A very short and super friendly Apache Kafka Tutorial

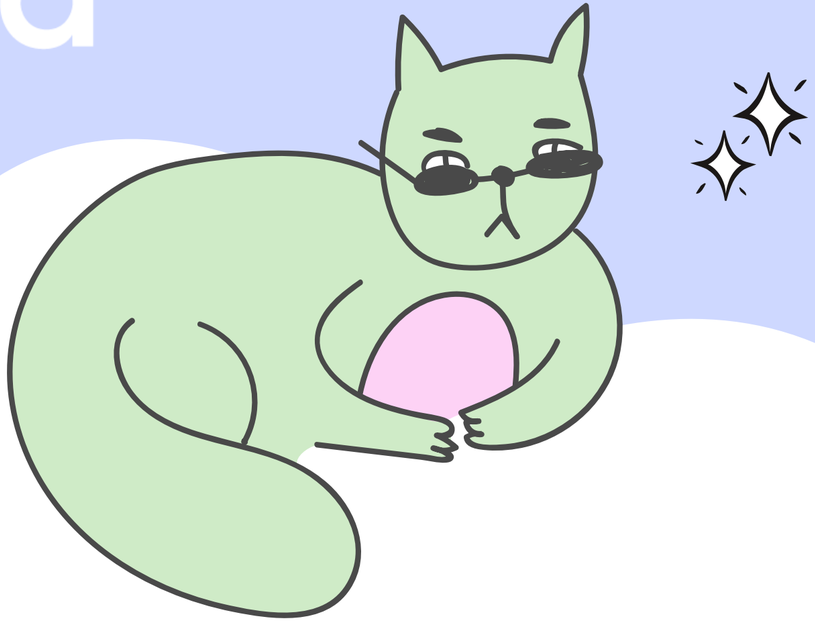
A. Karolewski, M. Kolano, M. Walczykowski

TSD 2023

What is Apache Kafka?

kafka

It's an open-source message broker with awesome high-performance heart.



That works in producer-consumer mode.

And can be deployed on bare hardware, docker containers or clouds.

What are main APIs of it?

- Admin API to manage and inspect topics, brokers, and other Kafka objects
- Producer API to publish (write) a stream of events to one or more Kafka topics
- Consumer API subscribe to (read) one or more topics and to process the stream of events produced to them
- Stream API to implement stream processing applications and microservices
- Connect API o build and run reusable data import/export connectors that consume (read) or produce (write) streams of events from and to external systems and applications so they can integrate with Kafka

Zookeeper

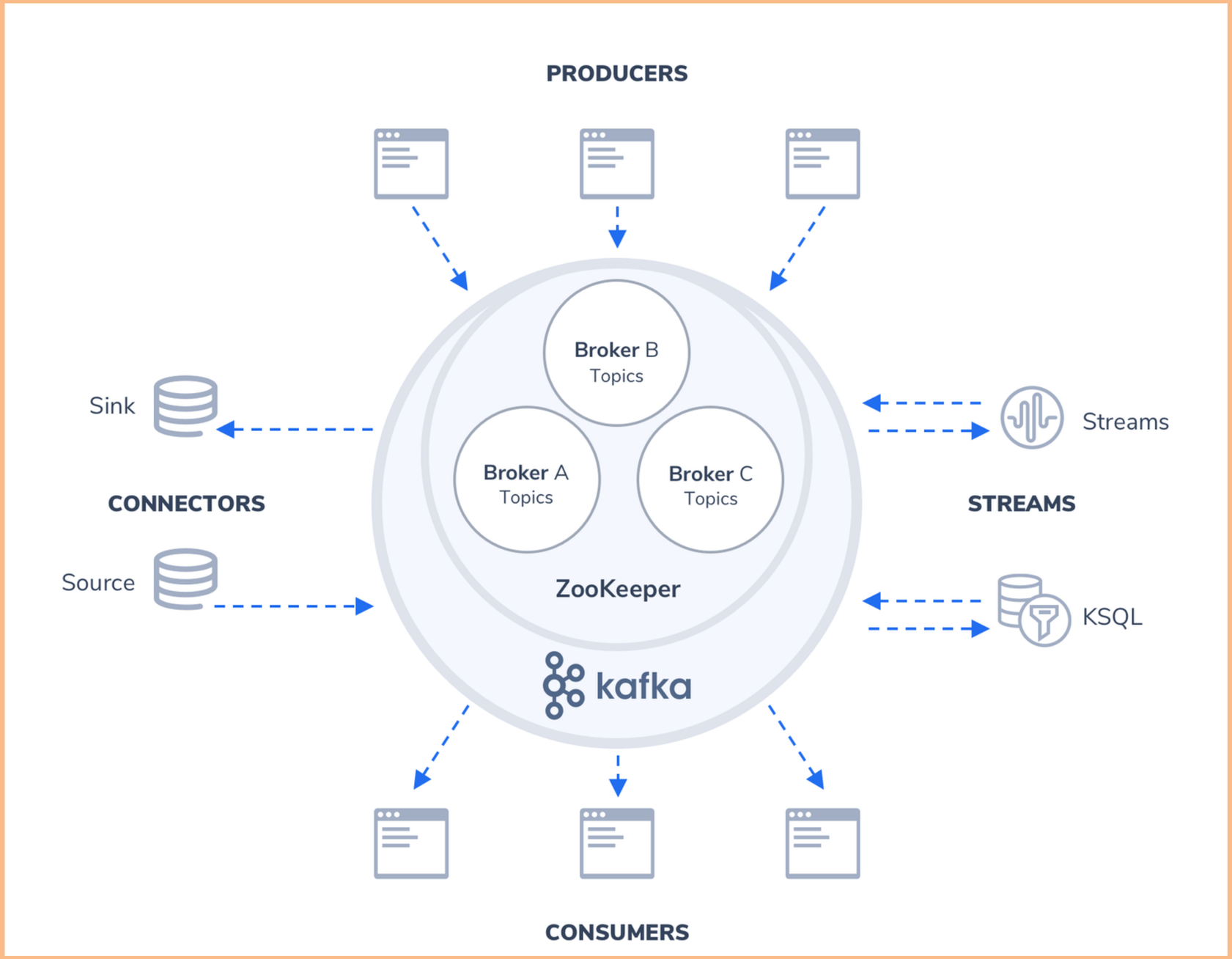
is the heart of Kafka cluster.

