# ACIT 3855 – Lab 9 – Issue Fixing

| | |
|---|---|
| **Instructor** | Mike Mulder (mmulder10@bcit.ca) – Sets A and C<br>Tim Guicherd (tguicherd@bcit.ca) – Set B |
| **Total Marks** | 10 |
| **Due Dates** | Demo by end of next class:<br>• March 25th for Set C<br>• March 28th for Sets A and B |

**Purpose**

- Fix the software issues we've encountered so far in building our software system
- Start to think about the process for troubleshooting and correcting software issues

**Make sure to test each of your fixes before moving on to the next.**

## Part 1 – Dashboard UI: Audit Log Indices Out of Synch with Data

**Note: This isn't needed if you are using the vanilla HTML/CSS/JS dashboard. But make sure you are getting a random audit event up to index from 0 to 100.**

Observed: The displayed Audit Log index and the corresponding Event are out of synch. The index is ahead of the data (i.e., the event data is for the previously displayed index).

Expected: The Audit Log index and data should be synchronized.

Solution: In the EndpointAudit.js file:

Add a state for the index:

```
const [index, setIndex] = useState(null);
```

Set the index upon successfully response from the audit endpoints:

```
setIndex(rand_val);
```

Update the view to display the index:

```
return (
    <div>
        <h3>{props.endpoint}-{index}</h3>
        {JSON.stringify(log)}
    </div>
)
```

## Part 2 – Kafka Topics Aren't Persisting

This was intentionally done to allow for debugging and troubleshooting without having to deal with old data in the Kafka message queue.

Observed: The events topic in the Kafka broker does not persistent when Kafka is brought back up, such as when you stop your VM, start it again and then run “docker-compose up –d”.

Expected: Kafka topics are meant to persist.

Solution: The kafka logs (i.e., where the messages on the topics are stored) are being stored on the container but are lost when the container is removed. The solution is to mount a volume or bind a VM folder to the container for the /kafka directory. In addition, the Zookeeper data also needs to persiste to keep it in synch with Kafka.

Create the following two folders on your VM:

- /home/<username>/zookeeper/data
- /home/<username>/kafka

Add the following to your docker-compose.yml file.

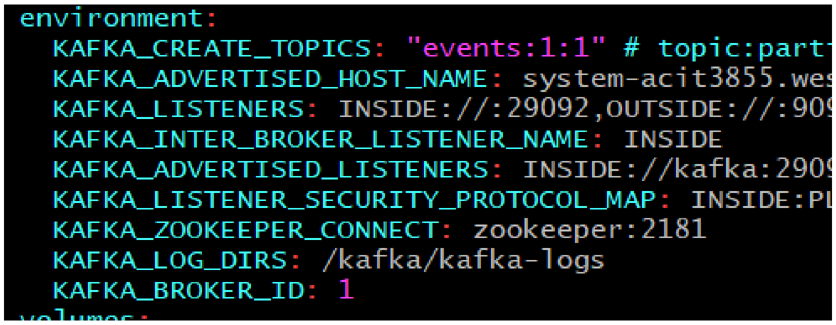
For the zookeeper service add a volume (bind mount):

```
  volumes:
    - /home/<username>/zookeeper/data:/opt/zookeeper-3.4.13/data
```

For the kafka service (bind mount):

```
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - /home/<username>/kafka:/kafka/kafka-logs
```

Also add the KAFKA_LOG_DIRS and KAFKA_BROKER_ID to the environment for the kafka service:



```
environment:
  KAFKA_CREATE_TOPICS: "events:1:1" # topic:part
  KAFKA_ADVERTISED_HOST_NAME: system-acit3855.we
  KAFKA_LISTENERS: INSIDE://:29092,OUTSIDE://:90
  KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE
  KAFKA_ADVERTISED_LISTENERS: INSIDE://kafka:290
  KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INSIDE:PL
  KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
  KAFKA_LOG_DIRS: /kafka/kafka-logs
  KAFKA_BROKER_ID: 1
volumes:
```

This assigns a broker id to kafka (which it uses for the stored data) and the folder in which to store the logs data (which refers to the data in the kafka topics and partitions, not application logs).

Now when you docker-compose down and then docker-compose up -d your services, the messages on the events topic in Kafka will persist.

**Part 4 – Storage Service Does Not Connect on Startup**

Observed: Storage Service sometimes does not connect to Kafka on startup, requiring the service to be stopped and restarted.

Expected: The Storage Service should be able to connect to Kafka on startup, even if it has to wait until Kafka is up and running.

Solution: The following code from the process_messages function is trying to connect to Kafka but is failing silently (no exception handling) and there is no re-try logic:

```
client = KafkaClient(hosts=hostname)
topic = client.topics[str.encode(app_config["events"]["topic"])]
```

You should add re-try logic to this function. Here is some pseudo code:

```
Define a maximum number of retries (this should be obtained from your configuration file for the service)
Set the current retry count to zero.
While the current retry count is less than the maximum number of retries:
    • Display an info log message indicating you are trying to connect to Kafka and the current retry count
    • Create a KafkaClient and get the topic (i.e., the code above)
    • If an exception is raised (hint: add exception handling - try/except - in the while loop):
        ▪ Display an error log message indicating that the connection failed
        ▪ Sleep for a configurable number of seconds (hint: import time and use time.sleep, obtain the sleep time for your configuration file)
        ▪ Increment the current retry count
```

**Part 5 – (Improvement) Receiver Service Reconnects to Kafka for Each Event**

This one you need to figure out yourself. Here is the approach you could take:

- Create the KafkaClient when your Receiver Service starts up, not in the functions for each endpoint.
- Add the re-try logic as in Part 4.
- The function for each endpoint will now produce the event on the topic.
- Make sure you include the producer = topic.get_sync_producer() line in your retry logic (and not in each function) to make your Receiver *really* fast!

**Part 6 – (Issue) Storage Service Losing Connection**

It appears that the Storage Service still may lose its Kafka connection after some time, approximately 5-10 minutes

Observed: After a period of time, when jMeter is used to send in events to the system, the Storage Service will receive the first event and then not process any subsequent events until it is stopped and restarted.

Expected: The Storage Service should process events indefinitely until it is stopped or the Kafka broker goes down.

Investigate the options on the create_engine function in SQLAlchemy, specifically for the pool_size, pool_recycle and pool_pre_ping.

The first student in each set to demonstrate a working solution (and then sharing it with the set) will get a 2 mark bonus on this lab.

**Grading and Submission**

Submit the following to the Lab 9 Dropbox on D2L:

- The app.py file for your Storage Service with the retry logic in process_messages.
- The app.py file for your Receiver Service with the single Kafka connection.

| | |
|---|---|
| Show the code for each of your solutions above. | 5 marks |
| Demonstrate your system with jMeter proving the following:<br>• The Dashboard UI shows the correct Audit Event data<br>• The events persist in the Kafka events topic after a docker-compose down followed by a docker-compose up –d<br>• The Processing Service has the correct statistics (within 5% of the expected)<br>• The Storage Service doesn’t need to be restarted after a docker-compose down followed by a docker-compose up –d<br>• Receiver Service still works with one-time Kafka connection<br>• Storage Service continues to connect to the MySQL database after 5-10 minutes | 5 marks |
| Storage Service connection solution (first in Set) | Bonus |
| **Total** | **10 marks** |