

Принципы разделения сервисов при проектировании информационной системы

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной работе описываются методики для построения системы основанной на микросервисной архитектуре. В содержании приводятся примеры и устойчивые шаблоны проектирования, обеспечивающие безопасную и удобную настройку взаимодействия между сервисами. Результаты работы позволяют определить оптимальную архитектуру построения сервисов как ограниченных контекстов, каждый из которых имеет свой доступ и представление о данных всей системы.

Ключевые слова: микросервисная архитектура, межсервисное взаимодействие, горизонтальное масштабирование.

Principles of service separation in information system design

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This paper describes techniques for building a system based on a microservice architecture. The content provides examples and sustainable design patterns that ensure secure and convenient configuration of interactions between services. The results of the work allow to determine the optimal architecture of building services as limited contexts, each of which has its own access and view of the data of the entire system.

Keywords: microservice architecture, inter-service communication, and horizontal scaling.

1 Введение

1.1 Актуальность

Микросервисы применяют для повышения автономности команд разработчиков. Детали реализации оставляют командам. В рамках каждого отдельного сервиса выбираются язык программирования, стек технологий, подходы к проектированию. Однако по мере роста системы и команд разработчиков принято ограничивать доступные разработчикам технологические средства. В противном случае компания сталкивается трудностями при эксплуатации и развитии систем. В число ограничений

входят языки программирования, базы данных, брокеры сообщений. Также это позволяет не распылять ресурсы, а существенно нарастить экспертизу и знания сотрудников для дальнейшей эксплуатации системы на возрастающих масштабах.

При проектировании, самой, системы основное внимание уделяется тому, что происходит на границах контекста, как общаются сервисы, где и как хранят данные, каким образом происходит мониторинг состояния системы, какие используются протоколы взаимодействия и системы управления базами данных. Стандартизация взаимодействия упрощает разработчикам работу с системой и помогает сконцентрироваться только на бизнес функциональности.

1.2 Обзор исследований

Статья А.М.Шитько посвящена изучению технологии построения микросервисной архитектуры, которая позволяет сохранять модульность и независимость развертывания компонент Enterprise приложения [1]. В работе Д.С.Кугушева дано описание принципа работы микросервисной архитектуры, приведены ее отличительные особенности, позволяющие существенно упростить процесс разработки больших продуктов, а также упомянуты основные технологические особенности микросервисного подхода, которые необходимо учитывать при выборе этой технологии разработки. [2]. В исследовании М.Э.Джалалова особое внимание уделяется шаблонам API Gateway, Backend For Frontend и Circuit Breaker, их реализации и оптимизации в контексте микросервисных архитектур. [3]. В качестве результата исследования А.Абдирашитулы, стала предложенная концепция LLM-ориентированной микросервисной архитектуры, обеспечивающая изоляцию ресурсов и независимое масштабирование компонентов. [4]. В статье М.Г.Городничев и Р.В.Полонский сделан обзор типов современных веб-приложений и архитектуры, используемой для их разработки, дано сравнение микросервисной и монолитной архитектуры [5]. М.Н.Карпович исследовал особенности проектирования микросервисно-событийных архитектур для высоконагруженных распределенных систем обработки информации [6]. С.В.Маличенко рассмотрел проблемы перехода от монолитной к микросервисной архитектуре [7]. Г.Р.Берайлик описал способы спроектирования информационных систем с помощью микросервисной архитектуры [8].

1.3 Цель исследования

Цель исследования – описать принципы разделения сервисов при проектировании информационной системы, основанной на микросервисной архитектуре.

2 Материалы и методы

В работе учитываются позиции со стороны организаторов предприятия, лидеров команд разработчиков и самих разработчиков для определения

совокупной связанности всей системы среди отдельных процессов. Устанавливается связь используемых практик разработчиков с целью соблюдения принципов лидеров команд направленных на достижение стратегических задач бизнеса.