Broker

is the main part of message transferring.

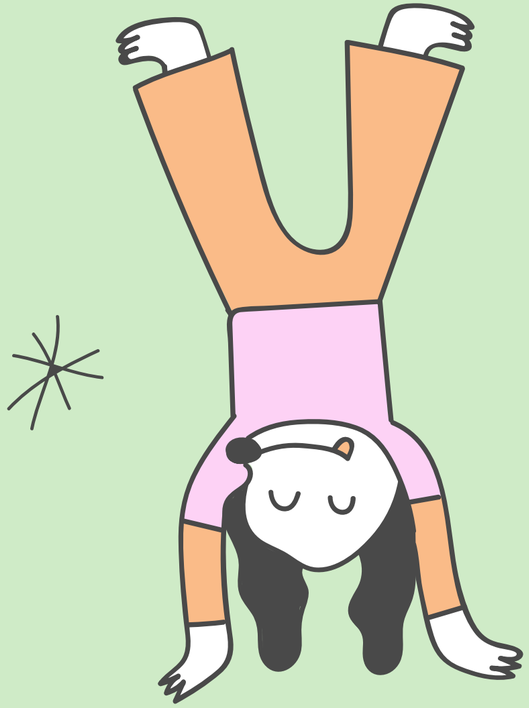
Topic

is a single node to transfer message event.

