MARTINFOWLER.COM

Microservices Resource Guide

"Microservices" became the hot term in 2014, attracting lots of attention as a new way to think about structuring applications. I'd come across this style several years earlier, talking with my contacts both in ThoughtWorks and beyond. It's a style that many good people find is an effective way to work with a significant class of systems. But to gain any benefit from microservice thinking, you have to understand what it is, how to do it, and why you should usually do something else.

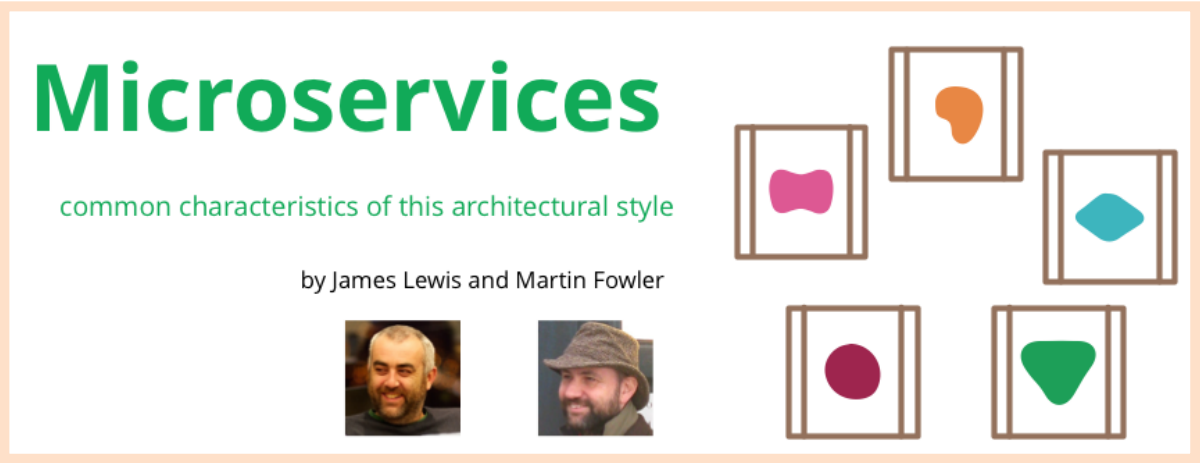
This is a guide to useful resources to find out more about microservices. It's a personal choice of articles, videos, books, and podcasts that can teach you more about the microservices architectural style. Primarily the selection helps to guide you through material on martinfowler.com, but I also cover additional material that I think is valuable. It does not attempt to be a comprehensive list (I've selected just a couple of introductory talks from the many out there) but this is my suggestion for places where your exploring should start.

Who Has Used Them?

What are Microservices?

> In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.
>
> -- James Lewis and Martin Fowler

Late in 2013, hearing all the discussion in my circles about microservices, I became concerned that there was no clear definition of microservices (a fate that caused many problems for SOA). So I got together with my colleague James Lewis, who was one of the more experienced practitioners of this style. Together we wrote



