Т БАНК

Сравнительный анализ **монолитного** и **микросервисного** подходов к разработке web приложений в условиях низкой загрузки

Егоров Гордей

Предпосылки

Начало любого проекта - выбор архитектуры

В веб разработке популярны:

Микросервисная vs Монолитная

Цель исследования:

Использовать разработанные мной приложения

Выделить критерии сравнения для низкой загруженности

Провести сравнительный анализ архитектур

Типы архитектуры

Монолитная

традиционный подход

весь функционал концентрируется в одном приложении или модуле

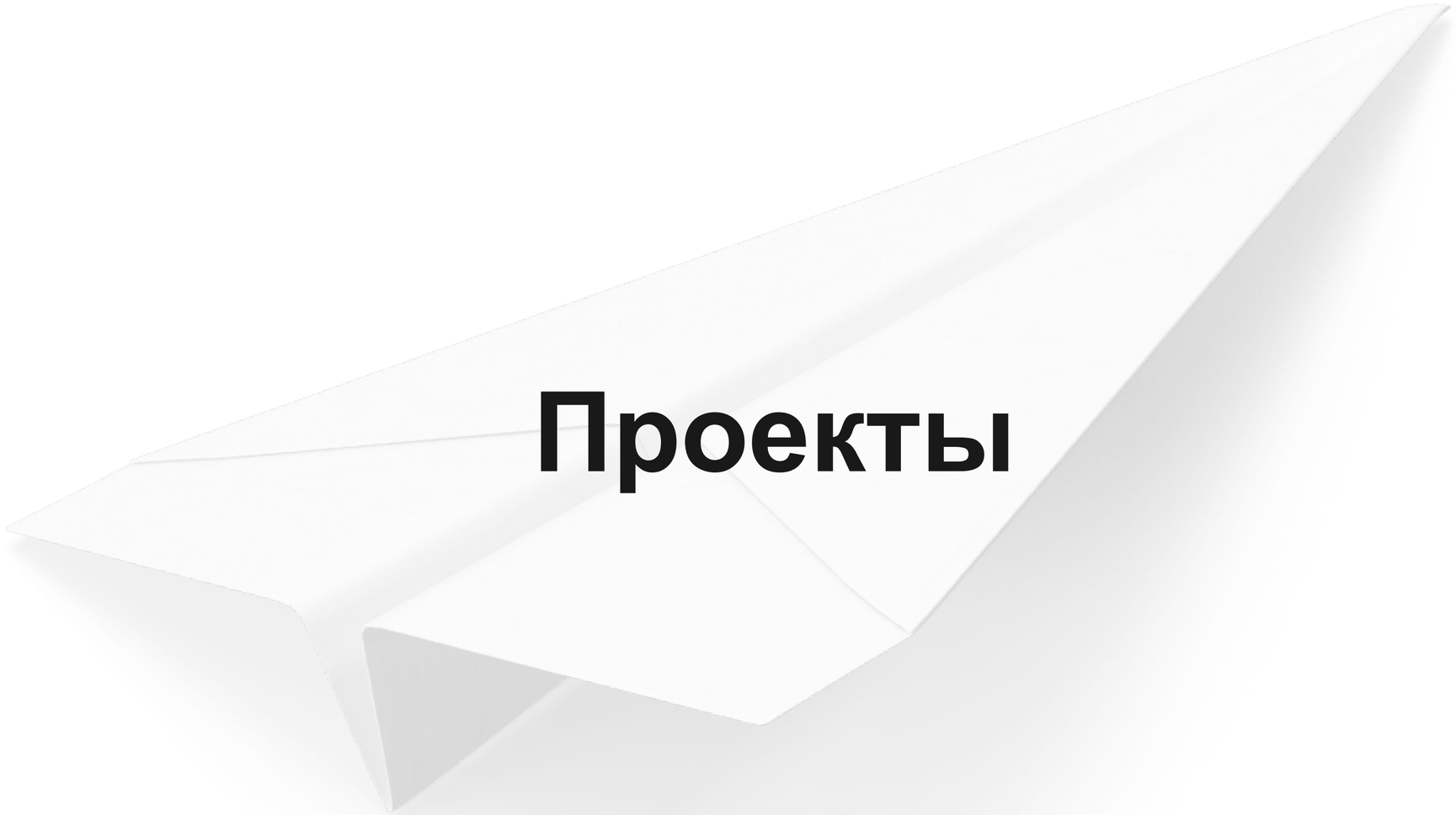
компоненты взаимодействуют напрямую

Микросервисная

микросервисы

разрабатываются и развертываются независимо

отвечают за конкретную функциональность

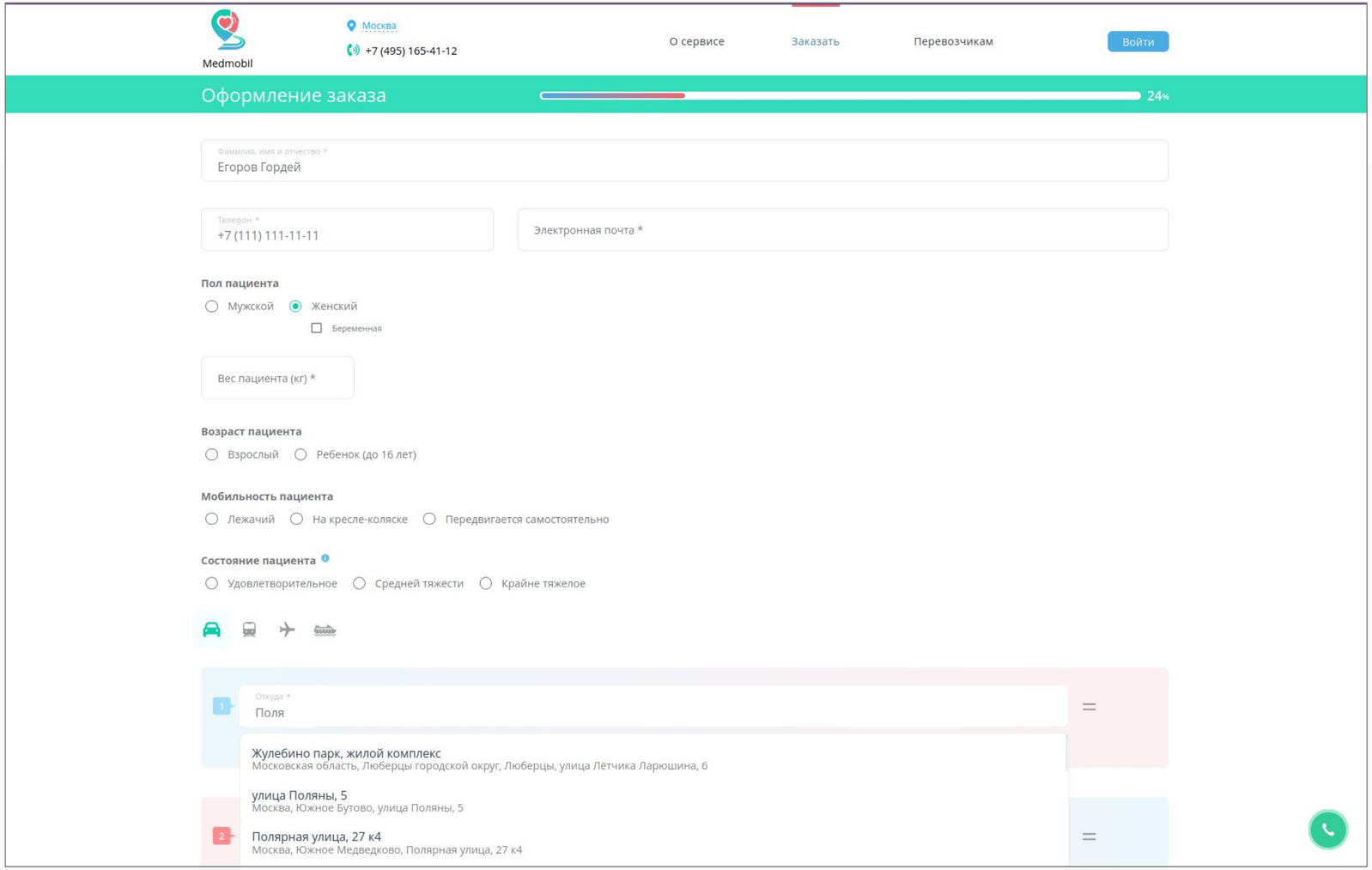


Проекты

Разработанные приложения

Медицинское такси “MedMobil”

