Embrace Simplicity

Sylvain ARBAUDIE · January 9, 2025

ARCHITECTURE SIMPLICITY OPINION DEVOPS

The Complexity Epidemic

The software industry has a complexity problem. Not the inherent complexity of the problems we solve — that is unavoidable. No, I am talking about the complexity we inflict on ourselves: accidental complexity.

A CRUD project with 10,000 users deployed on Kubernetes with Istio, Prometheus, Grafana, ArgoCD, a Kafka event bus, 12 microservices, 3 different databases and a service mesh. Why? Because that is what Netflix does. Because it is "cool" in 2025.

The result: a team of 5 developers spending more time managing infrastructure than developing features.

The Architecture Evolution

Let us step back. The history of software architectures is a story of increasing complexity:

Monolith -- One application, one deployment, one database. Simple, effective, and perfectly suited to the majority of projects.

SOA (Service-Oriented Architecture) -- Services communicating via an ESB (Enterprise Service Bus). More flexible than the monolith, but the ESB becomes a single point of failure and a bottleneck.

Microservices -- Independent services communicating via APIs. Theoretically, each service can be developed, deployed and scaled independently. In practice, the operational complexity is enormous.

EDA (Event-Driven Architecture) -- Services communicating through asynchronous events. Maximum decoupling, but debugging an event flow across 15 services is a nightmare.

Each step added complexity. And at each step, the industry presented the new architecture as the universal solution. Spoiler: no architecture is universal.

The KISS Principle

KISS — Keep It Simple, Stupid — is a principle that should be posted on the wall of every software architect's office.

The principle does not say "keep it simple because you are incapable." It says: **complexity has a cost, and that cost must be justified.**

Every component added to your architecture is:

- One more component to maintain
- One more failure point
- One more skill required on the team
- One more infrastructure cost
- More debugging time

Kubernetes: The Perfect Example

Kubernetes is an extraordinary tool. For orchestrating hundreds of containers at the scale of Google, Netflix or Spotify, it is indispensable.

For deploying a MariaDB / MySQL + PHP/Node.js application with 5,000 users? It is a cannon to kill a fly.

A dedicated server (or a VPS) with Docker Compose does the job for a fraction of the cost and complexity:

```
# docker-compose.yml — this is all you need
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:
```

No Kubernetes cluster. No Helm charts. No service mesh. No distributed monitoring. Just two containers doing their job.

Companies Moving Back

A reverse movement is underway. Companies are leaving complex architectures to return to simpler approaches:

- **Basecamp/37signals** brought their workloads back from the cloud to dedicated servers, saving millions of dollars per year
- **Amazon Prime Video** migrated a service from microservices to a monolith, reducing costs by 90%
- **DHH** (creator of Ruby on Rails) is actively advocating for "cloud exit"

These companies are not technophobes. They simply did the math: complexity costs more than simplicity, for equivalent functionality.

The Fundamental Question

Before choosing a technology, ask yourself this question: **what problem am I solving?**

Not "what technology is trendy." Not "what will impress on my resume." Not "what do the big tech companies use." The question is: what is the concrete problem, and what is the simplest solution that solves it?

If the answer to "why Kubernetes?" is "because that is what we do in 2025," you have a decision problem, not a technical one.

When Complexity Is Justified

Let us be clear: complexity is sometimes necessary.

If you are managing 10 million users with 50x load spikes, Kubernetes is justified. If you have 200 developers on the same product, microservices enable team autonomy. If you need to process 100,000 events per second, Kafka is the right answer.

But these cases are the exception, not the norm. 90% of web applications do not have these constraints. And for that 90%, a well-built monolith with a MariaDB / MySQL database, deployed on a server (or two for redundancy), is not only sufficient — it is optimal.

My Simplicity Manifesto

1. **Start with a monolith.** Split into services when (and only when) the pain of the monolith exceeds the pain of distribution.
2. **Use what you know.** A mastered stack is more performant than a "cool" stack poorly mastered.
3. **Count your components.** If your architecture has more components than developers on the team, that is a red flag.
4. **Measure the total cost.** Infrastructure + development time + debugging time + training time. Complexity is rarely free.
5. **Ask "why?"** For every infrastructure component, ask "why is it there?" If the answer is "just in case," remove it.

Conclusion

The best architecture is the simplest one that solves the problem. Not the most impressive, not the trendiest, not the most complete. The simplest.

Embrace simplicity. Your team, your budget, and your users will thank you.

This article was originally published on Medium.