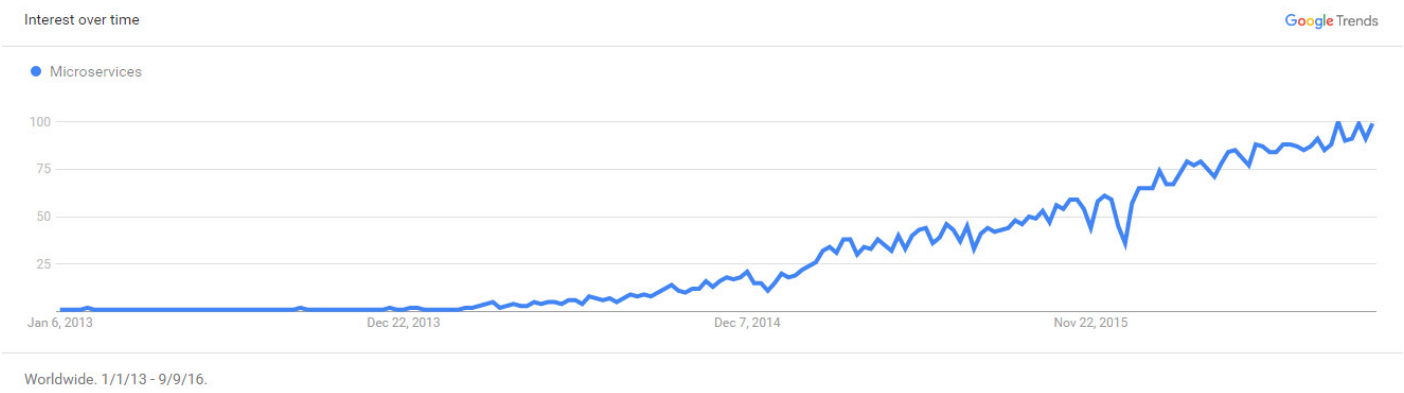
What's the Difference Between a Monolith and Microservices?

Application architectures have become a widely discussed and often debated topic in technology circles these days. The number of companies building complex applications requiring an architecture that will allow for high scalability, availability, and speed are growing rapidly. Many technology companies including Netflix, Amazon, Google, IBM, and Twitter Track this API have built applications that are extremely large and complex. These companies have chosen microservices instead of the traditional monolithic architecture because a microservices architecture is designed for handling complex systems. A microservices architecture also allows applications to be highly scalable, available, and fast.

This is the third article in ProgrammableWeb's microservices series and covers some of the key differences between a monolithic and microservices architecture. The first article in the series covers what microservices are and why they matter. The second article covers the role of APIs in microservices architectures.



The term microservices has been around since 2012 but didn't gain in popularity until 2014, when the term was used in several publications written by software developer and author Martin Fowler. - Google Search Interest Over Time - Data Source: Google Trends

Monolith vs. Microservices

Monolithic applications are simple to deploy as they usually require a one-time deployment to a single server. There are cases where a monolithic application could be deployed on several servers; however, a monolithic application is tightly coupled, typically has an entire IT stack dedicated to it, and the application is scaled as a single unit.

In contrast, a microservices architecture involves smaller discrete services-oriented software components that are loosely coupled but often grouped by shared commonality (functional adjacency, departmental ownership, etc) in what is known as bounded contexts. These individual services can be deployed and scaled independently of each other. A bounded context is a design pattern borrowed from the concept of Domain Driven Design. Bounded contexts often have specific integration points with other bounded contexts.

In some situations, decomposing a monolithic application into individual independent services allows the newly "composed" application to remain running and available even while a problem is occurring with one or more of the services powering the application. If a problem occurs with one of the components of a monolithic application, the usual result is that the problem will bring down the entire application. Back in 2008, when the Netflix application was monolithic, the entire Netflix website was brought down because of a single missing semicolon.

There are quite a few differences between a monolithic and microservices architecture. Some of the key differences involve the codebase, database, development teams, programming languages, and testing.

