MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Inginerie Software și Automatică

Admis la susținere
Șef departament: Fiodorov I. dr., conf.univ.

„___” _____ 2022

Instrument automatizat pentru furnizarea centrului și infrastructurii pentru crearea și rularea microserviciilor

Teză de master

Student: _____ **Șcebec Mihai, gr. IS-211M**
Coordonator: _____ **Braga Vasile, lect. univ.**
Consultant: _____ **Catruc Mariana, lect. univ.**

Chișinău, 2023

ADNOTARE

Numele și prenumele autorului: Șcebec Mihai

Titlul tezei: Instrument automatizat pentru furnizarea centrului și infrastructurii pentru crearea și rularea microserviciilor

Cuvinte cheie: Microservice Framework, paralelizare, Îmbunătățirea calității vieții, automatizare, modularitatea.

Structura: Primul capitol descrie problema afacerii, soluția aleasă și cerințele acesteia. Capitolul 2 prezintă structura soluției prin diferite diagrame. Al treilea capitol explică procesul de dezvoltare, instrumentele utilizate și deciziile luate.

Rezumat: Această lucrare își propune să rezolve problema cadrelor actuale de microservicii: le lipsește suportul unei configurații clare și concise, nu sunt scalabile în mod autonom din punct de vedere al performanței. Soluțiile existente susțin slab dezvoltarea proiectelor de dimensiuni medii. Unele dintre ele oferă prea puține funcții, altele au configurație și umflare inutilă a fișierelor care vor pierde prea mult timp pentru un proiect de dimensiune medie. Se petrece prea mult timp de fiecare dată, de exemplu pentru a migra versiunile bazei de date, pentru a configura mai multe fișiere de configurare sau chiar pentru a reține toate opțiunile necesare pentru o comandă de consolă. De asemenea, devine din ce în ce mai dificil cu cât dezvoltatorul are nevoie de mai multe servicii simultan, ceea ce duce la necesitatea de a lansa manual mai multe servicii, precum și de a oferi un mediu pentru acestea, instanțe de baze de date, conectând totul împreună, gestionând porturile, ținând evidența mediului. module.

Pentru a rezolva acest lucru, autorul propune să utilizeze abordarea unui sistem modular cu suport ridicat pentru fluxuri de lucru complexe automate și paralelizare încorporată pentru performanță, precum și suport pentru cele mai frecvent utilizate funcții din cutie, cu cât mai puțină muncă manuală. Aceasta înseamnă că va exista o listă de funcții prioritare, cele mai frecvent utilizate, pe care această soluție le va oferi la sfârșitul lucrării. Acestea vor fi enumerate în subcapitolul corespunzător. Cele mai notabile dintre ele sunt caracteristicile de revigorare a serviciilor eșuate, gestionarea automată a distribuției de porturi, activitățile de pregătire inițială a lansării și realizarea automată a celor mai comune lucruri, cum ar fi furnizarea de jurnale, baze de date, swagger, hub-uri de management, control al modulelor și bibliotecii.

ANNOTATION

Last name and first name of the author: Scebec Mihai

Title of thesis: Automated framework for providing hub and infrastructure for creating and running microservices

Keywords: Microservice Framework, parallelisation, Quality of Life, automation, modularity.

Structure: First chapter describes the business problem, chosen solution and its requirements. Chapter 2 shows the solution structure via various diagrams. Third chapter explains the development process, used tools and decisions taken.

Summary: This work aims to solve the problem of current microservice frameworks: they lack support of clear and concise configuration, they are not autonomously scalable from a performance standpoint. Existing solutions poorly support mid-size project development. Some of them provide too few features, the others have configuration and unnecessary file bloat that will waste too much time for a mid-size project. Too much time is spent every time for instance to migrate database versions, to setup multiple config files or even to remember all necessary options for a console command. It is also becoming more and more challenging the more services at once the developer needs, resulting in a need to manually launch multiple services as well as providing an environment for them, database instances, connecting everything together, managing the ports, keeping track of environment modules.

