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

Власова Екатерина Александровна

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

*(подпись)*

Принял:

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

*(должность, ФИО руководителя)*

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

*(оценка) (подпись)*

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

*(дата)*

**Пермь 2018**

**Содержание**

[Лабораторная работа №1 3](#_Toc532789993)

[Лабораторная работа №2 6](#_Toc532789994)

[Лабораторная работа №3 13](#_Toc532789995)

[Docker (устройство, swarm) 17](#_Toc532789996)

# **Лабораторная работа №1**

**Задание:** создать веб-приложение и разместить его в docker image. Запустить nginx как балансировщик перед приложением. Приложение запустить в нескольких экземплярах и в приложении считать запросы от балансировщика.

**Цель:** 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-nginx/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.

Используя функцию томов данных Docker, мы можем создать символическую связь между файловой системой Droplet и файловой системой контейнера. Это позволяет нам редактировать существующие файлы веб-страниц и добавлять новые в каталог, и контейнер будет автоматически получать к ним доступ.

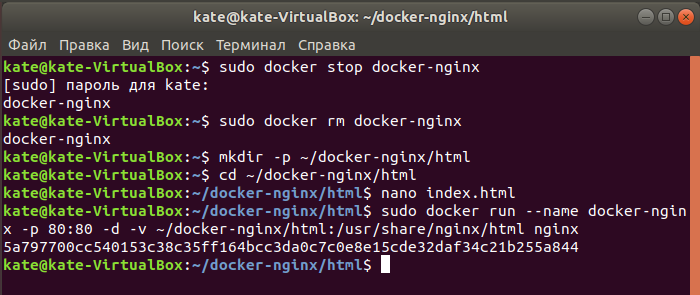
Контейнер Nginx по умолчанию настроен на поиск индексной страницы /usr/share/nginx/html, поэтому в новом контейнере Docker нужно предоставить ему доступ к файлам в этом месте.

Создание ссылки:

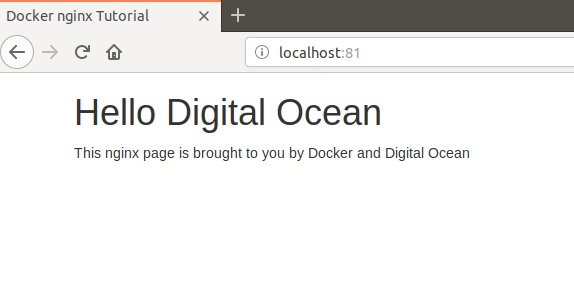
Для этого мы используем –v флаг для сопоставления папки с нашей локальной машины ( ~/docker-nginx/html) с относительным путем в контейнере ( /usr/share/nginx/html). Выполняем следующую команду:

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

* новым дополнением к команде -v ~/docker-nginx/html:/usr/share/nginx/html является ссылка на объем;
* -v указывает, что мы связываем том;
* часть слева от: местоположения каталога / каталога на виртуальной машине ( ~/docker-nginx/html);
* часть справа от: местоположения, с которым мы связываемся в контейнере ( /usr/share/nginx/html).



Можно загрузить больше контента в ~/docker-nginx/html/каталог, и он будет добавлен в веб-приложение. Если изменить индексный файл, и если перезагрузить окно браузера, то можно увидеть обновление файла в реальном времени.



# **Лабораторная работа №2**

**Задание:** Реализовать веб-приложение с минимальным пользовательским интерфейсом, которое загружает\удаляет объекты в redis. На практике пройти официальный tutorial по redis.

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

**Ход работы:**

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).

