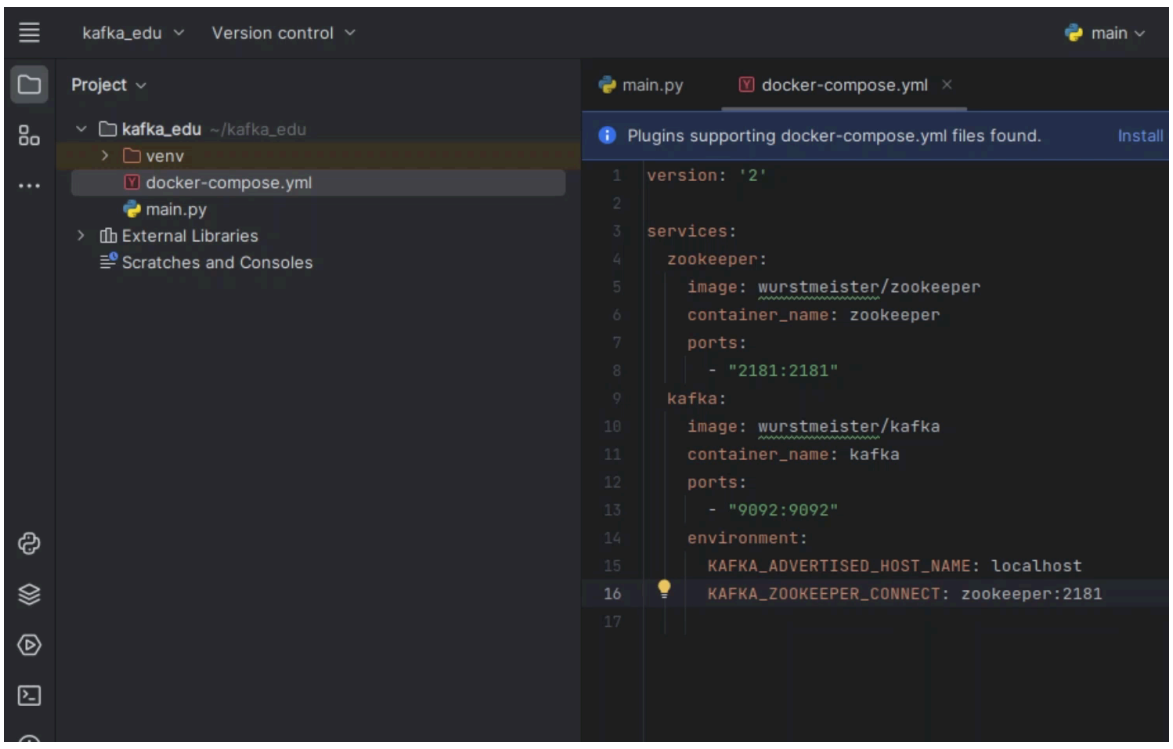
ОТПРАВКА СООБЩЕНИЙ В KAFKA ЧЕРЕЗ PYTHON

План:

- Работа в терминале.

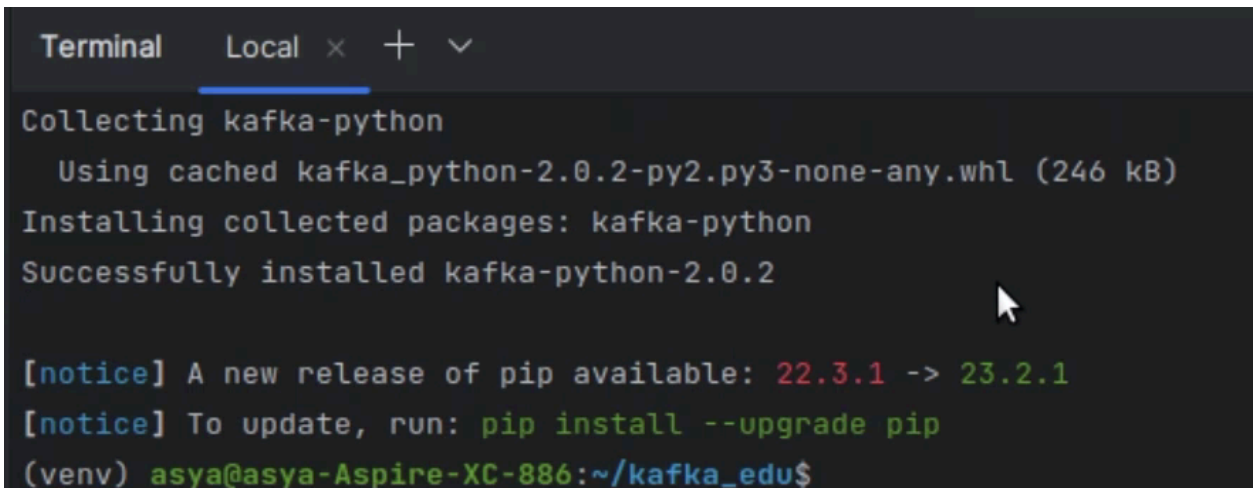
Работа в терминале

Мы будем работать в Kafka через Python. Для этого открываем PyCharm в той же директории, что и Kafka:



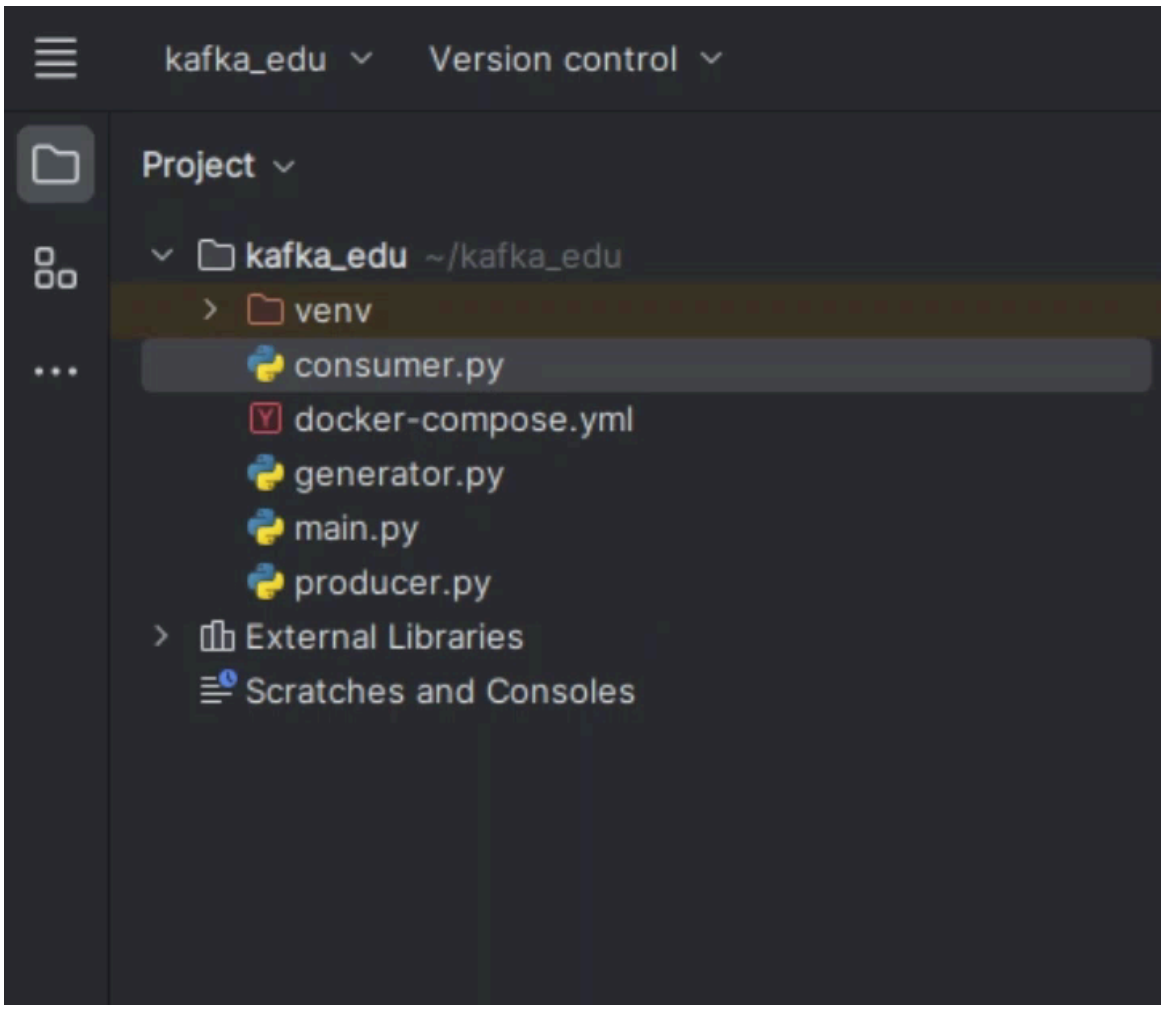
Открываем терминал в PyCharm, чтобы была активирована виртуальная среда, устанавливаем через «pip install kafka-python»:

Таким образом, в виртуальной среде будут инструменты для работы с Kafka через Python:

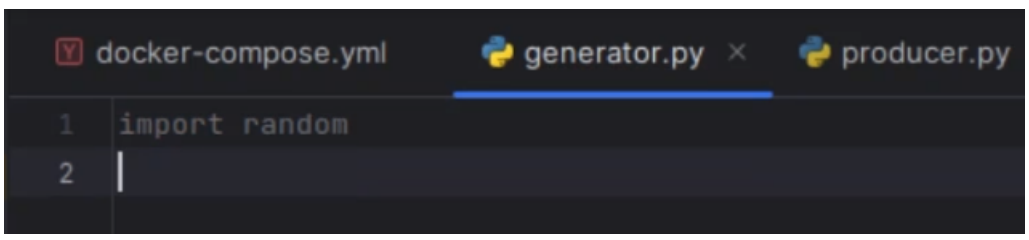


Если вы работаете без ID, вам нужно активировать виртуальную среду и установить туда Kafka-python package.

Мы создаем три файла Python, генератор. У нас будет генерироваться какая-то информация, поступать в продюсер и отправляться в Kafka, из которого будет читаться с помощью консьюмера. То есть, мы напишем генератор, продюсер и консьюмер:



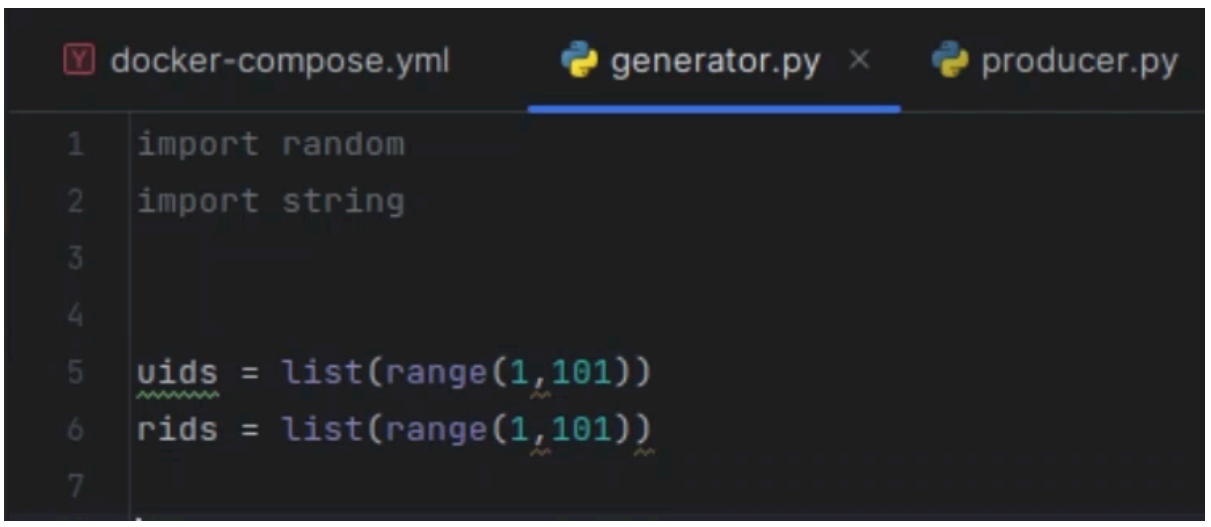
Добавляем код и импортим «Библиотека Random», которая будет помогать генерировать данные:



«Import String», чтобы выводить логи:

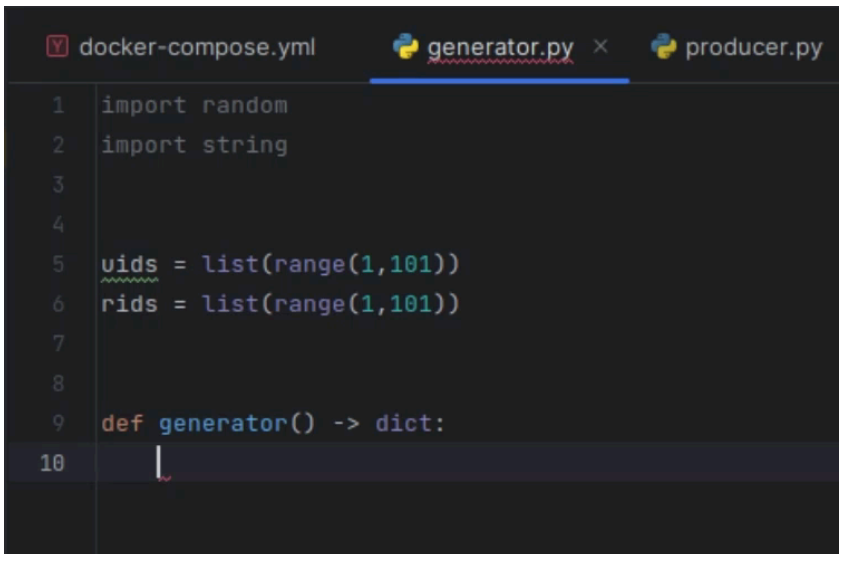
```
2  import string
```

Прописываем два листа со значениями наших получателей и отправителей:



Где «Uids» – отправители.

