hys-enterprise.com

Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience – HYS Enterprise

Anna Rud Blog Editor of HYS Enterprise

14-18 minutes

As startups, companies can't predict how large they'll become and how cumbersome their applications will get. They build their apps using technologies that satisfy their immediate needs. But as startups grow, these technologies slowly cease to meet their business requirements. One day, you find your business hampered by technologies.

Even the world's leading companies – including Netflix, Uber, and Amazon – have been through this. When building monolithic applications, they didn't know how difficult it would become to scale and maintain them in the future.

Let's dive into these leaders' stories and learn about all the pros and cons of monolithic apps and microservices. These insights will be useful while you're planning your own business application architecture.

A word on monolithic architectures

A monolithic application is a single-tiered software application in which all components are combined into a single program. Those

components include the client-side user interface, server-side operations, business logic, the database layer, and integrations. They're interconnected and interdependent, meaning when one component needs to be updated, you have to make changes to the entire application. And if one component goes down, it will harm the rest of the system as well.

A great example of a monolithic application is an e-commerce platform where customer profiles, order management systems, inventory management, and all other functions are interdependent, managed in one place, and stored in a single database.

While this may sound unreliable, a monolithic architecture has lots of benefits, and it's perfectly suited to certain types of applications and businesses.

First off, it's much simpler to develop, test, and deploy monolithic applications. Since a monolithic architecture has long been considered traditional, most developers have solid experience building them. This way of developing programs is easy to understand, and it works well with lots of existing tools, development environments, frameworks, and scripts.

Deploying monolithic applications is also easier than deploying other architectural solutions. Since all code is deployed at the same time, all you need to do is copy a single archive to a server.

The looser the dependency between app components, the harder it is to figure out what's broken. In a monolithic app, all functions are tightly coupled and can be tested at the same time, which simplifies the testing process.

Taking into account all of these ups and downs, there are several scenarios in which a monolithic architecture is the perfect choice:

- **A simple application.** When an application consists of a small number of features and doesn't require much business logic, scalability, and flexibility, monolithic works better than any other architecture. Microservices are a big trend, but that doesn't mean every single app needs them.
- **Quick launch.** If you need to get your app up and running as soon as possible, a monolithic app will be the right decision. Later on, you can break in into microservices if needed.
- **Proof of concept.** If you're building an app based on a new, unproven business idea, it's faster and easier to build it with a monolithic architecture.
- **Small team.** If there are only about five people on the team, a microservice architecture might be too complex and involve too much overhead. It's better to start with a monolithic app and break it into microservices when there are more specialists available.

  While perfectly suitable for small projects, a monolithic architecture starts to crack when an application becomes too bulky. It becomes harder and harder to maintain the system, especially when there are several development teams working on the project. One day, you realize that debugging takes more time than building new features.

These and some other challenges were reasons for many leading companies to switch to microservices. Without further ado, let's explore their stories.

Amazon: From a tiny bookstore to the world's largest e-commerce marketplace

Starting as a small bookstore, Amazon has grown into the world's largest e-commerce platform, selling books, magazines, music, DVDs, videos, electronics, computers, software, apparel and accessories, shoes, jewelry, tools and hardware, houseware, and more.

Much like other business giants, Amazon initially was built as a two-tier monolithic app. Over time, as they started to grow, Amazon faced a pressing problem with their system's scalability.

Here's what Rob Brigham, the Amazon AWS Senior Manager, said at Amazon's re:Invent 2015 conference:

> *"It [the Amazon.com service] was architected in multiple tiers, and those tiers had many components in them. But they're all very tightly coupled together, where they behaved like one big monolith. Now, a lot of startups and even projects inside of big companies start out this way. They take a monolith-first approach, because it's very quick, to get moving quickly. But over time, as that project matures, as you add more developers on it, as it grows and the code base gets larger and the architecture gets more complex, that monolith is going to add overhead into your process, and that software development lifecycle is going to begin to slow down."*

All the bottlenecks that come with a monolithic architecture started to appear: development and deployments were taking too long, vast databases were hard to manage, adding new features to the system became tough and risky, dealing with fluctuating website traffic became problematic. All this caused frequent outages and

financial losses for Amazon.

That's when Amazon's architects understood that the current path was limiting the company's growth and that it was time to move toward a more flexible architecture. The idea was that every feature of the platform would be provided by a microservice, and that all those services would interact with each other via APIs (application programming interfaces). The term microservices didn't even exist at the time. But basically, that was what they set out to build.

But while such a system seemed flexible and convenient, there was a big chance that breaking the system into hundreds of microservices would turn the development process into a total mess. Trying to prevent chaos, Amazon came up with several fundamental rules.

- The "two-pizza team" rule means that if a team responsible for one microservice couldn't be fed with two pizzas, it was too big. When a team consists of fewer than 10 people, they don't need lengthy meetings to keep everyone updated. Each team member knows what to do and the entire team is usually more productive.
- The "you build it, you run it" rule means that developers were fully responsible for the services they built – for developing the functionality and for all operations, including DevOps.

By implementing these rules, Amazon managed to build smooth and clear processes where small groups of people are responsible for certain parts of work and understand how they fit into the big picture. That said, productivity started to slow down as developers were spending lots of time on operational work, performing the same tasks over and over to keep their services available. That's when the idea of Amazon Web Services (AWS) hit them.

Using the cloud for microservices helped Amazon automate operational processes and allowed them to scale services depending on traffic and current business needs. This also resulted

in the adoption of a continuous delivery approach, which allowed Amazon to develop faster and deploy more flexibly.