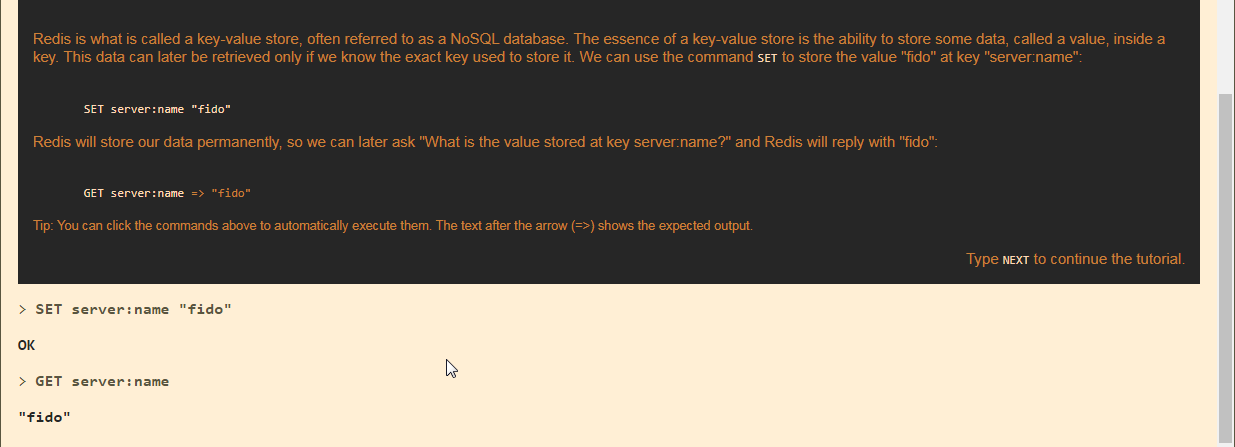


Рисунок 4. Set и Get - запросы в Redis

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.

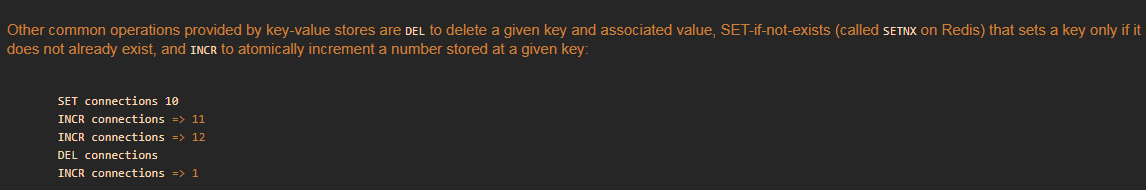


Рисунок 5. Примеры запросов INCR, DEL.

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).

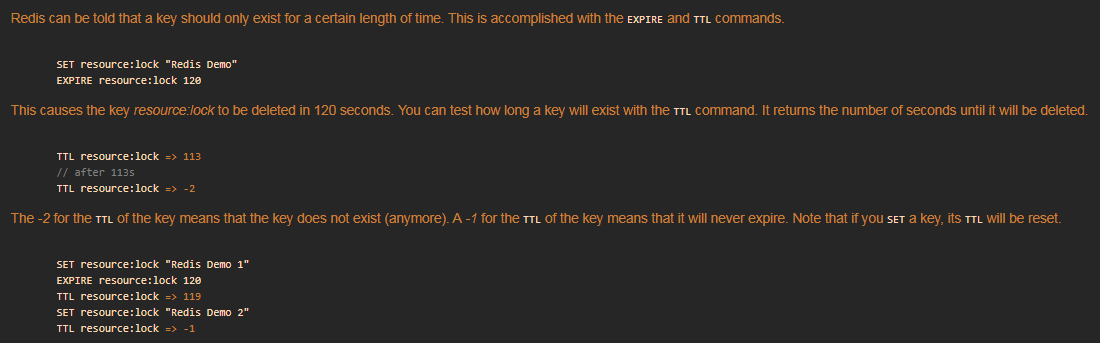


Рисунок 6. Выполнение команд EXPIRE и TTL

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).

