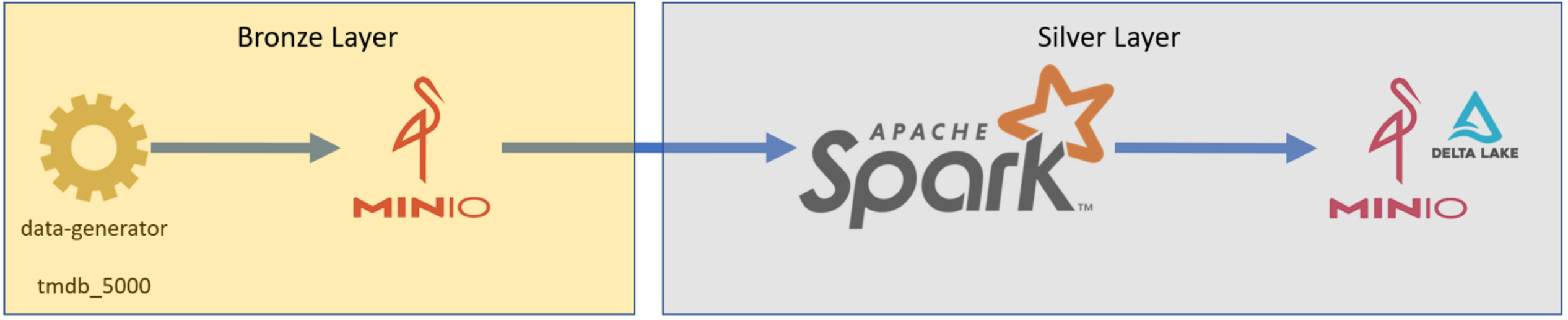
Final Project-4: Airflow



Proje Açıklaması:
Bize verilen docker ortamında, Tmdb_5000_movies.csv ve Tmdb_5000_credits.csv dosyalarını Data-generator kullanarak Minioda ki TMDB-BRONZE bucketine aktarmak. Sonra aktarılan ham verileri Apache-Spark kullanarak json formatındaki veriyi işlemek ve düzgün formattaki istenenen tablolara dönüştürmek ve de bu tabloları delta formatında TMDB-SILVER bucketine aktarmak.

Yukarıda Yapılan bu işlemleri Airflow kullanarak schedule etmek.

Apache Airflow

Tables

- *movies*
- *cast*
- *crew*
- *genres*
- *keywords*
- *production_companies*
- *production_countries*
- *spoken_languages*

Project Keywords

1. Flatten files
2. Upsert
3. Delta Lake
4. Normalization
5. Airflow
6. Spark

docker-compose.yaml

Bize verilen docker ortamında kullanacağım servisler şunlardır:

- Apache-Spark
- Airflow
- Minio

Proje kullanmak için Spark_client servisinde bazı değişiklikler uygulamam gerekliydi .yaml dosyasında şu değişiklikleri yaptım:

- Gerekli kütüphaneleri yükledim

```
pip install findspark && pip install delta-spark
```

- Bu kısımda airflowla spark servisine ssh ile erişmek için kullanacağım ssh_train kullanıcısını tanımladım

```
useradd -rm -d /home/ssh_train -s /bin/bash -g root -G sudo -u 1000 ssh_train && echo 'ssh_train:*****' | chpasswd
```

