Przyjmij prostotę

Sylvain ARBAUDIE · January 9, 2025

ARCHITECTURE SIMPLICITY OPINION DEVOPS

Epidemia złożoności

Branża oprogramowania ma problem ze złożonością. Nie złożonością inherentną rozwiązywanym problemom — ta jest nieunikniona. Nie, mówię o złożoności, którą sobie sami narzucamy: złożoności przypadkowej.

Projekt CRUD z 10 000 użytkowników wdrożony na Kubernetes z Istio, Prometheus, Grafana, ArgoCD, event bus Kafka, 12 mikroserwisami, 3 różnymi bazami danych i service mesh. Dlaczego? Bo tak robi Netflix. Bo to jest "cool" w 2025.

Rezultat: zespół 5 programistów spędzający więcej czasu na zarządzaniu infrastrukturą niż na tworzeniu funkcjonalności.

Ewolucja architektur

Cofnijmy się. Historia architektur oprogramowania to historia rosnącej złożoności:

Monolit → Jedna aplikacja, jedno wdrożenie, jedna baza danych. Proste, skuteczne i doskonale dopasowane do większości projektów.

SOA (Service-Oriented Architecture) → Usługi komunikujące się przez ESB (Enterprise Service Bus). Bardziej elastyczne niż monolit, ale ESB staje się pojedynczym punktem awarii i wąskim gardłem.

Mikroserwisy → Niezależne usługi komunikujące się przez API. Teoretycznie każda usługa może być rozwijana, wdrażana i skalowana niezależnie. W praktyce złożoność operacyjna jest ogromna.

EDA (Event-Driven Architecture) → Usługi komunikują się przez asynchroniczne zdarzenia. Maksymalne rozluźnienie, ale debugowanie przepływu zdarzeń przez 15 usług to koszmar.

Każdy krok dodawał złożoność. I na każdym kroku branża przedstawiała nową architekturę jako uniwersalne rozwiązanie. Spoiler: żadna architektura nie jest uniwersalna.

Zasada KISS

KISS — Keep It Simple, Stupid — to zasada, która powinna wisieć na ścianie każdego biura architekta oprogramowania.

Zasada nie mówi "rób prosto, bo jesteś kiepski". Mówi: **złożoność ma koszt, i ten koszt musi być uzasadniony**.

Każdy dodany do architektury komponent to:

- Kolejny komponent do utrzymania
- Dodatkowy punkt awarii
- Dodatkowa kompetencja wymagana w zespole
- Dodatkowy koszt infrastruktury
- Wydłużony czas debugowania

Kubernetes: idealny przykład

Kubernetes to nadzwyczajne narzędzie. Do orkiestracji setek kontenerów w skali Google, Netflix czy Spotify jest niezbędne.

Do wdrożenia aplikacji MariaDB / MySQL + PHP/Node.js z 5 000 użytkowników? To armata do zabicia muchy.

Dedykowany serwer (lub VPS) z Docker Compose wykonuje zadanie za ułamek kosztu i złożoności:

```
# docker-compose.yml — to wszystko, czego potrzebujesz
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:
```

Bez klastra Kubernetes. Bez Helm charts. Bez service mesh. Bez rozproszonego monitoringu. Tylko dwa kontenery wykonujące swoją pracę.

Firmy wracające do prostoty

Ruch odwrotny trwa. Firmy opuszczają złożone architektury, by wrócić do prostszych podejść:

- **Basecamp/37signals** przeniósł swoje obciążenia z chmury na serwery dedykowane, oszczędzając miliony dolarów rocznie
- **Amazon Prime Video** zmigrował usługę z mikroserwisów do monolitu, redukując koszty o 90%
- **DHH** (twórca Ruby on Rails) aktywnie agituje za "cloud exit"

Te firmy nie są technofobami. Po prostu zrobiły kalkulację: złożoność kosztuje więcej niż prostota, przy równoważnej funkcjonalności.

Fundamentalne pytanie

Przed wyborem technologii zadaj sobie to pytanie: **jaki problem rozwiązuję?**

Nie "jaka technologia jest modna". Nie "co zrobi wrażenie w moim CV". Nie "czego używają GAFA". Pytanie brzmi: jaki jest konkretny problem i jakie jest najprostsze rozwiązanie, które go rozwiązuje?

Jeśli odpowiedź na "dlaczego Kubernetes?" brzmi "bo tak się robi w 2025", masz problem decyzyjny, nie techniczny.

Kiedy złożoność jest uzasadniona

Bądźmy jasni: złożoność jest czasem konieczna.

Jeśli zarządzasz 10 milionami użytkowników ze skokami obciążenia 50x, Kubernetes jest uzasadniony. Jeśli masz 200 programistów na tym samym produkcie, mikroserwisy umożliwiają autonomię zespołów. Jeśli musisz przetwarzać 100 000 zdarzeń na sekundę, Kafka jest właściwą odpowiedzią.

Ale te przypadki to wyjątek, nie norma. 90% aplikacji webowych nie ma tych ograniczeń. A dla tych 90% dobrze zbudowany monolit z bazą MariaDB / MySQL, wdrożony na serwerze (lub dwóch dla redundancji), jest nie tylko wystarczający — jest optymalny.

Mój manifest na rzecz prostoty

1. **Zacznij od monolitu.** Dziel na usługi wtedy (i tylko wtedy), gdy ból monolitu przekracza ból dystrybucji.
2. **Używaj tego, co znasz.** Opanowany stos jest wydajniejszy niż "cool" stos źle opanowany.
3. **Licz swoje komponenty.** Jeśli twoja architektura ma więcej komponentów niż programistów w zespole, to sygnał alarmowy.
4. **Mierz koszt całkowity.** Infrastruktura + czas rozwoju + czas debugowania + czas szkolenia. Złożoność rzadko jest darmowa.
5. **Zadawaj pytanie "dlaczego?".** Dla każdego komponentu infrastruktury zapytaj "dlaczego tu jest?". Jeśli odpowiedź brzmi "na wszelki wypadek", usuń go.

Podsumowanie

Najlepsza architektura to najprostsza, która rozwiązuje problem. Nie najefektowniejsza, nie najmodniejsza, nie najkompletniejsza. Najprostsza.

Przyjmij prostotę. Twój zespół, budżet i użytkownicy będą wdzięczni.

Ten artykuł został pierwotnie opublikowany na Medium.