To solve this the author proposes to use the approach of a modular system with high support of automated complex workflows and built-in parallelisation for performance, as well as support of most commonly used features out of the box with as little manual work as possible. This means that there will be a list of priority, most commonly used features that this solution will provide at the end of the work. They will be listed in the corresponding subchapter. Most notable of them are features to revive failed services, automatically handle port distribution, initial launch preparation activities and automatically doing most common things like providing logs, database, swagger, management hubs, module and library control.

Table of Contents

Table of Figures

Introduction

Developing a well designed working network of services is hard. Sharing necessary data between services in request becomes simply sending all or most of the data, resulting in gigantic sizes of request which by itself looks uncomfortable to further develop, debug and simply maintain. The other approach would be to send many small requests of data. Also a bad design. What about brokers then? brokers indeed are a solution. But also as previous ones this needs more implementation work and possible architecture redesign on its own. Long story short, when developing microservices programmers nowadays don't often think about scalability much, as this part of the work is janky on many programming languages so the work is done by using Kubernetes[1] or similar stuff. The complexity also comes from setting up microservices on different devices, setting up configuration, managing dependencies, all that stuff.

On paper, proper architecture, code and repository management could solve many if not all these issues, but it comes at the cost of a lot of additional time spent on meetings, plannings, retrospectives and 'quick' calls with one's solution architect. In reality this means that devops and solution architects will be so busy explaining all the work details to many working teams that in the end inevitably a lot of things will end up overlooked. This would be a lot easier for, let's say, someone developing microservices in Go language as it is meant for such purposes. It literally has built in support for concurrency and async tasks run. A pity that it is often not considered even in the top 10 of popular or most used programming languages [2] according to Stack Overflow, one of the most ever used web resources for programmers.

Nowadays the absolute majority of the internet works around webpages and servers, be it cloud based ones or hardware. And inside those servers some sort of services usually run inside Kubernetes to provide better scalability. What the end consumer gets is a number of layers that need to be configured to work together. All these layers mean high probability of config error, config incorrect use or in general management issues.

Moreover, this way of development software makes developers implement single threaded solutions of their services, which leads to possible performance losses. Why would a developer implement a multithreaded solution if Kubernetes will spawn as many pods for him as the client needs? Here what could be observed is a performance loss due to using more pods instead of more processes and time loss for setting up all the configs. And how would a developer properly test behavior of 2 instances of his microservice if it really depends on how K8s is set up after the fact?

This project's scope is to provide a solution that will soften all these sharp corners and will implement a better way of developing, configuring and running a network of microservices.

The project's goal is to implement an automated framework, hub and infrastructure for microservices and for more comfortable creating and running microservices.

Nowadays people usually just pick the most suitable framework for them and just bare with the downsides that the solution comes with. Those aren't critical to the business part, so everyone just tries to forget every hour wasted on solving those issues. Those issues are usually something a regular developer meets every time. We are talking about hours of looping through configs, stack overflow answers of how to setup X database for Y framework or how to get all the dependencies needed without conflicts. Actually, the very first steps could be so time inefficient that they end up being done poorly, which leads to problems in the future. Having the Django framework as an example: the guy that knows the framework well could not even start showing some basic initial testing stuff without 30 minutes of preconfiguring. Given how many companies use Django there should be something really good in the end product, but the development process can be difficult, time consuming and problematic.

