# Why Companies Adopt Microservices And How They Succeed

Asim Aslam Follow

Jul 5, 2016 · 15 min read

This post delves into the non-technical aspects of adopting microservices within a company. With the world now being driven by technology, companies must learn to adapt, stay agile and continue to increase velocity in their core business.

The transition towards a microservice architecture are usually thought of as a process driven by technical limitations of an existing system. While that's true in most cases, many of the other reasons for moving in that direction are led by higher level requirements related to the business and team dynamics.

The post will cover motivations, the migration path, what success may actually look like and the tradeoffs which are made in such a transition.

Much of what's discussed will be familiar for people who've gone through this journey themselves. It's important to share those experiences so others can learn from them and avoid potential pitfalls, but also make informed decisions about whether migrating to microservices is right for them.

Before we get into that, just a little bit about what microservices are.

## What Are Microservices

> *Loosely coupled service oriented architecture with a bounded context*
>
> *Adrian Cockcroft*

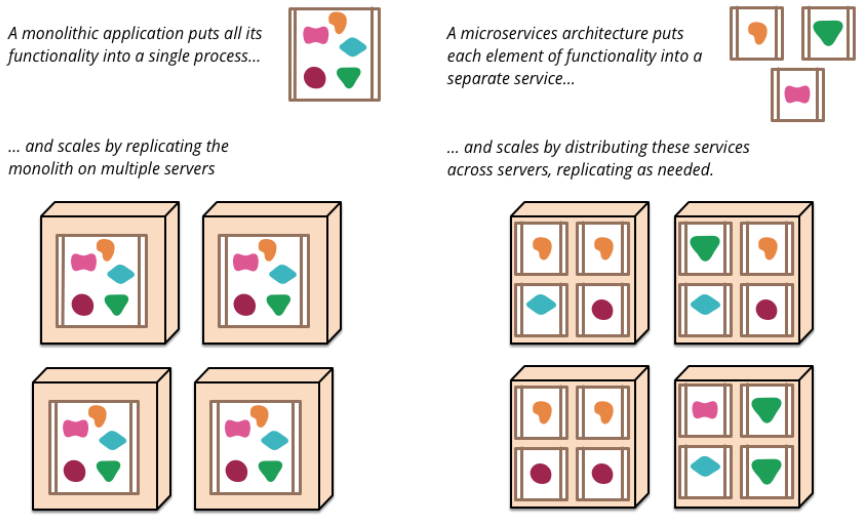
Microservices in a nutshell is a software architecture pattern used to decompose monolithic applications into smaller services that do one thing well, interact over the network and are built and managed independently.

In an introductory post we discussed what microservices are, their benefits and how Micro plays a role. This blog post assumes a basic

understanding of microservices so if you're interested in learning about them first check out the introductory post here.

If you want to read more about the technical aspects of migrating to microservices check out the blog post by Matt Heath on Hailo's Journey Into Microservices.

There's also a great article by Martin Fowler that covers the fundamental ideas behind microservices which can be found here.




Let's dive into the motivations for adopting microservices.

## The Motivations

> *Speed, Scale, Agility*

Have you ever heard a company speak the words:

*"We need to move slower"*

Most likely not.

Usually the goal is to deliver a quality product in the shortest time possible. It sounds pretty straight forward but it's not quite that simple. A product is constantly evolving and "feature complete" is most likely a moving target. We're defining new requirements, integrating with additional partners, navigating the intricacies of business in various regions of the world. The business is continuously growing and evolving.

In the early days, the total number of team members is still low, the codebase has a small footprint and the product is focused on doing one thing really well. It's fairly common to see extremely fast product cycles in the first couple years of a business and it's the most common trait identified with startups.

As things pick up the company adds more people to fuel growth, the focus starts to shift into new markets, there's a need to scale with the growing demand internally and externally. All the while the underlying system, which worked great in the early days, is now starting to show signs of wear and tear.