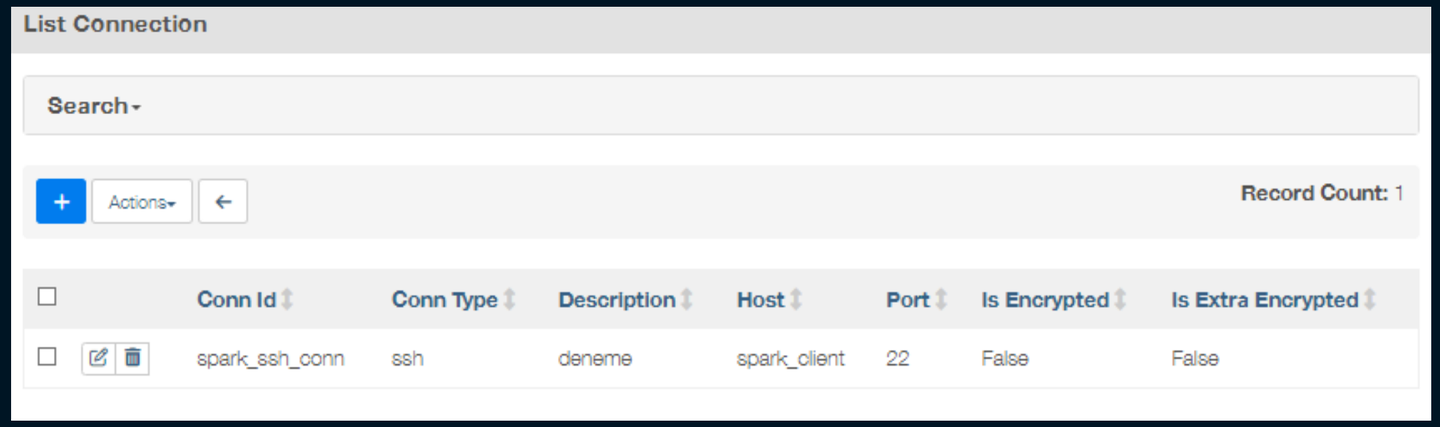
- chown ile /opt/spark/history dizin sahipliğini ssh_train kullanıcısını yetkilendirdim ve chmod -R 777 /opt/spark/history ile okuma yazma yetkisini açık hale getirdim

```
chown -R ssh_train /opt/spark/history && su ssh_train -c 'chmod -R 777 /opt/spark/history
```

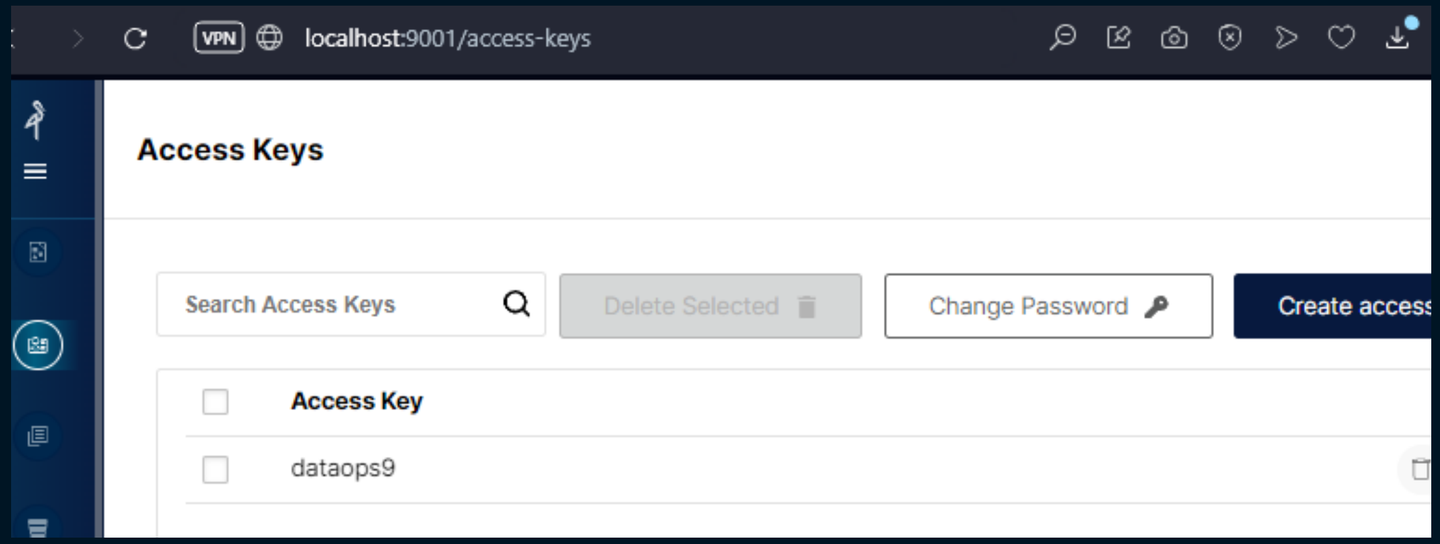
- ssh ile erişimi açmak için service ssh start komutunu kullandım

```
service ssh start
```

```
command: bash -c "pip install findspark && pip install delta-spark && useradd -rm -d /home/ssh_train -s /bin/bash -g root -G sudo
-u 1000 ssh_train && echo 'ssh_train:Ankara06' | chpasswd && chown -R ssh_train /opt/spark/history && su ssh_train -c 'chmod -R
777 /opt/spark/history' && service ssh start && sleep infinity"
```