- язык бэкенда - Java
- авторизация через keycloak, но ролевая модель своя
- монолитная архитектура
- запуск нативный, без контейнеров

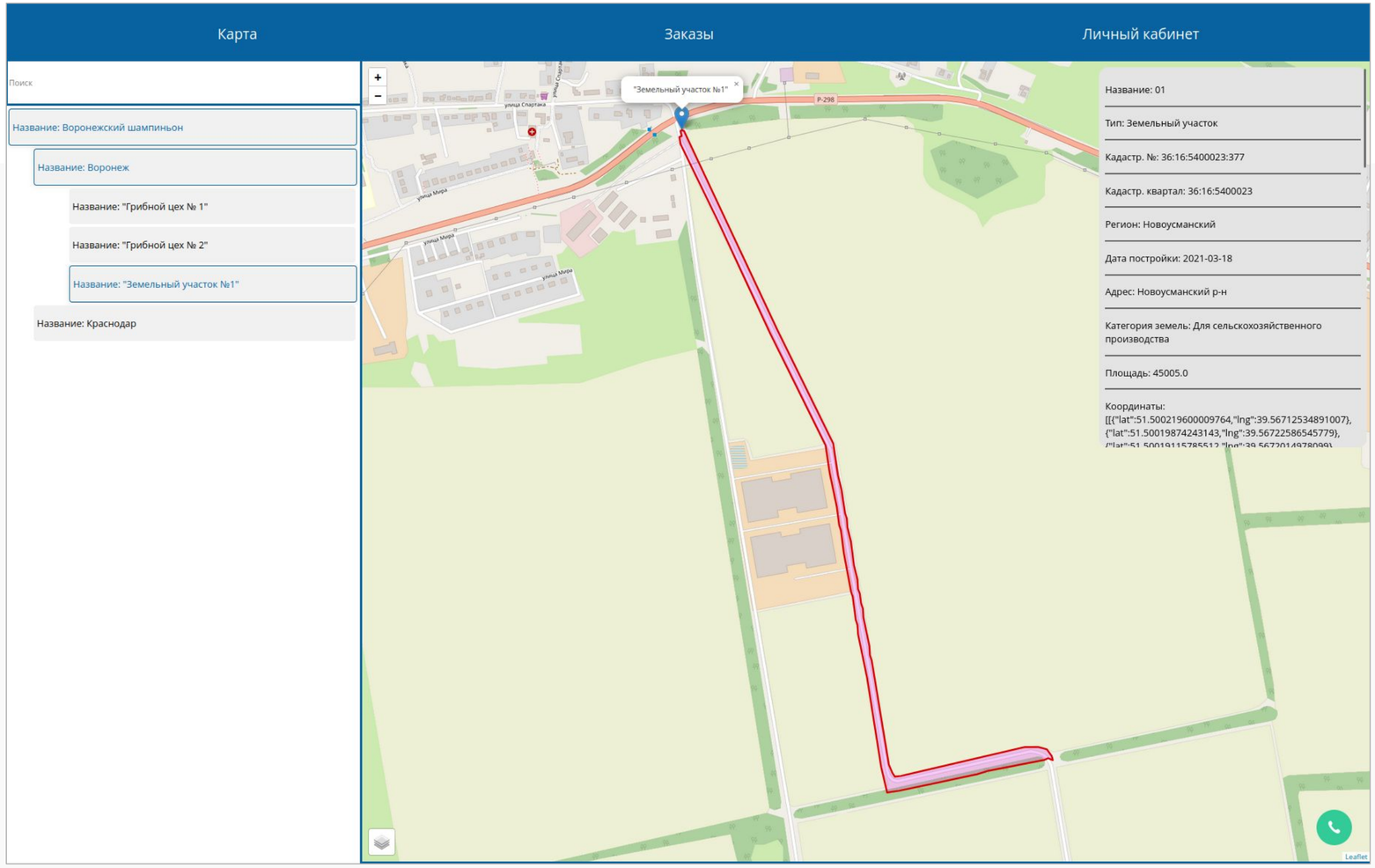


Разработанные приложения

Система структурирования данных “ЦИК”

