A lot of efforts and money can be saved with the help of proper IT architecture. In many cases, it generally determines whether your digital project will survive or not. What architecture will be the right choice for your company project? Let's figure it out.

Monolithic Architecture vs. Microservices: What's What

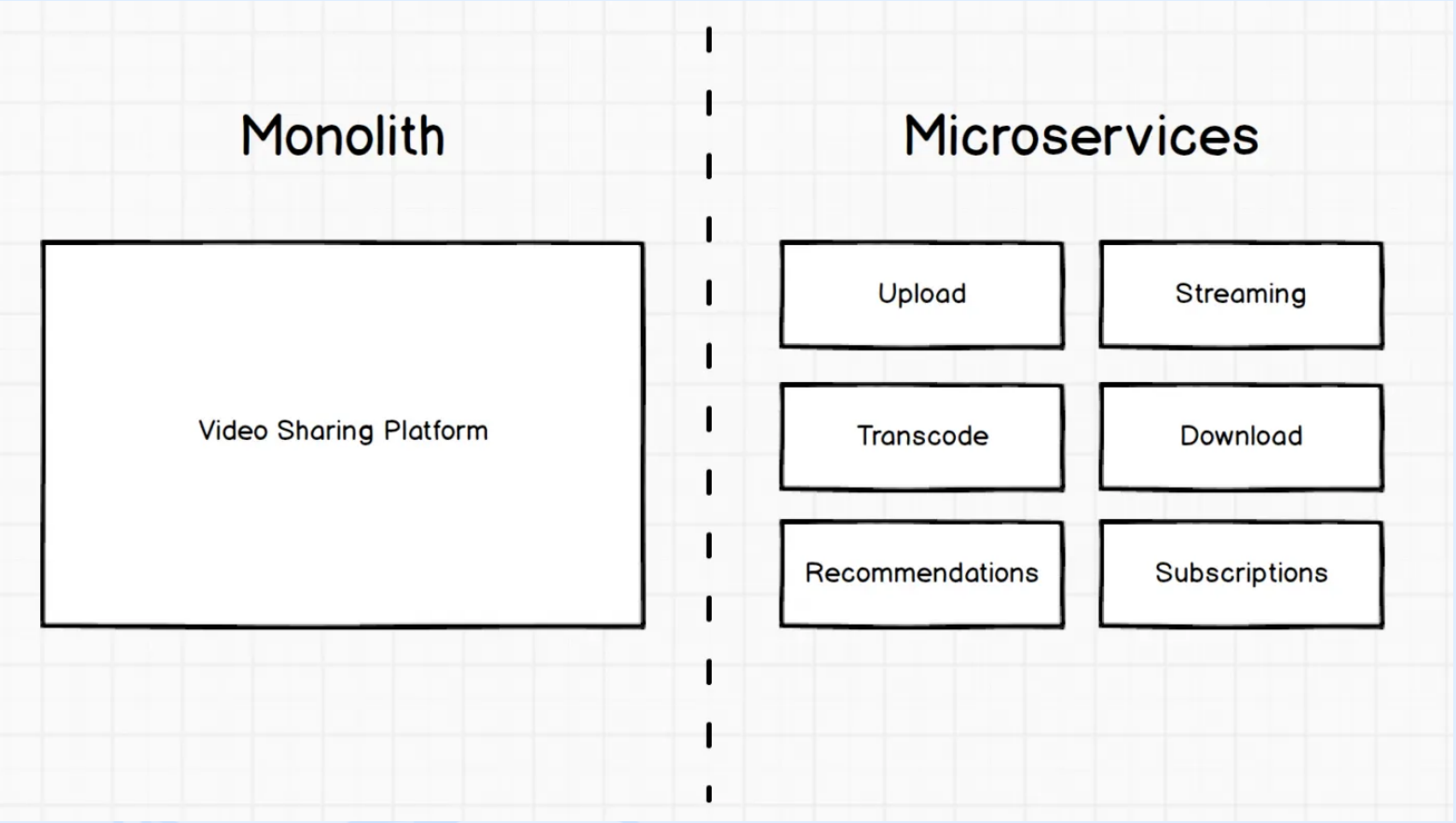
A **monolithic architecture** is a software development program designed to be self-contained and to work independently from any other software applications. So, a monolith programmed as a complete system in order to execute the whole cycle of operation to realise a particular business task.