What used to take a few days is now taking weeks if not longer. Teams are struggling to execute in parallel. There's a lot of cross coordination overhead and people are stepping on each others work in a monolithic codebase.

Developers dread pushing software to production, the entire system occasionally falls over because of traffic spikes. Teams start pushing Q1 deadlines to Q2 or even Q3, and the inevitable slow down begins.

The company is no longer executing effectively. Something has to be done.

The technical team may decide to iteratively break out pieces of the system. This can somewhat help but 6–12 months down the line there's a realisation that it's still a mess with most of the previous issues still in existence.

It's become impossible to add anything new without causing more problems.

. . .

These are issues faced by many companies at some point in their lifetime. From a business standpoint it becomes clear that the current organisational structure and technical architecture will mostly likely hinder the progress of the company in the long term.

The day 1 goal of the company still stands true. To deliver a quality product in the shortest space of time possible. However the company is now being limited more so by itself than the market or it's competitors.

At this point it becomes clear that there's a need to address these issues so the company can be structured in a way that will enable it to execute and scale.

## The Road To Re-architecture

> *It's not about technology, it's about people*

The motivations are clear but before pursuing any path it's important to begin by discussing the existing issues with those directly affected.

The team.

Jumping too quickly into architecting a new system without understanding the existing problems could not only result in building the wrong solution but also wasting a significant amount of time and money.

Every part of the organisation will be affected differently. Many of the problems you'll hear are not specifically technical but more so about cross team collaboration, prioritisation of goals, etc.

Technology may be at the core of the product offering but the day to day operation of the company is focused on people and how they work together.

### Seek Out Expertise

While discussing the company problems and potential solutions amongst the team is imperative it's also worthwhile seeking an outside perspective. The experience of those who may have seen these problems before can be immensely valuable.

This doesn't necessarily mean looking to consulting firms. The road to re-architecture is nothing new. There are many experienced teams or individuals from other companies who've already been down this path and can offer great insight.

Ask someone to spend a few days talking to people at the company and looking at the technology, so they can provide a well formed opinion about how you should proceed. By seeking out a number of outside experts you'll be able to make an informed decision about what the next steps should be.

In some cases, microservices may not actually be the needed solution. A lot of deficiencies may be process related and can be rectified by focusing on product lifecycle management with additional minor technical changes.

You may receive contrasting views about what the right approach is. This will be because of past experiences with specific elements in place such as strong growth and revenue, senior engineers with past architecture experience, etc.

It's important to listen to the various experiences and opinions, however at the end of the day your team needs to make a judgement call about what your collective priorities and the approach that's best suited to you.

## Existing Models

If you do decide that re-architecture is the path to take, again seek out the experts in the field who've done this before and identify models that already work.

The things you really want to know are:

1. What team structures and dynamics work
2. Which technologies make sense for you
3. How the architecture is built and scaled
4. What the migration path is from the legacy system

The patterns for microservice platforms have emerged in recent years with the likes of Netflix being one of first to actively speak about their journey. This process of re-architecture included a move to the public cloud leveraging Amazon Web Services.

Others like Spotify have discussed engineering culture and structural practices that worked for them. There's a wealth of information on the successes of squads, tribes, guilds, mission teams and so on.

## Prototyping

With all that in mind, diving head first into a complete rewrite is unadvisable. The best way to begin is with some rapid prototyping.

In this phase a group of engineers should be given time to identify and test potential technologies and architecture patterns. This is ideally a 2–

4 week period that results in sample solutions, demo platforms and a summary that can be presented to the rest of the company.

It's important to discuss what will and won't work as a team rather than making decisions in a silo. The entire company will inherently be users of the new platform and they should have a say in what they'll be using day to day.

A lot will be learned during this phase. Whether the team has the existing skills and experience to build a better system than before.

What timelines potentially look like in building the first version of said system. And whether it's actually going to solve the problems you're facing as an organisation.