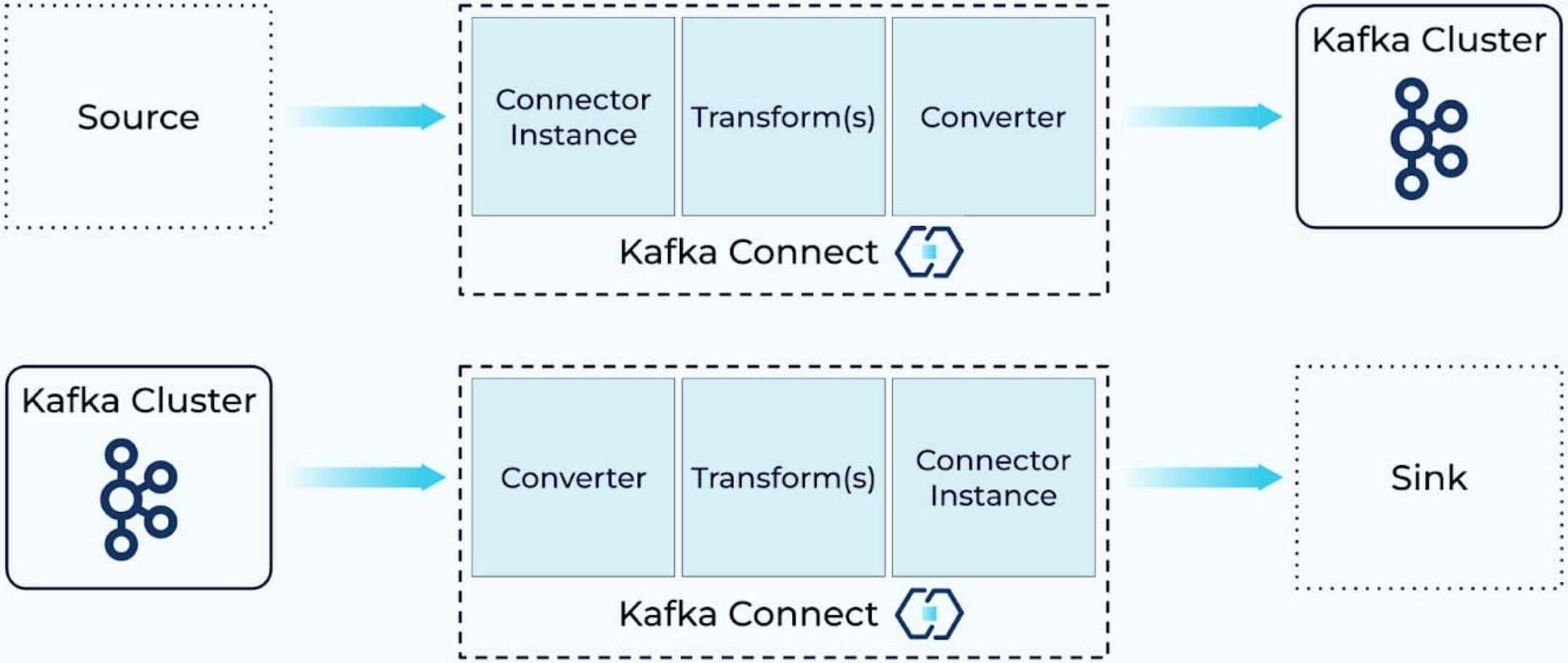

PRODUCERS
Broker B
Topics
Broker A
Topics
Broker C
Topics
ZooKeeper
kafka
Sink
CONNECTORS
Source
Streams
STREAMS
KSQL
CONSUMERS

Connectors

so useful...



Inside Kafka Connect



Let's code

Some code snippets

```yaml
---
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.3.0
    hostname: zookeeper
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-kafka:7.3.0
    container_name: broker
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://broker:29092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
```

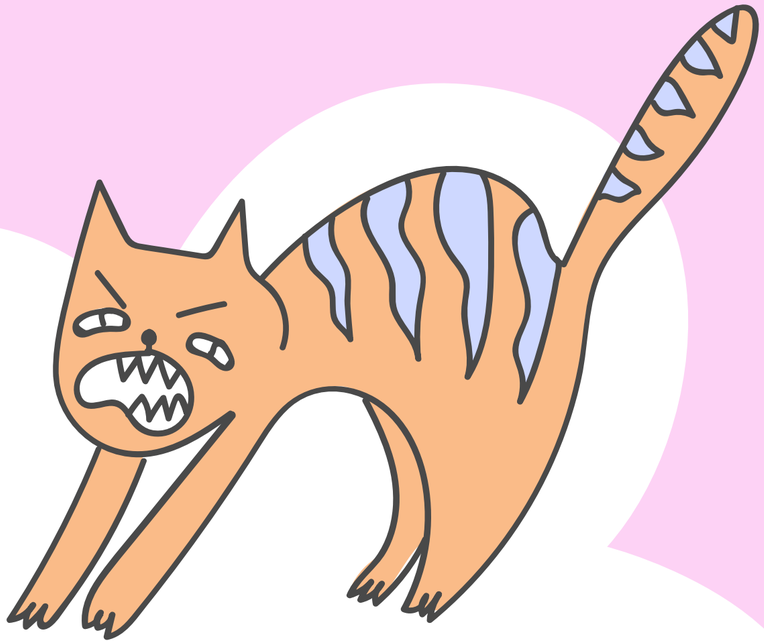
```bash
docker compose exec broker \
  kafka-topics --create \
    --topic purchases \
    --bootstrap-server localhost:9092 \
    --replication-factor 1 \
    --partitions 1
```

```python
#!/usr/bin/env python

import sys
from random import choice
from argparse import ArgumentParser, FileType
from configparser import ConfigParser
from confluent_kafka import Producer

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('config_file', type=FileType('r'))
    args = parser.parse_args()

    config_parser = ConfigParser()
    config_parser.read_file(args.config_file)
    config = dict(config_parser['default'])

    producer = Producer(config)

    def delivery_callback(err, msg):
        if err:
            print('ERROR: Message failed delivery: {}'.format(err))
        else:
            print("Produced event to topic {topic}: key = {key:12} value = {value:12}".format(
                topic=msg.topic(), key=msg.key().decode('utf-8'), value=msg.value().decode('utf-8')))

    # Produce data by selecting random values from these lists.
    topic = "purchases"
    user_ids = ['eabara', 'jsmith', 'sgarcia', 'jbernard', 'htanaka', 'awalther']
    products = ['book', 'alarm clock', 't-shirts', 'gift card', 'batteries']

    count = 0
    for _ in range(10):

        user_id = choice(user_ids)
        product = choice(products)
        producer.produce(topic, product, user_id, callback=delivery_callback)
        count += 1

    # Block until the messages are sent.
    producer.poll(10000)
    producer.flush()
```

```python
#!/usr/bin/env python

import sys
from argparse import ArgumentParser, FileType
from configparser import ConfigParser
from confluent_kafka import Consumer, OFFSET_BEGINNING

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('config_file', type=FileType('r'))
    parser.add_argument('--reset', action='store_true')
    args = parser.parse_args()

    config_parser = ConfigParser()
    config_parser.read_file(args.config_file)
    config = dict(config_parser['default'])
    config.update(config_parser['consumer'])

    consumer = Consumer(config)

    def reset_offset(consumer, partitions):
        if args.reset:
            for p in partitions:
                p.offset = OFFSET_BEGINNING
            consumer.assign(partitions)

    # Subscribe to topic
    topic = "purchases"
    consumer.subscribe([topic], on_assign=reset_offset)

    # Poll for new messages from Kafka and print them.
    try:
        while True:
            msg = consumer.poll(1.0)
            if msg is None:
                # Initial message consumption may take up to
                # `session.timeout.ms` for the consumer group to
                # rebalance and start consuming
                print("Waiting...")
            elif msg.error():
                print("ERROR: %s".format(msg.error()))
            else:
                # Extract the (optional) key and value, and print.

                print("Consumed event from topic {topic}: key = {key:12} value = {value:12}".format(
                    topic=msg.topic(), key=msg.key().decode('utf-8'), value=msg.value().decode('utf-
8')))
    except KeyboardInterrupt:
        pass
    finally:
        # Leave group and commit final offsets
        consumer.close()
```

