money is spent
    - Per month, quarter, year

- Per member, based on the age of the rider at account start

Based on the provided business outcomes above, we will create the following star schema using fact and dimension tables.

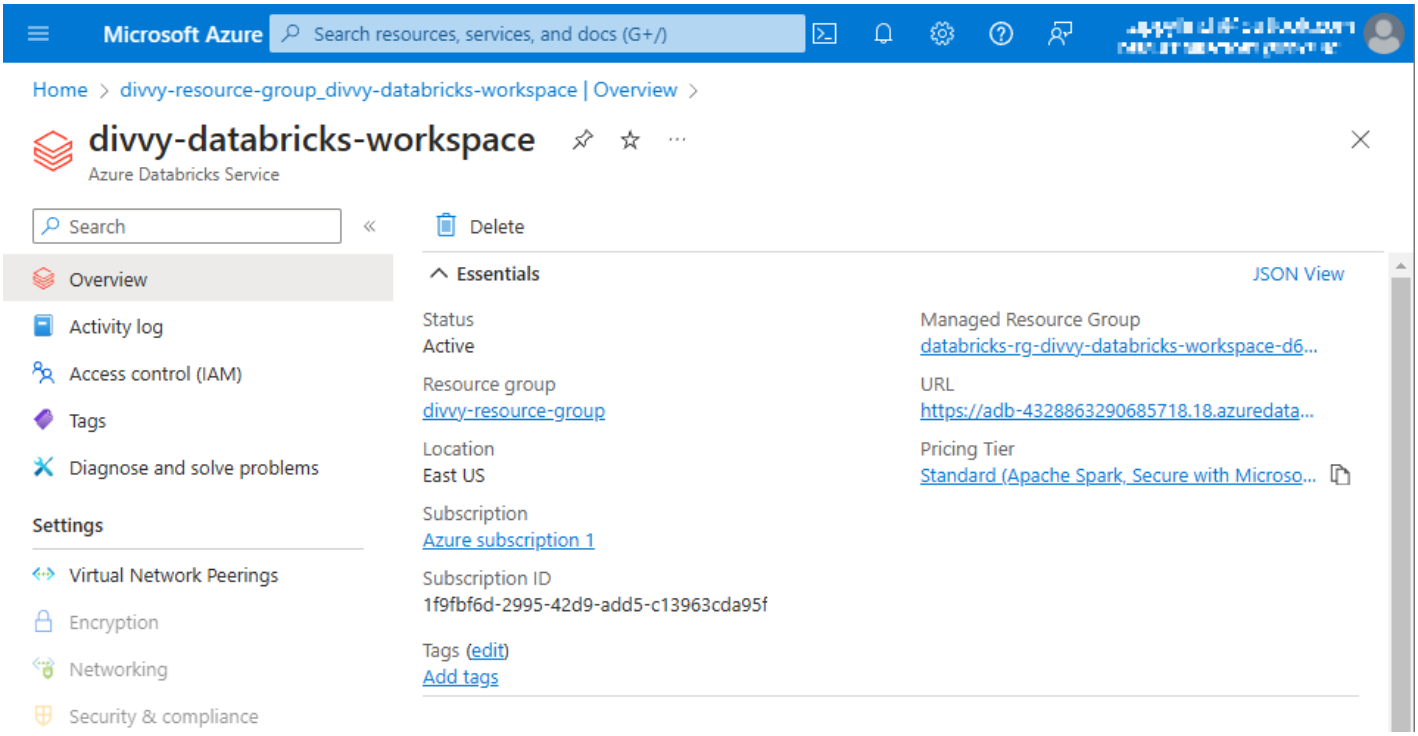


With this project, we'll build a data lake solution using Azure Databricks with a lake house architecture as follows.