If it looks like any of these issues are going to arise in the next phases of the process it's ideal that they're surfaced now in this low risk low cost setting.

Do not be afraid to take a step back to reassess if it no longer makes sense.

If the prototyping phase was successful, it's time to move on to the next step. Building the new platform and validating that it actually works.

## A New Architecture

Looking back, it may become clear that the existing system was cobbled together manually and formed a unique architecture which was only fully understood by a few engineers within the company.

Is there a disaster recovery plan? Is there automated failover? How long would it take to rebuild from scratch? Is it scalable? Is it secure?

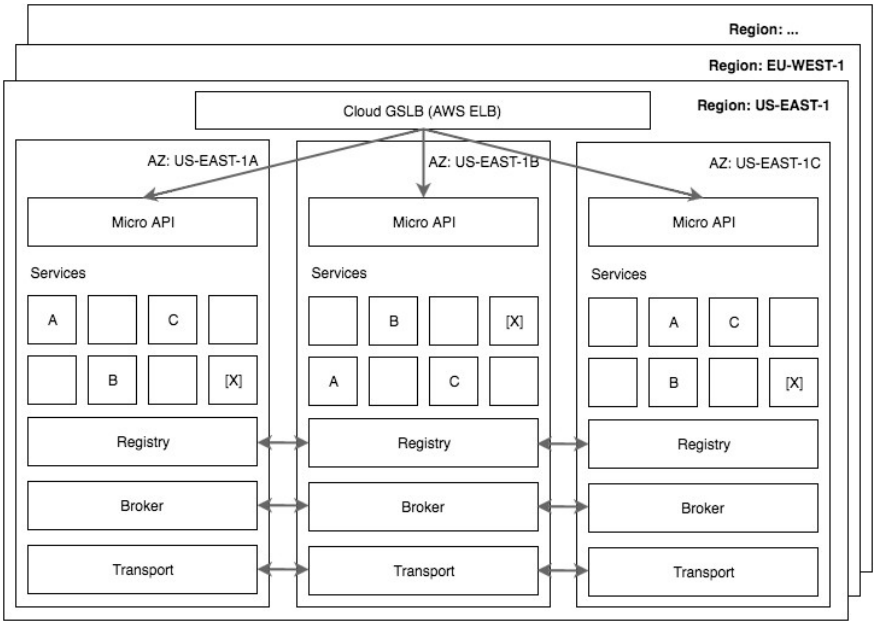
The answer to these questions probably wasn't met with a confident "yes". This is the time at which many take the opportunity to address the above questions along with all the other problems discussed before.

So what's needed?

A clean slate. A fully automated highly available cloud native platform.

As mentioned before, the pattern and tools to develop a globally scalable microservice platform already exist. Along with the results of the prototyping phase, the team should be able to produce a repeatable model that becomes the basis of the new system.

As this is a mostly non-technical post, we'll leave out the details of how that's actually achieved. Reach out if you would like to discuss it further.




## The Migration Path

If you're providing a service to customers then you know business continuity is absolutely critical. Any disruptions in your service affects the customer experience and ultimately the company's revenue.

A Service Level Agreement (SLA) is common practice and you want to ensure this is maintained throughout the migration process. It's key to remember that the business is first priority but maintaining this SLA during a migration process is a tradeoff between uptime and developement time.

If teams are spending a significant amount of time migrating from the legacy system to the new system, the company is essentially unable to execute on other important aspects of the business. It's something everyone must be wary of before even beginning this journey.

With that said, no matter which way you cut it, the migration path will take time. The tried and tested method is to take an existing function of the legacy system, build a new version as a microservice on the new platform and gradually shift request traffic over.

There's a few important points to keep in mind here.

1. **The entire company does not need to focus on the migration**. A single team, ideally the team which built the prototype, should take on this task. This allows the majority of teams to continue work on their existing projects and minimises any further disruption.