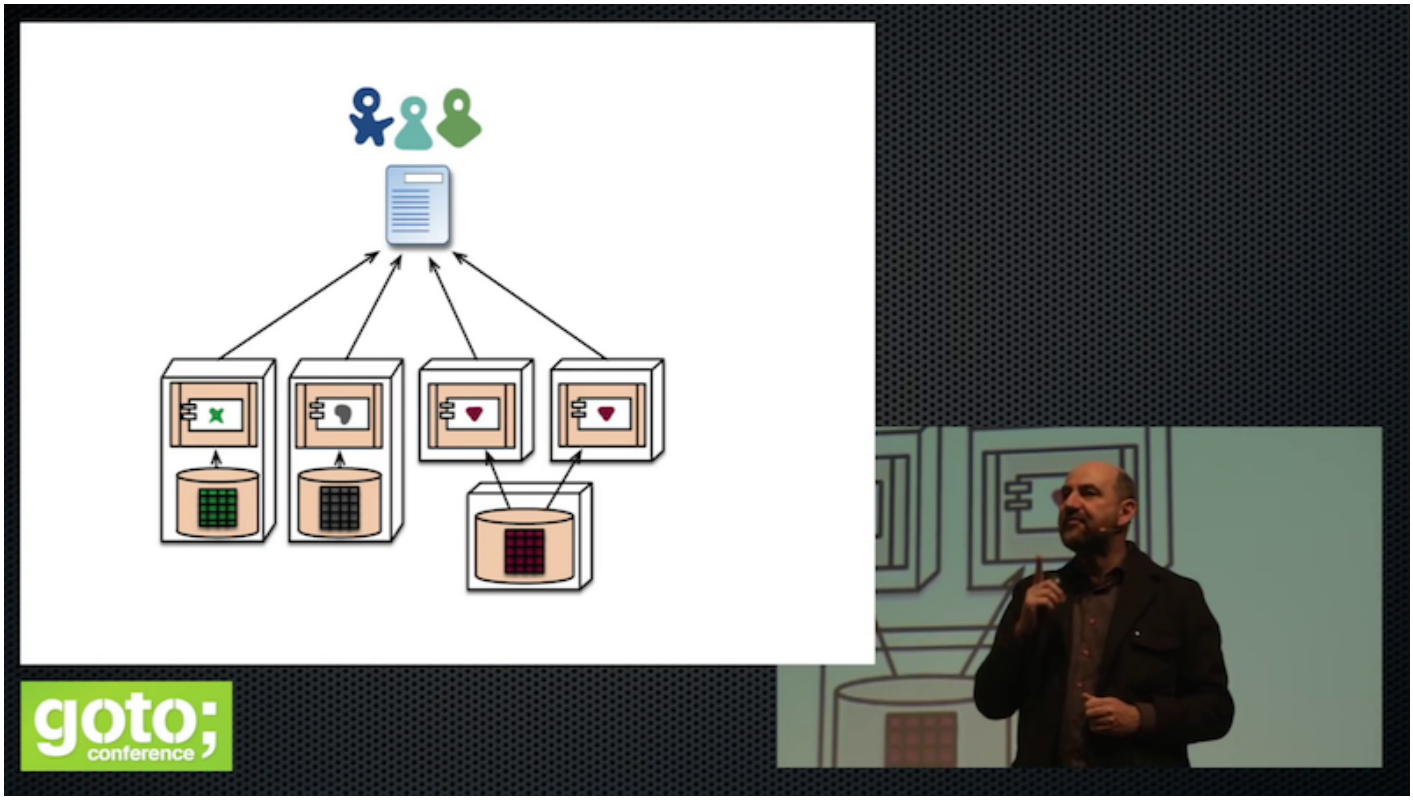
The article catalyzed interest in the microservices style, but we hope its true value is identifying the common characteristics of microservice architectures that we saw in the field.

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

We also look at common questions such as "how big is a microservice" and "what's the difference between microservices and Service-Oriented Architecture".

"Do we use it, do we not use it?

... and what on earth is it in the first place?"



In my short introductory talk (~25 minutes) I pick out the most important defining characteristics, compare microservices to monoliths, and outline the vital things do before putting a first microservice system into production.

James Lewis is one of our most experienced consultants with microservices. In this podcast with Software Engineering Radio (one hour) he covers most of the key points with the style, including deployment issues, size, comparison to SOA and the key figures in the community.

Microservices and Distributed Objects

When I wrote P of EAA I coined what I called the First Law of Distributed Object Design: "don't distribute your objects". This led led a few people to ask whether microservices are in contravention to this law, and if so why I am in favor of them?

Are Microservices just SOA

Right from the begining, many people have wondered about the link between microservices and Service-Oriented Architecture (SOA). James and I wrote a little about this in our original article. Matt McLarty goes into this in more detail, explaining the history of SOA, contrasting SOA and Microservices as movements rather than technology, and pointing out the lessons the microservices movement needs to learn from the fate of SOA.

When Should I Use Them?

Like any architectural style:

Microservices provide benefits...

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.

- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go

wrong.

- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

...but come with costs

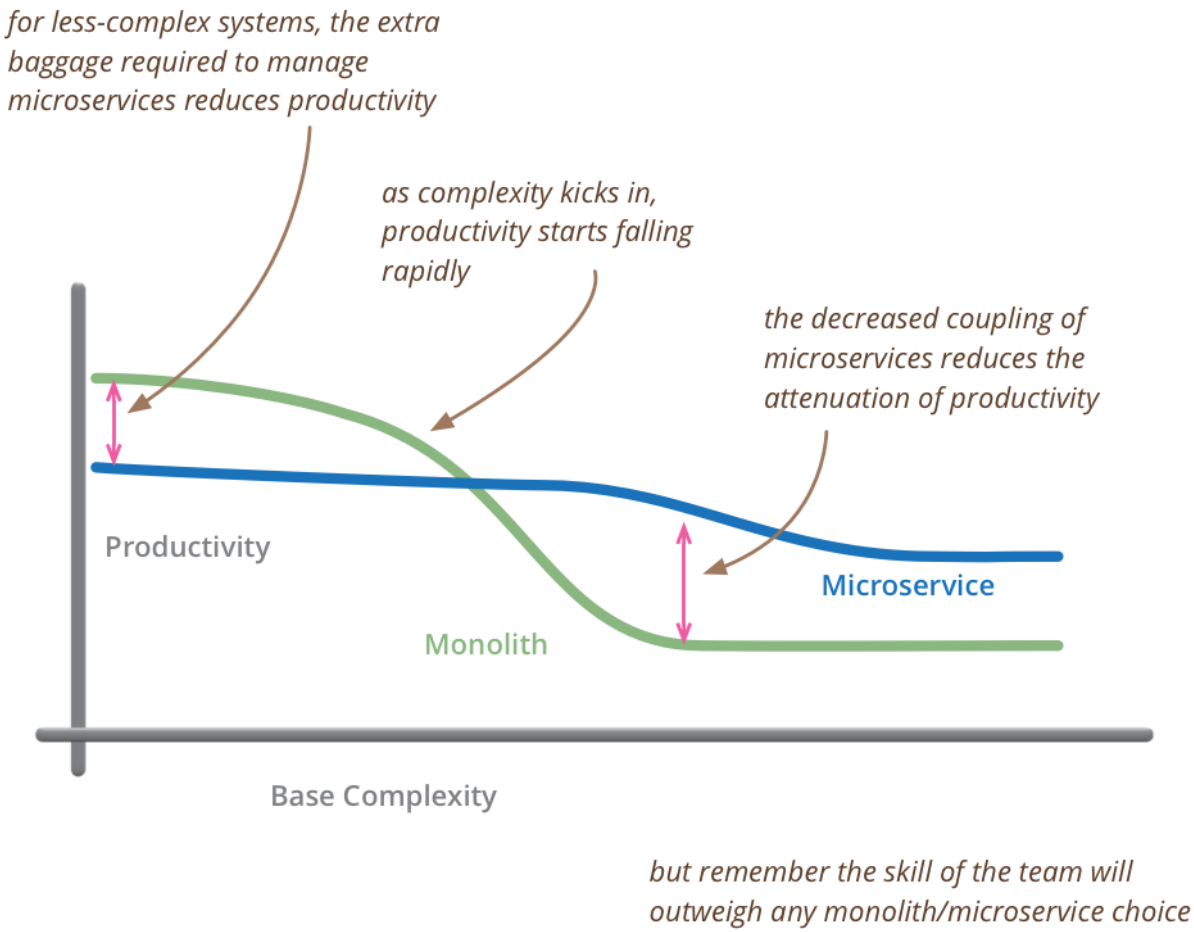
- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

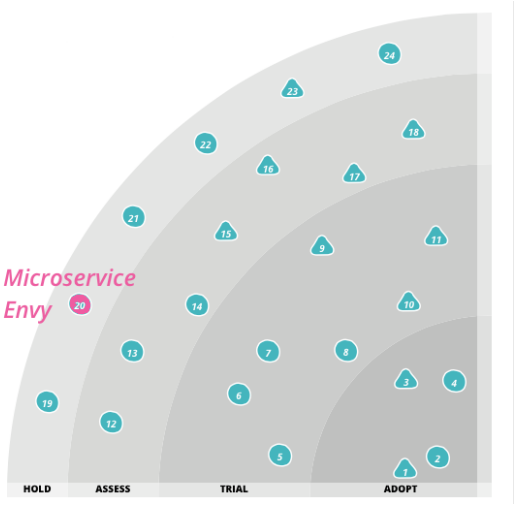
(from Microservice Trade-Offs)

I like to think of the MicroservicesPremium, a cost we pay in reduced productivity to learn and manage this style. As a system becomes more complex, this premium is outweighed by the fact that the style reduces the productivity loss that increasing system complexity imposes on us. But it's a price only worth paying for more complex software endeavors.



