МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра дискретной математики и информационных технологий

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ МОНОЛИТНОЙ И МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ GOLANG

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 271 группы
направления 09.04.01 — Информатика и вычислительная техника
факультета КНиИТ
Берденева Владислава Алексеевича

Научный руководитель
доцент, к. ф.-м. н. _____ В. А. Поздняков

Заведующий кафедрой
доцент, к. ф.-м. н. _____ Л. Б. Тяпаев

Саратов 2022

ВВЕДЕНИЕ

В настоящее время перед компаниями встает проблема выбора архитектуры для своих новых проектов, ведь это важно — позаботиться заранее о продуманной архитектуре или отложить её на потом. И выбор в основном встает между монолитной и микросервисной архитектурами.

Быстрое становление и распространение сетевых облачных сервисов привело к разочарованию в традиционном, например именуемом монолитном, варианте архитектуры приложений. По причине сложности отдельных частей системы, нередко представляющих собой целые программные блоки, к тому же по причине необходимости гарантировать сопоставимость между ними при помощи стандартных протоколов, внесение изменений и дополнений стало нетривиальной задачей, занимающей довольно продолжительное количество времени.

Была предложена микросервисная архитектура как распределенная система простейших и легко заменяемых модулей, выполняющих по возможности единственную элементарную функцию. При этом микросервисная система содержит симметричную одноранговую, а не иерархическую структуру, что позволяет отказаться от сложной организации взаимосвязей. В итоге формируется система, которая считается легкой в развёртывании и модернизации, с функциями автоматической разработки и обновления.

Проблемой микросервисной архитектуры является своевременный, а в случае микросервисов, непрерывный мониторинг. Мониторинг данной архитектуры гораздо сложнее в настройке и требует больших затрат на свою поддержку, нежели мониторинг монолитного приложения.

В настоящее время микросервисная архитектура является наиболее оптимальным подходом для разработки облачных приложений. Данные приложения состоят из большого количества слабо связанных и автономно разрабатываемых небольших модулей — сервисов.

Микросервисы поддерживаются практически всеми языками и представляют собой концепцию, а не определенную структуру или инструмент. При этом некоторые языки лучше подходят и, кроме того, имеют лучшую поддержку для микросервисов, чем другие. Одним из языков программирования с большей поддержкой является Golang.

Целью данной работы является исследование сетевых пакетов на языке

Golang и детальный анализ монолитной и микросервисной архитектуры для разработки приложения, с помощью которого будет выполнен сравнительный анализ эффективности работы приложения на той или иной архитектуре.

В ходе работы были поставлены следующие задачи:

- анализ пакетов для сетевого программирования для языка Golang и применение их в клиент-серверной системе;
- анализ монолитной архитектуры;
- анализ микросервисной архитектуры;
- проектирование проекта;
- тестирование и сравнение результатов быстродействия приложений на монолитной и микросервисной архитектуре.

Объектом исследования данной работы является сравнение монолитной и микросервисной архитектур. Предметом исследования является приложение, реализованное на монолитной и микросервисной архитектурах, основанное на исследуемых сетевых пакетах, написанных на языке Golang.

В первой главе рассматриваются основные концепции языка Golang в сетевом программировании, а именно обзор пакетов и инструментария, принципы работы веб-сервисов и клиент-серверных приложений. Вторая глава посвящена анализу различных типов архитектур, обзор их преимуществ и недостатков, также сравнивается их ценность в бизнес моделях и применение на практике. В третьей главе подробно рассматривается непосредственно используемая микросервисная архитектура, детали, которые отличают её от других архитектур, и инструментарий для реализация приложения. Четвёртая глава представляет собой поэтапную разработку программы на монолитной и микросервисной архитектурах, а именно создание простейшей клиент-серверной системы, проектирование и реализация монолитного приложения, которое в последующем перетекает в приложение на микросервисах, и в конечном итоге сравнение результатов быстродействия приложений.

Магистерская работа состоит из введения, четырех глав, заключения, списка используемых источников и приложения. Общий объем работы — 79 страниц, из них 70 страниц — основное содержание, включая 35 рисунков, список использованных источников информации — 27 наименования, 5 приложений.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

В первой главе рассказывается об основных концепциях языка Golang в сетевом программировании. В ней дается понятие, перечисляются основные особенности, преимущества и недостатки языка Golang, а также продемонстрировано огромное количество пакетов, предоставляющих самые разные функциональные возможности. Также поднимаются принципы работы веб-сервисов и клиент-серверных приложений, а именно как программа производит передачу и получение данных.

Вторая глава посвящена анализу основных типов архитектур, которые могут использоваться при написании программы: многослойная, многоуровневая, сервис-ориентированная, монолитная, микросервисная. В данной главе продемонстрирован краткий разбор первых трех архитектур и подробный анализ двух последующих, так как они являются популярными архитектурами для проектирования и непосредственно используются для реализации поставленной цели работы. В монолитной и микросервисной архитектурах описываются преимущества и недостатки их использования, также поднимается ценность данных архитектур в бизнес моделях и их применение на практике.