Aynı şekilde Airflow da spark_clienta erişmek için kullandığım connection ayarlarını belirledim.



Minio servisine erişim sağlamak için acces key oluşturdum.

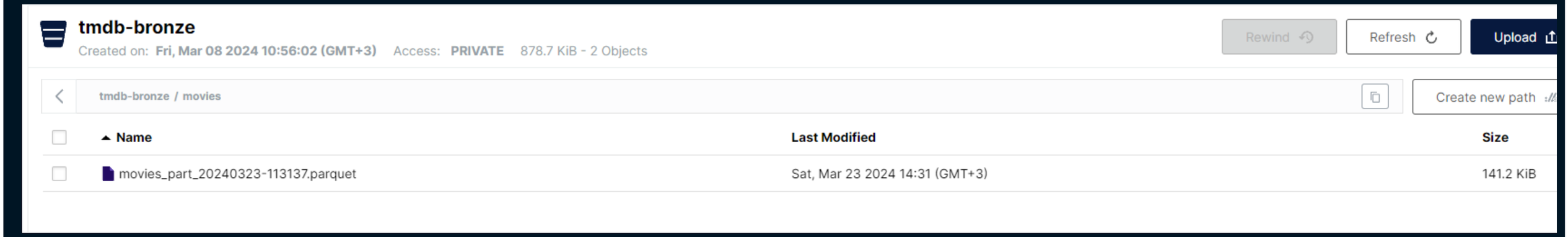
DATA GENERATOR KURULUMU

Data-generator ile verileri minio göndereceğimiz için data-generatoru spark_client servisine aşağıdaki talimatları uygulayarak yükledim.

- docker exec -it spark_client bash
- git clone https://github.com/erkansirin78/data-generator.git
- erkan@ubuntu:~$ python3 -m pip install virtualenv
- erkan@ubuntu:~$ cd data-generator/
- erkan@ubuntu:~/data-generator$ python3 -m virtualenv datagen
- erkan@ubuntu:~/data-generator$ source datagen/bin/activate
- (datagen) erkan@ubuntu:~/data-generator$ pip install -r requirements.txt

Burada doğru şekilde generate edilip edilmeyeceğini görmek bir deneme yaptım. başarılı şekilde Tmdb-bronze bucketine loglar eklendi.

```
python dataframe_to_s3.py -buc tmdb-bronze \ bucket bilgileri
-k movies/movies_part \ generate edilen verinin aktarılacağı konum
-aki dataops9 -sac ****** \ minio için gerekli olan acces key ve parolası
-eu http://minio:9000 \ belirlenen endpoint url
-i /dataops/tmdb_5000_movies_and_credits/tmdb_5000_movies.csv \ aktarılacak olan dosyanın konumu
-ofp True \ output_format_parquet dosyanın parquet formatında yazılması için
-z 500 \ dosyada tek seferde kaç satır verinin işleneceği
-b 0.1 \ her satır okunduktan sonra ne kadar bekleme süresi
-oh True \output header, loglarda header olup olmayacağı
```



```
cast_schema = StructType([
    StructField("movie_id", StringType(), nullable=True),
    StructField("title", StringType(), nullable=True),
    StructField("cast_id", IntegerType(), nullable=True),
    StructField("character", StringType(), nullable=True),
    StructField("credit_id", StringType(), nullable=True),
    StructField("gender", IntegerType(), nullable=True),
    StructField("id", IntegerType(), nullable=True),
    StructField("name", StringType(), nullable=True)
])
cast_table = spark.createDataFrame([], cast_schema)
cast_table.write.format("delta").mode("overwrite").save('s3a://tmdb-silver/credits_cast')
```

