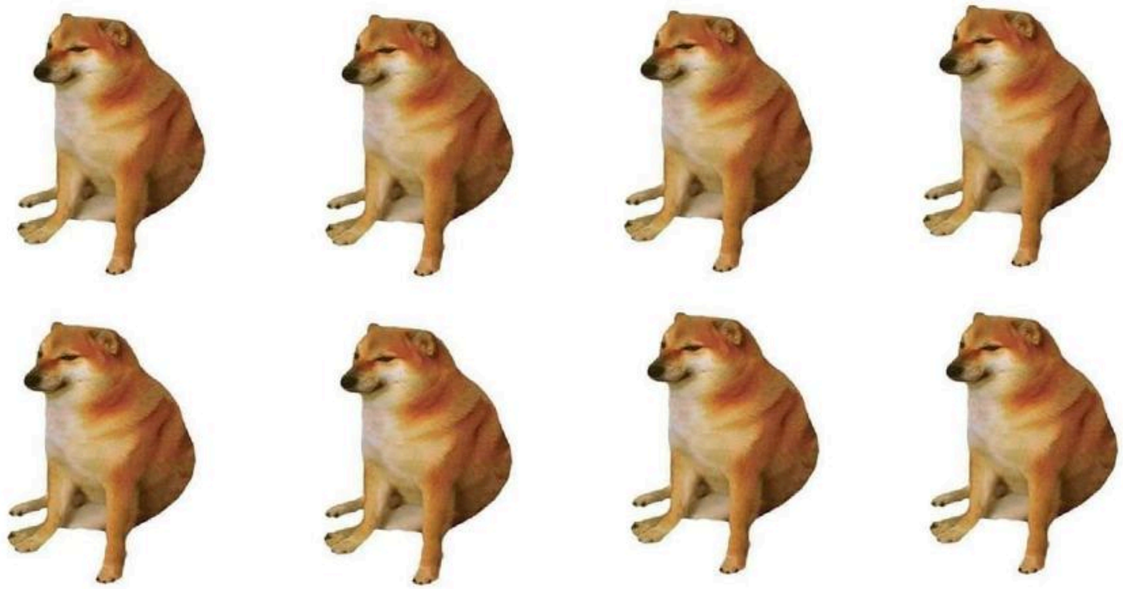
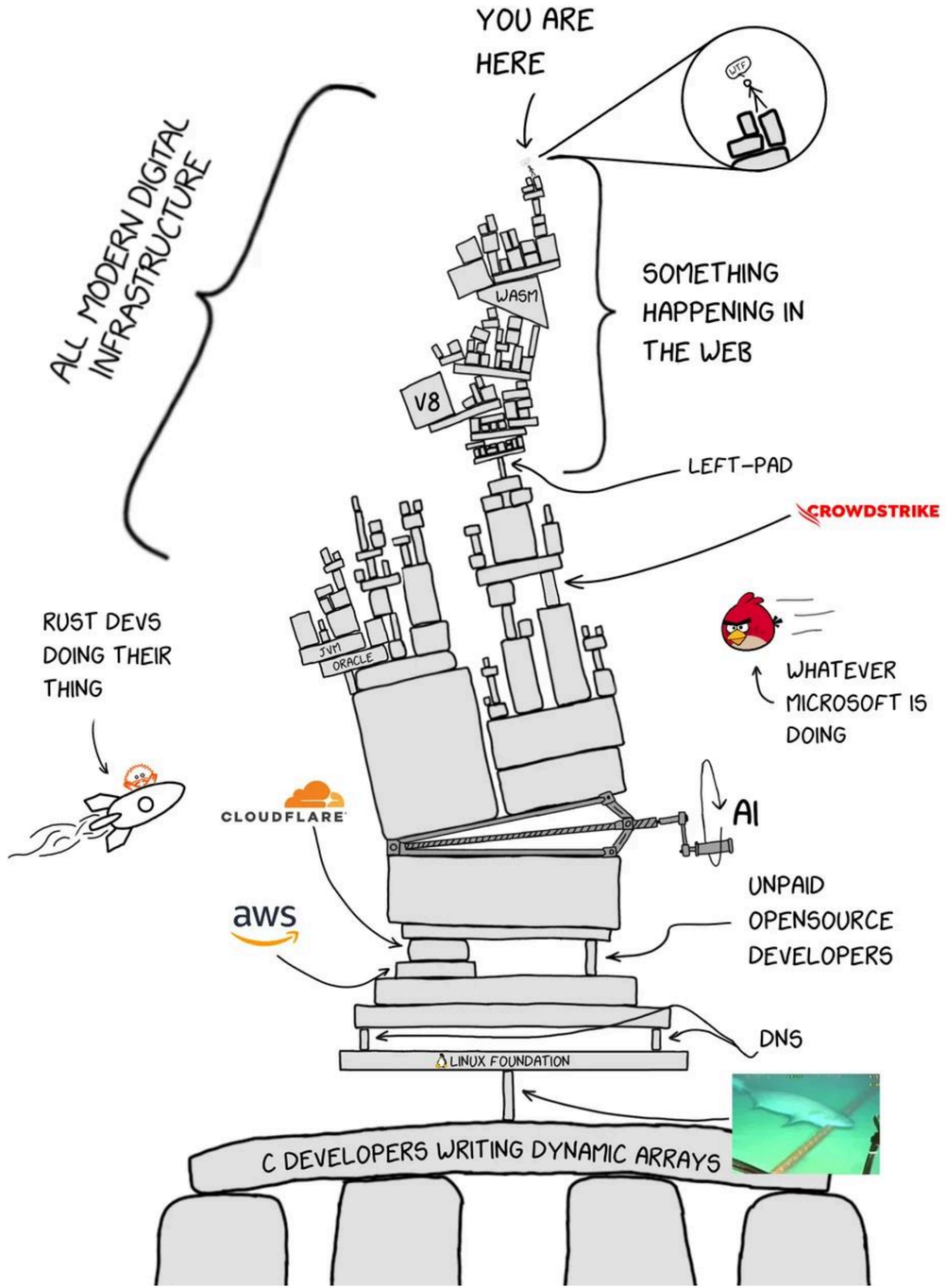
Монолитная и микросервисная архитектура сервисов