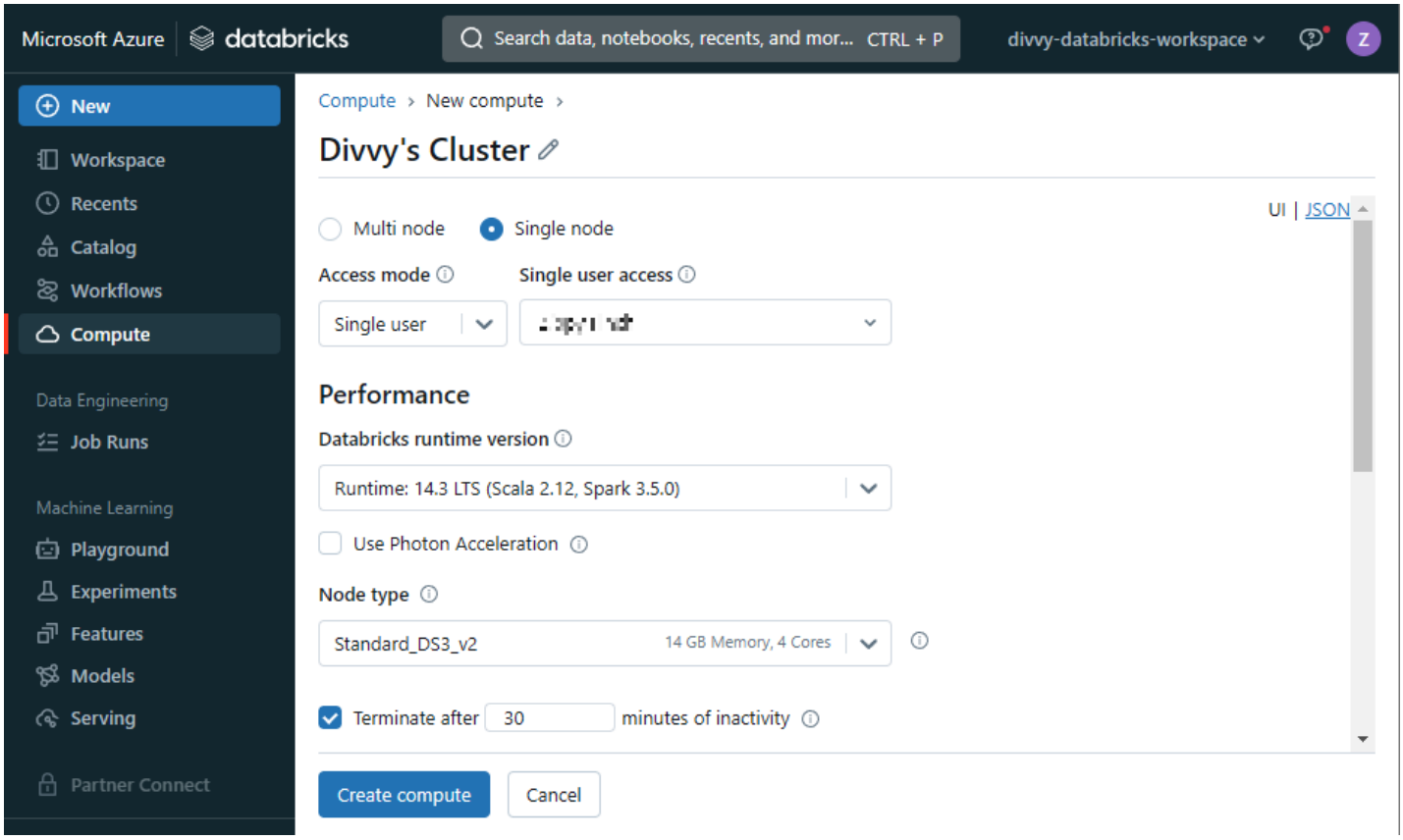

Process
Azure Databricks
jupyter
Workspace
APACHE Spark
Store
CSV
Azure Data Lake Storage Gen2
Delta Lake
Bronze
Silver
Gold