- İlk olarak, cast_schema adında bir StructType nesnesi tanımlanır. Bu nesne, veri setindeki sütunların isimlerini ve veri tiplerini belirtir. Her StructField, sütun adını, veri tipini ve opsiyonel olarak nullable özelliğini içerir.
- Ardından, spark.createDataFrame([], cast_schema) komutuyla boş bir DataFrame oluşturulur. Bu DataFrame, belirtilen şemaya uygun olarak sütunları içerir, ancak hiç satır içermez.
- Son olarak, cast_table.write.format("delta").mode("overwrite").save('s3a://tmdb-silver/credits_cast') komutuyla oluşturulan DataFrame, Delta formatında belirtilen yol altına kaydedilir. Burada 's3a://tmdb-silver/credits_cast' hedef yol olarak belirtilmiştir.

- Bir jupyter notebook dosyası açarak tmdb-bronze bucketinda işlediğim verileri tmdb-silver bucketina upsert kullanarak aktaracağım için tmdb-silver bucketinda verilerin tipine uygun, boş delta formatında tablolar oluşturdum.
- Yukarıda Cast tablosu için oluşturduğum örnek bulunuyor ve diğer crew, movies, genres, keywords, production_companies, production_countries ve spoken_languages benzer şekilde oluşturdum

Bronze bucketi altındaki credits verilerinin işlenip silver bucketine aktarılması

```
json_cast = ArrayType(StructType([
  StructField("cast_id", IntegerType()),
  StructField("character", StringType()),
  StructField("credit_id", StringType()),
  StructField("gender", IntegerType()),
  StructField("id", IntegerType()),
  StructField("name", StringType())
]))

df_credits1 = df_credits1.withColumn("cast", from_json(col("cast"), json_cast))

credits_cast = df_credits1.select("movie_id", "title", explode_outer("cast").alias("cast"))
credits_cast = credits_cast.select("movie_id", "title", "cast.cast_id", "cast.character", "cast.credit_id", "cast.gender",
"cast.id", "cast.name")
credits_cast = credits_cast.withColumn("movie_id", col("movie_id").cast("string"))
credits_cast = credits_cast.fillna({'credit_id': 0000000000})

# aynı değerleri içeren tekrarlanan satırları sil
un_credits_cast = credits_cast.dropDuplicates(['movie_id', 'title','cast_id', 'character', 'credit_id','id', 'name'])

cast_deltaPath = "s3a://tmdb-silver/credits_cast"
cast_delta = DeltaTable.forPath(spark, cast_deltaPath)

#minio upsert
cast_delta.alias("cast") \
  .merge(un_credits_cast.alias("cast_new"), "cast.movie_id = cast_new.movie_id AND cast.credit_id = cast_new.credit_id") \
  .whenMatchedUpdateAll() \
  .whenNotMatchedInsertAll() \
  .execute()
```

Öncesinde ilgili kütüphaneleri ekledim ve SparkSession oluşturdum. Örnek olarak cast tablosu için bronze bucketinde credits tablosunun dönüşümlerini yandaki kod parçacığında görüyoruz. Diğer 7 tablo için benzer işlemler yapıp silver layerina upsert ediyoruz.

