нетология

Введение в микросервисы

Михаил
Триполитов

Михаил Триполитов

Technical Lead

Михаил Триполитов

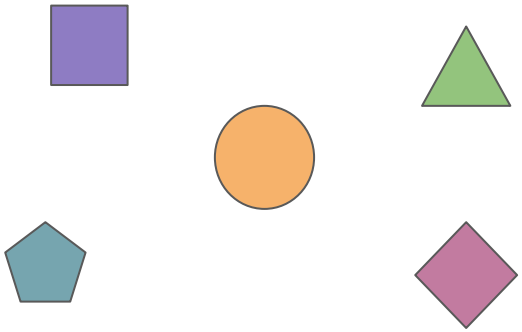
План занятия

1. Микросервисы
2. Преимущества
3. Сопутствующие проблемы
4. Антипаттерны
5. Сложные решения
6. Когда не стоит использовать?
7. Итоги
8. Домашнее задание

Микросервисы

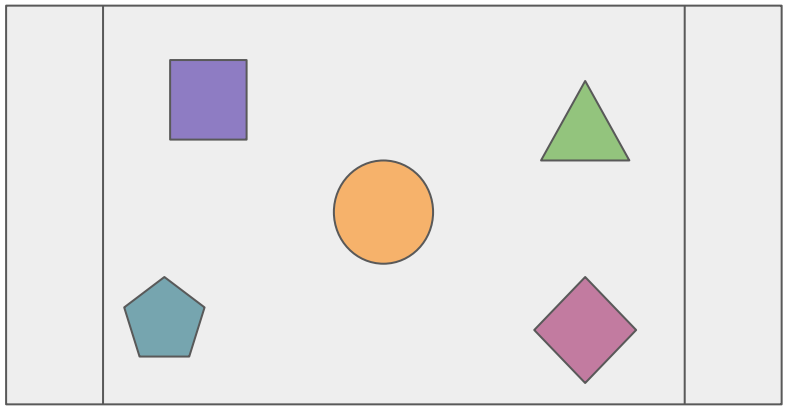
Что такое микросервисы?

Микросервисы — это архитектурный подход разделения системы на небольшие автономные сервисы, которые запускаются как отдельные процессы и взаимодействуют, используя API на основе легких протоколов, например, HTTP.



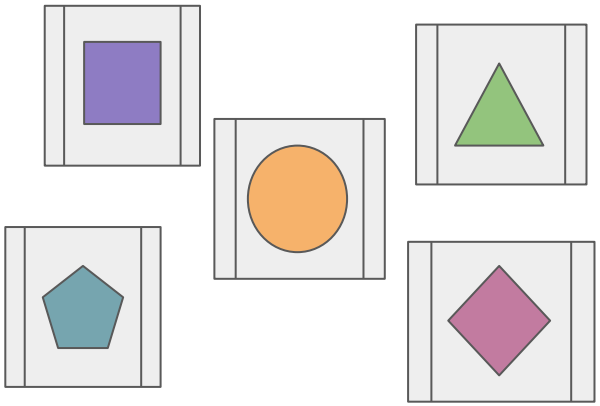
Что такое микросервисы?

Монолитное приложение содержит все бизнес-функции в одном процессе.



Что такое микросервисы?

Микросервисы распределяют бизнес-функции по разным независимым сервисам.



Характеристики микросервисов

Слабая связность → команда работает независимо от изменений в других сервисах

Независимы при выкладке → нет необходимости координировать выкладку с другими командами

Поддерживаемость и тестируемость → быстрая разработка и частые выкладки

Наличие кросс функциональной команды-владельца → сокращает расходы на коммуникации

Организация вокруг бизнес функций → глубокое понимание домена

Размер

Объединяйте вместе функциональность, изменяющуюся по одной причине, разделяйте функциональность, изменяющуюся по разным причинам. Это принцип единой ответственности.

Чем меньше сервисы, тем больше влияние преимуществ и негативных последствий микросервисной архитектуры.

Чем меньше сервисы, тем более они независимы друг от друга, но выше общая сложность системы.

Зачем использовать

Микросервисы повышают скорость, увеличивают частоту и надежность внесения изменений в крупные ИТ системы при часто меняющихся бизнес требованиях.

Обладает преимуществами при построении следующих систем:

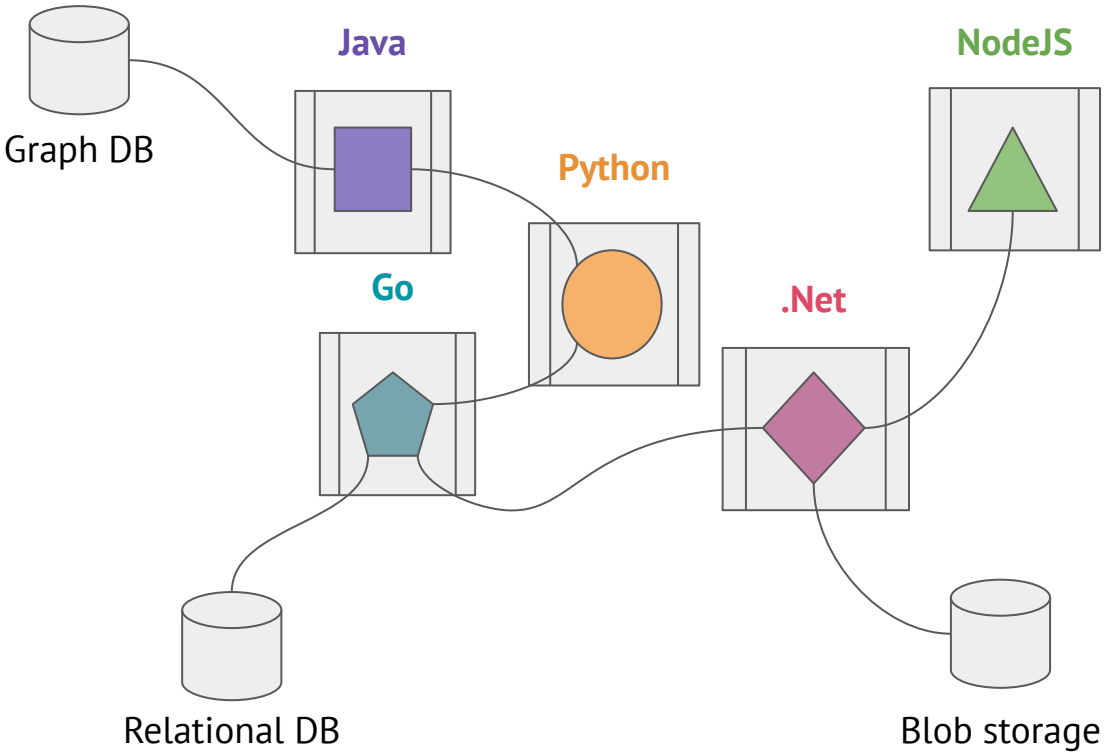
- с большим количеством интеграций
- с часто меняющейся неоднородной нагрузкой
- обеспечивающие работу различных бизнес подразделений

Преимущества

Преимущества

- Возможность использовать разные технологии
- Устойчивость к ошибкам
- Масштабируемость
- Простота развертывания
- Простота замены
- Отражение структуры организации

Разные технологии



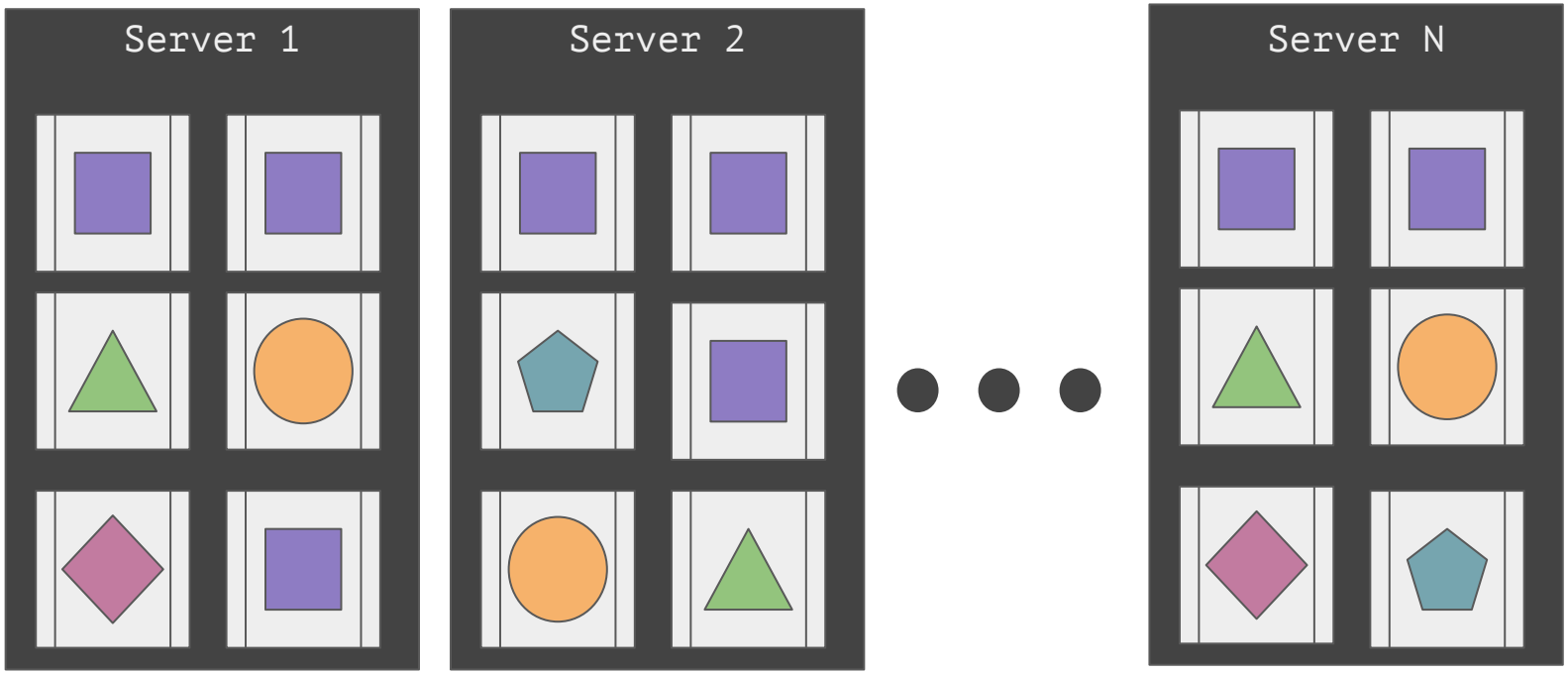
Устойчивость к ошибкам

Техники работы с ошибками:

- **Таймаут** → не занимать лишние ресурсы
- **Прерыватель цепи** → снизить нагрузку на сервисы, испытывающие проблемы
- **Повтор** → получить ответ, не смотря на сетевые проблемы
- **Идемпотентность** → исключить бизнес ошибки при повторах
- **Переборка** → снизить влияние разного функционала друг на друга
- **Изоляция** → снизить зависимость от других сервисов
- **Распределение нагрузки** → обеспечить ресурсами критичные процессы
- **Балансировка** → исключить единую точку отказа

Масштабируемость

Сервисы распределяются между серверами в зависимости от потребностей



Простота развертывания

- Небольшие изменения
- Частые выкладки
- Низкие риски
- Независимость

Простота замены

Отражение структуры организации

«Организации проектируют системы,
которые копируют структуру коммуникаций
в этой организации»

Закон Конвея

Сопутствующие проблемы

Проблемы разработки

- Совместимость API
- Версионирование артефактов
- Автоматизация сборки и тестирования
- Документация
- Инфраструктура разработки

Проблемы эксплуатации

- Мониторинг
- Сбор логов
- Управление настройками
- Управление инфраструктурой

Антипаттерны

Антипаттерн: серебряная пуля

Нет

Пытаться решить все проблемы разработки применением микросервисов

Да

Микросервисы - архитектурный стиль, который может помочь повысить скорость, частоту и надежность релизов

Антипаттерн: самоцель

Нет

Делать внедрение микросервисов целью, по которой измеряется успех разработки ИТ системы

Да

Цель - увеличить скорость, частоту и качество поставки

Хорошие метрики

- **Время выкладки** - время от коммита до выкладки
- **Частота выкладки** - количество выкладок в день на одного разработчика
- **Интенсивность отказов** - количество неуспешных выкладок
- **Время восстановления** - время восстановления после отказа

Антипаттерн: наносервисы

Нет

Создавать большое количество очень маленьких сервисов

Да

Один сервис на команду

Сложные решения

Как разделять систему на сервисы?

Low coupling and high cohesion:

- Небольшое количество внешних связей
- Решает близкие по смыслу задачи

Предметно-ориентированное проектирование:

Ограниченный контекст ⟶ сервис

Кто владелец сервиса?

Общий код - любой разработчик может изменить любой сервис и выложить его

По командам - только команда-владелец сервиса может изменить сервис и выложить

Общий код, но есть владелец сервиса - любой разработчик может изменить любой сервис, но выложить можно только по согласованию с владельцем сервиса

Взаимодействие синхронное или асинхронное?

Синхронное (Request/Response)

Да - простота понимания

Да - простота отладки и реализации

Нет - балансировка производительности

Нет - риск каскадных отказов

Нет - система балансировка нагрузки

Нет - организация Service Discovery

Асинхронное (Event-based)

