Event Processing Project SoSe 2022

Ahmet Turkmen

Abstract

In this paper, demonstration of event processing on Kafka + KSQL streaming platform will be considered and discussed. The input data is around 356 MB and it is structured data. Go programming language is used to parse and publish data to Kafka instances. Kafka + KSQL environment is set with official Docker-compose file from Github.

ACM Reference Format:
Ahmet Turkmen. 2022. Event Processing Project SoSe 2022. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Streaming Messages

In order to stream messages to Kafka cluster, there is a need of a producer which will read and send all data to Kafka cluster. The streaming part, the producer, of the project completed with Go programming language. The program creates the topic on Kafka cluster and pushes events on the given file directly to Kafka cluster.

1.1 Kafka Setup

Initially, Confluent cloud[1] trial version is used, however due to unreliable network connection on producer side and high latency on Cloud side, this approach is dismissed. The main idea became to setup Kafka cluster along with ksqldb. It can be achieved by simple Docker Compose file provided by confluent repository on Github[?] Provided "docker-compose.yaml" file contains all required services (Zookeeper, ksqldb, ksqldb-ci, control-center, kafka and more) and additionally a web interface to easy access to topics and streams. Running instances can be observed through given Figure 1.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
┌─[thinkpad@ThinkPad-T470s-W10DG]─[~/GolandProjects/event-processing]
└─► $docker-compose up -d
Creating network "event-processing_default" with the default driver
Creating zookeeper ... done
Creating broker    ... done
Creating schema-registry ... done
Creating connect         ... done
Creating rest-proxy      ... done
Creating ksqldb-server   ... done
Creating ksqldb-cli      ... done
Creating ksql-datagen    ... done
Creating control-center  ... done
```

Figure 1. Running Docker Instances

Once all the containers are in running state, now streaming messages can start through Go program.

1.2 Stream Events

The source of the events is a file which needs to be processed with a producer. Producer will read through the file and push them to Kafka cluster under given topic name. Figure 2 represents the source code which is responsible to stream data to created topic.

```
CreateTopic(p, TOPIC)

f, err := os.Open( name: "./data/events.json")
if err != nil {
    log.Fatal(err)
}
defer f.Close()
scanner := bufio.NewScanner(f)
for scanner.Scan() {
    p.Produce(&kafka.Message{
        TopicPartition: kafka.TopicPartition{
            Topic:     &TOPIC,
            Partition: kafka.PartitionAny,
        },
        Value: scanner.Bytes(),
        Key:   []byte("events"),
    }, deliveryChan: nil)
    e := <-p.Events()
    message := e.(*kafka.Message)
    if message.TopicPartition.Error != nil {
        fmt.Printf( format: "failed to deliver message: %v\n",
            message.TopicPartition)
    } else {
        fmt.Printf( format: "delivered to topic %s [%d] at offset %v\n",
            *message.TopicPartition.Topic,
            message.TopicPartition.Partition,
            message.TopicPartition.Offset)
    }
}
p.Close()

fmt.Printf("Events are produced to #{TOPIC} topic \n")
```

Figure 2. Source code for streaming messages

The given events source file is scanned and pushed to topic with help of channels in Go programming language shown

in Figure 2. The full source code of the project is accessible on Github[1]

2 Filtering

Creating filters are done through ksqlDB, it is purpose built-in for stream processing applications[2]. As events stream to a topic, it is possible to filter and stream filtered events to another topic based on conditions defined in the query. It is achievable through ksqlDB query in Kafka. In order to filter messages received from a producer, a stream needs to set, it can be done with clarifying incoming message skeleton with fields[2].

```
CREATE STREAM meetup_events_stream (
    utc_offset BIGINT,
    venue STRUCT <country VARCHAR, city VARCHAR, address_1 VARCHAR,
    name VARCHAR, lon DOUBLE, lat DOUBLE>,rsvp_limit INT, venue_visibility VARCHAR,
    visibility VARCHAR, maybe_rsvp_count INT, description VARCHAR, mtime BIGINT,
    event_url VARCHAR, yes_rsvp_count INT, payment_required INT, name VARCHAR,
    id BIGINT, `time` BIGINT,`group` STRUCT <joined_mode VARCHAR, country VARCHAR,
    city VARCHAR, name VARCHAR, group_lon DOUBLE,id BIGINT, urlname VARCHAR,
    category STRUCT <name VARCHAR, id INT, shortname VARCHAR>,
    group_photo STRUCT <highres_link VARCHAR, photo_link VARCHAR, photo_id BIGINT, thumb_link VARCHAR>,
    group_lat DOUBLE>, status VARCHAR) WITH (KAFKA_TOPIC='the-meetup-events', VALUE_FORMAT='JSON');
```

Figure 3. Creating a stream from a topic

The query on Figure 3, creates a stream from a Kafka topic which is fed by a producer. As "the-meetup-events" topic fed by producer, stream "meetup_events_stream" consumes messages from the topic. The streams enables data to analyze, manipulate, transform and filter incoming messages from given topic.

2.1 Country filtering

Kafka streams are straightforward to filter, in Figure 4, incoming messages filtered based on country information they include.