This complexity baggage that microservices bring to smaller applications is why many people favor a Monolith First strategy. With this approach, even if you think it's likely that microservices will pay for their premium, you begin with a monolith anyway. Once you realize microservices are worthwhile you decompose or sacrifice your initial monolith. Stefan Tilkov argues against this, saying that in practice it's too hard to build a monolith that will split apart easily. Both approaches, however, agree that you should not attempt microservices unless you know the domain really well.

As the microservice hype snowballed in 2014, the ThoughtWorks Technology Advisory Board (TAB) became concerned about projects choosing a microservices approach without good cause - an affliction we dubbed Microservice Envy.



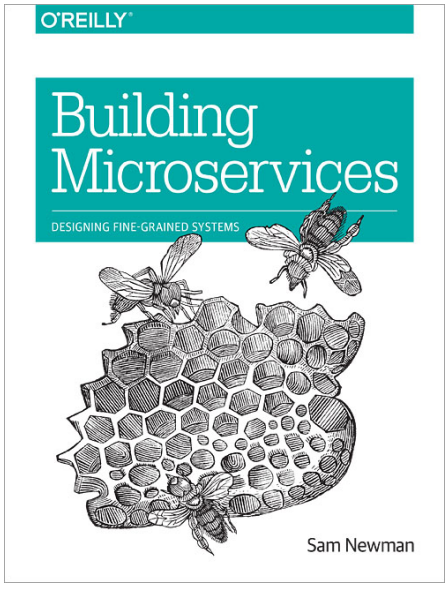
Scott Shaw, ThoughtWorks Head of Technology for Australia, discussed the problem of microservice envy in this podcast, recorded at a meeting of the TAB in March 2015 (18 minutes).

“While our experiences so far are positive compared to monolithic applications, we're conscious of the fact that not enough time has passed for us to make a full judgement.”

-- James Lewis and Martin Fowler

It's too early so far to make much of a deep judgment on how effective microservices are. You only really know how modifiable a software system is after several years, and after significant churn in the development team. It's likely that there are many factors in the surrounding environment, both political and technological, that affect the success of microservices. As with most of these issues, I don't expect we'll get conclusive answers any time soon - we can only hope to gather what evidence we can as teams share their experiences.

