semaphore

Sample

Transitioning from Monolith to Microservices Handbook

Transitioning from Monolith to Microservices Handbook

Converting monoliths to the microservice architecture

Semaphore

Contents

This book is open source: https://github.com/semaphoreci/book-microservices

Published on the Semaphore website: https://semaphoreci.com/resources/microservices

Sep 2022: First edition v1.0 (revision e932d9d)

Share this book:

> *I've just started reading "Transitioning from Monolith to Microservices Handbook" a free ebook by @semaphoreci: https://bit.ly/3eWMTA0 (Tweet this!)*

Preface

Microservices are the most scalable way of developing software. As projects grow in size and complexity, one of the possible ways forward is to break the system into autonomous microservices and hand them out to different teams.

Given the advantages, one would be forgiven for thinking that microservices are the superior architecture. But there are some caveats that, if ignored, can lead to development hell. This book aims to help you decide when migrating your monolith to the microservice architecture is a good idea, if so, navigate the choppy waters ahead.

Who Is This Book for, and What Does It Cover?

This book is intended for software engineers at every level and tech leaders who are either exploring microservice architecture or are faced with serious scalability problems in their monolith applications.

- In chapter 1 we define microservices and weight this architecture model against the alternatives. If you're unsure if microservices is right for your project, be sure to not skip this chapter.
- Chapter 2 talks about the cultural transformation a company must undergo to be effective at microservice design and operations.
- Chapter 3 covers design techniques for microservice application. We take a deep dive into Domain-Driven Design and how it applies to microservices.
- Chapter 4 goes to the core of breaking up the monolith. We lay a plan for the migration, discuss the steps required to prepare the monolith before the transition, and explore techniques for testing microservice applications.
- Chapter 5 covers the operational side of running a microservice application, including deployment and release management.

Additional recommended reading

You won't learn absolutely everything you need to design and run microservices in this book. Instead, the focus is to break up a monolith since this is the most common (and even recommended) path to microservices. As supplementary material, we recommend the following free ebooks also published by Semaphore:

- CI/CD with Docker and Kubernetes: it's common practice to run microservices with containers and orchestrate them with Kubernetes. This book will introduce both concepts and show step-by-step how to work with them.
- CI/CD for Monorepos: monorepos are a popular way of organizing and developing microservice codebases. This book will show you the best ways of working with monorepos.

How to Contact Us

We would very much love to hear your feedback after reading this book. What did you like and learn? What could be improved? Is there something we could explain further?

A benefit of publishing electronically is that we can continuously improve it. And that's exactly what we intend to do based on your feedback.

You can send us feedback by sending an email to learn@semaphoreci.com.

Find us on Twitter: https://twitter.com/semaphoreci

Find us on Facebook: https://facebook.com/SemaphoreCI

Find us on LinkedIn: https://www.linkedin.com/company/rendered-text

About the Authors

Pablo Tomas Fernandez Zavalia is an electronic engineer and writer. He started out developing for the City Hall of Buenos Aires (buenosaires.gob.ar). After graduating, he joined British Telecom as head of the Web Services department in Argentina. He then worked for IBM as a database administrator, where he also did tutoring, DevOps, and cloud migrations. In his free time, he enjoys writing, sailing, and board games. Follow Tomas on Twitter at @tomfernblog.

Lee Atchison is a software architect, published author, and frequent public speaker on the topics of cloud computing and application modernization. Follow Lee at @leeatchison.

About the Editor

Marko Anastasov is a software engineer, author, and entrepreneur. Marko co-founded Rendered Text, the software company behind the Semaphore CI/CD service. He worked on building and scaling Semaphore from an idea to a cloud-based platform used by some of the world's best engineering teams. Follow Marko on Twitter at @markoa.

About the Reviewer

Dan Ackerson picked up most of his soft and hardware troubleshooting skills in the US Army. A decade of Java development drove home to operations, scaling infrastructure to cope with the thundering herd. Engineering coach and CTO of Teleclinic.

Chapter 1 — What Are Microservices?

Beloved by tech giants like Netflix and Amazon, microservices have become the darlings in modern software development. But, despite the benefits, this is a paradigm that is easy to get wrong. So, let's explore what microservices are and, more importantly, what they are not.

1.1 What is microservice architecture?

The microservice architecture is a software design approach that decomposes an application into small independent services. These services communicate over well-defined APIs, which means that services can be developed and maintained by autonomous teams, making it the most scalable method for software development.

