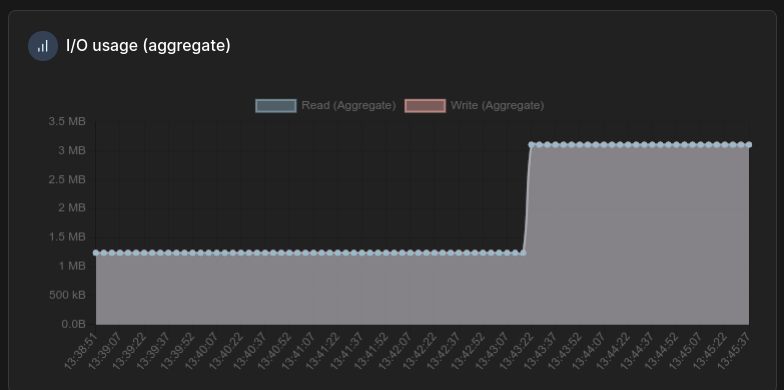
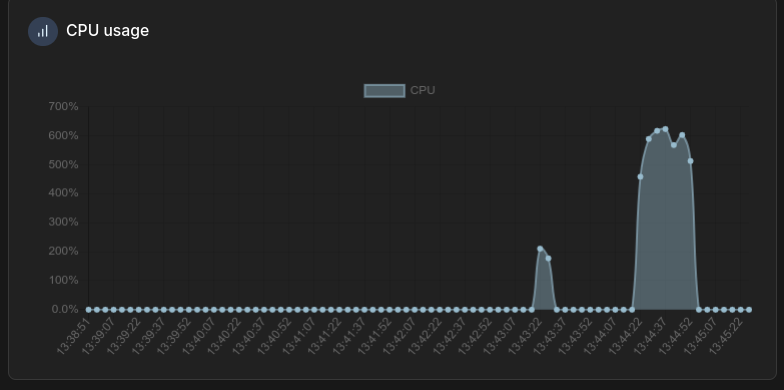
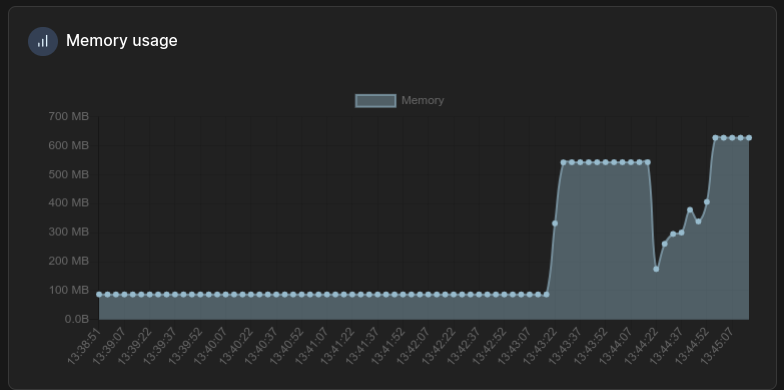
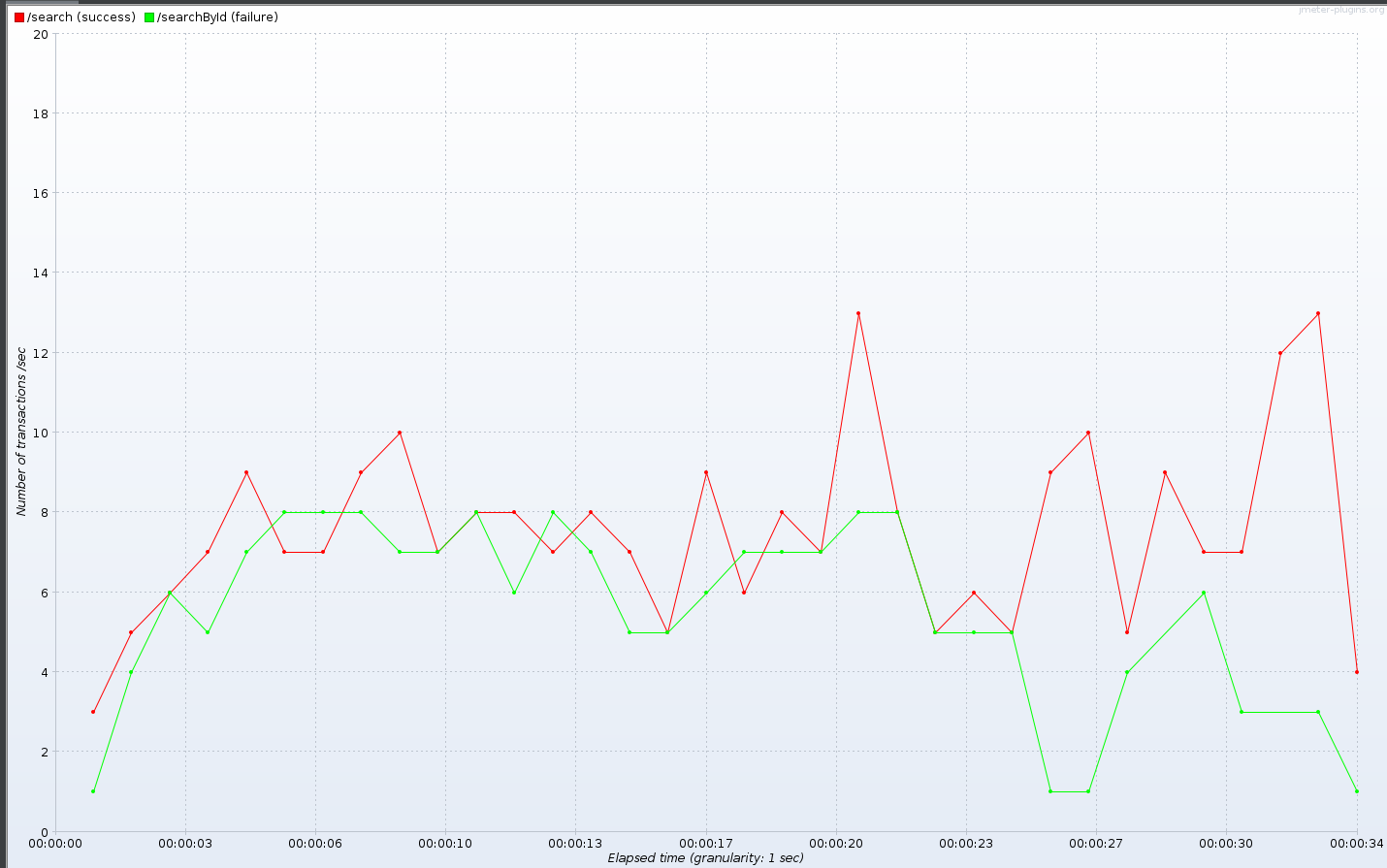
## **Отчет по домашней работе Репликация.**