A **microservice architecture, or just microservices,** is a software system whose structure is divided into separate blocks, so the architecture of the software consists of a number of independent modules. Such a structure helps to develop scalable software solutions, providing the transparency of separate functions and overall project flexibility. This setup is widely used in service-oriented architecture (SOA).

Let's make it simple. Here is an example of the potential implementation of a hypothetical video sharing platform: first in a monolithic representation (one large block) and then in a microservice one.

+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk



The difference is that the first is a single large block, i.e. monolith. The second is a set of small functional services. Each service has its own specific role.

Monolith

Software with a monolithic architecture comprises applications with a multi-layered structure. A classic example is the use of three-layer architecture, where one level is responsible for interaction with the user, the second for business logic processing and the third for communication with the server, providing access to data.

1. **User Interface.** This is the presentation layer with which the user interacts. It includes user interface components such as CSS styles, static html pages and JavaScript code. The main function of the presentation layer is to display information and interpret user input commands, transforming them into appropriate operations in the context of business logic.
2. **Business logic layer.** This comprises the set of components responsible for processing data received from the presentation layer. It directly interacts with the data access layer and can be implemented using Java EE and ASP.NET technologies.
3. **Data access layer.** The third layer stores data models used by entities within the business logic of the app. It is responsible for monitoring transactions and maintaining a consistent data state.

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

For most corporate software applications, most of the logic of the data access layer is concentrated in a DBMS (Database Management System) such as Oracle, MySQL or PostgreSQL.

Here's an example of a simple web app code with a monolithic architecture, located on the server, which displays a particular page depending on the query that comes to the server.

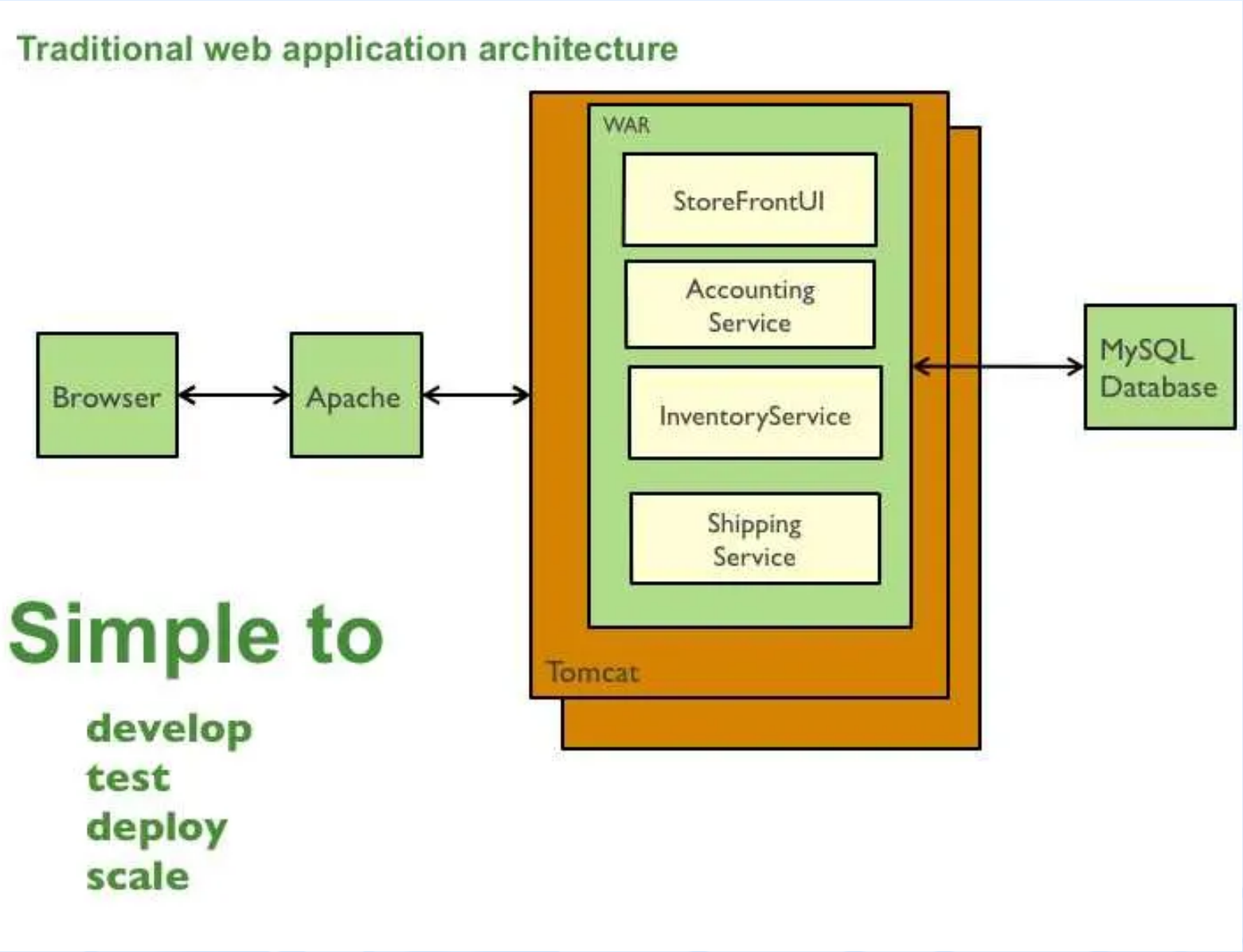
```php
require_once("./utils/get_server_params.php");
$uri=get_server_params("REQUEST_URI");
switch($uri){
case"/items":
require_once("items_page.php");
break;
default:
require_once("welcome_page.php");
};
?>
```