2. **Pick a well understood function of the system**. It's important that the team migrating over the first feature has a deep understanding of that feature, how it will be architected in the new system and how to rollback if there are any significant issues.

3. **Set achievable milestones**. A couple weeks is likely not long enough to migrate that first feature but if you're still working on it after 6 months, something is clearly going wrong. 3 months is a good timeframe in which to develop a plan, write the software, and by the end, serving production traffic on the new platform.

After successfully migrating over the first few features/services a pattern will emerge in terms of the technical approach, timelines and team requirements.

The process is one that can take many months or even years depending on how you prioritise the migration. Everyone should understand that this is not an overnight transition and the re-architecture path must be as much a part of the roadmap as any new feature or product development.

## Platform as a service

> *Your developers are customers*

As the migration is underway, patterns emerge and a playbook is formed for how this process should continue on successfully. The "platform" team will have developed a deep understanding of all that's involved.

The rest of the company should be kept in the loop about the progress of the migration and will have accrued some knowledge of how the new platform works during the journey.

Further effort now needs to be made to engage them in increasing this understanding and opening the platform up to them so they can participate in the migration of services themselves or build new products on the platform.

This is where we really start to address the need for Platform as a Service.

## The Need For A Platform Team

You'll have heard of PaaS as some form of hosted offering for running applications. The benefit of such a system is that it allows you to focus primarily on product development and less on infrastructure management.

When running a microservice platform internally you wan't this same advantage. Teams should be focused on delivering business value rather than on the complexities of automating and managing distributed systems.

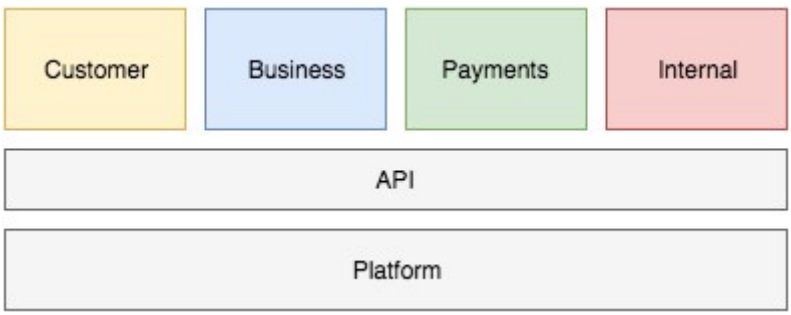
A platform team's primary objective would be to deliver platform as a service internally via APIs, dashboards and developer tools while maintaining a highly available and scalable system.

There are varying opinions on this methodology. Spotify for instance have opted for a polyglot architecture in which every team chooses their own technologies and manages their own infrastructure.

The likes of Google, Facebook, Twitter, Netflix and Hailo have opted for the alternative model where a set of services are provided to the company much like a cloud provider.

As with any choice, it's a tradeoff. In this case time versus flexibility. By allowing teams to manage their own infrastructure, much of the time is spent there. If platform as a service is provided instead, that time is recouped but a more opinionated service is offered.

Companies have to rationalise these decisions internally based on their skills, experiences and what they deem to be most important to them. An argument can be made for either methodology.



## Self Organising Teams

With the power of a microservices platform, feature development and testing new ideas should be quick. If anything 24–48 hour hackathons can become a regular occurrence and originate new products.

One method of really harnessing this power is around the idea of self organising teams. For the most part you will have mission teams or product teams which operate autonomously and focus on key long term objectives. Self organising teams are a way of augmenting this structure.

An example scenario. A group of 3–4 employees with complementary skills from various teams identify a problem that needs to be solved or they have an idea for a new feature which they feel strongly about developing.

With the approval of their managers, they form a temporary team for a two week sprint to build out a solution. They're able to reuse existing micro services on the platform and make use of all the self serve tools which really accelerates the process.

