universidade de aveiro
theoria poiesis praxis

Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Arquiteturas de Software (2021/22)

Trabalho Prático II

Grupo n.º 22

João Carvalho, Nmec: 89059 - 50% de participação
João Pedro Pereira, Nmec: 106346 - 50% de participação

Índice

Introdução

Este trabalho prático consiste, essencialmente, na implementação de uma plataforma, cujo intuito é representar uma simulação de processamento de dados de sensores num *Kafka Cluster*. O ficheiro que contém os sensores apresenta o seguinte formato:

XXXXX -> *string* que contém o *ID* do sensor;
ZZZ.ZZ -> número real que contém a temperatura in ºC;
YYYYYY -> *timestamp*.

Para o desenvolvimento da aplicação foi usada a linguagem *Java*, combinada com o *Apache Kafka.*

O problema apresenta seis casos de uso, para cada um deles foi implementada uma solução independente.

Para além do *Kafka Cluster* existem três processos adicionais:

- *PSource* - responsável por ler os dados dos sensores e enviá-los aos *producers* via *Java Sockets*;
- *PProducer* - responsável por receber os dados do *PSource* e enviá-los para o *Kafka Cluster*;
- *PConsumer* - responsável por consumir os dados do *Kafka Cluster* e processá-los.

Ao longo deste relatório serão explanados com mais detalhe os diferentes casos de uso e o processo do seu desenvolvimento.

Propriedades do Kafka

Cada caso de uso apresenta restrições, principalmente, a níveis de performance, ordem dos dados e perda de dados. Para satisfazer estas condições foram usadas propriedades do *kafka producer* e do *kafka consumer* na implementação.

Algumas propriedades usadas são comuns aos dois tipos, são elas:

1. **"BOOTSTRAP.SERVERS":** lista de pares de *host/ports* a serem usados para estabelecer a conexão inicial com o *cluster Kafka* [1].

2. **"KEY.DESERIALIZER":** classe de *deserializer* para a *key* que implementa a interface "*org.apache.kafka.common.serialization.Deserializer"* [1].

3. **"KEY.DESERIALIZER":** classe de *deserializer* para o *value* que implementa a interface "*org.apache.kafka.common.serialization.Deserializer"* [1].

Propriedades do Kafka Producer

No âmbito do *kafka producer* foram usadas sete propriedades que serão explanadas seguidamente.

1. **"ACKS"** : consiste no número de *acknowledgments* que o *producer* exige que o líder tenha recebido antes de considerar uma *request* concluída[1]. Pode tomar três valores:
    a. "0": o producer não esperará por nenhum reconhecimento do servidor, logo, nenhuma garantia pode ser feita de que o servidor tenha recebido o registo neste caso[1].
    b. "1": o líder gravará o registo no log local, mas responderá sem aguardar o reconhecimento total de todos os seguidores[1]. Ainda poderá haver perda de registos se o líder falhar logo após reconhecer o registo, mas antes que os seguidores o tenham replicado.
    c. "all": o líder aguardará que o conjunto completo de réplicas sincronizadas reconheça o registo. Isto garante que o registo não será perdido enquanto pelo menos uma réplica permanecer ativa[1].

2. **"BUFFER.MEMORY":** consiste no número total de bytes de memória que o *producer* pode usar para armazenar em buffer os registos que aguardam serem enviados ao servidor. O valor *default* é 33554432.

3. **"COMPRESSION.TYPE":** consiste em definir o tipo de compactação para todos os dados gerados pelo *producer*. Muito útil para melhorar o *throughput*. Valores válidos podem ser: *none*, *gzip*, *snappy*, *lz4*, ou *zstd* [1]. O valor *default* é *none*.

4. **"RETRIES":** consiste em definir quantas vezes o *producer* tentará enviar uma mensagem antes de marcá-la como falha[1]. O valor *default* é 2147483647.

