Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Факультет: Прикладная математика и механика

Кафедра: Вычислительная математика, механика и биомеханика

Направление: 09.04.02 «Информационные системы и технологии»

Программа: «Информационные технологии и системная инженерия»

**Отчет**

**По лабораторным работам в рамках дисциплины**

**«Разработка высоконагруженных web-приложений»**

Выполнил:

студент гр. ИТСИ-18-1м  
Кандаков А.А.

(*Ф.И.О.*)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись, дата)*

Оглавление

[Введение 3](#_Toc28111321)

[Лабораторная работа №1 «Балансировка nginx» 4](#_Toc28111322)

[Лабораторная работа №2 «Отказоустойчивость nginx» 5](#_Toc28111323)

[Лабораторная работа №3 «Redis» 5](#_Toc28111324)

[Лабораторная работа №4 «Postgres partitioning» 8](#_Toc28111325)

[Доклад: «OpenShift» 10](#_Toc28111326)

[Заключение 13](#_Toc28111327)

# Введение

В рамках курса «разработка высоконагруженных web приложений» был разобран теоретический материал и выполнен ряд лабораторных работ:

1. Использование **nginx** для балансировки нагрузки;
2. Использование **nginx** для обеспечения отказоустойчивости;
3. Использование **Redis** в качестве хранилища данных приложения;
4. Организация партиционирования в **БД Postgres**;

Также был подготовлен доклад на тему «OpenShift ».

# Лабораторная работа №1

# «Балансировка nginx»

**Задание.**  
Реализовать и продемонстрировать балансировку нагрузки на web-приложение с помощью технологии nginx.

**Ход работы.**

Было разработано простое web-приложение счетчик на языке java с двумя эндпоинтами: «/count» - Инкрементация; «/stat» - Чтение счетчика.

Затем в Docker было запущенно несколько экземпляров приложения, а так же сконфигурированный для балансировки nginx.

Структура docker-compose.yml:

**version: '3.7'**

**services:**

**RedisApplication:**

**image: leoperelygin/redis1**

**ports:**

**- 8082:8080**

**RedisApplication1:**

**image: leoperelygin/redis1**

**ports:**

**- 8085:8080**

**nginx:**

**image: nginx:1.17.2-alpine**

**ports:**

**- 81:80**

**volumes:**

**- ./default.conf:/etc/nginx/conf.d/default.conf**

Конфигурация nginx осуществляется по средствам файла default.conf имеющего следующее содержание:

**upstream application {**

**server RedisApplication:8080;**

**server RedisApplication1:8080;}**

**server {**

**listen 80;**

**charset utf-8;**

**access\_log off;**

**location / {**

**proxy\_pass http://application;**

**proxy\_set\_header Host $host:$server\_port;**

**proxy\_set\_header X-Forwarded-Host $server\_name;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;}**

В результате система получила возможность балансировки нагрузки реализуемую nginx при обратном проксировании.

# Лабораторная работа №2

# «Отказоустойчивость nginx»

**Задание.**

Обеспечить отказоустойчивость приложения из лабораторной работы №1 средствами nginx.

**Ход работы.**

Для реализации отказоустойчивости была проведена дополнительная настройка сервера nginx. Произошли следующие пополнения в конфигурационном файле:

**upstream application {**

**server RedisApplication:8080 weight=2 max\_fails=2 fail\_timeout=2s;**

**server RedisApplication1:8080 weight=2 max\_fails=2 fail\_timeout=2s;**

**}**

**max\_fails** – количество «провалов» сервера, после которого nginx перестаёт посылать на него запросы в течении следующих **fail\_timeout** секунд.

Тестирование нагрузки производилось с помощью утилиты **apachebenchmark.** Генерировалась нагрузка на систему в 10 000 запросов.

В момент обработки запросов, один из серверов принудительно отключался. Nginx корректно обработал аварийную ситуацию и перераспределил нагрузку между оставшимися серверами. В результате все запросы были доставлены.

