POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Design Patterns and Anti-Patterns in Microservices Architecture: A Classification Proposal and Study on Open Source Projects

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Ömer Esas

Advisor: Prof. Elisabetta Di Nitto

Academic year: 2021-2022

1. Introduction

As tech giants such as Amazon, Netflix and Uber adopted microservices architecture successfully in the past decade, there has been an increase in the microservices research in the academia and greater interest from many companies towards this method of building distributed, fault-tolerant and complex applications. More research and practical experience led to the emergence of several desirable and undesirable ways of solving problems faced in microservice-based systems, called microservices design patterns and anti-patterns. In this study, we aim to explore these patterns and anti-patterns, specifically, in terms of classification in the literature and detection of them on a group of open source microservices projects. We aim to observe the way the patterns and anti-patterns are classified in the literature, check if there exists a common way of classification and propose our own taxonomy in case there is no consensus in the literature. Then, we select ten open source microservice projects and manually inspect source code to detect the design patterns and anti-patterns of microservices architecture, in order to observe the correlation between the "theory" in the literature and practical cases to some extent.
To the best of our knowledge, there are two studies in the literature that are similar to our work. The researchers of the first similar study[1] inspect a set of thirty open source microservice projects using automated tools that check the dependency files and verify the use of the pattern by checking the documentation of the utilized framework. The researchers of the second similar study[2] manually check sixty seven projects to detect anti-patterns, which they discover as a result of their systematic literature review for microservices anti-patterns. Our study differs from the two studies in considering not only design patterns or anti-patterns, but both of these good and bad practices, in addition to using only manual inspection on projects that use many different technologies.

2. State of the Art

To be able to explain the methodology and the results of this study, we see it fit to describe microservices architecture and list its design patterns and anti-patterns, which are obtained from Microsoft Cloud patterns website[3], microservice patterns book[4] and two studies about microservices anti-patterns[2][5]. Although we present only the names of design patterns and anti-patterns here to keep it concise, they are explained in detail in the main thesis paper.

2.1. Microservices Architecture

In essence, microservice-based applications consists of a number of small software services, that are designed to carry out one business capability of the application. The system is not divided into technical layers, instead, the whole application domain is partitioned into bounded contexts, each of which is then, ideally, assigned to one small self-sufficient team to implement as a microservice. The team is responsible for the implementation, deployment and maintenance of the particular microservice, meaning that, they are free to choose their programming language, libraries and frameworks to implement, and free in their choice of CI/CD tools to utilize during various stages of software development. If required for the particular business capability, the team can persist the data, which is modeled around that bounded context, by making use of a database, again, selected by the team according to the needs and suitability of that particular microservice. Last but not least, the microservices communicate with each other by means of network calls, using well-defined APIs, and simple protocols like REST over HTTP. Unlike Service Oriented Architecture (SOA) that utilize Enterprise Service Buses (ESB) with additional capabilities, microservices opt for simple communication infrastructure that can do basic routing of messages, as summarised in the motto "simple endpoints, dumb pipes". Figure 1 illustrates the microservices architecture characteristics by comparing it with a typical monolithic architecture.

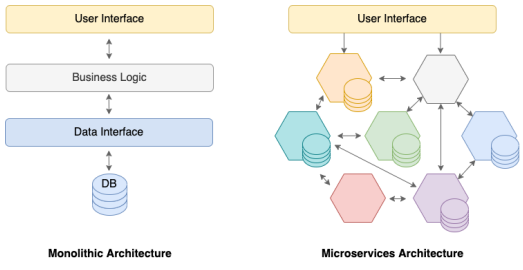

Figure 1: Monolithic vs Microservices Architecture

2.2. Design Patterns

The design patterns of microservices architecture considered in this study are listed below:

- API Gateway
- Service Mesh with Sidecar
- Service Registry and Discovery
- Backends for Frontends
- Asynchronous Messaging
- Database per Service
- Saga
- API Composition
- Command Query Responsibility Segregation (CQRS)
- Event Sourcing
- Service Instance per Virtual Machine
- Service Instance per Container
- Serverless
- Health Check API
- Log Aggregator
- Distributed Tracing
- Circuit Breaker

2.3. Anti-Patterns

Similarly, we present the list of anti-patterns considered during this study:

- Wrong Cut
- Nano Microservice
- Mega Microservice
- ESB Usage
- Hardcoded Endpoints
- No API Gateway
- Shared Persistence
- Shared Libraries
- No CI/CD Tools
- Multiple Services per Host
- No API Versioning
- No Health Check
- Local Logging

3. Research Methodology

As the goal of this study, we aimed to observe the classification regarding microservices design patterns and anti-patterns in the literature, present one in case there is no consensus, and check the presence of these patterns and anti-patterns in popular open source projects. Specifically, we established two research questions:

- **RQ1**: Is there a consistent categorization or classification of design patterns and anti-patterns of microservices architecture in the academia? If not, what could be an alternative way to classify these design patterns and anti-patterns?
- **RQ2**: Which of these design patterns and anti-patterns exist in popular open source

microservices applications?

