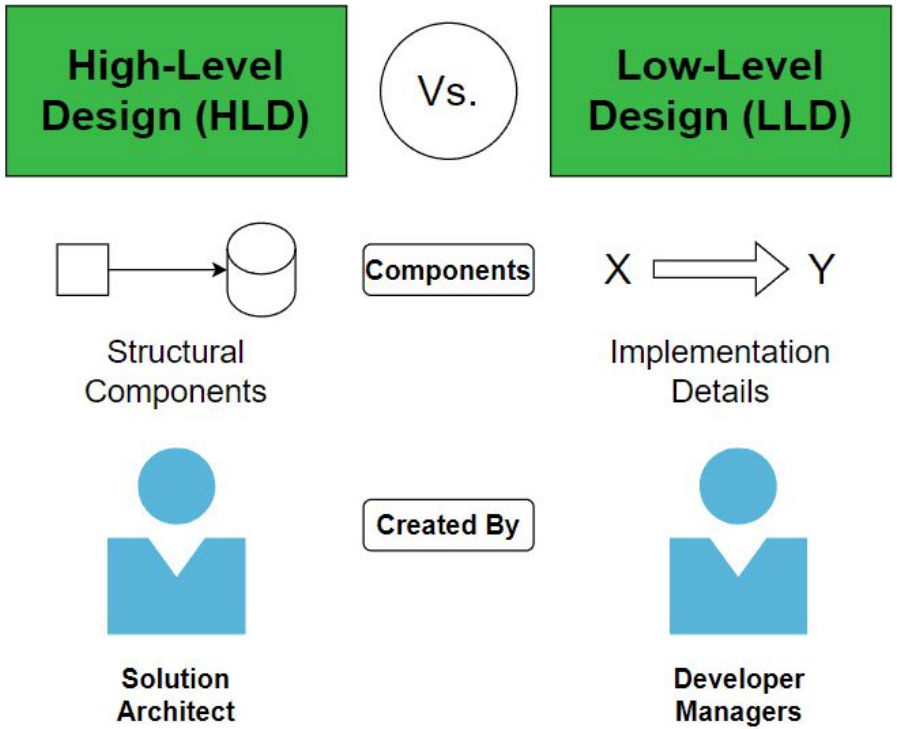
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Системный дизайн современных приложений

Лекция №3
Виды архитектуры.
HLD: основные сущности.

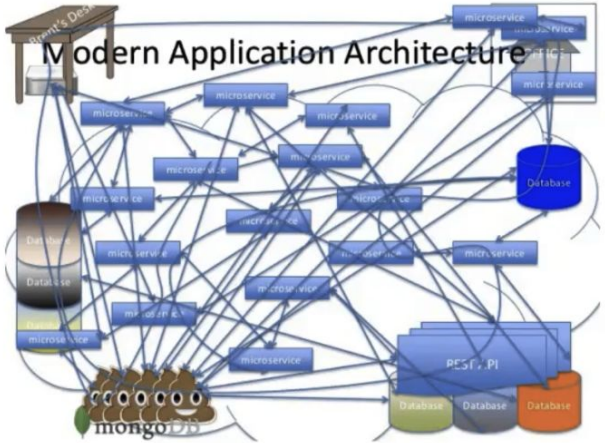
HLD vs LLD


High-Level Design (HLD)
Vs.
Low-Level Design (LLD)
Components
X
Y
Structural Components
Implementation Details
Created By
Solution Architect
Developer Managers

