Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

СОЗДАНИЕ КРЕДИТНОГО КОНВЕЙЕРА НА ОСНОВЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Мирзоева Никиты Романовича

Научный руководитель
к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кредитные конвейеры имеют важную роль в банковской сфере, так как именно через эти системы у банков появляются новые клиенты. Задача кредитных конвейеров автоматизировать и ускорить максимальное количество задач при выдаче кредитов. Со стороны бизнеса постоянно появляется множество новых требований, поэтому архитектура кредитных конвейеров должна строиться таким образом, чтобы различные части системы были независимыми и можно было вводить новый функционал быстро в эксплуатацию.

Создание кредитных конвейеров на основе микросервисов позволяет удовлетворить быстрорастущим требованиям банков. Поэтому в рамках данной работы была выделена актуальная задача: исследовать возможности и преимущества микросервисной архитектуры при создании кредитного конвейера. Для решения этой задачи были поставлены следующие цели:

1. изучить архитектуру построения систем на базе микросервисов;
2. создать служебные сервисы, необходимые для администрирования микросервисов и автоматизации их взаимодействия между собой;
3. реализовать сервис ввода данных о клиенте;
4. разработать сервис автоматического принятия решения о выдаче кредита;
5. использовать движок бизнес-процесса для маршрутизации кредитных заявок.

1 Использование микросервисной архитектуры при создании кредитного конвейера

Кредитный конвейер предназначен для автоматизации процесса обработки кредитной заявки на потребительские кредиты физических лиц [1].

Он обеспечивает бесперебойный доступ к актуальной информации по кредитным заявкам клиентов банка и позволяет:

- Подобрать кредитный продукт при помощи функционала кредитного калькулятора на основании представленной клиентом информации;
- Сформировать и распечатать предварительный график погашения платежей и предварительный расчет;
- зарегистрировать заявку и ввести краткую информацию по клиенту;
- прикрепить фото клиента, документы, предоставленные клиентом;
- осуществить полный ввод данных по всем участникам кредитной заявки по приложенным скан-копиям документов;
- ввести данные по имуществу, предоставляемому в залог;
- выполнить проверку залога, и приложить к заявке заключение о проведении проверки;
- выполнить проверку службой безопасности и телефонную верификацию участников кредитной заявки;
- на основании результатов автоматических и ручных проверок принять решение по заявке;
- распечатать договорную базу, автоматически сформированную по банковским шаблонам;
- приложить к заявке скан-копии документов, подписанных клиентом;
- осуществить контроль корректности приложенных к заявке документов;
- автоматически перевести заявку на следующий этап ее обработки и назначить на исполнение сотруднику подразделения;
- перераспределить заявки между сотрудниками подразделения;
- отследить действия пользователей и собрать статистику по текущим заявкам;
- отредактировать системные справочники и настроить поведение элементов экранных форм без внесения изменений в программный код;
- внести информацию по Партнерам.

Наличие большого количества возможностей кредитного конвейера при-

водит к декомпозиции общей системы на множество подсистем, которые должны взаимодействовать между собой. Сервис-ориентированная архитектура (SOA) давно применяется при разработке кредитных конвейеров в части интеграции с внешними системами. Наследником сервис-ориентированного подхода является микросервисная архитектура (MSA), которая разделяет приложение на части, каждая из которых выполняет свою конкретную задачу.

1.1 Микросервисная архитектура

Микросервисная архитектура — это подход к созданию приложения, подразумевающий отказ от единой, монолитной структуры. То есть вместо того чтобы исполнять все ограниченные контексты приложения на сервере с помощью внутрипроцессных взаимодействий, мы используем несколько небольших приложений, каждое из которых соответствует какому-то ограниченному контексту. Причём эти приложения работают на разных серверах и взаимодействуют друг с другом по сети, например по протоколу HTTP [2].

