Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики и механики

Направление: 09.04.02 «Информационные системы и технологии»

Программа: «Информационные технологии и системная инженерия»

Отчёт по лабораторным работам

по дисциплине

«РАЗРАБОТКА ВЫСОКОНАГРУЖЕННЫХ

WEB-ПРИЛОЖЕНИЙ»

Выполнила:

студент гр. ИТСИ-17-1м

Боталова Юлия Евгеньевна

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

*(подписи)*

Принял:

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

*(должность, ФИО руководителя)*

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

*(оценка) (подпись)*

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

*(дата)*

**Пермь 2018**

Оглавление

[Глава 1. Лабораторные работы 3](#_Toc531694656)

[Лабораторная работа №1. Docker, NGINX 3](#_Toc531694657)

[Лабораторная работа №2. Redis. Tutorial 5](#_Toc531694658)

[Лабораторная работа №3. Партиционирование 12](#_Toc531694659)

[Глава 2. DDOS-атака 15](#_Toc531694660)

[Что такое DDOS-атака? 15](#_Toc531694661)

[Методы защиты 17](#_Toc531694662)

[Как устроить DDOS-атаку 22](#_Toc531694663)

[Использованные источники 23](#_Toc531694664)

# Глава 1. Лабораторные работы

## Лабораторная работа №1. Docker, NGINX

**Цель работы:** получить навыки работы с docker, создания приложений (которые могут быть запущены в docker), получить навык запуска балансировщика в docker.

**Ход работы:**

1. Устанавливаем NGINX через пакетный менеджер apt-get.
2. Создаем новый каталог для контента веб-приложения в домашнем каталоге и перейдем к нему, выполнив приведенные ниже команды:

* **$mkdir -p /docker-nginxx/html**
* **$cd= ~/docker-nginx/html**

1. Создаем HTML-файл:

* **$sudo nano index.html**

Вставляем следующее содержимое:

**<html>**

**<head>**

**<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" rel="stylesheet" integrity="sha256-MfvZlkHCEqatNoGiOXveE8FIwMzZg4W85qfrfIFBfYc= sha512-dTfge/zgoMYpP7QbHy4gWMEGsbsdZeCXz7irItjcC3sPUFtf0kuFbDz/ixG7ArTxmDjLXDmezHubeNikyKGVyQ==" crossorigin="anonymous">**

**<title>Docker nginx Tutorial</title>**

**</head>**

**<body>**

**<div class="container">**

**<h1>Hello</h1>**

**<p>This nginx page is brought to you by Docker</p>**

**</div>**

**</body>**

**</html>**

Сохраняем файл, он и будет индексной страницей для замены целевой страницы Nginx по умолчанию.

1. Связываем контейнер с локальной файловой системой

Контейнер Nginx будет доступен через порт 80, и подключим его к содержимому нашего сайта на сервере. Docker позволяет связывать каталоги из локальной файловой системы виртуальной машины с контейнерами.

Контейнер Nginx по умолчанию настроен на поиск индексной страницы /usr/share/nginx/html, поэтому в новом контейнере Docker нужно предоставить ему доступ к файлам в этом месте.

Для создания ссылки мы используем флаг –v для сопоставления папки с нашей локальной машины ( ~/docker-nginx/html) с относительным путем в контейнере ( /usr/share/nginx/html). Выполняем следующую команду:

* **$sudo docker run --name docker-nginx -p 81:80 -d -v /docker-nginxx/html:/usr/share/nginx/html nginx**