1.2 Microservice vs. monolith architectures

Microservice design is the polar opposite of monolith development. A monolith is one big codebase ("the kitchen sink") that implements all functionalities. Everything is in one place, and no single component can work in isolation.

On the plus side, monoliths are easy to get up and running. Airbnb, to give an example, started with The Monorail, a Ruby on Rails application. While the company was still small, developers could iterate fast. Making broad changes was easy as the relationships between the different parts of the monolith were transparent.

As a company grows and teams increase in size, however, monolith development becomes troublesome. Soon, the system can no longer fit in a single head — there are just too many moving parts, so things slow down.

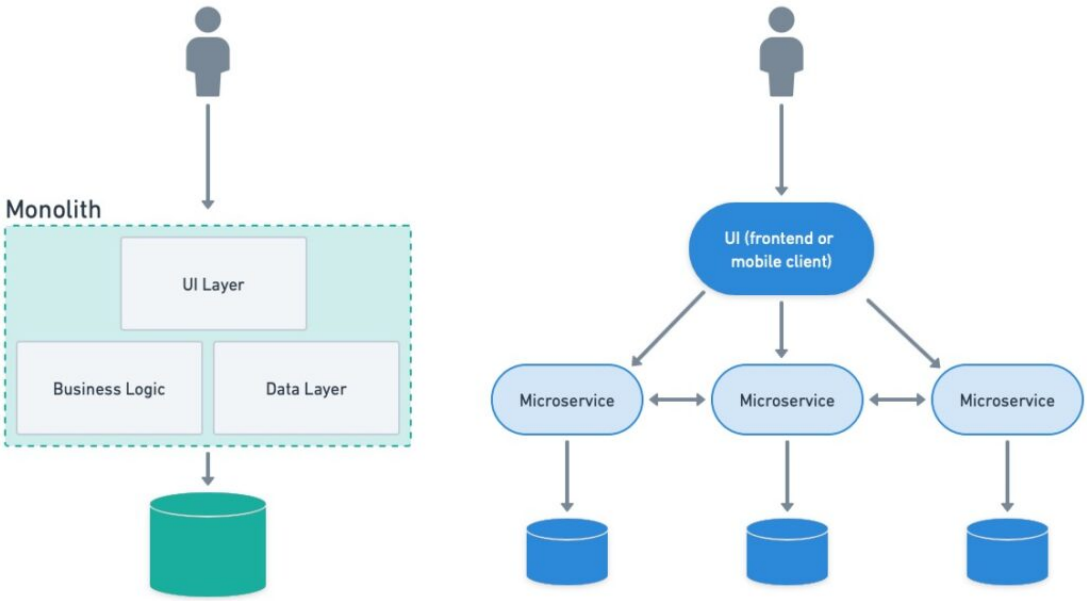

Figure 1: Monolith vs microservice architecture

1.3 Benefits of microservices

Microservices allow companies to keep teams small and agile. The idea is to decompose the application into small services that can be autonomously developed and deployed by tightly-knitted teams.

1.3.1 Scalability

The main reason that companies adopt microservices is **scalability**. Services can be developed and released independently without arranging large-scale coordination efforts within the organization.

1.3.2 Fault isolation

A benefit of having a distributed system is the ability to avoid single failure points. You can deploy microservices in different availability zones with cloud-enabled technologies, ensuring that your users never experience an outage.

1.3.3 Smaller teams

With microservices, the development team can stay small and cohesive. The smaller the group, the less communication overhead and the better the collaboration.

Amazon, as an example, takes team size to the extreme with their two pizza teams. Meaning that a team should be small enough to be fed by two pizzas.

1.3.4 The freedom to choose the tech stack

With a monolith, language and tech stack options are pretty much set in stone from the beginning. New developers must adapt to whatever choices were made in the past.

In contrast, each microservice can use the tech stack that is most appropriate for solving the task at hand. Thus, the team can pick the best tool for the job and based on their skills. For example, you can implement a high-performing service in Go or C and a high-tolerance microservice with Elixir.

1.3.5 More frequent releases

The development and testing cycle is shorter as small teams iterate quick. And, because they can also deploy their updates at any time, microservices can be updated much more frequently than a monolith.

1.4 The caveats of microservice design

At first glance microservices sound wonderful. They are modular, scalable, and fault tolerant. A lot of companies have had great success using this model, so microservices might naturally seem to be the superior architecture and the best way to start new applications.