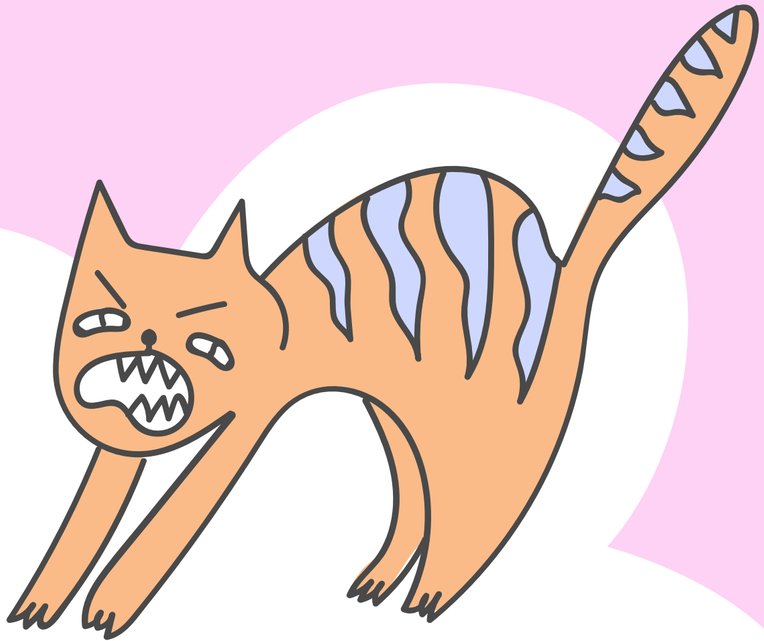
Task 1 (easy)

Set up provided Kafka Docker instance and try sample code. Your task is to change the topic and sent Data Structure. Let it be list of your classes and marks. The consumer must calculate your GPA.



Task 2 (medium)

Change your code so the data is no longer hard coded. Right now producer should be able to write it and send ad-hoc. It's something like telnet. Also add second consumer and producer. Everything should work in one group and topic - right know change the partitioning in topic to 5.



Task 3 (hard)

In this task all you need to start is one producer that creates infinite number of messages to one topic with partitioning and 5 consumers assigned to 3 groups (2, 2 and 1 in each). Message "finished" from producers forces consumers to stop receiving. Each group should communicate (via another topic, acting as a producer) and calculate a mean. The mean must be sent back to producer (another topic!). At the end producer should display all means.