Да - устойчивость к пиковым нагрузкам

Да - балансировка нагрузки за счет брокеров очередей

Да - слабая связность системы

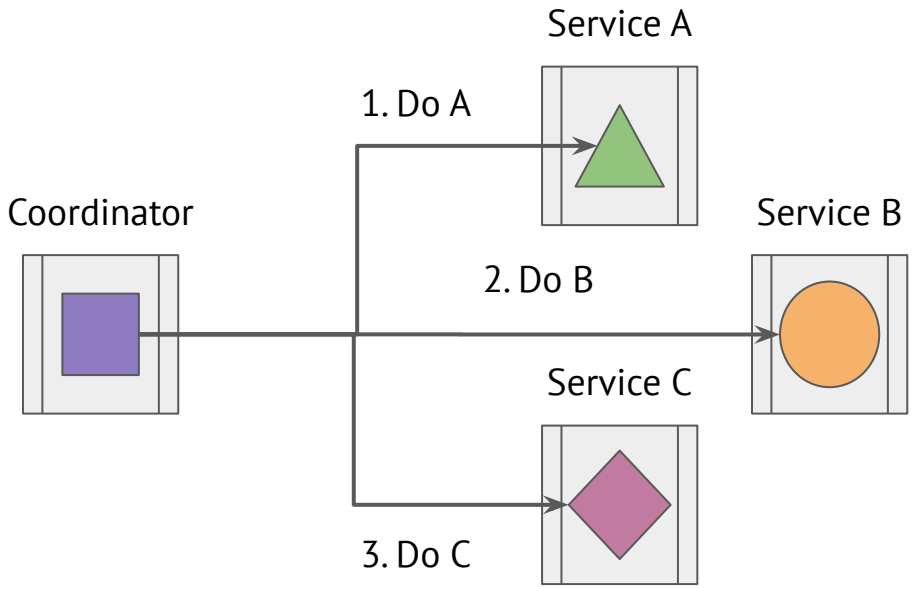
Нет - общая высокая сложность системы

Нет - запросы на чтение требуют дополнительных посредников

Оркестрация или хореография?

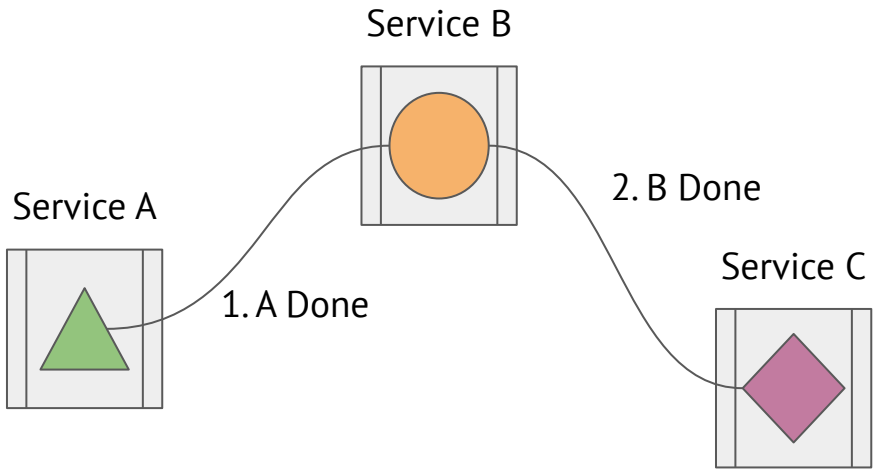
Оркестрация

Отдельный координатор управляет сервисами указывая какие операции выполнять в какой момент времени.



Хореография

В процессе выполнения операции каждый сервис публикует события, которые запускают операции в других сервисах.



Протоколы интеграции?

- **REST** - производительный, масштабируемый, простой
- **Grpc** - легкий, эффективный
- **GraphQL** - адаптивный, эффективный, гибкий
- **JSON-RPC -** легкий, понятный

Подход к безопасности

- **No authentication** - доверять всем запросам внутри периметра
- **HTTP(S) Basic Authentication -** передавать логин и пароль в заголовках запроса
- **Aouth2 и OpenID Connect** - использовать SSO для межсервисного взаимодействия
- **Client Certificates and mutual TLS** - каждому сервису выпускать свой клиентский сертификат
- **HMAC over HTTP** - подписывать каждый HTTP запрос секретным ключем
- **API Keys** - каждому сервису выпускать API ключ по которому определять его права

Когда не стоит использовать?

Когда не стоит использовать?

- **Незнакомая предметная область** - чем меньше вы понимаете предметную область, тем сложнее найти границы контекстов
- **Не определена структура организации** - разделение на сервисы не принесет выгоды, если не будет соответствовать разделению на команды.
- **Система с чистого листа** - гораздо легче делить существующую систему на микросервисы, чем пытаться разделить на сервисы то, чего нет.

Итоги

- Узнали, что такое микросервисы и чем они отличаются от монолитной системы
- Узнали, какие основные преимущества микросервисов
- Узнали о сложностях, с которыми придётся столкнуться и о решениях, которые придётся принять
- Разобрали ситуации, когда не стоит использовать микросервисы

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Михаил Триполитов

Михаил Триполитов