2. Create Azure Resources

2.1. Create an Azure Databricks workspace

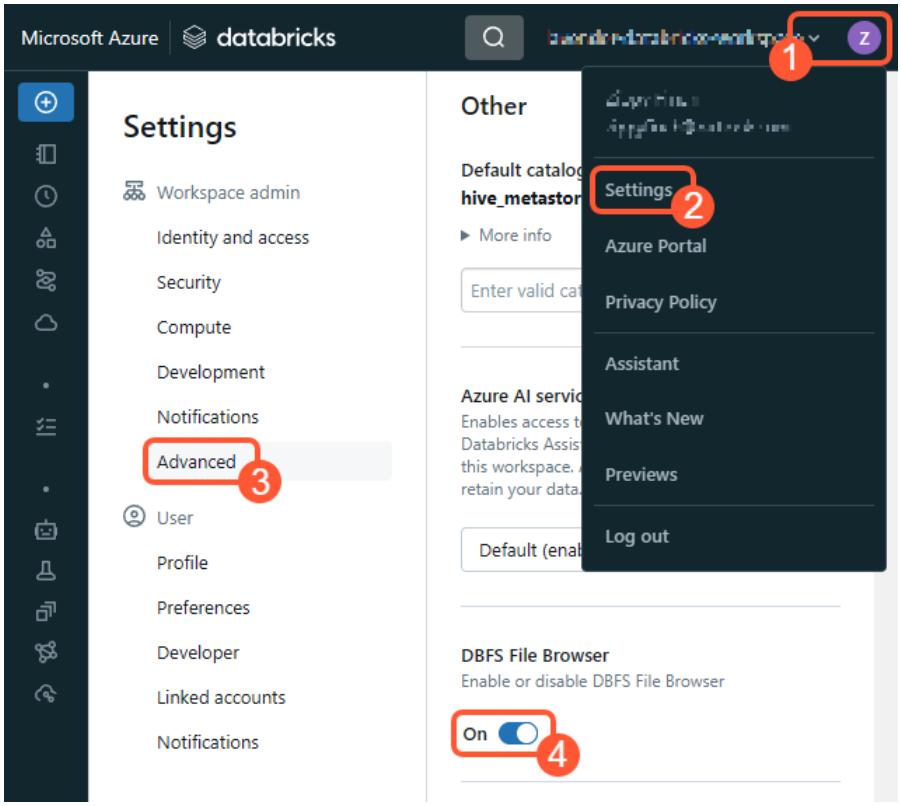


2.2. Create compute to run workloads

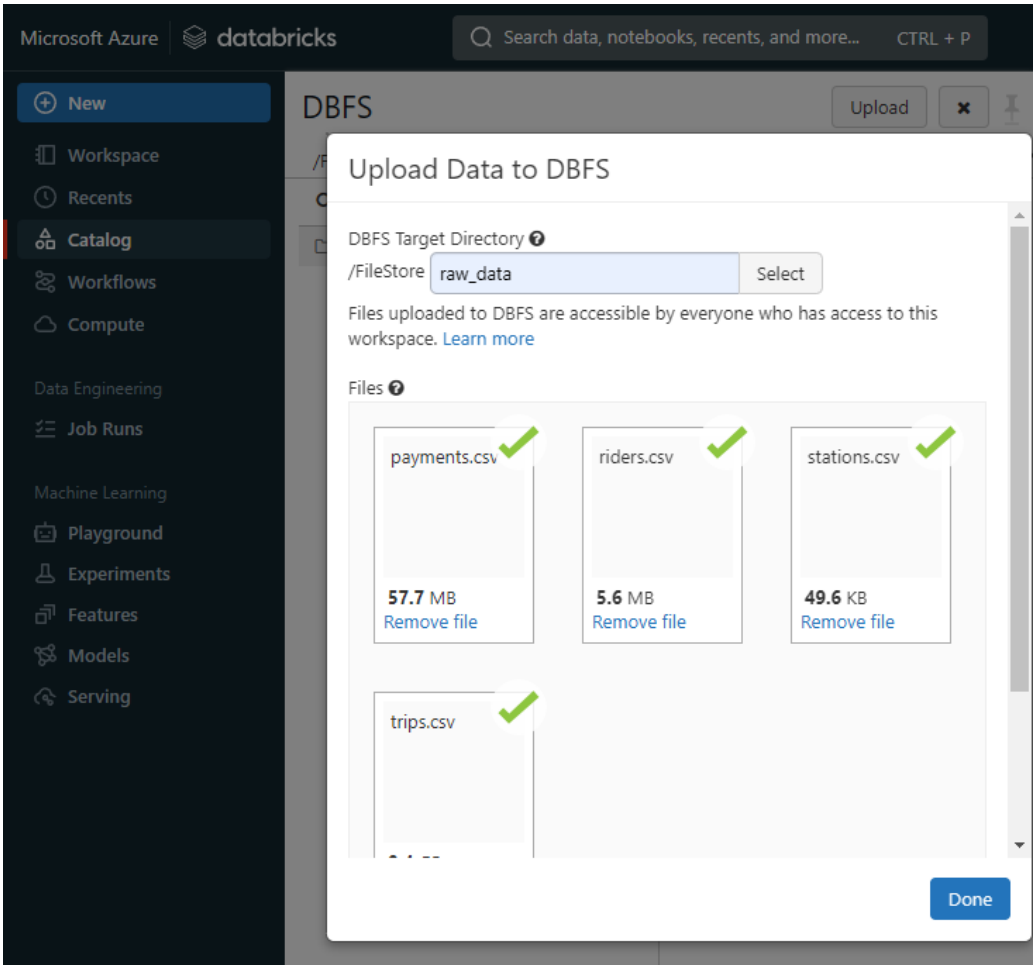

Microsoft Azure
databricks
Search data, notebooks, recents, and mor... CTRL + P
divvy-databricks-workspace
Z
New
Workspace
Recents
Catalog
Workflows
Compute
Data Engineering
Job Runs
Machine Learning
Playground
Experiments
Features
Models
Serving
Partner Connect
Compute › New compute ›
Divvy's Cluster
UI | JSON
Multi node
Single node
Access mode
Single user access
Single user
Performance
Databricks runtime version
Runtime: 14.3 LTS (Scala 2.12, Spark 3.5.0)
Use Photon Acceleration
Node type
Standard_DS3_v2
14 GB Memory, 4 Cores
Terminate after
30
minutes of inactivity
Create compute
Cancel

3. Import Data (Bronze Store)

3.1. Enable DBFS File Browser



3.2. Upload Files into Databricks



3.3. Create a notebook and ingest the data to the bronze layer

```
from pyspark.sql.functions import unix_timestamp, col, floor, months_between, expr
import pyspark.sql.functions as F

# Load CSV files into Spark DataFrames
df_payment = spark.read.format("csv").option("inferSchema", "true").option("header",
"false").option("sep", ",").load("/FileStore/raw_data/payments.csv")
df_rider = spark.read.format("csv").option("inferSchema", "true").option("header",
"false").option("sep", ",").load("/FileStore/raw_data/riders.csv")
df_station = spark.read.format("csv").option("inferSchema", "true").option("header",
"false").option("sep", ",").load("/FileStore/raw_data/stations.csv")
df_trip = spark.read.format("csv").option("inferSchema", "true").option("header",
"false").option("sep", ",").load("/FileStore/raw_data/trips.csv")

# Write DataFrames to Delta Lake format for bronze layer storage
df_payment.write.format("delta").mode("overwrite").save("/delta/bronze/payment")
df_rider.write.format("delta").mode("overwrite").save("/delta/bronze/rider")
df_station.write.format("delta").mode("overwrite").save("/delta/bronze/station")
```

```
df_trip.write.format("delta").mode("overwrite").save("/delta/bronze/trip")
```

