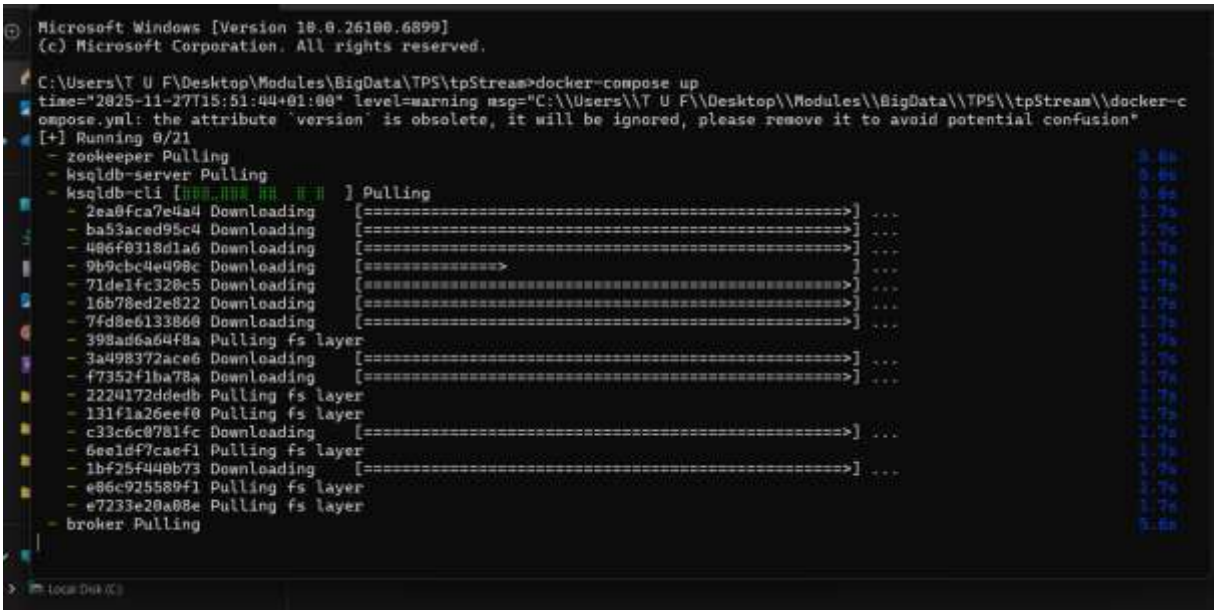
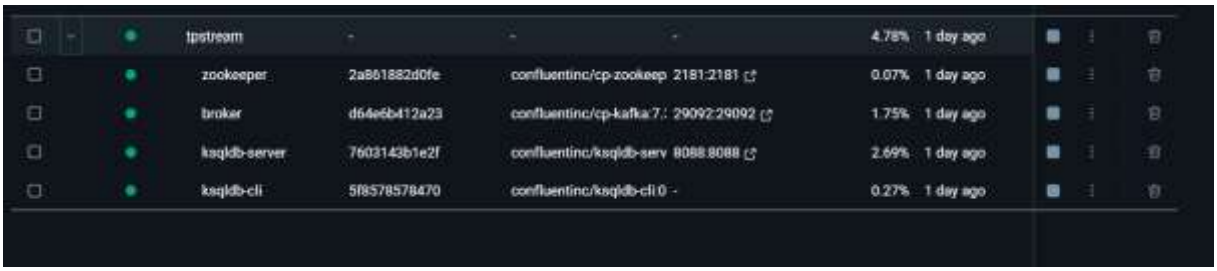


Ceci est la premiere etape pour lancer les conteneurs.

Apres lancement de la requete pour avoir access à sql et taper les requtes :
docker exec -it ksqldb-cli ksql http://ksqldb-server:8088

Dans DOCKER :



1) Créer en ksqldb le type personnalisé **season_length** qui représente le type des deux attributs **before** et **after**.