Пишем процедуру:



Вначале будет выбираться рандомный отправитель:

```
9  def generator() -> dict:
10     rand_uid = random.choice(uids)
11
```

Выбираем рандомного получателя. Вначале сделаем копию всех получателей, поскольку оттуда нужно будет выбросить отправителя (он не может отправлять сообщения самому себе):

```
12     rids_copy = rids.copy()
13     rids_copy.remove(rand_uid)
14     rand_rid = random.choice(rids_copy)
```

Отправляем сообщение, которое есть сгенерированная строка из 32 символов (причем рандомных символов):

```
15
16     message = ''.join(random.choice(string.ascii_letters) for i in range(32))
```

Делаем «Return» словарика:

```
18     return {'user_id': rand_uid, 'recipient_id': rand_rid, 'message': message}
```

Переходим к коду продюсера. Для написания нам нужен «Import time»:

docker-compose.yml | generator.py | producer.py

```
1  import time
2
```

Также нужен «Import json», поскольку генерируются данные Python. Kafka не может читать файлы Python, поэтому их нужно сериализировать с помощью библиотеки Json. Также используем «Random» для таймаутов между отправкой сообщений. Пишем «From Time Import Datetime», а также «From Kafka Import» (где объект – «Kafka Producer»):

docker-compose.yml | generator.py | producer.py

```
1  import time
2  import json
3  import random
4  from datetime import datetime
5  from kafka import KafkaProducer
```

Из написанного нами модуля «Generator» импортим функцию «Generator»:

```
9  def generator() -> dict:
10     rand_uid = random.choice(uids)
11
```

Переходим к написанию функции для сериализации:

```
9  def serializer(message):
10     return json.dumps(message).encode('utf-8')
11
```

Создаем Producer Kafka:

```
producer = KafkaProducer(
    bootstrap_server=['localhost:9092'],
    value_serializer=serializer
)
```

Напишем функцию, которая будет запускать продюсер в бесконечном цикле. Пишем следующее:

```
if __name__=='__main__':
    while True:
        some_msg = generator()

        print(f'Message produced @ {datetime.now()} | Message = {some_msg}')
```

С помощью продюсера отправляем сообщение в нашу очередь/топик:

```
        print(f'Message produced @ {datetime.now()} | Message = {some_msg}')
        producer.send('kafka_edu', some_msg)
```

Сделаем между отправками сообщений рандомные интервалы сна:

```
            time_to_sleep = random.randint(1,10)
            time.sleep(time_to_sleep)
```

Продюсер готов.

Перейдем к консьюмеру. Здесь тоже используем «Json», чтобы данные были читаемыми, с помощью Python:

docker-compose.yml generator.py producer.py

```
import json
from kafka import KafkaConsumer

```

Здесь будет работать следующий скрипт:

```
if __name__=='__main__':
```

Создаем консьюмер, который будет передаваться:

docker-compose.yml generator.py producer.py

```
import json
from kafka import KafkaConsumer


if __name__=='__main__':
    consumer = KafkaConsumer(
        'kafka_edu',
        bootstrap_servers='localhost:9092',
        auto_offset_reset='earliest'
    )
    for message in consumer:
        print(json.loads(message.value))
```

if __name__=='__main__' for message in consumer

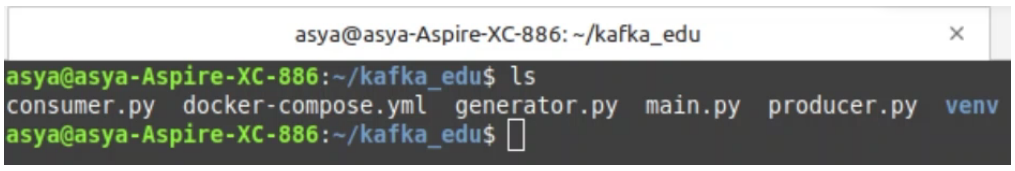
Обратите внимание на следующую строчку:

Это указание на то, какие данные нужно считывать первыми.