Check out this talk by Werner Vogels, VP and CTO of Amazon, to find out more about the company's exciting journey from bulky monolithic system to cloud-based microservices:

Werner Vogels – Amazon and the Lean Cloud from HackFwd on Vimeo.

Netflix: Serving 139 million subscribers in seconds

Netflix was one of the first businesses to realize that a monolithic architecture doesn't work well for a complex application.

Way back in 2008, a single mistake caused massive data corruption and led to several days of downtime. That's when Netflix architects decided to move the entire application from a monolithic architecture to AWS cloud-based microservices.

The main goal of this migration was improving availability, scalability, and speed. They wanted Netflix to be available around the clock, work fast, and scale easily.

Since components in a monolithic app are tightly coupled, a single issue can bring down the entire system. A microservices architecture allowed Netflix to break up the system into independent services: one service stores all watched shows, one is responsible for monthly credit card payments, one analyzes watching history and suggests similar shows and movies.

Breaking the application into over 700 microservices, each responsible for one functionality, allowed engineers to make changes to any part of the system and be sure the application would never go down entirely: issues in a microservice can harm that microservice but won't affect the rest of the system. This helped Netflix minimize outages and achieve better availability.

At the time, Netflix was growing rapidly: more subscribers, more hours of video watched per day, more countries and devices supported. They couldn't build data centers fast enough to satisfy their needs.

Moving to AWS and a microservice architecture solved their problems completely. With the AWS cloud, Netflix engineers were able to scale the capacity of services in a matter of minutes.

A microservices architecture is what allowed Netflix to independently scale certain services instead of scaling the entire monolithic system. For instance, they can independently increase the capacity of the service that deals with customer profiles. This also means they pay only for the resources they use and never pay extra for what they don't need.

Another distinct advantage of microservices is the speed and agility of development. Netflix engineers got the opportunity to develop, test, and deploy services independently, which allowed them to build more than 30 teams that could work on different parts of the system without having to wait for each other to finish. This helped the company develop faster and boost overall performance.

Today, Netflix streams about 250 million hours of video per day to more than 139 million subscribers from 190 countries, and the company continues to grow. Taking into account all the advantages gained by adopting a cloud-based microservices architecture, it becomes clear that timely migration was one of the reasons why Netflix skyrocketed.

You can learn more about the challenges Netflix engineers encountered during and after the microservices migration as well as what lessons they learned from it.

Uber: Built for a single service in a single city

As a startup, Uber needed only a few features: connecting drivers and riders, billing, and payments. The application needed to offer a single service in a single city: San Francisco. That's why it was built as a monolithic app: at that time, this app was able to meet all the business requirements, implementing core business functionality and offering a clean codebase.

When Uber expanded into more cities, they started to introduce new products and services. The application started to grow rapidly, and that's when maintaining the monolithic system became a real challenge.

First of all, deploying the codebase meant deploying everything at once, which was hampering continuous integration. The other drawback was that only developers who had been working for a long time at Uber could make changes to the system: a single change became a huge responsibility because of dependencies between the app's components. Here is what they say on the Uber Engineering blog:

> *"Adding new features, fixing bugs, and resolving technical debt all in a single repo became extremely difficult. Tribal knowledge was required before attempting to make a single change."*

Uber decided to break up the monolith into multiple codebases to form a service-oriented architecture (SOA) – or, to be more precise, a microservices architecture.

Since software built as microservices is broken into small, independent services, each component can be written in its own language, use its own framework, and have its own database. This helped Uber solve all their problems in the development process, build continuous delivery, and accelerate their business growth.

Migrating to SOA solved lots of problems. At the same time, it caused a few challenges related to obviousness, safety, and resilience.

Obviousness. Uber had over 500 services, each structured in its own way and sometimes written in different programming languages. Finding the appropriate service and using it became difficult. The solution they found was using Apache Thrift, a binary communication protocol for defining and creating services for numerous languages. Thrift helped them simplify the management of 500 services.

Safety. Various network connections and application programming interfaces (APIs) used for communication between components make microservices-based applications less secure. Thrift also solved this problem out of the box by binding services to strict contracts.

Resilience. To achieve resilience of components, Uber engineers wrote latency and fault tolerance libraries that ensure clients can

deal with failure scenarios successfully.

Microservices helped Uber unlock opportunities for business growth and overcome constraints.

If you're eager to dive deeper into service-oriented architectures and what Uber got from leveraging this approach, check out their story about how adopting microservices facilitated their business transformation.

Summing up

These are stories of leading companies that understood they were on the wrong path and decided to take a chance and completely rewrite the system. That choice appears to be one of the reasons they've become the businesses they are today.

The reasons for migrating to microservices were pretty much the same for all these companies: starting out small, they managed to significantly grow their businesses, at which point a monolithic structure stopped meeting their needs.

Here are some indicators that it's the right time to change the way you build your application and move to microservices:

- As your application has grown, the development process has gotten complicated; development and deployment take too long.
- Small changes can break the entire system, leading to downtime.
- Continuous integration can't be implemented properly.
- Collaboration between developers is complicated, as is the integration of new developers into the development process.

- You’re paying for redundant resources because of problems with scaling.

  If you’re encountering any of these issues right now, feel free to get in touch with us for a quick consultation with our CEOs.

  > Related success case: Find out how we helped Maxeda DIY group migrate to microservices and improve their business performance.