3 Результаты и обсуждения

Каждая организация имеет цели, которые не привязаны к технологическому стеку. Но технологическое развитие системы должно быть направленно на достижение стратегических целей. Для этого применяются (принципы) правила, которые позволяют направить усилия организации на достижения стратегических целей. Принципы сопоставляют, что нужно делать с тем, как нужно делать. Чтобы притворить принципы в жизнь используются практики. Практиками являются набор инструкций и ограничений, которые понятны всем разработчикам команды (рис.1).



Рис. 1. Взаимосвязь целей, принципов и практик предприятия

Практики могут включать правила кодирования, форматы записи логов, способы меж сервисного взаимодействия, протоколы обмена данными. Разные практики должны обеспечивать одни и те же принципы. При проработке практик важно понять какое разнообразие можно позволить в системе и выделить те части сервисов, которые должны быть одинаковыми, чтобы потом было проще эксплуатировать систему, понимать, что происходит с системой и как исправить непредвиденные ситуации. Таким образом, важно найти баланс независимости каждого сервиса при этом не терять общую структуру системы. В необходимый минимум практик входят централизованный мониторинг и сбор логов, ограниченный набор допустимых интерфейсов (протоколов) и безопасное поведение.

Каждый конкретный сервис должен обладать слабой связанностью и сильным зацеплением. При этом слабая связанность означает, что изменения в одном сервисе не должны требовать изменения в других. Такие изменения включают внутренние изменения, изменения API или конфигурации. Сервис

должен отвечать за конкретную задачу и иметь только один повод для изменений. Таким образом, поставка сервисом происходит независимо друг от друга.

Принцип сильного зацепления требует, чтобы связанные поведения размещались в одном месте. Такой подход избавляет от сложностей и рисков, которые происходят при необходимости изменения поведения и поставки нескольких сервисов.

В проектно-ориентированном проектировании существует шаблон ограниченного контекста. Предметная область разбивается на набор ограниченных моделей, каждый из которых отвечает за собственный набор функций или часть предметной области, при этом предоставляя остальным ограниченным моделям чёткий интерфейс для взаимодействия. При таком подходе предметная область разделяется на зоны, где каждая описывает часть системы. Например, в предметной области интернет магазина ограниченными контекстами могут быть витрина, корзина, система учёта заказов, склад, система доставок (рис.2).

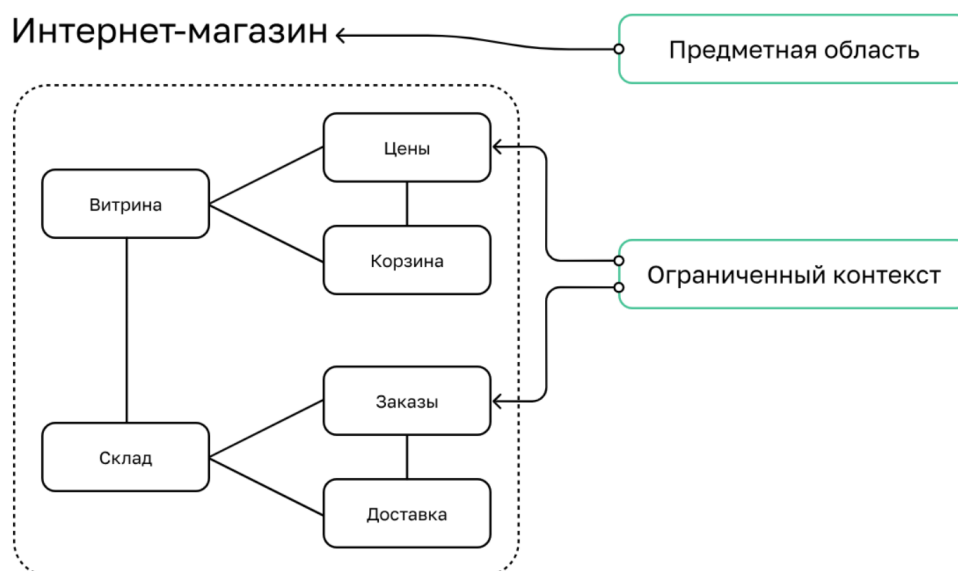


Рис. 2. Шаблон проектирования ограниченного контекста интернет магазина

В каждом контексте одна и та же сущность представляется несколько по-разному. Для витрины товаром является картинка с ценой, а для склада артикул с местом хранения. Таким образом, вся система целиком имеет полное представление о сущностях, но каждый ограниченный контекст описывает только ограниченную часть сущностей, которые нужны для выполнения его функций. Преимуществами такого разбиения является то, что границы сервисов выбраны не случайно, а на основе принципов и подходов предметно-ориентированного проектирования.

Помимо разбиения системы, на сервисы, используя подход ограниченных контекстов, следует ограничить не только размер сервисов и внешний контракт, но и то кто и как им владеет. Ограниченный контекст

означает, что у сервисов выделены отдельные команды, репозитории, схемы баз данных, процедуры тестирования и выкладки.

Несмотря на то, что микросервисный подход в своём названии указывает на размер сервисов, стоит учитывать, что размер не должен быть причиной, чтобы разделять систему на сервисы. Важно иметь представления о том, зачем выделяется каждый отдельный сервис. Иногда случается необходимость разбить систему на ещё более мелкие сервисы. В число причин входит частота изменения.

В случае, когда определённые части системы приходится менять чаще, чем остальные есть возможность выделять их в отдельные сервисы, которые могут реализовывать часть контекста. Это позволит обеспечить независимость выкладки часто меняющегося кусочка. Другим фактором уменьшения сервисов и выделения функционала в отдельный сервис может быть необходимость к масштабированию частей системы. Следующей причиной является независимость ресурсов хоста, такими как оперативная память и процессорное время. В таком случае компоненты сервиса разделяются по разным хостам.

4 Выводы

Таким образом, стандартизация протоколов взаимодействия сервисов и хранения данных направлены на повышение общей скорости разработки системы, а общая устойчивость системы важнее, чем независимость каждого отдельного сервиса. Равносильно и то, что когда со временем сервис усложняется, при этом увеличивается кодовая база и количество функций реализующих сервисом, в него сложнее и дольше вносить изменения. Тогда логичнее становится разделения сервиса с целью уменьшения сложности.

Библиографический список

1. Шитько А.М. Проектирование микросервисной архитектуры программного обеспечения //Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2017. №. 9 (200). С. 122-125.
2. Кугушева Д.С. Проектирование сложного программного обеспечения с использованием микросервисной архитектуры //Инновации и инвестиции. 2020. №. 5. С. 188-190.
3. Джалалов М.Э. Применение шаблонов проектирования для управления API в микросервисной архитектуре //Экономика и качество систем связи. 2024. №. 1 (31). С. 128-136.
4. Абдирашитулы А. Микросервисная архитектура в NLP платформах: проектирование и масштабируемость //Вестник науки. 2026. Т. 1. №. 5 (98). С. 609-619.
5. Городничев М.Г., Полонский Р.В. Оценка возможности использования микросервисной архитектуры при разработке пользовательских интерфейсов клиент-серверного программного обеспечения //Экономика и качество систем связи. 2020. №. 3 (17). С. 33-43.

6. Карпович М.Н. Особенности проектирования микросервисно-событийных архитектур для высоконагруженных распределенных систем обработки информации //Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2023. №. 1 (266). С. 89-95.
7. Маличенко С.В. Проблемы перехода от монолитной к микросервисной архитектуре //Евразийский научный журнал. 2022. №. 5. С. 8-19.
8. Берайлик Г.Р. Проектирование информационных систем с помощью микросервисной архитектуры //Инновационные подходы к решению технико-экономических проблем. 2020. С. 187-191.