5. **"BATCH.SIZE":** basicamente controla quantos bytes de dados devem ser acumulados antes de enviar mensagens para o *Kafka broker*. O *producer* tentará agrupar registos em menos *requests* sempre que vários registos estiverem a ser enviados para a mesma partição[1]. O valor *default* é 16384.

6. **"LINGER.MS":** é o número de milissegundos que um *producer* está disposto a esperar antes de enviar um *batch*. Pode-se comparar à propriedade “batch.size”, porém, em vez do limite ser por *size* é por tempo. O valor *default* é 0.

7. **"DELIVERY.TIMEOUT.MS":** consiste num limite superior no tempo para reportar sucesso ou falha após o retorno de uma chamada *send()*. Limita o tempo total que um registo será atrasado antes do envio, o tempo de espera de confirmação do broker (se esperado) e o tempo permitido para falhas de envio que podem ser repetidas[1]. O valor *default* é 120000.

Propriedades do Kafka Consumer

No casa do *kafka consumer* foram usadas duas propriedades.

1. **"FETCH.MIN.BYTES":** consiste na quantidade mínima de dados que o servidor deve retornar para uma *request* de *fetch*. Se não houver dados suficientes, a *request* aguardará que quantidade de dados seja acumulada antes de responder[1]. O valor *default* é 1.

2. **"ENABLE.AUTO.COMMIT":** Se *true*, o *offset* do *consumer* será confirmado periodicamente em segundo plano[1].

3. **"AUTO.OFFSET.RESET":** consiste no que fazer quando não houver *offset* inicial no *Kafka* ou se o *offset* atual não existir mais no servidor[1].

4. **"PARTITION":** nomeia uma partição específica da qual o *consumer* irá receber a informação. Útil para garantir ordem no consumo de mensagens[1].

5. **“GROUP.ID”:** Uma string exclusiva que identifica o grupo de *consumers* ao qual esse *consumer* pertence[1].

Casos de Uso

Neste bloco será apresentado o processo para o desenvolvimento de cada caso de uso(UC). Foram implementados seis casos, cada um com diferentes condições.

- **UC1**

Inclui um *kafka cluster* com: 1 Broker, 1 Topic chamado “Sensor”, 1 Partition.

  - **PSource**

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado para o *PProducer* via *Sockets* como já mencionado.

  - **PProducer**

Recebe as mensagens com os dados do *PSource* e envia-las como registo para o tópico do *kafka cluster*.

Contém apenas um *kafka producer*.

Com respeito aos valores das propriedades usados, estes foram:

- “**acks**” = 0: uma vez que mensagens podem ser perdidas.
- **“buffer.memory”** = 33554432: valor *default*.
- **“compression.type”** = none: valor *default*.
- **"retries"** = 2147483647: valor *default*. Contudo, o seu uso não é relevante uma vez que o valor de “acks” é 0 então, mensagens podem ser perdidas.
- **“batch.size”** = 16384: valor *default*.
- **"linger.ms"** = 0: valor *default*.
- **"delivery.timeout.ms"** = 120000: valor *default*.

  - **PConsumer**

Consome os registos do tópico.

Contém apenas um *kafka consumer*.

Com respeito aos valores das propriedades usados, estes foram:

- “**fetch.min.bytes**” = 1: valor *default*.
- **"enable.auto.commit"** = true: valor *default*.
- **“auto.offset.reset”** = latest: valor *default*.

- **UC2**

Inclui um *kafka cluster* com: 1 Broker, 1 Topic chamado “Sensor”, 6 Partitions.

  - **PSource**

Contém uma lista de *Sockets clients* associados a cada *Kafka producer*(seis *Kafka Producers* nesta *UC*).

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado ao *Kafka producer* de índice(na lista de *clients*) igual ao *ID* do sensor(na verdade *ID* - 1, uma vez que os índices começam em 0 e os *IDs* dos sensores em 1).

