RENO (WK3 ↔ WK4 synchronizer)

A quick introduction

Reno

- Reno is a synchronization service, it keeps two system’s data sources in-sync (WK3 ↔ WK4)
- It is written in pull approach, meaning that it pulls for changes and then based on the outcome, it performs the necessary actions in order to synchronize the other end

Reno

- Flow
  - When a data change occurs in WK3, it produces a notification (with the help of entity lifecycle hooks) with the entity name and the primary key(s) of this affected entity to a specified kafka topic, called **reno-notifications**
  - From there, reno consumes the data in batches (by polling at fixed time interval), aggregating eventual duplicates, saves these notifications to persistence layer (which in our case is MongoDB) and then commits the offsets (processing first and then commit offsets, with having offsets managed by Kafka, we have at least once semantic delivery)
    - Note: Before refactoring, it was consuming the data, apply on these transformations and then commit the offsets, transformations took a lot of time, so when commit the offsets you had CommitFailedException due to partition rebalances or processing timeouts (changed leader of that topic partition, etc.)

      When a CommitFailedException occurred, reno it was written in a way to terminate the jvm process (System.exit(..)) and because we have 2 replicas defined in Kubernetes, a new jvm process would be started from Kubernetes. So when you experienced a lot of CommitFailedException errors, due to slow transformations or infra problems you were in a loop of:
      {START → CommitFailedException→ Exit Jvm → Kubernetes start new Jvm → goto START}
      so two systems where out of sync for a long period of time.

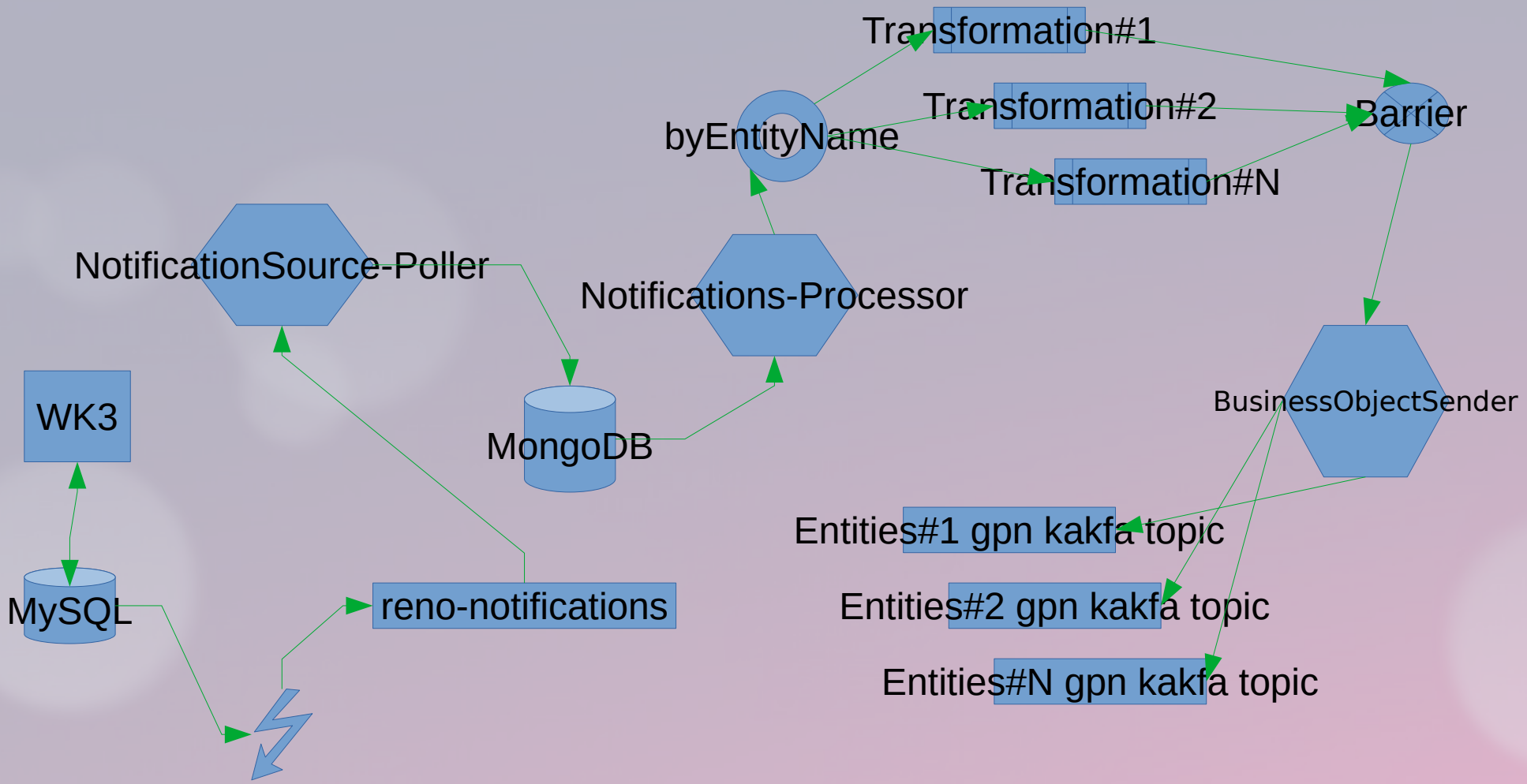
Reno

- Flow
  - These stored notifications in MongoDB are picked up from an app thread, called `Notifications-Processor`, which based on the entity name of these notifications get forwarded to the correct business transformer (AdvertiserTransformation, CampaignTransformation, etc), in order to calculate the new entity state (represented in JSON) via hitting WK3 DB…
  - ...and then we are sending (via BusinessObjectSender.java/kafka producer) the just new calculated entity state to the correct GPN kafka topic, based on entity name, eg: if we have processed campaign entities, we are producing messages to `campaigns` kafka topic.
  - We have one worker-thread per business transformation

Reno

- Flow

```
private final Map<String /*entity name*/, TransformationEntry> transformations;


....

Map<String, List<Notification>> notificationGroupByEntity = notifications
      .stream()
      .collect(Collectors.groupingBy(Notification::getEntity));

return notificationGroupByEntity
      .entrySet()
      .stream()
      .map(entry ->
            CompletableFuture.supplyAsync(
                  () -> entry
                        .getValue()
                        .stream()
                        .map(notification -> transform(notification, batchId))
                        .collect(Collectors.toList()),
                  transformationWorkers)
      ) ...

....

public Pair<Notification, Optional<TransformationResult>> transform(Notification notification, String batchId) {

   ....

   TransformationEntry entry = transformations.get(notification.getEntity());

   ....

   Optional<TransformationResult> transformationResult = entry.getTransformation().apply(currentConnection, notification);

   ....

}
```


Transformation#1
Transformation#2
Transformation#N
byEntityName
Barrier
NotificationSource-Poller
Notifications-Processor
WK3
MongoDB
BusinessObjectSender
Entities#1 gpn kakfa topic
Entities#2 gpn kakfa topic
Entities#N gpn kakfa topic
MySQL
reno-notifications

Reno

- Supported Transformations
  - Advertiser
  - Campaign
  - Publisher
  - Theyget
  - User
  - Weget

Thank you!