To answer first research question, the following steps have been adopted.

- In order to find out whether there exists a consistent classification of microservice design patterns and anti-patterns, a literature review on digital libraries such as IEEE Explore, ACM Digital Library, Springer, Scopus, and Google Scholar has been carried out. The keywords used in the search queries included "microservice pattern", "microservice pattern classification", "microservice anti-pattern" and "microservice anti-pattern classification".
- After initial review of the studies found, few more studies have been added through snowballing technique, and from this extended set, the studies that do not contain a classification or grouping of patterns and anti-patterns have been eliminated.
- To be able to propose a classification of patterns and anti-patterns on a sound basis, various perspectives through which microservices architecture can be examined needed to be identified. For this reason, the systematic mapping studies found in the literature review process have been consulted.
- Considering the classifications found in the literature and the findings resulted from papers that conducted systematic mapping studies, a classification proposal has been developed.

As for the second research question, the following steps have been carried out.

- Open source repository hosting service GitHub is queried with he keyword pattern "microservice OR micro-service". To find the most popular projects, the search result is sorted using "most stars" option.
- Ten microservice repositories that have the most number of stars are selected. While doing so, the repositories that contain libraries, frameworks or tool-kits have been eliminated, in addition to the ones that contain "microservice" keyword but rather implemented using other architectures.
- Because of the technological heterogeneity among the set of projects, two design patterns, namely "saga" and "shared libraries" which require a competent understanding of the programming language being used and a thorough comprehension of the business logic are excluded from the list of patterns and anti-patterns to be identified.
- For the detection of the rest of the microservices design patterns and anti-patterns, we manually inspected the repositories and the source codes of projects, without utilizing any automated tool. We tried to find any information on the repository page that indicates the use of a particular pattern or anti-pattern, checked the source code and tried to verify the use of a pattern by consulting to the reference document of a library or framework at hand. To detect hardcoded IP addresses, we made use of the simple regular expression "\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b" and examined the context of the found hardcoded IP addresses. For patterns and anti-patterns that can be implemented to various extents, such as API composition and shared persistence, we stated the criteria that we used to indicate the presence or absence of particular pattern or anti-pattern. In addition to source codes, we examined deployment files such as "docker-compose.yaml" and dependency files such as "pom.xml" when available.

4. Results

4.1. Classification of Patterns and Anti-Patterns

As a result of the research process related to the classification of microservices patterns and anti-patterns in the literature, we found out that there are three types of classifications for design patterns and three types of categorizations for the anti-patterns, which result from a set of nine studies and one book chapter. Therefore, we concluded that there is no consensus in the literature regarding the classification of patterns and anti-patterns of microservices architecture. We then presented our taxonomy proposal by suggesting to use "architectural", "deployment" and "monitoring & reliability" categories. The categorization of the design patterns and anti-patterns into "architectural", "deployment" and "monitoring & reliability" categories are presented in Table 1, Table 2 and Table 3 respectively.

Architectural Patterns	Architectural Anti-Patterns
API Gateway	Wrong Cut
Service Mesh with Sidecar	Nano Microservice
Service Registry & Discovery	Mega Microservice
Backends for Frontends	ESB Usage
Asynchronous Messaging	Shared Libraries
Database per Service	Hardcoded Endpoints
Saga	No API Gateway
API Composition	Shared Persistence
CQRS	
Event Sourcing	

Table 1: Architectural patterns and anti-patterns of microservices architecture

Deployment Patterns	Deployment Anti-Patterns
Service Instance per Container	No CI/CD
Service Instance per VM	Multiple Service Instances per Host
Serverless	No API Versioning

Table 2: Deployment patterns and anti-patterns of microservices architecture

Monitoring & Reliability Patterns	Monitoring & Reliability Anti-Patterns
Health Check	No Health Check
Distributed Tracing	Local Logging
Log Aggregator	
Circuit Breaker	

Table 3: Monitoring & reliability patterns and anti-patterns of microservices architecture

4.2. Patterns and Anti-Patterns in Microservice Projects

To detect the patterns and anti-patterns in open source projects, we first selected ten open source microservice applications that have the most number of stars on GitHub, as shown with their names and corresponding number of stars in Table 4.

Repository Name	#Stars
dotnet-architecture/eShopOnContainers	20k
GoogleCloudPlatform/microservices-demo	11.8k
sqshq/piggymetrics	11.4k
cer/event-sourcing-examples	2.9k
microservices-patterns/FTGO-application	2.3k
vietnam-devs/coolstore-microservices	2k
Crizstian/cinema-microservice	1.6k
asc-lab/dotnetcore-microservices-poc	1.5k
elgris/microservice-app-example	1.4k
aspnetrun/run-aspnetcore-microservices	1.1k

Table 4: List of examined projects

Next, starting from the first project on the list, we examined repository page and source code to detect the aforementioned patterns and anti-patterns. To report the results, we first gave a short summary of the application domain and used technologies, indicated the presence of each pattern and anti-pattern in a table and then explained each pattern and anti-pattern by referring to a particular file in the repository when available. For exemplary purposes, the presence of design patterns and anti-patterns discovered in the "eShopOnContainers" application is indicated in Table 5 and Table 6 respectively.

