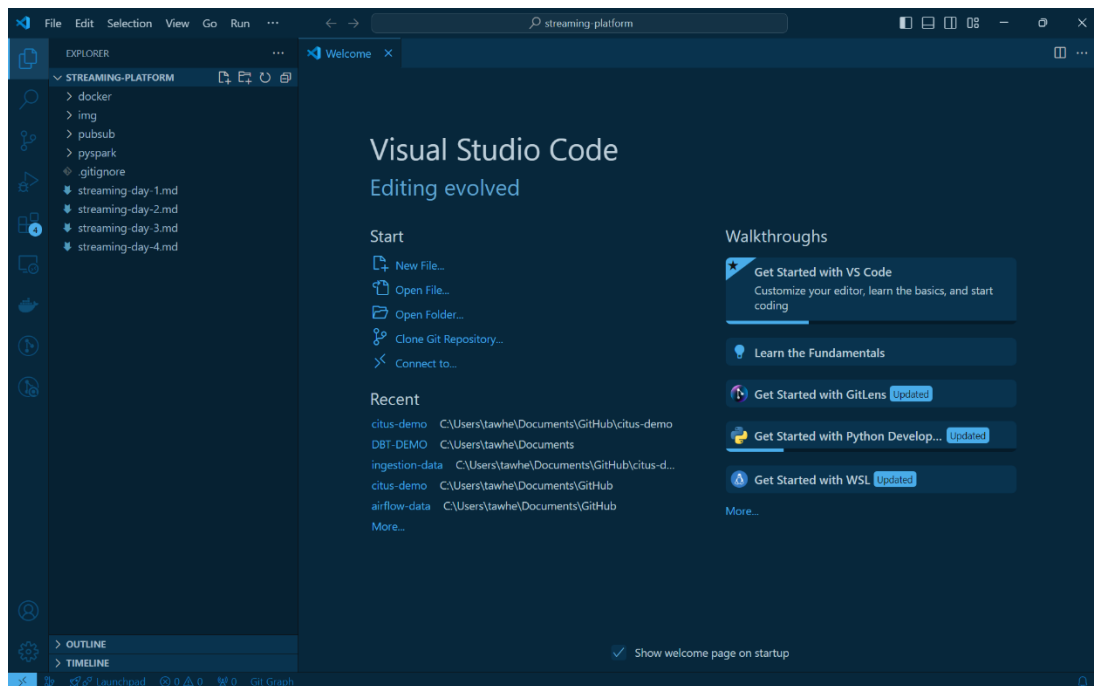
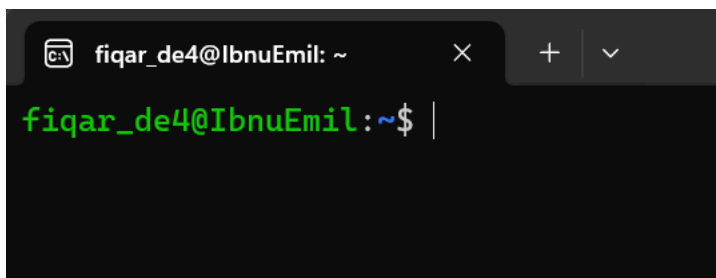


## Stream Processing Task

1. Pertama lakukan clone repository Immersive-DataEngineer-Resource / streaming-platform, lalu buka dengan visual studio code.



2. Kemudian buka aplikasi Ubuntu pada windows.



3. Kemudian masukkan secure shell “ssh [raja\\_rahmanakmaludin@34.101.224.54](mailto:raja_rahmanakmaludin@34.101.224.54)” dan password “mentoralterra2024”.

```
fiqar_de4@IbnuEmil:~$ ssh raja_rahmanakmaludin@34.101.224.54
raja_rahmanakmaludin@34.101.224.54's password:
Linux instance-20240714-035051 6.1.0-23-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.99-1 (2024-07-15) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Aug  1 12:03:38 2024 from 203.175.125.135
```

Berikut ini tampilan setelah melakukan langkah nomor 3.

4. Kemudian lakukan perintah “ls” untuk melihat apa saja folder / file yang ada.

```
raja_rahmanakmaludin@instance-20240714-035051:~$ ls
airflow-data  dbt-demo  extract-load-demo  ingestion-data  streaming-platform
```

Berikut ini folder yang tersedia.

5. Lalu masuk ke folder “streaming-platform” dengan menjalankan perintah “cd streaming-platform”

```
raja_rahmanakmaludin@instance-20240714-035051:~$ cd streaming-platform/  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform$ |
```

Sekarang posisinya sudah masuk ke folder “streaming-platform”

6. Di dalam folder tersebut, jalankan perintah “ls” lagi untuk melihat apa saja folder / file yang ada. Kemudian jika ada folder “docker”, masuk ke folder tersebut dengan menjalankan perintah “cd docker”. Di dalam folder tersebut jalankan perintah “ls” lagi untuk melihat folder / file yang tersedia. Jika ada folder redpanda, maka masuk ke folder tersebut dengan menjalankan perintah “cd redpanda”. Setelah itu di dalam folder “redpanda” lakukan perintah “ls” untuk melihat file apa yang ada di folder tersebut. Berikut ini adalah gambaran pada server saat melakukan langkah – langkah yang sudah dijelaskan sebelumnya.

