Towards a Tool-based Approach for Microservice Antipatterns Identification

Rafik Tighilt¹, Manel Abdellatif², Naouel Moha¹, Yann-Gaël Guéhéneuc³

tighilt.rafik@courrier.uqam.ca, manel.abdellatif@polymtl.ca, moha.naouel@uqam.ca, yann-gael.gueheneuc@concordia.ca

¹ Université du Québec à Montréal
² Polytechnique Montréal
³ Concordia University
About the presenter

RAFIK TIGHILT

I’m a MSc student in computer science at Université du Québec à Montréal, currently working on microservice-based system antipatterns.

I previously got a MSc in IT systems management. I worked on the industry for 3 years including founding my own startup before starting the current MSc program in computer science at UQAM.

My fields of interest include software engineering, web development and competitive programming.
Outline

- About microservices
- Approach
- Microservice antipatterns
- Meta-model definition
- Meta-model components
- Detection rules
- Discussion
- Threats to validity
- Conclusion
About Microservices

- Microservice architecture is more and more popular
- Adopted by industry leaders (Amazon, Netflix, Riot Games, etc...)
- Consists of independently deployable and manageable services
- Each microservice fulfils a single business capability
- Allow automated deployment
- Offer greater agility
- Reduce the complexity of handling application scalability
But...

- Microservices are highly volatile
- Microservices are very dynamic
- Microservices are continuously developed and deployed
- Like any other system, microservices face challenges with maintainability and evolution

This factors can lead to the introduction of poor solutions to recurring design and implementation problems (ie. Antipatterns).
How can we automatically identify antipatterns in microservice-based systems?
Approach

- Literature and open-source systems reviews
- 27 papers obtained describing microservices antipatterns
- 67 open-source microservice-based systems manually analyzed
- Establishment of a catalog of microservice antipatterns
- Definition of a meta-model for the automatic analysis
- Definition of detection rules for each of the antipatterns
Microservice antipatterns

Wrong cuts

Microservices organized around technical layers instead of business capabilities.

Cyclic dependencies

Circularly co-dependent microservices
Microservice antipatterns

Mega Service

A microservice serving multiple business capabilities, not manageable by a single team.

Nano Service

Multiple microservices working together to fulfil a single business capability.
Microservice antipatterns

### Shared libraries

- μservice 1
- μservice 2
- μservice 3
- μservice 4

Library bin

Single libraries / files used by multiple microservices.

### Hardcoded endpoints

```javascript
module.exports = {
    catalogueUrl: util.format("http://catalogue\%s", domain),
    tagsUrl: util.format("http://catalogue\%s/tags", domain),
    cartsUrl: util.format("http://carts\%s/carts", domain),
    ordersUrl: util.format("http://orders\%s", domain),
    customersUrl: util.format("http://user\%s/customers", domain),
    addressUrl: util.format("http://user\%s/addresses", domain),
    cardsUrl: util.format("http://user\%s/cards", domain),
    loginUrl: util.format("http://user\%s/login", domain),
    registerUrl: util.format("http://user\%s/register", domain),
};
```

URLs, IP addresses, hostnames and other endpoints hardcoded in the source code.
Microservice antipatterns

**Manual configuration**

Configuration manually pushed for each microservice.

**No API Gateway**

Consumer applications communicate directly with individual microservices.
Microservice antipatterns

Timeouts

Timeout values are set and hardcoded in HTTP requests inside microservices source code.

No CI/CD

No Continuous Integration / Continuous Delivery machinery in the process of developing, testing and deploying microservices.
Microservice antipatterns

Multiple service instances per host

Host A
- μservice 1
- μservice 2

Host B
- μservice 3
- μservice 4

Multiple microservices are deployed on a single host.

Shared persistence

Multiple microservices share a single database.

μservice 1

μservice 2

μservice 3

μservice 4

Shared database
Microservice antipatterns

No API versioning

No information available about a microservice version.

Local logging

Microservices have their own logging mechanism.
Microservice antipatterns

No healthcheck

Microservices are not periodically health-checked to verify availability.

Insufficient monitoring

Microservices performance and failures are not tracked or monitored.
Meta-model definition

- The meta-model encapsulate the needed information to identify microservice antipatterns
- Consists of 13 components
- Each component contains some information about the system and its microservices
Meta-model components

System
Holds information about the system itself (Name, version, is git repository, etc...)

