Mestrado em Engenharia Informática

Universidade de Aveiro

Practical Assignment II – Kafka

Centralized Control Platform to Supervise the Status of Cars

Arquitetura de Software

Docente: Doutor Óscar Mortágua Pereira

Grupo:

Dante Marinho (83672)
Tiago Rodrigues (84040)

2º Semestre

2019/2020

Índice

Introdução

O presente documento descreve o 2º trabalho prático da disciplina de Arquitetura de Software do Mestrado em Engenharia Informática da Universidade de Aveiro.

O problema baseia-se em supervisionar o status de carros através de uma plataforma centralizada de controlo, onde cada carro envia informações que serão armazenadas em um ficheiro remoto para ser processado pela plataforma centralizada.

Esta plataforma foi desenhada e implementada tendo como base o Apache Kafka, uma plataforma de processamento de streams de alta capacidade e baixa latência direcionada a tratamento de dados em tempo real.

Os dados originados pelos carros são armazenados num ficheiro CAR.TXT e dividem-se em três tipos de mensagens: “heartbeat”, “car speed” e “car status”.

Essas mensagens possuem os seguintes formatos:

Heartbeat:

| car_registration | timestamp | 00 |

Car speed:

|car_registration | timestamp | 01 |car_speed | - Onde car_speed compreende um inteiro >= 0

Car status:

|car_reg | timestamp | 02 | car_status | - Onde car_status pode ser Ok ou KO

A plataforma utiliza Topics do Kafka para transmitir informação dentro da aplicação, através de entidades criadas com a finalidade de produzir e consumir informação. Há uma entidade encarregada de coletar toda a informação contida no ficheiro CAR.TXT e há outras entidades consumidoras destas informações, onde estas processam dos dados consumidos e registam informação em outros ficheiros de texto.

Ferramentas de Trabalho

O grupo elegeu a IDE Eclipse para o desenvolvimento do trabalho devido a melhor adaptação ao contexto encontrado face às dificuldades encontradas para poder executar em outras IDEs.

Desenvolvimento

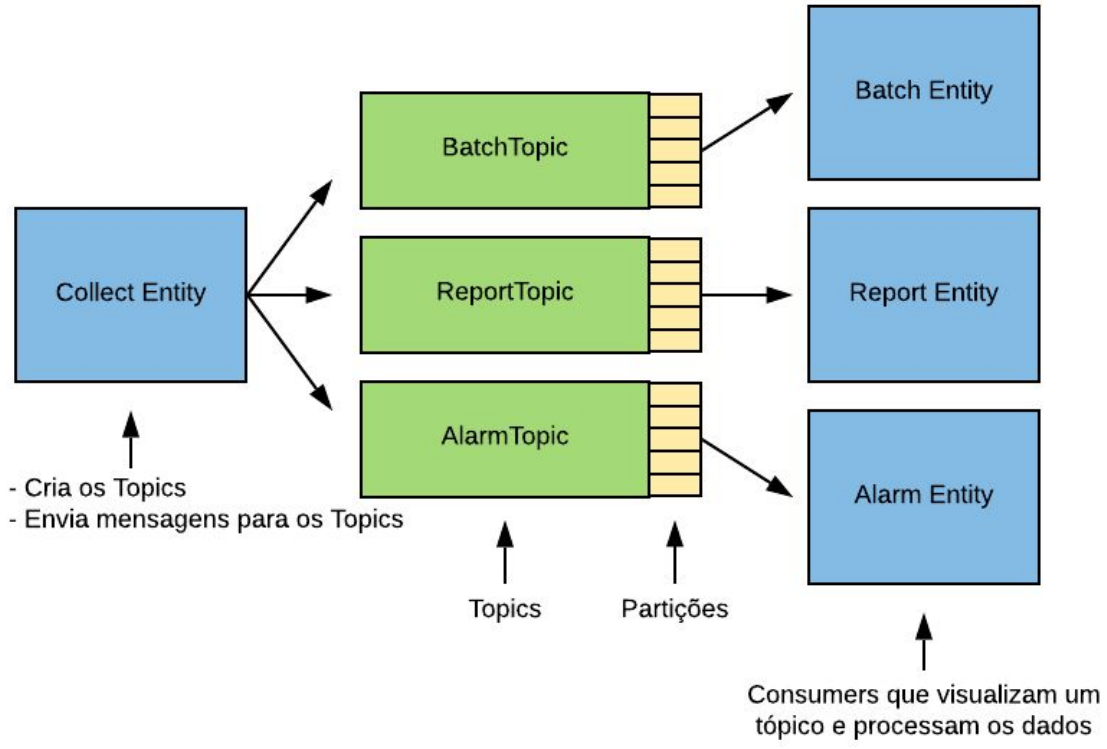
O desenvolvimento da plataforma vai basear-se na criação de quatro programas para poderem gerir três Topics do Kafka. Os Topics são:

- BatchTopic
- ReportTopic
- AlarmTopic

O programa CollectEntity, recolhe os dados que estão no ficheiro CAR.TXT, este utiliza o método readFile que retorna os dados do ficheiro txt numa arrayList que é fornecida ao método splitInfo() que vai separar a informação em 3 array lists que irão ser publicadas via mensagem nos seus respetivos tópicos pelo método CollectEntity. Os restantes programas são:

- BatchEntity:
  Consome as mensagens do tópico BatchTopic e armazena a informação no ficheiro BATCH.TXT.
- ReportEntity:
  Consome as mensagens do tópico ReportTopic e armazena a informação no ficheiro BATCH.TXT.
- AlarmEntity:
  Consome as mensagens do tópico BatchTopic, processa a informação e ativa ou desativa alarmes de acordo com a velocidade em que o carro se encontra. Este é ativado quando o carro excede a velocidade máxima permitida de 120 e é desativado quando volta a ficar abaixo desta velocidade. Sempre que o status do alarme muda é armazenada num ficheiro ALARM.TXT com o seguinte formato:
  | car_reg | 01 | speed | status |, onde status pode ser ON ou OFF

O seguinte diagrama ilustra, de maneira geral, a relação entre os componentes da plataforma.



Limitações

Foram implementadas 3 limitações, 'pode ser perdido', 'pode ser reordenado' e 'pode ser reprocessado'.

Pode ser perdido

Alterando a configuração Acks do produtor para 0, que significa que o cliente não tem que retornar que a mensagem foi recebida com sucesso. Isto permite que no caso de uma falha existam mensagens perdidas.

```
// ACKS_CONFIG  0 means that the client does not require any acknowledgement back, if one consumer fails and there might be loss messages
// when the messages are being splited to adjacent consumers
props.put(ProducerConfig.ACKS_CONFIG, "0");
```

Pode ser reordenado

Alterando a configuração max_in_flight_requests_per_connection para um valor superior a 1 causa uma situação onde no caso de ocorrer o erro no envio de uma mensagem a próxima mensagem pode ocupar o lugar da mensagem que falhou antes de ela voltar a ser enviada.

```
// this may cause the order of the messages to be changed since one message may fail and the next one takes that message offset
props.put(ProducerConfig.MAX_IN_FLIGHT_REQUESTS_PER_CONNECTION, "50");
```

Pode ser reprocessado

Alterando a configuração enable.auto.commit para verdade o consumidor vai enviar os offsets para o kafka periodicamente, a frequência desses envios é determinado pelo parámetro auto.commit.interval.ms caso a aplicação crashe antes do progresso ser guardado a informação vai ser enviada novamente permitindo a ocorrência de reprocessamento.

```
//Constrain, reprocessing is not guaranteed
props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "5000");
```

Orientações para Correr o Projeto

Para correr o projeto devem ser utilizados os scripts fornecidos que possuem um README para a sua utilização,depois disso devem ser abertas as interfaces dos consumidores desejados, AlertEntity, ReportEntity, BatchEntity e depois disso pode ser executado o CollectEntity que vai publicar a informação para os tópicos. Todos os ficheiros criados são guardados na package “data”.

Versão do Java

O programa foi desenvolvido com o NetBeans IDE v11.1 e está a executar com o JDK 13 do Java.

Divisão de Tarefas

O aluno Tiago Rodrigues contribuiu de forma total no projeto participando da criação e desenvolvimento de todas as packages e classes do projeto, exceto das classes para a criação do GUI (BatchEntityFrame, AlarmEntityFrame e ReportEntityFrame). Contribuiu também em fixes do projeto.

O aluno Dante Marinho contribuiu nas classes BatchEntity, ReportEntity, AlamEntity, classes para a criação do GUI (BatchEntityFrame, AlarmEntityFrame e ReportEntityFrame). Contribuiu também em fixes do projeto.

Percentagens de contribuição de cada aluno:

- 80% Tiago Rodrigues
- 20% Dante Marinho

Referências

[1] V. Jack, “How to Lose Messages on a Kafka Cluster - Part 1” [Online] Disponível em: https://jack-vanlightly.com/blog/2018/9/14/how-to-lose-messages-on-a-kafka-cluster-part1

[2] C. Kamil, “Does Kafka really guarantee the order of messages?” [Online] Disponível em: https://blog.softwaremill.com/does-kafka-really-guarantee-the-order-of-messages-3ca849fd19d2

[3] B. Andy, “Processing guarantees in Kafka” [Online] Disponível em: https://medium.com/@andy.bryant/processing-guarantees-in-kafka-12dd2e30be0e

[4] G. Gaurav, “Kafka Producer and Consumer Examples Using Java” [Online] Disponível em: https://dzone.com/articles/kafka-producer-and-consumer-example