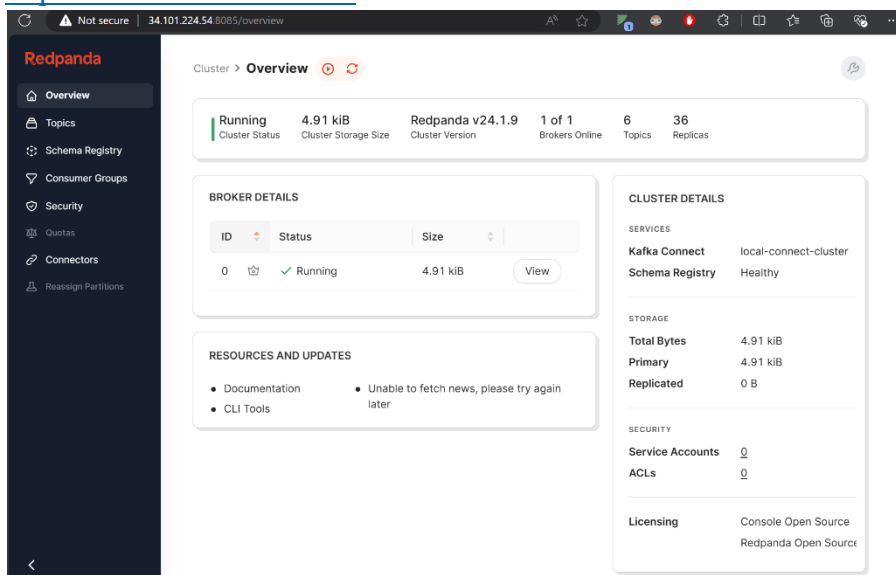
```
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform$ ls  
docker img pubsub pyspark streaming-day-1.md streaming-day-2.md streaming-day-3.md streaming-day-4.md  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform$ cd docker/  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform/docker$ ls  
redpanda spark  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform/docker$ cd redpanda/  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform/docker/redpanda$ ls  
docker-compose-spark.yml docker-compose.yml  
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform/docker/redpanda$ |
```

7. Pada folder tersebut jalankan docker dengan perintah “sudo docker compose -f docker-compose.yml up -d”.

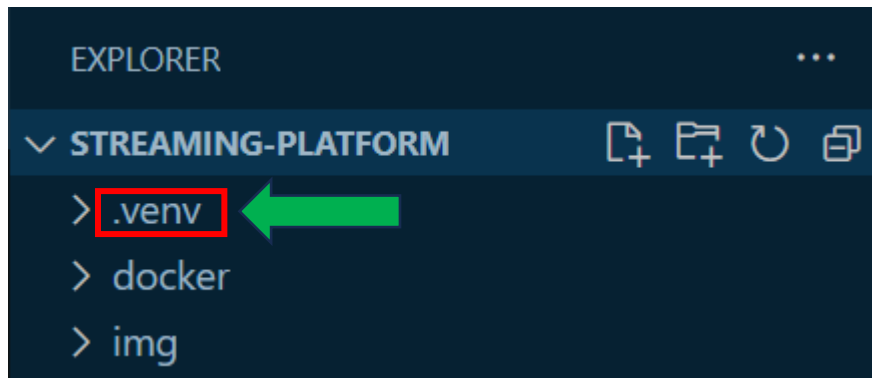
```
raja_rahmanakmaludin@instance-20240714-035051:~/streaming-platform/docker/redpanda$ sudo docker compose -f docker-compose.yml up -d  
WARN[0000] /home/raja_rahmanakmaludin/streaming-platform/docker/redpanda/docker-compose.yml: 'version' is obsolete  
[+] Running 7/7  
✔ Network redpanda_redpanda_network Created 0.2s  
✔ Container ksqldb-server Started 1.2s  
✔ Container connect Started 1.1s  
✔ Container postgres Started 0.9s  
✔ Container redpanda Started 1.1s  
✔ Container ksqldb-cli Started 2.2s  
✔ Container console Started 2.0s
```

Berikut ini adalah tampilan saat perintah sudo docker compose -f docker-compose.yml up -d” berhasil dijalankan.

8. Setelah itu masuk ke website redpanda dengan memasukkan URL <http://34.101.224.54:8085/>.



9. Lalu kembali ke aplikasi visual studio code dan jalankan environment dengan perintah “python -m venv .venv” lalu aktifkan juga environment tersebut dengan perintah “source .venv/Scripts/activate”.



10. Kemudian install “confluent\_kafka” dengan menjalankan perintah “pip install confluent\_kafka”.

```
tawhe@Ibnu_Emil MINGW64 ~/Documents/streaming-platform/pubsub/json (main)
• $ pip install confluent_kafka
Collecting confluent_kafka
  Downloading confluent_kafka-2.5.0-cp312-cp312-win_amd64.whl.metadata (2.4 kB)
  Downloading confluent_kafka-2.5.0-cp312-cp312-win_amd64.whl (3.5 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.5/3.5 MB 848.1 kB/s eta 0:00:00
Installing collected packages: confluent_kafka
Successfully installed confluent_kafka-2.5.0

[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv)
tawhe@Ibnu_Emil MINGW64 ~/Documents/streaming-platform/pubsub/json (main)
```

Berikut ini tampilan setelah install “confluent\_kafka”.