This application is written in PHP. The entry point to the app is the index.php file, which processes all incoming requests. When you log into a web application, PHP includes the get_server_params method from the file of the same name, which is located in the utils folder, and later uses it to obtain the value contained in the "REQUEST_URI" property that came with the request to the server.

In fact, applications, even in the case of a monolithic architecture, will appear more structured and coherent, and will also have a more elaborate system of routes – in this example, the usual **switch-case** is responsible for this.

Advantages of monolithic architecture:

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

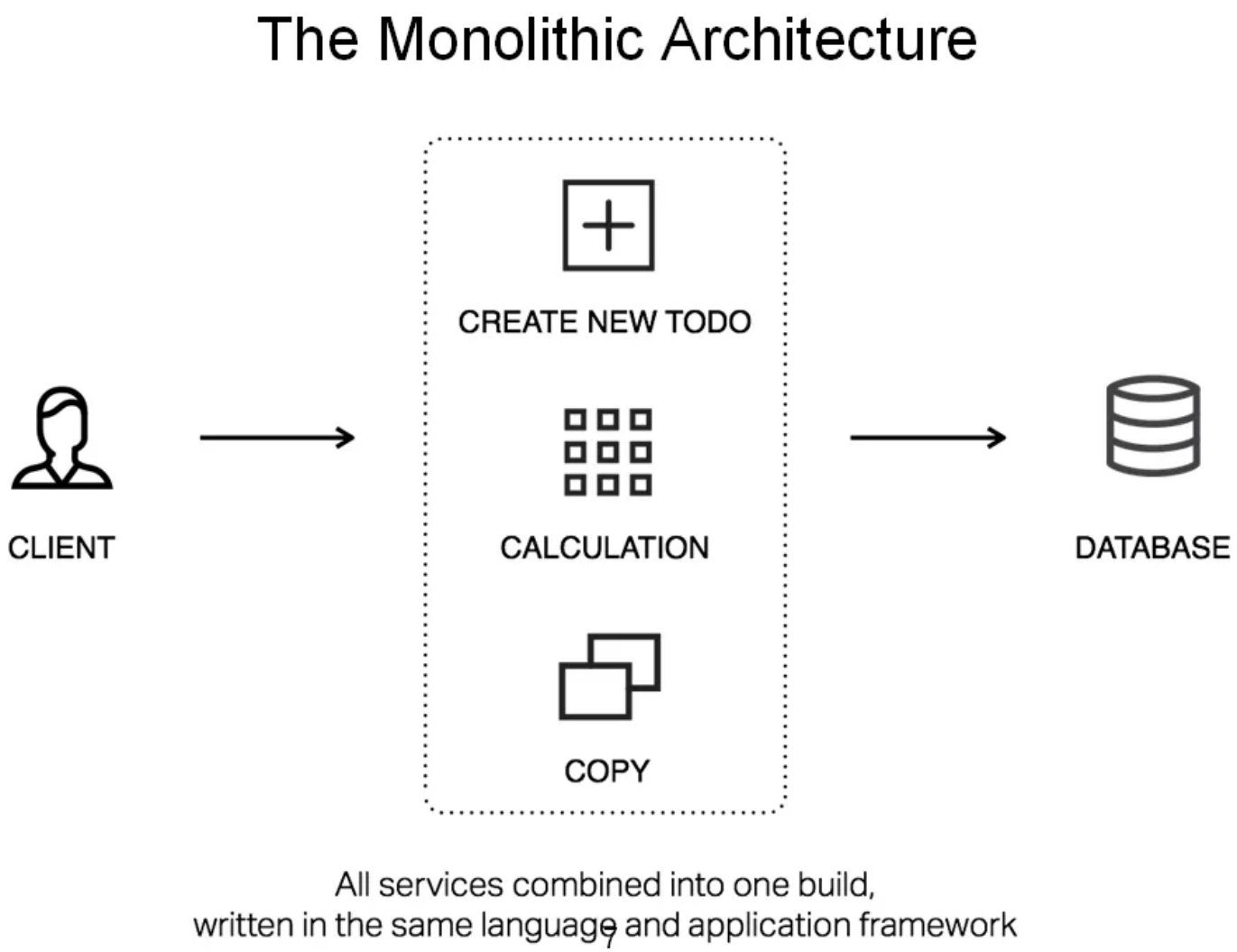


- **Simplicity.** Monolithic architecture is easier to implement, deploy and control.
- **Consistency.** With a monolithic architecture, it is easier to maintain code consistency, handle errors and so on.
- **Inter-module refactoring.** A single architecture makes it easier to work in situations where several modules must interact with each other or when we want to move classes from one module to another.

Meanwhile, corporate solutions are rapidly evolving, becoming fragmented. They can provide some functionality and can be used as a part of another app using web services based on REST, SOAP and XML-RPC protocols. This leads to applications with a monolithic architecture becoming more vulnerable in terms of security due to a growing number of different systems that should communicate and get

