Министерство образования и науки Российской Федерации

Пермский национальный исследовательский политехнический университет

Разработка высоконагруженных веб-приложений

**О Т Ч Е Т**

**Лабораторные работы и презентация**

Выполнил студент гр. ИТСИ-19-1м

Романов Никита Сергеевич

Пермь 2020

**Лабораторные работы**

**Небольшое вступление начёт лабораторных**

Материал и файлы по всем лабораторным работам можно найти в репозитории по ссылке: <https://github.com/RomanovNS/WebDev-Highload-and-Cloud->

Некоторые лабораторные работы выполнены с помощью приложения LabsLoader, в котором находятся классы, названные соответственно выполненной лабораторной работе. Эти классы реализуют лабы, т.е. могут соответсвующими методами, например, собрать снимки контейнеров докера, запустить все необходимые для лабы контейнеры или остановить их. В свою очередь классы лабораторных работ, используют класс Docker, созданный для упрощения, чтобы не приходилось возиться с терминалом. В нём реализован вызов докер-команд из Java-приложения, за счёт метода runCommand (да, можно было использовать sh-скрипты, но я сделал как сделал, мне так удобнее).

**1. Балансировка нагрузки с Nginx**

**О чём эта лаба**

Для реализации данной лабы надо было создать простое веб-приложение, содержащее HTTP endpoint, при обращении к которому возвращается ответ вида {“counter”: “1”} (сделал по-другому: { "WebAppID": 0, "counter": 1 } ). При каждом обращении счетчик должен увеличиваться. Далее необходимо запустить несколько экземпляров данного приложения и Nginx, который будет балансировать нагрузку между ними.

В данном решении все экземпляры приложения и nginx запускаются в контейнерах Docker.

**Файлы к этой лабе**

Директории и файлы, используемые для данной лабы:

* DockerFiles
  + dockerfile\_counterApp (для сборки контейрера с нашим веб-приложением)
  + dockerfile\_nginxLoadBalancer\_Hash (для сборки контейнера с nginx, балансирующего по алгоритму Hash)
  + dockerfile\_nginxLoadBalancer\_IpHash (для сборки контейнера с nginx, балансирующего по алгоритму IP Hash)
  + dockerfile\_nginxLoadBalancer\_LeastConn (для сборки контейнера с nginx, балансирующего по алгоритму Least Connections)
  + dockerfile\_nginxLoadBalancer\_LeastTime (для сборки контейнера с nginx, балансирующего по алгоритму Least Time)
  + dockerfile\_nginxLoadBalancer\_Random (для сборки контейнера с nginx, балансирующего по алгоритму Random)
  + dockerfile\_nginxLoadBalancer\_RoundRobin (для сборки контейнера с nginx, балансирующего по алгоритму Round Robin)
* JARs
  + SimpleSpringBootApp.jar (JAR с нашим веб-приложением)
* NginxConfigFiles
  + nginxLoadBalancer\_Hash.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Hash)
  + nginxLoadBalancer\_IpHash.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму IP Hash)
  + nginxLoadBalancer\_LeastConn.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Least Connections)
  + nginxLoadBalancer\_LeastTime.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Least Time)
  + nginxLoadBalancer\_Random.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Random)
  + nginxLoadBalancer\_RoundRobin.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Round Robin)
* SimpleSpringBootApp (папка с maven-проектом веб-приложения, из которого получили JARник)

**Как проверить эту лабу**

Чтобы запустить контейнеры с экземплярами веб-приложения и Nginx-ом, нужно всего лишь прописать в main у LabsLoader-a

LabHighLoad1.prepare();

LabHighLoad1.start();

и запустить его. Он создаст и запустит 4 контейнера с веб-приложением и 1 с nginx.

Чтобы остановить контейнеры, нужно прописать в main у LabsLoader-a

LabHighLoad1.stop();

и запустить проект.

Для выбора способа балансировки нужно раскомменить соответствующую строку в LabsHighLoad1.start(){}, взамен той, что используется.

**2. Разделяемое хранилище данных с использованием Redis**

**О чём эта лаба**

В рамках данной лабы было нужно доработать так приложение из предыдущей лабораторной, чтобы счётчик входящих запросов хранился в Redis и, таким образом, был общим.

**Файлы к этой лабе**

Директории и файлы, используемые для данной лабы:

* DockerFiles
  + dockerfile\_counterAppRedis (для сборки контейрера с нашим веб-приложением)
  + dockerfile\_nginxLoadBalancer\_Random (для сборки контейнера с nginx, балансирующего по алгоритму Random)
  + docker-compose\_redis.yml
* JARs
  + CounterAppForRedis.jar (JAR с нашим веб-приложением)
* NginxConfigFiles
  + nginxLoadBalancer\_Random.conf (конфигурационный файл nginx, в котором определена балансировка по алгоритму Random)
* CounterAppForRedis (папка с maven-проектом веб-приложения, из которого получили JARник)

**Как проверить эту лабу**

Чтобы подготовить все докер-снимки, нужно всего лишь прописать в main у LabsLoader-a

LabHighLoad2.prepare();

и запустить его.

Чтобы запустить все контейнеры, нужно поменять запускаемый метод на

LabHighLoad2.start();

Данный метод запустит командную строку для docker-compose.

Чтобы остановить контейнеры, нужно прописать в main у LabsLoader-a

LabHighLoad2.stop();

и запустить проект.

**3. Балансировки нагрузки с использованием очередей Kafka**

**О чём эта лаба**

В рамках данной лабораторной работы необходимо было реализовать 2 приложения: одно неравномерно загружает данные в очередь Kafka-топика, а другой с определённой задержкой берёт данные из очереди.

В данной работе для создания Kafka-сервера использовался docker-compose.yml от Confluence.

**Файлы к этой лабе**

Директории и файлы, используемые для данной лабы:

* DockerFiles
  + docker-compose\_kafkaConfluence.yml
* KafkaConsumer (приложение, которое будет читать из очереди)
* KafkaProducer (приложение, которое будет писать в очередь)

**Как проверить эту лабу**

1. Запустите кафка-сервер с помощью docker-compose\_kafkaConfluence.yml;
2. Запустите KafkaConsumer и KafkaProducer;
3. Наблюдайте, как пишутся и считываются сообщения из очереди.

**4. Реализация партиционирования с использованием Postgres**

**О чём эта лаба**

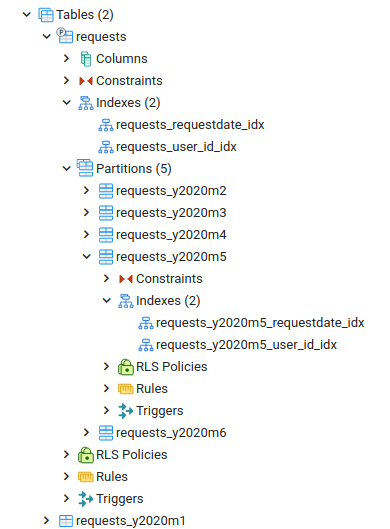
В данной работе нужно было создать разделённую таблицу и написать запросы создания/удаления/добавления таблицы/партиций создания глобальных и локатьных индексов, выборки данных

**Файлы к этой лабе**

* SQLfiles
  + requests.sql

**Как проверить эту лабу**

Отправьте запросики из файла к своей БД. Результат будет примерно таким же как на картинке

[](https://github.com/RomanovNS/WebDev-Highload-and-Cloud-/blob/master/ImagesForReadme/postgresResults.png)

**Презентация**