11. Pada folder “pubsub\json\produce.py” buatlah script seperti berikut ini :

```
pubsub > json > produce.py > ...
1  from confluent_kafka import Producer
2  from datetime import datetime
3  import random
4  import json
5  import uuid
6  #from config import TOPIC
7
8  TOPIC="fiqarrachman_stock_json_topic"
9
10 def produce():
11     # Configure the Producer
12     p = Producer({
13         'bootstrap.servers': '34.101.224.54:19092', # Assuming you're running this on the same machine as the compose
14         'client.id': 'python-producer'
15     })
16
17     # Produce a message
18     countdata = 0
19     try:
20         while True:
21             stock = {
22                 'event_time': datetime.now().isoformat(),
23                 'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
24                 'price': round(random.random() * 100, 2)
25             }
26             p.produce(TOPIC, key=str(uuid.uuid4), value=json.dumps(stock), callback=delivery_report)
27             countdata+=1
28             if countdata ==2000:
29                 break;
30             else:
31                 pass
32     except Exception as e:
33         print(str(e))
34
35     # Wait for any outstanding messages to be delivered
36     p.flush()
37
38 def delivery_report(err, msg):
39     if err is not None:
40         print('Message delivery failed: {}'.format(err))
41     else:
42         print('Message delivered to {} [{}]' .format(msg.topic(), msg.partition()))
43
44 def main():
45     produce()
46
47 if __name__ == "__main__":
48     main()
```

12. Kemudian pada folder “pubsub\json” eksekusi script yang sudah dibuat dengan menjalankan perintah “python produce.py”.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
Message delivered to fiqarrachman_stock_json_topic [0]
(.venv)
tawhe@Ibnu_Emil MINGW64 ~/Documents/streaming-platform/pubsub/json (main)
$
```

Berikut ini adalah tampilan bahwa perintah menjalankan file dengan nama “produce.py” sudah berhasil.

13. Setelah itu, buat script seperti berikut ini pada file dengan nama “consume.py”.

```
pubsub > json > consume.py > ...
1  from confluent_kafka import Consumer, KafkaException, KafkaError
2  #from config import TOPIC
3
4
5  def consume():
6      # Configure the Consumer
7      c = Consumer({
8          'bootstrap.servers': '34.101.224.54:19092', # Assuming you're running this on the same machine as the compose
9          'group.id': 'fiqarrachman',
10         'auto.offset.reset': 'latest'
11     })
12
13     # Subscribe to the topic
14     TOPIC="fiqarrachman_stock_json_topic"
15     c.subscribe([TOPIC])
16
17     # Process messages
18     try:
19         while True:
20             msg = c.poll(timeout=1.0)
21             if msg is None:
22                 continue
23             if msg.error():
24                 if msg.error().code() == KafkaError._PARTITION_EOF:
25                     # End of partition event
26                     print('%s [%d] reached end at offset %d\n' %
27                           (msg.topic(), msg.partition(), msg.offset()))
28                 elif msg.error():
29                     raise KafkaException(msg.error())
30             else:
31                 print('Received message: {}'.format(msg.value().decode('utf-8')))
32     except KeyboardInterrupt:
33         pass
34     finally:
35         # Close down consumer to commit final offsets.
36         c.close()
37
38 def main():
39     consume()
40
41 if __name__ == '__main__':
42     main()
```

14. Lalu coba jalankan juga file dengan nama “consume.py” dengan menjalankan perintah “python consume.py”

```
(.venv)
tawhe@Ibnu_Emil MINGW64 ~/Documents/streaming-platform/pubsub/json (main)
$ python consume.py
```

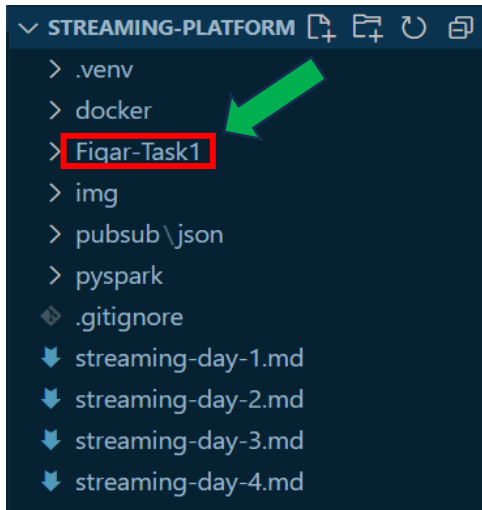
15. Kemudian kita cek apakah topic sudah berhasil dibuat dalam website redpanda.

Create Topic ☐ Show internal topics

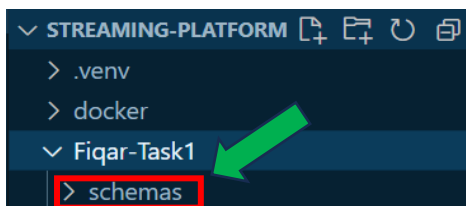
Name	Partitions	Replicas	CleanupPolicy	Size
connect_configs	1	1	compact	1.26 KiB
connect_offsets	25	1	compact	2.83 KiB
connect_statuses	5	1	compact	580 B
Fiqar_DE4_stock_avro_topic	1	1	delete	176 KiB
fiqarrachman_stock_json_topic	1	1	delete	667 KiB
stock_avro_topic_has	1	1	delete	10.7 MiB
Task1_AnggiSetiawan_DE4	1	1	delete	116 B

Total 7 items < 1 > 50 / page

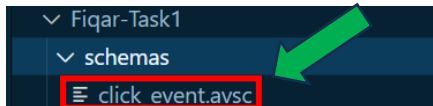
16. Setelah itu untuk tugas avro, sebelum membuat file ekstensi python, perlu membuat terlebih dahulu folder yang diinginkan. Dalam hal ini membuat folder dengan nama “Fiqar-Task1”.



17. Setelah membuat folder, buat lagi di dalam folder “Fiqar-Task1” dengan nama folder “schemas”.



18. Di dalam folder “schemas” buat file dengan nama “click\_event.avsc”.

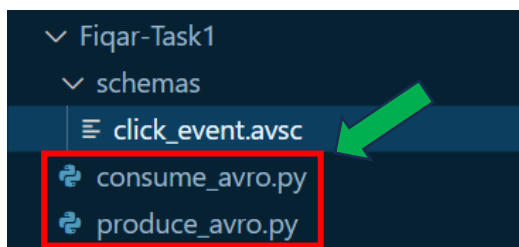


19. Buat script seperti berikut ini pada file dengan nama “click\_event.avsc” tersebut.

