American Journal of Computer Science and Engineering
2016; 3(3): 20-24
http://www.openscienceonline.com/journal/ajcse

Microservices: A Flexible Architecture for the Digital Age Version 1.0

Keshab Katuwal[1, 2]

[1]Software Architecture and Development, Syntel Inc., Troy, Michigan

[2]Graduated in Master of Science in Computer Science from Maharishi University of Management, Fairfield, IA, USA

Email address

keshab_katuwal@syntelinc.com, keshab.katuwal@gmail.com

To cite this article

Keshab Katuwal. Microservices: A Flexible Architecture for the Digital Age Version 1.0. *American Journal of Computer Science and Engineering.* Vol. 3, No. 3, 2016, pp. 20-24.

Received: August 20, 2016; **Accepted**: August 29, 2016; **Published**: September 2, 2016

Abstract

In today's always-on world, it is no longer feasible to release software products on a multi-month or multi-year development cycle. Traditional "monolithic" applications have inherent risks and limitations that cannot always meet the demands of the Digital Age. In this paper, we explore microservices, a new approach that may be better suited to developing applications for today's fast-moving business climate.

Keywords

Microservices, Microservice Architecture, Distributed Software Architecture

1. Introduction

Software applications are now being developed such that each autonomous component of the application (such as a web services component, messaging component, etc.) can be deployed independently in different deployment systems. There has been a gradual decrease in the size of monolithic applications by isolating these functional components into different deployment machines and allowing them to communicate to the main application and with each other using loose coupling interfaces — through either a synchronous HTTP channel or an asynchronous messaging channel. This style of software development is in itself a move towards microservice architecture.

While applying the Agile and Scrum software development processes, a software application is decomposed into multiple incremental components which are designed and developed, and then unit and functional testing are carried out before deployment. Later, these components are integrated during the integration deployment and testing phases. This approach to software development leads to the microservice architectural style of development.

James Lewis and Martin Fowler describe the microservice architectural style as:

> "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." [1]

Microservice architecture breaks down a complex application into several autonomous services which are small, independent processes that communicate with each other using language-independent APIs. These services are highly decoupled and focus on doing a small task, which enables a modular approach to system building. Services like this are easily replaceable, designed to fulfill specific business needs, and can be implemented using different technologies. In addition, because microservices are autonomous, they are independently built and deployed using a continuous delivery software development methodology.