4. Refine Data (Silver Store)

```
# Rename columns

df_payment = (
    df_payment.withColumnRenamed("_c0", "payment_id")
    .withColumnRenamed("_c1", "payment_date")
    .withColumnRenamed("_c2", "amount")
    .withColumnRenamed("_c3", "rider_id")
)

df_rider = (
    df_rider.withColumnRenamed("_c0", "rider_id")
    .withColumnRenamed("_c1", "first_name")
    .withColumnRenamed("_c2", "last_name")
    .withColumnRenamed("_c3", "address")
    .withColumnRenamed("_c4", "birthday")
    .withColumnRenamed("_c5", "account_start_date")
    .withColumnRenamed("_c6", "account_end_date")
    .withColumnRenamed("_c7", "is_member")
)

df_station = (
    df_station.withColumnRenamed("_c0", "station_id")
    .withColumnRenamed("_c1", "station_name")
    .withColumnRenamed("_c2", "latitude")
    .withColumnRenamed("_c3", "longitude")
)

df_trip = (
    df_trip.withColumnRenamed("_c0", "trip_id")
    .withColumnRenamed("_c1", "rideable_type")
    .withColumnRenamed("_c2", "start_at")
    .withColumnRenamed("_c3", "ended_at")
    .withColumnRenamed("_c4", "start_station_id")
    .withColumnRenamed("_c5", "end_station_id")
    .withColumnRenamed("_c6", "rider_id")
)

# Deduplicate
df_payment = df_payment.dropDuplicates(df_payment.columns)
df_rider = df_rider.dropDuplicates(df_rider.columns)
df_station = df_station.dropDuplicates(df_station.columns)
```

```
df_trip = df_trip.dropDuplicates(df_trip.columns)

# Write DataFrames to Delta Lake format for silver layer storage
df_payment.write.format("delta").mode("overwrite").save("/delta/silver/payment")
df_rider.write.format("delta").mode("overwrite").save("/delta/silver/rider")
df_station.write.format("delta").mode("overwrite").save("/delta/silver/station")
df_trip.write.format("delta").mode("overwrite").save("/delta/silver/trip")
```

5. Transform to Star Schema (Gold Store)

```
# Add a new column 'trip_duration_minutes' to 'df_trip' with the difference in minutes between
'ended_at' and 'start_at'
df_trip = df_trip.withColumn(
    "trip_duration_minutes",
    F.round((unix_timestamp(col("ended_at")) - unix_timestamp(col("start_at"))) / 60),
)

# Join 'df_trip' with 'df_rider' on 'rider_id' and add 'rider_age_at_trip' column to 'df_trip'
df_trip = (
    df_trip.join(df_rider.select("rider_id", "birthday"), on="rider_id", how="inner")
    .withColumn("trip_date", col("start_at").cast("date"))
    .withColumn(
        "rider_age_at_trip",
        floor(months_between(col("trip_date"), col("birthday")) / 12),
    )
    .drop("birthday")  # Drop 'birthday' as no longer needed
)

# Add a new column 'rider_age_at_signup' to 'df_rider' to calculate the rider's age at the time
of signup
df_rider = df_rider.withColumn(
    "rider_age_at_signup",
    floor(months_between(col("account_start_date"), col("birthday")) / 12),
)

# Create Date Dimension DataFrame

# Get minimum and maximum dates from 'df_trip' and 'df_payment'
trip_min_date, trip_max_date = df_trip.agg(
    F.min("trip_date"), F.max("trip_date")
).first()
payment_min_date, payment_max_date = df_payment.agg(
    F.min("payment_date"), F.max("payment_date")
).first()
```

```
# Determine overall min and max dates
min_date = min(trip_min_date, payment_min_date)
max_date = max(trip_max_date, payment_max_date)

# Convert datetime.date objects to strings in 'YYYY-MM-DD' format
min_date_str = min_date.strftime("%Y-%m-%d")
max_date_str = max_date.strftime("%Y-%m-%d")

# Generate date range DataFrame
df_date = (
    spark.range(0, (max_date - min_date).days + 1, 1)
    .withColumn("id", col("id").cast("int"))
    .withColumn("date", expr(f"date_add('{min_date_str}', id)"))
)

# Add other date attributes
df_date = (
    df_date.withColumn("year", F.year(col("date")))
    .withColumn("quarter", F.quarter(col("date")))
    .withColumn("month", F.month(col("date")))
    .withColumn("day", F.dayofmonth(col("date")))
    .withColumn("week", F.weekofyear(col("date")))
    .withColumn("is_weekend", F.dayofweek(col("date")).isin([1, 7]).cast("boolean"))
)

# Write DataFrames to Delta Lake format, and save as a Delta tables for gold layer
df_date.write.format("delta").mode("overwrite").saveAsTable("gold_dim_date")
df_station.write.format("delta").mode("overwrite").saveAsTable("gold_dim_station")
df_rider.write.format("delta").mode("overwrite").saveAsTable("gold_dim_rider")
df_trip.write.format("delta").mode("overwrite").saveAsTable("gold_fact_trip")
df_payment.write.format("delta").mode("overwrite").saveAsTable("gold_fact_payment")

# Display the first 5 rows from each Delta table in the gold layer for verification

print("gold_dim_date:")
spark.read.table("gold_dim_date").show(5)

print("gold_dim_station:")
spark.read.table("gold_dim_station").show(5)

print("gold_dim_rider:")
spark.read.table("gold_dim_rider").show(5)

print("gold_fact_trip:")
spark.read.table("gold_fact_trip").show(5)
```