```
Fiqar-Task1 > schemas > click_event.avsc
1  {
2      "type": "record",
3      "namespace": "com.redpanda.examples.avro",
4      "name": "Event",
5      "fields": [
6          { "name": "user_id", "type": "int" },
7          { "name": "event_type", "type": { "type": "enum", "name": "EventType", "symbols": ["CLICK", "PAGE_VIEW", "PURCH"] },
8          { "name": "ts", "type": "string" },
9          { "name": "page_url", "type": ["null", "string"], "default": null },
10         { "name": "product_id", "type": ["null", "string"], "default": null },
11         { "name": "user_email", "type": ["null", "string"], "default": null },
12         { "name": "location", "type": ["null", "string"], "default": null },
13         { "name": "device_type", "type": ["null", "string"], "default": null }
14     ]
15 }
```

Berikut ini adalah tampilan dari isi file “click\_event.avsc”

20. Setelah itu buat file dengan nama “produce\_avro.py” dan “consume\_avro.py”.



21. Pada file “produce\_avro.py” buat script seperti berikut ini :

```
Fiqr-Task1 > produce_avro.py > produce
1  import json
2  from uuid import uuid4
3  from confluent_kafka import KafkaException
4  from confluent_kafka.avro import AvroProducer
5  from confluent_kafka import avro
6
7  def delivery_callback(error, message):
8      if error:
9          print(f"Failed to send the message: {error}")
10     else:
11         print(f"Message with the key {message.key()} has been produced to the topic {message.topic()}")
12
13 def load_avro_schema_from_file():
14     key_schema_string = """
15     {"type": "string"}
16     """
17     key_schema = avro.loads(key_schema_string)
18     value_schema = avro.load('/Users/tawhe/Documents/streaming-platform/Fiqr-Task1/schemas/click_event.avsc')
19     return key_schema, value_schema
20
21 def produce():
22     config = {
23         'bootstrap.servers': "34.101.224.54:19092",
24         'schema.registry.url': "http://34.101.224.54:18081"
25     }
26
27     key_schema, value_schema = load_avro_schema_from_file()
28
29     producer = AvroProducer(
30         config,
31         default_key_schema=key_schema,
32         default_value_schema=value_schema
33     )
34
35     event_types = ["CLICK", "PAGE_VIEW", "PURCHASE", "USER_REGISTRATION", "LOGIN", "LOGOUT"]
36     countdata = 0
37
38     try:
39         while countdata < 2000:
40             key = str(uuid4())
41             event_type = event_types[countdata % len(event_types)]
42
43             value = {
44                 "user_id": 2,
45                 "event_type": event_type,
46                 "ts": "2023-12-12",
47                 "page_url": "http://example.com" if event_type in ["PAGE_VIEW", "CLICK"] else None,
48                 "product_id": "prod123" if event_type == "PURCHASE" else None,
49                 "user_email": "user@example.com" if event_type == "USER_REGISTRATION" else None,
50                 "location": "New York" if event_type in ["LOGIN", "LOGOUT"] else None,
51                 "device_type": "mobile" if event_type in ["CLICK", "PAGE_VIEW"] else None
52             }
53
```

```

54         producer.produce(
55             topic="Fiqar_DE4_stock_avro_topic",
56             key=key,
57             value=value,
58             on_delivery=delivery_callback
59         )
60         countdata += 1
61
62         producer.poll(10000)
63         producer.flush()
64
65     except KafkaException as e:
66         print(f"Error occurred during message production: {e}")
67
68     print("Done!")
69
70 def main():
71     produce()
72
73 if __name__ == "__main__":
74     main()

```

Gambar diatas merupakan isi dari file “produce\_avro.py”.

22. Setelah itu buat juga script untuk file “consume\_avro.py”.

