**Assignment 8: Kafka**

In this assignment, we are trying to build another part of a Big Data pipeline - collection tier plus messaging tier. All work for this assignment can be done either on a local PC/laptop or on an EC2 instance.

For a 3-node Kafka cluster setup:

* you have to run the 3-node Kafka cluster, as well as all other utilities (console consumer and producer), in Docker containers, using Docker compose
* we provide full instructions on how to do this using Docker compose in the Lab
* you can elect to run a 1-node Kafka cluster installed locally on your laptop, and everything else too - the easiest option - but you will be deducted 20 points from your total HW points

Example code will be provided in the usual GIT repo, "week8" folder/package.

**Make sure you submit both the Solution document (this one, with your answers), as well as all your code in an archive**

**Problem 1: [Points: 25] Basic Kafka setup**

* Create a 3-node Kafka cluster using provided docker compose file
* Once the cluster is started (you can check it by running **docker logs broker1 | grep -i started** command or **nc -z localhost 9092**), issue **docker exec -it broker1 /bin/kafka-topics --bootstrap-server broker1:29092 --describe** command

and verify there are some topics in this cluster.

Show the output here: [10 points]

|  |
| --- |
|  |

* Create a topic "problem1" with 2 partitions and replication factor = 2
* Describe the cluster and show its state via kafka-topics --describe command - show the new topic info: [5 points]

|  |
| --- |
|  |

* Start a kafka console consumer in a new terminal and leave it running
  + **docker exec -it broker1 /bin/kafka-console-consumer.sh --bootstrap-server broker1:29092 --topic problem1**
* Open another terminal and start kafka console producer in an interactive mode and try sending some messages. Messages sent in the producer should show up in the console consumer
  + **docker exec -it broker1 /bin/kafka-console-producer.sh --bootstrap-server broker1:29092 --topic problem1**

Demo the console producer/consumer work [5 points]

|  |
| --- |
|  |

* Examine kafka data logs, by using the kafka.tools.DumpLogSegments tool , for each partition of the "problem1" topic. Show the output output: [5 points]

|  |
| --- |
|  |

**Problem 2: [Points: 25] Filebeat + Kafka integration**

* Create a new topic, "problem2" , with 10 partitions and replication factor = 2
* Modify provided filebeat.yml to set up your new topic as output for Filebeat.
* Make sure your output kafka sink points to your kafka cluster

Paste your Filebeat config file below; highlight the config for the Kafka output - which port is Filebeat using to connect to Kafka? [5 point]

|  |
| --- |
|  |

* Use a setup demonstrated in the lab, Apache server + Filebeat (see provided filebeat docker compose file - it will also be demoed in the Lab)
  + we will be running Apache httpd service in a docker container and simulate hitting some url by issuing a health check of the service - this will generate apache log entries in the apache log files
  + the apache log files will be read/harvested by the Filebeat and sent to Kafka

Show your Docker compose file content below; show all running containers - output of the 'docker ps -a' command [10 point]

|  |
| --- |
| version: "3.8"  networks:    default:      external:        name: lab8-demo-net  *#  lab8-demo:*  *#    name: lab8-demo-net*  *#    driver: bridge*  services:  *#Filebeat container*    filebeat:      container\_name: filebeat      hostname: filebeat      user: root      image: "docker.elastic.co/beats/filebeat:8.1.0"      volumes:  *#Mount the apache2 logs into the filebeat container so we can access and index them using the filebeat*        - apachelog:/usr/local/apache2/logs/  *#Mount the filebeat configuration so users can make edit*        - ./config/filebeat.yml:/usr/share/filebeat/filebeat.yml  *#    networks:*  *#      - lab8-demo*      restart: on-failure      command: filebeat -d "\*" -strict.perms=false      deploy:        mode: global  *# depends\_on:*  *#   #wait for the these services to come up and healthy.*  *#   apache2: { condition: service\_healthy }*  *#Apache2 container*    apache2:      container\_name: apache2      hostname: apache2      build: ./apache2  *#    image: httpd:2.4*      ports:        - 8080:80  *#    image: 'bitnami/apache:latest'*  *#    healthcheck:*  *#      test: ["CMD-SHELL", "apt-get update -y && apt-get install -y curl && curl --fail http://localhost || exit 1"]*  *#      interval: 3s*  *#      timeout: 10s*  *#      retries: 3*  *#      start\_period: 0s*  *#    networks:*  *#      - lab8-demo*      volumes:        - apachelog:/usr/local/apache2/logs/  *#      - ./apache2/httpd.conf:/usr/local/apache2/conf/httpd.conf*  volumes:    apachelog:      driver: local |

* Start a Kafka console consumer for "problem2" topic - verify you are getting your web server logs (based on apache health checks gets)