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

access to these layers. Difficulties with the implementation of asynchronous communication between apps thus emerge and the need arises across the whole system for complex mechanisms for managing transactions between logically separate apps and the monolith levels.



Main disadvantages of monolithic architecture:

- It is difficult or almost impossible to change the technological stack of a web application after development. -> Limitations in providing the new functionality can lead to the creation of a new system from scratch, demanding global investment from the business.
- The need for a complete system upgrade when minor details of the app are changed; -> Complicated programming and a long testing period for each system alteration lead to an

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

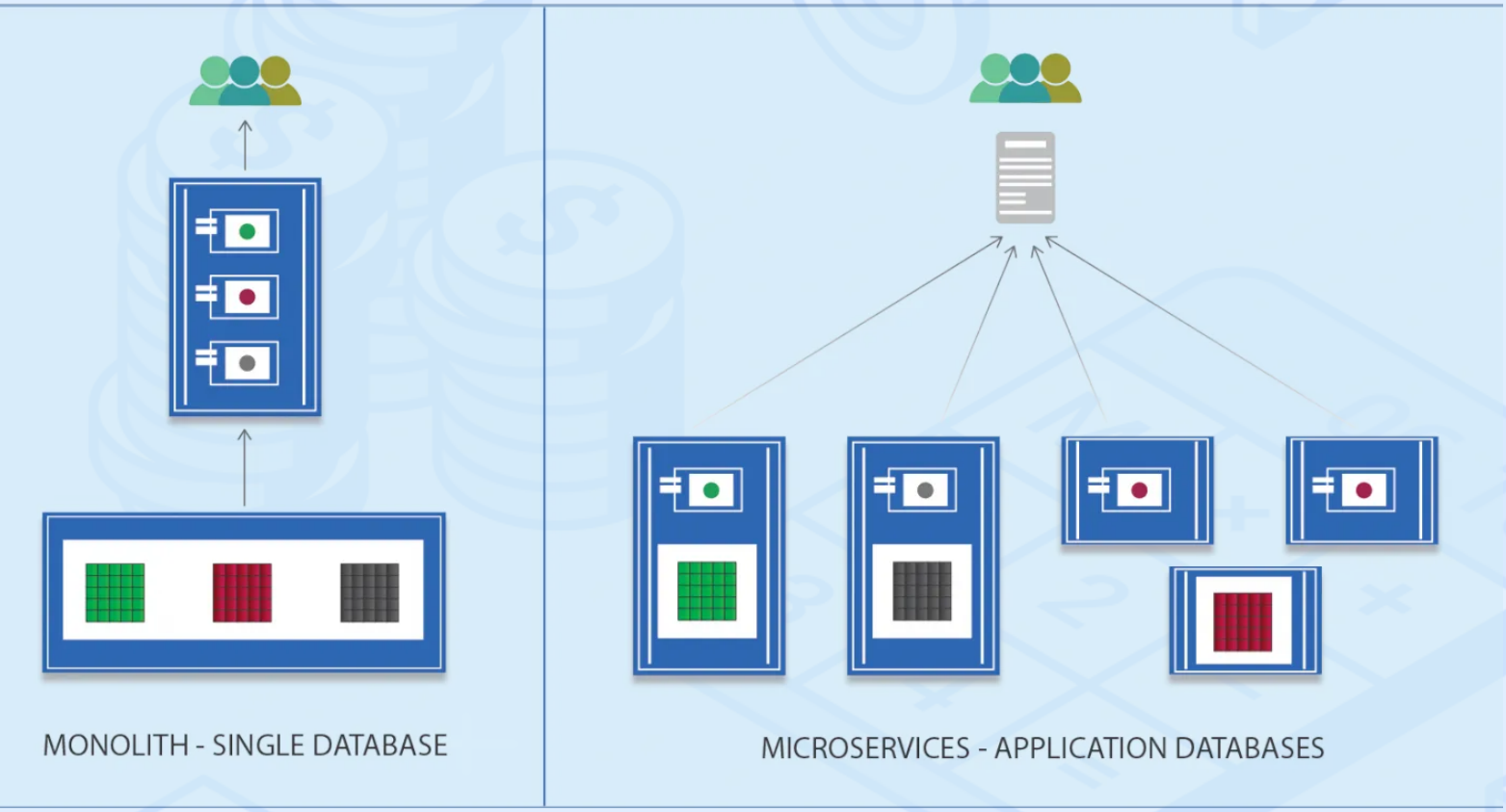
increase in the number of possible errors for the user and a more expensive redevelopment process for the owners.
- If the app crashes, then all functionality is rendered unavailable to users;-> the impact on the whole business is immeasurable.
- Complexity in scaling -> Lack of flexibility for the growing company.