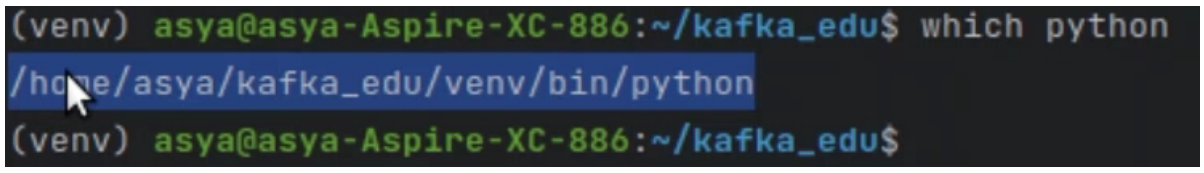
Мы написали код всех трех модулей:

- Генератор;
- Продюсер;
- Консьюмер.

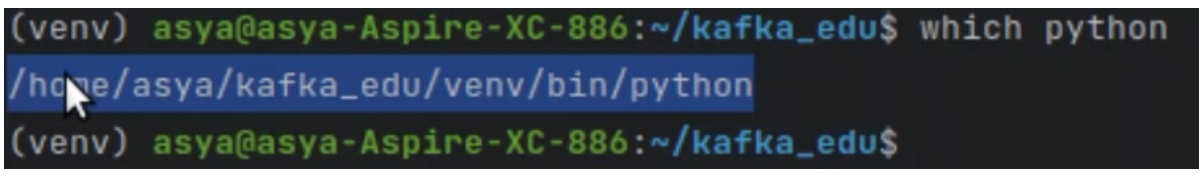
Посмотрим, как они работают. Для этого возвращаемся в директорию, где лежал код. Нам нужно запустить продюсер в виртуальной среде. Чтобы это сделать, нужно зайти в PyCharm (если пользуетесь PyCharm; если нет, то вы и так находитесь в виртуальной среде):



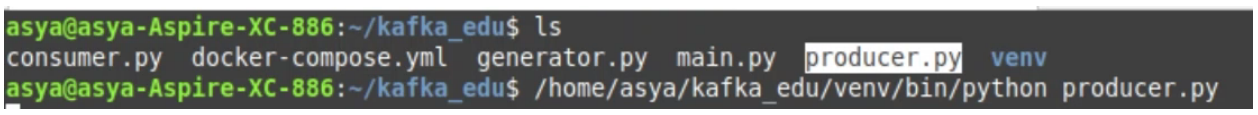
В PyCharm (в терминале) пишем следующее:



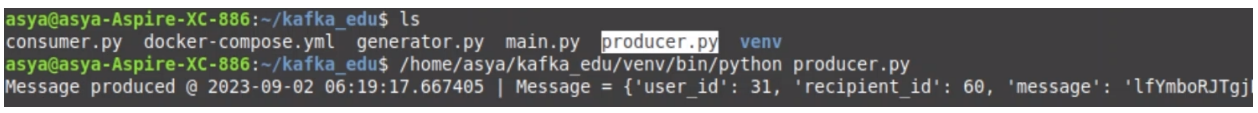
Копируем данную строчку нашему интерпретатору, находящемуся внутри виртуальной среды:



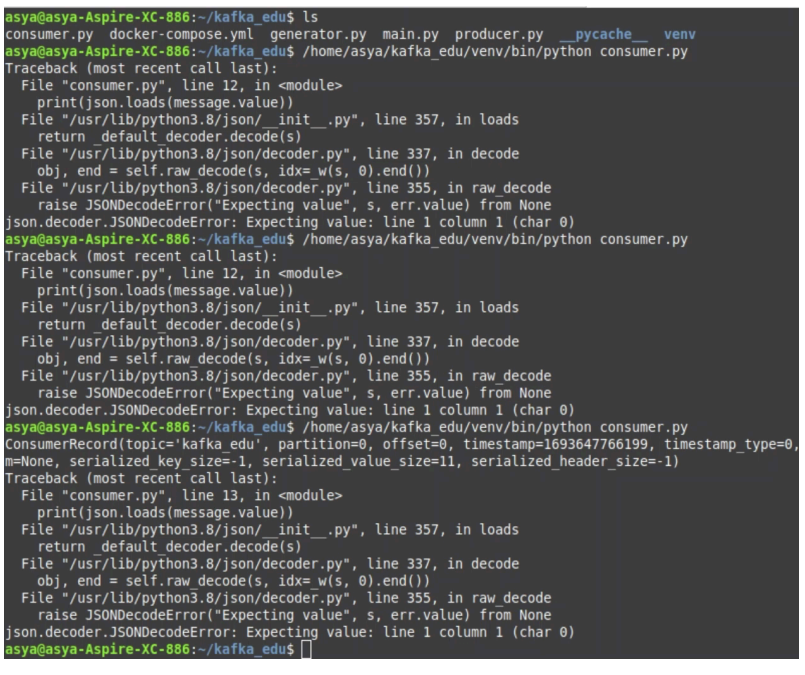
Дальнейший запуск необходимо проводить с помощью этого Python:



Запускаем продюсер. Видим, что он начинает генерировать сообщения:



Запустим консьюмер в соседнем окне. Для этого таким же образом выбираем интерпретатор, который используется внутри виртуальной среды. Используем на этот раз «consumer.py»:



Видим ошибку. Причина – лежащее вначале очереди сообщение «Hello, World!», которое не было зашифровано json’ом. Поэтому декодер на него ругается.