3

```
ksql> CREATE TYPE season_length AS STRUCT<
>    season_id INT,
>    episode_count INT
>>;
>

 Message
----------------------------------------------------------------------------------------------------------------------------
 Registered custom type with name 'SEASON_LENGTH' and SQL type STRUCT<`SEASON_ID` INTEGER, `EPISODE_COUNT` INTEGER>
----------------------------------------------------------------------------------------------------------------------------
ksql>
```

Pour verifier le type :

```
---------------
ksql> SHOW TYPES;

 Type Name     | Schema
-----------------------------------------------------------------
 SEASON_LENGTH | STRUCT<SEASON_ID INTEGER, EPISODE_COUNT INTEGER>
-----------------------------------------------------------------
ksql>
```

2) Choisir le type de collection le plus approprié pour **titles** et pour **production_changes** en justifiant votre choix.

Pour titles => tableau parceque on a besoin de la derniere valeur mis à jours
Pour production_changes = > Stream parceque on garde tous l'historique des evenements

3) Créer la collection **titles** et la collection **production_changes,** sachant que :
   - **id** est l'identifiant de la collection **titles.**
   - **rowkey** est l'identifiant de la collection **production_changes**.
   - La colonne **created_at** de **production_changes** contient le timestamp que ksqlDB doit utiliser pour les opérations temporelles (e.g., windowed aggregations and joins).
   - on suppose que le nombre de partition = 4 dans les deux collections.

Creation de Struct(record) : type personalisé:

Creation de stream production_change par ce que on a beosin de tout l'hstorique :

```
ksql> CREATE STREAM production_changes (
>  rowkey STRING,
>  title_id INT,
>  change_type STRING,
>  before season_length,
>  after season_length,
>  created_at STRING
>) WITH (
>  KAFKA_TOPIC='production_changes',
>  VALUE_FORMAT='JSON'
>);
>

 Message
----------------
 Stream created
----------------
ksql>
```

Et pour titles on crée un tabelau par ce que on a besoin de la derniere information mis à jours ,c haque id doit avoir un seul titre:

```
ksql> CREATE TABLE titles (
>  id INT PRIMARY KEY,
>  title STRING
>) WITH (
>  KAFKA_TOPIC='titles',
>  VALUE_FORMAT='JSON',
>  PARTITIONS=4
>);
>

 Message
--------------
 Table created
--------------
ksql> show tables;

 Table Name | Kafka Topic | Key Format | Value Format | Windowed
-----------------------------------------------------------------
 TITLES     | titles      | KAFKA      | JSON         | false
-----------------------------------------------------------------
ksql>
```

4) Insérer des données dans les deux collections **titles** et **production_changes**.

es requêtes

```
ksql> INSERT INTO titles (id, title) VALUES (1, 'LOVE TRAP');
>INSERT INTO titles (id, title) VALUES (2, 'QUALMS');
>INSERT INTO titles (id, title) VALUES (3, 'THORNY HEART');
>
ksql> INSERT INTO production_changes (
>  rowkey, title_id, change_type, before, after, created_at
>) VALUES (
>  'change_1',
>  1,
>  'SEASON_CREATED',
>  NULL,
>  STRUCT(season_id := 1, episode_count := 12),
>  '2023-11-08T08:15:30.929Z'
>);
>
>INSERT INTO production_changes (
>  rowkey, title_id, change_type, before, after, created_at
>) VALUES (
>  'change_2',
>  1,
>  'season_length',
>  STRUCT(season_id := 1, episode_count := 12),
>  STRUCT(season_id := 1, episode_count := 10),
>  '2024-01-01T10:00:00.000Z'
>);
>
```

On met : SET 'auto.offset.reset' = 'earliest'; pour qu'on nous puissons lire des le début des collections :

```
ksql> SET 'auto.offset.reset' = 'earliest';
Successfully changed local property 'auto.offset.reset' to 'earliest'. Use the UNSET command to revert your change.
ksql>
```

Les requetes :

5) Ecrire une requête Push qui permet de retourner tous les changements de production créés avant le 2023-04-14 à 12:00:00.

```
ksql> SELECT *
>FROM production_changes
>WHERE created_at < '2023-04-14T12:00:00'
>EMIT CHANGES;
>
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
|ROWKEY           |TITLE_ID         |CHANGE_TYPE      |BEFORE           |AFTER            |CREATED_AT       |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+

Press CTRL-C to interrupt
```