В третьей главе подробно рассматривается используемая микросервисная архитектура, а именно произведены сравнения данной архитектуры по отношению к остальным и реализована схема перехода от монолитной архитектуры к микросервисам. Также формулируются требования к проекту, в плане технических средств, то есть необходимость использования фреймворка gRPC и средства передачи данных Protocol Buffers, и производится его проектирование с использованием различных шаблонов.

Четвертая глава посвящена поэтапной разработке программы. Для начала была создана клиент-серверная система, использующая TCP протокол, для обмена данными. В последующем было реализовано монолитное приложение с применением всех требований, объявленных ранее при проектировании, и приведено сравнение быстродействительности работы данной программы при развёртывании на локальной машине и при её контейниризации. Следующим этапом было проектирование микросервисной программы и план ухода от монолитной архитектуры. В конце мы имеем реализованное приложение на микросервисной архитектуре и сформированные диаграммы, представленные на рисунках 1, 2, 3, для сравнения результатов быстродействия приложений,

где видна высокая эффективность использования микросервисной архитектуры.

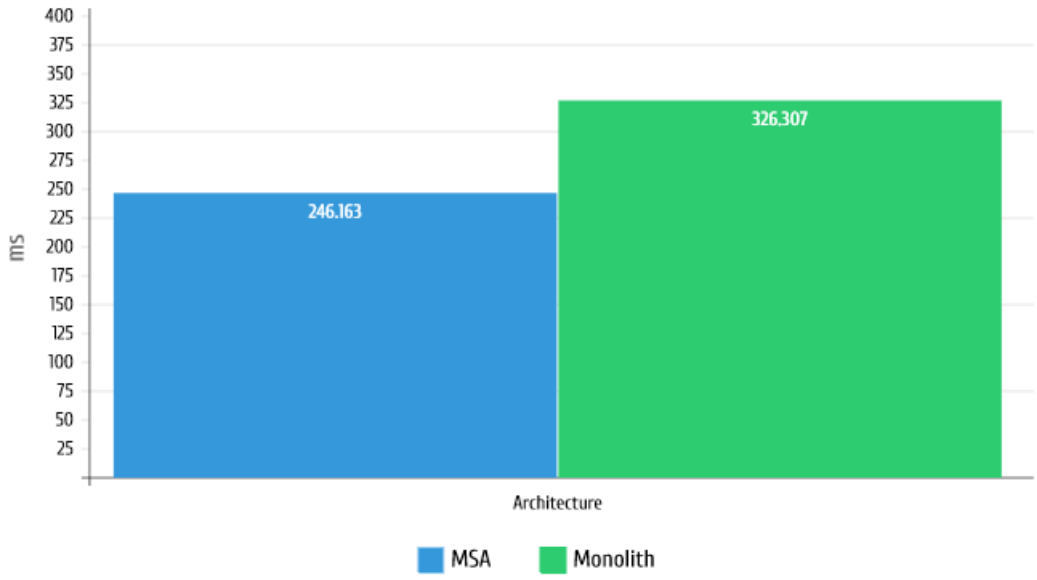

Рисунок 1 – Диаграмма Deploy time

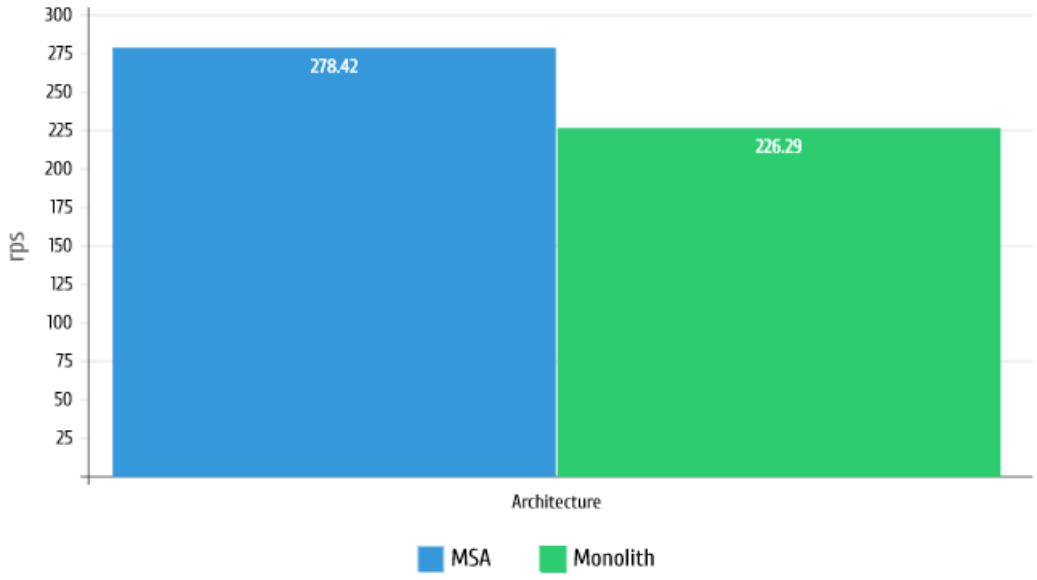

Рисунок 2 – Диаграмма RPS

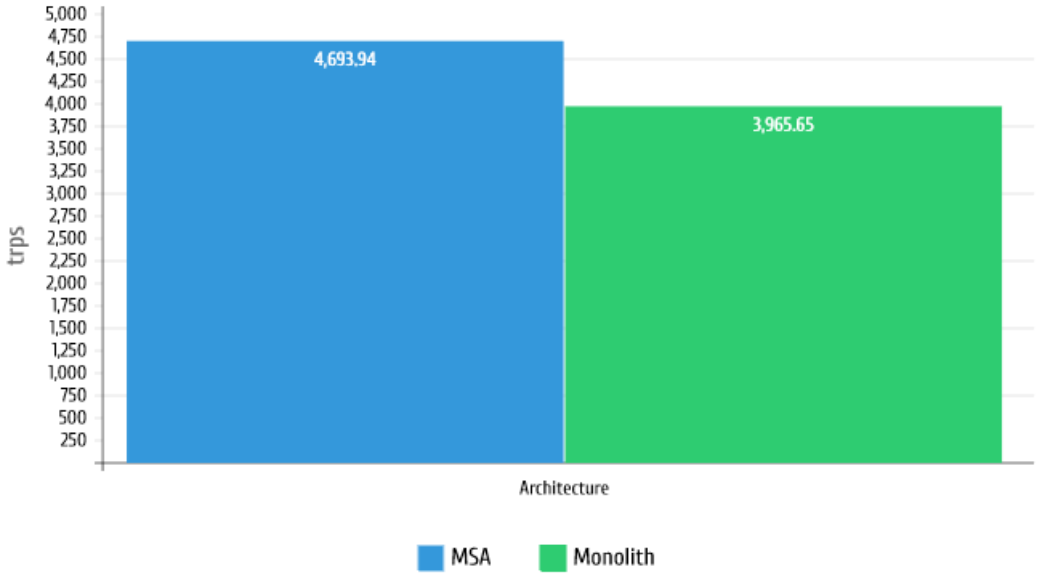

Рисунок 3 – Диаграмма TRPS

ЗАКЛЮЧЕНИЕ

В работе представлен разбор и применение пакетов для сетевого программирования на практике для реализации простейшей клиент-сервисной системы. В последующем проанализированы основные типы архитектур и подробно исследованы монолитная и микросервисная архитектуры, применяемые для реализации программы. Проведено проектирование проекта и его последующая поэтапная реализация от клиент-сервисной системы до функционирующего приложения на микросервисах, которое принимает входные данные для последующего вычисления и произведения сравнения с реализованным ранее монолитным приложением.