Design Pattern	✓\–
API Gateway	✓
Service Mesh with Sidecar	✓
Service Registry & Discovery	✓
Backends for Frontends	✓
Asynchronous Messaging	✓
Database per Service	–
API Composition	✓
CQRS	✓
Event Sourcing	–
Service Instance per VM	–
Service Instance per Container	✓
Serverless	–
Health Check	✓
Distributed Tracing	–
Log Aggregator	✓
Circuit Breaker	–

Table 5: Presence of microservice design patterns in eShopOnContainers application

Anti-Pattern	✓\–
Wrong Cut	–
Nano Microservice	–
Mega Microservice	–
ESB Usage	–
Hardcoded Endpoints	–
No API Gateway	–
Shared Persistence	✓
No CI/CD	–
Multiple Service Instances per Host	–
No API Versioning	–
No Health Check	–
Local Logging	–

Table 6: Presence of microservice anti-patterns in eShopOnContainers application

4.3. Discussion of Findings

As a result of the detection process to find out which patterns and anti-patterns exists in prominent open source projects, we also think that it is a valuable effort to take a look at the total number of patterns and anti-patterns found in the ten projects examined.

Design Pattern	#
API Gateway	10
Service Mesh with Sidecar	3
Service Registry & Discovery	8
Backends for Frontends	1
Asynchronous Messaging	7
Database per Service	2
API Composition	2
CQRS	5
Event Sourcing	2
Service Instance per VM	–
Service Instance per Container	10
Serverless	–
Health Check	6
Distributed Tracing	5
Log Aggregator	3
Circuit Breaker	2

Table 7: Total number of design patterns in examined projects

As shown in Table 7, we see that the API gateway, service registry and discovery and asynchronous messaging are the most widely used architectural design patterns among the ten projects. Next, we observe that using containers is the preferred approach by far when compared to virtual machine images and serverless deployment when it comes to deploying a microservice application.

Anti-Pattern	#
Wrong Cut	–
Nano Microservice	–
Mega Microservice	–
ESB Usage	–
Hardcoded Endpoints	5
No API Gateway	–
Shared Persistence	6
No CI/CD	5
Multiple Service Instances per Host	–
No API Versioning	8
No Health Check	4
Local Logging	7

Table 8: Total number of anti-patterns in examined projects

As for the anti-patterns, we notice in Table 8 that the most frequent anti-pattern among the ten projects is the no API versioning anti-pattern, possibly because the examined applications are not actual microservice products that are maintained by a number of different development teams, it might be deemed not necessary by developers of examined repositories to make use of API versioning practice. Coming to the architectural anti-patterns, we observe that the design principles of microservice architectures are well digested by the practitioners. The microservices are designed around business capabilities in a balanced way and the principle of "smart endpoints, dumb pipes" is put into practice in those designs.

5. Conclusions

With this study, we investigated the literature about classifications regarding microservice patterns and anti-patterns, and observed that there are a number of different categorizations. By taking into account the way these studies categorize the patterns and anti-patterns and by constructing our own argumentation, we presented our taxonomy proposal by suggesting to utilize "architectural", "deployment" and "monitoring & reliability" categories, in order to provide a simple and valid structure in terms of classification for patterns and anti-patterns. Furthermore, we manually inspected ten open source microservice projects to see if those patterns and anti-patterns are actually present in implemented microservice architectures. By inspecting the total number of patterns and anti-patterns, we observed the imbalance in frequency of patterns and anti-patterns and discussed the probable reasons.
Regarding the possible future work, by inspecting more open source projects, the ability to generalise the result might be increased, and focusing on projects that make use of the same framework and the same technologies, a precise and thorough understanding of the implemented business logic could be achieved, enabling more observations about patterns and anti-patterns related to the logic of the application.

References

[1] Gastón Márquez and Hernán Astudillo. Actual use of architectural patterns in microservices-based open source projects. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 31–40, 2018. doi: 10.1109/APSEC.2018.00017.

[2] Rafik Tighilt, Manel Abdellatif, Naouel Moha, Hafedh Mili, Ghizlane El Boussaidi, Jean Privat, and Yann-Gaël Guéhéneuc. On the study of microservices antipatterns: A catalog proposal. In *Proceedings of the European Conference on Pattern Languages of Programs 2020*, EuroPLoP '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450377690. doi: 10.1145/3424771.3424812.

[3] Microsoft Docs. Cloud design patterns, 2022. URL `https://docs.microsoft.com/en-us/azure/architecture/patterns/`.

[4] Chris Richardson. *Microservice Patterns*. Manning Publications Co., 2019.

[5] Thomas Schirgi and Eugen Brenner. Quality assurance for microservice architectures. In *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*, pages 76–80, 2021. doi: 10.1109/ICSESS52187.2021.9522227.