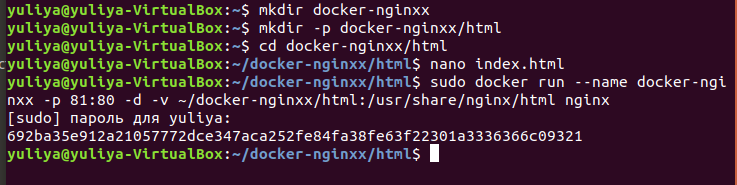


Рисунок 1. Настройка работы Docker-NGINX

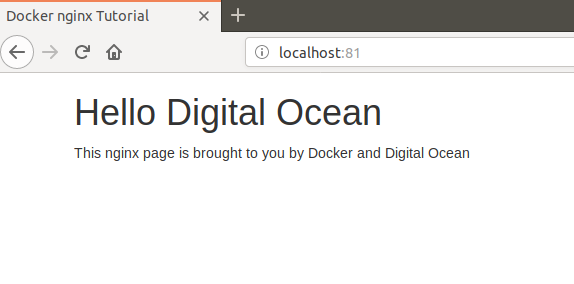


Рисунок 2. Результат работы Docker-NGINX

## Лабораторная работа №2. Redis. Tutorial

Redis (англ. remote dictionary server) — сетевое журналируемое хранилище данных типа «ключ — значение» с открытым исходным кодом. Redis - это так называемое хранилище ключей, которое часто называют базой данных NoSQL. Суть хранилища ключей - возможность хранить некоторые данные, называемые значением, внутри ключа. Эти данные могут быть впоследствии получены только в том случае, если мы знаем точный ключ, используемый для его хранения.

Redis является открытым исходным кодом (лицензия BSD), хранилище данных в памяти, используемое в качестве базы данных, кэша и брокера сообщений. Он поддерживает структуры данных:

* строки
* списки
* наборы
* отсортированные наборы с запросами диапазона
* растровые изображения.

Поддерживает атомарные операции над типами, например:

* добавление к строке;
* увеличение значения в хэше;
* нажатие элемента на список;
* пересечение вычислительных множеств, объединение и разность;
* получение члена с наивысшим рейтингом в отсортированном наборе.

Для достижения своей производительности Redis работает с набором данных в памяти. В зависимости от варианта использования, можно сохранить либо путем временного сброса набора данных на диск, либо путем добавления каждой команды в журнал.

**Цель работы**: получить навыки работы с протоколом Redis.

1. Заходим на сайт Redis в tutorial <https://try.redis.io/> (рисунок 1).



Рисунок 3. Начальная страница Tutorial

1. Redis - это так называемое хранилище ключей, которое часто называют базой данных NoSQL. Суть хранилища ключей - возможность хранить некоторые данные, называемые значением, внутри ключа. Эти данные могут быть впоследствии получены только в том случае, если мы знаем точный ключ, используемый для его хранения. Мы можем использовать команду SET для сохранения значения «fido» в ключе «server: name» (рисунок 2).

Redis будет хранить наши данные на постоянной основе, поэтому мы можем спросить: «Какое значение хранится с ключом «server: name» и Redis ответит «fido» (рисунок 2).

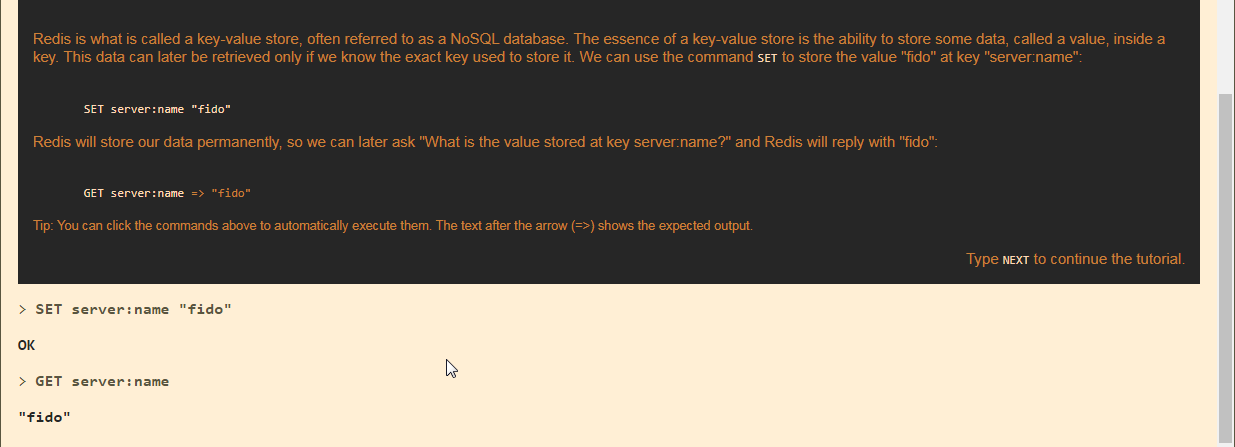


Рисунок 4. Set и Get - запросы в Redis

1. Другими распространенными операциями являются DEL для удаления заданного ключа и связанного с ним значения, SETNX, который устанавливает ключ, только если он еще не существует, и INCR для атомарного увеличения числа, сохранённого в заданном ключе. На рисунке 3 прописаны примеры запросов.