- язык бэкенда - Java
- ролевая модель через keycloak
- микросервисная архитектура

  Eureka - service discovery
  ApiGateway - точка входа
  отдельные сервисы авторизации, данных компании, …
- docker compose



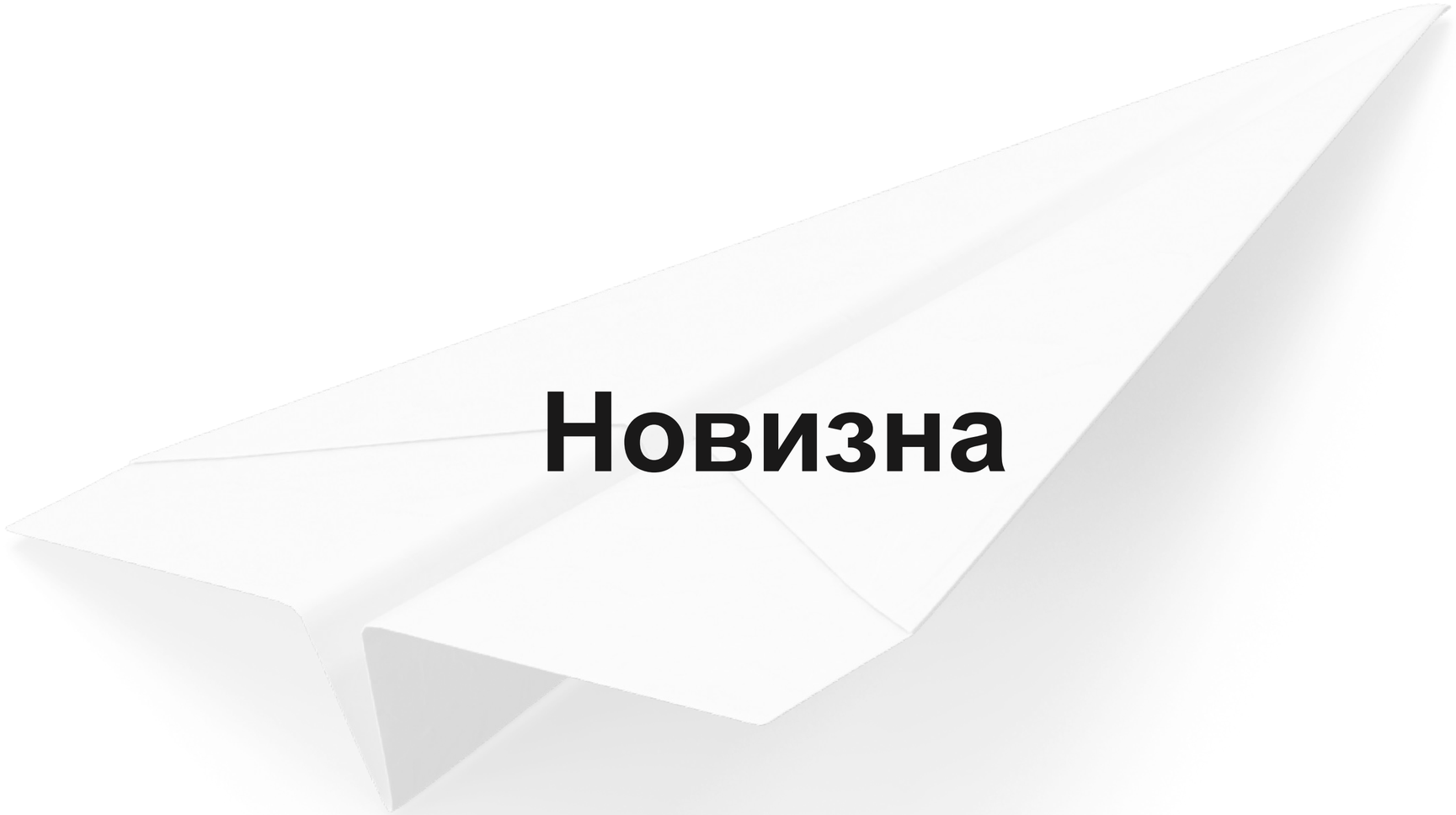
Корректность

Проекты разработаны одинаковой командой

Похожие стэки (Java, PostgreSQL, Keycloak)

Схожий высокоуровневый функционал (запросы к БД, промежуточная логика)

Сравнение корректно



Новизна

Существующие исследования

- “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”
  Сравнивает масштабируемость
- “A Comparative Review of Microservices and Monolithic Architectures”
  Сравнивает производительность при распараллеливании:
  “...concurrency testing showed better performance in throughput by 6%...”
- “Migrating from monolithic architecture to microservices: A Rapid Review”
  Концентрируется на миграции от монолита к микросервисам

Не рассматриваются проекты с низкой загруженностью

Сравнительные характеристики

~~Масштабирование~~

~~Производительность~~

~~Отказоустойчивость~~

~~Надёжность~~

~~Переиспользуемость~~

~~Юридические вопросы~~

Команда

Скорость разработки

Совместимость

Переносимость

Наблюдаемость

Потребление ресурсов

Анализ

Численность команды

Монолит

- ❏ фронтендер
- ❏ бэкендер
- ❏ тимлид

Микросервисы

- ❏ фронтендер
- ❏ бэкендер (+ разработчик БД)
- ❏ тимлид
- ❏ девопс
- ❏ архитектор микросервисов

Скорость разработки

Монолит

- MVP с нуля - 2 месяца
- отдан заказчику - 7 месяцев

Микросервисы

- первый этап - 2 месяца
- MVP - 4 месяца (с привлечением дополнительного архитектора)
- спустя 8 месяцев передан в дальнейшую разработку

Совместимость и переносимость