На существующем приложения, до использования репликации сделала нагрузочные тесты. Графики нагрузки CPU,Mem,I/O docker контейнера pg\_master представлены ниже:



## Также для оценки производительности приложения замерила количество транзакций в секунду. График представлен ниже.



## **Асинхронная репликация**

## Согласно инструкции к домашнему заданию настроила асинхронную рекпликацию по схеме 1 мастер и два слэйва. Docker контейнеры pg\_master и pg\_slave, pg\_asyncslave соответственно.

Далее модернизировала код приложения homework\_backend для перевода методов чтения из БД (аналогичные /user/get/{id} и /user/search из спецификации) на слэйв. Конкретный слэйв указывается в файле конфигурации приложения параметром storage\_path\_for\_read. Пример содержимого файла конфигурации ниже:

*env: "local"*

*storage\_path: "host=pg\_master port=5432 user=postgres password=example dbname=otus\_homework sslmode=disable"*

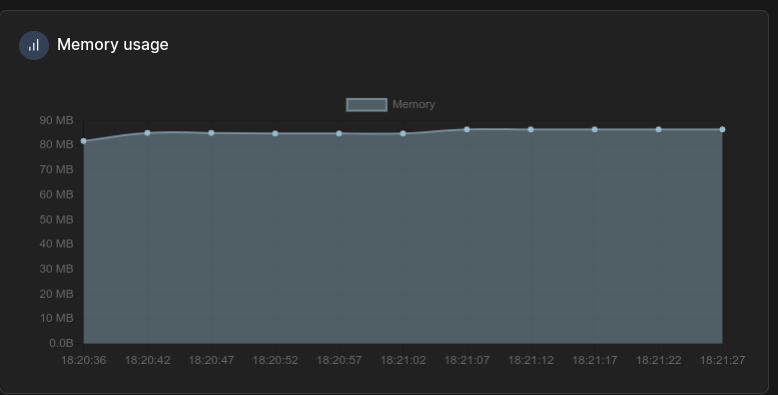
*storage\_path\_for\_read: "host=pg\_slave port=5432 user=postgres password=example dbname=otus\_homework sslmode=disable"*

