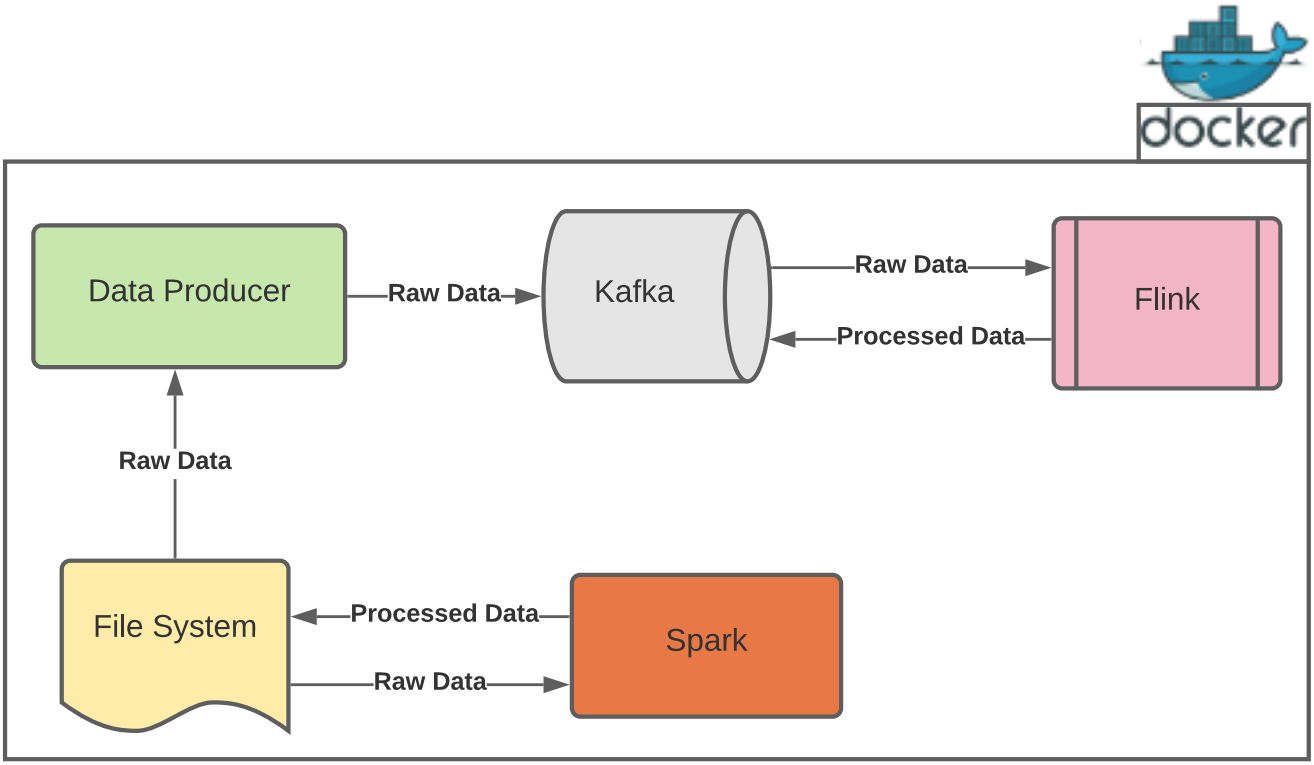
Data Engineering Bitirme Projesi

Data Engineering'e dair öğrendiklerinizi sınayabileceğiniz bu projede aşağıdaki işleri içeren problemlerle uğraşacaksınız:

- Batch Processing
- Stream Processing
- DevOps



Yukarıdaki diyagramda uçtan uca bir Data Pipeline görülüyor. Burada Data Producer, Kafka, Flink ve Spark uygulamaları var. Sizlere

- Data Producer
- Kafka

uygulamalarını bir `docker-compose.yml` dosyasında sunuyoruz. Sizden beklentimiz:

- Dosya sistemindeki verileri kullanarak batch Spark uygulama(lar) yazmanız.
- Kafka'daki kaynağı kullanarak bir Flink uygulaması yazmanız.
- Tüm uygulamaları Dockerize etmeniz ve sistemi ayağa kaldıran tek bir komut (script ya da docker-compose up) sunmanız.

Problem Detayları

Veri

`orders` ve `products` verileri üzerinde çalışacaksınız. Bunların şemaları şu şekilde:

```
orders
 |-- customer_id: string
 |-- location:    string
 |-- seller_id:   string
 |-- order_date:  string
 |-- order_id:    string
 |-- price:       double
 |-- product_id:  string
 |-- status:      string

products
 |-- brandname:    string
 |-- categoryname: string
 |-- productid:    string
 |-- productname:  string
```

Bu iki veriyi şu şekilde `join`'leyebilirsiniz:

```
orders.product_id = products.productid
```

Batch Processing

Burada Spark ile 3 farklı problem üzerinde uğraşacaksınız. Bunları tek bir job ile çözebilirsiniz, her biri için farklı job'lar da yazabilirsiniz, tasarım kararı size ait.