In general, the monolithic architecture of software solutions doesn't allow for rapid response to changes in the market and the new business needs.

Microservices

The microservice architecture was born from the limitations that emerged within monolithic systems. The main difference between microservice and monolithic architectures is the use of specialised blocks (modules) for executing the separate functions of an app. Microservices are autonomous.

What is microservice architecture?



Microservice is an architectural template whereby your app consists of many small services that interact with each other by messaging. These can be requests from Cloud Haskell, Erlang, Akka, or REST API, Thrift, Protobuf, MessagePack and so on. All the services that conform to this template are:

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

- **Small.** The service should not require many people to develop. One team can develop several services.
- **Focused.** One service equals one task.
- **Weakly-bound.** Changes to one service do not affect the other.
- **Highly coordinated.** A component or class is created taking into account all methods of solving a business problem.

A classic monolithic application usually has a standard Interface structure -> Business logic -> Data.

- Microservices are based on the following roots:

One service must solve one task and these tasks are determined by the relevant, responsible parts of the app. The main advantage of this architecture is the quick response to data requests outside the block, be it the internal service or external agent.

For example, there may be different services for building an internet ecommerce system with integrated logistics services. In this case it is worth isolating the internet-orders, delivery and warehouse inventory functions into separate microservices. Working smoothly, each module will exchange necessary information with the others, decreasing the pressure on the whole system and providing results quickly and responsively, provided the communication is well-organised.

- Microservices must communicate separately:

Devising the optimal communication schema is a difficult task. You have to organise the data exchange for each of the modules separately, providing authorised access to internal and external visitors at the same time.

Watch this video with the IT expert Josh Evans for a clear explanation of the above-mentioned approach to microservices.

If you're interested in the technical part of the data exchange, here is the relevant discussion of the programming intricacies.

of the data exchange, here is the relevant discussion of the programming intricacies.