Создадим новую очередь. Для этого заходим в контейнер с Kafka. Все скрипты в контейнере лежат в /bin и запускаются без .sh. Мы создаем новый топик:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# cd opt
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt#
```

```
asya@asya-Aspire-XC-886:~/kafka_edu$ docker exec -it kafka /bin/bash
root@f61a38164238:/# cd opt
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1/bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh          kafka-producer-perf-test.sh         kafka-verifiable-producer.sh
connect-mirror-maker.sh       kafka-delegation-tokens.sh           kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh              kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                    kafka-run-class.sh                  zookeeper-security-migration.sh
kafka-broker-api-versions.sh  kafka-features.sh                    kafka-server-start.sh               zookeeper-server-start.sh
kafka-cluster.sh              kafka-leader-election.sh             kafka-server-stop.sh                zookeeper-server-stop.sh
kafka-configs.sh              kafka-log-dirs.sh                    kafka-storage.sh                    zookeeper-shell.sh
kafka-console-consumer.sh     kafka-metadata-shell.sh              kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh                kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh  kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
```

Если данные хранятся в разном формате, то это важно учитывать.

Через «kafka-topics» зададим новый топик (назовем «kafka_edu_2»), в котором не было никаких данных. Также делаем topics list (в командах вместо - -zookeeper zookeeper:2181 ввести `--bootstrap-server localhost:9092` и скрипты `в bin, запускаются без .sh):`

```
root@f61a38164238:/# cd opt
root@f61a38164238:/opt# ls
kafka  kafka_2.13-2.8.1  overrides
root@f61a38164238:/opt# cd kafka_2.13-2.8.1/bin
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# ls
connect-distributed.sh        kafka-consumer-perf-test.sh          kafka-producer-perf-test.sh         kafka-verifiable-producer.sh
connect-mirror-maker.sh       kafka-delegation-tokens.sh           kafka-reassign-partitions.sh        trogdor.sh
connect-standalone.sh         kafka-delete-records.sh              kafka-replica-verification.sh       windows
kafka-acls.sh                 kafka-dump-log.sh                    kafka-run-class.sh                  zookeeper-security-migration.sh
kafka-broker-api-versions.sh  kafka-features.sh                    kafka-server-start.sh               zookeeper-server-start.sh
kafka-cluster.sh              kafka-leader-election.sh             kafka-server-stop.sh                zookeeper-server-stop.sh
kafka-configs.sh              kafka-log-dirs.sh                    kafka-storage.sh                    zookeeper-shell.sh
kafka-console-consumer.sh     kafka-metadata-shell.sh              kafka-streams-application-reset.sh
kafka-console-producer.sh     kafka-mirror-maker.sh                kafka-topics.sh
kafka-consumer-groups.sh      kafka-preferred-replica-election.sh  kafka-verifiable-consumer.sh
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic kafka_ed
u_2
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, bu
t not both.
Created topic kafka_edu_2.
root@f61a38164238:/opt/kafka_2.13-2.8.1/bin# kafka-topics.sh --list --zookeeper zookeeper:2181
```

Видим следующее:

```
kafka_edu_2
```

Это топик, в котором нет никаких данных.

Возвращаемся к продюсеру, делаем паузу:

```
asya@asya-Aspire-XC-886: ~/kafka_edu                                    asya@asya-Asp
Message produced @ 2023-09-02 06:22:48.867695 | Message = {'user_id': 46, 'recipient_id': 1, 'message': '
Message produced @ 2023-09-02 06:22:55.871580 | Message = {'user_id': 57, 'recipient_id': 62, 'message':
Message produced @ 2023-09-02 06:22:58.873291 | Message = {'user_id': 99, 'recipient_id': 17, 'message':
Message produced @ 2023-09-02 06:23:07.881582 | Message = {'user_id': 32, 'recipient_id': 63, 'message':
Message produced @ 2023-09-02 06:23:17.891654 | Message = {'user_id': 24, 'recipient_id': 68, 'message':
Message produced @ 2023-09-02 06:23:26.901113 | Message = {'user_id': 46, 'recipient_id': 78, 'message':
Message produced @ 2023-09-02 06:23:36.911573 | Message = {'user_id': 58, 'recipient_id': 84, 'message':
Message produced @ 2023-09-02 06:23:42.915597 | Message = {'user_id': 94, 'recipient_id': 9, 'message': '
Message produced @ 2023-09-02 06:23:44.918234 | Message = {'user_id': 42, 'recipient_id': 73, 'message':
Message produced @ 2023-09-02 06:23:48.921178 | Message = {'user_id': 68, 'recipient_id': 8, 'message': '
Message produced @ 2023-09-02 06:23:54.927644 | Message = {'user_id': 82, 'recipient_id': 42, 'message':
Message produced @ 2023-09-02 06:23:56.929841 | Message = {'user_id': 60, 'recipient_id': 18, 'message':
Message produced @ 2023-09-02 06:23:59.931566 | Message = {'user_id': 44, 'recipient_id': 84, 'message':
Message produced @ 2023-09-02 06:24:06.936437 | Message = {'user_id': 54, 'recipient_id': 2, 'message': '
Message produced @ 2023-09-02 06:24:13.944107 | Message = {'user_id': 33, 'recipient_id': 100, 'message':
Message produced @ 2023-09-02 06:24:19.949247 | Message = {'user_id': 68, 'recipient_id': 64, 'message':
Message produced @ 2023-09-02 06:24:29.959674 | Message = {'user_id': 98, 'recipient_id': 20, 'message':
Message produced @ 2023-09-02 06:24:30.961228 | Message = {'user_id': 97, 'recipient_id': 11, 'message':
Message produced @ 2023-09-02 06:24:40.967737 | Message = {'user_id': 87, 'recipient_id': 50, 'message':
Message produced @ 2023-09-02 06:24:43.971266 | Message = {'user_id': 48, 'recipient_id': 93, 'message':
Message produced @ 2023-09-02 06:24:48.975653 | Message = {'user_id': 52, 'recipient_id': 62, 'message':
Message produced @ 2023-09-02 06:24:55.982310 | Message = {'user_id': 62, 'recipient_id': 30, 'message':
Message produced @ 2023-09-02 06:24:59.983593 | Message = {'user_id': 43, 'recipient_id': 88, 'message':
Message produced @ 2023-09-02 06:25:08.991726 | Message = {'user_id': 70, 'recipient_id': 75, 'message':
Message produced @ 2023-09-02 06:25:10.994543 | Message = {'user_id': 26, 'recipient_id': 32, 'message':
Message produced @ 2023-09-02 06:25:18.001080 | Message = {'user_id': 32, 'recipient_id': 59, 'message':
Message produced @ 2023-09-02 06:25:27.007611 | Message = {'user_id': 90, 'recipient_id': 2, 'message': '
Message produced @ 2023-09-02 06:25:35.015655 | Message = {'user_id': 8, 'recipient_id': 91, 'message': '
Message produced @ 2023-09-02 06:25:44.024438 | Message = {'user_id': 15, 'recipient_id': 60, 'message':
Message produced @ 2023-09-02 06:25:52.031523 | Message = {'user_id': 69, 'recipient_id': 35, 'message':
Message produced @ 2023-09-02 06:25:58.036414 | Message = {'user_id': 26, 'recipient_id': 71, 'message':
Message produced @ 2023-09-02 06:26:01.040353 | Message = {'user_id': 28, 'recipient_id': 100, 'message':
^CTraceback (most recent call last):
  File "producer.py", line 27, in <module>
    time.sleep(time_to_sleep)
KeyboardInterrupt

asya@asya-Aspire-XC-886:~/kafka_edu$
```

Возвращаемся в PyCharm. В нем, в продюсере, пропишем новое название топика (в продюсере и консьюмере):

```
docker-compose.yml   generator.py   producer.py
17
18
19  if __name__=='__main__':
20      while True:
21          some_msg = generator()
22
23          print(f'Message produced @ {datetime.now()}
24          producer.send('kafka_edu_2', some_msg)
25
26          time_to_sleep = random.randint(1,10)
27          time.sleep(time_to_sleep)
28
```