```
CREATE STREAM country_filtered AS
    SELECT * FROM meetup_events_stream WHERE VENUE -> COUNTRY='de' EMIT CHANGES;
```

Figure 4. Filter country from a stream

New stream called country_filtered is created and it is saved as persistent query in ksqlDB. Persistent queries run and listen given topic to consume in background.

2.2 City Filtering

In order to narrow down the search to Munich or München, two different streams are created from "meetup_events_stream" stream as source.

```
CREATE STREAM city_filtered_munich AS
    SELECT * FROM COUNTRY_FILTERED WHERE VENUE -> CITY='Munich' EMIT CHANGES;
```

Figure 5. Filter "Munich" city

[1]https://github.com/mrtrkmn/kafka-event-processing
[2]https://ksqldb.io/

```
CREATE STREAM city_filtered_muenchen AS
    SELECT * FROM COUNTRY_FILTERED WHERE VENUE -> CITY='München' EMIT CHANGES;
```

Figure 6. Filter "München" city

Both streams, "city_filtered_munich" and "city_filtered_muenchen" are consuming messages by filtering from "country_filtered" stream. Overall stream flow of different stream connections is shown in Figure 7.

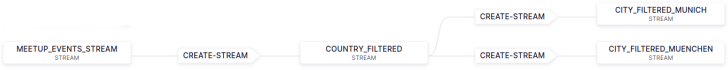

Figure 7. Overall stream flow

3 Conclusion

In this project, Kafka + ksqldb are used as main environment, provided events file processed with created producer. The messages are filtered with help of streams on ksqlDB. However, the environment used in this project is not fault tolerant, since replication count is limited to one due to local instance. Even though number of partitions is not limited to one, it is not fault tolerant. Since a problem on the instance may cause to lost all data due to replication count being one. Throughput, latency graphs along with topics view included in Appendix. The end to end average latency is higher at the beginning and decreases to a point where it continues linear. The latency can be minimized with batch processing of events however, it has not been implemented in this experiment. The throughput of the environment is affected by system's capabilities (CPU power) and processing method of the events.