Свойства, характерные для архитектуры микросервисов:

- Модули можно легко заменить в любое время
- Модули организованы вокруг функций, например, пользовательский интерфейс, логистика, выставление счета и т. д.
- Модули могут быть реализованы с использованием различных языков программирования, баз данных, аппаратных средств и программного обеспечения, в зависимости от того, что подходит лучше всего
- Архитектура симметричная, а не иерархическая (производитель-потребитель)

Типы микросервисной архитектуры:

- Service Discovery — сервисы знают друг о друге и общаются напрямую
- Message Bus — сервисы реализованы по шаблону издатель-подписчик, но при этом ничего не знают друг о друге, т.е. самым важным объектом являются сообщения
- Hybrid — смешанный вариант Service Discovery и Message Bus

Простейший вариант Service Discovery, когда клиент напрямую обращается к сервисам. При таком подходе сервис клиента сильно связан с остальными сервисами, т.к. в его конфигурации зашиты адреса сервисов. В такой ситуации нельзя создавать экземпляры уже существующих сервисов.

Существует два вида решения такой проблемы. Первым вариант — Server-

Side Service Discovery. При Server-Side Service Discovery клиент взаимодействует не напрямую с конкретным сервисом, а с выравнивателем нагрузки (load balancer).

Load balancer берет все данные у service registry. Таким образом, задача load balancer — просто брать данные о местоположении сервисов из service registry и раскидывать запросы к ним. А задача service registry — хранить регистрационные данные сервисов, и он это делает по-разному: может опрашивать сервисы сам, брать данные из внешнего конфигурационного файла и т. д.

Второй вариант — Client-Side Service Discovery. Здесь нет load balancer, и сервис обращается напрямую к service registry, откуда берет адрес сервиса. Цепочка вызовов сервисов короче, а значит работа приложения быстрее. Но у такого решения есть минус — клиентский сервис имеет прямой доступ к данным, поставляемым остальными сервисами.

У Service Discovery есть несколько недостатков. При реализации большого бизнес-процесса сервисы становятся более связанными друг с другом. Более того при такой архитектуре отсутствует согласованность, а значит нарушена транзакционность.

Для решения проблемы транзакционности можно использовать брокера, который будет отвечать за согласованность всех вызовов сервисов. Брокер лежит в основе подхода Message Bus. Сервисы взаимодействуют друг с другом через шину (брокера), который передает данные нужным сервисам в виде сообщений. Такой подход по-другому называется Event-Driven, т.к. он основан на событиях, происходящих в сервисах.

Однако взаимодействие через сообщения влечет проблему: передача больших объемов данных в сообщениях замедляет работу приложения. Поэтому лучшим решением становится гибридная архитектура. Необходимо по Message Bus отправить сообщение, что какие-то данные поменялись. После этого подписчики реагируют на эти данные, идут в registry, забирают по идентификатору отправителя место, куда надо сходить за данными, и уже идут напрямую. Таким образом нагрузка на шину уменьшается, сообщения обрабатываются быстрее, а значит приложение ускоряется [3].

2 Разработка прототипа кредитного конвейера на основе микросервисной архитектуры

2.1 Описание требований

Для получения практических навыков построения микросервисной архитектуры был разработан прототип кредитного конвейера, разбитый на несколько микросервисов в соответствии с различными этапами БП.

Атрибутный состав бизнес-объекта «Кредитная заявка»:

- Сумма кредита
- Клиент
- Дата заведения
- Процентная ставка
- Срок кредита
- Название кредитного продукта
- Исполнитель
- Статус

Атрибутный состав бизнес-объекта «Клиент»:

- Фамилия, имя, отчество
- Дата рождения
- ИНН
- СНИЛС
- Адрес
- Общий доход
- Семейное положение

Сервис принятия решений должен на основе информации, введенной на ручном этапе ввода данных о клиенте, анализировать надежность клиента банка и возвращать решение по данной кредитной заявке.

Помимо этого одной из задач было изучение новых технологий в рамках разработки данного решения и расширение технологического стека, используемого при построении систем автоматизации выдачи кредитов.

