МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Факультет №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 311 «Прикладные программные средства и математические методы»

**ЛАБОРАТОРНАЯ РАБОТА**

**по дисциплине:**

**«Информационная безопасноть»**

Выполнил студент группы М3О-417Бк-21

Корсаков Е.А.

Москва 2024 г.

**Развёртывание Kafka на Node.js**

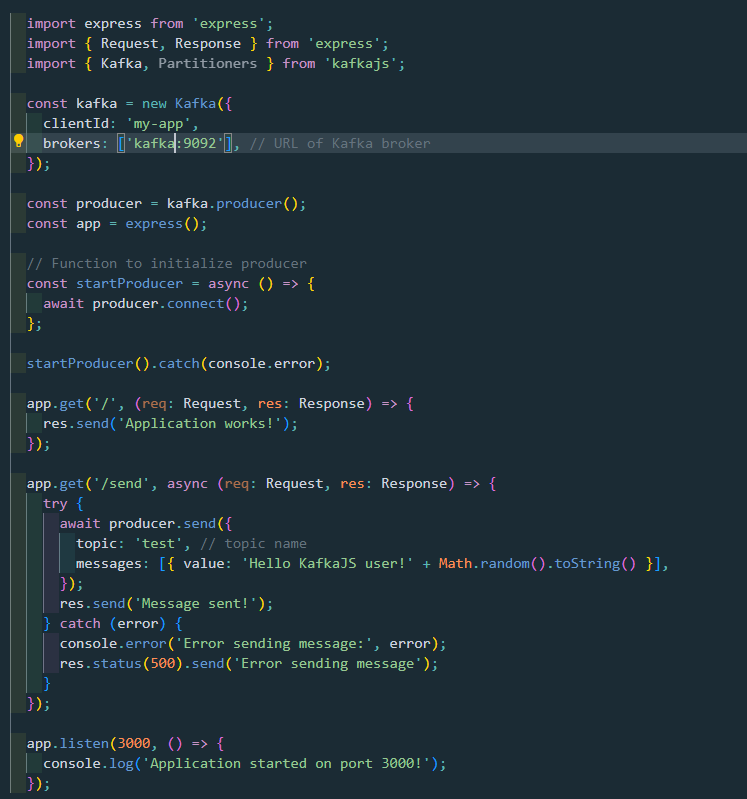
**Задача**: Сэмулировать работу сервера, отправляющего идентичные сообщения нескольким (двум) серверам – потребителя.

**Введение**: В реальности вышеупомянутый кейс встречается в больших высоконагруженных коммунальных приложениях: Например, в банке есть сервис платежей, к которому подключено большое число систем и возможен кейс, когда 2ум системам может понадобиться одна и та же информация о платеже клиента, чтобы обеспечить надёжность поставки сообщений банк использует брокер сообщений Kafka (в случае выхода из строя сервера поставщика информации о платежах (producer), данные останутся в Kafka, не пропадут). За счёт чего обеспечивается надёжность ? Один из базовых элементов брокера сообщения Kafka – топик. Топик это сущность, полностью содержащая всю информацию получаемую от producer, которую затем Kafka отдаст потребителям (consumers). Топик разделён на партиции. Это сделано для распределения нагрузки. Например, есть топик test, содержащий 3 партиции. При поступлении данных в этот topic от producer, он поочереди записывает сообщения в эти партиции: Информацию о платеже от клиента 1 в первую партицию, от клиента 2 во вторую, следующую информацию от клиента 1 в третью партицию. Этот топик можно разнести на несколько серверов, чтобы распределить нагрузку. Т.е. топик test может быть разнесён по нескольким серверам: 1-ая партиция на первый сервер, 2, 3я – на второй, именно для этого нужны партиции. Ещё есть сущность брокер (не Kafka, а именно техническое понятие) брокер – клиент Kafka для каждого из серверов, это оболочка для партиций. Т.е. партиции разносятся не по серверам, а по брокерам на этих серверах : 1 сервер = 1 брокер на нём (но может быть и больше, например, если выделять на сервере виртуальные машины и на них ставить брокеры или просто на один сервер создать несколько брокеров, однако это лишено смысла (на мой взгляд), т.к. мощности у сервера конечны, можно яблоко хоть на 100 частей разрезать, но больше его не станет). Соответственно: если есть 1 сервер, то и логично разворачивать на нём один брокер, вероятно поэтому Kafka называют брокер сообщений, а не брокерЫ сообщений.

