Примите простоту

Sylvain ARBAUDIE · January 9, 2025

ARCHITECTURE SIMPLICITY OPINION DEVOPS

Эпидемия сложности

ИТ-индустрия имеет проблему со сложностью. Не с той сложностью, которая присуща решаемым задачам — она неизбежна. Нет, я говорю о сложности, которую мы создаём сами: случайной сложности.

CRUD-проект на 10 000 пользователей, развёрнутый на Kubernetes с Istio, Prometheus, Grafana, ArgoCD, шиной событий Kafka, 12 микросервисами, 3 различными базами данных и service mesh. Почему? Потому что так делает Netflix. Потому что это «круто» в 2025 году.

Результат: команда из 5 разработчиков, которая тратит больше времени на управление инфраструктурой, чем на разработку функциональности.

Эволюция архитектур

Вернёмся назад. История программных архитектур — это история нарастающей сложности:

Монолит -> Одно приложение, один деплой, одна база данных. Просто, эффективно и идеально подходит для большинства проектов.

SOA (Service-Oriented Architecture) -> Сервисы, взаимодействующие через ESB (Enterprise Service Bus). Гибче монолита, но ESB становится единой точкой отказа и узким

местом.

Микросервисы -> Независимые сервисы, взаимодействующие через API. Теоретически каждый сервис может быть разработан, развёрнут и масштабирован независимо. На практике операционная сложность огромна.

EDA (Event-Driven Architecture) -> Сервисы обмениваются асинхронными событиями. Максимальная развязка, но отладка потока событий через 15 сервисов — это кошмар.

Каждый этап добавлял сложности. И на каждом этапе индустрия представляла новую архитектуру как универсальное решение. Спойлер: ни одна архитектура не является универсальной.

Принцип KISS

KISS — Keep It Simple, Stupid — принцип, который должен висеть на стене каждого офиса архитектора программного обеспечения.

Принцип не говорит «делайте просто, потому что вы не способны на большее». Он говорит: **сложность имеет стоимость, и эта стоимость должна быть обоснована**.

Каждый добавленный в архитектуру компонент — это:

- Ещё один компонент для поддержки
- Ещё одна точка отказа
- Ещё одна компетенция, необходимая в команде
- Ещё одна строка расходов на инфраструктуру
- Увеличенное время отладки

Kubernetes: идеальный пример

Kubernetes — выдающийся инструмент. Для оркестрации сотен контейнеров в масштабе Google, Netflix или Spotify он незаменим.

Для развёртывания приложения MariaDB / MySQL + PHP/Node.js с 5 000 пользователями? Это пушка для стрельбы по воробьям.

Выделенный сервер (или VPS) с Docker Compose делает эту работу за долю стоимости и сложности:

```
# docker-compose.yml — это всё, что вам нужно
services:
  app:
    image: myapp:latest
    ports:
      - "80:80"
    depends_on:
      - db
  db:
    image: mariadb:11.4
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MARIADB_ROOT_PASSWORD_FILE: /run/secrets/db_password
volumes:
  db_data:
```

Никакого кластера Kubernetes. Никаких Helm-чартов. Никакого service mesh. Никакого распределённого мониторинга. Просто два контейнера, которые делают свою работу.

Компании, которые возвращаются назад

Обратное движение уже идёт. Компании уходят от сложных архитектур к более простым подходам:

- **Basecamp/37signals** перенесли нагрузки из облака на выделенные серверы, экономя миллионы долларов в год
- **Amazon Prime Video** мигрировали сервис с микросервисов на монолит, сократив затраты на 90%
- **DHH** (создатель Ruby on Rails) активно выступает за «выход из облака»

Эти компании не технофобы. Они просто посчитали: сложность стоит дороже простоты, при равных функциональных возможностях.

Фундаментальный вопрос

Прежде чем выбрать технологию, задайте себе вопрос: **какую проблему я решаю?**

Не «какая технология в моде». Не «что впечатлит в моём резюме». Не «что используют FAANG-компании». Вопрос: какова конкретная проблема, и какое самое простое решение её решает?

Если ответ на «почему Kubernetes?» — «потому что так принято в 2025 году», у вас проблема с принятием решений, а не техническая проблема.

Когда сложность оправдана

Будем ясны: сложность иногда необходима.

Если у вас 10 миллионов пользователей с пиками нагрузки в 50 раз, Kubernetes оправдан. Если у вас 200 разработчиков на одном продукте, микросервисы обеспечивают автономию команд. Если вам нужно обрабатывать 100 000 событий в секунду, Kafka — правильный ответ.

Но эти случаи — исключение, а не норма. 90% веб-приложений не имеют таких ограничений. И для этих 90% хорошо построенный монолит с базой MariaDB / MySQL, развёрнутый на одном сервере (или двух для резервирования), не просто достаточен — он оптимален.

Мой манифест простоты

1. **Начинайте с монолита.** Разделяйте на сервисы, когда (и только когда) боль от монолита превысит боль от распределённости.
2. **Используйте то, что знаете.** Освоенный стек работает лучше, чем «крутой» стек, освоенный плохо.
3. **Считайте свои компоненты.** Если в вашей архитектуре больше компонентов, чем разработчиков в команде, — это тревожный сигнал.
4. **Измеряйте полную стоимость.** Инфраструктура + время разработки + время отладки + время обучения. Сложность редко бывает бесплатной.
5. **Задавайте вопрос «зачем?».** Для каждого компонента инфраструктуры спрашивайте «зачем он здесь?» Если ответ «на всякий случай» — удалите его.

Заключение

Лучшая архитектура — самая простая, которая решает задачу. Не самая впечатляющая, не самая модная, не самая полная. Самая простая.

Примите простоту. Ваша команда, ваш бюджет и ваши пользователи скажут вам спасибо.

Эта статья была первоначально опубликована на Medium.