2.2 Архитектура решения

Было разработано приложение, основанное на микросервисной архитектуре, состоящее из компонентов, взаимодействующих между собой по REST API. За счет этого достигнута слабая связность кода и разделение обязанностей. При разработке использовались технологии, содержащие в себе множество решений «из коробки», что позволило ускорить процесс конфигурации и развертывания без большого количества ресурсов. Также при разработке использовались такие шаблоны, как Service Discovery, Gateway, Dependency Injection.

Настройки каждого Docker-контейнера, в котором разворачивается отдельный сервис, прописаны в соответствующих Dockerfile.

Исходный код сервисов написан на Kotlin, сами модули собираются инструментом сборки Gradle. Для развертывания приложений во встроенном контейнере сервлетов Apache Tomcat используется Spring Boot [4]. Для реализации RESTful интерфейсов разработаны REST-контроллеры, обрабатывающие GET и POST запросы по HTTP протоколу.

Состав сервисов:

- bpm — Сервис работы с БП
- data-input — Сервис ввода данных
- make-decision — Сервис принятия решения
- service-registry — Реестр сервисов
- gateway — Шлюз

2.3 Результат работы программы

Для демонстрации работы прототипа кредитного конвейера необходимо зайти в веб-интерфейс сервиса ввода данных и создать заявку. Далее заполнить заявку данными и нажать кнопку «Продолжить». После возвращения на стартовую страницу можно увидеть результат принятия решения, проверив колонку «Статус» в таблице с кредитными заявками (рисунок 1).

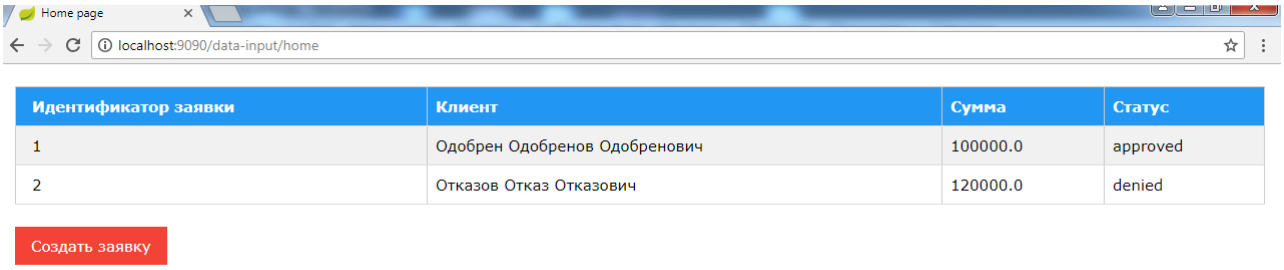

Рисунок 1 – Результат принятия решения

ЗАКЛЮЧЕНИЕ

В рамках данной работы были исследованы возможности и преимущества микросервисной архитектуры при создании кредитного конвейера. Для решения этой задачи было сделано следующее:

1. изучена архитектура построения систем на базе микросервисов;
2. созданы служебные сервисы, необходимые для администрирования микросервисов и автоматизации их взаимодействия между собой;
3. реализован сервис ввода данных о клиенте;
4. разработан сервис автоматического принятия решения о выдаче кредита;
5. использован движок бизнес-процесса для маршрутизации кредитных заявок.

При выполнении работы я познакомился с принципами построения микросервисных систем и закрепил полученные знания на практике. Разработанный прототип кредитного конвейера на микросервисах стал основой при создании более масштабного проекта Neoflex в рамках участия в тендере на разработку программного обеспечения, проводимого одним из крупнейших российских банков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Кредитный конвейер [Электронный ресурс]. — URL: https://www.neoflex.ru/solutions/front-ofis (Дата обращения 12.05.2018). Загл. с экр. Яз. рус.

2 *Newman, S.* Building Microservices / S. Newman. — London: O'Reilly, 2015.

3 What are microservices? [Электронный ресурс]. — URL: http://microservices.io (Дата обращения 16.05.2018). Загл. с экр. Яз. англ.

4 *Guthrie, J.* Spring Microservices in Action / J. Guthrie. — New York: Manning, 2017.