BREAK THE MONOLITH INTO MICROSERVICES LIKE A PRO: OUR GUIDE TO A (ALMOST) SEAMLESS TRANSITION

Microservices | Software as a Service (SaaS)

beSharp | 31 October 2019

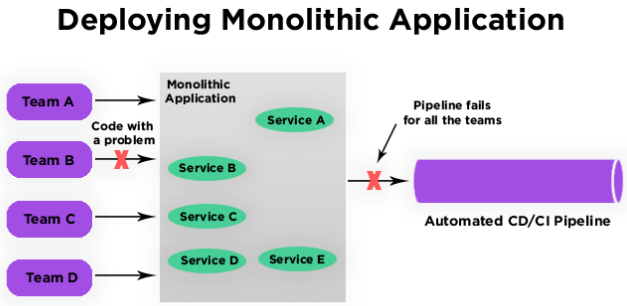
As applications grow in time, adding new features, with complex logic, many different interactions and coupling between components, it becomes very difficult to deal with them and so companies are driven to try and break them into small, manageable and reusable parts, built around **business capabilities**, microservices.

Within a microservices architecture, each application component runs as its own service and communicates with other services via a well-defined API respecting the **Single Responsibility Principle**.

Migrating to an **ecosystem of microservices** is not a simple journey but is a worthy one; embarking on it means giving your application the ability to **grow at a faster pace**, **reducing the risks of code changes** and also the **high costs** related to it.

Imagine that more of a team work on a monolithic application, mistakes on a logic which may be sticky can potentially block other team's work as usually the CD/CI pipeline is the same for all the teams; with microservices, every team can work and maintain its own codebase.

This approach is also heavily used in the SaaS paradigm, as **Software as a Service** delivery model is based on applications being centrally hosted on Cloud and being accessed via a Thin Client or a Web Browser, thus a microservices ecosystem serves this purpose very well.



You give your teams the ability to **grow in number** and to **split and focus on each separate feature in parallel**, making them more efficient; also the **cost to introduce a new member to the team in terms of code knowledge is significantly reduced**.

The costs of Experimentations are also greatly reduced, as every feature is **atomic** thus **changes cannot damage other parts of the application**. This allows the teams to generate **business value** faster.

Each microservice can be written **using different frameworks** and **programming languages**, and you can **deploy them independently**; thanks to their atomic nature and the fact that they adhere to the **isolation principle**, microservices guaranteed more easy and fast **disaster recovery solutions**.

LET'S START THE JOURNEY

As the argument is very complex and vast, we'll try to focus on general well-known techniques and tips and not on anything specifically related to programming languages or infrastructure.

Let's start by saying that "micro" in microservices is just a **label** "not" a **description**, in the sense that you don't need to try to split your application in tons of really small (and maybe useless) services on the cloud, but grasp the idea that you "really need" to find and split up **well-known part of your code that does very specific things**, with a **high value** and with a **domain** that is **clear** and **understandable**.

PREREQUISITES

Even if working with microservices can be extremely appealing for the developers, your company must have some degree of readiness to start taking on the microservices' journey.

As a matter of fact the developers must have a clear understanding, and ideally some working experience, of technologies like Docker Containers, Kubernetes or maybe AWS Lambda for hosting

your microservices.

You must be able to provide an operational service in less time as possible so you must be able to define **pipelines** for **Continuous Deployment** and **Continuous Integration**.

Finally, you must be able to provide **fitting monitoring tools** for rapidly inspecting your microservices-ecosystem (imaging in AWS an intensive use of tools such as X-Ray, CloudWatch, CloudTrail, and ElasticSearch with Kibana).

These prerequisites require your company to follow the **DevOps Culture**, in which every team member must have a clear understanding of both **coding tasks** and **operational environment**.

SPLIT YOUR APPLICATION TIERS

This is by far the first step and probably the easiest one in terms of understanding how it can be achieved. When you start to break a monolith an easy win is to find areas that can be naturally decoupled like for example **front-end** and **back-end,** because ideally front-end must be only a consumer of API; a simple example is given by a Rails Application: the HTML pages are usually rendered from **.erb templates** filled with data **served by the Rails back-end** and the **content is highly coupled** with it. Creating a Single Page Application with Angular for example and make it communicate with a back-end **through APIs** naturally decouple the two application's tiers. This process is of course long in terms of development time but is very important so always strive to achieve it, and can be, by a certain degree, parallelized.

For now, we can end here our analysis as we have seen pros of a microservice approach and how your team and in general your company must be prepared to start this process. Stay tuned for part 2 in which we will start describing step by step strategies to complete the migration from Monolith to a Microservices-ecosystem.

See you, guys!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it