- `json_cast` adında bir `ArrayType` içinde `StructType` tanımladım. Bu, JSON formatındaki verilerin nasıl yapılandırılacağını belirtir. JSON'daki her öğe, içinde belirtilen yapıdaki bir dizi olacaktır.
- `df_credits1` DataFrame'ine `withColumn` yöntemi kullanılarak yeni bir sütun eklenir. Bu yeni sütun, "cast" sütununu ayrıştırılmış JSON verileri içerecek şekilde tanımlanır.
- **explode_outer** fonksiyonu, iç içe geçmiş bir diziyi düzleştirir ve dışarıda kalan diğer sütunların verilerini korur. Bu, JSON içindeki yapısal bilgileri çıkarmak için kullanılır.
- Son olarak, istenen sütunlar seçilir ve **credits_cast** adında yeni bir DataFrame oluşturulur. Ayrıca, "movie_id" sütununun veri tipi "string" olarak değiştirilir.
- `fillna` ile `credit_id` kolonu `null` olan değerlere `000000` değeri atanır
- `DeltaTable.forPath()` yöntemi kullanılarak Delta Lake tablosunun yolunu belirleyerek ve **cast_delta** adında bir DeltaTable nesnesi oluşturarak başlar.
- Ardından, **merge** işlemi gerçekleştirilir. Bu, iki DataFrame'in birleştirilmesini ve belirli koşullara göre güncellenmesini sağlar. İki DataFrame'in birleştirilmesi sırasında, "movie_id" ve "credit_id" alanlarının eşleşmesi kontrol edilir.
- **alias** yöntemi, tablo isimlerini belirtmek için kullanılır. Bu durumda, mevcut Delta tablosu "cast" olarak ve yeni DataFrame "cast_new" olarak adlandırılır.
- `merge` işleminde, `whenMatchedUpdateAll()` metodu kullanılarak eşleşen kayıtlar güncellenir ve `whenNotMatchedInsertAll()` metodu kullanılarak eşleşmeyen kayıtlar eklenir.
- Son olarak, **execute()** yöntemi ile bu işlem gerçekleştirilir ve Delta Lake tablosu güncellenir.

Airflow Kullanarak İşlemleri Schedule Etmek

```
default_args = {
  'owner': 'airflow',
  'start_date': datetime(2024, 3, 22),
  'retries': 1,
  'retry_delay': timedelta(seconds=5)
}

with DAG('bronze_to_silver_dag', default_args=default_args, description='bronze to silver', schedule_interval='@daily',
catchup=False) as dag :
  datagen_credits_bronze = SSHOperator(task_id='bronze_credits', ssh_conn_id='spark_ssh_conn',conn_timeout = None,
  cmd_timeout = None,
  command =""" cd /dataops/data-generator && \
  source datagen/bin/activate && \
  python /dataops/data-generator/dataframe_to_s3.py \
  -buc tmdb-bronze \
  -k credit/credits_part \
  -aki dataops9 -sac ***** \
  -eu http://minio:9000 \
  -i /dataops/tmdb_5000_movies_and_credits/tmdb_5000_credits.csv \
  -ofp True -z 500 -b 0.1 -oh True""")

  datagen_movies_bronze ...

  movies_s3_silver_task = SSHOperator(task_id='silver_movies', ssh_conn_id='spark_ssh_conn', conn_timeout = None,
  cmd_timeout = None,
  command=""" cd /dataops/airflowenv && \
  source bin/activate && \
  python /dataops/tmdb_movies_Tables_to_s3.py""")

  credits_s3_silver_task...

  datagen_movies_bronze>>movies_s3_silver_task
  datagen_credits_bronze>>credits_s3_silver_task
```

- default_args tanımlayarak bu, DAG'in genel özelliklerini içerir. Kimin oluşturduğu (owner), ne zaman başlayacağı (start_date), yeniden deneme sayısı (retries) ve yeniden deneme gecikmesi (retry_delay) gibi bilgileri içerir.

- Oluşturduğum dag @daily çalışıyor ve catchup=false komutunu false seçereek belirlenen tarihten önceki tarihlerde geriye dönük işlemleri gerçekleştirmek istememiz.

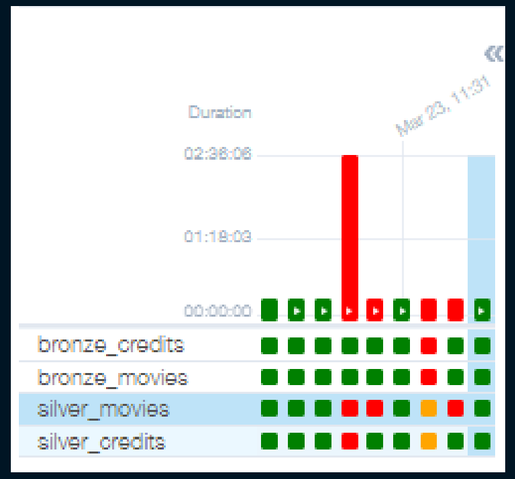
- SSHOperator kullanılarak dört adet task tanımladım: datagen_credits_bronze, datagen_movies_bronze ve movies_s3_silver_task, credits_s3_silver_task. datagen ile başlayan ve movies_s3_silver_task , credits_s3_silver_task taskleri benzer olduğu için sunum için birer örnek gösterdim.

