**Loi Cheng**

October 25 2019

Harvard Extension - Big Data Principles e88

Homework 8: Kafka

Please identify which problems were completed. If any were incomplete, please identify where you encountered problems.

|  |
| --- |
| Problem 1: 100%  Problem 2: 100%  Problem 3: 100%  Problem 4: 100%  Bonus Problem A: 100%  Bonus Problem B: 100% |

**Problem 1: Simple Kafka Setup [20 points]**

Show main steps and config settings you had to do to setup your 3-node Kafka cluster; show commands you used to start Zookeeper, Kafka cluster and create a new topic; and a few lines of the output on the console [5 points]

|  |
| --- |
| * Creating a 3-node Kafka cluster using docker compose ( as demoed in the Lab) * Bringing it up using --scale kafka=3      * Connecting to Running kafka container-1 * issuing kafka-topics --describe command and verify there are no topics in this cluster yet      * Creating a topic "problem1" with 2 partitions and replication factor = 2 |

Show a screenshot of your Kafka cluster state - output of the kafka-topics --describe command [5 points]

|  |
| --- |
| * Describing the cluster and show its state via kafka-topics --describe command |

Demo the console producer/consumer work [5 points]

|  |
| --- |
| * Starting a kafka console consumer in a new terminal and leave it running      * Open another terminal and start kafka console producer in an interactive mode and try sending some messages. Messages sent in producer should show up in console consumer |

Show output of the DumpLogSegments tool for the 'problem1' topic [5 points]

|  |
| --- |
|  |

**Problem 2: Flume + Kafka integration [25 points]**

Paste your Flume config file below [5 point]

|  |
| --- |
| flume-conf-kafka1.properties  # Licensed to the Apache Software Foundation (ASF) under one  # or more contributor license agreements.  See the NOTICE file  # distributed with this work for additional information  # regarding copyright ownership.  The ASF licenses this file  # to you under the Apache License, Version 2.0 (the  # "License"); you may not use this file except in compliance  # with the License.  You may obtain a copy of the License at  #  #  http://www.apache.org/licenses/LICENSE-2.0  #  # Unless required by applicable law or agreed to in writing,  # software distributed under the License is distributed on an  # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  # KIND, either express or implied.  See the License for the  # specific language governing permissions and limitations  # under the License.  # The configuration file needs to define the sources,  # the channels and the sinks.  # Sources, channels and sinks are defined per agent,  # in this case called 'a1’  a1.sources = r1  #  a1.sinks = kafkasink  a1.channels = fc  a1.sources.r1.type = exec  a1.sources.r1.channels = fc  a1.sources.r1.command = tail -F /var/log/apache2/access.log  # Interceptor  a1.sources.r1.interceptors = i1  a1.sources.r1.interceptors.i1.type = org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder  a1.sources.r1.interceptors.i1.headerName = UUID  #Each sink's type must be defined  a1.sinks.loggerSink.type = logger  # Specify the channel the sink should use  a1.sinks.loggerSink.channel = fc  a1.sinks.kafkasink.type = org.apache.flume.sink.kafka.KafkaSink  a1.sinks.kafkasink.topic = problem2\_a  a1.sinks.kafkasink.brokerList = assignment8kafka\_kafka\_1:9092  a1.sinks.kafkasink.channel = fc  a1.sinks.kafkasink.batchSize = 20  # set roll to 10 seceonds  a1.sinks.kafkasink.sink.rollInterval = 30  # add header and text to sink  a1.sinks.kafkasink.sink.serializer = header\_and\_text  a1.sinks.kafkasink.sink.serializer.appendNewline = true  # channel configuration  a1.channels.fc.type = file  a1.channels.fc.capacity = 100  a1.channels.fc.transactionCapacity = 100  a1.channels.fc.checkpointDir = /dockershared/flume\_checkpoint  a1.channels.fc.dataDirs = /dockershared/flume\_data |