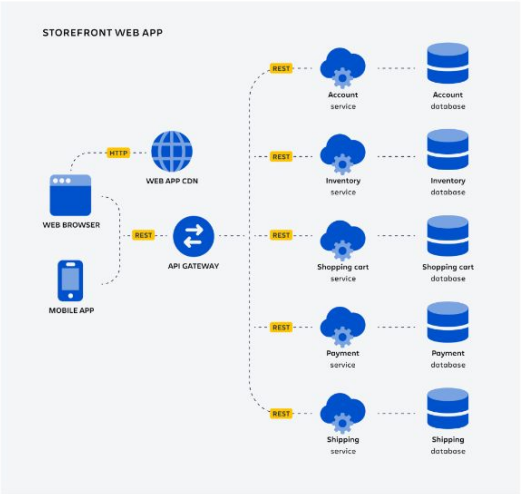
Эволюция архитектуры

Монолит

Переходная стадия: Распределенный монолит



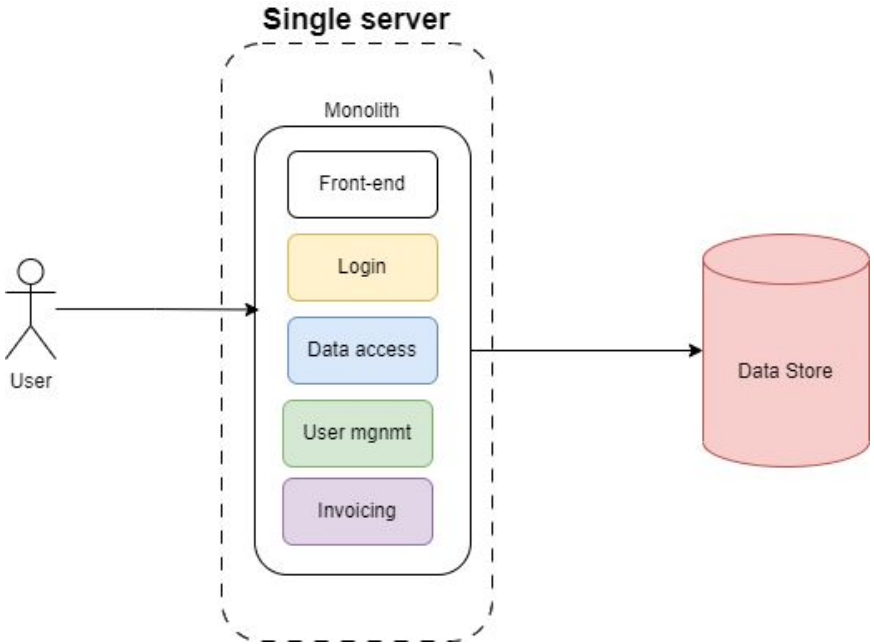
Микросервисы



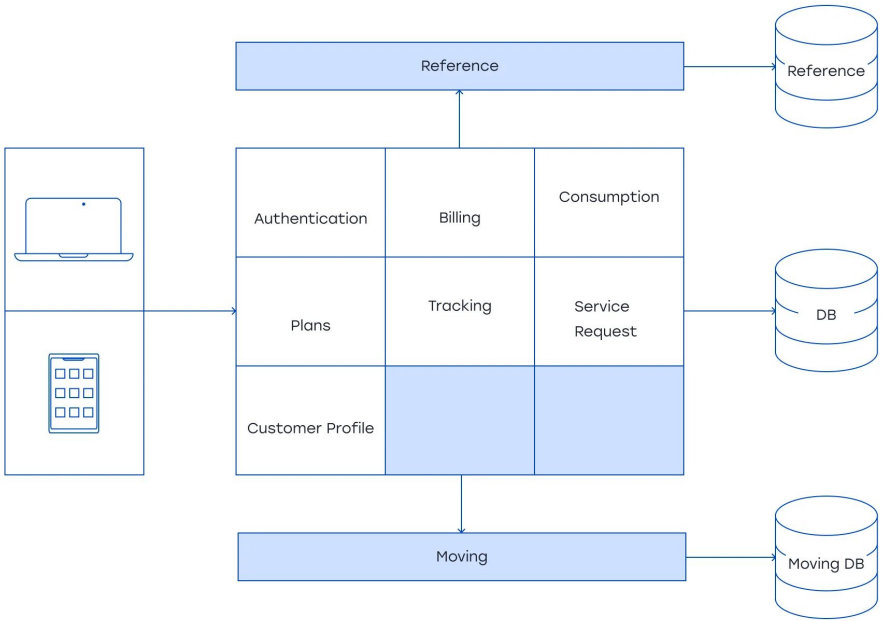
Монолит

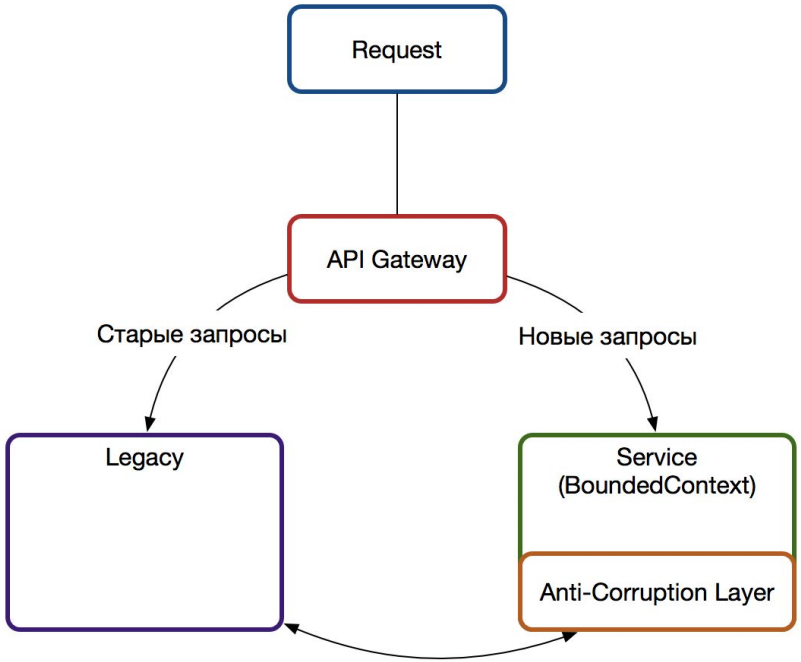
Монолитная архитектура

1. *Единая точка разработки и деплоя*
2. *Единая база данных*
3. *Единый цикл релиза для всех изменений*
4. *В одной системе реализовано несколько бизнес-задач*



Легаси



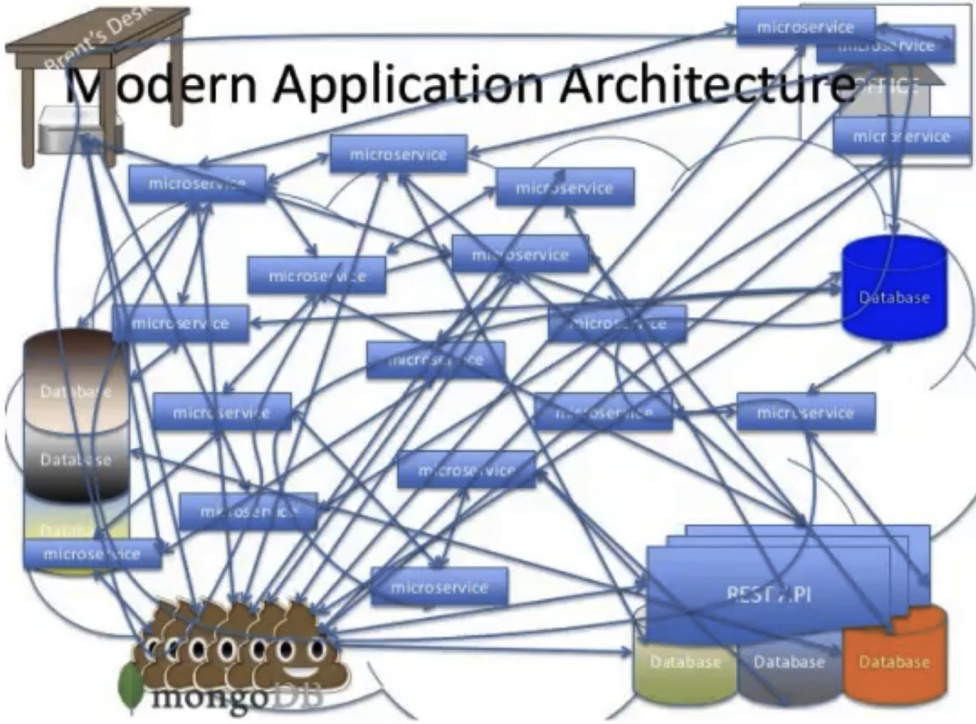


https://www.litres.ru/book/sem-numen/ot-monolita-k-mikroservisam-67727850/

Распределенный монолит

Определение

Архитектурный паттерн, в котором компоненты системы, хотя и развернуты ***в виде отдельных сервисов****, фактически зависят друг от друга настолько, что не могут развиваться и масштабироваться независимо.*



Распределенный монолит

Признаки распределенного монолита

1. Высокая связанность между сервисами, требующие координации при деплое
2. Глобальные транзакции, которые охватывают несколько сервисов.
3. Отсутствие автономности сервисов.
4. Общие схемы данных. Использование общей базы данных или общей схемы между сервисами.

Причины появления распределенного монолита

1. Недостаточное понимание микросервисных принципов.
2. Излишняя гранулярность микросервисов.
3. Общие ресурсы. Использование общих баз данных (Shared Database).
4. Плохой дизайн API.

