How to build a calculator?

Software architecture is hard to pick. The market today puts a lot of emphasis on scalability and uptime. By knowing which architecture you should use you can save a lot of time and make a better application. It can take more time to set up the right environment for your development, but it will help separating the different business functionalities from each other and therefore make it easier for a new developer to add new features.

Hypothesis

We believe that setting up for scalability will take more time to setup but in the end will be easier to scale and get familiar with for new developers, thus saving time in the long run.

What can we gain from this?

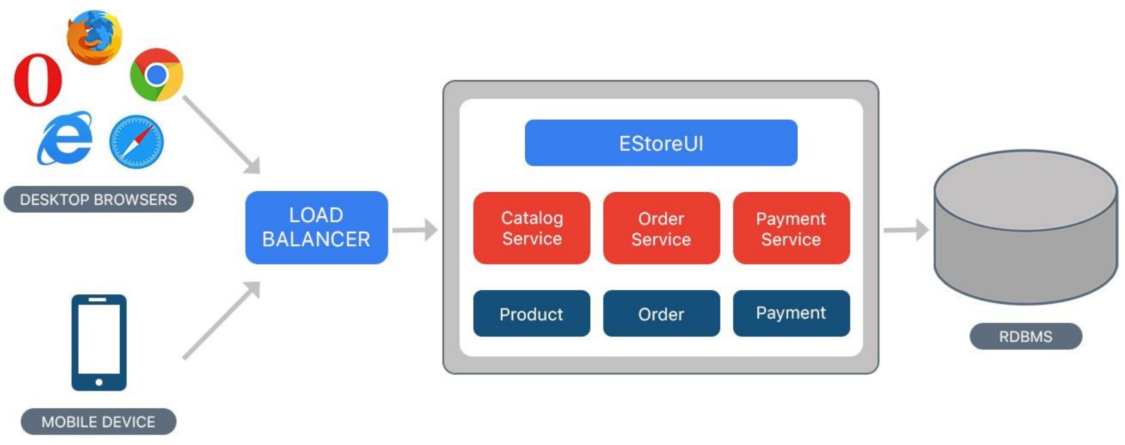
So we know that there are two different ways to build systems or projects and want to explore what makes them different. In this article we will be circling around the idea of making a calculator app, or system. Having built it in two different ways we can look at some of the differences and what they mean, and with that knowledge, this should easily be applicable to other systems that have the same core concepts.

What is microservices and monolithic

So we will be building the calculator and measure certain metrics that will help you decide what works best.

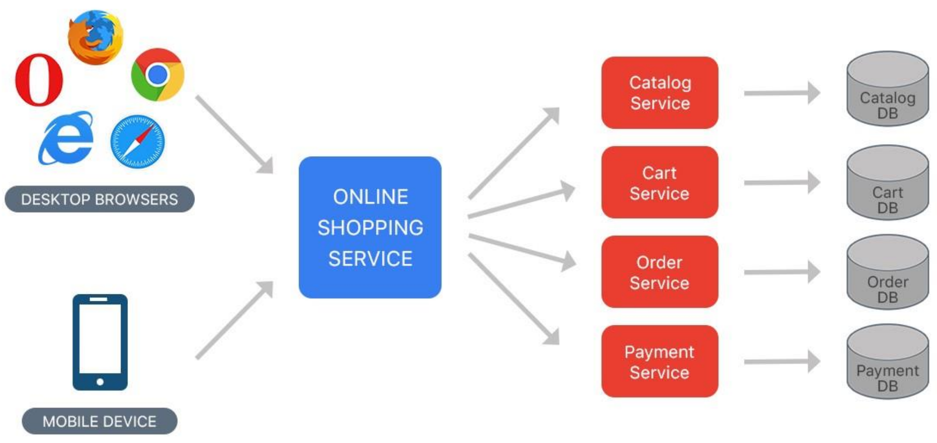
Let us start with a simple explanation of the differences between monolithic and microservices and what they essentially are.