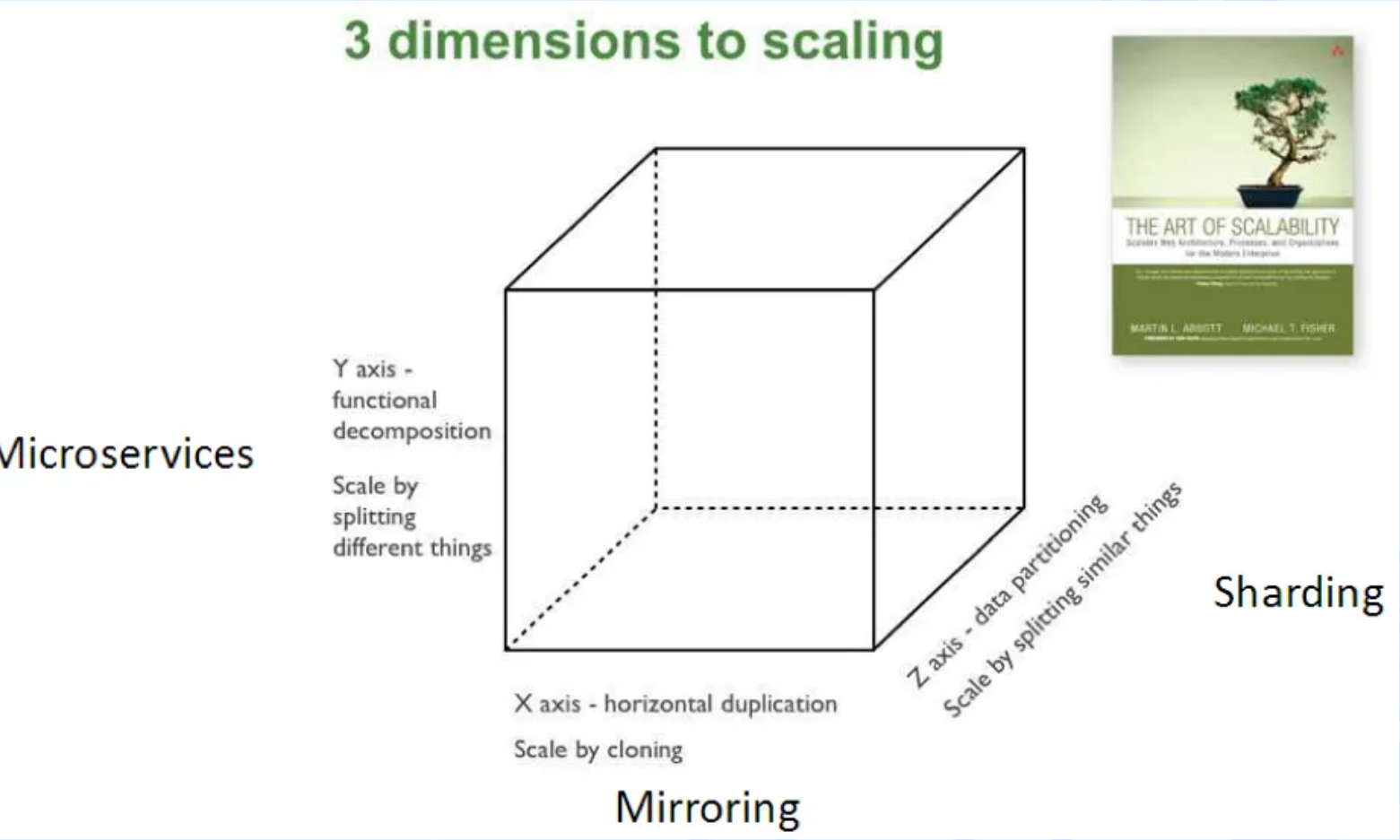
How microservices work – a practical example:

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

Amazon is a classic example of the implementation of a microservice architecture system. If you're switching from the products to the recommendation block and it isn't working, you'll see "recommendations are not available", but all the other features are working well. Nothing fatal has happened and the transactions are functioning on the usual basis. If it was a monolith, however, you'd get no answer to your request at all.

When applied

Usually, microservice architecture is used as one of the options for scaling an app. There are three such options:



1. **Microservices** – the functionality is broken down into business tasks and each service can be created with its own development tools.
2. **Sharding** (also called "splitting" or simply "shading") – the data and tools for access to them are placed on different nodes.
3. **Mirroring** – duplication of all data on a set of identical nodes.

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

Despite the overall simplicity of the program structure with microservices, building the composite service for the data transfer within all the separated blocks is becoming a non-trivial task. If you need more technical details, read this article about the technologies for creating microservices.

The microservice approach to software design is not the best choice and not the worst, but simply one of the options available. To decide whether to build an app from a set of unrelated parts, you need to weight the pros and cons of this approach.