```

Fiqar-Task1 > consume_avro.py > ...
1  from confluent_kafka import KafkaError, KafkaException
2  from confluent_kafka.avro import AvroConsumer
3  from confluent_kafka.avro.serializer import SerializerError
4
5  def consume():
6      config = {
7          'bootstrap.servers': '34.101.224.54:19092',
8          'group.id': 'Fiqar_DE4_consumer_group',
9          'schema.registry.url': 'http://34.101.224.54:18081',
10         'auto.offset.reset': 'earliest'
11     }
12
13     consumer = AvroConsumer(config)
14     consumer.subscribe(['Fiqar_DE4_stock_avro_topic'])
15
16     try:
17         while True:
18             try:
19                 message = consumer.poll(1.0)
20
21                 if message is None:
22                     continue
23
24                 if message.error():
25                     if message.error().code() == KafkaError._PARTITION_EOF:
26                         print(f"Reached end at {message.topic()} [{message.partition()}] offset {message.offset()}")
27                     else:
28                         raise KafkaException(message.error())
29                 else:
30                     value = message.value()
31                     event_type = value.get("event_type")
32
33                     if event_type == "CLICK":
34                         # Handle CLICK event
35                         pass
36                     elif event_type == "PAGE_VIEW":
37                         # Handle PAGE_VIEW event
38                         pass
39                     elif event_type == "PURCHASE":
40                         # Handle PURCHASE event
41                         pass
42                     elif event_type == "USER_REGISTRATION":
43                         # Handle USER_REGISTRATION event
44                         pass

```



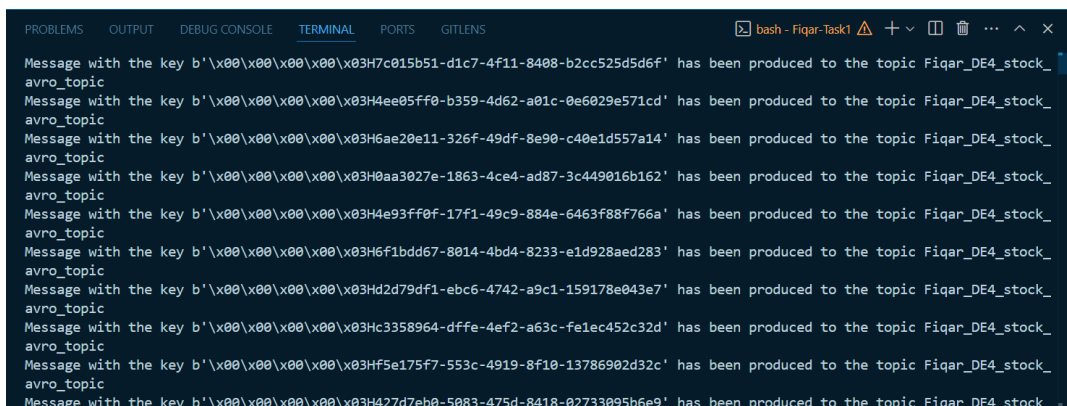
```

45         elif event_type == "LOGIN":
46             # Handle LOGIN event
47             pass
48         elif event_type == "LOGOUT":
49             # Handle LOGOUT event
50             pass
51
52         print(f"Consumed message from topic {message.topic()}, partition {message.partition()}, offset {message.offset()}")
53         print(f"Key: {message.key()}, Value: {value}")
54
55     except SerializerError as e:
56         print(f"Message deserialization failed: {e}")
57         continue
58
59     except KeyboardInterrupt:
60         pass
61     finally:
62         consumer.close()
63         print("Consumer closed")
64
65 def main():
66     consume()
67
68 if __name__ == "__main__":
69     main()

```

Gambar diatas merupakan isi dari file “consume\_avro.py”

23. Setelah file “produce\_avro.py” dan “consume\_avro.py” sudah dibuat. Jalankan file tersebut dengan perintah “python produce\_avro.py” untuk file “produce\_avro.py”. Dan perintah “python consume\_avro.py” untuk file “consume\_avro.py”.

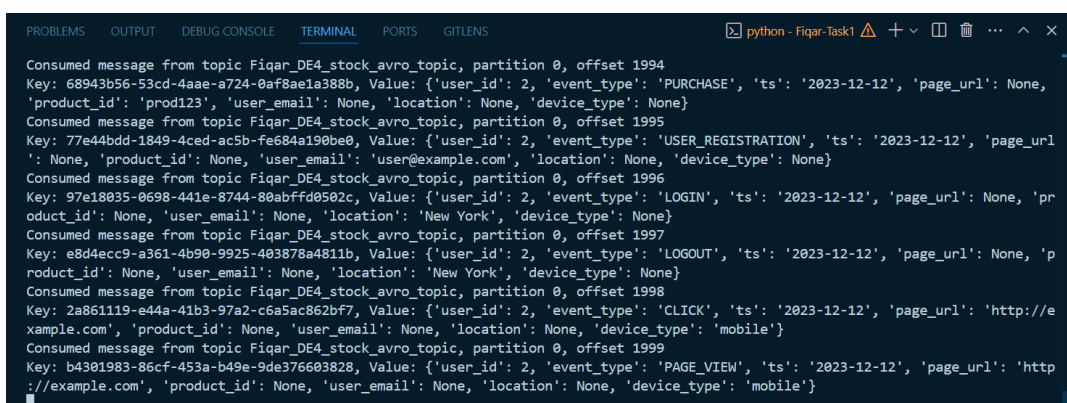


