ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет прикладной математики и механики

Кафедра вычислительной математики и механики

Реферат

Тема: “Разработка высоконагруженных веб-приложений”

Группа: ИТСИ-17-1м

Студент: Ермакова О.А.

Преподаватель: Истомин Д.А.

Пермь 2018

# 

Содержание

[Часть 1 3](#_Toc535340217)

[1. Docker, nginx 3](#_Toc535340218)

[2. Redis 4](#_Toc535340219)

[3. Партиционирование в PostgreSQL 5](#_Toc535340220)

[Часть 2 6](#_Toc535340221)

[1. Файловые системы и организация хранилища (raid, zfs) 6](#_Toc535340222)

# Часть 1 - Лабораторные работы

# Лабораторная работа №1 - Docker, nginx

Docker – это программная платформа для быстрой сборки, отладки и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются контейнерами. Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker пользователи могут быстро развертывать и масштабировать свои приложения в любой среде и сохранять уверенность в том, что код будет работать.

В основе работы Docker лежит стандартизированный способ исполнения кода. Docker – это операционная система для контейнеров. Подобно тому как виртуальная машина создает виртуальное представление аппаратного обеспечения сервера (то есть устраняет необходимость непосредственно управлять таковым), контейнеры создают виртуальное представление серверной операционной системы. После установки на каждый сервер Docker предоставляет доступ к простым командам, необходимым для сборки, запуска или остановки контейнеров.

Использование Docker позволяет быстрее и эффективнее доставлять или перемещать код, стандартизирует выполняемые приложениями операции и в целом экономит средства, оптимизируя использование ресурсов. Благодаря Docker пользователи получают объект, который с высокой надежностью можно запускать на любой платформе. Простой синтаксис Docker предоставляет пользователям полный контроль над выполняемыми операциями. Повсеместное внедрение контейнеров подразумевает доступ к разнообразным инструментам и готовым приложениям, которые можно использовать с Docker.

NGINX — программное обеспечение, написанное для UNIX-систем. Основное назначение — самостоятельный HTTP-сервер, или, как его используют чаще, фронтенд для высоконагруженных проектов. Возможно использование NGINX как почтового SMTP/IMAP/POP3-сервера, а также обратного TCP прокси-сервера.

NGINX является широко используемым продуктом в мире IT, по популярности уступая лишь Apache. Как правило, его используют либо как самостоятельный HTTP-сервер, используя в бекенде PHP-FPM, либо в связке с Apache, где NGINX используется во фронтэнде как кеширующий сервер, принимая на себя основную нагрузку, отдавая статику из кеша, обрабатывая и отфильтровывая входящие запросы от клиента и отправляя их дальше к Apache. Apache работает в бекэнде, работая уже с динамической составляющей проекта, собирая страницу для передачи её в кеш NGINX и запрашивающему её клиенту. Это если в общих чертах, чтобы понимать суть работы, так-то внутри всё сложнее.

# Лабораторная работа №2 - Redis

Redis (англ. remote dictionary server) — сетевое журналируемое хранилище данных типа «ключ — значение» с открытым исходным кодом. Нереляционная высокопроизводительная СУБД.

Redis поддерживает репликацию типа master-slave. Данные с любого сервера Redis могут реплицироваться произвольное количество раз. Репликация полезна для масштабирования чтения (но не записи) или при очень больших объёмах данных. Все данные, которые попадают на один узел Redis (который называется master), будут попадать также на другие узлы (называются slave). Для конфигурирования slave-узлов можно изменить опцию slaveof или аналогичную по написанию команду (узлы, запущенные без подобных опций, являются master-узлами).

Репликация помогает защитить данные, копируя их на другие сервера. Репликация также может быть использована для увеличения производительности, так как запросы на чтение могут обслуживаться slave-узлами. Эти узлы могут ответить слегка устаревшими данными, но для большинства приложений это приемлемо.

Система репликации Redis ещё не поддерживает автоматическую отказоустойчивость. Если master-узел выходит из строя, необходимо вручную выбрать новый master из списка slave-узлов. Необходимо использовать Redis Sentinel для мониторинга и автоматического переключения master-узлов, если необходима устойчивая к сбоям система.

Алгоритм реализации приложения:

1. Создание приложения:

var redis = require('redis'); // подключаем Redis к файлу   
var redisClient = redis.createClient({host : 'localhost', port : 6379});  
  
redisClient.on('ready',function() {   
console.log("Redis is ready"); // если Redis подключен  
});  
  
redisClient.on('error',function() {  
console.log("Error in Redis"); // если не подключился  
});  
  
redisClient.hmset('frameworks', 'javascript', 'AngularJS', 'css', 'Bootstrap', 'node', 'Express'); // устанавливаем хеш-поля и значения к ним  
  
redisClient.hgetall('frameworks', function(err, object) {  
console.log(object); // вывод в консоли всех полей и значений в кэше  
});   
  
redisClient.set("language","nodejs"); // устанавливаем ключ-значение  
  
redisClient.get("language",function(err,reply) {  
console.log(reply); // вывод в консоли значение ключа   
});  
  
redisClient.exists('language',function(err,reply) { //проверка наличия уже существующих ключей в Redis   
if(!err) {  
if(reply === 1) {  
console.log("Key exists");  
} else {  
console.log("Does't exists");  
}  
}  
});  
  
redisClient.del('redisClient',function(err,reply) { //удаление ключей из Redis  
if(!err) {  
if(reply === 1) {  
console.log("Key is deleted");  
} else {  
console.log("Does't exists");  
}  
}  
});