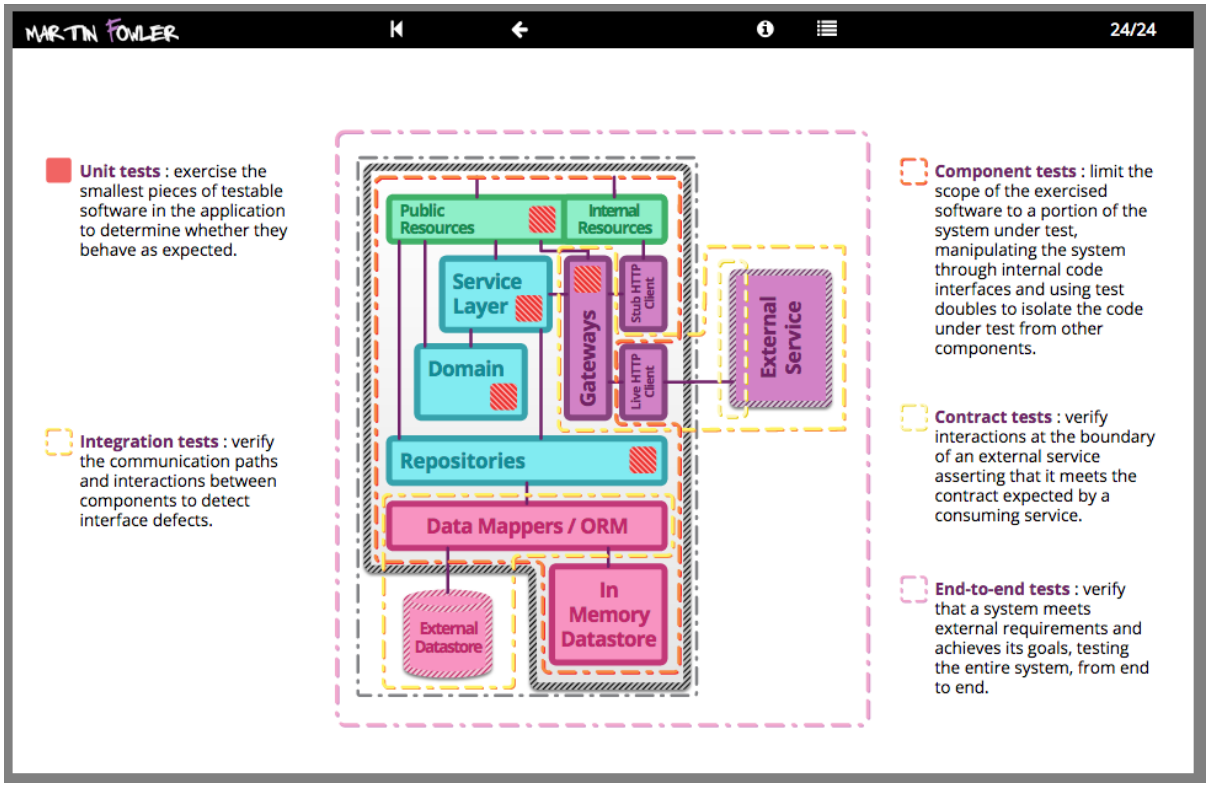
How to Build Microservices



My colleague Sam Newman has been involved in many of our efforts with microservices worldwide during the last few years. This book pulls together lessons both from our work and what we've learned from other organizations who have been sharing their experiences. As such it is the best place to start for a cohesive account of what it takes to build a system using the microservices style.

Testing

I've always been a firm advocate of integrating testing into your development process, pushing towards the notion of Self-Testing Code.



Toby Clemson put together an infodeck that examines testing when building a microservices systems: the various kinds of tests to consider, and how various forms of tests provide different ways to probe a system for bugs.

How big?

The term *micro*services puts a lot of emphasis on the size of the services, a point that most practitioners find to be rather unfortunate.

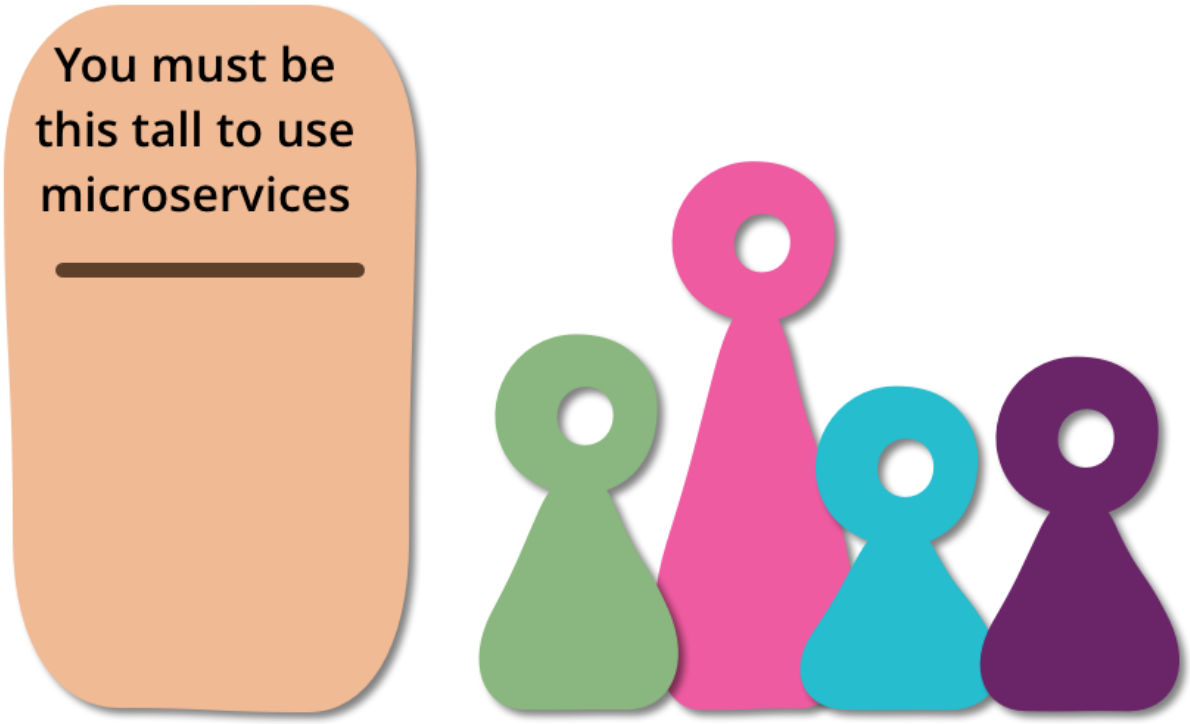
- Stefan Tilkov argues that you should start from the overall application and figure how it can sensibly be carved up.
- Sam Newman's talk emphasizes that you should derive your microservices based on the DDD notion of Bounded Context

Security

Microservices raise important security questions. Potentially they provide an opportunity to provide fine-grained control with data that has different authorization needs. But like most advantages of a microservices approach, there is complexity in setting things up. Chapter 9 of Sam Newman's book covers this in some detail, and he gave an overview of these issues in a talk at microXchg. You can also find a handy list of important questions to ask from Graham Lea.

