Microservices vs Monolithic Architecture

Microservices are an important software trend and one that can have profound implications not just on the enterprise IT function, but the digital transformation of the entire business. Microservices vs monolithic architecture represents a fundamental shift in how IT approaches software development, and one which has been successfully adopted by organizations like Netflix, Google, Amazon, and others. But what are microservices (https://www.mulesoft.com/resources/api/what-are-microservices)' advantages over a monolithic architecture?

Microservices architecture vs monolithic architecture

First, let's compare microservices vs monolithic architecture. A monolithic application is built as a single unit. Enterprise Applications are built in three parts: a database (consisting of many tables usually in a relational database management system), a client-side user interface (consisting of HTML pages and/or JavaScript running in a browser), and a server-side application. This server-side application will handle HTTP requests, execute some domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser. It is a monolith – a single logical executable. To make any alterations to the system, a developer must build and deploy an updated version of the server-side application.

By contrast, microservice capabilities are expressed formally with business-oriented APIs (https://www.mulesoft.com/platform/api). They encapsulate a core business capability, and as such are valuable assets to the business. The implementation of the service, which may involve integrations with systems of record, is completely hidden as the interface is defined purely in business terms. The positioning of services as valuable assets to the business implicitly promotes them as adaptable for use in multiple contexts. The same service can be reused in more than one business process or over different business channels or digital touchpoints, depending on need. Dependencies between services and their consumer are minimized by applying the principle of loose coupling. By standardizing on contracts expressed through business-oriented APIs, consumers are not impacted by changes in the implementation of the service. This allows service owners to change the implementation and modify the systems of record or service compositions which may lie behind the interface and replace them without any downstream impact.

Traditional software development processes (waterfall, agile, etc) usual relatively large teams working on a single, monolithic deployment artifact. Project managers, developers, and operational staff can reach varying degrees of success with these models, releasing application candidates that can be verified by the business, particularly as they gain experience using a particular software and deployment stack. There are, however, some lurking issues traditional approaches:

- Monolithic applications can evolve into a “big ball of mud”; a situation where no single developer (or group of developers) understands the entirety of the application.
- Limited reuse is realized across monolithic applications.
- Scaling monolithic applications (https://www.mulesoft.com/resources/api/microservices-security) can often be a challenge.
- It’s difficult to achieve operational agility in the repeated deployment of monolithic application artifacts.
- By definition monolithic applications are implemented using a single development stack (ie, JEE or .NET), which can limit the availability of “the right tool for the job.”

A microservice architecture, in concert with cloud deployment technologies, API management (https://www.mulesoft.com/platform/api/manager), and integration technologies, provides a different approach to software development. The monolith is instead disassembled into a set of independent services that are developed, deployed and maintained separately. This has the following advantages:

- Services are encouraged to be small, ideally built by a handful of developers.
- If microservices’ interfaces are exposed with a standard protocol, such as a REST-ful API, they can be consumed and reused by other services and applications without direct coupling through language bindings or shared libraries.
- Services exist as independent deployment artifacts and can be scaled independently of other services.
- Developing services discretely allows developers to use the appropriate development framework for the task at hand. The tradeoffs of microservices architecture vs monolithic architecture

- Project teams need to easily discover services as potential reuse candidates. Those services should provide documentation, test consoles, etc so re-using them is easier than building from scratch.

- Interdependencies between services need to be closely monitored. Downtime of services, service outages, service upgrades, etc can all have cascading downstream effects and such impact should be proactively analyzed.

It's important to make sure that your microservices delivery is carefully managed and that the SDLC is automated as much as possible. A lack of DevOps-style team coordination and automation will mean that your microservices initiative will bring more pain than benefits.

Benefits of microservices vs monolithic architecture

The business benefits of a microservices vs a monolithic architecture are significant. When deployed properly, a microservices based architecture can bring significant value to the business. That value can be expressed in both technical debt being avoided and a substantial increase in efficiency.

For example, technical debt from a monolithic code base is a measurable reality in traditional DevOps. With monolithic code, even segregated components share the same memory, as well as sharing access to the program itself. While that may make it a little easier to code interfaces and implement applications, it ultimately takes away the flexibility that should be a part the agile development process.

What's more, a monolithic code base introduces an exponential level of inefficiency that increases technical debt. For example, chores such as bug resolution, interface modifications, adding capabilities, and other changes to applications, impacts the application as a whole, introducing downtime, as well as creating an environment where inefficiencies can unintentionally be introduced. Simply put, monolithic code bases are more time consuming to work with, are less adaptable and ultimately, more expensive to maintain, which in turn increase technical debt.

A microservices-based architecture eschews many of monolithic architecture's problems that can create technical debt, and in turn, brings measurable cost savings in both time and speed to market.

Reducing technical debt is an important benefit offered by microservices; however, measurable savings do not end with just technical debt. Microservices offers other benefits to the business which can reduce costs and impact the bottom line. Those benefits include:

application. This removes the painful process of integration normally associated with monolithic applications. Microservices speed development, turning this process that can be accomplished in weeks and not months.

- Efficiency: Leveraging a microservices based architecture can result in a far more efficient use of code and underlying infrastructure. It is not uncommon to experience significant cost savings by as much as 50% by reducing the amount of infrastructure required to run a given application.

- Resiliency: By dispersing functionality across multiple services eliminates an applications susceptibility to a single point of failure. Resulting in applications which can perform better, experience less downtime and can scale on demand.

- Revenue: Faster iterations and decreased downtime can increase revenue (either using efficiencies created via a chargeback ideology or by improving user engagement). User retention and engagement increases with continuous improvements offered by microservices.

Organizations looking to maximize productivity, embrace agility and improve customer experience should look beyond yesterday's monolithic Web applications and embrace microservices, whose loosely-coupled architectures speed development, testing, and deployment, accommodating today's and tomorrow's digital requirements.

Take a look at more resources on best practices for implementing microservices (https://www.mulesoft.com/lp/whitepaper/api/microservices-best-practices) in your organization.

Related Articles

- Microservices and DevOps: Better together (/resources/api/microservices-devops-better-together)
- Microservices and Security: Increasing security by increasing surface area (/resources/api/microservices-security)