References

[1] Confluent. 2022. *Confluent Cloud: Fully Managed Kafka as a Cloud-Native Service.* Retrieved June 20, 2022 from https://www.confluent.io/confluent-cloud/

[2] Confluent. 2022. *Writing Streaming Queries Against Apache Kafka Using KSQL.* Retrieved July 5, 2022 from https://docs.confluent.io/5.4.2/ksql/docs/developer-guide/create-a-stream.html

[3]]cp-all-in-one Rick Spurgeon Srini Dandu Allison Walther Arnaud Esteve JT Smith Brett Randall Yeva Byzek, andrewegel. [n. d.]. *Confluent: cp-all-in-one.* Retrieved June 25, 2022 from https://github.com/confluentinc/cp-all-in-one

Appendix

4 Overview of the cluster

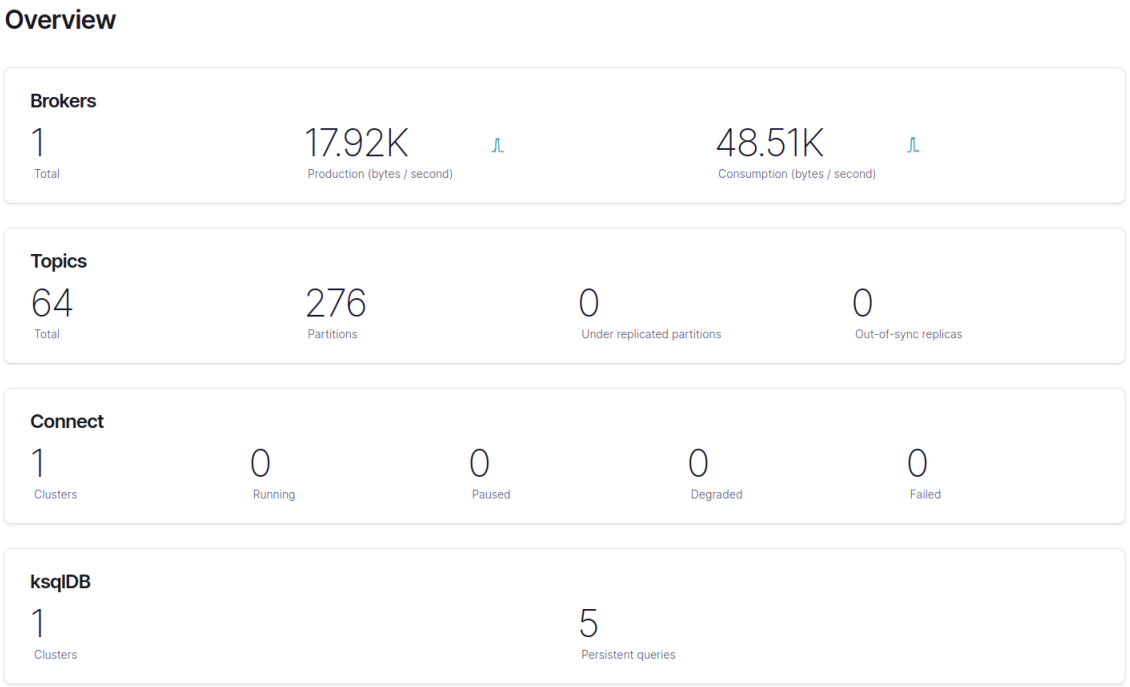

Figure 8. Cluster overview

* Shown number of topics include hidden internal topics on Cluster Overview

5 The average end-to-end latency

Figure 9. The average end-to-end latency

6 Throughput graph

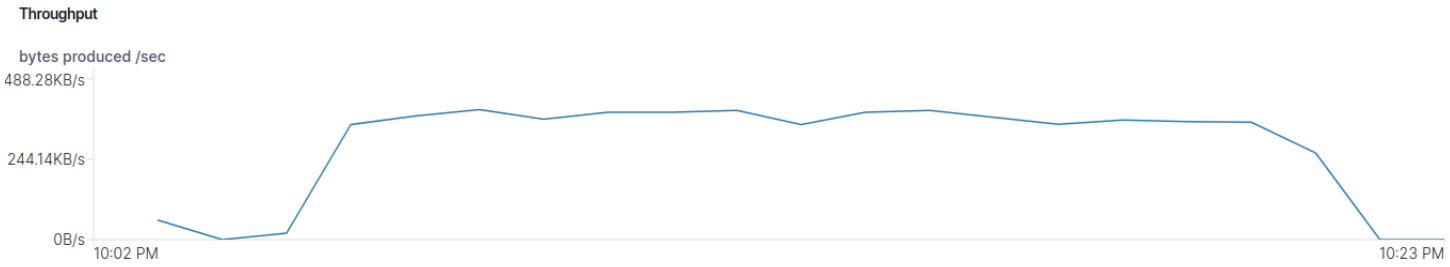

Figure 10. Throughput graph

7 Consumed throughput graph

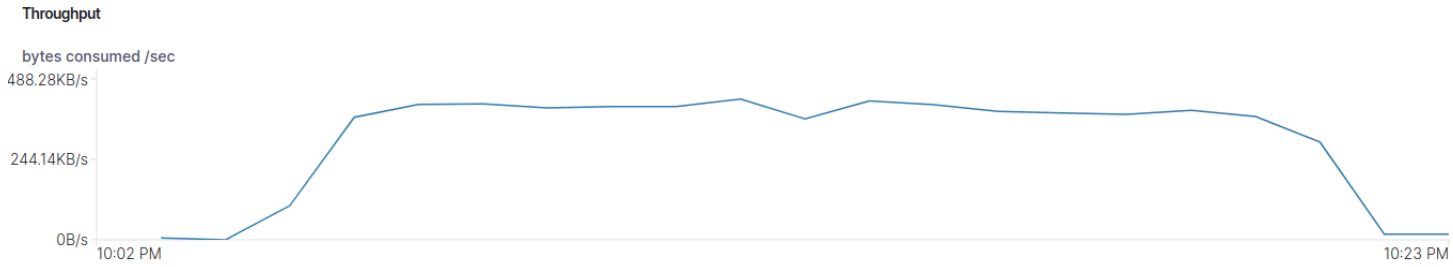

Figure 11. Consumed throughput graph

8 Topics

Topic name	Status	Partitions	Production (last 5 mins)	Consumption (last 5 mins)
CITY_FILTERED_MUENCHEN	Healthy --	1	--	--
CITY_FILTERED_MUNICH	Healthy --	1	--	--
COUNTRY_FILTERED	Healthy --	1	3.23KB/s	9.68KB/s
default_ksql_processing_log	Healthy --	1	--	--
docker-connect-configs	Healthy --	1	0B/s	16B/s
docker-connect-offsets	Healthy --	25	--	0B/s
docker-connect-status	Healthy --	5	--	0B/s
the-meetup-events	Healthy --	1	0B/s	34.13KB/s

Figure 12. Topics table

* Topics are automatically generated, topics which starts with 'docker' and 'default' are internal topics.