  - **PProducer**

Recebe as mensagens com os dados do *PSource* e envia-las como registo para o tópico do *kafka cluster*.

Contém seis *Kafka producers* e envia os registos para a partição de número igual ao seu *ID*(dado na criação dos *producers* - valor entre 0 e 5).

Com respeito aos valores das propriedades usados, estes foram:

- “**acks**” = 1: reduzir a perda de mensagens.
- **“buffer.memory”** = 33554432: valor *default*.
- **“compression.type”** = none: valor *default*, se for usado um método de compressão aumentaria a latência.
- **"retries"** = 2147483647: valor *default*.
- **“batch.size”** = 16384: valor *default*, se fosse maior aumentaria a latência.
- **"linger.ms"** = 0: valor *default*, se fosse maior aumentaria a latência.
- **"delivery.timeout.ms"** = 120000: valor *default*.

  - **PConsumer**

Consome os registos do tópico.

Contém seis *kafka consumers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**fetch.min.bytes**” = 1: valor *default*.
- **"enable.auto.commit"** = true: valor *default*.
- **“auto.offset.reset”** = latest: valor *default*.
- **"partition"** = *id* do *consumer*: recebe os registos somente de uma partição assim, cada *consumer* consome registos de apenas um *sensor ID*.

- **UC3**

Inclui um *kafka cluster* com: 1 Broker, 1 Topic chamado “Sensor”, 6 Partitions.

  - **PSource**

Contém uma lista de *Sockets clients* associados a cada *Kafka producer*(três *Kafka Producers* nesta *UC*).

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado ao *Kafka producer* de índice(na lista de *clients*) com o valor do resto da divisão do *sensor ID*(*ID* - 1) pelo número de *producers*, assim, os sensores de *IDs* “1” e “4” são enviados ao primeiro *producer* (índice 0).

  - **PProducer**

Tendo em conta que as mensagens não necessitam de serem consumidas por ordem, o *Producer r*ecebe as mensagens com os dados do *PSource* e executa uma *thread* para enviar o registo para o *kafka cluster*.

Contém três *Kafka producers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**acks**” = 1: reduzir a perda de mensagens, mas não reduzindo demasiado o *throughput* como no “acks” = all (*Trade-off*).
- **“buffer.memory”** = 33554432: valor *default*.
- **“compression.type”** = lz4: usado um método de compressão, o *throughput* aumenta.
- **"retries"** = 0: aumenta o *throughput,* apesar de propiciar a perda de mensagens.
- **“batch.size”** = 130000: valor mais elevado aumenta o *throughput*.
- **"linger.ms"** = 100: valor maior que 0 aumenta o *throughput*.
- **"delivery.timeout.ms"** = 120000: valor *default*.

  - **PConsumer**

Consome os registos do tópico.

Contém um grupo de três *kafka consumers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**fetch.min.bytes**” = 100000: valor mais elevado aumenta o *throughput*.
- **"enable.auto.commit"** = false: com o intuito de evitar o reprocessamento de dados, o commit é feito manualmente de forma síncrona.
- **“auto.offset.reset”** = latest: valor *default*.
- **"partition"** = *ID* do *consumer*: Uma vez que os *consumers* fazem parte de um grupo, cada um recebe os registos somente de uma partição.

- **UC4**

  Inclui um *kafka cluster* com: 6 Brokers, 1 Topic chamado “Sensor”, 1 Partitions, 3 réplicas e 2 min.insync.replicas. Usando *Kafka replication* consegue-se ter cópias dos dados em diferentes *brokers*, logo, mantém-se uma *availability* grande do sistema e evita perda de dados caso, por exemplo, um broker falhar.

  - **PSource**

Contém uma lista de *Sockets clients* associados a cada *Kafka producer*(seis *Kafka Producers* nesta *UC*).

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado ao *Kafka producer* de índice(na lista de *clients*) igual ao *ID* do sensor(na verdade *ID* - 1, uma vez que os índices começam em 0 e os *IDs* dos sensores em 1).