Demo that your Kafka console consumer now receives events from your Web server's log in the 'problem2' topic; show content of the web server logs - and corresponding events in the Kafka topic [10 point]

|  |
| --- |
| Curl python generated 15 call    15 entries in web log    Kafka consumer received 15 messages      Offsets show 7+8 = 15 |

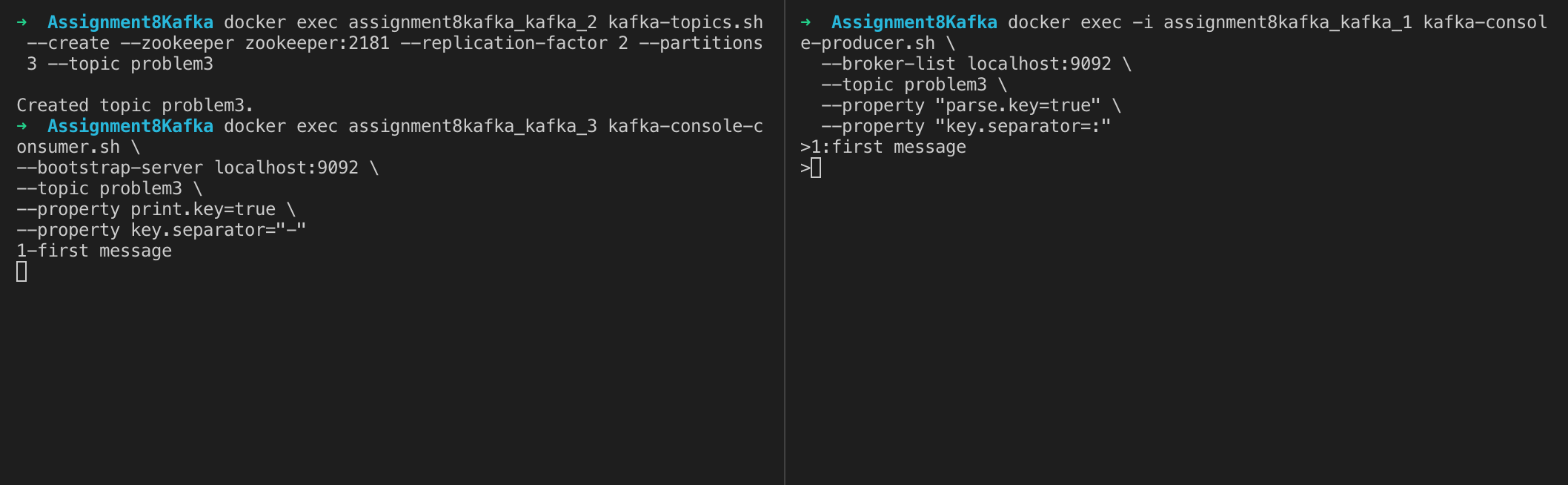
Show changes in your Flume config file to add UUID interceptor [5 points]

|  |
| --- |
| The UUID interceptor header name is renamed to “key”  flume-conf-kafka2.properties  # Licensed to the Apache Software Foundation (ASF) under one  # or more contributor license agreements.  See the NOTICE file  # distributed with this work for additional information  # regarding copyright ownership.  The ASF licenses this file  # to you under the Apache License, Version 2.0 (the  # "License"); you may not use this file except in compliance  # with the License.  You may obtain a copy of the License at  #  #  http://www.apache.org/licenses/LICENSE-2.0  #  # Unless required by applicable law or agreed to in writing,  # software distributed under the License is distributed on an  # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY  # KIND, either express or implied.  See the License for the  # specific language governing permissions and limitations  # under the License.  # The configuration file needs to define the sources,  # the channels and the sinks.  # Sources, channels and sinks are defined per agent,  # in this case called 'a1’  a1.sources = r1  #  a1.sinks = kafkasink  a1.channels = fc  a1.sources.r1.type = exec  a1.sources.r1.channels = fc  a1.sources.r1.command = tail -F /var/log/apache2/access.log  # Interceptor  a1.sources.r1.interceptors = i1  a1.sources.r1.interceptors.i1.type = org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder  a1.sources.r1.interceptors.i1.headerName = key  #Each sink's type must be defined  a1.sinks.loggerSink.type = logger  # Specify the channel the sink should use  a1.sinks.loggerSink.channel = fc  a1.sinks.kafkasink.type = org.apache.flume.sink.kafka.KafkaSink  a1.sinks.kafkasink.topic = problem2\_b  a1.sinks.kafkasink.brokerList = assignment8kafka\_kafka\_1:9092  a1.sinks.kafkasink.channel = fc  a1.sinks.kafkasink.batchSize = 20  # set roll to 10 seceonds  a1.sinks.kafkasink.sink.rollInterval = 30  # add header and text to sink  a1.sinks.kafkasink.sink.serializer = header\_and\_text  a1.sinks.kafkasink.sink.serializer.appendNewline = true  # channel configuration  a1.channels.fc.type = file  a1.channels.fc.capacity = 100  a1.channels.fc.transactionCapacity = 100  a1.channels.fc.checkpointDir = /dockershared/flume\_checkpoint  a1.channels.fc.dataDirs = /dockershared/flume\_data |