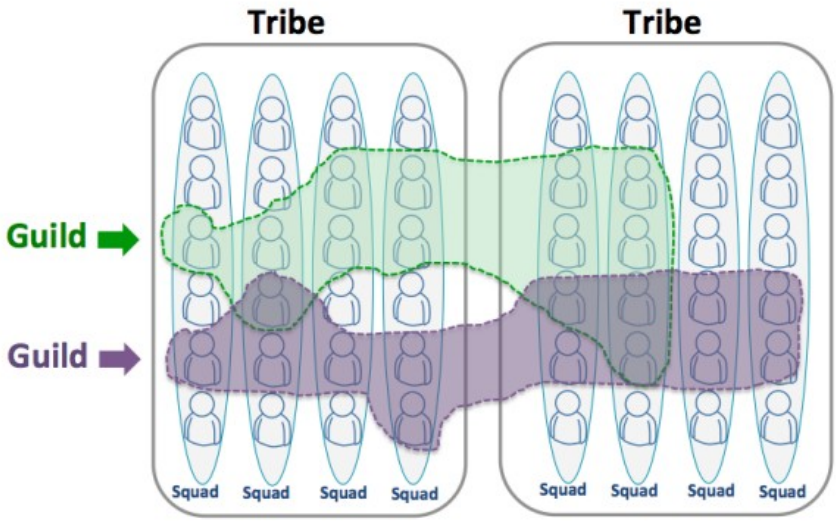
By the end of the sprint they ship a new set of services to production without having to coordinate with any other teams and also more importantly without having to modify a single line of code in any existing applications.

If the solution proves successful they make the case for creating a permanent team to further product development.

Self organising teams are a really great organisational enhancement driven by the advantages for microservices.

Another common name for self organising teams is Guilds. It's a term popularised by Spotify as part of their organisation structuring methods.

Learn more about Spotify's model here .




## What Success Looks Like

> *Zero to production in hours or days, not weeks or months*

Migrating a running business from one platform to another can be a long and arduous process. The measure of success should not solely be judged on reaching the end of that process but the goals achieved incrementally along the way.

Revisiting the problems discussed earlier on.

1. Slow development process
2. Cross team coordination
3. Brittle monolithic software

Even while working on the migration, all of these issues should more than likely be solved. Microservices offer a form organisational speed, scale and agility via distributed systems technical architecture.

The monolith application is decomposed into a number of smaller applications that can be worked on and released independently by teams. They do not have to be aware of technical details of services developed by other teams but can consume them via strongly defined APIs.

Teams are likely to have faster product development cycles because of this and are able to operate autonomously. Where cross team collaboration does need to occur, it can be done in a self organising fashion or by simply consuming APIs of their services.

Further to that, the platform should now offer a highly available and globally scaling system which product teams can leverage without having to think about the details of building distributed systems.

Success is still measured as the ability to deliver a quality product in the shortest time possible but now with an ability to scale the organisation and technology.

. . .

While undergoing this process a lot will have changed, some of which is under your control and a lot of which is not. The market may have evolved, the company grown in size, priorities perhaps shifted.

It's important to evolve and adapt with the shifting lanscape. Do not be afraid to try restructuring teams, launching new features, etc.

The platform is a strong foundation which should allow you to move fast, launch new initiatives and iteratively reshape the business in any way imaginable. Think of it as a competitive advantage.

## The Big Bang

> *Big bang is the death of a company*

No post on re-architecture is complete without a few words on the big bang.

Big bang product development is a process by which the product is only launched once completed. The entire endeavour can range from anywhere between 6–12 months at the very least but is likely significantly longer.

Following this method in a re-architecture process would essentially involve rebuilding the entirety of the existing product and then cutting over to the new platform when complete. You can already picture the countless ways in which this could possibly go wrong.