- datagen_credits_bronze ve datagen_movies_bronze görevleri, belirtilen konumdaki verileri almak ve bunları S3'teki tmdb-bronze konumuna konuma yüklemek için SSH aracılığıyla belirli bir komut çalıştırmaktır.

- movies_s3_silver_task ve credits_s3_silver_task görevleri, tmdb-bronze bucketindaki verileri işleyip dönüştürmek ardından tmdb-silver bucketine belirlenen tabloya aktarmak için SSH aracılığıyla belirli bir komut çalıştırır. Bu komutlar, belirli bir dizine gidip bir sanal ortamı etkinleştirir ve ardından belirli bir Python betiğini çalıştırır.

- >> operatörü, görevler arasında bağımlılıkları belirtir. Yani, datagen_movies_bronze görevinin tamamlanmasını bekleyip ardından movies_s3_silver_task görevini çalıştırır ve aynı şekilde datagen_credits_bronze görevinin tamamlanmasını bekleyip ardından credits_s3_silver_task görevini çalıştırır.

Airflow



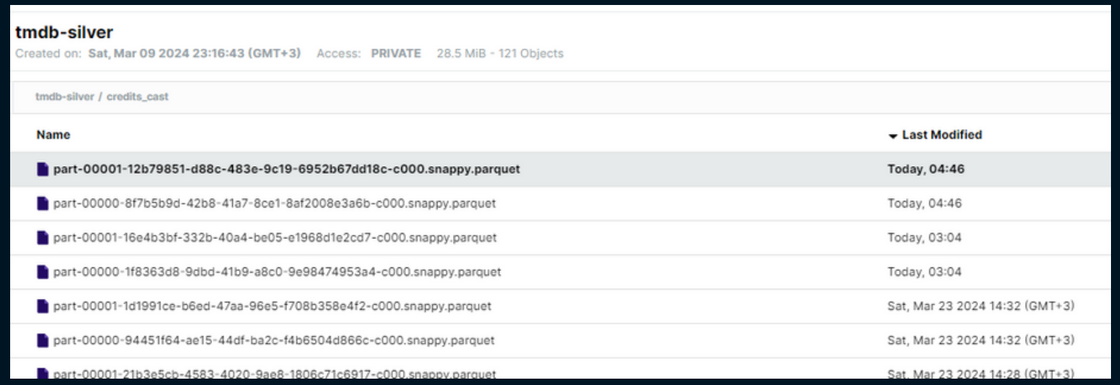
- tasklerin başarı durumu

```
movies_deltaPath = "s3a://tmdb-silver/movies"
movies = DeltaTable.forPath(spark, movies_deltaPath)
movies = movies.toDF()
movies.printSchema()
movies.count()
```

```
root
 |-- movie_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- budget: double (nullable = true)
 |-- homepage: string (nullable = true)
 |-- original_language: string (nullable = true)
 |-- original_title: string (nullable = true)
 |-- overview: string (nullable = true)
 |-- popularity: float (nullable = true)
 |-- release_date: date (nullable = true)
 |-- revenue: double (nullable = true)
 |-- runtime: integer (nullable = true)
 |-- status: string (nullable = true)
 |-- tagline: string (nullable = true)
 |-- vote_average: float (nullable = true)
 |-- vote_count: integer (nullable = true)
```

```
4803
```

- movies tablosu şemasıve satır sayısı



- tmdb-silver bucketina aktarılan veriler

- Airflowda dagimi oluşturdum ve başlattım. Tasklerim başarılı bir şekilde tamamlandı ve çıkan tabloların şemasını gösterdi.