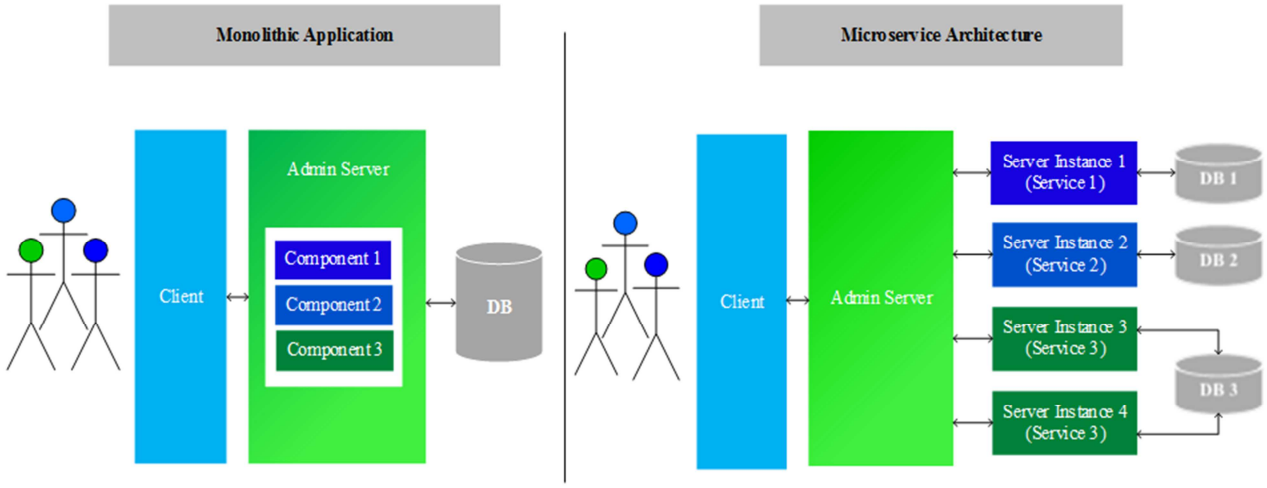

Figure 1. *Architectural overview of monolith and microservices.*

Microservices can manage their own database, and each microservice can be scaled independently from both application and database points, irrespective of other microservices. This allows an enterprise to scale only those specific domains of the enterprise application which are resource consuming.

The functioning principles of microservices resemble the UNIX philosophy of "*Do one thing and do it well.*"

Each microservice is small, fine-grained, and intended to perform a single function. Microservices can call other microservices. They are elastic, resilient, composable, minimal and complete. Microservices can be exposed through a simple URI interface. They receive a request and produce a response in UNIX style [2].

Microservice architecture differs from a service-oriented architecture (SOA) in that the primary aim of SOA is to integrate various business applications, whereas each service in a microservice architecture is a part of the whole, which represents an application.

Microservice architecture, therefore, solves the problems faced by a monolithic approach to software development.

2. Issues with Monolithic Applications

A monolithic application is an enterprise application consisting of a presentation layer, a data-access layer, and a server-side layer. Due to this rigid, tightly-coupled architecture, monoliths have a number of inherent risks and limitations, which we will outline below.

2.1. Scaling

A monolith can scale with increasing transaction volumes by running more copies of the same application. However, this architecture can't scale with increasing data volume. Each copy of the application instance will access all of the data, which makes caching less effective and increases memory consumption and I/O traffic.

2.2. Resilient Challenges (React to Failure)

Because the monolithic design is tightly coupled, failure in one part of the application brings the whole application down. It is difficult to isolate and pin-point the domain or the component of the application that caused the failure. It is, therefore, almost impossible for the monolithic application to self-recover from a failure. Because of this, a monolithic design responds slowly and its failure recovery is also slow in most cases. By the time the application restores to its original working state, a significant amount of time is invested.

2.3. Technology and Vendor Lock

Monolithic architecture binds you to stick to the technology stack and in some cases, to a particular version of the technology chosen at the start of development. For example, components written in non-JVM languages do not have a place in the monolithic application. If there is a need to migrate an application from an obsolete technology to a newer version or a better technology, the entire application must be rewritten, which is risky and time consuming.

2.4. Issues Implementing Agile Methodology

Because the Agile software development process requires frequent integration of software components, each time a new component or a new change is integrated to the application, the entire application will have to be redeployed. This will interrupt background tasks (e.g. background jobs) and some components may not initialize or update, which causes a failure during the application startup process. Therefore, the cost of frequent redeployment of the production system is very high.

2.5. Change Management or Business Agility

Change cycles are tied together, so changes made to a small part of the application require the entire monolith to be rebuilt and deployed. Over time, it's often hard to maintain a

good modular structure, making it harder to restrict changes to affect only one module.

Monolithic software development requires all the components of the application to be integrated in order to carry out development, testing, staging, and production deployment. After production deployment, a considerable amount of maintenance will be required, which translates to frequent redeployment of the entire application. This style of software development causes poor business agility.

3. Features and Benefits of the Microservice Architecture

3.1. Functional Decomposition Via Service

Microservice architecture decomposes software into a set of functions, each of which carries a specific business capability. Each of these functions represents a service and is designed and built as a component that is independently deployable.

A monolithic application encapsulates a component in the form of a library that is linked into a program and called using in-memory function calls, while microservice architecture represents a component-as-a-service. As such, these services are out-of-process components which communicate via mechanisms like web service requests or remote procedure calls.