# Лабораторная работа №3 «Redis»

**Задание.**

Модифицировать работу программы из лабораторной работы №2. Перенести счетчик на отдельный сервер с key-value хранилищем Redis.

**Ход работы.**

Redis (от англ. remote dictionary server) - резидентная система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа «ключ-значение». Используется как для баз данных, так и для реализации кэшей, брокеров сообщений. Ориентирована на достижение максимальной производительности на атомарных операциях.

Для работы с Redis в java используется библиотека Jedis. Для ее подключения в POM файле проекта прописана следующая зависимость:

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>1.5.2</version>

</dependency>

При запуске приложения устанавливается коннект с redis через jedis и пересоздается переменная «counter» с нулевым значением:

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

Jedis jedis = new Jedis("redis");

jedis.del("counter");

jedis.set("counter", "0");}

Методы «count» и «stat» модифицируются следующим образом:

@RequestMapping("/count")

public String count() {

String buff = i.incrementAndGet() + " | " + jedis.incr("counter" ) + "\r";

return buff;}

@RequestMapping("/stat")

public String stat() {

String buff = i + " | " + jedis.get("counter" ) + "\r";

return buff;}

В данном случае решено было сохранить оба счетчика в целях демонстрации наглядной работы балансировщика нагрузки.

Так же, для непосредственного развертывания redis в контейнере docker, yaml-файл получил следующее дополнение:

redis:

image: 'bitnami/redis:latest'

environment:

- ALLOW\_EMPTY\_PASSWORD=yes

restart: always

volumes:

- ./storage/redis/data:/data

ports:

- 6379:6379

Результатом работы стал вынесенный в отдельный контейнер, общий, потокобезопасный и быстрый счетчик.

# Лабораторная работа №4

# «Postgres partitioning»

**Задание.**

Продемонстрировать работу инструмента секционирования (partitioning) на примере базы данных PostgresSQL.

**Ход работы.**

Секционированием данных называется разбиение одной большой логической таблицы на несколько меньших физических секций.

PostgreSQL предлагает поддержку следующих видов секционирования:

***Секционирование по диапазонам:*** Таблица секционируется по «диапазонам», определённым по ключевому столбцу или набору столбцов, и не пересекающимся друг с другом. Например, можно секционировать данные по диапазонам дат или по диапазонам идентификаторов определённых бизнес-объектов.

***Секционирование по списку:*** Таблица секционируется с помощью списка, явно указывающего, какие значения ключа должны относиться к каждой секции.

***Секционирование по хешу:*** Таблица секционируется по определённым модулям и остаткам, которые указываются для каждой секции. Каждая секция содержит строки, для которых хеш-значение ключа разбиения, делённое на модуль, равняется заданному остатку.

В качестве образца воспользуемся официальными материалами предлагаемыми postgres: <https://postgrespro.ru/docs/postgresql/11/ddl-partitioning>.

Создадим пример таблицы для компании торгующей мороженым:

CREATE TABLE measurement (

city\_id int not null,

logdate date not null,

peaktemp int,

unitsales int

)PARTITION BY RANGE (logdate);

При создании таблица указывается как секционированная. Выбран метод секционирования RANGE а в качестве столбцов образующих ключ разбиения «logdate».

Затем создадим несколько секций:

CREATE TABLE measurement\_y2006m02 PARTITION OF measurement

FOR VALUES FROM ('2006-02-01') TO ('2019-11-01');

CREATE TABLE measurement\_y2006m03 PARTITION OF measurement

FOR VALUES FROM ('2006-03-01') TO ('2019-12-01');

В результате «под капотом» таблица measurement оказывается разделенной на секции, и соответствующие записи помещаются именно в них. Таблица теперь не имеет общей индексации, а каждая секция имеет свой отдельный индекс. При обращении к таблице postgres работает с соответствующими секциями исходя из ключа разбиения. Такой подход может принести следующую пользу:

* В определённых ситуациях кардинально увеличивает быстродействие, особенно когда большой процент часто запрашиваемых строк таблицы относится к одной или лишь нескольким секциям.
* Когда в выборке или изменении данных задействована большая часть одной секции, последовательное сканирование этой секции может выполняться гораздо быстрее, чем случайный доступ по индексу к данным, разбросанным по всей таблице.
* Массовую загрузку и удаление данных можно осуществлять, добавляя и удаляя секции, если это было предусмотрено при проектировании секционированных таблиц. Операция ALTER TABLE DETACH PARTITION или удаление отдельной секции с помощью команды DROP TABLE выполняются гораздо быстрее, чем массовая обработка. Эти команды также полностью исключают накладные расходы, связанные с выполнением VACUUM после DELETE.
* Редко используемые данные можно перенести на более дешёвые и медленные носители.

Всё это обычно полезно только для очень больших таблиц. Какие именно таблицы выиграют от секционирования, зависит от конкретного приложения, хотя, как правило, это следует применять для таблиц, размер которых превышает объём ОЗУ сервера.

# Доклад: «OpenShift»

OpenShift - это открытая и расширяемая платформа приложений-контейнеров, которая позволяет использовать Docker и Kubernetes на предприятии.

**Обзор**

OpenShift включает Kubernetes для оркестрации контейнеров и управления ими. Здесь добавлены инструменты для разработчиков и операций, позволяющие:

* быстро разрабатывать приложения;
* легко развертывать и масштабировать решения;
* обслуживать жизненный цикл команд и приложений в долгосрочной перспективе.

Существует несколько версий OpenShift:

* Платформа контейнеров OpenShift;
* OpenShift в Azure (полностью управляемый OpenShift ожидается в начале 2019);
* OKD (прежнее название — OpenShift Origin);
* OpenShift Dedicated;
* OpenShift Online.

**Платформа контейнеров OpenShift**

OpenShift Container Platform — это коммерческая версия корпоративного класса, которая разрабатывается и поддерживается компанией Red Hat. Чтобы использовать эту версию, клиенты должен приобрести необходимые права доступа к платформе OpenShift Container Platform. Кроме того, они сами устанавливают и администрируют всю инфраструктуру. Так как клиенты являются владельцами платформы, они могут установить ее в локальном центре обработки данных или в общедоступном облаке (Azure, AWS или Google).

**OpenShift в Azure**

OpenShift в Azure — это полностью управляемое предложение OpenShift, выполняемое в Azure. Эта служба управляется и поддерживается совместно корпорацией Майкрософт и Red Hat. Кластер будет развернут в подписку Azure клиента. В настоящий момент служба находится в закрытой предварительной версии, а в начале CY 2019 планируется выпустить общедоступную версию. Дополнительные сведения будут представлены, когда предложение будет ближе к общедоступной версии. OKD (прежнее название — OpenShift Origin); OKD — это высокоуровневый проект OpenShift с открытым кодом, который поддерживается сообществом. OKD можно установить в ОС CentOS или Red Hat Enterprise Linux (RHEL).

**OpenShift Dedicated**

OpenShift Dedicated — это управляемая Red Hat однотенантная (получающая доступ к другим службам в выделенной среде) OpenShift, которая использует платформу контейнеров OpenShift. Red Hat управляет всей базовой инфраструктурой (виртуальными машинами, кластером OpenShift, сетью, хранилищем и т. д.). Кластер связан только с одним клиентом и выполняется в общедоступном облаке (AWS или Google). Начальный кластер включает четыре узла приложений, и все затраты — ежегодные и оплачиваются заранее.

**OpenShift Online**

OpenShift Online — это мультитенантная(обращающаяся к службам в общей среде в нескольких организациях) платформа OpenShift (на базе Container Platform) под управлением Red Hat. Red Hat управляет всей базовой инфраструктурой (виртуальными машинами, кластером OpenShift, сетью и хранилищем).

