University of Patras - Polytechnic School
Department of Electrical Engineering
and Computer Technology

Department: Electronics and Computers Department
Laboratory: NaN

Thesis

of the student of Department of Electrical Engineering
and Computer Technology of the Polytechnic School of the University of Patras

Anastasios Zampetis of Michael

record number: 228322

Subject

Data Search & Extraction with Microservices

Supervisor

Professor Dr. Evangelos Dermatas, University of Patras

Thesis Number:

Patras, February 2024

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα

Αναζήτηση & Εξόρυξη Δεδομένων με Microservices

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Αναστασίου Ζαμπέτη του Μιχαήλ

(Α.Μ.: 228322)

παρουσιάστηκε δημόσια και εξετάστηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

__/__/____

Ο Επιβλέπων

Ο Διευθυντής του Τομέα

ΔΡ. Ευάγγελος Δερματάς
Καθηγητής

Ο Συν-Επιβλέπων

NaN
NaN

NaN
NaN

CERTIFICATION

It is certified that the Thesis with Subject

Data Search & Extraction with Microservices

of the student of the Department of Electrical Engineering & Computer Technology

Anastasios Zampetis of Michael

(R.N: 228322)

Was presented publicly and defended at the Department of Electrical Engineering & Computer Technology at

__/__/____

The Supervisor

Dr. Evangelos Dermatas
Professor

The Director of the Division

The Co-Supervisor

NaN
NaN

Thesis details

Subject: **Data Search & Extraction with Microservices**

Student: **Anastasios Zampetis of Michael**

Supervising Team
Professor Dr. Evangelos Dermatas

NaN NaN

Laboratories:
NaN

Thesis research period:
September 2018 - September 2019

This thesis was written in LaTeX.

Abstract

Internet of Things (IoT) is an enabling technology for numerous domains worldwide, such as smart cities, manufacturing, logistics and critical infrastructure. On top of Internet of Things, the architectural paradigm shifts from cloud-centric to Edge-centric, offloading more and more functionality from the cloud to the Edge devices. Edge computing devices are transformed from simply aggregating data to performing data processing and decision making, accelerating the decentralization of the IoT domain. Given the considerable increase of the number of IoT devices and the size of the generated data, an increase of Edge devices with augmented functionality is expected. We propose an ”Edge as a service” scheme, where Edge devices will be able to procure unused resources and run services that are requested and consumed by IoT devices that belong to different stakeholders. In this work, we outline a high-level architecture of this scheme and give a reference implementation of a narrow part of the system, boasting high modularity and the extensive use of Open Source Technologies.

Acknowledgements

I would like to thank, first and foremost, Christos Tranoris, Researcher at the NAM group, who guided me at every step of the thesis, put up with incoherent messages and ideas during the most improper hours. I have the privilege to call him a mentor, a colleague a and a friend. I would also like to thank Professor Spiros Denazis for introducing me to the NAM group and for his insightful comments on this work. Most importantly, I want to thank John Gialelis who introduced me to the world of Academic Research and who has been my mentor for over 2 years. Finally, I want to thank the whole team of NAM group for the fruitful conversations, as also the online communities and developers of the projects: *EdgeX Foundry*, *Filecoin*, *loraserver.io*, *Node-RED* and *balena*, for their continuing assistance and guidance throughout the implementation. I dedicate this work to my Parents, *Despoina* and *George*, who laid the foundations for everything I have accomplished thus far, and to my friends and my companion, who have supported me, put up with me and pushed me to become a better human being.