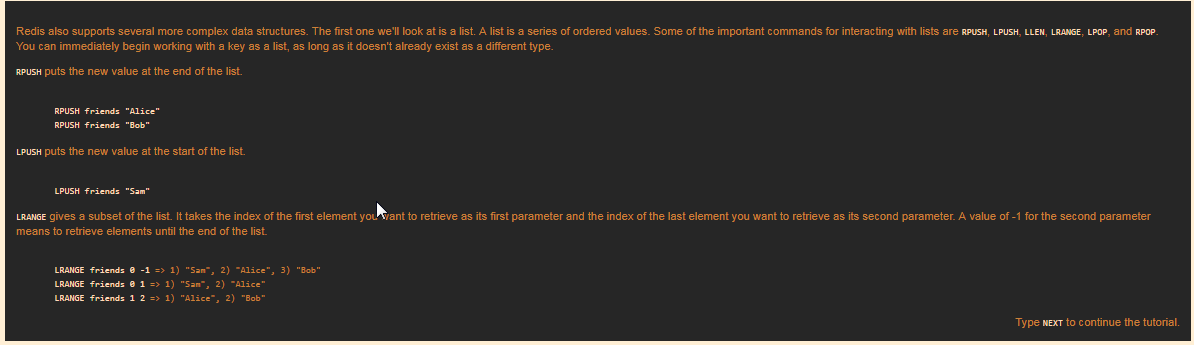


Рисунок 7. Работа со списком командами RPUSH, LPUSH, LRANGE

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

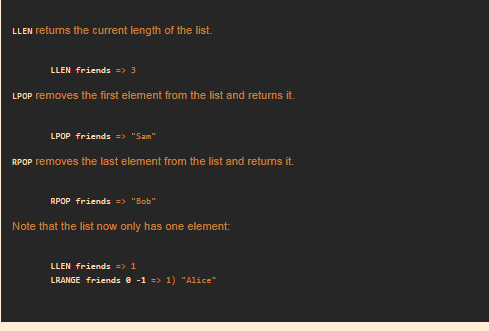


Рисунок 8. Работа со списком командами LLEN, LPOP и RPOP

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

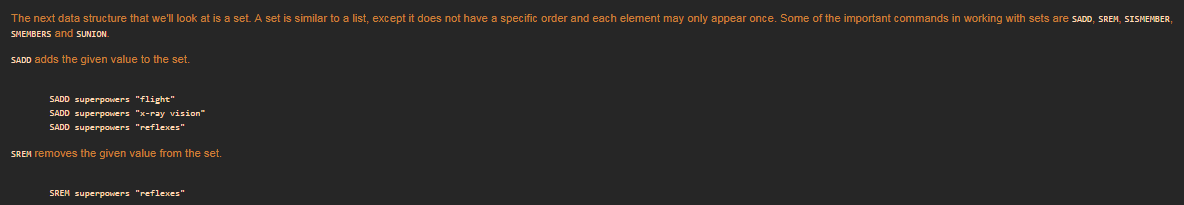


Рисунок 9. Работа с набором командами SADD, SREM

* 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.

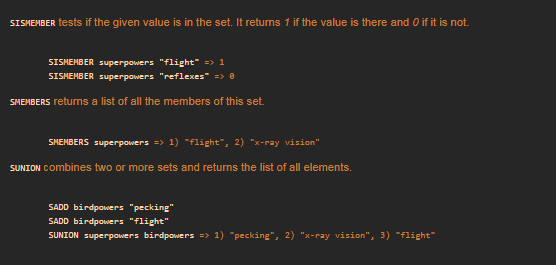


Рисунок 10. Работа с набором командами SISMEMBER, SMEMBERS, SUNION

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.

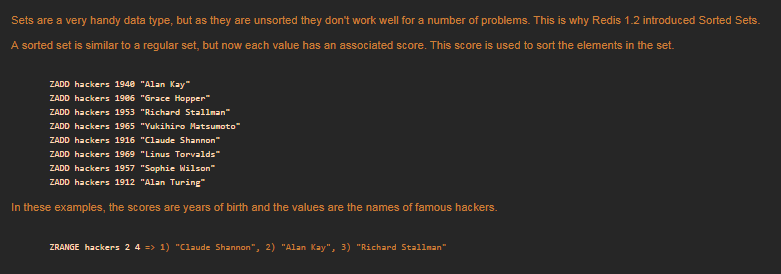


Рисунок 11. Работа с сортированными наборами

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.