Diagram showing dependencies and components related to the System.
Meta-model components

Git repository
If the system is a git repository, contains information such as repository URL, number of commits, number of contributors, etc...
Meta-model components

Microservice
Stores information about single microservices (programming languages, LOCs, etc...)
Meta-model components

Dependency
Contains information about a single dependency (Name, type, etc...)
Meta-model components

Config
Holds information about configuration (Key value pairs, etc...)
Meta-model components

Env
If available, represents environment variables used by the system
Meta-model components

Deploy
Contains deployment information (Docker instructions, etc...)
Meta-model components

**Code**
Stores information about a single source code file (Programming language, LOCs, etc...)

Meta-model components

If available, this component contains information about container images of the system.
Meta-model components

Server
Holds information about deployment servers of the system
Meta-model components

HTTP
Stores information about HTTP requests (Endpoints, source file, etc...)
Meta-model components

Database
Stores information about database queries (query type, source file, etc...)

Diagram showing dependencies and components such as System, Microservice, Dependency, Config, Env, Deploy, Code, HTTP, DB, Image, Server, Import.
Meta-model components

Import
Contains information about imported classes and packages
Detection rules

For each antipattern, we established a set of detection rules to assess its presence or not in a given microservice based system.

This rules are detailed in the following slides.
<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrong cuts</td>
<td>One file type in source code, connects to other microservices containing only one file type</td>
</tr>
<tr>
<td>Cyclic dependencies</td>
<td>Microservice performing calls to each other in a circular way</td>
</tr>
<tr>
<td>Mega service</td>
<td>Relatively high number of lines of code, relatively high number of dependencies</td>
</tr>
<tr>
<td>Nano service</td>
<td>Relatively low number of lines of code, relatively low number of dependencies</td>
</tr>
<tr>
<td>Shared libraries</td>
<td>Binary files, source files and libraries shared between two or more microservices</td>
</tr>
<tr>
<td>Hardcoded endpoints</td>
<td>No service discovery libraries in the dependencies, URLs and endpoints in the source code</td>
</tr>
<tr>
<td>Manual configuration</td>
<td>No configuration management libraries in the dependencies, independent configuration files</td>
</tr>
<tr>
<td>No API gateway</td>
<td>No API gateway libraries in the dependencies</td>
</tr>
<tr>
<td>Issue</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Timeouts</td>
<td>No circuit breaker libraries in the dependencies, hardcoded timeout values in the source code</td>
</tr>
<tr>
<td>No CI / CD</td>
<td>No CI / CD libraries in the dependencies, no CI / CD information in the repository if available</td>
</tr>
<tr>
<td>Multiple service instances per host</td>
<td>Shared deployment scripts between two or more microservices</td>
</tr>
<tr>
<td>Shared persistence</td>
<td>Shared data / database configuration between two or more microservices</td>
</tr>
<tr>
<td>No API versioning</td>
<td>No version information in endpoints and URLs, no version headers in HTTP requests</td>
</tr>
<tr>
<td>Local logging</td>
<td>No distributed logging libraries in the dependencies</td>
</tr>
<tr>
<td>No healthcheck</td>
<td>No healthcheck libraries in the dependencies, no healthcheck URL, no healthcheck instructions</td>
</tr>
<tr>
<td>Insufficient monitoring</td>
<td>No monitoring libraries in the dependencies</td>
</tr>
</tbody>
</table>
Discussion

- There are no fully automated tool-based approach to identify microservice antipatterns in the literature, and our approach aims to fill this gap.
- Our meta-model only requires source code and do not rely on documentation and other artifacts.
- We aim to help developers minimize antipatterns in their source code.
- We contribute to the maintenance and evolution of microservice-based system with a generic, comprehensive and consensual definitions of antipatterns and an approach to identify these antipatterns.
Threats to validity

- They are potentially other antipatterns that we did not find / include in our study.
- The detection rules are subject to our interpretation and can be subjective (eg. Mega service). However, we tried to minimize this by considering microservices as part of the system and not standalone.
- We rely on lists of libraries and frameworks to identify some antipatterns, this lists can be extended to include more libraries and frameworks.
Conclusion

- This presentation describes the foundation of our automatic tool-based approach for the identification of antipatterns in microservice-based systems.
- We propose an approach that is robust enough to identify the described antipatterns, yet still extensible and flexible to evolve.
- The tool itself is currently work in progress.
- The tool will be validated by manually analysing 28 microservice-based systems to calculate precision and recall.
THANK YOU!