```

Message with the key b'\x00\x00\x00\x03H7c015b51-d1c7-4f11-8408-b2cc525d5d6f' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H4ee05ff0-b359-4d62-a01c-0e6029e571cd' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H6ae20e11-326f-49df-8e90-c40e1d557a14' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H0aa3027e-1863-4ce4-ad87-3c449016b162' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H4e93ff0f-17f1-49c9-884e-6463f88f766a' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H6f1bdd67-8014-4bd4-8233-e1d928aed283' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03Hd2d79df1-ebc6-4742-a9c1-159178e043e7' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03Hc3358964-dffe-4ef2-a63c-fe1ec452c32d' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03Hf5e175f7-553c-4919-8f10-13786902d32c' has been produced to the topic Fiqr_DE4_stock_avro_topic
Message with the key b'\x00\x00\x00\x03H427d7eb0-5083-475d-8418-02733095b6e9' has been produced to the topic Fiqr_DE4_stock_avro_topic

```

Berikut ini adalah proses saat menjalankan perintah “python produce\_avro.py”.



```

Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1994
Key: 68943b56-53cd-4aae-a724-0af8ae1a388b, Value: {'user_id': 2, 'event_type': 'PURCHASE', 'ts': '2023-12-12', 'page_url': None, 'product_id': 'prod123', 'user_email': None, 'location': None, 'device_type': None}
Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1995
Key: 77e44bdd-1849-4ced-ac5b-fe684a190be0, Value: {'user_id': 2, 'event_type': 'USER_REGISTRATION', 'ts': '2023-12-12', 'page_url': None, 'product_id': None, 'user_email': 'user@example.com', 'location': None, 'device_type': None}
Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1996
Key: 97e18035-0698-441e-8744-80abffd0502c, Value: {'user_id': 2, 'event_type': 'LOGIN', 'ts': '2023-12-12', 'page_url': None, 'product_id': None, 'user_email': None, 'location': 'New York', 'device_type': None}
Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1997
Key: e8d4ecc9-a361-4b90-9925-403878a4811b, Value: {'user_id': 2, 'event_type': 'LOGOUT', 'ts': '2023-12-12', 'page_url': None, 'product_id': None, 'user_email': None, 'location': 'New York', 'device_type': None}
Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1998
Key: 2a861119-e44a-41b3-97a2-c6a5ac862bf7, Value: {'user_id': 2, 'event_type': 'CLICK', 'ts': '2023-12-12', 'page_url': 'http://example.com', 'product_id': None, 'user_email': None, 'location': None, 'device_type': 'mobile'}