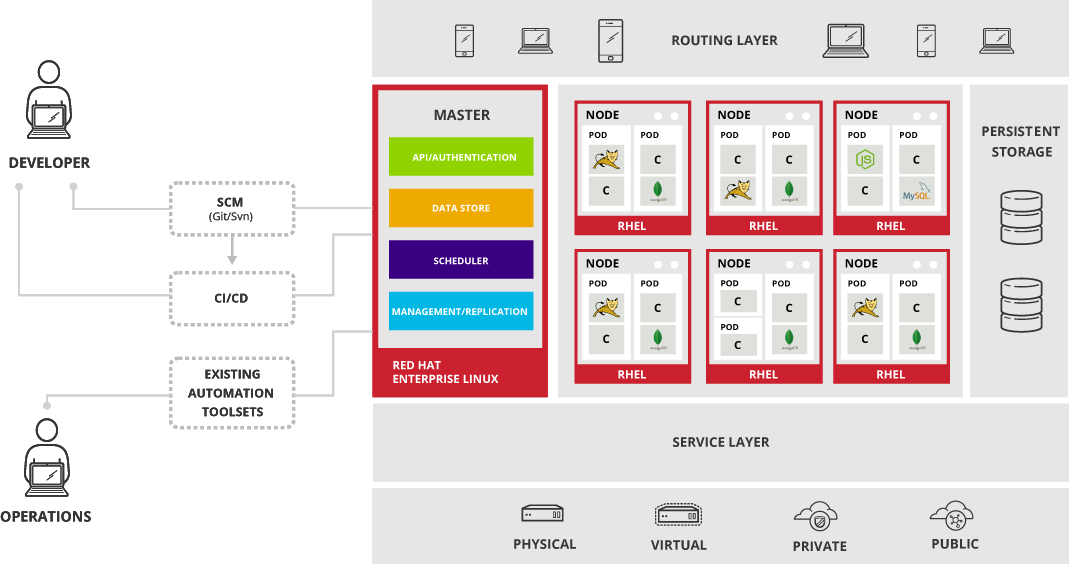
Чтобы использовать эту версию, клиенты развертывают контейнеры, но при этом они не могут выбирать, на каких узлах эти контейнеры выполняются. Так как OpenShift Online — это мультитенантная среда, контейнеры могут размещаться на тех же узлах виртуальных машин, что и контейнеры других клиентов. Стоимость вычисляется из расчета на один контейнер.

**Преимущества для разработчиков**

OpenShift Container Platform предоставляет оптимальную платформу для подготовки (provisioning), сборки и развертывания приложений и их компонентов в режиме самообслуживания. Средства автоматизации, наподобие встроенной конвертации S2I (source -to-image), значительно упрощают сборку контейнерных образов в формате docker на основе кода, извлеченного из системы контроля версий. Интеграция с инструментами непрерывной интеграции и доставки (CI/CD) превращает OpenShift Container Platform в идеальное решение для любой организации.

**Преимущества для специалистов IT-систем**

OpenShift Container Platform – это корпоративная Kubernetes -платформа приложений с развитыми средствами автоматизации и управления на основе политик. Встроенные средства кластеризации, планирования и оркестрации обеспечивают эффективную балансировку нагрузки и автомасштабирование. Функции безопасности полностью устраняют риски вмешательства tenant - клиентов в работу других приложений или хоста, а возможность подключения постоянного хранилища непосредственно к контейнерам Linux® позволяет одновременно использовать эту платформу для приложений stateful и stateless.



# Заключение

В рамках дисциплины «Разработка высоконагруженных web-приложений» были проведены следующие работы:

* Изучен теоретический материал по темам курса.
* Выполнен цикл лабораторных работ по теме.
* А так же подготовлен и представлен доклад о технологии «Red Hat OpenShift Container Platform».