Модульный и распределенный монолит

Критерий	Модульный монолит	Распределенный монолит
Развертывание	Один артефакт	Много сервисов, но деплой связан
Коммуникация	Локальные вызовы	Сетевые вызовы (HTTP/RPC)
Данные	Общая БД, но разделенные схемы	Общая БД или жесткие контракты
Автономность	Модули зависят друг от друга	Сервисы зависят друг от друга
Стоимость изменений	Низкая (рефакторинг в рамках одного кода)	Высокая (нужно координировать несколько сервисов)

Микросервисы

Определение

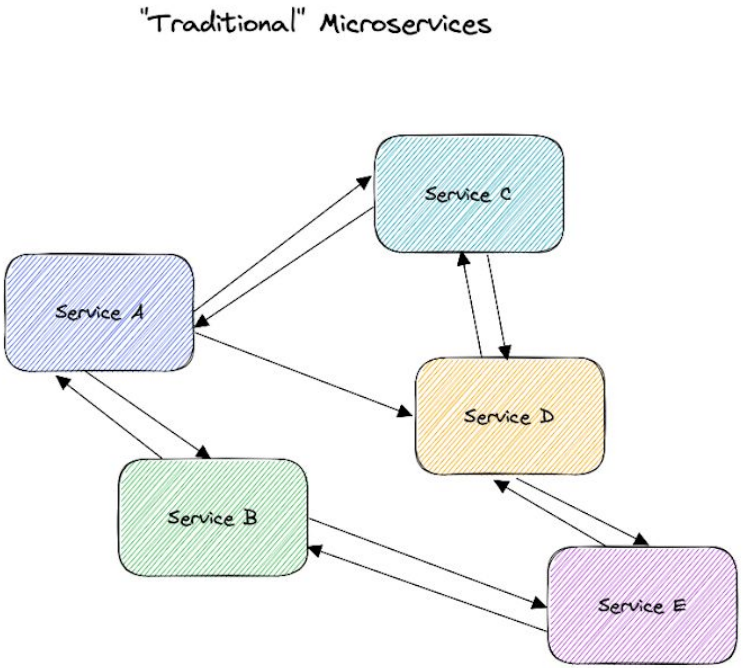
Архитектурный стиль микросервисов — это подход, при котором единое приложение строится как набор ***небольших сервисов****, каждый из которых работает* ***в собственном процессе*** *и коммуницирует с остальными используя легковесные механизмы, как правило HTTP.*
Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием ***полностью автоматизированной среды****. Сами по себе сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.*

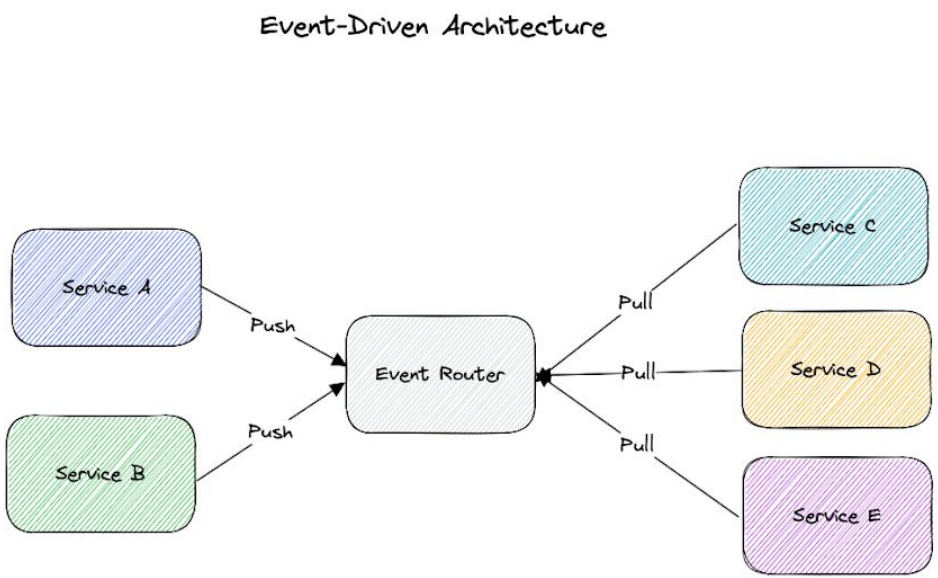
Микросервисы: принципы

Table 1 Relationship among microservices lifecycle stages, principles, features, and tools

Stage	Principle	Example features	Tools/practices
Design	Modeled around business domain	Contract, business, domain, functional, interfaces, bounded context, domain-driven design, single responsibility	Domain-driven design (DDD), bounded context
Design	Hide implementation details	Bounded contexts, REST, RESTful, hide databases, data pumps, event data pumps, technology-agnostic	OpenAPI, Swagger, Kafka, RabbitMQ, Spring Cloud Data Flow
Dev	Culture of automation	Automated, automatic, continuous*(deployment, integration, delivery), environment definitions, custom images, immutable servers	Travis-CI, Chef, Ansible, CI/CD
Dev	Decentralize all	DevOps, Governance, self-service, choreography, smart endpoints, dumb pipes, database-per-service, service discovery	Zookeper, Netflix Conductor
Dev/ Ops	Isolate failure	Design for failure, failure patterns, circuit-breaker, bulkhead, timeouts, availability, consistency, antifragility,	Hystrix, Simian Army, Chaos Monkey
Ops	Deploy independently	versioning, one-service-per-host, containers	Docker, Kubernetes, canary|A/B|blue/ green testing
Ops	Highly observable	Monitoring, logging, analytics, statistics, aggregation	ELK, Elasticsearch, Logstash, Kibana

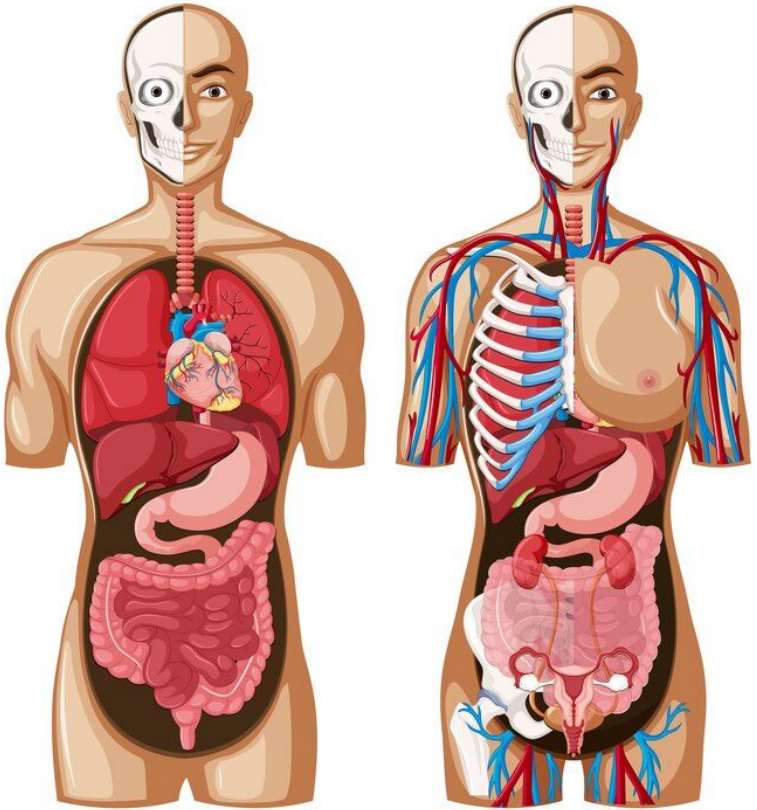
EDA


"Traditional" Microservices
Service C
Service A
Service D
Service B
Service E


Event-Driven Architecture
Service A
Push
Service B
Push
Event Router
Pull
Pull
Pull
Service C
Service D
Service E

Аналогии

Какой архитектурный стиль у человека?



Что выбираем?

Этап	Цели	Рекомендуемая архитектура
Стартап (MVP)	Проверить гипотезу, быстро выпустить продукт.	?
Рост (появление пользователей)	Масштабировать функционал, обеспечить стабильность.	?
Масштабирование	Обрабатывать высокие нагрузки, изолировать сбои.	?
Зрелость	Оптимизация, глобальная доступность.	?

Основные аспекты SD

1. Масштабируемость
2. Производительность
3. Надежность
   a. Отказоустойчивость
   b. Доступность
4. Безопасность
5. Адаптивность
6. Управляемость и мониторинг
7. Интеграции

Основные сущности

1. Клиент
2. Сервис
3. Интеграция
4. База данных

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