Consumed message from topic Fiqr_DE4_stock_avro_topic, partition 0, offset 1999
Key: b4301983-86cf-453a-b49e-9de376603828, Value: {'user_id': 2, 'event_type': 'PAGE_VIEW', 'ts': '2023-12-12', 'page_url': 'http://example.com', 'product_id': None, 'user_email': None, 'location': None, 'device_type': 'mobile'}

```

Berikut ini adalah proses saat menjalankan perintah “python consume\_avro.py”

24. Setelah menjalankan kedua file tersebut, langkah selanjutnya adalah memeriksa ke dalam website redpanda, apakah sudah berjalan sesuai yang diinginkan atau belum. Berikut ini adalah seluruh hasil dari script yang sudah dijalankan.

### “Topics”

The screenshot shows the Redpanda web interface. On the left is a sidebar with navigation links: Overview, Topics, Schema Registry, Consumer Groups, Security, Quotas, Connectors, and Reassign Partitions. The main panel is titled 'Cluster > Topics' and shows 8 total topics and 36 total partitions. A 'Create Topic' button is visible. Below it is a table of topics:

Name	Partitions	Replicas	CleanupPolicy	Size
connect_configs	1	1	compact	701 B
connect_offsets	1	1	compact	2.83 kiB
connect_statuses	1	1	compact	580 B
<b>Fiqr_DE4_stock_avro_topic</b>	1	1	delete	176 kiB
fiqarrachman_stock_json_topic	1	1	delete	422 kiB
stock_avro_topic_has	1	1	delete	10.7 MiB
Task1_AnggiSetiawan_DE4	1	1	delete	116 B
task1_wartadi_clickstream_avro	1	1	delete	177 kiB

A green arrow points to the 'Fiqr\_DE4\_stock\_avro\_topic' row, which is also highlighted with a red box. At the bottom right, it says 'Total 8 items' and '50 / page'.

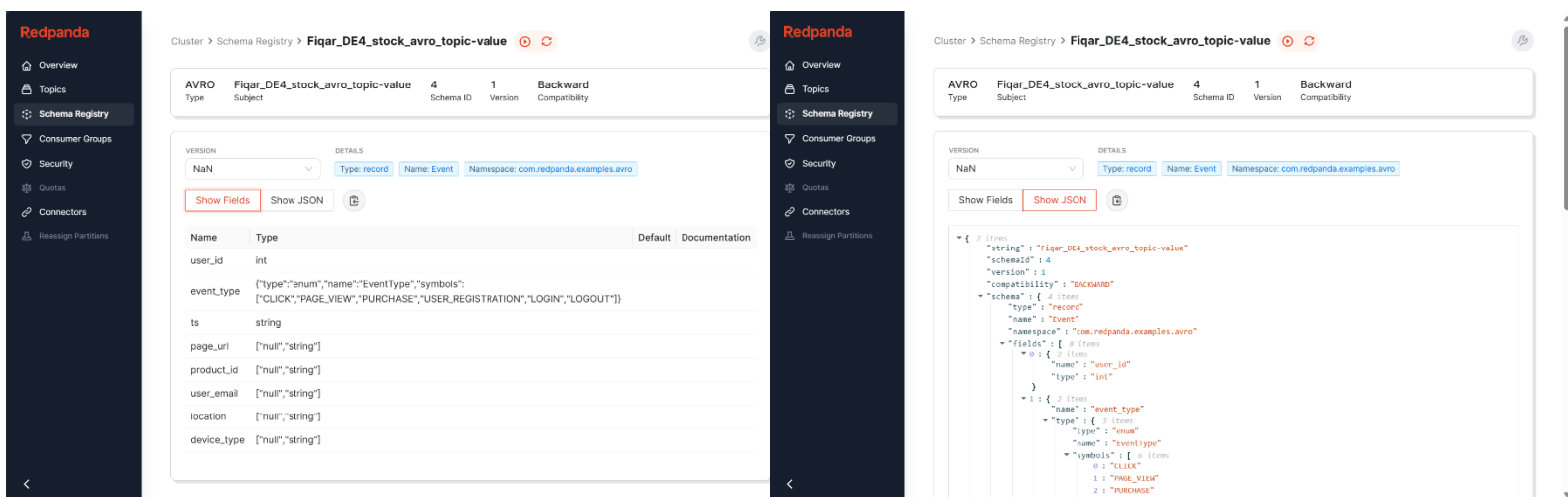
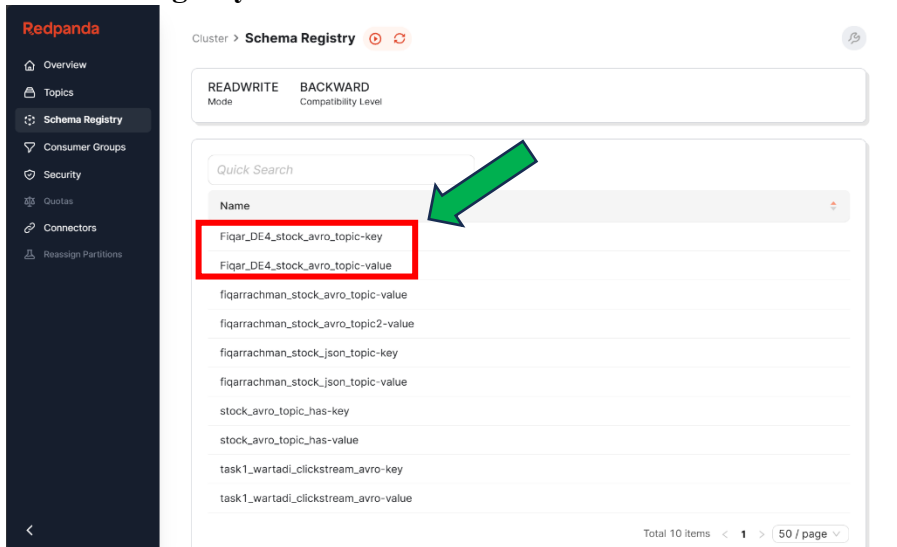
Berikut ini adalah tampilan list topic yang sudah dibuat, pada gambar diatas dapat terlihat bahwa topic dengan nama “Fiqr\_DE4\_stock\_topic” berhasil dibuat.

The screenshot shows the Redpanda web interface for the topic 'Fiqr\_DE4\_stock\_avro\_topic'. The top bar shows the topic name and some statistics: 176 kiB Size, 2,000 Messages, delete cleanup.policy, 7 days retention.ms, and Infinite retention.bytes. Below this are tabs for Messages, Consumers, Partitions, Configuration, ACL, and Documentation. The 'Messages' tab is selected, showing a list of messages with columns: Offset, Partition, Timestamp, Key, and Value. The messages are sorted by offset in descending order (Newest-50).

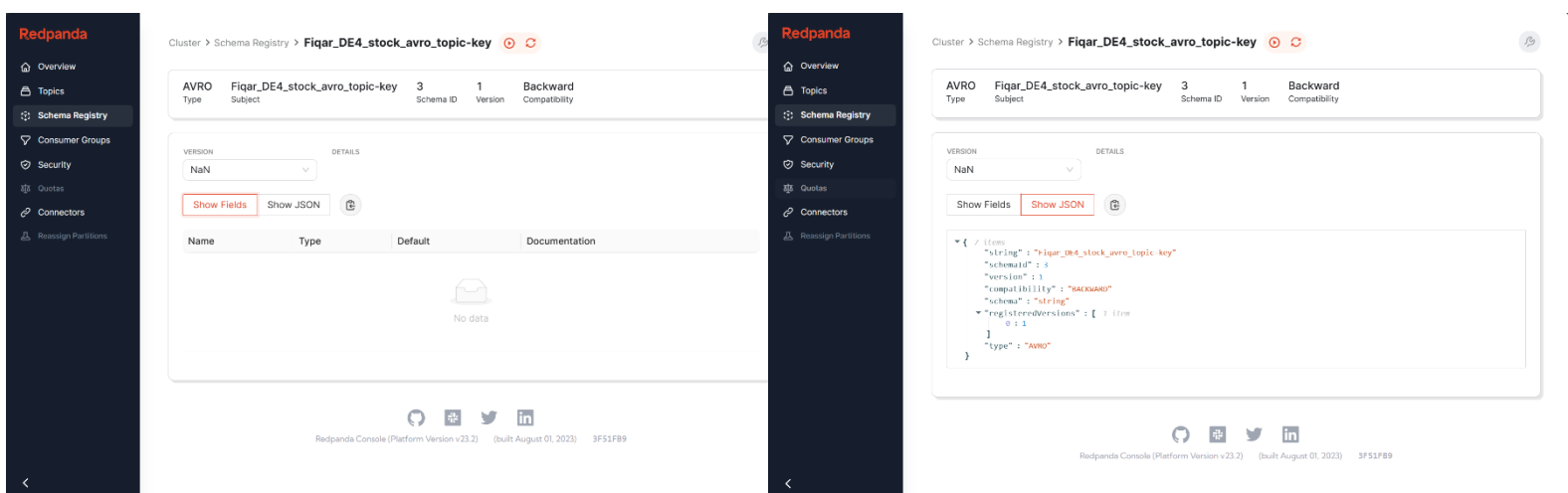
