МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

Сравнение микросервисной и монолитной архитектур на примере приложения, подсчитывающего сетевой трафик

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студента 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Маленова Владислава Сергеевича

Научный руководитель:
Старший преподаватель _____ М.С. Портенко

подпись, дата

Зав. кафедрой:
к.ф.-м.н., доцент _____ М.В. Огнева

подпись, дата

Саратов 2020

ВВЕДЕНИЕ

Актуальность темы. В современном мире при разработке приложений часто возникает ситуация, в которой уже на этапе разработки меняются требования к программному обеспечению. Также требуется постоянный прирост функциональности в уже существующем программном обеспечении и устанавливаются крайне высокие требования по производительности и отказоустойчивости.

Количество высоконагруженных интернет-сервисов постоянно растет. Это объясняется тем, что как любая фирма или компания, так и отдельно взятый программист, адаптирует свой продукт под веб-пространство. Однако, помимо удобства, такие интернет-сервисы должны быть быстродействующими и выдерживать высокую посещаемость.

На данный момент разработка высоконагруженных веб-сервисов ведется многими компаниями. Число пользователей интернета постоянно растет, и многие приложения за определенный промежуток времени набирают высокую посещаемость. Таким образом, на сегодняшний день стоит задача разработки оптимального программного кода.

Выбранная архитектура оказывает значительное влияние на производительность и отказоустойчивость приложения. Использование микросервисной архитектуры приложения может значительно увеличить общую производительность системы, а также упростить процесс масштабируемости приложения.

Цель бакалаврской работы – рассмотреть особенности микросервисной и монолитной архитектур приложения и реализовать приложение, подсчитывающее сетевой (netflow) трафик, используя эти архитектуры. Для достижения данной цели требуется решить следующие задачи:

1. Рассмотреть плюсы и минусы монолитной архитектуры приложения;
2. Рассмотреть плюсы и минусы микросервисной архитектуры приложения;

3. Сравнить монолитную архитектуру приложения с микросервисной;

4. Для демонстрации преимуществ микросервисной архитектуры разработать приложение, которое подсчитывает трафик из netflow пакетов;

5. Сравнить производительность, простоту масштабируемости, отказоустойчивость и другие факторы на примере этих приложений, выполняющих одну задачу.

В теоретической части необходимо рассмотреть плюсы и минусы монолитной и микросервисной архитектур приложения.

В практической части работы необходимо разработать приложение, подсчитывающее сетевой трафик.

Методологические основы. Сравнение микросервисной и монолитной архитектур на примере приложения, подсчитывающего сетевой трафик представлены в работах таких авторов как: Alvaro Videla, Jason J.W, Yuan Luo, Ioannis Deliyannis.

Теоретическая значимость бакалаврской работы. В работе приведены основные сведения о микросервисной архитектуре приложения. Приведены основные определения масштабирования приложения. Показана идея использования микросервисного подхода. Рассмотрены способы и проведено сравнение различных способов взаимодействий между сервисами. Проведен анализ самых часто используемых брокеров сообщений и показаны примеры их использования. Проведено сравнение и рассмотрены преимущества и недостатки микросервисного и монолитного подхода при выборе архитектуры приложения.

Практическая значимость бакалаврской работы. В работе описано и приведены фрагменты кода приложения, подсчитывающего netflow трафик. Рассмотрено предназначения и возможности netflow протокола. На примере данного приложения показаны преимущества использования микросервисного подхода, а также приведены данные о снижении нагрузки на вычислительную мощность процессора после изменения архитектуры приложения.

Структура и объём работы. Бакалаврская работа состоит из введения, четырех разделов, заключения, списка использованных источников и шести приложений. Общий объем работы – 63 страниц, из них 41 страница – основное содержание, включая 14 рисунков, список использованных источников информации – 20 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Монолитная архитектура приложения» посвящен рассмотрению в общих чертах монолитной архитектуры приложения. Рассмотрены преимущества и недостатки использования монолитной архитектуры, зависящие от размера и предназначения приложения.

Монолитная архитектура означает, что приложение – это большой связанный модуль, где все компоненты спроектированы так, чтобы работать вместе с друг другом, общая память и ресурсы.

Небольшое монолитное приложение имеет несколько вполне очевидных преимуществ:

1. Сравнительно простая реализация, развертывание и управление;
2. При минимальных трудозатратах возможно организовать работу нескольких модулей, которые должны взаимодействовать между собой, а также переместить классы из одного модуля в другой;
3. Достаточно просто добавить типичный функционал к уже готовым компонентам системы.
4. Скорость разработки приложения.

Описаны примеры решения недостатков такой архитектуры, а также показаны примеры удачного использования данной архитектуры.

Для решения этих недостатков используют архитектуру, в который клиент управляет исключительно пользовательским интерфейсом, а логикой приложения управляет дополнительный уровень программного обеспечения, который работает на серверной части приложения. Такая архитектура называется трехуровневой архитектурой. Между клиентской частью приложения и сервером передается лишь минимальный объем данных – аргументы вызываемых функций и значения, которые функция возвращает.

Показаны и описаны промежуточные слои монолитной архитектуры. User Interface – слой с которым работает пользователь, пользовательский интерфейс. Business Logic Layer – слой с основной логикой приложения. Data

Access Layer – слой, который работает с базой данных. DataBase – база данных.

Второй раздел «Микросервисная архитектура приложения» посвящен рассмотрению в общих чертах микросервисной архитектуры приложения.