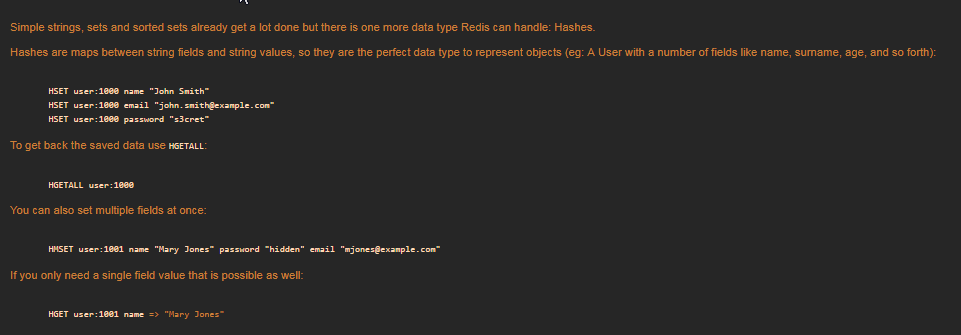


Рисунок 12. Работа с хэщ-полями

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).

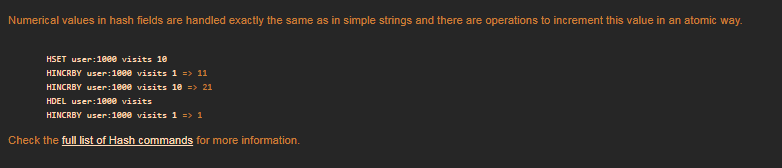


Рисунок 13. Обработка хэш-полей

# **Лабораторная работа №3**

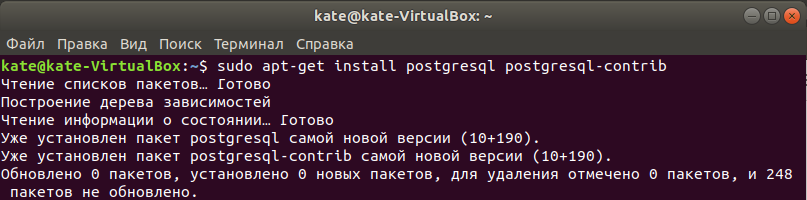
**Задание:** спроектировать БД с партиционированием в postgres.

**Цель:** получить навыки проектирования БД с партиционированием.

**Ход работы:**

1. Запускаем docker:

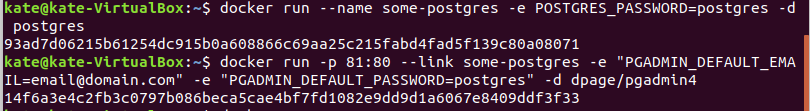
*$sudo apt-get install postgresql postgresql-contrib*

*~~~~*

1. Запускаем образы:

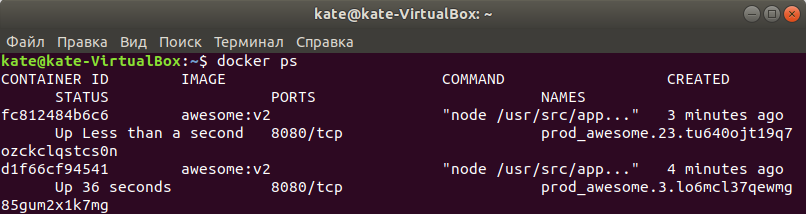
*$docker run --name some-postgres –e POSTGRES\_PASSWORD=postgres –d postgres*