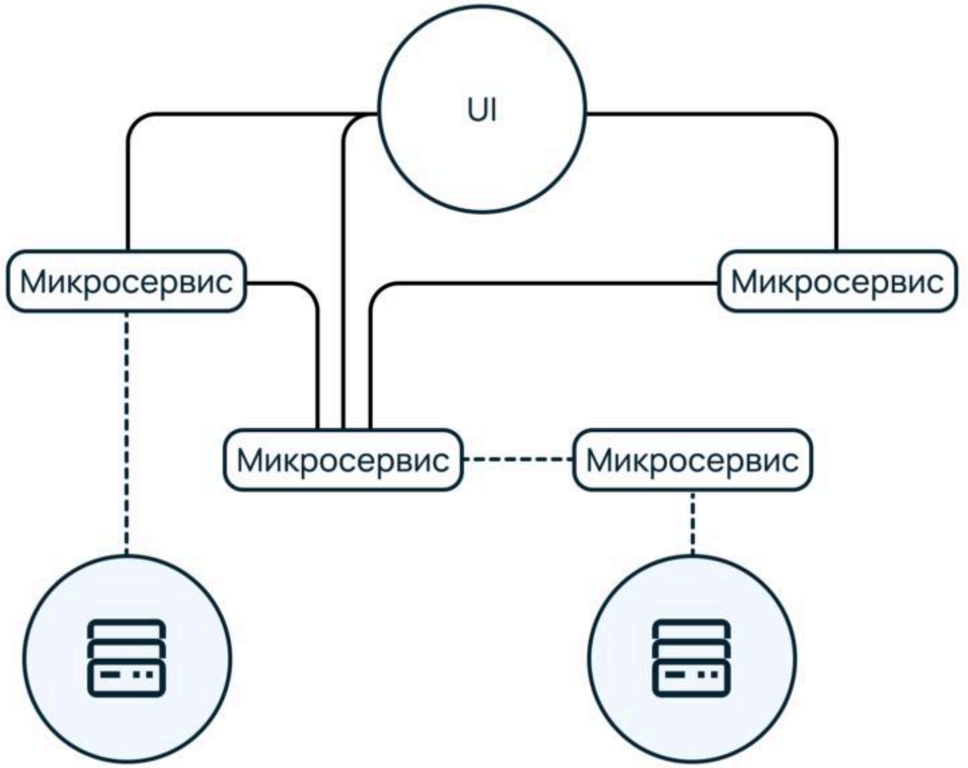


Что такое архитектура сервиса?