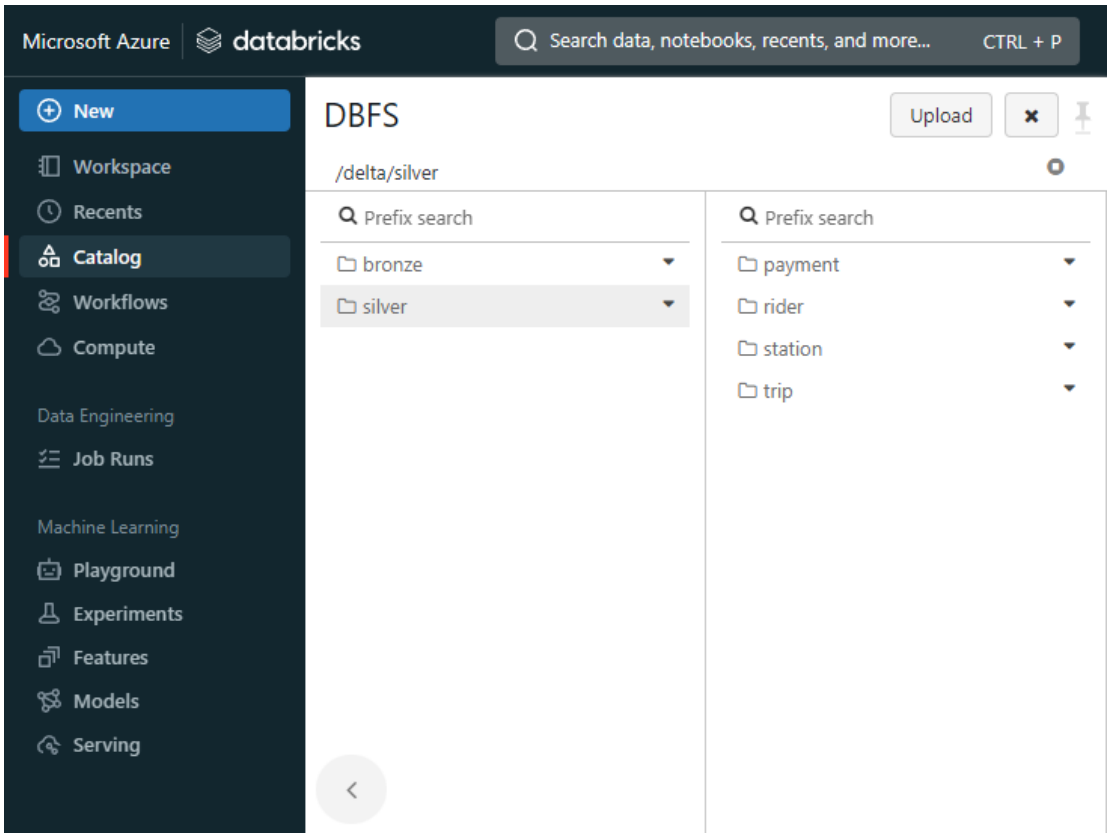
```
print("gold_fact_payment:")
spark.read.table("gold_fact_payment").show(5)
```

```
%sql
-- Uncomment if you need to drop Delta tables
--DROP TABLE IF EXISTS gold_dim_date;
--DROP TABLE IF EXISTS gold_dim_station;
--DROP TABLE IF EXISTS gold_dim_rider;
--DROP TABLE IF EXISTS gold_fact_trip;
--DROP TABLE IF EXISTS gold_fact_payment;
```

