POLITECNICO
MILANO 1863

Apache Kafka

Alessandro Margara
alessandro.margara@polimi.it
https://margara.faculty.polimi.it

Rules

- Rename the ConsumersXX.java file replacing XX with the number of your group
- Write in the comments on top of the class your group number and the name of all group members
- Write additional information on topic partitioning and consumer groups in the comments on top, as discussed in the following slides
- Submit only a single java file with your solution
  - Submitted from the contact email provided in the group registration document

Assumptions

- Some producers produce messages on two topics *sensors1* and *sensors2*
  - The provided *Producer* class simulates these producers
  - You may set the number of partitions for *sensors1* and *sensors2* using the *TopicManager* class
  - Message keys are strings, message values are integers
  - You may assume that producers never crash and the communication between the producers and the *sensors1* and *sensors2* topics are reliable
    - All messages are always delivered once and only once

Assumptions

- A *Merger* component reads from both *sensors1* and *sensors2*

- For every key, *Merger* outputs the sum of the last value read for that key from *sensors1* and the last value read for that key from *sensors2*
  - Output messages have the same key as the corresponding input messages
  - Output messages are written on a *merged* topic
  - You may assume the initial value to be 0 for every key

Assumptions

- The *Merger* component needs to guarantee *at least once* semantics, meaning that each input message is evaluated at least once (producing an output message at each evaluation)

- Your implementation should try to minimize the number of duplicated messages in the case of a failure
  - Always restarting from the beginning of the topic in the case of failure is not an option ☺

Assumptions

- A *Validator* component consumes messages from the *merged* topic and forwards them to both an *output1* and an *output2* topics

- Validator needs to guarantee *atomicity* of propagation, meaning that each message is propagated to either both or none of the *output1* and *output2* topics

Exercise

- Complete the implementation of *Merger* and *Validator* in the ConsumersXX.java file

- The Consumers main method takes in input two parameters:
  - The component to start (1=*Merger* or 2=*Validator*)
  - The consumer group
  - You may add additional parameters, if needed

- Answer to the questions on top of the file on how you can configure topics, partitions, instances, and consumer groups

Hints

- You can add print statements to your consumers, to verify that they work as expected
  - You can make the producers deterministic to test your solution

- You can assume the assignment of keys to partitions to be determinitic and identical for any topic that has the same number of partitions

- You can assume the assignment of partitions to consumers within a consumer group to be deterministic and identical for any topic that has the same number of partitions

- You may use the producer.flush() method to synchronously wait for an acknowledgement of that all the messages sent by a producer have been stored in the corresponding topic