These services take a broad-stack implementation of software for that business area, including user-interface, persistent storage, and external collaboration. Consequently, development teams must be cross-functional, including the full range of skills required: user-experience, database and project management.

3.2. Design for Failure

Microservices are elastic, resilient, and responsive — designed to react to failure quickly. A consequence of using services as components is that applications must be built so that they can tolerate the failure of one or more services. Any service call could fail due to unavailability of the supplier, so clients must respond as gracefully as possible. This is a disadvantage compared to a monolithic design, as it introduces additional complexity to handle failures.

3.3. Evolutionary

In today's "always on" digital world, it is no longer feasible to release products on a multi-month or multi-year development cycle. One key benefit of microservice architecture is that you can evolve towards this approach one service at a time — by identifying a business capability, implementing it as a microservice, and integrating using a loose coupling pattern, with the existing monolith acting as a bridge to the new architecture.

Over time, as more and more business capabilities are migrated, the scope of the monolith will shrink, until it has been completely migrated to the new application and the old application is no longer needed.

3.4. Compatibility with Agile Methodology

A microservice architecture approach requires organizing cross-functional teams for the purpose of owning the microservices product lifecycle from design through deployment. It is therefore, better suited for adopting Agile principles and the Scrum process. Microservices are built and deployed independently during each step of software development, an approach that favors frequent deployment, integration and software delivery.

3.5. Service Versioning

Since there is complete control over the deployment for microservices, it becomes possible to have multiple versions of services running side by side, providing backward compatibility and easy integration. Moreover, a new version of the microservices API can be released without impacting clients that are using previous versions. It is also possible to provide ongoing updates and enhancements to existing services while in production.

Using this approach, services can be fully replaced while maintaining the current API, or new implementations can be released under a new version.

3.6. Service Monitoring

Microservice architecture emphasizes choreography and event collaboration. This, in turn, leads to emergent behavior which enables quick detection of bad behavior. Since services can fail at any time, it's important to be able to detect failures quickly and, if possible, automatically restore service.

Microservice applications emphasize real-time application monitoring, checking both architectural elements (database requests per second, etc.) and business metrics (orders per minute, etc.). Semantic monitoring can provide an early warning system that triggers alerts to prompt development teams to follow up and investigate.

3.7. Automation Using Continuous Delivery

Software is developed in a high-velocity, iterative approach and is deployable throughout its lifecycle. Because each service carries defined business capabilities that represent a part of the product and are independently deployable, multiple-stage automation tests can be run around each service during development. These include unit and functional testing, acceptance testing, integration testing, UAT, performance testing, and deploy to production.

4. Microservices Trends

After the evolution of World Wide Web (WWW), the dynamic web applications were built in two-tier architecture in the beginning. The two-tier architecture comes under the client-server architecture. The two tier architecture

comprised of the client tier (client application) and database placing in the separate systems. The presentation logic, the business logic, and the database access logic were put together to build the client application, which was deployed in a separate server system. In term of desktop application, this client application was installed at the end users systems. Such client application was also known as the thick client. This client application directly communicated to the database server in order to fulfill the end users request. In the two-tier architecture, the client application, because of its tight coupling nature, could not be scaled to the extent in order to overcome the performance bottleneck caused by increasing user requests.

The three-tier architecture was later evolved which overcame most of drawbacks introduced by the two-tier architecture. The three-tier architecture also falls under client-server software architecture pattern. The three logical layers which were mixed together and tightly coupled in the client application of the two-tier architecture are separated into the three independent layers encapsulated by the granular interfaces and could be deployed into the three separate systems. The three-tier architecture therefore consists of presentation tier, application tier (business logic, logic tier, or middle tier) and data tier. The application tier also serves as the application controller. The term tier refers to the physical separation of components into different deployment systems whereas layer refers to both logical and physical separation of components. In logical layering, components are separated by granular interfaces and deployed in same deployment system. Each tier of the three-tier architecture could be deployed into separate systems and scaled independently to overcome the performance bottleneck caused by increasing user requests. Nowadays most of the dynamic web applications are built using the three-tier architecture.