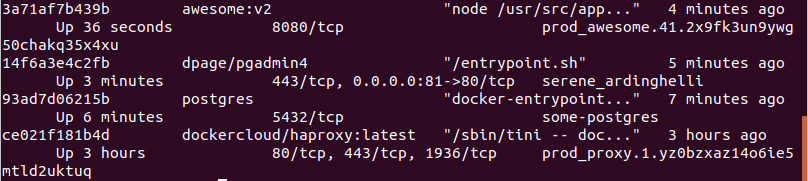
*$doker run –p 81:80 --link some-postgres –e “PGADMIN\_DEFAULT\_EMAIL=email@domain.com” –e “PGADMIN\_DEFAULT\_PASSWORD=postgres” –d dpage/pgadmin4*

**

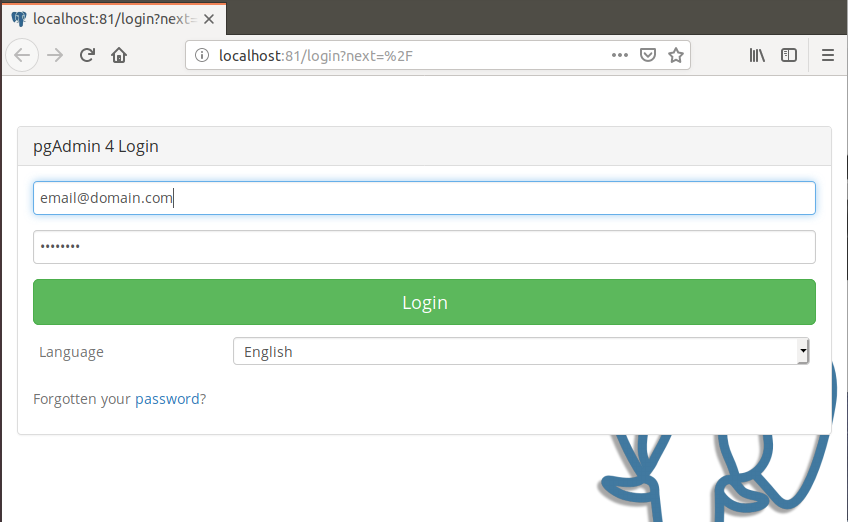
1. Проверяем запущенные контейнеры:

*$docker ps*

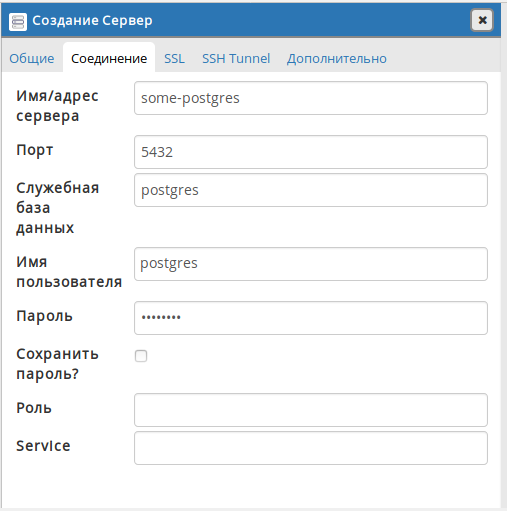
**

**

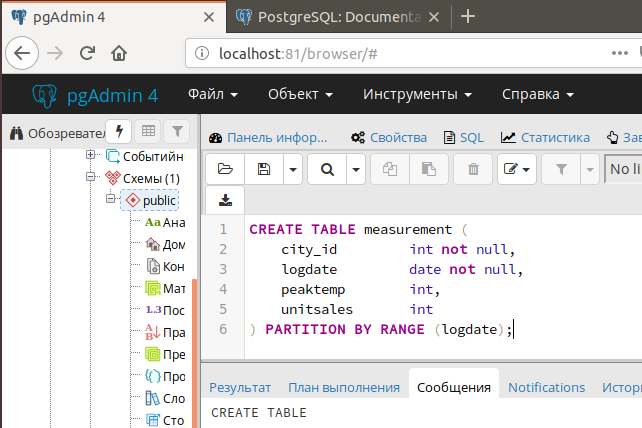
1. Переходим на localhost:81:



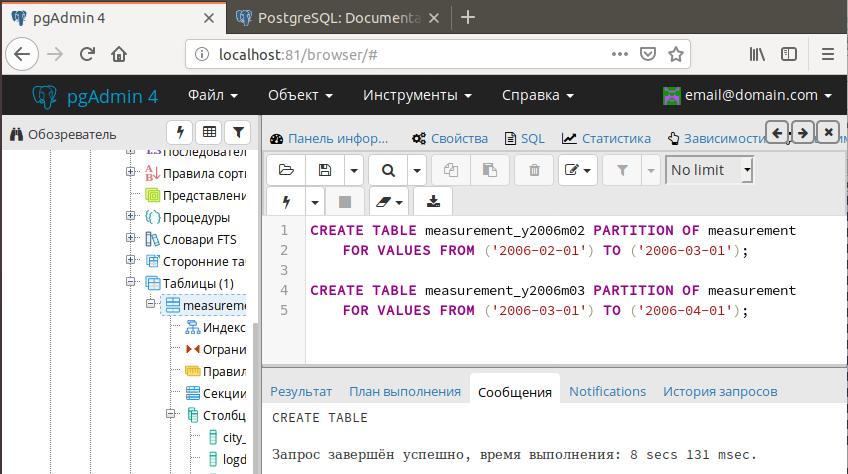
1. Создаем сервер:



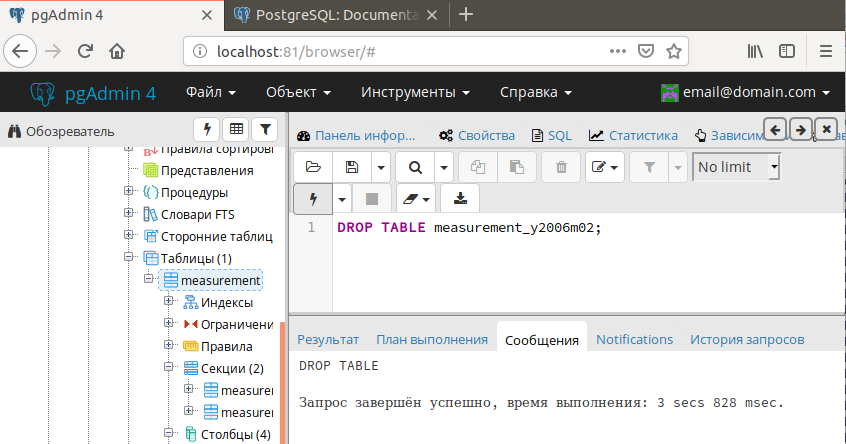
1. Чтобы использовать декларативное разбиение создаем  таблицу measurement - как разделенную таблицу, указав PARTITION BY предложение, которое включает метод разделения и список столбцов, которые будут использоваться в качестве ключа разделения.