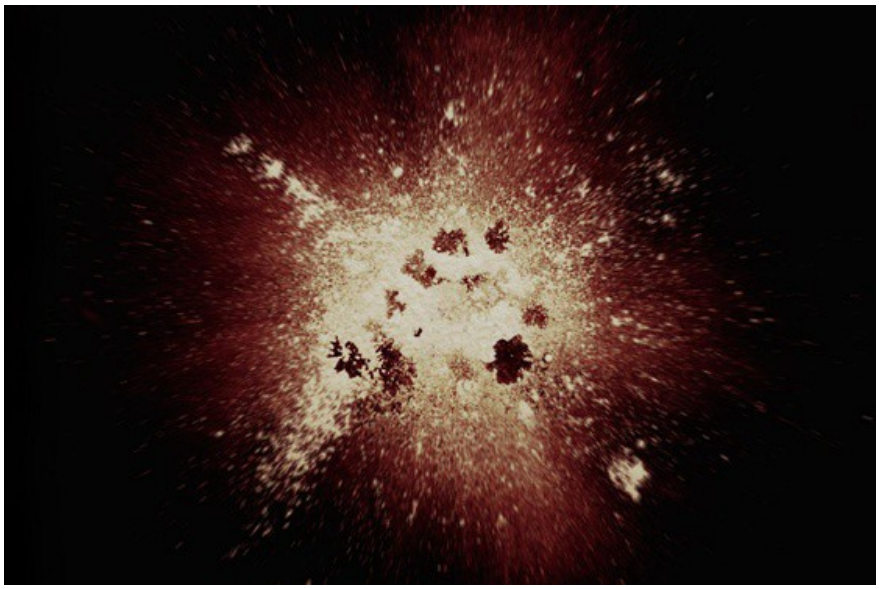
Building with a big bang methodology can be a conscious decision but in most cases it's a slow growing beast that no one sees coming. What's deemed feature complete is a boundless list. The product needs some tweaking and then a little more and then just a little bit more. It's never quite ready to be launched.

There is a high failure rate associated with this form of development and rightfully so. Significant time and money can be lost to big bang development.

How does the team know what they're building is actually still viable in the market today? The opportunity could be lost and without feedback along the way it's hard to tell whether you're still going down the right path.

When taking on the process of re-architecture we know what the end goal is but the task of getting there should be broken down into smaller deliverables that can be achieved in short spaces of time.

Avoid the big bang at all costs. Aim for two week sprint deliverables with quarterly milestones for larger chunks of work. Each cycle moves you closer to the end goal while producing tangible results in the interim.



## The Tradeoffs

> *Microservices are not a panacea*

When it comes to scaling technology and teams, there's no silver bullet. Success is a moving target. What works in a team of 10 will not work for 100 will not work for 1000 and so on. The same could be said for most choices within a company, whether their technology related or not.

Everything is a tradeoff. Managing a microservices architecture can require significantly more operational work but the benefits of such a

system are seen in the increased speed of execution and scalability.

We see a natural path towards the adoption of microservices within organisations. At a certain scale it becomes more difficult to manage monolithic systems and the tradeoffs in decentralising both people and technology are worth making for the benefits they provide.

Many companies have undergone these learnings by trial and error, in the days before we learned of microservices, before GitHub and open source software became popular, back when it was deemed a competitive advantage to keep all things a secret.

Google is a prime example. They were really forced to learn, iterate and evolve due to the scaling requirements and their need to compete with the giants of their era like Microsoft and Yahoo.

Most of us will never quite reach the scale of Google but their experiences and knowledge of scale is incredibly valuable and we can leverage those to our advantage.

More and more companies are choosing to adopt microservices at some point in their journey, whether it be a conscious effort with large scale R&D or a more iterative and methodical approach.

It's clear that there are tradeoffs in adopting microservices and many challenges along the way. There's no one path to success but we're now seeing patterns and processes emerge that may result in microservices being the ideal way forward for a number of companies.

. . .

If you're interested in adopting microservices contact Micro to help increase the odds of success and leverage technology for your needs.