Talks

- Sam Newman on Deploying and Testing microservices
- Sam discusses the practical implications of running more fine-grained microservices and how to design, manage, and monitor such systems.
- Eric Evans talks about the interplay of Domain-Driven Design, microservices, event-sourcing, and CQRS.



Before you go into production with a microservices system, you need to ensure that you have key prerequisites in place

- Rapid Provisioning
- Basic Monitoring
- Rapid Application Deployment
- Devops Culture

"Any organization that designs a system ... will inevitably produce a design whose structure is a copy of the organization's communication structure."

-- Melvin Conway, 1968

An important characteristic of microservices is that they are organized around business capabilities. Software architecture is generally a human construction

and most wise architects understand the power of **Conway's Law**.

- Melvin Conway's definition of his law and links to the original article.
- Sam Newman expands on the role of Conway's Law in software architecture and its relevance to microservices.
- At this talk from goto Chicago, James Lewis explains why the big issue with implementing microservices isn't testing, deployment, or versioning. Instead it's organization design
- Microservices fit well with the notion of a Business-Capability Centric IT organization - defined here by Sriram Narayan

Who Has Used Them?

The discussion around microservices is built upon early experiences with this style (whether they used the name or not). Fortunately many early users of microservices have been happy to talk about their experiences: here's a few samples.

Lessons from the Frontline

My colleague Zhamak Dehghani pulls together the lessons that ThoughtWorks has learned from our microservice experiences, where we've seen both successful projects and had to rescue projects with ill-conceived microservice architectures.

The Netflix Experience

Netflix is one of the poster children for microservices, having gone into a major redesign of their systems along microservice lines in recent years. Adrian Cockcroft led that effort and his talk on Migrating to Microservices sums up much of what they learned.

In the early 00s, **Amazon** made a now-famous transformation from the Obidos monolithic application to a service-oriented architecture with encapsulated databases and small, "two-pizza" teams. Although Amazon has never used the term "microservices", those in the microservices movement draw a lot of inspiration from their experience. The 2006 ACM Queue interview with Werner Vogels, remains the best source for information on what they did.

Werner Vogels, Amazon's CTO, talked about Amazon's experience with services in 2011.

> *For us service orientation means encapsulating the data with the business logic that operates on the data, with the only access through a published service interface. No direct database access is allowed from outside the service, and there's no data sharing among the services.*
>
> *-- Werner Vogels*

REA Group is an Australian web company focusing on real estate, running the realestate.com.au website. In the early 2010s they began to replace their monolithic applications with a microservices approach, with around 60 microservices deployed after a couple of years. Beth Skurrie, Evan Bottcher, and Jon Eaves share their experiences with this, describing how they attacked the monolith hairball, and why they discarded their integration tests in favor of

contract tests which led them to build (and open-source) the useful contract test library Pact.

Soundcloud moved from a Ruby on Rails monolith to a suite of microservices. Phil Calçado described their experiences over several blog posts.

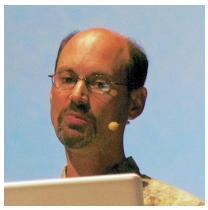
- How we ended up with microservices describes how the desire to improve cycle time and productivity led them to microservices.

  This post was written later than the others, but is best read first, followed by a three part series that goes into the engineering details.

- Part 1 talks about microservices that build on top of their existing monolithic "mothership".
- Part 2 discusses how they broke apart the monolith.
- Part 3 looks at how they build microservices in Scala and Finagle

Here's a couple more blogs I found useful for describing practical lessons on microservice usage.

- Alexandre Carvalho
- Karma - Stefan Borsje



Randy Shoup describes experiences of working with microservices from his time at **eBay** and **Google** . He focuses on the common evolutionary path from monoliths to microservices and paints a picture of a mature services environment at Google. It's also worth watching his follow-up interview (with transcript), which elaborates on some of the lessons from this experience.