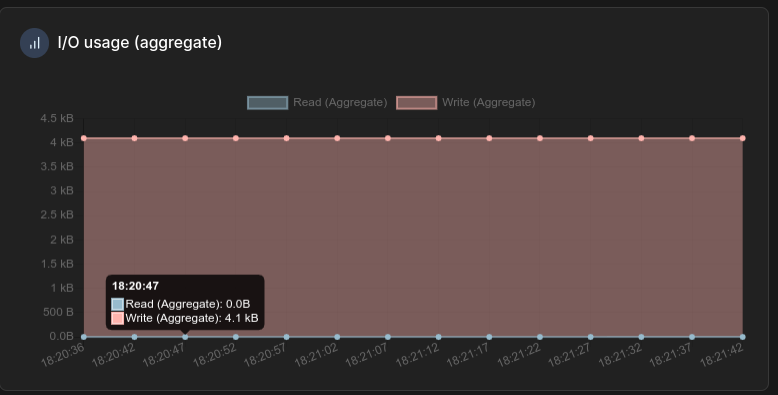
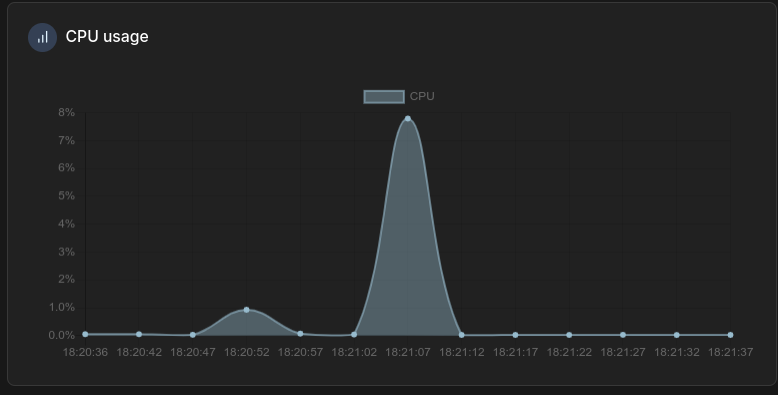
*grpc:*