BIBLIOGRAPHY

1. Kubernetes. *Production-Grade Container Orchestration.* Kubernetes.io. [Accessed December 22, 2022]. Available: https://kubernetes.io/

2. CARRERO, LOLA. *Most Popular Programming Languages 2022 [Ranking].* Stackscale. September 8, 2022. [Accessed December 3, 2022]. Available: https://www.stackscale.com/blog/most-popular-programming-languages/

3. Stack Overflow.. *Stack Overflow Developer Survey 2021.* Stack Overflow. May 2021. [Accessed December 22, 2022]. Available: https://insights.stackoverflow.com/survey/2021

4. Spring.io *Spring.io*. 2019. [Accessed December 22, 2022] Available: https://spring.io/

5. Docs.spring.io *Spring Data JPA - Reference Documentation.*[Accessed December 15, 2022]. Available: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

6. Django. *The Web Framework for Perfectionists with Deadlines | Django.* Djangoproject.com. 2019. [Accessed December 22, 2022] Available: https://www.djangoproject.com/

7. *9 Examples of Companies Using Django in 2022 | Trio Developers.* n.d. Www.trio.dev. [Accessed December 22, 2022] Available: https://www.trio.dev/blog/django-applications

8. *Welcome to Django Q — Django Q 0.4.6 Documentation.*Django-Q.readthedocs.io. [Accessed December 22, 2022]. Available: https://django-q.readthedocs.io/en/v0.4.6/

9. *Welcome to Flask — Flask Documentation (2.2.x).* Flask.palletsprojects.com. [Accessed December 22, 2022]. Available: https://flask.palletsprojects.com/en/2.2.x/

10. *SQLAlchemy - the Database Toolkit for Python.* 2018. Sqlalchemy.org. [Accessed December 22, 2022]. Available: https://www.sqlalchemy.org/

11. *Werkzeug — Werkzeug Documentation (2.2.x).* Werkzeug.palletsprojects.com. [Accessed December 22, 2022]. Available: https://werkzeug.palletsprojects.com/en/2.2.x/

12. *What Is the Actor Model? - Definition from Techopedia.* Techopedia.com. [Accessed December 15, 2022]. Available: https://www.techopedia.com/definition/25150/actor-model

13. Paloalto Networks.*What Is a Denial of Service Attack (DoS) ? - Palo Alto Networks.* Paloaltonetworks.com. 2019. [Accessed December 22, 2022]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos

14. MARTIN, MATTHEW. *Functional Requirements vs Non Functional Requirements: Key Differences.* Guru99.com. August 31, 2019. [Accessed December 22, 2022]. Available: https://www.guru99.com/functional-vs-non-functional-requirements.html

15. *The Actor Model in 10 Minutes*. Brianstorti.com. July 9, 2015. [Accessed August 22, 2022]https://www.brianstorti.com/the-actor-model/

16. *What Is Remote Procedure Call (RPC)? Definition from SearchAppArchitecture.* SearchAppArchitecture. [Accessed December 22, 2022]. Available: https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC

17. Python. *Welcome to Python.org.* Python.org. August 30, 2019. [Accessed December 22, 2022]. Available: https://www.python.org/about/

18. *Chapter 2: Hello World App | Django for Beginners.* Djangoforbeginners.com. [Accessed December 15, 2022]. Available: https://djangoforbeginners.com/hello-world/

19. *PyPI · the Python Package Index.* PyPI. [Accessed December 15, 2022]. Available: https://pypi.org/

20. *FastAPI.* Fastapi.tiangolo.com. [Accessed December 15, 2022]. Available: https://fastapi.tiangolo.com/

21. *The Best APIs Are Built with Swagger Tools | Swagger.* 2019. Swagger.io. 2019. [Accessed December 15, 2022]. Available: https://swagger.io/

22. *Multiprocessing vs. Threading in Python: What Every Data Scientist Needs to Know.* FloydHub Blog. September 7, 2019. [Accessed December 15, 2022]. Available: https://blog.floydhub.com/multiprocessing-vs-threading-in-python-what-every-data-scientist-needs-to-know/

23. *Асинхронный Python без головной боли (часть 1).* Хабр. [Accessed December 15, 2022]. Available: https://habr.com/ru/post/667630/

24. *Асинхронный Python без головной боли (часть 2)*. Хабр. [Accessed December 15, 2022]. Available: https://habr.com/ru/post/671798/

25. *PyPy*. PyPy. December 28, 2019. [Accessed December 22, 2022]. Available: https://www.pypy.org/