A Microservice Overdose: When engineering trends meet the startup reality

Microservices are a significant software trend that can have far-reaching repercussions for enterprise IT and the entire business's digital transformation. As a result, companies are increasingly using microservice architecture to design applications.

But how is the microservices trend reflected in the everyday reality of start-ups?

Compared to other organizations, many characteristics are unique to startups, including the rapid growth and the need to deliver fast and respond quickly.

Before we look into the unique case of microservices architecture and other engineering trends in startups, let's briefly summarize the benefits of microservice architecture and why it threatens to replace monolithic architecture, which has been the standard in the market for so many years.

Microservices architecture vs. monolithic architecture

A monolithic application is built as a single, indivisible piece of software, and

Its most significant advantage is keeping it simple. It is easy to set up, simple to monitor, debug, etc.

Its big downside, however, is the fact that it doesn't scale.
The trendy Microservices, on the other hand, are very scalable. Their main benefits are:

- Independently deployable
- Fault isolation
- Clear ownership
- Can use different programming languages
- You can say you're doing Microservices (Trust us, it's fun)

The microservice reality

So what happens when start-ups want to join the Microservices trend? Well, they discover pretty quickly that alongside the benefits, this trend is also causing them problems. Here are some of them:

Problem #1: code duplication
One of the things that most characterize startups is that they generate about ten times more code than regular organizations, or how we like to call it: 10 times more code, ten times more pain.

If you are setting up a new microservice, no matter what programming language you use, you will find that you have to write **a lot** of code.
A great example would be enterprise utilities. First, you will need to write yourself any piece of code not provided to you by a third party. Then, most likely, you'll find out that you need to start duplicating all sorts of bits of code between microservices.

The same goes for configurations as well. You may create a configuration and then reuse it, or each team will create configurations for itself, which will cause each team to operate in its own world, and then you will find that you have a new problem of consistency.

The same is true for Data. One of the most important things is that each microservice is responsible for its own data. But no matter how much you try, you can't avoid an inevitable overlap in the database, and that leads us to duplicate schemes or parts of them.

Problem #2: DevOps overhead
DevOps overhead is a significant and painful problem, which was born due to the transition to microservices. After all, we are replacing the complexity of monolith with the complexity of architecture and networking. Here are some examples:

Service interfaces - one of these issues we have to face is the interface. What happens when my microservice wants to communicate with the outside world? For example, another microservice, storage, or third-party application?

Databases & Data migrations - you may have several databases. You may have one database with many more tables, creating one of the most painful things in start-up reality - data migration.

Deployment / CI - If at the moment we had a very defined Deploit here, we are talking about a lot of variations, another issue that causes us pain. In addition, the architecture is becoming much more complex.

Problem #3 - Observability is painful

Last but not least, we have Monitoring & logging. How do you monitor your operation?

Monitoring is a very significant problem that several startups are trying to address.

The decentralization of working with microservices made it very difficult to see the big picture. When there are many teams with many databases, no one sees the big picture. It gets more complicated as your product branches out and starts to include more microservices, and the architecture becomes more complex.

This problem is excruciating when you need to debug an elusive bug.

The reality of start-ups is such that sometimes one customer can make all the difference. If the largest bank in the world becomes your customer, that alone can lead the company to an exit. In this reality, giving a quick response for the customer is super critical for the start-up, and debugging can be the deciding factor.

The flexible alternative: Miniservices

In light of all the problems we mentioned above, we started looking for a solution that would give us a little more flexibility, and we found it in miniservices, or as we like to call them: microservices' cousins . The concept is similar, but the architecture is a bit more flexible.

Here are the key benefits:

- A flexible microservices architecture
- Miniservices may share data
- Communication methods may vary
- Typically fewer services
- Typically fewer data stores

The bottom line is we made a conscious decision to give up architectural perfection in

favor of business value, and it works great for us - for now. However, in the future, when we grow up and reach the correct scale, we will be able to switch to microservices easily.

Multy-repo Vs. Monorepo

Microservices are not the only big engineering trend that happing right now. Another big trend that you may have heard about is Multy-repo.

The multi-repo strategy enables the microservice team to maintain a separate and isolated repository for each responsibility area. As a result, one group may own a codebase end to end, developing and deploying features autonomously.

But then again, multi-repo is great until you realize that code duplication and config duplication still not solved.

Code duplication is expected with a multi-repo strategy because multi-repo encourages a segmented culture. Each team does its own thing, making it challenging to prevent groups from solving the same problem repeatedly.

A better alternative might be the mono-repo approach. In a mono-repo approach, all services and codebase are kept in a single repository. The only problem is

that mono-repo is fantastic if you're Google / Twitter / Facebook. Otherwise, it doesn't scale very well. It is can also create Clone pain (CI, local) and

Many CI/CD/VCS work better out-of-the-box with multi-repo.

Our solution: we use mono-repo that could be easily separated to Multi-repo as it scales. We also share code via internal libraries (Our case: NPM packages) and use symlinks for LINT, test configurations, etc.

We also use a multi-repo code-sharing alternative like Git submodules, and this combination is working for us very well.

The trendy continuous deployment

Continuous Deployment (CD) is a software release procedure that utilizes automated testing to determine whether changes to a codebase are correct and stable enough to be sent to a production environment immediately and autonomously.

Continuous deployment has some great benefits. Deploy changes to production in

minutes, the short feedback loop, and it goes very well with TDD (Test-Driven Development). The problem? these benefits come with a price tag:

- Very high test coverage is essential
- E2E/Integration tests may require complex infrastructure
- CI gets slower with time
- Tests take time to write
- Tests take time to review
- Tests take time to maintain

Our solution to overcome these issues? We put the testing pyramid on a diet (calculated investments in testing). We are also focusing on UTs for the critical path. We use A staging environment and very effective monitoring and alerts system.

In conclusion

In this review, we have gone over some of the important engineering trends in the IT world and how they fit into start-up reality. We saw why not everything suitable for a large company is ideal for a start-up because of its unique nature.
Our best advice for start-ups dealing with the same dilemmas would be to leave room for scale without being fanatic about it.
Not everything that fits Netflix and Google is right for your start-up, and it is best to make decisions that fit your company's specific needs.