Совместимость - взаимодействие между развернутыми компонентами
Переносимость - способность ПО работать на разных платформах

- Монолит:
  - взаимодействие с компонентами решается созданием классов
  - вызов быстрый, прямой
  - запускается напрямую - смена сервера зависит от ОС, окружения (локаль, версия java и т. п.)
- Микросервисы:
  - взаимодействие с сервисами решается новым сервисом
  - вызов через обращение к сервису долгий
  - запускается через docker - универсально

Наблюдаемость

Наблюдаемость - мониторинг; понимание, что происходит в программе в каждый момент времени

- Монолит:
  - логирование естественным образом в одном месте
  - дебаг с помощью одного дебаггера
  - возможен доступ напрямую между классами произвольным образом
- Микросервисы:
  - логирование разобщено
  - дебаг с помощью нескольких логгеров / дебаггеров
  - интерфейсы микросервисов обеспечивают изолированность данных

Потребление ресурсов

- Монолит:
  - всё приложение весит больше, но не несёт оверхедов
  - 550 мб оперативной памяти
  - отсутствие копий данных в БД
- Микросервисы:
  - сервис легче всего приложения, но их много
  - 400 Мб оперативной памяти каждый, итого 400*13~5.2 Гб
  - копирование данных в БД для ускорения выполнения запросов

x2.2 доп памяти

Итоги

Тип сравнения	Микросервисы	Монолит
Скорость разработки	дольше	быстрее
Совместимость	медленное	прямое взаимодействие
Переносимость	докер, разделимость	зависимость от среды
Потребление ресурсов	каждый отдельно - меньше в целом - больше	большой, но один
Наблюдаемость	больше логических узлов	прямое взаимодействие

Продолжение исследования: разработать промежуточный вариант с тем же стэком, совмещающий плюсы обоих проектов (например, “модульный монолит”)

Т БАНК

Спасибо!