Offset	Partition	Timestamp	Key	Value
+ 1999	0	8/11/2024, 01:11:49	b4301983-86cf-453a-b49e-9de376603828	{"device_type":...
+ 1998	0	8/11/2024, 01:11:49	2a861119-e44a-41b3-97a2-c6a5ac862bf7	{"device_type":...
+ 1997	0	8/11/2024, 01:11:49	e8d4ecc9-a361-4b90-9925-403878a4811b	{"device_type":...
+ 1996	0	8/11/2024, 01:11:49	97e18035-0698-441e-8744-80abff0502c	{"device_type":...
+ 1995	0	8/11/2024, 01:11:49	77e44bdd-1849-4ced-ac5b-fe684a190be0	{"device_type":...
+ 1994	0	8/11/2024, 01:11:49	68943b56-53cd-4aae-a724-0af8ae1a388b	{"device_type":...
+ 1993	0	8/11/2024, 01:11:49	c673d45c-4f34-4866-bdf1-99e3f9442fcc	{"device_type":...
+ 1992	0	8/11/2024, 01:11:49	b4a2cb3d-cf1b-4cd5-89be-5c81ecfad825	{"device_type":...

Berikut ini adalah sebagian tampilan dari isi topic dengan nama “Fiqr\_DE4\_stock\_topic”.

## “Schema Registry”



Berikut ini adalah tampilan fields dan json dari schema registry yang bernama “Fiqar\_DE4\_stock\_avro\_topic\_value”.



Berikut ini adalah tampilan fields dan json dari schema registry yang bernama “Fiqar\_DE4\_stock\_avro\_topic\_key”.

## “Consumer Groups”

Cluster > Consumer Groups

2 Total Groups | 2 Stable

Quick Search

State	ID	Coordinator	Protocol	Members	Lag (Sum)
Stable	Protocol: connect 1	0	sessioned	1	0
Stable	<b>Fiqar_DE4_consumer_group</b>	0	range	1	0

Total 2 items < 1 > 50 / page

Redpanda Console (Platform Version v23.2) (built August 01, 2023) 3F51FB9

Berikut ini adalah tampilan list consumer groups yang sudah dibuat, pada gambar diatas dapat terlihat bahwa consumer groups dengan nama “Fiqar\_DE4\_consumer\_group” berhasil dibuat.

Cluster > Consumer Groups > **Fiqar\_DE4\_consumer\_group**

Stable State | 1 Assigned Partitions | consumer Protocol Type | 0 Coordinator ID | 0 Total Lag

Partitions

VIEW: Members | Topics | Show All | With Lag | Edit Group | Delete Group

Fiqar\_DE4\_stock\_avro\_topic lag: 0 assigned partitions: 1 View Topic

Partition	Assigned Member	Host	Log End Offset	Group Offset	Lag
0	rdkafka-0740def17-48555-4c0b-87f2-0a77544efe7a	103.172.71.17	2 000	2 000	0

Redpanda Console (Platform Version v23.2) (built August 01, 2023) 3F51FB9

Berikut ini adalah tampilan dari isi consumer groups dengan nama “Fiqar\_DE4\_consumer\_group”.

Dari hasil – hasil yang sudah dipaparkan diatas melalui gambar, dapat disimpulkan bahwa proses pembuatan **Topics**, **Schema Registry**, maupun **Consumer Groups** berhasil dan berjalan sesuai yang diharapkan.