```python
import json
from kafka import KafkaConsumer


if __name__=='__main__':
    consumer = KafkaConsumer(
        'kafka_edu_2',
        bootstrap_servers='localhost:9092',
        auto_offset_reset='earliest'
    )
    for message in consumer:
        print(message)
        print(json.loads(message.value))
```

Запустим:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ /home/asya/kafka_edu/venv/bin/python consumer.py
ConsumerRecord(topic='kafka_edu_2', partition=0, offset=0, timestamp=1693650396698, timestamp_type=0, key=None, value=b'{"user_id": 41, "recipient_id":
 48, "message": "cmtAVxCUolcdZhLOWoovfcrxuStUhErh"}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=82, serialized_header_si
ze=-1)
{'user_id': 41, 'recipient_id': 48, 'message': 'cmtAVxCUolcdZhLOWoovfcrxuStUhErh'}
ConsumerRecord(topic='kafka_edu_2', partition=0, offset=1, timestamp=1693650404704, timestamp_type=0, key=None, value=b'{"user_id": 6, "recipient_id":
2, "message": "MGusgxKlxObaVxyjAzNRvILEmNvHAusU"}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=80, serialized_header_size
=-1)
{'user_id': 6, 'recipient_id': 2, 'message': 'MGusgxKlxObaVxyjAzNRvILEmNvHAusU'}
ConsumerRecord(topic='kafka_edu_2', partition=0, offset=2, timestamp=1693650406706, timestamp_type=0, key=None, value=b'{"user_id": 77, "recipient_id":
 86, "message": "cFKNmTiVdARJSCXvLsHlmTMafYbqtTDO"}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=82, serialized_header_si
ze=-1)
{'user_id': 77, 'recipient_id': 86, 'message': 'cFKNmTiVdARJSCXvLsHlmTMafYbqtTDO'}
ConsumerRecord(topic='kafka_edu_2', partition=0, offset=3, timestamp=1693650416716, timestamp_type=0, key=None, value=b'{"user_id": 21, "recipient_id":
 38, "message": "PlsqkxlZKWRDVYGgYJbVnDNTwPgIDSGo"}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=82, serialized_header_si
ze=-1)
{'user_id': 21, 'recipient_id': 38, 'message': 'PlsqkxlZKWRDVYGgYJbVnDNTwPgIDSGo'}
ConsumerRecord(topic='kafka_edu_2', partition=0, offset=4, timestamp=1693650417718, timestamp_type=0, key=None, value=b'{"user_id": 50, "recipient_id":
 54, "message": "rKdtFtFwiyPxipFbGLVPpLOnWYvmSIov"}', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=82, serialized_header_si
ze=-1)
{'user_id': 50, 'recipient_id': 54, 'message': 'rKdtFtFwiyPxipFbGLVPpLOnWYvmSIov'}
```

Видим, что сообщения считываются.

Остановим консьюмер, уберем следующее:

```python
print(message)
```

```python
if __name__=='__main__':
    consumer = KafkaConsumer(
        'kafka_edu_2',
        bootstrap_servers='localhost:9092',
        auto_offset_reset='earliest'
    )
    for message in consumer:
        print(json.loads(message.value))
```

Запустим консьюмер, чтобы выводились уже расшифрованные сообщения. Видим следующее:

```
asya@asya-Aspire-XC-886:~/kafka_edu$ /home/asya/kafka_edu/venv/bin/python consumer.py
{'user_id': 41, 'recipient_id': 48, 'message': 'cmtAVxCUolcdZhLOWoovfcrxuStUhErh'}
{'user_id': 6, 'recipient_id': 2, 'message': 'MGusgxKlxObaVxyjAzNRvILEmNvHAusU'}
{'user_id': 77, 'recipient_id': 86, 'message': 'cFKNmTiVdARJSCXvLsHlmTMafYbqtTDO'}
{'user_id': 21, 'recipient_id': 38, 'message': 'PlsqkxlZKWRDVYGgYJbVnDNTwPgIDSGo'}
{'user_id': 50, 'recipient_id': 54, 'message': 'rKdtFtFwiyPxipFbGLVPpLOnWYvmSIov'}
{'user_id': 79, 'recipient_id': 81, 'message': 'METOSWZPOImTyXEezJPJRyHCIAqpQmgZ'}
{'user_id': 29, 'recipient_id': 6, 'message': 'yisQcRyHTgfGcapGwIobIGPVSDNFmElv'}
{'user_id': 2, 'recipient_id': 96, 'message': 'QHMINAluQqmNPgsqJSiFXDuelWkHRadr'}
{'user_id': 31, 'recipient_id': 25, 'message': 'CyKRcbLsFnRTbAiCOjeuwUzQcwXhVycJ'}
{'user_id': 10, 'recipient_id': 41, 'message': 'EZRqOcEvlDAwOJRIenQcBWYwQTkcJWyX'}
{'user_id': 99, 'recipient_id': 89, 'message': 'qSMkWUTjvwrGuLqFtkdJTtiDyFqUdzPE'}
{'user_id': 19, 'recipient_id': 48, 'message': 'WGVrIXTOZCSbfnZEyPowYubzHIcAUVwV'}
{'user_id': 85, 'recipient_id': 83, 'message': 'sTtOxDHoBGZwFoTUsVpsCoBfsGPgxVEz'}
```

Мы научились отправлять и получать сообщения в Kafka с помощью Python.