However, most firms that have succeeded with microservices did not begin with them. Consider the examples of Airbnb and Twitter, which went the microservice route after outgrowing their monoliths and are now battling its complexities. Even successful companies that use microservices appear to still be figuring out the best way to make them work. It is evident that microservices come with their share of tradeoffs.

1.5 Microservice design challenges

Migrating from a monolith to microservices is also not a simple task, and creating an untested product as a new microservice is even more complicated. Microservices should only be seriously considered after evaluating the alternative paths. So, before embarking on a costly transition to microservices, we should talk about their shortcomings and limitations:

- **Small**: applies both to the team size and the codebase. A microservice must be small enough to be entirely understood by one person. As a rule of thumb, a microservice is too big if it would take it more than one sprint to rewrite it from scratch.
- **Focused on one thing**: a microservice must focus on one aspect of the problem or perform only one task.
- **Autonomous**: each microservice has its own database or persistence layer that is not shared with other services.
- **Aligned with the bounded context**: in software, we create models to represent the problem we want to solve. A bounded context represents the limits of a given model. Contexts are natural boundaries for services, so finding them is the most difficult and crucial part of designing a good microservice architecture.
- **Loosely-coupled**: while microservices can depend on other microservices, we must be careful about how they communicate. Each time a bounded context is crossed, some level of abstraction and translation is needed to prevent behavior changes in one service from affecting too much the rest.
- **Independently deployable**: being autonomous and loosely-coupled, a team can deploy their microservice with little external coordination or integration testing. Microservices should communicate over well-defined APIs and use translation layers to prevent behavior changes in one service from affecting the others.

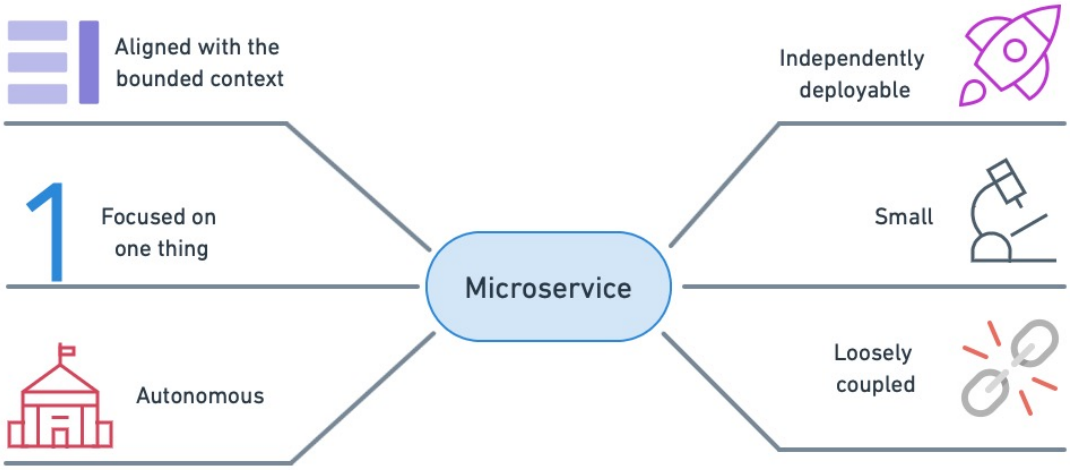

Figure 2: The key properties of microservice architecture

1.6 Reasons not to migrate to microservices

The caveats and strict limitations make microservices a bad fit for some types of workloads and applications. Let's see some common cases where microservices architecture is not recommended.

1.6.1 Microservices are only viable for mature products

On the topic of starting a new project with microservices, Martin Fowler observed that:

1. Almost all the successful microservice stories started with a monolith that got too big and was broken up.
2. Almost all the cases where a system that was built as a microservice system from scratch, ended up in serious trouble.

This pattern has led many to argue that you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.

The crux of the matter is that the first design is rarely optimal. The first few iterations of any new product are spent finding what users really need. Therefore, success hinges on staying agile and being able to quickly improve, redesign, and refactor. In this regard, microservices are manifestly worse than a monolith. If you don't nail the initial design, you're in for a rough start, as it's much harder to refactor a microservice than a monolith.

1.6.2 Microservices are not a good fit for startups

As a startup, you already are running against the clock, looking for a breakthrough before running out of capital. You don't need the scalability at this point (and probably not for a few years yet), so why make things harder by using a complicated architecture model?

Download the full ebook for free

We hope you have enjoyed this small sample of the ebook.

Download the the full 70+ page handbook for free here:

https://semaphoreci.com/resources/microservices