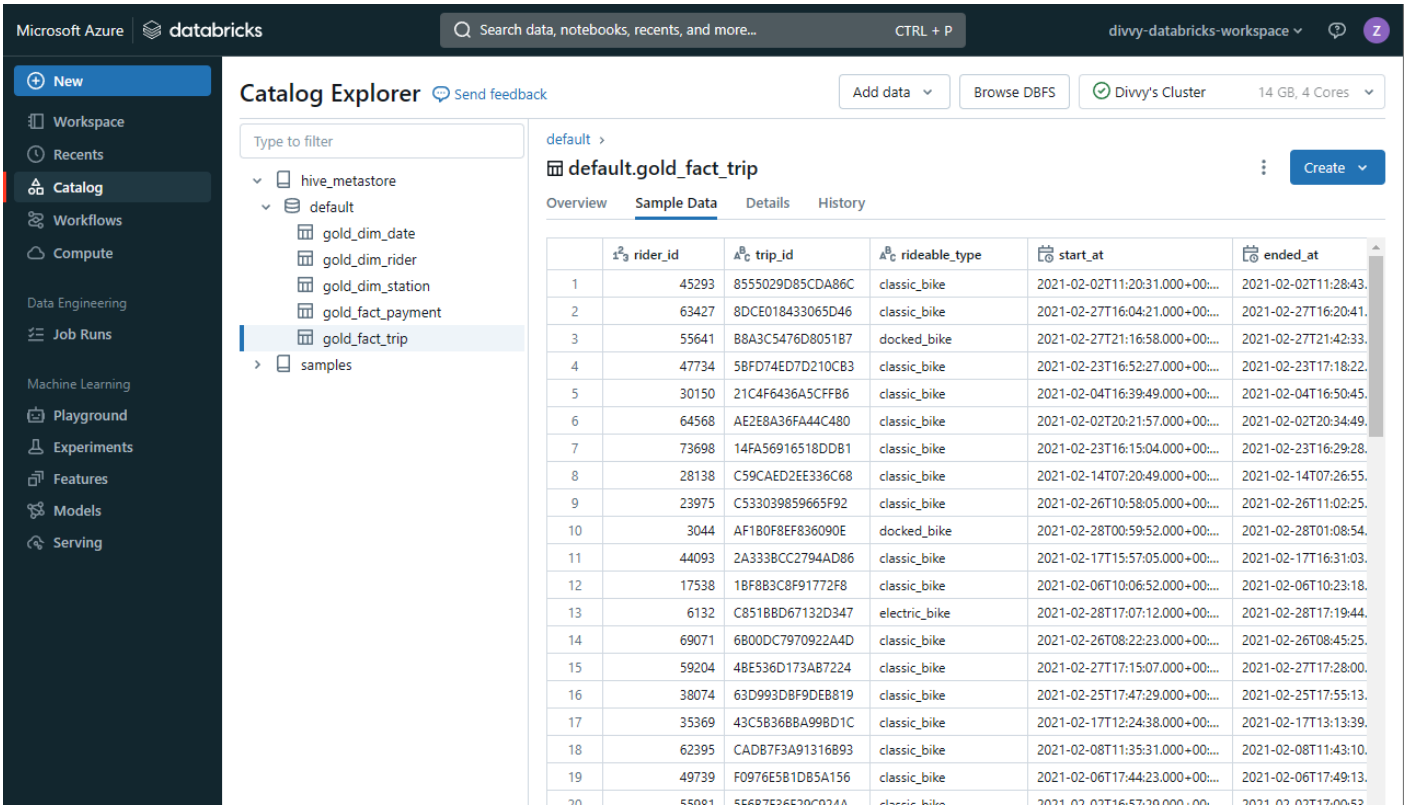
```
# Uncomment if you need to remove the entire '/delta' directory and all its contents
#dbutils.fs.rm("/delta", recurse=True)
```