Demo that events in the 'problem2' topic now have UUIDs - how can you show that? [5 points]

|  |
| --- |
| The consumer now shows UUID as the key |

**Problem 3: Programming with Kafka APIs [30 points]**

****

Paste your Problem3Producer code below [5 points]

|  |
| --- |
| # Using Kafka API, create a new Problem3Producer application that generates and sends events to the "problem3" topic - generate the same type of log events we've worked with so far, plus add a UUID:  # one event is one line in the format: <uuid> <timestamp> <url> <userId>  # make sure you have at least 3 different userIDs  from kafka import KafkaProducer  import time  import random  import uuid  producer = KafkaProducer(bootstrap\_servers='kafka:9092')  # debug only  # producer.send('problem3', key=str.encode(str(1)), value=str.encode(str(2))) #this is async  # producer.send('problem3', key=b'foo', value=b'bar') #this is async  # while True:  #     time.sleep(1)  userList = ['Loi','Mao','Jesus']  urlList = ['www.awesome.cow','somthing.onion','peacefulriot.org']  partNum = 0  var = 1  while var == 1 :      #set partition      partNum += 1      partNum = 0 if partNum == 3 else partNum      #build message      newUUID = uuid.uuid1()      userId = random.choice(userList)      timestamp = time.time()      newUrl = random.choice(urlList)      msg = str(newUUID) +':' +str(timestamp) +':' +newUrl +':' +userId      msg = str.encode(msg)      print(msg)      #send message      producer.send('problem3', partition=partNum, value=msg, key=b'pythonproduce')      print('message sent')      #set partition      time.sleep(2) |

Show how you run your producer and demo the results of its work by using the Kafka console consumer [5 points]

|  |
| --- |
| The producer is run from the docker image with python – see right console below  The consumer outputs the same messages that were produced – see left console below |

Paste your Problem3Consumer code below [5 points]

|  |
| --- |
| # Using Kafka APIs, create a Problem3Consumer (single-threaded is OK), with the "p3consumer" consumer group.id,  that listens to the "problem3" topic - demonstrate that it receives events generated by the Problem3Producer  # For each event received, the Problem3Consumer should print out its offset, partition number, event key and event body  from kafka import KafkaConsumer  var = 1  while var == 1 :      consumer = KafkaConsumer(bootstrap\_servers='kafka:9092',group\_id='p3consumer',auto\_offset\_reset='latest')      #consumer = KafkaConsumer(bootstrap\_servers='localhost:32787',client\_id="2",group\_id='consumer-1',auto\_offset\_reset='latest')      consumer.subscribe(['problem3'])      for message in consumer:          #print (type(message)) #debug          #print (message) #debug          print ('offset:', end=' ')          print (message.offset, end=' ')          print ('partition:', end=' ')          print (message.partition, end=' ')          print ('event key:', end=' ')          print (message.key.decode("utf-8"), end=' ')          print ('event body:', end=' ')          print (message.value.decode("utf-8")) |