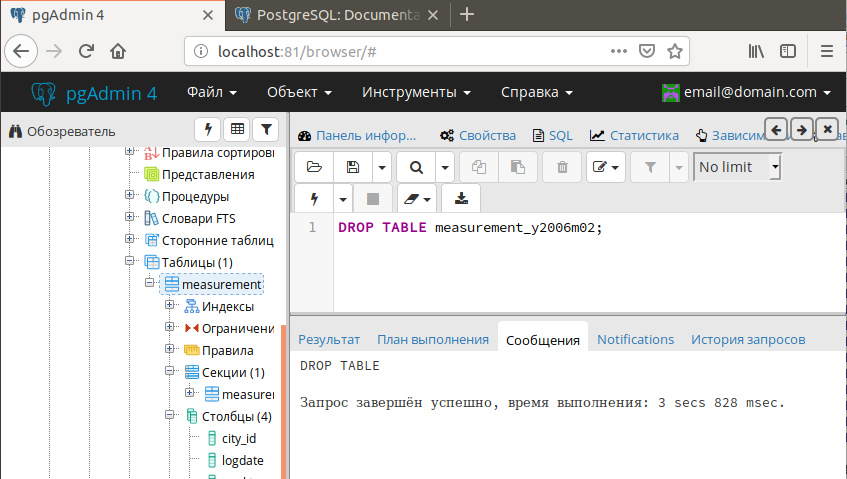


1. Создаем разделы. Определение каждого раздела должно указывать границы, которые соответствуют методу разделения и ключу раздела родителя.

**

1. Для удаления старых данных просто удаляем ненужный раздел:

****

****

# **Docker (устройство, swarm)**

Docker - программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализация на уровне операционной системы. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами.

Docker удобно использовать для решения следующих задач:

* Развертывания легко переносимых веб-приложений, баз данных и сервисов.
* Запуска программного обеспечения, требующего разных дистрибутивов GNU/Linux.
* Создания легковесных PaaS окружений.

**Компоненты:**

1. Образы. Docker запускает контейнер из файлов специального формата, называемых «образами». При запуске Docker создает в ОЗУ файловую систему bootfs, которая обеспечивает монтирование из образа корневой файловая системы rootfs. Файловая система rootfs монтируется «только для чтения». После запуска контейнера, поверх этой файловой системы монтируется файловая система на запись.
2. Реестры. Docker хранит образы в реестрах. Существует два типа реестров: публичные, приватные. Официальный реестр называется  Docker Hub. Создав в нем учетную запись, можно сохранять свои образы и делиться ими с другими пользователями.
3. Контейнеры. Docker создаёт контейнеры (изолированные окружения), внутри которых можно запускать приложения и сервисы. Файловая система контейнера состоит из двух «слоёв»:

-слой, содержащий файловую систему из образа (монтируется в режиме «только для чтения»);

-слой, хранящий все изменения в процессе работы контейнера.

После запуска нового контейнера, второй слой пуст. Данные, накапливающиеся во втором слое в процессе работы контейнера можно объединить с данными первого слоя в новом образе с помощью операции commit.

1. Пакет программ docker. ПО для организации и управления изолированными окружениями.
2. Docker-compose. ПО для организации группы связанных между собой контейнеров и управления ими.
3. Docker-machine. Кроссплатформенная утилита для создания виртуальной машины с ядром Linux и системой Docker.
4. Docker Swarm. Одиночный Docker Engine запускает контейнеры на локалохосте, из образов, либо собранных тоже на локалхосте, либо выкачанных с Docker Hub. Docker Swarm — это кластер Docker. Множество хостов, на каждом из которых запущен отдельный Docker Engine, объединены под общим управлением, и выглядят как один большой Docker. Хосты могут добавляться в кластер, и вычислительная ёмкость вашего большого Docker будет тоже увеличена. Поднять свою стаю Docker довольно легко. Главный момент: компоненты кластера — это тоже Docker контейнеры

**Работа с контейнерами:**

Вывод списка контейнеров:

*docker ps* -выводит список запущенных контейнеров

*docker ps -a* - список всех контейнеров, включая остановленные.

Выводимый список представляет собой таблицу, состоящую из следующих столбцов:

*CONTAINER ID– уникальный идентификатор*

*IMAGE– имя образа в реестре*

*COMMAND– стартовая команда (запускается при старте контейнера)*

*CREATED– дата создания образа*

*STATUS– состояние (работает,остановлен,приостановлен …)*

*PORTS– доступные порты TCP или UDP*

*NAMES– имя*

Созданиеи запуск контейнера: *docker create <опции> <образ> <команда>* создает, но не запускает контейнер.

*docker run <опции> <образ> <команда>* создает, и запускает контейнер.