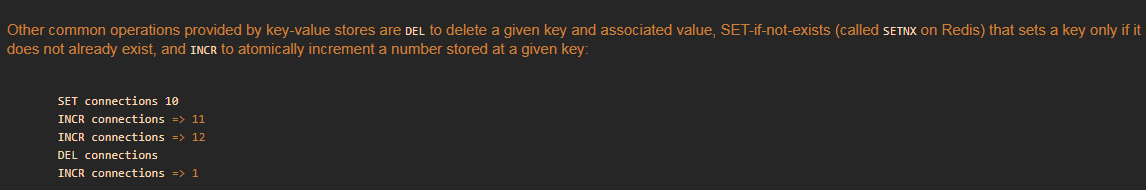


Рисунок 5. Примеры запросов INCR, DEL.

1. В Redis можно задать значение времени с помощью команды EXPIRE, в течение которого ключ может существовать. Команда TTL возвращает значение оставшегося времени (рисунок 4).

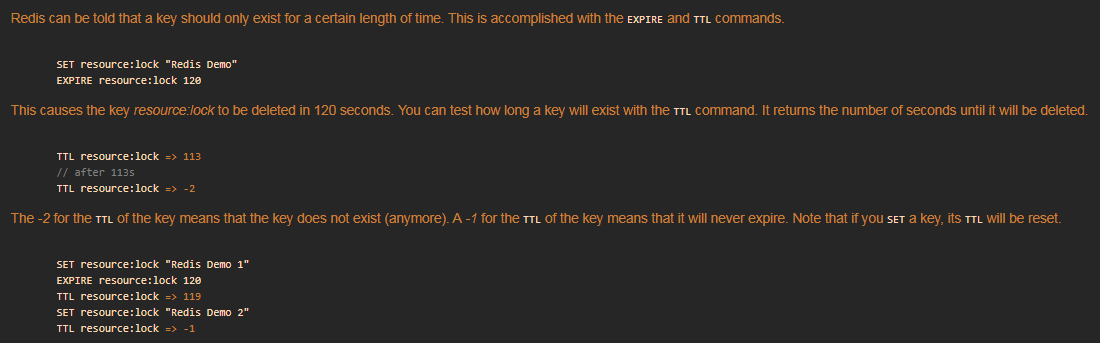


Рисунок 6. Выполнение команд EXPIRE и TTL

1. Redis также поддерживает несколько более сложных структур данных. Первый, который мы рассмотрим, - это список. Список представляет собой серию упорядоченных значений. Некоторые из важных команд для взаимодействия со списками - RPUSH, LPUSH, LLEN, LRANGE, LPOP и RPOP. Вы можете сразу начать работу с ключом в виде списка, если он еще не существует как другой тип.

* RPUSH добавляет новое значение в конец списка
* LPUSH ставит новое значение в начале списка
* LRANGE дает подмножество списка.

LRANGE принимает индекс первого элемента, который вы хотите получить в качестве его первого параметра, и индекс последнего элемента, который вы хотите получить в качестве второго параметра. Значение -1 для второго параметра означает получение элементов до конца списка (рисунок 5).

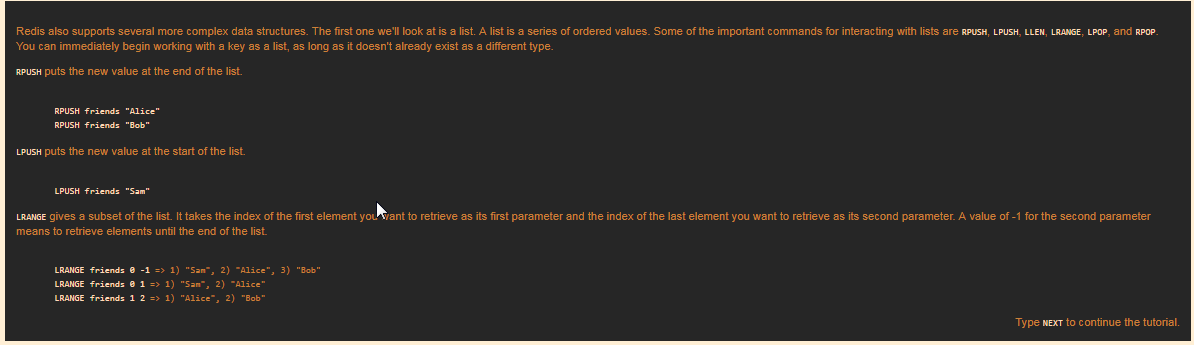


Рисунок 7. Работа со списком командами RPUSH, LPUSH, LRANGE

* LPOP удаляет первый элемент из списка и возвращает его
* RPOP удаляет последний элемент из списка и возвращает его
* LLEN возвращает текущую длину списка