Benefits:

- **Flexibility**: changes can be easily applied to any part of the system (module) and independence from other modules guarantees bug-free operation and a quick and simple testing process.
- **Clear division by modules.** Transparent module structure, simple code review and decompilation of the tasks among the internal development team or external vendors. Simple integration of new features and changes to the existing functional blocks.
- **High availability.** If any part of the monolith breaks, the whole app will break. With microservices, it's different: some services (except for critical modules, such as authorisation) may not work, but the application will remain available for users.
- **Various technologies.** When developing each service, you are free to choose the tools and the developers that are best suited to the specific task. Microservice architecture also allows you to try a new technology on a separate service, without overwriting the entire system. This flexibility offers the chance to divide system development among several separated teams, speeding up progress and diminishing the risks of fatal errors.
- **Relative simplicity of deployment.** Each service is raised independently, which makes the deployment and debugging process simpler, the structure - transparent and the compilation - quicker.

Challenges:

- **Implementation:** there are more units, which means more complex scripts, configuration files and transfer areas are needed to monitor implementation.
- **Performance:** Microservices are more likely to face the need of communication through the network, while the monolith can benefit from local calls.
- **Management:** The workload of management operations increases as there are more log files, runtime components and point-to-point interactions to monitor.
- **Testability:** integration tests are more difficult to configure and execute because they can span microservices in different runtime environments.
- **Runtime autonomy:** The business logic is distributed via microservices. Therefore, under other identical conditions, microservices are more likely to interact with other services through the network; this interaction reduces autonomy. We can use techniques such as event-based architecture, final consistency, caching (data replication), CQRS and microservice alignment with contexts delimited by DDD to avoid runtime autonomy.

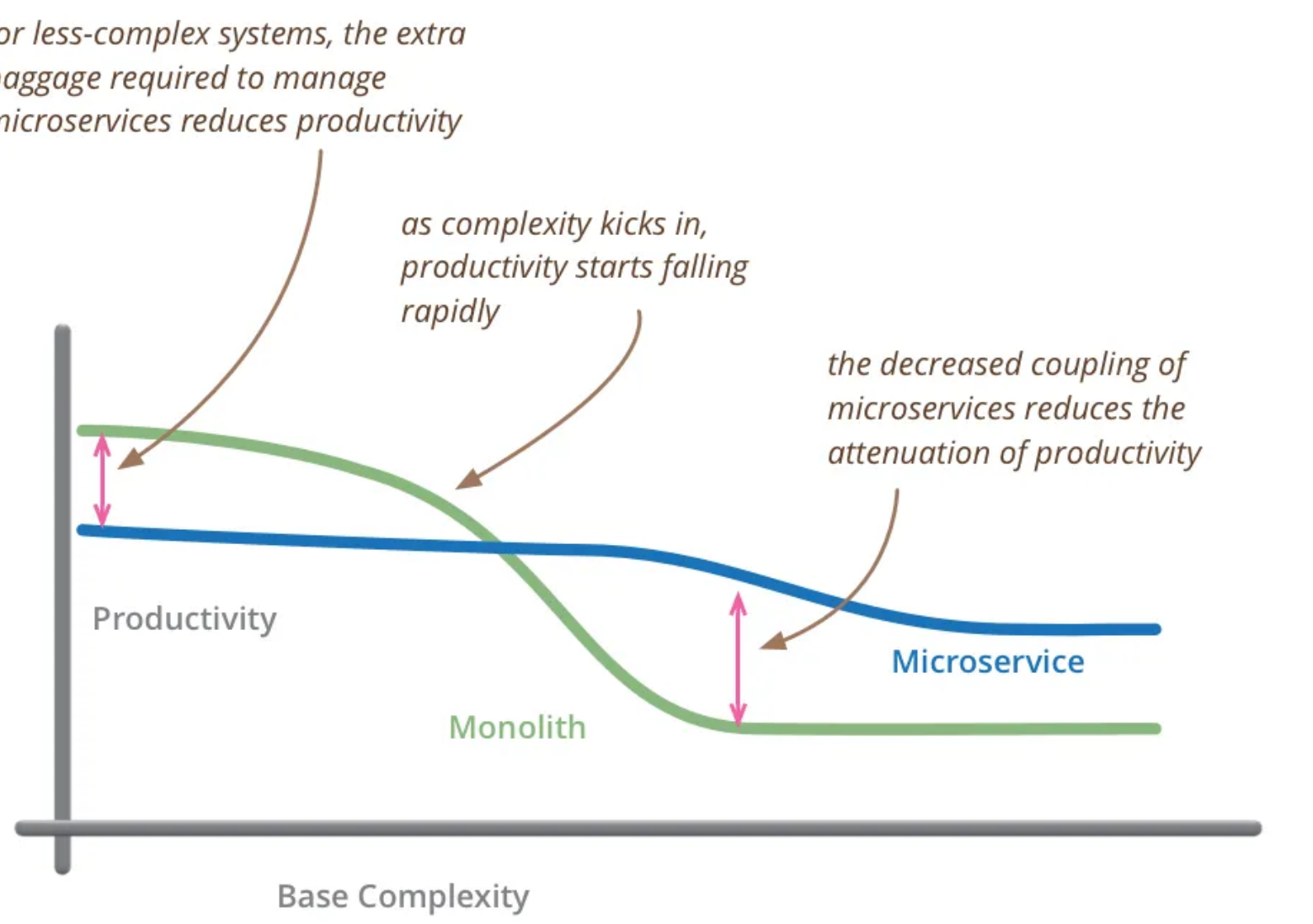
+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

