Master's Thesis Engineering Technology

2022-2023

Microservices coverage detection

Maarten Baeten

Engineering Technology Master of Electronics and ICT Engineering

Introduction

Microservices are small **reusable** and easily **scalable** components that can reside in multiple physical and virtual **locations**. Each service is responsible for its own specific task and can operate on its own database or share one with other services. This **flexibility** makes microservice architecture (MSA) more interesting than monoliths. However, this distribution of the services makes **testing** the application more difficult. This thesis investigates which **methodology** ensures that all services are being tested. Figure 1 shows the general structure of a monolith and a microservice-based application

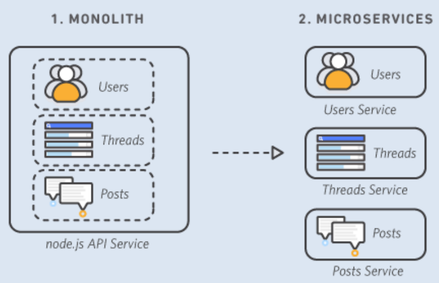

Figure 1: General structure of a monolith and microservices application [1]

Situation

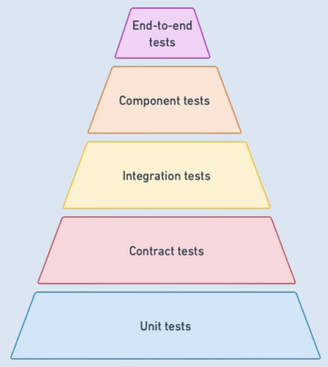

Figure 2: Testing pyramid for microservices [2]

Figure 2 shows the **testing pyramid** for microservices. Testing can be done on the different levels as shown in the pyramid. The testing pyramid shows two extra types of tests **compared to the testing pyramid for monoliths**: **component** tests and **contracts** tests. A considerable number of **code coverage** tools already exist for the unit level, but no code coverage tools exist for the **integration level**. **Therefore**, this thesis focuses on developing a **methodology** for testing the integration level.

Methods

Defining the scope comprises the **identification** of the different **microservices** and their **connection**. This should result in an **overview** of the microservices architecture. **Reviewing** the **documentation** can result in a better understanding of the architecture. Tools such as Swagger, Confluence and RAML can help with structuring the documentation.

Call-graphs help with the visual representation of the microservices architecture. Figure 3 shows an example of a call-graph. Eclipse TPTP, VisualVM and Dynatrace are some tools that can create these graphs. With a microservice-based application, services can be developed in **different languages**, but most call-graphs tools are **language specific**. However, it is possible to create **multiple** call-graphs of each part of the system that is in a different language and then **combine** them.

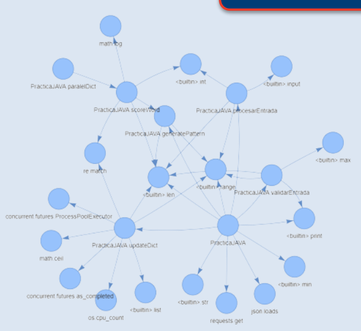

Figure 3: Call-graph example [3]

Figure 4: Directed graph

The proposed methodology applies the **Depth-First-Search (DFS)** algorithm to the **call-graphs** with the known microservices structure to initially determine (offline) the number of tests required for full coverage. The effective **test phase** then again applies DFS and tracks how many unique tests were successfully performed. The **ratio** of the DFS algorithm outcome on the call graphs and the DFS algorithm outcome on the actual microservice application is a measure of service coverage.

Figure 4 shows an example of a **directed** graph. The DFS algorithm first chooses a **random** node to start from. Then it follows this path as far as possible. Finally, the algorithm **backtracks**. For example, if the DFS starts with node A in figure 4, a possible output is: ABCEFD.

In a microservice-based system, the integration tests are used to identify defects when certain microservices **communicate** with each other. Integration tests use real services to verify the calls between the services and their cooperation and not mocked services like contract testing.

In order to have an accurate metric it is required that the test cases are comprehensive**.** The test cases should include all the different types as shown in figure 6. Then the tests can be **executed** and should contain a component that applies the **DFS** algorithm and track how many unique tests were **successfully performed**. If some tests were not successfully performed, then they should be updated before continuing with the coverage metrics, since the methodology assumes that the tests work.

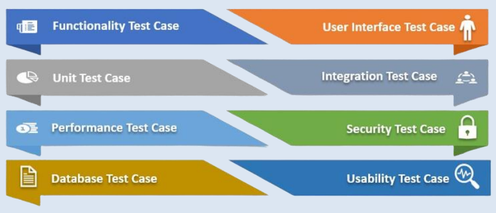

Figure 6: Most important test cases [5]

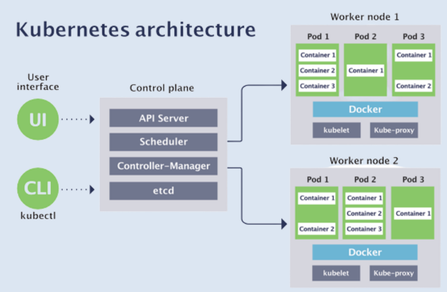

Figure 5: Kubernetes cluster structure [4]

The proposed methodology is generically applicable and provides a benchmark for checking service coverage and was empirically tested, by means of an example, in a **train-ticket** microservices application.

Kubernetes is an open-source container orchestration tool that is used to deploy the train-ticket system. A **container orchestration** tool is required to manage all the microservices. The microservices reside all in their own container, because containers allow them to be deployed **separately** and **quickly**. Figure 5 shows the structure of a Kubernetes cluster. The most important components of Kubernetes are the **control** plane and the **worker** nodes. The control plane communicates with the different worker nodes that contain several pods. The pods accommodate containers with the services.

Conclusion & Future work

This thesis contains a **comprehensive guide** to modern topics like **microservices**, **testing** of microservices and **coverage metrics**. It gives a clear high-level approach for applying coverage detection for a MSA application along with certain **tools** and tips that can help this process. The deployment of the train-ticket system encountered some unidentified server issues. Therefore, the methodology could not be validated to the full extent. However, the thesis contains the necessary knowledge about **Kubernetes** to successfully **deploy** the train-ticket system in the future.

Supervisors / Co-supervisors / Advisors:

Prof. Kris Aerts
Prof. Davide Taibi
Dr. Dario Amoroso D Aragona

[1] Anonymous, "What are Microservices?," AWS, 2023. [Online]. Available: https://aws.amazon.com/microservices/. [Accessed:4 May 2023].
[2]Fernandez and D. Ackerson, "Testing strategies for microservices," Semaphore, 20 July 2022. [Online]. Available: https://aws.amazon.com/microservices/. [Accessed:3 March 2023]
[3] D. G. Solla, "What is a Call Graph? And How to Generate them Automatically," FreeCodeCamp, 3 January 2023. [Online]. Available: https://www.freecodecamp.org/news/how-to-automate-call-graph-creation/. [Accessed 6 May 2023].
[4] J. Spaleta, "How kubernetes works." https://sensu.io/blog/how-kubernetes-works. [Accessed: 2 May 2023]
[5] T. Swati, "Types of test case." https://www.educba.com/types-of-test-case/. [Accessed: 4 May 2023]

Tampere University

De opleiding industrieel ingenieur is een gezamenlijke opleiding van UHasselt en KU Leuven

UHASSELT

KU LEUVEN