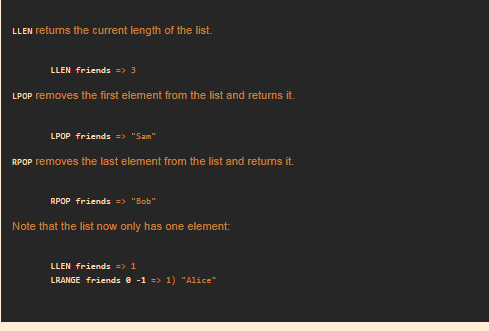


Рисунок 8. Работа со списком командами LLEN, LPOP и RPOP

1. Следующая структура данных, на которую мы будем смотреть, - это набор. Набор похож на список, за исключением того, что он не имеет определенного порядка, и каждый элемент может появляться только один раз. Некоторые из важных команд при работе с наборами - SADD, SREM, SISMEMBER, SMEMBERS и SUNION.

* SADD добавляет заданное значение в набор
* SREM удаляет заданное значение из набора

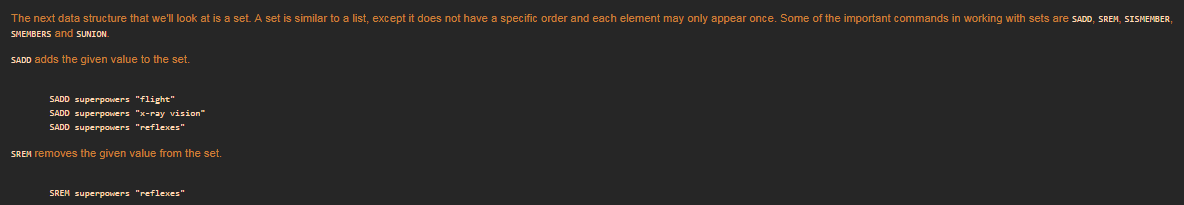


Рисунок 9. Работа с набором командами SADD, SREM

* SISMEMBER проверяет, находится ли заданное значение в наборе. Он возвращает 1, если значение есть, а 0 - нет.
* SMEMBERS возвращает список всех членов этого набора.
* SUNION объединяет два или более набора и возвращает список всех элементов.

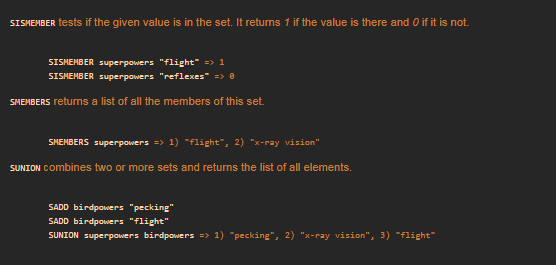


Рисунок 10. Работа с набором командами SISMEMBER, SMEMBERS, SUNION

Наборы - очень удобный тип данных, но поскольку они не отсортированы, поэтому с рядом проблем они хорошо не работают. Вот почему Redis 1.2 представил отсортированные наборы. Сортированный набор похож на обычный набор, но теперь каждое значение имеет связанный счет.

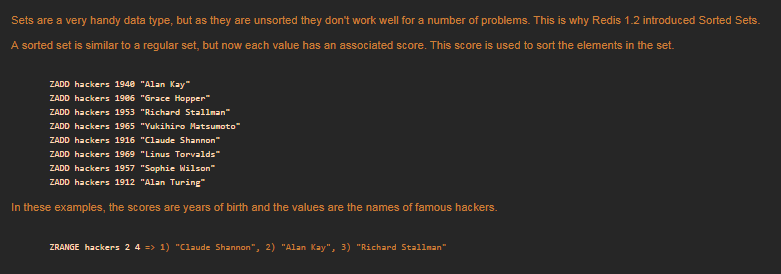


Рисунок 11. Работа с сортированными наборами

1. Хэши - это карты между строковыми полями и строковыми значениями, поэтому они являются идеальным типом данных для представления объектов (например: Пользователь с несколькими полями, такими как имя, фамилия, возраст и т. Д.).