Buradaki problemlerde temelde yapılacak işlem dosya sisteminden ham verileri okumak, işlemek ve dosya sistemine geri yazmak, istediğiniz formatta (CSV, JSON, vs.) yazabilirsiniz.

Problem 1

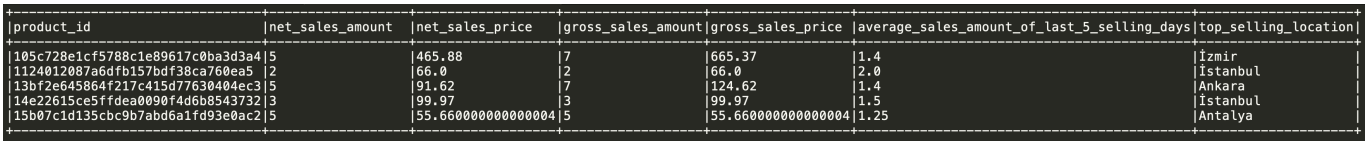
Her ürün için aşağıdakileri hesaplayın ve dosya sistemine sonuçları yazın:

- net satış adedi ve tutarı (toplam satış adetleri - iptal / iade adetleri)
- brüt satış adedi ve tutarı (toplam satış adetleri)
- **satıldığı** son 5 gündeki satış sayısı ortalamaları
- en çok sattığı yer

Aşağıda örnek bir ürün için son satılan 5 günün nasıl olabileceğine bakabilirsiniz, dikkat ederseniz satıldığı son 5 gün herhangi bir zaman olabilir, ardışık günler olmak zorunda değil.

```
+------------+-------------------+
|product_id  |order_date         |
+------------+-------------------+
|<some-id-1> |2021-01-22 22:20:32|
|<some-id-1> |2020-12-23 13:50:20|
|<some-id-1> |2020-12-23 09:45:10|
|<some-id-1> |2021-01-14 07:57:24|
|<some-id-1> |2021-01-03 11:44:47|
|<some-id-1> |2021-01-03 14:34:43|
|<some-id-1> |2021-01-02 12:05:29|
+------------+-------------------+
```

İşlem sonucunda elde edeceğiniz kolonlar ve örnek bir çıktı şu şekilde olabilir:



```
+--------------------------+---------------+-----------------+-----------------+-----------------+-----------------------------------------+-------------------+
|product_id                |net_sales_amount|net_sales_price  |gross_sales_amount|gross_sales_price|average_sales_amount_of_last_5_selling_days|top_selling_location|
+--------------------------+---------------+-----------------+-----------------+-----------------+-----------------------------------------+-------------------+
|105c728e1cf5788c1e89617c0ba3d3a4|5              |465.88           |7                |665.37           |1.4                                      |İzmir              |
|1124012087a6dfb157bdf38ca760ea5 |2              |66.0             |2                |66.0             |2.0                                      |İstanbul           |
|13bf2e645864f217c415d77630404ec3|5              |91.62            |7                |124.62           |1.4                                      |Ankara             |
|14e22615ce5ffdea0090f4d6b8543732|3              |99.97            |3                |99.97            |1.5                                      |İstanbul           |
|15b07c1d135cbc9b7abd6a1fd93e0ac2|5              |55.660000000000004|5                |55.660000000000004|1.25                                     |Antalya            |
+--------------------------+---------------+-----------------+-----------------+-----------------+-----------------------------------------+-------------------+
```

Problem 2

Günlük kırılımda net satışlar üzerinden en çok gelir elde eden 10 satıcı için aşağıdakileri hesaplayın ve dosya sistemine yazın.

- satış adedi
- en çok gelir elde ettiği kategori

İşlem sonucunda elde edeceğiniz kolonlar şu şekilde olabilir:

```
top_selling_sellers
 |-- seller_id:            string
 |-- sales_amount:         integer
 |-- top_selling_category: string
 |-- date:                 timestamp
```

Buradaki `date` kolonu günlük precision'da olmalı.

Problem 3

Ürün bazlı fiyat değişimlerini tarihleriyle birlikte çıkarıp dosya sistemine yazın.

Burada `orders` verisi üzerinde aynı `product_id`'nin sonraki order'da fiyatı arttıysa `rise`, azaldıysa `fall` gibi bir çıktı üretebilirsiniz.

