**Содержание**

1. Реализация балансировки и отказоустойчивости с использованием NGINX 1

1.1. Постановка задачи 1

1.2. Решение 1

2. Реализация разделяемого хранилища данных с использованием Redis 3

2.1. Постановка задачи 3

2.2. Решение 3

3. Реализация очередей задач с использованием очередей Redis 5

3.1. Постановка задачи 5

3.2. Решение 5

4. Реализация партиционирования с использованием POSTGRES 6

4.1. Постановка задачи 6

4.2. Решение 6

4.2.1. Создание таблицы 6

4.2.2. Добавление партиций 6

4.2.3. Добавление индексов 6

# Реализация балансировки и отказоустойчивости с использованием NGINX

## Постановка задачи

В рамках данной лабораторной работы необходимо:

1. Разработать приложение, содержащее HTTP endpoint, при обращении к которому возвращается ответ вида {“counter”: “1”}. При каждом обращении счетчик должен увеличиваться;
2. Запустить несколько экземпляров данного приложения на разных портах;
3. Запустить Nginx, который балансирует нагрузку между запущенными веб-сервисами;
4. Реализовать и проанализировать различные алгоритмы балансировки: round robin, hash, least conn, least time, random;
5. Реализовать и проанализировать функционал реализации отказоустойчивости (fail\_timeout, max\_fails) при отключении одного или нескольких веб-сервисов;
6. Проанализировать результаты нагрузочного тестирования с использованием Apache Benchmark;

## Решение

В рамках работы реализован простой «Counter», который при нажатии увеличивает значение (рис.1).

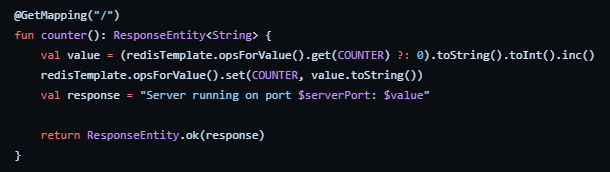


Рисунок 1 – листинг кода: Counter

Для работы с nginx был составлен файл конфигурации (рис. 2)



Рисунок 2 – содержание файла nginx.conf

Для реализации нагрузочного тестирования использовался Apache Benchmark. Nginx и Apache Benchmark были установлены и запущены в Ubuntu, через WSL на Windows.

Таблица 1 – Результаты применения различных методов балансировки

|  |  |  |  |
| --- | --- | --- | --- |
| Метод балансировки | Применение | Затраченное время, с | Самый долгий запрос, мс |
| Round Robin | default | 7,2 | 152 |
| Least Connections | least\_conn | 5,4 | 96 |
| Hash | hash $remote\_addr | 5,5 | 74 |
| Random | random | 6,2 | 102 |

# Реализация разделяемого хранилища данных с использованием Redis

## Постановка задачи

В рамках работы необходимо:

1. Запустить redis;
2. Доработать приложение из предыдущей лабораторной работы таким образом, чтобы счетчик входящих запросов хранился в redis;
3. Доработать приложение, создав соответствующий HTTP REST API, чтобы в redis использовалась структура Redis Sorted Set.

## Решение

Для выполнения работы был установлен Redis, на ВМ с Ubuntu. Для его запуска необходимо прописать команду: sudo systemctl start redis*.*

Также был доработан Controller для работы с Redis и таблицей лидеров.

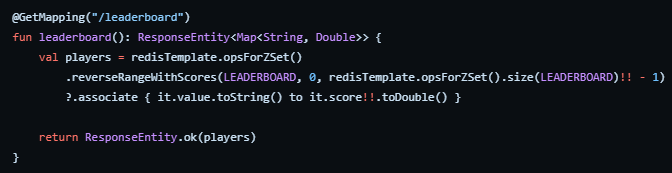


Рисунок 3 – описание endpoints для нового метода

Для проверки были добавлены следующие игроки:

1. Игрок неудачник (количество очков: 10.0);
2. Игрок нормис (количество очков: 20.0);
3. Игрок машина XXL (количество очков: 30.0).

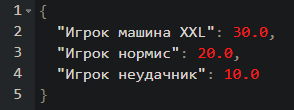


Рисунок 4 – вывод рейтинга игроков

Из рисунка видно, что игроки автоматически подверглись сортировки. Для этого использовались так называемые Z-операции. Их применяют для сортировки данных.

# 3. Реализация очередей задач с использованием очередей Redis

## 3.1. Постановка задачи

В рамках реализации третьей лабораторной работы необходимо:

1. Запустить redis;
2. Реализовать веб-сервис, который с неравномерной скоростью записывает сообщения в очередь. Например, загружает данные из файла и записывает их в очередь;
3. Реализовать веб-сервис, который с определенной задержкой обрабатывает данные в очереди.

## Решение

Для выполнения лабораторной работы было решено использовать приложение из прошлых лабораторных работ, просто дополнить его, а не создавать новое.

Для её «обновления» было сделано следующее:

1. В контроллер добавлен метод, который в цикле со случайной задержкой (от 1 до 5 секунд) будет записывать итерации в redis
2. Добавлен endpoint /start – запуск цикла записи в очередь сообщений, который выполняется асинхронно и сразу запускает чтение из очереди с неизменяемой задержкой в 1 секунду между сообщениями.

# Реализация партиционирования с использованием POSTGRES

## Постановка задачи

В рамках работы необходимо:

1. Необходимо запустить СУБД с поддержкой партиционирования;
2. Создать таблицу с использованием партиционирования;
3. Написать запросы:
   * вставки данных;
   * добавления и удаления партиций;
   * создания глобальных и локальных индексов;
   * выборки данных и использованием индексов.

## Решение

### Создание таблицы

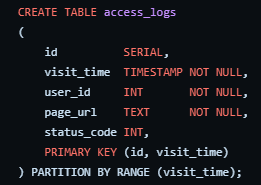


Рисунок 5 – создание таблицы

### Добавление партиций

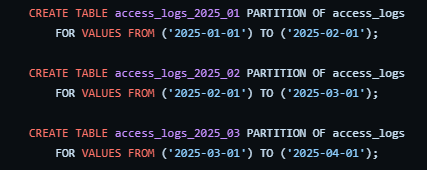


Рисунок 6 – добавление партиций по месяцам

### Добавление индексов

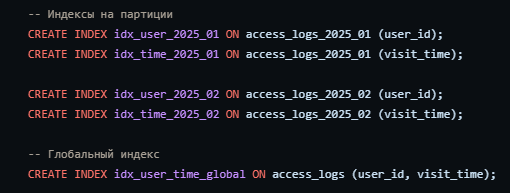


Рисунок 7 – добавление индексов

### Заполнение данными

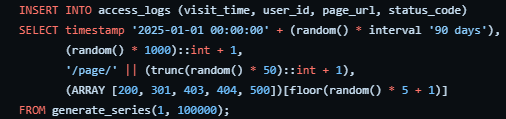


Рисунок 8 – заполненение данными