* Чтобы вернуть сохраненные данные, используйте HGETALL
* Чтобы установить сразу несколько полей – HMSET
* Получить одно значение поля – HGET

На рисунке 10 отображена работы с хэшами.

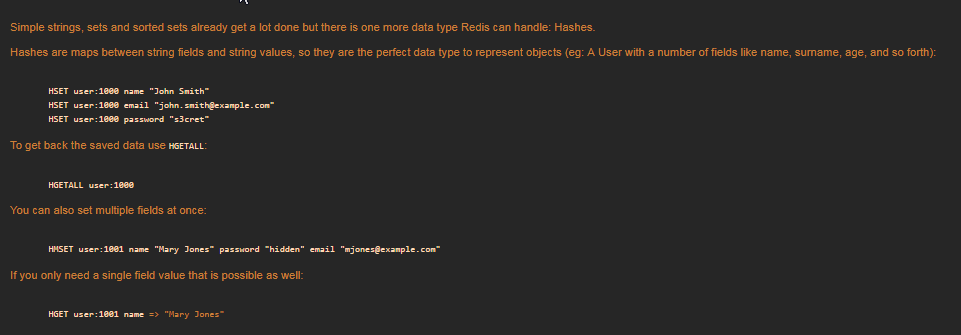


Рисунок 12. Работа с хэщ-полями

1. Числовые значения в хэш-полях обрабатываются точно так же, как и в простых строках, и есть операции для увеличения этого значения в атомном режиме (рисунок 11).

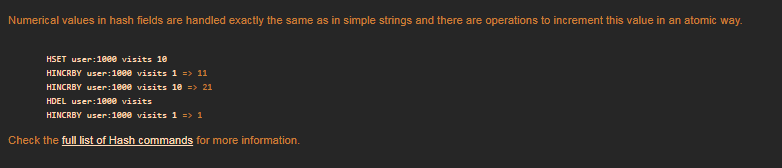


Рисунок 13. Обработка хэш-полей

## Лабораторная работа №3. Партционирование

**Партиционирование (секционирование) в PostgreSQL** — это разбиение таблиц, содержащих большое количество записей, на логические части для повышения скорости и удобства выполнения запросов.

Секционирование может принести следующую пользу:

В определённых ситуациях оно кардинально увеличивает быстродействие, особенно когда большой процент часто запрашиваемых строк таблицы относится к одной или лишь нескольким секциям. Секционирование может сыграть роль ведущих столбцов в индексах, что позволит уменьшить размер индекса и увеличит вероятность нахождения наиболее востребованных частей индексов в памяти.

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 секций. Команды ALTER TABLE NO INHERIT и DROP TABLE работают гораздо быстрее, чем массовая загрузка. Эти команды также полностью исключают накладные расходы, связанные с выполнением операции VACUUM после команды DELETE.

В настоящее время PostgreSQL реализует секционирование таблиц через механизм наследования. Каждая секция одной таблицы должна создаваться как её дочерняя таблица. Сама же родительская таблица обычно остаётся пустой; она существует только для того, чтобы представлять единый набор данных.

В PostgreSQL можно реализовать следующие типы 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.

**Цель работы**: спроектировать БД с партиционированием в PostgreSQL.

Для начала работы устанавливаем PostgreSQL следующей командой:

* ***sudo apt install postgresql postgresql-contrib***

Запускаем docker с именем контейнера some-postgres.

* ***docker run --name some-postgres -e POSTGRES\_PASSWORD=postgres -d postgres***

Запускаем наше контейнер на порту 81:80 PGAdmin 4.

* ***docker run -p 81:80 --link some-postgres -e "PGADMIN\_DEFAULT\_EMAIL=email@domain.com" -e "PGADMIN\_DEFAULT\_PASSWORD=postgres" -d dpage/pgadmin4***

Для того чтобы увидеть на деле партиционирование, создаём таблицу с предложением PARTITION BY, указав метод разбиения RANGE (по диапазонам) и список столбцов, которые будут образовывать ключ разбиения – в нашем случае, по дате (рисунок 14). При разбиении по диапазонам в качестве ключа разбиения при желании можно использовать набор из нескольких столбцов.

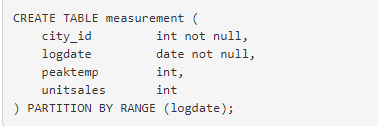


Рисунок 14. Создание секционированной таблицы

После создания таблицы заполняем её данными.