Örnek girdi:

```
+-------------+------+-------------------+
|product_id   |price |order_date         |
+-------------+------+-------------------+
|<some-id-1>  |3.00  |2021-01-22 01:20:32|
|<some-id-1>  |3.00  |2021-01-22 02:50:20|
|<some-id-1>  |3.25  |2021-01-22 03:45:10|
|<some-id-2>  |3.25  |2021-01-22 13:45:10|
|<some-id-2>  |3.25  |2021-01-22 14:45:10|
|<some-id-2>  |3.45  |2021-01-22 15:45:10|
|<some-id-1>  |3.25  |2021-01-22 04:57:24|
|<some-id-1>  |2.99  |2021-01-22 05:44:47|
|<some-id-1>  |2.99  |2021-01-22 06:34:43|
|<some-id-1>  |3.50  |2021-01-22 07:05:29|
+-------------+------+-------------------+
```

Örnek çıktı:

```
+-------------+------+-------------------+--------+
|product_id   |price |order_date         |change  |
+-------------+------+-------------------+--------+
|<some-id-1>  |3.25  |2021-01-22 03:45:10|rise    |
|<some-id-1>  |2.99  |2021-01-22 05:44:47|fall    |
|<some-id-1>  |3.50  |2021-01-22 07:05:29|rise    |
|<some-id-2>  |3.45  |2021-01-22 15:45:10|rise    |
+-------------+------+-------------------+--------+
```

Stream Processing

Flink ile 5 dakikalık zaman aralıklarında konum kırılımında aşağıdakileri hesaplayın ve Kafka'da bir topic'e yazın:

- satış yapılan farklı satıcı sayısı
- toplam satılan ürün sayısı

İşlem sonucunda elde edeceğiniz alanlar şu şekilde olabilir, veriyi Kafka'ya JSON formatında yazabilirsiniz:

```
seller_and_sold_product_counter
 |-- seller_count:  long
 |-- product_count: long
 |-- location:      string
 |-- date:          timestamp
```

Buradaki `date` kolonu 5 dakikalık precision'da olmalı.

DevOps

Projedeki tüm uygulamaları dockerize etmeli ve bunları da ayağa kaldırmak için 2 farklı `docker-compose.yml` dosyası oluşturmalısınız.

İlk compose dosyası aşağıdaki servisleri ayağa kaldırmalı:

- Spark Master
- Spark Worker

Diğer dosya ise şu servisleri ayağa kaldırmalı:

- Data Producer
- Kafka
- Zookeeper
- Flink Job Manager
- Flink Task Manager

Not: 2 farklı compose yaml'ı tüm servisleri aynı anda ayağa kaldırırken sistem kaynaklarının yetmeme olasılığını önlemek için istiyoruz. Compose yaml'ları tek tek çalıştırıp durdurabilirsiniz ya da isterseniz tüm servisleri tek bir compose dosyasında toplayabilirsiniz.

Önerilerimiz

- Tüm projeyi yetiştirmekte zorlanabilirsiniz, yapabildiğiniz kadarını yapmanızı istiyoruz.
- DevOps kısmını en sona bırakın. Burada çok uğraştırıcı durumlar var ve biz Spark ve Flink'te yapacağınız analizleri daha çok önemsiyoruz. DevOps'u sona bıraktığınız durumda vereceğemiz JSON dosyalarını doğrudan Spark ile okuyup işlemeye başlayabilirsiniz. Flink için de Kafka'yı local'de ayağa kaldırıp problemin üstünde çalışmaya başlayabilirsiniz.
- Spark problemleri için bu sayfadaki `Window functions` kısmını detaylı incelemenizi öneriyoruz.

DevOps Özelinde Öneriler

- DevOps eğitiminde paylaşılan kaynaktaki komutlardan ve Flink için hazırlanan compose dosyasından faydalanabilirsiniz.
- Spark için burada bulunan compose yaml'daki servislerden faydalanabilirsiniz. (Bu yaml'ın kullandığı `Dockerfile`'ı düzenleyerek istediğiniz Spark sürümü için çalıştırabilirsiniz.)
- Beklentimiz `docker-compose up` yaptığımızda veya hazırlayacağınız bir initializer script'ini çalıştırdığımızda tek komutta her şeyin ayağa kalkması, fakat çözemezseniz eğer Spark ve Flink container'ları ayağa kalktıktan sonra job'ları elle submit edebilirsiniz.
- Spark ve Flink container'ları kalktığında job'ları otomatik submit etmek için bu container'ların ayağa kalktığından emin olan ve job'ları submit eden shell script'ler yazabilirsiniz, bunun için until döngüsünü kullanabilirsiniz. Bu döngü Spark UI veya Flink UI'a `curl` ile 5 saniyede 1 health-check isteği atıp cevap geldiğinde `spark-submit` veya `flink run` komutlarını çalıştırabilir.

Yukarıdaki yazdıklarımız sadece öneri, çok farklı çözümler bulunabilir, istediğiniz şekilde çözümü yapabilirsiniz.

Değerlendirme Kriterlerimiz

- Kod Kalitesi
- Testler
- Spark
- Flink
- DevOps
- Dokümantasyon

Proje Teslimi

Projeyi yaptıktan sonra **private** bir GitHub repo'suna yükleyip bizimle paylaşmanızı bekliyoruz.