Contents

List of Figures

List of Tables

1. Design concepts

1.1 Monolithic Architecture

Before going into Microservices and the Microservice architecture, the Monolithic architecture approach must be explained first. The Monolithic architecture approach was till recently the preferred design option for software. In a Monolithic application all different components and functions of the business logic are combined into one indivisible program [1]. Generally these components are the user interface, business rules and data access. While individual components might be developed separately, they remain tightly coupled [2] and any change completed in any of them requires the whole program to be rebuild and redeployed [3]. More often than not development in one component requires functional changes in multiple other, adding on the development cost, complicating the build and testing process and inducing delays in deployment. A single bug in any one component can potentially halt the entire application's operation and create a nightmarish situation for on-call engineers trying to figure out the root cause and usually resulting in multiple unrelated to the issue teams joining in till root cause analysis is complete. Additionally Monolithic applications usually have large codebases, which can be cumbersome when implementing changes and difficult to manage over time [2]. Another major issue with Monolithic applications is scalability. Usually different components have conflicting resource requirements but because of the unified design all requirements must be handled together making scaling up the application impossible vertically, only allowing horizontal scaling through multiple copies. Scaling horizontally is very resource consuming and restricted. Finally Monolithic design allows for little to no flexibility for incorporating newer, state of the art technologies, slowly resulting in legacy applications that have to be completely redesigned and reconstructed when performance degrades. Despite the many drawbacks of Monolithic architecture, it is still favored for certain applications because of some core benefits. The most important one is performance. In most cases Monolithic applications outperform their modular counterparts [2]. Also initial design and implementation are easier since individual components are usually clearly defined at later stages. Additionally a single codebase and unified build and deployment process simplifies configuration management, testing and monitoring [2]. It is clear that the Monolithic architecture approach works well for smaller applications and helps to get things up and running faster. Furthermore when development complexity and deployment time come second to performance a Monolithic application usually has the edge over a modular approach.

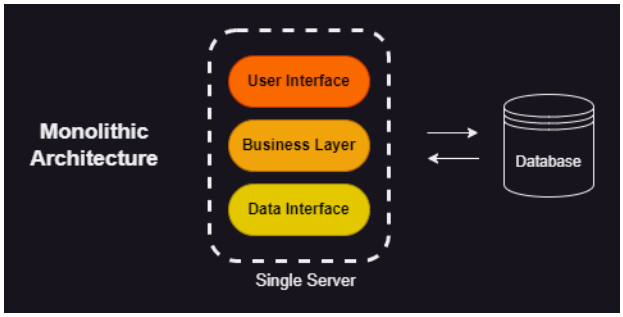

Figure 1.1: Monolithic Architecture

1.2 Microservice Architecture

Microservices and the Microservice Architecture are, the last few years, one of the most popular design option for software applications. In the Microservice Architecture the application is structured as a collection of independent services, called Microservices. Each Microservice corresponds to a different part of the business logic, executing a well defined unique process [1] [4]. These Microservices utilize lightweight communication mechanisms, such as API interfaces, so that they can operate in unison and achieve the same final results as a Monolithic application but without being co-depended. Microservices can be build and tested separately and be deployed and scaled independently. It's Microservice should only facilitate a single function of the application and be easily comprehensible. [5]. It is not hard to understand the advantages such an architecture brings compared to it's Monolithic counterpart. Each individual Microservice can be developed separately, by different teams, without compromising or delaying the development of other parts of the application. That way, different components of the application can be improved and updated asynchronously resulting in quicker deployments of the final application and the build and deployment process is straightforward and cheaper resource wise. Also, testing is incomparably easier and faster since we don't have to build and test the whole application but only a singular Microservice. Same to testing, debugging, can be quickly and effortlessly delegated to the responsible team because it is simple to pinpoint the Microservice that failed. While all these definitely play a major factor on why the Microservice Architecture is continuously gaining in popularity, the most important advantage of it is scalability. In the era of cloud native applications where being able to scale up and down on demand is of outmost importance and where costs are usage derived, Microservice based applications significantly outclass Monolithic ones. On a Microservice application scaling is possible both vertically and horizontally. If needed the whole application can be copied just like a Monolithic one but also a single Microservice can be upscaled if demand is high. More over since Microservice applications can be readily instantiated there's no need for binding resources, opposite to Monolithic application where usually at least a couple of instances need to be always up and running to cover demand spikes resulting in exponentially increased operational costs. All these advantages, thought, cannot be achieved without drawbacks. Initial development of Microservice applications requires careful and time consuming planning and design as well as a certain level of expertise since requirements and features are not yet well defined at this stage. Also Microservice applications, more often than not, lag behind Monolithic ones in performance and therefore are not suitable for time critical operations such as load balancers. Finally, Microservice architecture is not the best option for on-prem applications where customers have to setup everything manually. [6]