В определении каждой секции должны задаваться границы, соответствующие методу и ключу разбиения родительской таблицы. В нашем случае партиции созданы по ключу разбиения даты с первого числа месяца и до первого числа следующего месяца (рисунок 15).

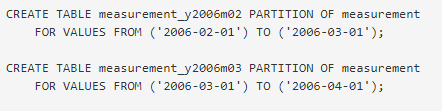


Рисунок 15. Создание секций

В БД при создании партиций в главной таблице отмечается идентификатор этой партиции. Чтоб сохранить лёгкий поиск по идентификатору в главной таблице, при удалении партиции будут удаляться не запись и её идентификатор (множество id), а партиция с её единственным id. Скрипт с удалением на рисунке 16.



Рисунок 16. Удаление партиции

# Глава 2. DDOS-атака

### Что такое DDOS-атака?

В общем случае DDOS (по-простому "дедос") - атака, во время которой злоумышленники атакуют ваш сервер (выделенный, облачный или виртуальный) большим количеством запросов, обработка которых отбирает драгоценные ресурсы у сервера, который в свою очередь не может обслужить обычных посетителей.

Все DDoS атаки можно разделить на три обширные группы:

* **Атаки, направленные на объём**

Данная категория атак направлена на насыщение полосы пропускания, соответственно, сила атаки измеряется в битах в секунду.

К этой категории разносятся различные виды флудов:

* + UDP

На случайные порты хост-машины приходит бесчисленное количество пакетов, принуждая постоянно проверять, слушает ли данный порт какое-то приложение, и в случае ошибки возвращать пакет «ICMP Destination Unreachable». Естественно, такая активность поглощает ресурсы хост-машины, приводя к её недоступности.

* + ICMP

Схож с UDP. На хост-машину с максимальной частотой посылаются ping-запросы, принуждающие её давать эхо-ответы. Инструмент проверки доступности сетевого узла становится злобным пожирателем системных ресурсов и прочие потоки сфальсифицированных пакетов.

* **Атаки на уровне протоколов**

Эта категория направлена на ограничения оборудования или уязвимости различных протоколов. Такие атаки забивают ресурсы сервера либо промежуточного оборудования (фаерволы, балансировщики нагрузки и т.п.) паразитными пакетами, в результате чего системы оказываются неспособны обрабатывать полезные. Сила атаки измеряется в пакетах в секунду.

К этой категории относятся

* + SYN-флуд

Машина, инициирующая соединение, отправляет хост-машине SYN-пакет. Хост отвечает пакетом SYN-ACK, на что машина-инициатор должна ответить ACK-пакетом. В случае SYN-флуда ACK-пакет не отправляется, в результате чего соединение некоторое время висит открытым и закрывается по тайм-ауту. Так как количество подключений, которые одновременно могут поддерживаться хост-машиной открытыми, ограничено, рано или поздно наступает насыщение, приводящее к отказу в обработке полезных пакетов.

* + «Пинг смерти»

Максимальный размер IP пакета — 65535 байт. Однако в процессе передачи по сети пакет дробится на части, соответствующие размеру окна. В результате манипуляций с получившимися субпакетами можно добиться, что при обратной «сборке» получится пакет, превышающий максимальный размер. Это может привести к переполнению выделенного буфера памяти и отказу в обслуживании для других пакетов.

* **Атаки на уровне приложений**

Как можно понять из названия, атаки направлены на уязвимости в приложениях и операционных системах (Apache, Windows, OpenBSD и т.п.). Они приводят к неработоспособности какого-либо приложения или ОС в целом. Как правило, состоящие из вполне невинных с виду запросов, такие атаки «ложат» веб-сервер. Интенсивность измеряется в запросах в секунду.

Данный тип атак наиболее «убийственный». Они чрезвычайно узко направлены, благодаря чему могут создать весьма серьёзные проблемы атакуемому при малых затратах ресурсов атакующего. За последние 3-4 года данный тип атак становится преобладающим, и простой флуд HTTP GET запросов является одним из наиболее распространённых видов.

Среди таких атак:

* + Slowloris

Атакующий сервер пытается открыть как можно больше HTTP соединений и держать их как можно дольше, понемногу отправляя частичные запросы. Лимит одновременных подключений на атакуемом сервере весьма быстро заканчивается, и он перестаёт принимать полезные запросы

* + Атаки нулевого дня

К этому типу относят атаки, которые фиксируются впервые.