6. Verify Delta Lake Files and Tables

Check Delta Lake files in DBFS explorer:



Check Delta Lake tables:



	rider_id	trip_id	rideable_type	start_at	ended_at
1	45293	8555029D85CDA86C	classic_bike	2021-02-02T11:20:31.000+00:...	2021-02-02T11:28:43.
2	63427	8DCE018433065D46	classic_bike	2021-02-27T16:04:21.000+00:...	2021-02-27T16:20:41.
3	55641	B8A3C5476D8051B7	docked_bike	2021-02-27T21:16:58.000+00:...	2021-02-27T21:42:33.
4	47734	5BFD74ED7D210CB3	classic_bike	2021-02-23T16:52:27.000+00:...	2021-02-23T17:18:22.
5	30150	21C4F6436A5CFFB6	classic_bike	2021-02-04T16:39:49.000+00:...	2021-02-04T16:50:45.
6	64568	AE2E8A36FA44C480	classic_bike	2021-02-02T20:21:57.000+00:...	2021-02-02T20:34:49.
7	73698	14FA56916518DDB1	classic_bike	2021-02-23T16:15:04.000+00:...	2021-02-23T16:29:28.
8	28138	C59CAED2EE336C68	classic_bike	2021-02-14T07:20:49.000+00:...	2021-02-14T07:26:55.
9	23975	C533039859665F92	classic_bike	2021-02-26T10:58:05.000+00:...	2021-02-26T11:02:25.
10	3044	AF1B0F8EF836090E	docked_bike	2021-02-28T00:59:52.000+00:...	2021-02-28T01:08:54.
11	44093	2A333BCC2794AD86	classic_bike	2021-02-17T15:57:05.000+00:...	2021-02-17T16:31:03.
12	17538	1BF8B3C8F91772F8	classic_bike	2021-02-06T10:06:52.000+00:...	2021-02-06T10:23:18.
13	6132	C851BBD67132D347	electric_bike	2021-02-28T17:07:12.000+00:...	2021-02-28T17:19:44.
14	69071	6B00DC7970922A4D	classic_bike	2021-02-26T08:22:23.000+00:...	2021-02-26T08:45:25.
15	59204	4BE536D173AB7224	classic_bike	2021-02-27T17:15:07.000+00:...	2021-02-27T17:28:00.
16	38074	63D993DBF9DEB819	classic_bike	2021-02-25T17:47:29.000+00:...	2021-02-25T17:55:13.
17	35369	43C5B36BBA99BD1C	classic_bike	2021-02-17T12:24:38.000+00:...	2021-02-17T13:13:39.
18	62395	CADB7F3A91316B93	classic_bike	2021-02-08T11:35:31.000+00:...	2021-02-08T11:43:10.
19	49739	F0976E5B1DB5A156	classic_bike	2021-02-06T17:44:23.000+00:...	2021-02-06T17:49:13.