1.3 Containers

Hand in hand with the Microservice Architecture came containers. Containers are a form of virtualization similar to virtual machines (VMs), but unlike traditional virtual machines, containers share the host system's kernel while running in isolated user spaces. This architecture makes them significantly more lightweight, efficient, and versatile compared to VMs. The primary advantage of containers lies in their efficient utilization of host hardware resources and their rapid, straightforward deployment, which is ideal for scalable environments. In comparison, Vms require a separate guest operating system, adding a large overhead both resource and time wise. Consequently, compared to VMs, containers require significantly less memory and processing power, allowing for more performance out of the same hardware infrastructure. [7] Containers are spun up from images, which are built using COntainerFiles. A ContainerFile specifies a base image and a series of steps to execute on top of it, creating what are known as layers. Each step in the build process forms a new layer, which is a critical feature for development and deployment workflows, because layers are reusable, meaning that if multiple container images share common steps, those layers need to be built only once. This reuse

of layers drastically reduces build time and storage space, making containerization highly compatible with agile development processes where frequent builds and deployments are common. Container images can easily be distributed using container image registries, like Docker Hub which is the registry of the most popular container platform, Docker. Containers provide a consistent runtime environment, ensuring smooth transitions between development, staging and production environments, streamlining testing and deployment process and thus minimizing release time. Efficient use of containers, especially for application deployments, require some form of orchestration. Container orchestration is usually handled through orchestration platforms, such as Kubernetes, which automates deployment, scaling and management of containerized applications ensuring high availability, resilience and incidence recovery. [8]

1.4 IoT device Simulators

There's little doubt that the Internet of Things is here to stay. IoT reshaped the way humans and machines interact with the environment and is now strongly influencing most industries. In recent years more and more everyday devices are coming equipped with all sorts of sensors and internet connectivity abilities, generating big amounts of data. Along with IoT enabled devices and sensors, there's a lot of development done on IoT applications in order to make use of all the generated data and better utilize the devices. Same as all other applications, IoT apps need to be thoroughly tested before deployment to production. One way for this to be done is to create an actual IoT network of devices, specific to the application, to generate all the data and test the application. It is not hard to notice the issues with this implementation. First of all, it is very expensive to create an actual IoT network of devices, especially when there's a need to test on large ecosystems. Furthermore, it is very common for an application early in development to be redesigned and changed often and in such case IoT ecosystem used for testing will have to be redesigned and changed as well. This adds to the development cost and introduces delays to the development process. Even if an IoT network of devices is already in place for deployment reasons, testing an application using the live network can introduce security risks. Instead of using actual data from real IoT devices, synthetic data can be generated, simulating real IoT networks.

Bibliography

[1] Konrad Gos and Wojciech Zabierowski. The comparison of microservice and monolithic architecture. pages 150–153, 04 2020.

[2] Ivy Wigmore Rahul Awati. What is monolithic architecture? https://www.techtarget.com/whatis/definition/monolithic-architecture, 2022.

[3] Freddy Tapia, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. From monolithic systems to microservices: A comparative study of performance. *Applied Sciences*, 10(17), 2020.

[4] Martin Fowler James Lewis. Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html, 2014.

[5] A. Chandrinos. *Upgrading Existing Allergy Symptoms Monitoring Lab Infrastructure (AllergyMap) to Containerized Environment*. Master's thesis, Department of Computer Engineering and Informatics, University of Patras, 2021.

[6] Dan Ackerson Tomas Fernandez. When microservices are a bad idea. https://semaphoreci.com/blog/bad-microservices, 2022.

[7] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, May 2015.

[8] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), mar 2014.

University of Patras, Polytechnic School
Department of Electrical Engineering & Computer Technology
Anastasios Zampetis of Michael