Получение базовой информации о контейнере: *docker info <контейнер>*

Получение полной информации о контейнере: *docker inspect <контейнер>*

Запуск контейнера: *docker start <контейнер>*

Остановка контейнера: *docker stop <контейнер>*

Подключение текущего терминала к запущенному контейнеру: *docker attach <контейнер>*

Удаление контейнера: *docker rm <контейнера>*

Остановка все процессов во всех контейнерах и их удаление: *docker rm $(docker kill $(docker ps -q))*

или:

*docker rm -f $(docker ps -a -q)*

*docker rm -f $(docker ps -a -q --filter 'exited=0')*

или:

*docker ps -a | grep Exit | awk '{print $1}' | xargs docker rm*

**Обмен данными:**

Копирование данных в контейнер: *docker cp <откуда> <контейнера>:<куда>*

Копирование данных из контейнера: *docker cp <контейнера>:<откуда> <куда>*

«Проброс» каталога с хост-системы в контейнер: *docker run <опции> -v <откуда>:<куда> <образа>*

где <откуда> – путь к каталогу на хост-системе, а <куда> – путь к каталогу в контейнере.

**Работа с образами:**

Поиск образа по ключевому слову в интернет репозитории: *docker search <шаблон для поиска>*

Вывод списка всех локальных образов: *docker [-t] images*

Ключ -t, показывает иерархию наследования доступных образов.

Клонирование образа из интернет репозитория без создания контейнера: *sudo docker pull <образ>*

Удаление образа: *docker rmi <имя\_образа>*

Удаление всех образов: *docker rmi $(docker images -q)*

Запись изменений, сделанных во время работы контейнера <контейнер> в новый образ <образ>: *docker commit <контейнер> <образа>*

Сохранение образа в виде файла: *docker save имя\_образа > <файл>.tar*

**Примеры использования контейнеров. Интерактивный контейнер:**

Создание нового контейнера из образа ubuntu и запуск в нем оболочки bash: *docker run -i -t ubuntu /bin/bash*

где: -i -оставляет STDIN открытым;

-t -назначает псевдо-tty для интерактивной связи с контейнером;

/bin/bash -выполняемая в контейнере команда.

После выполнения этой команды появляется приглашение командной строки: *[root@<компьютер> ~]*

Выход из контейнера: *нажатие комбинации клавиш:Ctrl+d;*

выполнение команды: *exit*

**Примеры использования контейнеров. Контейнер-сервис:**

Создание контейнера при запуске которого в бесконечном цикле на стандартное устройство вывода STDOUT передается сообщение «Я работаю»:

*docker run -d ubuntu /bin/bash -c 'while true; do echo "Я работаю"; sleep 1; done‘*

-d запускать в режиме daemon;

-c опция относится к bassh.

**Пример создания контейнера с http-сервисом:**

Создание чистого контейнера с именем nginx с открытыми 80 и 443 портами:

*docker run -i -t -p 80:80 -p 443:443 --name nginx ubuntu*

В результате запускается командная строка внутри контейнера. Следующие команды выполняются в командной строке созданного контейнера.

Добавление репозитория стабильной версии nginx:

*deb http://nginx.org/packages/ubuntu/ trusty nginx*

*deb-src http://nginx.org/packages/ubuntu/ trusty nginx*

Установка nginx:

*apt-key update*

*apt-get update*

*apt-get install nginx*

Проверка:

- запуск сервиса (выполняется в контейнере)

*/etc/init.d/nginx start*

- соединение с контейнером по http с помощью браузера (выполняется на хост-машине):

Сохранение контейнера в репозиторий пользователя <пользователь> под именем nginx:

*docker commit nginx <пользователь>/nginx*

Для выполнения последней операции необходимо создать учетную запись на сервисе Docker Hub и подключиться к этому сервису с помощью команды: *docker login*

Сохранение образа <пользователь>/nginx на Docker Hub для повторного использования на других серверах:

*docker push <пользователь>/nginx*