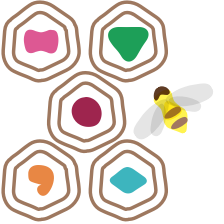
**Microservices**

**a definition of this new architectural term**

*The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.*

"Microservices" - yet another new term on the crowded streets of software architecture. Although our natural inclination is to pass such things by with a contemptuous glance, this bit of terminology describes a style of software systems that we are finding more and more appealing. We've seen many projects use this style in the last few years, and results so far have been positive, so much so that for many of our colleagues this is becoming the default style for building enterprise applications. Sadly, however, there's not much information that outlines what the microservice style is and how to do it.

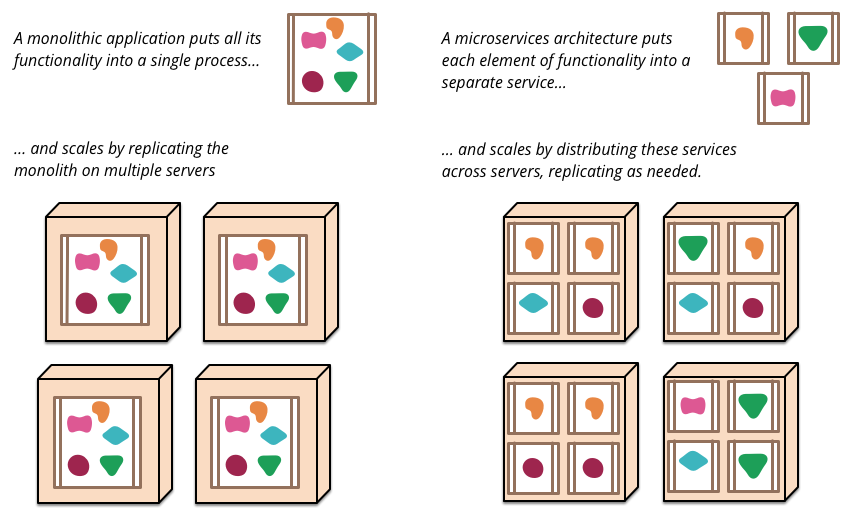
In short, the microservice architectural style [[1]](https://martinfowler.com/articles/microservices.html#footnote-etymology) is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource 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.

**[](https://martinfowler.com/microservices)**

To start explaining the microservice style it's useful to compare it to the monolithic style: a monolithic application built as a single unit. Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and javascript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a *monolith* - a single logical executable[[2]](https://martinfowler.com/articles/microservices.html" \l "footnote-monolith). Any changes to the system involve building and deploying a new version of the server-side application.

Such a monolithic server is a natural way to approach building such a system. All your logic for handling a request runs in a single process, allowing you to use the basic features of your language to divide up the application into classes, functions, and namespaces. With some care, you can run and test the application on a developer's laptop, and use a deployment pipeline to ensure that changes are properly tested and deployed into production. You can horizontally scale the monolith by running many instances behind a load-balancer.

Monolithic applications can be successful, but increasingly people are feeling frustrations with them - especially as more applications are being deployed to the cloud . Change cycles are tied together - a change made to a small part of the application, requires the entire monolith to be rebuilt and deployed. Over time it's often hard to keep a good modular structure, making it harder to keep changes that ought to only affect one module within that module. Scaling requires scaling of the entire application rather than parts of it that require greater resource.

**

*Figure 1: Monoliths and Microservices*