Монолит

Микросервисы



Единая кодовая база	Весь код приложения находится в одном репозитории и собирается в один исполняемый файл. Все модули (авторизация, заказы, оплата) работают в рамках одного процесса
Общая база данных	Все модули приложения используют одну БД. Это упрощает работу с транзакциями и согласованностью данных, но создаёт тесную связанность между частями системы
Централизованное управление	Одна команда отвечает за весь проект, используется единый стек технологий. Все зависимости и версии библиотек общие для всего приложения
Единое развёртывание	Приложение деплоится целиком как один артефакт. Любое изменение — даже в одном модуле — требует пересборки и перезапуска всего приложения

Преимущества	Недостатки и сложности
Простота разработки на старте: весь код в одном месте, легко ориентироваться и отлаживать	Сложность масштабирования: нельзя масштабировать отдельные части, только всё приложение целиком
Простое развёртывание: один артефакт, минимальная инфраструктура, не нужен Kubernetes и оркестрация	Медленные релизы при росте: любое изменение требует пересборки всего приложения, что занимает много времени
Высокая производительность: вызовы между модулями происходят внутри одного процесса без сетевых задержек	Технологическая зависимость: весь монолит привязан к одному языку и фреймворку, смена стека требует переписывания всего проекта
Простота тестирования: легко писать end-to-end тесты, не нужно поднимать множество сервисов	Риск полного отказа: критическая ошибка в одном модуле может положить всё приложение
Надёжная работа с транзакциями: полная поддержка ACID-транзакций в одной БД без дополнительных паттернов	Рост сложности со временем: большая кодовая база становится трудно поддерживаемой, команды начинают мешать друг другу

Принцип	Описание
Независимость	Каждый микросервис работает автономно, в отдельном процессе. Это значит, что сбой в одном сервисе не влияет на работу всей системы и снижает риск глобального отказа. Также у каждого микросервиса собственная база данных — это позволяет сохранять автономность данных и избегать ситуаций, когда при изменении схемы в одном сервисе нужно одновременно корректировать схемы во всех остальных
Децентрализация	Команды разработки выбирают различные технологии под конкретные задачи, например, Java, Python™, Go или Node.js, ускоряя разработку и повышая эффективность, но усложняя поддержку. Команды полностью отвечают за разработку и эксплуатацию своих сервисов
API-коммуникация	Сервисы взаимодействуют между собой через различные сетевые интерфейсы и протоколы: REST, gRPC, GraphQL и Messaging — обмен сообщениями через очереди и брокеры сообщений, такие как Kafka® или RabbitMQ™. Использование Messaging позволяет снизить зависимость сервисов друг от друга
Непрерывная доставка	Независимое развёртывание и масштабирование сервисов ускоряет выпуск обновлений, снижает риски и повышает адаптивность к бизнес-требованиям. Для этого необходимы зрелая инфраструктура автоматизации (CI/CD, GitOps, Infrastructure as Code) и высокие стандарты инженерного качества

Преимущества подхода	Недостатки и сложности
Скорость разработки и вывода новых функций	Управление распределённой системой гораздо сложнее монолита. Требуются продвинутые DevOps-практики и развитая инфраструктура
Легко масштабируемые компоненты под изменяющиеся нагрузки	Нужна развитая инфраструктура мониторинга, логирования и трассировки для оперативного реагирования на сбои и неполадки
Снижение рисков отказов	Высокие требования к DevOps-культуре и инженерной зрелости команд. Высокие затраты на обучение сотрудников и создание инфраструктуры
Гибкость выбора технологий для разных частей системы	Сетевая задержка: множество межсервисных вызовов создают дополнительные проблемы производительности и могут замедлять работу приложения
Удобство обновления и поддержки: небольшие изменения можно вносить и тестировать изолированно, что упрощает управление кодом и уменьшает риски ошибок	Сложности с согласованностью данных из-за CAP-теоремы. Трудности с поддержкой ACID-транзакций требуют реализации сложных подходов (Saga, Event Sourcing)

https://github.com/Jahamars/microservices-example/

Тренд	Суть	Ценность для бизнеса
Serverless-микросервисы	Функции как сервис (FaaS) в виде «ультра-микро»-сервисов	Мгновенное масштабирование под нагрузки и оплата только по фактическому использованию
Zero Trust в кластере	Взаимная аутентификация сервисов по протоколу mTLS, централизованные политики безопасности (например, OPA)	Предотвращение горизонтального распространения атак внутри системы («боковое» перемещение атак)
WebAssembly-микро-ВМ	Лёгкие изолированные среды («песочницы») с мгновенным запуском, основанные на технологии WebAssembly	Поддержка периферийных вычислений и задач машинного обучения без контейнеров
Мульти-облака по умолчанию	Использование сразу нескольких облачных платформ	Избежание зависимости от одного поставщика и географическое резервирование
AI-Ops	Автоматизированный анализ логов и инцидентов с помощью LLM-ассистентов	Снижение нагрузки на команды разработки и поддержки при росте количества сервисов до сотен и тысяч