N-tier architecture is the three-tier architecture with multi-tier middle tier. In order to meet todays diversified business needs, enterprise application may need to communicate with external applications to fulfill the client request. These external systems could be the Web Service provider, the Messaging service provider, and any other information systems. In such case, the middle tier is partitioned into multiple tiers and each of such tiers is encapsulated with fine-grained interface. Nowadays, enterprise software applications (monolithic applications) are built using n-tier architecture. But development of enterprise software application could take significant efforts, time, and resources.

The evolution of microservice architecture is expected to overcome problems arise during the development of enterprise software application.

Microservice architecture is now the approach of choice for the design and development of highly scalable, highly available software applications. While designing and developing software applications, some application domains may get higher transaction load, while others get less. Domain modules with high transaction volumes and their own database management systems are designed, developed, and deployed independently, while also scaling from both application and database points. Each of the application's domain modules is built to be highly scalable and highly available, which makes the entire application highly available.

As of today, many companies have already been utilizing microservice architecture for building software applications, including Amazon, Netflix, Uber, Comcast Cable, eBay, Karma, and Capital One. [3]

5. Conclusion

Microservice architecture is a highly scalable, resilient approach to developing software products that is well suited to today's digital world. Microservices are ideally suited for Agile development, and their inherent independence, scalability and replaceability make microservices an ideal choice for building highly-available, highly-scalable software applications.

As long as development teams are assembled with the right mix of skills and proper attention is paid to designing for service failure, microservice architecture can help enterprises develop the scalability, velocity and flexibility required to compete in the Digital Age.

References

[1] Martin Fowler. (2014). Microservices Resource Guide [Online]. Available: http://martinfowler.com/microservices/

[2] Wikipedia, the free encyclopedia. Microservices [Online]. Available: https://en.wikipedia.org/wiki/Microservices

[3] Chris Richardson. Microservice architecture patterns and best practices, Microservices.io (Online). Available: http://microservices.io/

[4] James Lewis and Martin Fowler. (2014, March 25). Microservices (Online). Available: http://martinfowler.com/articles/microservices.html

[5] Bob Familiar, Microservices, IoT and Azure: Leveraging DevOps and Microservice Architecture to deliver SaaS Solutions, 1st ed. New York: Apress, 2015.

[6] Eric Knorr. (2015, Jan 5). Why 2015 will be the year of microservices (Online). Available: http://www.javaworld.com/article/2863409/soa/why-2015-will-be-the-year-of-microservices.html/

[7] Chris Richardson. (2014, May 25). Microservices: Decomposing Applications for Deployability and Scalability (Online). Available: http://www.infoq.com/articles/microservices-intro

[8] Tori Wieldt. (2015, December 9). Bert Ertman on the Microservices Mindset (Online). Available: https://blog.newrelic.com/2015/12/09/microservices-bert-ertman-java-one/

[9] Vinh D. Le et al., "Microservice-based Architecture for the NRDC," in Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on, 2015©IEEE. doi: 10.1109/INDIN.2015.7281983

[10] Wikipedia, the free encyclopedia. Multitier architecture [Online]. Available: https://en.wikipedia.org/wiki/Multitier_architecture

[11] Dong Guo et al., "Microservices Architecture Based Cloudware Deployment Platform for Service Computing," in Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on, 2016©IEEE. doi: 10.1109/SOSE.2016.22

[12] Armin Balalaie et al., "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," IEEE Xpl., vol.33, no.3, pp.42-52, May-Jun., 2016.

[13] Martin Fowler, Patterns of Enterprise Application Architecture, 1st ed.: Addison-Wesley, 2002.

Biography

Keshab Katuwal

Keshab Katuwal is a Project Lead with Syntel Inc. He has extensive experience in architecture, design, and development of enterprise software applications. He graduated in Master of Science in Computer Science from Maharishi University of Management, Fairfield, IA, USA. He has a strong interest in the field of distributed software architecture.