1. Устанавливаем связь между приложением и redis.

npm init

1. После команды npm init создается файл package.json, который содержит в себе информацию о приложении.
2. Запускаем приложение с помощью команды

# Лабораторная работа №3 - Партиционирование в PostgreSQL

**Партиционирование** (partitioning) — **это** разбиение больших таблиц на логические части по выбранным критериям.

Алгоритм выполнения:

1. Создание таблицы:

CREATE TABLE measurement (

city\_id int not null,

logdate date not null,

peaktemp int,

unitsales int

) PARTITION BY RANGE (logdate);

CREATE TABLE measurement\_y2006m02 PARTITION OF measurement

FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');

CREATE TABLE measurement\_y2006m03 PARTITION OF measurement

FOR VALUES FROM ('2006-03-01') TO ('2006-04-01');

1. Создание индексов:

CREATE INDEX ON measurement\_y2006m02 (logdate);

CREATE INDEX ON measurement\_y2006m03 (logdate);

1. **Вывод значений:**

INSERT INTO measurement

(city\_id, logdate, peaktemp, unitsales)

VALUES

(1, '2006-03-02', 12, 234);

INSERT INTO measurement

(city\_id, logdate, peaktemp, unitsales)

VALUES

(1, '2006-03-04', 12, 234);

INSERT INTO measurement

(city\_id, logdate, peaktemp, unitsales)

VALUES

(1, '2006-02-02', 12, 9999);

INSERT INTO measurement

(city\_id, logdate, peaktemp, unitsales)

VALUES

(1, '2006-02-10', 12, 8888);

INSERT INTO measurement

(city\_id, logdate, peaktemp, unitsales)

VALUES

(1, '2006-02-12', 12, 7777);

1. **Удаление значений:**

DROP TABLE measurement\_y2006m02;

1. **Выделить раздел и сохранить как отдельную таблицу:**

ALTER TABLE measurement DETACH PARTITION measurement\_y2006m02;

# Часть 2 - Индивидуальная тема для изучения

# Файловые системы и организация хранилища (raid, zfs)

**Что такое RAID и зачем оно нужно**

**RAID** — это дисковый массив из нескольких устройств (жестких дисков). Этот массив служит для повышения надёжности хранения данных и/или для повышения скорости чтения/записи информации.

Собственно, то чем именно занимается оная связка из дисков, – ускорением работы или повышением безопасности данных, зависит от Вас, а точнее, от выбора текущей конфигурации рейда(ов). Разные типы этих конфигураций как раз и отмечаются разными номерами, – **1, 2, 3, 4** и пр, – и выполняют разные функции.

**RAID 0**

Данные разбиваются на блоки, и каждый блок записывает/считывает информацию на отдельный диск поочередно. Если из строя выходит один из дисков, то полностью и безвозвратно пропадает вся информация. RAID 0 дает ускорение в работе из-за чередования чтения/записи. RAID 0 часто используют для размещения временных файлов.

*Применение: Графика, видео*

**RAID 1**

Зеркалирование дисков. На одном диске хранятся данные, на другом – его копия.

*Применение: малые файл-серверы*

**RAID 2**

Применяется код Хемминга. Диски делятся на два типа: один под данные, другой под коды коррекции ошибок. Если диск с данными выходит из строя, то всю информацию можно восстановить со второго диска.

*Применение: мейнфреймы*

**RAID 3**

Данные разбиваются на куски (обычно не больше 1024 байт) и распределяется по дискам. Один диск используют для хранения блоков четности. Нет коррекции ошибок в отличии от RAID2.

*Применение: Графика, видео*

**RAID 4**

Отличается от RAID 3 тем, что данные делятся не на байты, а на блоки. Из-за этого скорость немного увеличивается.

*Применение: файл-серверы*

**RAID 5**

Блоки данных и контрольные суммы циклически записываются на все диски. Под контрольными суммами здесь понимается операция XOR. Этот метод обеспечивает отказоустойчивость. Результаты XOR хранятся на отдельном диске. Минимальное количество дисков 3.

*Применение: серверы баз данных*

**RAID 6**

Похож на RAID 5, но имеет более высокую степень надёжности — три диска данных и два диска контроля чётности. Основан на [кодах Рида — Соломона](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A0%D0%B8%D0%B4%D0%B0_%E2%80%94_%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0) и обеспечивает работоспособность после одновременного выхода из строя любых двух дисков.