- **Memory usage:** multiple classes and libraries are generally replicated in each package and the overall memory footprint is increased.

To find out more about the potential challenges of microservice logic development and see some concrete examples realised in real cases, read this article.

Monolith or Microservices: Which to Choose?

Monolith vs. Microservices is a complex choice between two options. Both are fuzzy definitions, and many systems are in a diffuse boundary area. There are also other systems that don't fit into either of these categories. But here are some tips to help you make up your mind.



+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

"When you use microservices you have to work on automated deployment, monitoring, dealing with failure, eventual consistency, and other factors that a distributed system introduces." Resource: https://martinfowler.com/bliki/MicroservicePremium.html

You may start with a monolith if:

- ***You're building an MVP or trying to proof your idea.*** In this case your project is likely to evolve over time, but for now, when you need to get to market as soon as possible, a monolith is ideal to start off with.
- ***You're developing a small app with simple functionality.*** You may update it in the future, but it's not going to grow into something more than what it is. So, a monolith can be the right fit.

You may start with microservices if:

- ***You need independent service delivery.*** If your main goal is to have individual parts within a larger, integrated system, microservice architecture is the right way to go.
- ***You plan to grow your software.*** Starting with microservice architecture, you get a lot of space for further development. You get high scalability of the system and don't have to worry about the team, as new parts of the system can be written in other languages or with the use of other tools.

Typical Problems

When you choose microservice architecture, you will increase decoupling and separation of interests – because you are actually splitting your app. This makes your code base easier to manage (each application is kept separate). So if you do it right, it will be easier to add new features to your app in the future. If you use a monolith and your app is large, this can become very difficult (and you can assume this will happen at some point).

Deploying applications is also easier because you can create separate services and deploy them to different servers. This means that you can develop and deploy them at any time without having to re-create the rest of the software. This is particularly useful with service apps that can be extended separately from the rest of the system.

However, for a software system that will not be managed much in the future it's best to keep it in a monolithic architecture. There are some serious difficulties with the microservice architecture. Using microservices increases the complexity of distribution of services to different servers in different locations, and you need to find a way to manage all of them.

Looking for a compromise? Have your heard about the Semi-monolith?

Instead of creating a very complicated structure, there is a way to divide this complete system into 2-3 parts. This is semi-monolith architecture. It could be a good alternative... though in reality, such a structure can be quite harmful: decoupling works leads to all the difficulties associated with communication between separate blocks of program, which must be worked out for any microformat architecture and have the same flexibility issues and limitations as the classical monolith. So, it's the

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

way to even more complications…

If your product is oriented towards integration with external services (like the solutions for Amazon, eBay, PayPal, Twitter and other tech stars, building a microservice will be a beneficial choice in the long run, but for smaller apps, it's much easier to stay in a monolith.

Are You Ready to Make the Choice?

We've tried here to share with you all the pros and cons of each of the software architecture approaches. Each advantage and disadvantage should be assessed from the point of view of business values and future perspectives.

Sometimes an advantage becomes a disadvantage and vice versa (for example, hard module boundaries are good in complex systems, but an excessive load for simple ones).

The decisions you make depend on applying these criteria to your situation, assessing the critical factors for your system and how they affect the results. In most cases, it is very difficult to find the right solution without an expert's advice. If this is what you need, just contact us.

+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk