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 greater interest from the academia and the industry towards the microservices architecture and its principles. The attempts to implement applications using microservices architecture led to the emergence of desirable and undesirable ways of solving common problems faced in distributed applications, namely the design patterns and anti-patterns. In this study, , we aim to observe the way the design 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

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. Each microservice is implemented and maintained by a small team who is free in choosing the kind of tech stack they use during various stages of a typical software lifecycle. If required for the particular business capability, the team can persist the data, which is modeled around that business capability, by making use of a database selected by the team according to the needs of the microservice. Last but not least, the microservices communicate with each other by means of network calls, us-

ing well-defined APIs, and simple protocols like REST over HTTP. Microservices opt for simple communication infrastructure that can do basic routing of messages, as summarised in the motto "simple endpoints, dumb pipes".

2.2. Design Patterns

The design patterns of microservices architecture considered in this study are shortly described below, which are obtained after reviewing sources such as microservices patterns book [3], Microsoft Cloud patterns website [4] and papers that contain classifications of design patterns [5][6][7][8][1][9].

- API Gateway: Using a intermediary service between frontend and microservices to route requests to specific microservices depending on the request.
- Service Mesh with Sidecar: Utilizing proxies configured via a master service to help with infrastructure-related tasks.
- Service Registry and Discovery: Having an "address-book" in terms of host and port numbers for microservices to let them locate each other and make API calls.
- Backends for Frontends: Creating multiple microservices that deal with the same business capability but return different data according to the type of the client (mobile/web/third party etc).
- Asynchronous Messaging: Utilizing a helper service to receive, temporarily store (buffer) and distribute messages between microservices.
- Database per Service: Using separate database instances to be used only by a single microservice.
- Saga: Having a coordinator service or distributed logic to implement the atomicity in distributed transactions.
- API Composition: Using multiple microservices to receive different data and combining them to serve a separate API call.
- Command Query Responsibility Segregation (CQRS): Having multiple microservice instances that solely focus on either "read" or "write" tasks of the application.
- Event Sourcing: Embracing event-driven communication, storing events in a event store to persist data in a temporal way.
- Service Instance per Virtual Machine: Creating and deploying VM images per microservice.
- Service Instance per Container: Creating and deploying Docker containers per microservice.
- Serverless: Delegating infrastructure/server management tasks to a cloud distributor, deploying the application simply by uploading the code.
- Health Check API: Implementing an additional API endpoint in microservices to check the status of instances periodically
- Log Aggregator: Receiving logs of microservices in a central service for ease of monitoring and debugging.
- Distributed Tracing: Emitting metadata from microservices about recent API calls, to be received by a central service for combining metadata to observe the "journey" of an API call as a chain of different microservices.
- Circuit Breaker: Configuring microservices to "fail-fast", i.e., return an error immediately if the requested microservice is unresponsive according to some criteria (time accumulated, number of trials, etc), and start retries in small numbers.

2.3. Anti-Patterns

Similarly, we briefly describe the anti-patterns considered during this study, which are obtained from two papers [10][2].

- Wrong Cut: Dividing application into technical layers or services, instead of business capabilities and bounded contexts.
- Nano Microservice: Designing a microservice to carry out unnecessarily small task or part of application.
- Mega Microservice: Designing a microservice to carry out unnecessarily large task or part of application.
- ESB Usage: Utilizing "smart" message brokers that help with auxiliary tasks (integration, transforming messages, etc).
- Hardcoded Endpoints: Instead of using service discovery, hardcoding the address (host and port number) of a microservice.
- No API Gateway: Directly calling backend microservices from the client without intermediary service.
- Shared Persistence: Utilization of the same

database instance by multiple microservices.
- Shared Libraries: Sharing a run-time or in-house development library instead of code duplication and independence.
- No CI/CD Tools: Not making use of CI/CD tools in shared repositories in a distributed development context.
- Multiple Services per Host: Deployment of multiple services onto the VM or physical machine.
- No API Versioning: Not having version prefixes inside the URLs of API definitions.
- No Health Check: Not checking the status of microservices, absence of health check API pattern.
- Local Logging: Not aggregating logs of microservices in a central service, absence of log aggregator pattern.

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.

Querying digital libraries 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".

Applying snowballing 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.

Consulting systematic mapping studies 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.

Developing classification proposal 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.

Querying GitHub 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.

Selecting projects 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.

Excluding saga pattern and shared libraries anti-pattern 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.

Detecting patterns and anti-patterns We manually inspected the repositories and tried to find any information on the repository page that indicates the use of a particular pattern or anti-pattern. We examined the source code, deployment files such as "docker-compose.yaml" and dependency files such as "pom.xml" when available, and then tried to verify the use of a pattern by consulting to the reference document of a library or framework at hand.

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 [5][6][7][8][1][9][11][2][12][10]. 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 as shown in Table 1, Table 2 and Table 3 respectively. While developing the proposed classification, we noticed that there are categories in the reviewed papers such as "design", "communication", "back-end", "coordination", "data" and "architectural". We argued that one pattern or anti-pattern that is placed in one of those categories in fact also affects or is affected from the ones placed in other similar categories. As an example, while the "hardcoded endpoints" anti-pattern is placed under "implementation" category in [2], the bad practice results from a lack of service discovery mechanism in the architecture of the application. We therefore combined the patterns and anti-pattern that we think are about the major design decision, i.e., architecture of the application into "architectural" category. For the remaining ones, we realised that they reflect the processes of deployment and monitoring stages of a software development lifecycle and they are about the decisions that affect the most those stages.

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

The set of examined projects is presented in Table 4, with their names and number of GitHub stars.

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.

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 5: Total number of design patterns in examined projects

As shown in Table 5, 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. As for the anti-patterns, we notice in Table 6 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.

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 6: Total number of anti-patterns in examined projects

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)*.

[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*.

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

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

[5] D Taibi, V Lenarduzzi, and Claus Pahl. Architectural patterns for microservices: A systematic mapping study. In *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science*.

[6] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, and Bedir Tekinerdoğan. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, 180, 2021.

[7] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández. Patterns related to microservice architecture: a multivocal literature review. *Programming and Computer Software*, 46, 2020.

[8] Gastón Márquez, Mónica M. Villegas, and Hernán Astudillo. A pattern language for scalable microservices-based systems. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ECSA '18, 2018.

[9] José A. Valdivia, Xavier Limón, and Karen Cortes-Verdin. Quality attributes in patterns related to microservice architecture: a systematic literature review. In *2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT)*.

[10] Thomas Schirgi and Eugen Brenner. Quality assurance for microservice architectures. In *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*.

[11] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. *Microservices Anti-patterns: A Taxonomy*. Springer International Publishing, Cham, 2020.

[12] Justus Bogner, Tobias Boceck, Matthias Popp, Dennis Tschechlov, Stefan Wagner, and Alfred Zimmermann. Towards a collaborative repository for the documentation of service-based antipatterns and bad smells. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*.