Adopter la simplicité

Sylvain ARBAUDIE · 9 janvier 2025

ARCHITECTURE SIMPLICITY OPINION DEVOPS

L'épidémie de complexité

L'industrie du logiciel a un problème de complexité. Pas la complexité inhérente aux problèmes que nous résolvons — celle-là est inévitable. Non, je parle de la complexité que nous nous infligeons : la complexité accidentelle.

Un projet CRUD avec 10 000 utilisateurs déployé sur Kubernetes avec Istio, Prometheus, Grafana, ArgoCD, un event bus Kafka, 12 microservices, 3 bases de données différentes et un mesh service. Pourquoi ? Parce que c'est ce que Netflix fait. Parce que c'est ce qui est "cool" en 2025.

Le résultat : une équipe de 5 développeurs qui passe plus de temps à gérer l'infrastructure qu'à développer des fonctionnalités.

L'évolution des architectures

Retour en arrière. L'histoire des architectures logicielles est une histoire de complexité croissante :

Monolithe → Une application, un déploiement, une base de données. Simple, efficace, et parfaitement adapté à la majorité des projets.

SOA (Service-Oriented Architecture) → Des services communiquant via un ESB (Enterprise Service Bus). Plus flexible que le monolithe, mais l'ESB devient un point de défaillance unique et

un goulot d'étranglement.

Microservices → Des services indépendants communiquant via des APIs. Théoriquement, chaque service peut être développé, déployé et scalé indépendamment. En pratique, la complexité opérationnelle est énorme.

EDA (Event-Driven Architecture) → Les services communiquent par événements asynchrones. Découplage maximal, mais debugger un flux d'événements à travers 15 services est un cauchemar.

Chaque étape a ajouté de la complexité. Et à chaque étape, l'industrie a présenté la nouvelle architecture comme la solution universelle. Spoiler : aucune architecture n'est universelle.

Le principe KISS

KISS — Keep It Simple, Stupid — est un principe qui devrait être affiché au mur de chaque bureau d'architecte logiciel.

Le principe ne dit pas "faites simple parce que vous êtes nul". Il dit : **la complexité a un coût, et ce coût doit être justifié**.

Chaque composant ajouté à votre architecture est :

- Un composant de plus à maintenir
- Un point de défaillance supplémentaire
- Une compétence supplémentaire requise dans l'équipe
- Un coût d'infrastructure supplémentaire
- Un temps de debugging allongé

Kubernetes : l'exemple parfait

Kubernetes est un outil extraordinaire. Pour orchestrer des centaines de containers à l'échelle de Google, Netflix ou Spotify, c'est indispensable.

Pour déployer une application MariaDB / MySQL + PHP/Node.js avec 5 000 utilisateurs ? C'est un canon pour tuer une mouche.

Un serveur dédié (ou un VPS) avec Docker Compose fait le travail pour une fraction du coût et de la complexité :

```
# docker-compose.yml — c'est tout ce dont vous avez besoin
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:
```

Pas de cluster Kubernetes. Pas de Helm charts. Pas de service mesh. Pas de monitoring distribué. Juste deux containers qui font leur travail.

Les entreprises qui reviennent en arrière

Un mouvement inverse est en cours. Des entreprises quittent les architectures complexes pour revenir à des approches plus simples :

- **Basecamp/37signals** a rapatrié ses workloads du cloud vers des serveurs dédiés, économisant des millions de dollars par an
- **Amazon Prime Video** a migré un service de microservices vers un monolithe, réduisant les coûts de 90%
- **DHH** (créateur de Ruby on Rails) milite activement pour le "cloud exit"

Ces entreprises ne sont pas technophiles. Elles ont simplement fait le calcul : la complexité coûte plus cher que la simplicité, à fonctionnalités équivalentes.

La question fondamentale

Avant de choisir une technologie, posez-vous cette question : **quel problème suis-je en train de résoudre ?**

Pas "quelle technologie est à la mode". Pas "qu'est-ce qui impressionnera dans mon CV". Pas "qu'est-ce que les GAFA utilisent". La question est : quel est le problème concret, et quelle est la solution la plus simple qui le résout ?

Si la réponse à "pourquoi Kubernetes ?" est "parce que c'est ce qu'on fait en 2025", vous avez un problème de décision, pas un problème technique.

Quand la complexité est justifiée

Soyons clairs : la complexité est parfois nécessaire.

Si vous gérez 10 millions d'utilisateurs avec des pics de charge 50x, Kubernetes est justifié. Si vous avez 200 développeurs sur le même produit, les microservices permettent l'autonomie des équipes. Si vous avez besoin de traiter 100 000 événements par seconde, Kafka est la bonne réponse.

Mais ces cas sont l'exception, pas la norme. 90% des applications web n'ont pas ces contraintes. Et pour ces 90%, un monolithe bien construit avec une base MariaDB / MySQL, déployé sur un serveur (ou deux pour la redondance), est non seulement suffisant — c'est optimal.

Mon manifeste pour la simplicité

1. **Commencez par un monolithe.** Découpez en services quand (et seulement quand) la douleur de la monolithique dépasse la douleur de la distribution.
2. **Utilisez ce que vous connaissez.** Une stack maîtrisée est plus performante qu'une stack "cool" mal maîtrisée.
3. **Comptez vos composants.** Si votre architecture a plus de composants que de développeurs dans l'équipe, c'est un signal d'alarme.
4. **Mesurez le coût total.** Infrastructure + temps de développement + temps de debugging + temps de formation. La complexité est rarement gratuite.
5. **Posez la question "pourquoi ?".** Pour chaque composant d'infrastructure, demandez "pourquoi est-il là ?" Si la réponse est "au cas où", supprimez-le.

Conclusion

La meilleure architecture est la plus simple qui résout le problème. Pas la plus impressionnante, pas la plus à la mode, pas la plus complète. La plus simple.

Adoptez la simplicité. Votre équipe, votre budget et vos utilisateurs vous remercieront.

Cet article a été initialement publié sur Medium.