### Методы защиты

1. **Отказаться от Windows Server**

Практика подсказывает, что сайт, который работает на винде (2003 или 2008 — неважно), в случае DDoS обречен. Причина неудачи кроется в виндовом сетевом стеке: когда соединений становится очень много, то сервер непременно начинает плохо отвечать.

1. **Расстаться с Apache**

Если у вас стоит Apache, то, как минимум, поставьте перед ним кеширующий прокси — nginx или lighttpd. Apache-у крайне тяжело отдавать файлы, и, что еще хуже, он на фундаментальном уровне (то есть неисправимо) уязвим для опаснейшей атаки Slowloris, позволяющей завалить сервер чуть ли не с мобильного телефона.

1. **Использовать модуль testcookie**

Если на ваш сайт приходит DDoS, то максимально действенным способом дать отпор может стать модуль testcookie-nginx. Идея простая. Чаще всего боты, реализующие HTTP-флуд, довольно тупые и не имеют механизмов HTTP cookie и редиректа. Testcookie-nginx работает как быстрый фильтр между ботами и бэкендом во время DDoS-атаки, позволяющий отсеивать мусорные запросы. Что входит в эти проверки? Умеет ли клиент выполнять HTTP Redirect, поддерживает ли JavaScript, тот ли он браузер, за который себя выдает (поскольку JavaScript везде разный и если клиент говорит, что он, скажем, Firefox, то мы можем это проверить). Проверка реализована с помощью кукисов с использованием разных методов:

1. **Код 444**

Типичный пример — поиск, который выполняет сложные запросы к базе. Естественно, этим могут воспользоваться злоумышленники, зарядив сразу несколько десятков тысяч запросов к поисковому движку. Что мы можем сделать? Временно отключить поиск. Пускай клиенты не смогут искать нужную информацию встроенными средствами, Nginx поддерживает нестандартный код 444, который позволяет просто закрыть соединение и ничего не отдавать в ответ

1. **Баним по геопризнаку**

Вы можете жестко ограничить отдельные страны, от которых испытываете неудобство. Скажем, вряд ли у интернет-магазина фотоаппаратов из Ростова-на-Дону много пользователей в Египте. Это не очень хороший способ (прямо скажем — отвратительный), поскольку данные GeoIP неточны, а ростовчане иногда летают в Египет на отдых. Но если вам терять нечего, то следуйте инструкциям::

1. Подключите к nginx GeoIP-модуль (wiki.nginx.org/HttpGeoipModule).
2. Выведите информацию о геопривязке в access log.
3. Далее, модифицировав приведенный выше шелл-скрипт, проgrepайте accesslog nginx’а и добавьте отфутболенных по географическому признаку клиентов в бан.
4. **Юзайте профайлер и отладчик**

Для наиболее распространенной платформы создания веб-сайтов — PHP + MySQL — узкое место можно искать с помощью следующих инструментов:

– профайлер Xdebug покажет, на какие вызовы приложение тратит больше всего времени;

– встроенный отладчик APD и отладочный вывод в лог ошибок помогут выяснить, какой именно код выполняет эти вызовы;

– в большинстве случаев собака зарыта в сложности и тяжеловесности запросов к базе данных. Здесь поможет встроенная в движок базы данных SQL-директива explain.

1. **Анализируйте ошибки**

Проанализируйте объем трафика, время ответа сервера, количество ошибок. Для этого смотрите логи. В nginx время ответа сервера фиксируется в логе двумя переменными: request\_time и upstream\_response\_time. Первая — это полное время выполнения запроса, включая задержки в сети между пользователем и сервером; вторая сообщает, сколько бэкенд (Apache, php\_fpm, uwsgi…) выполнял запрос. Значение upstream\_response\_time чрезвычайно важно для сайтов с большим количеством динамического контента и активным общением фронтенда с базой данных, им нельзя пренебрегать. В качестве формата лога можно использовать такой конфиг:

1. **Отслеживайте количество запросов в секунду**

Также посмотрите на число запросов в секунду. В случае nginx вы можете примерно оценить эту величину следующей shell-командой (переменная ACCESS\_LOG содержит путь к журналу запросов nginx в combined-формате):

По сравнению с нормальным для этого времени дня уровнем количество запросов в секунду может как падать, так и расти. Растут они в случае, если пришел крупный ботнет, а падают, если пришедший ботнет обрушил сайт, сделав его полностью недоступным для легитимных пользователей

1. **Лимитируем ресурсы (размеры буферов) в nginx**

Про что нужно помнить в первую очередь? Каждый ресурс имеет лимит. Прежде всего это касается оперативной памяти. Поэтому размеры заголовков и всех используемых буферов нужно ограничить адекватными значениями на клиента и на сервер целиком. Их обязательно нужно прописать в конфиге nginx.

