1. 15

Всем добрый день. Меня зовут Егоров Гордей, сегодня я представлю презентацию на основе моего диплома на тему "Сравнительный анализ монолитного и микросервисного подходов к разработке web приложений в условиях низкой загрузки"

2. 30

[клик]

Начало любой разработки - это выбор архитектуры проекта

[клик]

В веб-разработке два основных подхода - это микросервисная и монолитная архитектура

[клик]

Цель данной дипломной работы - использовать разработанные мной и моей командой приложения, и сконцентрировавшись на сравнении в условиях низкой загруженности, провести сравнительный анализ этих архитектур

3. 40

Но прежде охарактеризую сами центральные для моей презентации понятия

Монолитная архитектура - традиционный способ построения приложения, при котором весь функционал собирается в одном приложении или модуле, и все компоненты, функции, классы взаимодействуют друг с другом напрямую а рамках одного процесса

Микросервисная архитектура, в противовес монолитной - современный подход, использующий разделение логически разных частей приложения на физически разные компоненты - микросервисы. Каждый микросервис разрабатывается и разворачивается независимо, общаются между собой чаще всего через http запросы

4. 15

Теперь опишу актуальность и новизну исследования. Для начала - актуальность, я расскажу про два проекта, которые я и моя команда из 3 программистов разработали, для которых были выбраны разные архитектуры - что покажет, что вопрос о выборе архитектуры актуален в современной разработке

5. 30

Собственно, сами проекты

Первое - Медицинское такси "MedMobil". Сайт, позволяющий обычным пользователям создавать заказы на поездки в стиле Яндекс Go, а также позволяет регистрироваться как перевозчики и диспетчеры для отслеживания заказов и управления статусами заказов. Написан на Java, авторизация через сторонний сервис keycloak (то есть присутствует связь с внешним сервисом), монолитная архитектура, запуск нативный, без контейнеров, просто устанавливается линукс, джава и всё запускается

6. 25

Второе - Система структурирования данных "ЦИК". Сайт разработан для сбора, хранения и структурирования данных с различных производств с целью их унификации и переиспользования. Написан тоже на Java, авторизация также реализована через Keycloak, но, в отличии от предыдущего проекта - микросервисная архитектура с 8 микросервисами, запускающаяся через docker compose

7. 25

Но корректно ли сравнивать эти два приложения? Конечно, они отличаются, но в своей основе они очень похожи:

[клик]

разработаны одинаковой командой,

[клик]

используются похожие стэки

[клик]

и схожая модель поведения.

[клик]

Следовательно, сравнительный анализ архитектур на примере данных проектов допустим

8. 5

Теперь что касается новизны

9. 35

Разумеется, уже существуют исследования, сравнивающие микросервисную и монолитную архитектуры. Но все они отличаются от данного в ключевой детали. Для демонстрации я выбрал три типичных представителя исследований на данную тему:

[клик]

первый тип исследований концентрируется на масштабируемости

[клик]

второе тип сравнивает производительность при распараллеливании

[клик]

третий тип вообще разбирает переход от монолита к микросервисам

[клик]

Все эти исследования концентрируются на высоконагруженных сервисах, не затрагивают анализ архитектур в проектах с низкой загруженностью - где нет тысяч пользователей и террабайт данных.

10. 20

Поэтому, концентрируясь на низкой загруженности, мы не будем рассматривать такие сравнительные характеристики как

[клик]

Масштабирование, Производительность, Отказоустойчивость и прочие типичные показатели

Но сфокусируемся на оставшихся, о которых будет подробнее рассказано в оставшейся части презентации

---4:00— (если быстро, то 3:10)

11. 5

Собственно, сам анализ

12. -

13. 30

Простейший параметр - скорость разработки. В монолитном проекте MVP мы создали за 2 месяца, и проект был передан заказчику через 7 месяцев. В микросервисном же проекте мы затратили на разработку в принципе минимально рабочего примера 2 месяца, и ещё два месяца нам понадобилось для доведения проекта до MVP, и это с привлечением дополнительного архитектора баз данных. И только спустя 8 месяцев проект был доведён до промежуточной стадии и передан в дальнейшую разработку

14. 55

Следующие понятия требуют определения.

Совместимость - это взаимодействие между развёрнутыми компонентами ПО. Переносимость - способность ПО работать на разных платформах с минимальными изменениями

[клик]

В случае монолита, совместимость осуществляется созданием нового класса, к которому остальные компоненты обращаются напрямую - в результате связь быстрая и надёжная. Переносимость в классическом случае ограничена, поскольку вызов нативный - в нашем случае много проблем принёс переход с ubuntu18 на ubuntu20 ввиду смены сервера

[клик]

В случае микросервисов, взаимодействие с сервисами решается созданием нового микросервиса, который общается с другими через их API. В результате связь долгая. Переносимость осуществляется за счёт докера, который позволяет разворачивать приложение в практически любых условиях

Следует отметить, что, в целом, возможно использование докера и в монолитном проекте, но это уходит в сторону будущих исследований

15. 40

Наблюдаемость - это мониторинг приложения в целом, так и понимание того, что происходит в программе в каждый момент времени, в том числе сюда входет и отладка

[клик]

В монолите логирование происходит естественным образом в одном месте и дебаг происходит с помощью одного дебаггера. Также возможен доступ напрямую между классами, отчего, с одной стороны, упрощается процесс отладки, но с другой стороны в коде появляется много логических связей, что усложняет и запутывает код

[клик]

В микросервисах логирование разобщено, хотя и конфигурируется. Процесс отладки усложняется необходимостью запускать несколько дебаггеров, или следить за поведением нескольких сервисов сразу. Тем не менее, в результате логической и физической изолированности компонентов и их данных взаимодействие более ясное и простое для восприятия

16. 30

Потребление ресурсов я тоже включил в исследование. Несмотря на то, что обычно про него говорят в контексте масштабируемости, для небольших проектов очень важно оценивать изначальные инвестиции и лёгкость образов для локальной разработки

[клик]

В результате тестирований, оказалось, что монолитное приложение в целом весит больше, но не несёт оверхедов вроде докера и копий данных в БД

[клик]

В микросервисном приложении сами сервисы легче, то есть их можно разворачивать на более дешёвых серверах, но их существенно больше

17. 15

Подводя итоги, в табличке отмечены сравнительные плюсы и минусы разных типов архитектур. В следующих исследованиях можно разработать промежуточный вариант архитектуры для приложения с тем же стэком, который бы совмещал плюсы обоих подходов. Например, можно посмотреть в сторону модульного монолита

18. 5

Спасибо за внимание

--7:00— (если быстро, то 5:50)