**Решение**:

Имплементация задачи: Я поместил в docker 3 контейнера: server.ts – producer, контейнер с kafka, контейнер c zookeeper, который является балансировщиком запросов для kafka. После этого запустил эти контейнеры, server.ts успешно подключился к kafka, на сервере будет 1 АПИ эндпоинт: get(/send). По его вызове server.ts (producer) будет класть в kafka сообщение в топик test

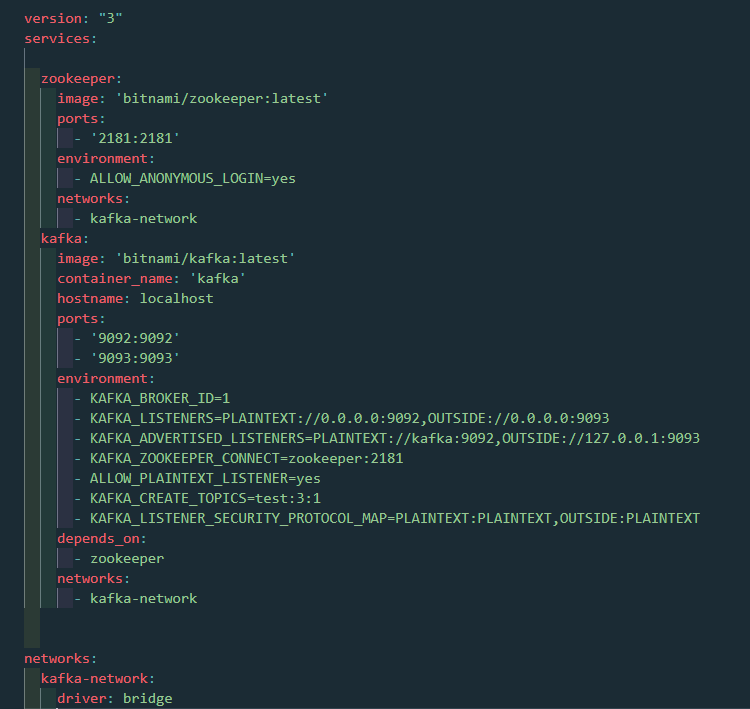
Cервер (server.ts) на node.js:



Здесь: brokers – список брокеров, с которыми сервер будет общаться, указал только 1 – kafka:9092 т.к. впоследствии я его помещу в контейнер, в одну и ту же сеть с контейнером kafka и подключаться он будет именно к этому контейнеру.

Перед перемещением сервера в контейнер и его запуском я развернул kafka:

docker-compose.yml:



Здесь описал 2 образа: zookeeper, kafka. В ports – порты, по которым контейнеры будут общаться 2181 для zookeeper, 9092 и 9093 для kafka. Рассмотрю переменные kafka\_listeners (1) и kafka\_advertised\_listeners (2). (1) переменная используется кафкой для приёма сообщений: 0.0.0.0:9092 – значит, что сообщения будет прослушивать со всех возможных интерфейсов. (2) – самая важная переменная, она используется для подключения других систем к кафке.

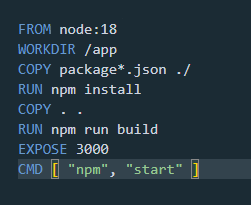
PLAINTEXT://kafka:9092

Это означает, что внутри сети docker необходимо именно так подключаться к брокеру, plaintext – протокол: в данном случае общение будет в виде открытого текста, но может быть и по TLS.

OUTSIDE://127.0.0.1:9093

Это подключение во вне сети docker. Consumer-ов я запускал через терминал, не включал в docker для большей реалистичности. Потребители будут общаться по localhost. Т.е. здесь я бы мог прописать url, если бы мои клиенты были не в локальной сети, а подключались извне (из интернета). Например, 227.24.13.32:9092 или kafka.net.com:9092 – если бы был зарегистрированный домен. Переменные ниже необходимы для того, чтобы сказать kafka, что подключение безопасно. После выполнения команды docker-compose up в папке с файлом docker-compose.yml kafka успешно развернётся в docker

Но как я запустил server.ts в docker ? Для этого я отдельно создал Dockerfile, в котором описал каким образом интерпретатору переместить server.ts в docker и запустить:

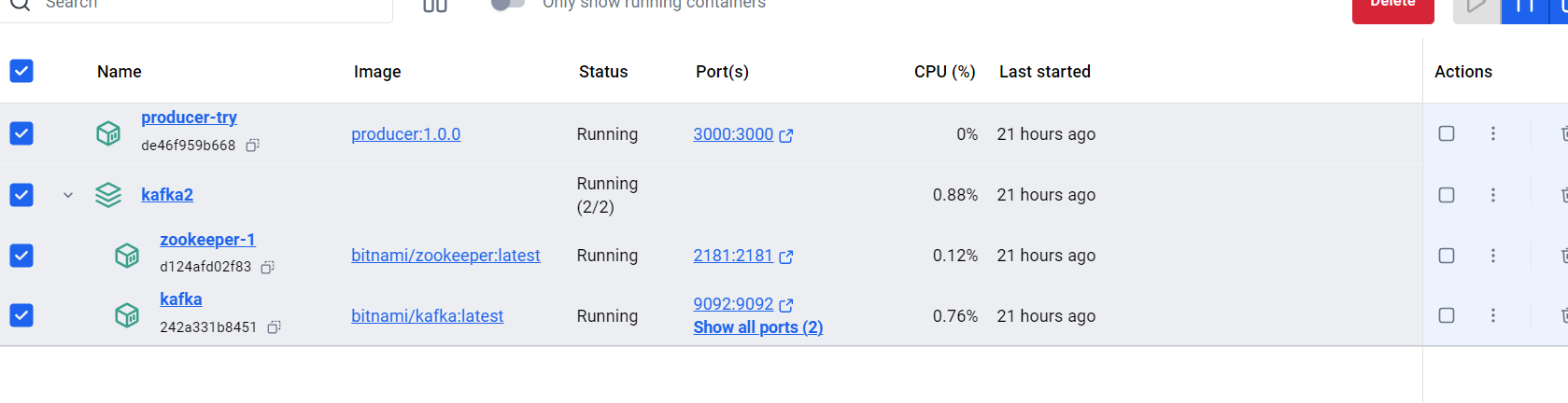


После запуска команд:

1. docker build --tag producer:1.0.0 . – создаёт образ контейнера, ещё пока не контенер
2. docker run -p 3000:3000 --name producer-try producer:1.0.0 – запускает контейнер из созданного выше образа на порту 3000, контейнер будет называться producer-try (важно для следующего шага)

сервер будет развёрнут в kafka. Файлы c описанием скрипта build в package.json будут на гитхабе

Итак, после развёртывания брокера Кафка и продюсера в docker:



Далее самая важная часть: почему-то по умолчанию эти контейнеры не видят друг друга. Т.е. при попытке подключиться producer выдаёт ошибку, для решения этой задачи я выполнил следующие команды:

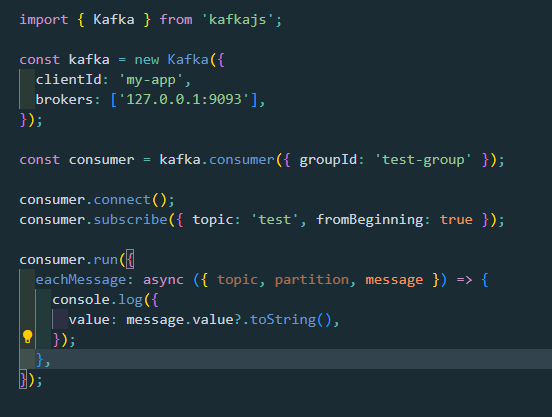
1. docker network create kafka-network
2. docker network connect kafka-netwokr producer-try (обратите внимание в сеть kafka-network помещаю уже работающий контейнер, а не образ. указываю название контейнера, а не его id)
3. docker network connect kafka-netwokr kafka2-zookeeper-1 – включаю в сеть ещё и контейнер балансировщика
4. docker network connect kafka-netwokr kafka – брокер тоже должен быть в той же сети.

Также при запуске producer-a с уже указанным топиком, которого как бы нет, kafka автоматически создаёт этот топик с 1 партицией, я его изменил командой

docker exec kafka /opt/bitnami/kafka/bin/kafka-topics.sh --bootstrap-server kafka:9092 --alter --topic test --partitions 10

И брокер и продюсер запущены. Осталось описать потребителей и подключить к kafka.