• client\_header\_buffer\_size\_\_ Задает размер буфера для чтения заголовка запроса клиента. Если строка запроса или поле заголовка запроса не помещаются полностью в этот буфер, то выделяются буферы большего размера, задаваемые директивой large\_client\_header\_buffers.

• large\_client\_header\_buffers Задает максимальное число и размер буферов для чтения большого заголовка запроса клиента.

• client\_body\_buffer\_size Задает размер буфера для чтения тела запроса клиента. Если тело запроса больше заданного буфера, то все тело запроса или только его часть записывается во временный файл.

• client\_max\_body\_size Задает максимально допустимый размер тела запроса клиента, указываемый в поле «Content-Length» заголовка запроса. Если размер больше заданного, то клиенту возвращается ошибка 413 (Request Entity Too Large).

1. **Настраиваем тайм-ауты в nginx**

Ресурсом является и время. Поэтому следующим важным шагом должна стать установка всех тайм-аутов, которые опять же очень важно аккуратно прописать в настройках nginx.

• reset\_timedout\_connection on;

Помогает бороться с сокетами, зависшими в фазе FIN-WAIT.

• client\_header\_timeout

Задает тайм-аут при чтении заголовка запроса клиента.

• client\_body\_timeout

Задает тайм-аут при чтении тела запроса клиента.

• keepalive\_timeout

Задает тайм-аут, в течение которого keep-alive соединение с клиентом не будет закрыто со стороны сервера. Многие боятся задавать здесь крупные значения, но мы не уверены, что этот страх оправдан. Опционально можно выставить значение тайм-аута в HTTP-заголовке Keep-Alive, но Internet Explorer знаменит тем, что игнорирует это значение

• send\_timeout

Задает тайм-аут при передаче ответа клиенту. Если по истечении этого времени клиент ничего не примет, соединение будет закрыто.

• Знайте, с кем связаться

Вы должны чётко знать, к кому обратиться в случае, если Вы уже под атакой либо чувствуете её приближение. Это может быть внутренний отдел безопасности, удалённый сотрудник, инженер дата-центра, сервис сопровождения по вопросам безопасности и т.п.

• Зачем это нужно?

• недобросовестная конкуренция

• мести обиженного клиента неугодному сервису

• интернет-вандализм

• идеологический хактивизм (использование компьютеров и компьютерных сетей для продвижения политических идей, свободы слова, защиты прав человека и обеспечения свободы информации)

Хактиви́зм — использование компьютеров и компьютерных сетей для продвижения политических идей, свободы слова, защиты прав человека и обеспечения свободы информации

### Как устроить DDOS-атаку

* Приложения для проведения DDoS-атак

Безмерно доступно. На сегодняшний день существует немало свободно доступных в интернете приложений для проведения DDoS атак. Некоторые из них используют механизмы атак, которым сложно противодействовать, другие позволяют объединить всех пользователей в добровольный ботнет, что даёт возможность пользоваться чужими ресурсами для проведения атак и раздавать взамен свои. При этом таким на первый взгляд аматорским атакам бывает сложно противостоять даже хорошо подготовленным коммерческим системам.

* Аренда ресурсов ботнета: 5$ за час, от 40$ за сутки

Другой способ — аренда ресурсов ботнета. Интернет полон ресурсов, предоставляющих подобные услуги по весьма символическим ценам: от $5 за час, от $40 за сутки. За такие, можно сказать, смешные деньги запросто можно «заказать» своих конкурентов по электронной коммерции и принести им намного более существенные убытки.

# Использованные источники

1. <https://web-creator.ru/articles/partitioning_replication_sharding>
2. <https://postgrespro.ru/docs/postgresql/9.6/ddl-partitioning>
3. <https://redis.io/topics/introduction>
4. <https://try.redis.io/>
5. <https://postgrespro.ru/docs/postgresql/10/ddl-partitioning>