6) Ecrire une requête Push qui permet de retourner tous les enregistrements de **production_changes** où la colonne "change_type" commence par le mot "season".

```
ksql> SELECT *
>FROM production_changes
>WHERE change_type LIKE 'season%'
>EMIT CHANGES;
>
+------------------+------------------+------------------+------------------+------------------+------------------+
|ROWKEY            |TITLE_ID          |CHANGE_TYPE       |BEFORE            |AFTER             |CREATED_AT        |
+------------------+------------------+------------------+------------------+------------------+------------------+
|change_2          |1                 |season_length     |{SEASON_ID=1, EPIS|{SEASON_ID=1, EPIS|2024-01-01T10:00:0|
|                  |                  |                  |ODE_COUNT=12}     |ODE_COUNT=10}     |0.000Z            |
Press CTRL-C to interrupt
```

7) Créer à partir de **production_changes** un stream dérivé nommé **season_length_changes**, qui ne contient que les changements de production de type **season_length**.

- On suppose que **season_length_changes** écrit dans un topic qui porte le même nom, le nombre de partition = 4, le nombre de réplique = 1 et les messages stockés dans le topic sont encodés en JSON.
- season_length_changes contient les colonnes :
  - ROWKEY, title_id, created_at
  - season_id : cette colonne reçoit la valeur de after->season_id,

    Si after->season_id est null, season_id reçoit la valeur de before->season_id.

  - old_episode_count : cette colonne reçoit la valeur de before->episode_count

  - new_episode_count : cette colonne reçoit la valeur de after->episode_count

```
ksql> CREATE STREAM season_length_changes
>WITH (KAFKA_TOPIC='season_length_changes', VALUE_FORMAT='JSON', PARTITIONS=4, REPLICAS=1) AS
>SELECT
>  ROWKEY,
>  title_id,
>  created_at,
>  COALESCE(after->season_id, before->season_id) AS season_id,
>  before->episode_count AS old_episode_count,
>  after->episode_count AS new_episode_count
>FROM production_changes
>WHERE change_type = 'season_length'
>EMIT CHANGES;
>

 Message
----------------------------------------------------
 Created query with ID CSAS_SEASON_LENGTH_CHANGES_15
----------------------------------------------------
ksql>
```

8) Ecrire une requête Push qui permet de retourner les titres de toutes les vidéos contenues dans **season_length_changes**.

```
ksql> SELECT t.title
>FROM season_length_changes s
>LEFT JOIN titles t
>  ON s.title_id = t.id
>EMIT CHANGES;
>
+---------------------------------------+
|TITLE                                  |
+---------------------------------------+
|LOVE TRAP                              |
```

9) Créer le stream **season_length_changes_enriched** qui contient tous les attributs de **season_length_changes** plus l'attribut **title** de la collection **titles**.
   - La colonne **created_at** de **season_length_changes_enriched** contient le timestamp que ksqlDB doit utiliser pour les opérations temporelles.

```
ksql> CREATE STREAM season_length_changes_enriched AS
>SELECT
>  s.ROWKEY,
>  s.title_id,
>  t.title,
>  s.created_at,
>  s.season_id,
>  s.old_episode_count,
>  s.new_episode_count
>FROM season_length_changes s
>LEFT JOIN titles t
>  ON s.title_id = t.id
>EMIT CHANGES;
>

 Message
-------------------------------------------------------------------
 Created query with ID CSAS_SEASON_LENGTH_CHANGES_ENRICHED_17
-------------------------------------------------------------------
ksql>
```

10) Créer la table dérivée **season_length_change_counts** à partir de **season_length_changes_enriched.**
   - La table **season_length_change_counts** représente le nombre de changement effectués pour chaque **title**.
   - De plus, elle contient l'attribut **episode_count** qui représente la dernière valeur de **new_episode_count**.
   - La table **season_length_change_counts** est créé à partir d'un WINDOW TUMBLING d'une heure.

```
ksql> CREATE TABLE season_length_change_counts AS
>SELECT
>  title,
>  COUNT(*) AS change_count,
>  LATEST_BY_OFFSET(new_episode_count) AS episode_count
>FROM season_length_changes_enriched
>WINDOW TUMBLING (SIZE 1 HOUR)
>GROUP BY title
>EMIT CHANGES;
>

 Message
---------------------------------------------------------------
 Created query with ID CTAS_SEASON_LENGTH_CHANGE_COUNTS_19
---------------------------------------------------------------
ksql>
```