*Применение: используется крайне редко*

**RAID 7**

Для понимания архитектуры RAID 7 рассмотрим ее особенности: 

|  |  |  |
| --- | --- | --- |
| **1.** |  | Все запросы на передачу данных обрабатываются асинхронно и независимо. |
| **2.** |  | Все операции чтения/записи кэшируются через высокоскоростную шину x-bus. |
| **3.** |  | Диск четности может быть размещен на любом канале. |
| **4.** |  | В микропроцессоре контроллера массива используется операционная система реального времени ориентированная на обработку процессов. |
| **5.** |  | Система имеет хорошую масштабируемость: до 12-ти host-интерфейсов, и до 48-ми дисков. |
| **6.** |  | Операционная система контролирует коммуникационные каналы. |
| **7.** |  | Используются стандартные SCSI диски, шины, материнские платы и модули памяти |
| **8.** |  | Используется высокоскоростная шина X-bus для работы с внутренней кэш-памятью. |
| **9.** |  | Процедура генерации четности интегрирована в кэш. |
| **10.** |  | Диски, присоединенные к системе, могут быть задекларированы как отдельно стоящие. |
| **11.** |  | Для управления и мониторинга системы можно использовать SNMP агент. |

**Комбинированные RAID-массивы**

Под RAID 1+0 имеют в виду вариант RAID 10, когда два RAID 1 объединяются в RAID 0. Вариант, когда два RAID 0 объединяются в RAID 1 называется RAID 0+1, и "снаружи" представляет собой тот же RAID 10.

**Реализация RAID систем**

* программная (software-based);
* аппаратная — шинно-ориентированная (bus-based);
* аппаратная — автономная подсистема (subsystem-based).

Главное преимущество программной реализации — низкая стоимость. Но при этом у нее много недостатков: низкая производительность, загрузка дополнительной работой центрального процессора, увеличение шинного трафика. Программно обычно реализуют простые уровни RAID — 0 и 1, так как они не требуют значительных вычислений. Учитывая эти особенности, RAID системы с программной реализацией используются в серверах начального уровня.

Аппаратные реализации RAID соответственно стоят больше чем программные, так как используют дополнительную аппаратуру для выполнения операций ввода вывода. При этом они разгружают или освобождают центральный процессор и системную шину и соответственно позволяют увеличить быстродействие.

Шинно-ориентированные реализации представляют собой RAID контроллеры, которые используют скоростную шину компьютера, в который они устанавливаются (в последнее время обычно используется шина PCI).

Учитывая то, что шинно-ориентированные реализации подключаются прямо к внутренней PCI шине компьютера, они являются наиболее производительными среди рассматриваемых систем (при организации одно-хостовых систем). Максимальное быстродействие таких систем может достигать 132 Мбайт/с (32bit PCI) или же 264 Мбайт/с (64bit PCI) при частоте шины 33MHz.

**ZFS (Zettabyte File System)**

Для создания отказоустойчивого и масштабируемого хранилища данных используют технологии RAID массива, а у файловой системы ZFS имеется свой инструмент для работы с физическими дисками и организации их в **RAID-Z**массивы (аналог RAID5). При этом, в отличие от аналогичных технологий, данная FS самостоятельно восстанавливает поврежденные блоки и исправляет их на лету без вмешательства пользователя.

К основным плюсам данной FS — практически неограниченное дисковое пространство и встроенные инструменты построения отказоустойчивых RAID-Z массивов, относится также и **атомарная запись данных**. Под этим термином понимается следующее — данные либо полностью записываются на диск, либо не записываются вовсе. Такая технология позволяет избежать проблем, которые возникают в случаях сбоев работы сети, подсистемы питания и отказа операционной системы. Поддерживает ОС: Solaris, OpenSolaris, Apple Mac OS X 10.5, FreeBSD, Linux.

**Существует несколько разновидностей RAID-Z массивов, наиболее популярны raid-z1 и raid-z2:**

— raid-z1 — здесь используется для контроля четности один диск из пула (минимум дисков для организации данного вида массива — 3 шт). При выходе из строя одного диска, массив будет работать корректно, при его замене массив перестроится самостоятельно. При выходе из строя двух дисков — массив разрушается и данные восстановлению не подлежат.

— raid-z2 — в данном случае для контроля четности выделяется 2 диска (минимум дисков для такой конфигурации — 5 шт.). Эта система является более отказоустойчивой.

**Достоинства ZFS:**

* встроенные инструменты для работы с разделами HDD и организации RAID-Z
* нет привязки к оборудованию
* нечувствительна к незапланированным отключениям электропитания
* автоматическая подмена вышедших из строя HDD, исправление ошибок и перестроение RAID
* поддерживаются огромные размеры томов, файлов, пулов, а также легкая масштабируемость хранилища
* быстрое и удобное администрирование ZFS Pool’ов
* при увеличении HDD повышается производительность хранилища
* дедупликация и сжатие данных

**Недостатки:**

* высокие требования к ресурсам CPU и RAM
* хранилища, используемые для важных корпоративных данных, должны быть построены при использовании ECC RAM.