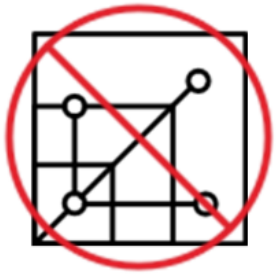
Microservices Anti-Patterns

While there are many patterns for doing microservices well, an equally significant number of patterns exist that can quickly get any development team into trouble. The following are some of the don'ts while developing microservices:

Don't build microservices



The first rule of microservices is don't start with microservices. When you determine that the monolithic application's complexity negatively affects application development and maintenance, consider refactoring that application into smaller services.

When the application becomes too large to update and maintain easily, these microservices will become ideal for breaking down the complexity and making the application more manageable.

However, until you feel that pain, you don't even have a monolith that needs refactoring.

Not taking automation seriously

If you have a monolith application, you only need to deploy one piece of software. Once you move to a microservices architecture, you will have more than one application with each having different code, test, and deploy cycles.

Attempting to build microservices without either:

- proper deployment and monitoring automation, or
- managed cloud services to support your now sprawling, heterogenous infrastructure

is asking for a lot of unnecessary trouble.

So, when you are building microservices, always use DevOps or cloud services.

Don’t build nanoservices

If you go too far with the micro in microservices, you could easily find yourself building nanoservices! The complexity of which will outweigh the overall gains of microservices architecture.

Lean toward creating larger services and create smaller services when:

- Deploying changes becomes difficult
- The common data model becomes overly complex
- Loading and scaling requirements no longer synchronize and affect application performance

Don’t turn into SOA

The two concepts; microservices and service-oriented architecture (SOA) are often confused with one another because, at their most basic level, both build reusable individual components that can be consumed by other applications.

However, microservices are fine-grained with independent data storage for each, that is, the bounded context.

A microservices project that morphs into an SOA project will likely buckle under its own weight.

Don’t build a gateway for each service

Instead of implementing end-user authentication, throttle, orchestrate, transform, route, and analytics in each service, you should use an API Gateway.

An API gateway is an API management tool that sits between a client and your collection of backend services.

This will become central to the above-mentioned non-functional concerns and will avoid re-engineering them with each service.

Conclusion

The aim of microservices is to solve the three most frequent challenges, that is, enhance customer experience, be flexible to new requirements, and reduce costs by providing business functions as fine-grained services.

But while doing so, you should avoid the pitfall of the above-mentioned anti-patterns making microservices a nuisance to your development, delivery, and management requirements.