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]

|  |
| --- |
|  |

**Problem 3 [Points: 25]: Programming with Kafka APIs**

* Create a new topic, "problem3" , **with 3 partitions**
* Using Kafka API, create a new Problem3Producr application (by updating the provided Producer application) to generate events in the format: <uuid>, <timestamp>, <userId = your\_first\_last\_name> to the "problem3" topic. Paste your Problem3Producer code below [5 points]

|  |
| --- |
| from kafka import KafkaProducer  import time  import datetime  import uuid as id  producer = KafkaProducer(bootstrap\_servers='localhost:9092')  while True:      uuid = str(id.uuid4())      timestamp = str(datetime.datetime.utcnow())      userID = "Choi\_Mason"      producer.send("problem3", value=str.encode(**f**"{uuid}, {timestamp}, {userID}"))      print(**f**"{uuid}, {timestamp}, {userID}")      time.sleep(1) |

* Verify your producer works correctly by receiving generated events via Kafka console consumer. Show how you run your producer and demo the results of its work by using the Kafka console consumer [5 points]

|  |
| --- |
|  |

* Using Kafka APIs, create a Problem3Consumer (by updating the provided code), with the "p3consumer" consumer group.id, that listens to the "problem3" topic; For each event received, the Problem3Consumer should print out its offset, partition number, event key and event body. Paste your Problem3Consumer code below [5 points]

|  |
| --- |
| from kafka import KafkaConsumer  partition\_spread = [0, 0, 0]  while True:      consumer = KafkaConsumer(bootstrap\_servers='localhost:9092',group\_id='p3consumer',auto\_offset\_reset='latest')      consumer.subscribe(['problem3'])      for message in consumer:          print(**f**"offset={message.offset}, partition={message.partition}, key={message.key}, value={message.value}")          partition\_spread[message.partition] += 1          print("Partition spread:", partition\_spread) |

* demonstrate that it receives events generated by the Problem3Producer and that it receives events from all 3 partitions and they are equally distributed.

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 [5 points]

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

* Building up on Problem 3's setup: start a second instance of the Problem3Consumer, with the same consumer group ID , 'p3consumer'
* Demonstrate how partitions are re-balanced between the two consumer instances - which partitions are processed by which instance now?

[10 points]

|  |
| --- |
| With 2 consumers within the same consumer group, they try to share the partitions. My second consumer, shown in the screenshot, is only getting the first 2 partitions while my first is only getting the 3 partition of 3, demonstrating how Kafka is trying to spread out the partitions among all of the consumers within a single group. |

* Start two more instances of the Problem3Consumer with the same group ID - to the total of 4 instances now. Demonstrate the partition re-balancing and explain the results. Which instances are processing which partitions, if any? Explain the results [5 points]

|  |
| --- |
| With the addition of 2 more consumers, Kafka tries to balance the 3 partitions among 4 consumers, making it so that 3 consumers handle 3 different partitions while the 4 does nothing. In the screenshot, I see that this consumer which used to receive partitions 1 and 2 is now only receiving partition 1 to account for the additional consumers. |

* Stop two instances of the Problem3Consumer - demonstrate the partition re-balancing again. Explain the results[10 points]

|  |
| --- |
| When I removed 2 consumers, the others went back to consuming 1 partition and the other 2 partitions. In the screenshot I see that a consumer that was previously only consuming the third partition is now only consuming the first 2. |

**Problem 5: Bonus: [Points: 10] ISR or not ISR?**

* use the same 3-node Kafka cluster setup
* Create a topic, "problem5", with 4 partitions and replication factor 2
* Examine the state of the cluster and show how all replicas are spread over the cluster nodes - show all ISRs. [2 points]

|  |
| --- |
|  |

* Using console producer send a few events to the "problem5" topic; verify you have received them using the console consumer: [2 points]

|  |
| --- |
|  |

* Stop Kafka on one of the nodes, wait for a few seconds - show the state of the cluster now, and all ISRs of the "problem5" topic: [2 points]

|  |
| --- |
|  |

* Run console producer and consumer again, see if you can still send/receive messages: [2 points]

|  |
| --- |
|  |

* re-start Kafka on that node - observe how fast the cluster gets into the full ISR state:[2 points]

|  |
| --- |
|  |

**Problem 6: Bonus: [Points: 10] partitioning by userID**

* Create a new Problem6Producer, by modifying your Problem3Producer, to send events to Kafka partitioned based on the userIDs:
  + Events with the same userID should go to the same partition number
  + Make sure that there is at least one userID corresponding to each partition (you have 3 partitions, remember)

Show the code of your producer: [5 points]

|  |
| --- |
|  |

* Start your Problem6Consumer and verify that the events for the same userID are received from the same partition - demo this: [5 points]

|  |
| --- |
|  |