Для потребителей я использовал также node.js (далее file consumer.ts):



Здесь ключевые параметры:

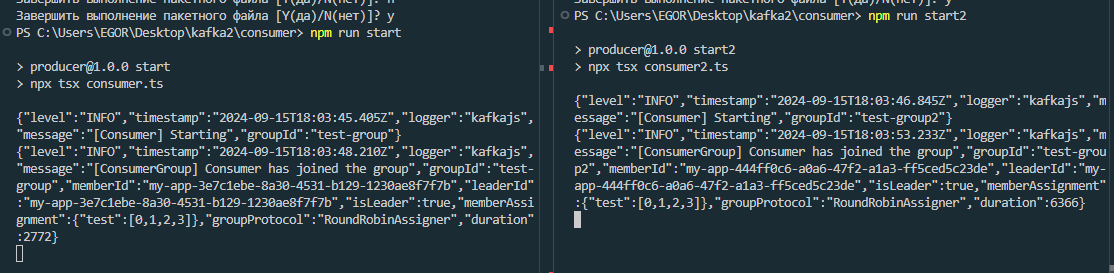
clientId – нужен, чтобы брокер различал клиентов,

brokers – список брокеров, из которых будет читать Кафка, обратите внимание, что consumer читает из сети отличной от docker – сети: 127.0.0.1:9093, если бы consumer находился в одной сети с producer и kafka, я бы заменил на kafka:9093,

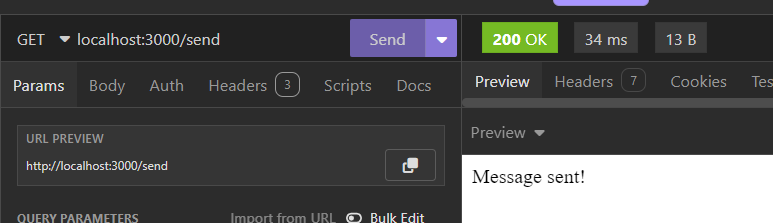
groupId – очень важный параметр, если у 2ух потребителей будет одинаковый groupId, то вместо дублирования информации, на этих потребителей, каждый потребитель будет получать часть информации, которую будет генерировать producer: kafka разделяет топик на партиции и записывает сообщения попеременно в них (описал выше), получится так, что потребители разделят цельную информацию между собой, поэтому вместо запусков 2ух инстансов этого файла, я написал ещё один файл consumer2.ts (абсолютно такой же, только другое название groupId)

topic – топик, партиции которого будет считывать consumer.

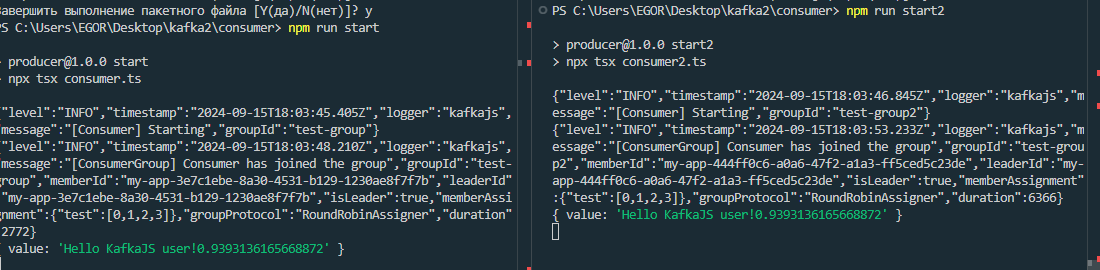
Дальше я запускаю 2 этих потребителя:



После этого я открываю Insomnia и дёргаю запрос, который заставит producer сгенерировать сообщение, положить его в топик Кафка и продюсеры его получат:



В результате оба потребителя получили это сообщение:



**Заключение**:

Для большей реалистичности можно было бы добавить ещё один брокер, который бы осуществлял репликацию на другом сервере (Основная функция Kafka), в этом случае партиции топика можно было бы либо полностью скопировать на другой хост (брокер) в целях безопасности, либо, наоборот, разделить топик test по этим 2ум серверам (брокерам) Т.е. 1ая и 3я партиция топика хранятся на одном сервере, 2ая на другом и консюмеры попеременно считывают информацию с этих брокеров (маштабируемость). Но примера, который я описал выше достаточно для понимания общих принципов работы Kafka.

**GitHub c проектом**:

**Литература**:

<https://chatgpt.com/>

https://www.youtube.com/watch?v=EiDLKECLcZw&t=75s&ab\_channel=KrisFoster

https://kafka.js.org/docs/getting-started

<https://habr.com/ru/articles/738874/>

<https://stackoverflow.com/questions/35437681/kafka-get-partition-count-for-a-topic>

<https://github.com/thoqbk/kafka-plain-java/blob/main/config/docker-compose.yml>

<https://www.confluent.io/blog/kafka-listeners-explained/>

<https://docs.docker.com/guides/language/nodejs/containerize/>

https://yandex.ru/video/preview/8090281345622297749