- Projedeki amaçlaramızdan biri Bronze bucketinda verileri işleyip silver bucketina upsert yöntemiyle aktaracağımız için credits ve movies veri setlerinin satır sayısını düşürdüm ve doğru şekilde beklenen verilerin aktarıldığını gördüm.

- Dagimi son kez tüm veri setiyle tekrar trigger ettiğimde ham veri setimle ve tmdb-silver bucketindaki tablomun aynı sayıda satıra(4803) sahip olduğunu görerek işlemin başarılı olduğunu gördüm.

İşlenmiş Veriler Üzerinde Yapılan Örnek Sorgular

```
c_cast = c_cast.select("movie_id","character","name")
c_cast = c_cast.withColumnRenamed("movie_id", "cast_movie_id")
```

```
joined_df = movies.join(c_cast, movies.movie_id == c_cast.cast_movie_id)
```

```
joined_filt_tcruise = joined_df.filter(joined_df["name"] == "Tom Cruise")
```

```
high_filt_tcruise = joined_filt_tcruise \
    .orderBy(desc("revenue")) \
    .select("title", "revenue","character","name") \
    .limit(1)
high_filt_tcruise.toPandas()
```

	title	revenue	character	name
0	Mission: Impossible - Ghost Protocol	694713380.0	Ethan Hunt	Tom Cruise

En yüksek hasılata sahip tom cruise filmi hangisidir?

- Cast tablosundaki movie_id, character,name sürunları seçilir.
- cast ve movies tablosunu joinleme işlemi yaptım.
- joinlenmiş tabloda ismi Tom Cruise olan kişi filter kullanarak bulunur. Ardından revenue kolonuna göre desc ile sıralama yapılır, ilgili satırlar seçilir ve sonuç çıktısı alınır

En yüksek hasılata sahip Tom Cruise filmi Ethan Hunt karakterini canlandırdığı Mission Impossible: Ghost Protocol 'dur.

```
spoken_lang = spoken_lang.withColumnRenamed("movie_id", "spk_movie_id")
mv_spk_joined = movies.join(spoken_lang, movies.movie_id == spoken_lang.spk_movie_id)
```

```
movie_counts = mv_spk_joined.groupBy("movie_id", "title") \
    .count() \
    .orderBy(desc("count")) \
    .limit(1)
movie_counts.show()
```

```
[Stage 119:>                                                (0 + 2) / 2]
+--------+-----+-----+
|movie_id|title|count|
+--------+-----+-----+
|   14161| 2012|    9|
+--------+-----+-----+
```

Konuşulan farklı dil sayısı en çok hangi filmdedir

- spoken_lang tablosundaki movie_id kolon ismini spk_movie_id olarak değiştirdim
- spoken_lang ve movies tablosunu joinleme işlemi yaptım.
- joinlenmiş tabloda count ile sayma işlemi yaparak en fazla count değeri olan film bulunur

En fazla farklı dilin konuşulduğu film 9 farklı lisan ile 2012 filmidir

```
mov_id = movie_counts.select("movie_id").collect()[0]["movie_id"]

most_repeat2 = mv_spk_joined \
    .filter(col("movie_id") == mov_id) \
    .select("title","iso_639_1","name") \
# Sonucu gösterme
most_repeat2.toPandas()
```

	title	iso_639_1	name
0	2012	zh	普通话
1	2012	ru	Русский
2	2012	pt	Português
3	2012	la	Latin
4	2012	it	Italiano
5	2012	hi	हिन्दी
6	2012	fr	Français
7	2012	en	English
8	2012	bo	

Yukarıdaki belirlenen sorguya göre bu filmde konuşulan diller hangileridir

- movie_counts DataFrame'inden en fazla tekrar eden filmi (most_repeat) bulur. Bu, collect() fonksiyonuyla tüm verilerde en fazla sayıya sahip olan film idsini getirdim.
- mv_spk_joined DataFrame'inde, mov_id değerine sahip olanı filtredim.
- Bu filme ait konuşulan dillerin hepsini gösterdim

2012 filminde çince, ingilizce, fransızca gibi toplam 9 farklı dil konuşulmuştur.