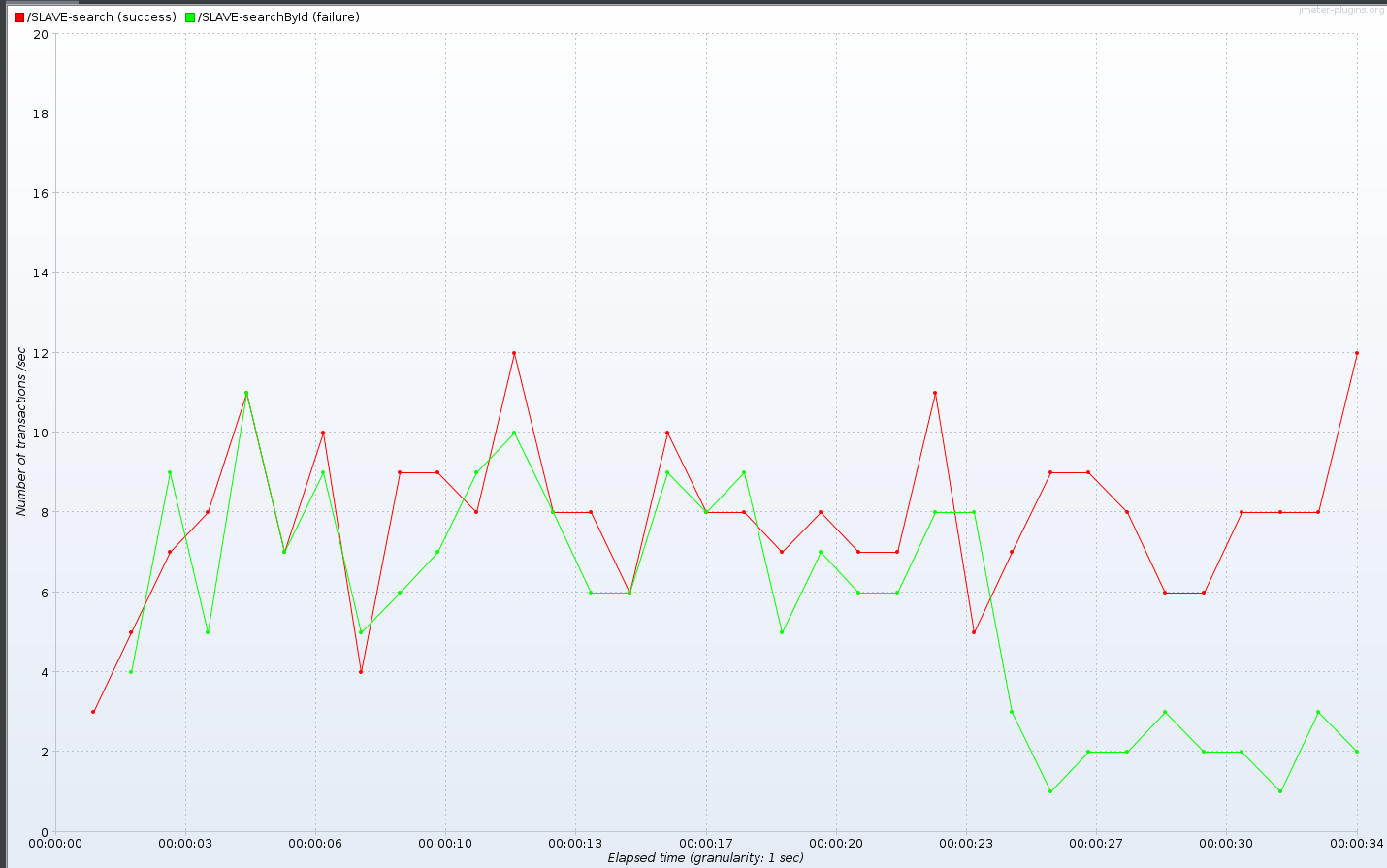
*port: 9001*

*timeout: 10h*

Провела нагрузочные тесты по этим методам и удостоверилась, что в момент чтения нагрузка перешла с контейнера pg\_master на контейнер pg\_slave, который я указала в конфигурационном файле. Графики нагрузки CPU,Mem,I/O docker контейнера pg\_master представлены ниже:







**Синхронная репликация**

## Согласно инструкции к домашнему заданию настроила синхронную репликацию по схеме 1 мастер и два слэйва, один из которых все еще поддерживает асинхронную репликацию. Docker контейнеры pg\_master и pg\_slave, pg\_asyncslave соответственно.

Проверку работы репликации осуществляла путем создания таблицы test в существующей базе otus\_homework и вставки в нее строк.

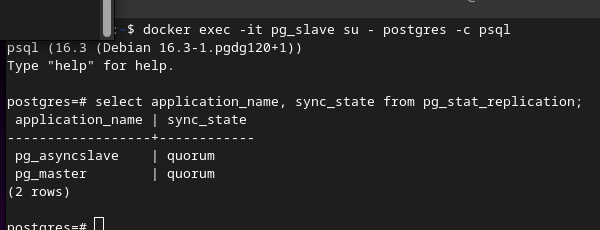
Первый тест: кладем контейнер pg\_asyncslave и проверяем работу pg\_master путем вставки данных в таблицу test запросом к pg\_master. Как результат все операции выполняются без ошибок. Далее поднимаем реплику pg\_asyncslave и делаем запрос на чтение из таблицы test. Данные соответствуют данным на pg\_master.

Второй тест: кладем контейнер с синхронной репликой pg\_slave и проверяем работу pg\_master путем вставки данных в таблицу test запросом к pg\_master. Операция вставки зависает. Поднимаем контейнер pg\_slave и операция вставки завершается с успехом.

Третий тест: кладем контейнер pg\_master и промоутим согласно инструкции синхронную реплику pg\_slave до нового мастера и пробуем вставить строку в таблицу test на новом мастере pg\_slave. Вставка успешна.

После этого переключаем асинхронную реплику pg\_asynclave на новый мастер путем изменения конфигурационного файла и перечитывания его. Также путем создания файла standby.signal на старом мастере pg\_master и изменением конфигурационного файла делаем из старого мастера реплику.

Убеждаемся что кворумная синхронизация настроена в конфиге новый мастер (pg\_slave) и два слэйва (pg\_asyncslave и pg\_master). Ниже принт скрин результатов запроса:



**Основные выводы**

* Использование репликации в целом позволило снизить нагрузку по чтению с мастера и перенести ее на слэйв.
* Использование синхронной репликации и возможность переключения мастера на одну из реплик позволяет избежать сбоев и восстановиться после них. Однако плохо то, что в данной конфигурации приходится переключение осуществлять вручную, либо написать скрипт переключения самой.