Codebase

A monolithic architecture means that there is a single and often very large codebase for the entire application. It is possible for a monolithic architecture to be well-written and of a sound modular structure in theory. However in practice, a monolith almost always ends up having a “big ball of mud” architectural pattern. In the words of Brian Foote and Joseph Yoder, authors of the paper “Big Ball of Mud,” most monoliths end up a “haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle.” A large tangled codebase makes it difficult for developers to understand and maintain the application.

A microservices architecture consists of multiple, smaller sized codebases; each service powering the application has its own codebase. It is possible for a microservices architecture to also become a “big ball of mud.” However, a microservices architecture is designed for better modularity making it harder for it to become a tangled mess of code. Multiple services, each with their own small codebase, are easier for developers to understand, maintain, and make changes.

Database

Most monolithic architectures use a single logical database, often relational, for an entire application or set of applications. Monoliths allow multiple pieces of data to be updated together in a single transaction.

A microservices architecture allows each service to have its own unique database system or share the same database system but have different instances. The persistent data for each microservice is kept private and is only accessible via its API. One drawback of this approach is that implementing transactions that update data owned by more than one service can be difficult. Instead of using distributed transactions, event-driven approaches are often used instead for database consistency.

Development Teams

Microservices teams often work independently of each other with each team in charge of building and deploying one or more microservices to production. Microservices teams tend to be smaller in size compared to the teams of a monolithic architecture and because each team is in charge of a specific service or set of services, are able to manage the entire microservices lifecycle. Smaller, independent teams with their own release schedules can help increase the speed, agility, and productivity of the application development process.

Languages, Frameworks, and Libraries

A monolithic architecture tends to be built with a small set of specific programming languages. Because the components of a monolith application are tightly coupled, it can only use a single version of a library which makes upgrading to a newer version of a library difficult. One or more parts of the system may break if the newer version of the library is implemented. However, other parts of the system may need the newer library to leverage new features. It is also difficult to implement other languages and frameworks if the application is monolithic as the entire application would have to be rewritten.

A microservices architecture consists of individual, autonomous services that can be built using any programming language. Each service can be developed with a different language, framework, and/or library. This allows each service to adopt new technologies and developers to experiment with new features without the risk of breaking other parts of the application. A microservices architecture also allows applications to be highly scalable and reliable. Each service can be scaled independently, while a monolithic application can only be scaled as an entire unit.

Testing

A monolithic application is built, deployed, and scaled as a single unit making it much easier to test than a distributed application. End-to-end testing can be easily implemented for a monolithic application. A distributed application is more difficult to test than a monolithic application. If one of the microservices of an application needs to be tested, it would have to be launched along with all of the other services that it depends on. The use of continuous integration can help offset the complexities that arise when regression testing new microservice codebases for their impact on the overall application.

Does the Path to Microservices Begin with a Monolith?

When it comes to a microservices architecture, there are basically two schools of thought regarding using a monolith-first strategy. Martin Fowler, the software developer and author whose publications helped increase the popularity of the term microservices, makes a case that it is better to start with a monolithic architecture and then move to microservices if necessary. There are many examples of applications that started out with a monolithic

architecture but later moved to microservices; Netflix is one of the most well-known examples of this.

The other school of thought is that it is better to start with a microservices architecture at the very beginning of the application development process. Stefan Tilkov, co-founder and principal consultant at innoQ, is convinced that "starting with a monolith is usually exactly the wrong thing to do." He argues that if an application starts out with a monolithic architecture, its components will eventually become tightly coupled to each other. This makes it extremely difficult to split up that monolith into separate, individual services later on down the road.

The Right Architecture is Key to a Successful Application

While Netflix and many other tech companies have made the switch from a monolith to microservices, there are a few tech companies still holding on to their monoliths. Both Facebook and Etsy Track this API still have applications that are arguably running on a monolithic architecture, and successfully so, proving that there are some cases where a very large application or set of applications do not necessarily have to move from a monolith to microservices.

Choosing the architecture that is best suited to run your application is very important. The choice of architecture depends on the application, in many cases a monolith may be the right choice. Both Fowler and Tilkov both agree that it is best to keep an application simple enough so that microservices are not needed, although in many cases this is not possible. Distributed applications are inherently complex, therefore, choosing a

microservices architecture over monolithic and other types of application architectures should only be done if the situation warrants it.