Идея использования микросервисов приобретает все большую популярность в мире, она подразумевает деление приложения на небольшие функциональные модули. Данная архитектурная модель подразумевает масштабируемость и гибкость системы, а также способствует выбору лучших вариантов технологий для каждого конкретного компонента.

Микросервисный подход подразумевает под собой такой стиль архитектуры, в котором сложные приложения состоят и небольших, независимых и маленьких процессов (приложений), общающихся между собой с помощью запросов с использованием реализованных на каждом из таких сервисов API. Как правило отдельный микросервис решает конкретную бизнес задачу. Также эти приложения могут работать на разных серверах и взаимодействовать друг с другом по сети. Иными словами, необходимо инкапсулировать определённые контексты приложения в микросервисы, по одному на каждый.

Приведены основные причины масштабирования приложения.

Для того чтобы данное программное обеспечение соответствовало таким постоянно растущим требованиям, происходило наращивание вычислительных мощностей, то есть приложения вертикально масштабировались. Однако эффективность такого масштабирования оказалась достаточно ограниченной, так как прирост вычислительных мощностей определенного сервера не давал необходимого прироста производительности всему приложению, что повлекло за собой появление горизонтального масштабирования.

Горизонтальное масштабирование осуществляется за счет добавления новых модулей в систему, а не за счет увеличения вычислительных

мощностей определенного сервера. Архитектура, которая учитывает возможность горизонтального масштабирования еще на этапе проектирования системы называется микросервисной или конвеерной.

Третий раздел «Выбор системы управления базами данных» описывает факторы, которые необходимо принимать во внимание при выборе базы данных для различных приложений:

При выборе базы данных нужно принимать во внимание многие факторы. К наиболее важными критериям относятся:

1. Тип модели данных, которую поддерживает данная СУБД, адекватность модели данных структуре рассматриваемой ПО;
2. Характеристики производительности СУБД;
3. Запас функциональных возможностей для дальнейшего развития информационной системы;
4. Степень оснащённости СУБД инструментарием для персонала администрирования данными;
5. Удобство и надежность СУБД в эксплуатации; о наличие специалистов по работе с конкретной СУБД;
6. Стоимость СУБД и дополнительного программного обеспечения.

Четвертый раздел «Реализация приложения» описывает идею финального приложения, которое подсчитывает сетевой трафик. На примере данного приложения показаны преимущества использования микросервисного подхода, а также приведены данные о снижении нагрузки на вычислительную мощность процессора после изменения архитектуры приложения. Также рассматривается определение и предназначение netflow протокола.

Задача приложения заключается в подсчете neflow трафика. NetFlow – проприетарный открытый протокол, разработанный Cisco для мониторинга трафика в сети. Netflow предоставляет возможность анализа сетевого трафика на уровне сеансов, делая запись о каждой транзакции TCP/IP.

Архитектура такой системы строится на сенсоре и коллекторе. Сенсор собирает статистику по проходящему через него трафику. Сенсоры устанавливаются в «узловых точках» сети, например, на граничных маршрутизаторах сегментов сети. Коллектор осуществляет сбор информации от сенсоров. Полученные данные он обрабатывает и сохраняет в базу данных.

При архитектуре монолитного приложения вся нагрузка обработки netflow пакетов со всех сенсоров ложится на одно приложение. При большом кол-ве трафика, вычислительной мощности процессора может стать недостаточно. В таком случае возникает потребность в горизонтальном масштабировании приложения, разделении его на модули.

Из полученных данных в этом разделе можно сделать вывод, что при горизонтальном масштабировании микросервисного приложения, добавлением еще одного обработчика – получилось снизить нагрузку на процессор приблизительно в 2 раза при минимальных трудозатратах.

ЗАКЛЮЧЕНИЕ

В ходе исследования, проведенного в рамках данной дипломной работы, были выявлены преимущества при использовании микросервисной архитектуры в приложении.

Анализ, проведенный в рамках данной работы, показал, что для уменьшения нагрузки на вычислительную мощность процессора сервера необходимо закладывать в архитектуру приложения возможность горизонтального масштабирования. А также в ходе исследования микросервисной архитектуры было выявлено, что данная архитектура очень просто горизонтально масштабируется добавлением дополнительных обработчиков. А, следовательно, позволяет проводить большое количество вычислительных операций при минимальных трудозатратах. Также удалось показать, что использование разных языков программирования для разных компонентов системы микросервисного приложения позволяет значительно снизить нагрузку на сервер-обработчик.

Дополнительно стоит отметить, что результаты исследований, проведенных в рамках данной работы, применены на практике в ИТ компании ООО «Надым Связь Сервис», а именно: использовано микросервисное приложение для подсчета сетевого трафика более 50 тысяч абонентов в нескольких городах России.

Основные источники информации:

1. Alvaro Videla, Jason J.W. RabbiMQ in Action: Distributed Messaging for EveryOne – Manning Publications Corparation, 2012.
2. Lovisa Johansson. Getting Started with RabbitMQ and CloudAMQP – 84 Codes AB, 2017.
3. Карпова И.П. Базы данных. Учебное пособие. – Московский государственный институт электроники и математики. – М., 2009.
4. Martin L. Abbott. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. – Addison-Wesley Professional, 2015. – 624 с.
5. Микросервисы – за и против. [Электронный ресурс]. – URL: http://devopsru.com/news/2016-05-10-microservice-trade-offs.html (Дата обращения 25.11.2019).