Show commands you use to run your consumer and demo how it works - receiving events sent by your Problem3Producer and printing out offsets, partitions, headers and bodies of the events [10 points]

|  |
| --- |
|  |

Demo that the events sent by your producer are fairly distributed across all partitions of the 'problem3' topic [5 points]

|  |
| --- |
| Using kafka-consumer-groups.sh, the offsets show the messages are fairly distributed across all the partition at 22 each. |

**Problem 4: Kafka consumer groups [25 points]**

Demo running **two** instances of your Problem3Consumer, with the same consumer group ID, listening to the 'problem3' topic with 3 partitions - which instances are processing which partitions? [10 points]

|  |
| --- |
| It looks like one consumer is processing from partition 0 and 1, and the second consumer is processing only from partition 2 |

Demo running **four** instances of your Problem3Consumer, with the same consumer group ID, listening to the 'problem3' topic with 3 partitions - which instances are processing which partitions? Are all instances processing data? Explain the results [10 points]

|  |
| --- |
| It looks like one consumer has stopped processing data, while the other three are each processing from one partition only. |

Demo re-balancing of the consumers again, after bringing down two instances of the Problem3Consumer [5 points]

|  |
| --- |
| After bringing down two instances so that there are only two remaining, one consumer is processing from partition 0 and 1, and the second consumer is processing only from partition 2 |

**Bonus Problem A: partitioning by userID [+10 points]**

Paste your ProblemAProducer code below

|  |
| --- |
| # Using Kafka API, create a new Problem3Producer application that generates and sends events to the "problem3" topic - generate the same type of log events we've worked with so far, plus add a UUID:  # one event is one line in the format: <uuid> <timestamp> <url> <userId>  # make sure you have at least 3 different userIDs  from kafka import KafkaProducer  import time  import random  import uuid  producer = KafkaProducer(bootstrap\_servers='kafka:9092')  # debug only  # producer.send('problem3', key=str.encode(str(1)), value=str.encode(str(2))) #this is async  # producer.send('problem3', key=b'foo', value=b'bar') #this is async  # while True:  #     time.sleep(1)  userList = ['Loi','Mao','Jesus']  urlList = ['www.awesome.cow','somthing.onion','peacefulriot.org']  partNum = 0  var = 1  while var == 1 :      #build message      newUUID = uuid.uuid1()      userId = random.choice(userList)      timestamp = time.time()      newUrl = random.choice(urlList)      msg = str(newUUID) +':' +str(timestamp) +':' +newUrl +':' +userId      msg = str.encode(msg)      print(msg)      #set partition      partNum = userList.index(userId)      #send message      producer.send('problem3', partition=partNum, value=msg, key=b'pythonproduce')      print('message sent to ' + str(partNum))      #set partition      time.sleep(2) |

Demo its work by running the Problem3Consumer - show that events with the same userID are received from the same partition

|  |
| --- |
| With three consumers running, each of them is only processing from one partition, and the same userID are received from the same partition |

**Bonus Problem B: ISR or not ISR? [+15 points]**

Show status of the cluster by describing all topics; show state of the new 'problemb' topic - which nodes are leaders for which partitions?

|  |
| --- |
| Node 1002 is leader for partition 0, 1003 for 1, 1001 for 2, 1002 for 3 |

Show commands you used to bring down Kafka on one node. Demo the results - show how ISRs are changing for the 'problemb' topic; are you still able to send/receive events via console producer/consumers ? Demo these operations

|  |
| --- |
| Docker network disconnect command was used on kafka\_1  The leaders were automatically reassigned  The ISR in 1001 is lost  We can still send/received events via console producer/consumers by connecting to the other nodes |

Demo effects of bringing the 'failed' Kafka node back up - how long did it take to get ISRs back in sync?

|  |
| --- |
| Kafka\_1 is reconnected to the network, and the ISR’s were back in sync almost immediately    We can connect to kafka\_1 again as consumer and producer |