  - **PProducer**

Recebe as mensagens com os dados do *PSource* e envia-las como registo para o tópico do *kafka cluster*.

Contém seis *Kafka producers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**acks**” = all: garante que o registo não será perdido enquanto pelo menos uma réplica permanecer ativa.
- **“buffer.memory”** = 33554432: valor *default*.
- **“compression.type”** = none: valor *default*.
- **"retries"** = 2147483647: valor *default*.
- **“batch.size”** = 16384: valor *default*.
- **"linger.ms"** = 0: valor *default*.
- **"delivery.timeout.ms"** = 120000: valor *default*.

  - **PConsumer**

Consome os registos do tópico.

Contém um *kafka consumers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**fetch.min.bytes**” = 1: valor *default*.
- **"enable.auto.commit"** = true: valor *default*.
- **“auto.offset.reset”** = latest: valor *default*.

- **UC5**

  Inclui um *kafka cluster* com: 6 Brokers, 1 Topic chamado “Sensor”, 6 Partitions, 3 réplicas e 2 min.insync.replicas.

  - **PSource**

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado ao *Kafka producer*.

  - **PProducer**

Recebe as mensagens com os dados do *PSource* e envia-las como registo para o tópico do *kafka cluster*.

Contém um *Kafka producer*.

Com respeito aos valores das propriedades usados, estes foram:

- “**acks**” = all: valor *default*.
- **“buffer.memory”** = 33554432: valor *default*.
- **“compression.type”** = none: valor *default*.
- **"retries"** = 2147483647: valor *default*.
- **“batch.size”** = 16384: valor *default*.
- **"linger.ms"** = 0: valor *default*.
- **"delivery.timeout.ms"** = 120000: valor *default*.

  - **PConsumer**

Consome os registos do tópico.

Contém três grupos de três *kafka consumers*.

Com respeito aos valores das propriedades usados, estes foram:

- “**fetch.min.bytes**” = 1: valor *default*.
- **"enable.auto.commit"** = true: valor *default*.
- **“auto.offset.reset”** = latest: valor *default*.

Neste caso de uso não foi possível implementar a *Voting Replication tactic*.

- **UC6**

  Inclui um *kafka cluster* com: 6 Brokers, 1 Topic chamado “Sensor”, 6 Partitions, 3 réplicas e 2 min.insync.replicas.

  - **PSource**

Lê o ficheiro linha a linha e guarda a informação numa lista “*content*” e, com auxílio de um contador, a cada três linhas(registo completo - id do sensor, temperatura e *timestamp*) envia o registo guardado ao *Kafka producer*.

- **PProducer**

Recebe as mensagens com os dados do *PSource* e envia-las como registo para o tópico do *kafka cluster*.

Contém um *Kafka producer*.

Com respeito aos valores das propriedades usados, estes foram:

- "**acks**" = all: valor *default*.
- **"buffer.memory"** = 33554432: valor *default*.
- **"compression.type"** = none: valor *default*.
- **"retries"** = 2147483647: valor *default*.
- **"batch.size"** = 16384: valor *default*.
- **"linger.ms"** = 0: valor *default*.
- **"delivery.timeout.ms"** = 120000: valor *default*.

- **PConsumer**

Consome os registos do tópico.

Contém um *kafka consumers*.

Com respeito aos valores das propriedades usados, estes foram:

- "**fetch.min.bytes**" = 1: valor *default*.
- **"enable.auto.commit"** = true: valor *default*.
- **"auto.offset.reset"** = latest: Se o *consumer* falhar, as suas partições serão re-atribuídas a outro membro, que iniciará o consumo a partir do último *offset* confirmado de cada partição[3].

Referências

[1] https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html
[2] https://docs.confluent.io/cloud/current/client-apps/optimizing/latency.html
[3] https://docs.confluent.io/platform/current/clients/consumer.html