Проведён сравнительный анализ монолитной архитектуры с микросервисной на основе использования контейниризации для придачи приложениям условий изолированности для произведения вычислений числа Пи.

Предоставленные результаты показывают высокую эффективность применения микросервисной архитектуры в процессе ведения бизнеса и реализации тривиальных задач, к примеру это позволяет получить следующие преимущества:

- простота редактирования кода;
- возможность разных подразделений взаимодействовать с необходимыми именно им модулями;
- компоненты имеют возможность масштабироваться автономно друг от друга, что в свою очередь позволяет снизить затраты и стоимость масштабирования всей программы в тех случаях, если уязвимым местом является какая-то одна функция.

Основные источники информации:

1 Butcher, M. Go in practice / M. Butcher, M. Farina. — Greenwich: Manning, 2016. — 312 pp.

2 Chang, S. S. Go web programming / S. S. Chang. — Shelter Island New York: Manning Publcations Co, 2016. — 311 pp.

3 McGavren, J. Head first Go / J. McGavren. — First edition edition. — Beijing: O'Reilly, 2019. — 560 pp.

4 Tsoukalos, M. Mastering Go / M. Tsoukalos. — Second edition edition. — Birmingham: Packt Publishing Ltd, 2019. — 798 pp.

5 Distributed Systems Architecture | MDN [Электронный ресурс]. — URL:

https://jan.newmarch.name/go/arch/chapter-arch.html (Дата обращения 24.05.2021). Загл. с экр. Яз. англ.

6 Introduction to Microservices | MDN [Электронный ресурс]. — URL: https://www.nginx.com/blog/introduction-to-microservices/ (Дата обращения 02.02.2022). Загл. с экр. Яз. англ.

7 Newman, S. Building Microservices: Designing Fine-Grained Systems / S. Newman. — O'Reilly Media, Inc., 2015. — 280 pp.