To keep it short here is a picture of what monolithic design is:



We see that everything is in the same 'box' or 'container'. This makes it work differently than microservices as you cannot scale a single component and everything is very tightly coupled, this means everything is reliant on each other. It also means that if something breaks down, everything with break down.

Here it is in microservices!



We see that each part is its own service and that means, if we need more of one specific service, we can just clone it! We can simply just make another service of a specific one instead of having to 'clone' the entire system with everything in it. This is not only bad for performance but can also cause other issues as well. This is made super easy with Kubernetes for example, where you can change a variable called 'replica' and you will get 'x' more or less instances of that service.

Advantages

So to go back to the calculator, we can make it in a monolithic design where everything will be packaged into the same system, makes sense right? Yeah, kind of.

Here's why making it with microservices could be beneficial. Assuming the calculator has more than a couple operations, and we know that we will be adding more as time goes on, we can take advantage of microservices. Instead of having it all in the same service, we can split it up so each operation will be in its own service! This has a couple benefits. It means that we can integrate new ones without touching the older operations thus ensuring we will not break them. It also means that parts of the calculator can break, but the entire system still work, without the broken part!

How does that make sense? It means that one of the operations can break, without the entire calculator going down. We can have a functioning calculator without the '+' operation for example.

The downside of this is that it takes longer to get the calculator to a minimum viable product. Making sure you can integrate the services or operations requires more work than just throwing them in there.

The experiment

Let us say we can use the same frontend or client for both projects so we don't need to worry about it. We just need to make the business aspect of the calculator.

We decided to use Docker to make one of the projects and tried to use the microservice structure. We concluded this was the best way to really show the scalability potential. Docker is good for many different things but it can be integrated easily with Kubernetes which truly will make the scalability great.

It makes it very easy to integrate new services and also control them. So this is quite a lot to setup before our simple plus minus operations will be available to our client.

We also want to add some tests to be sure our logic actually works so we will add that too. We are starting to accumulate quite a lot of different files.

Our second project we made using the monolithic design. We just threw in all the code into the same file and it was made rather quickly. We again made a couple of tests to ensure the logic works as we expect. Because we did not use any exterior tools and simply just wrote the code we needed it was very fast to get a working system.

What's better?

Time to develop:

Monolithic: **~15minutes**

Microservices: **~60minutes**

This is roughly the same as the code produced. The project that was built on microservices had about 4-5 times as much code as the monolithic one.

So we made more code and it took longer to get to a minimum viable product.

Conclusion

Based on our experiences with this experiment, we have found out that it only makes sense to build a project using microservices if:

It is to be developed by multiple people, and it has a substantial sized business case with multiple business functionalities.

In such a small project as the one we have done as explained above the monolithic design is both faster to setup, as we do not have to setup different tools. It is still very easy to get an overview of how the application works due to its small size and therefore it has not got the same complexity as a huge monolithic program normally would have.

Microservices still have advantages as we mentioned earlier, where we can easily scale one of the business functions instead of having to scale everything. In our experiment it probably causes more problems than advantages as it is hard to split up such a small project in multiple services, but one of the clear advantages that it still has is that one of our operations can break and the rest of the calculator will still function as intended.

We can therefore see that a microservice structure doesn't utilize its full potential in such a small business case. It would make a lot of sense to do microservices if had a huge project/business case and there is a clear difference in the business functionalities, so it would be easy to split up. Due to the size of this project we experimented on we can't confirm or deny our hypothesis as our experiment was not adequate.

We believe that any project that fits the criteria mentioned in the top of the conclusion and that takes more than a few days to completely finish should be built with microservices.

Resources

Kubernetes replica and scalability source:
https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Microservices: https://en.wikipedia.org/wiki/Microservices

Monolithic architecture: https://en.wikipedia.org/wiki/Monolithic_application

Pictures of architecture: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

Microservice calculator source code:
https://github.com/niichtsShaiaz/UFO/tree/master/Exam/ufo-microservices

Monolithic calculator source code:
https://github.com/niichtsShaiaz/UFO/tree/master/Exam/CalculatorMonolith