POLITECNICO DI MILANO

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

Ömer Esas

Advisor: Prof. Elisabetta Di Nitto

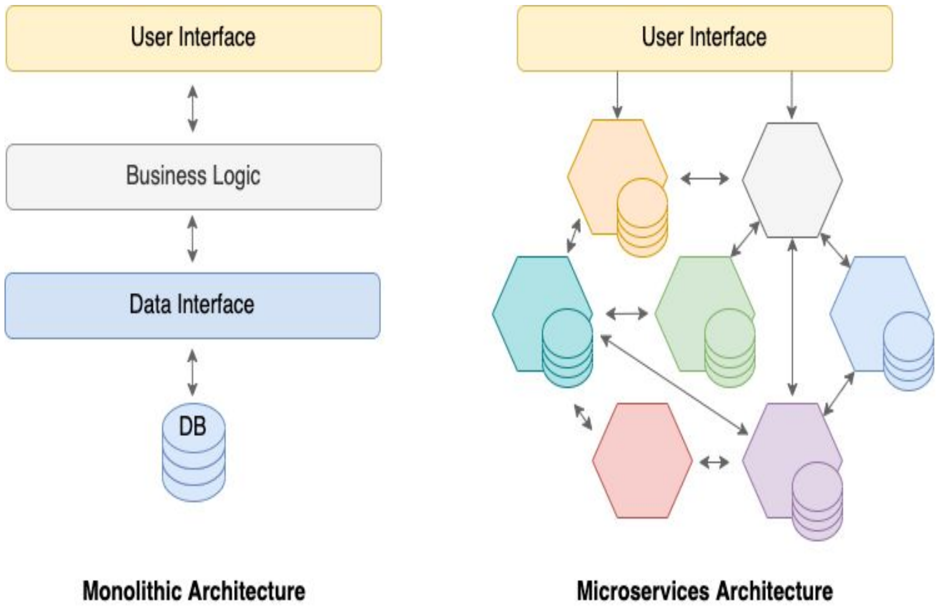
Objectives & Contribution of the Study

- Objectives:
  - Inspect classifications of design patterns and anti-patterns of microservices in the literature
  - Propose one if there is no consensus
  - Observe the presence of patterns and anti-patterns in practical cases
- Contribution:
  - A classification of patterns and anti-patterns of microservices
  - Data about the prevalence of patterns and anti-patterns in open source projects

State of the Art:
Microservices Architecture

General Characteristics:

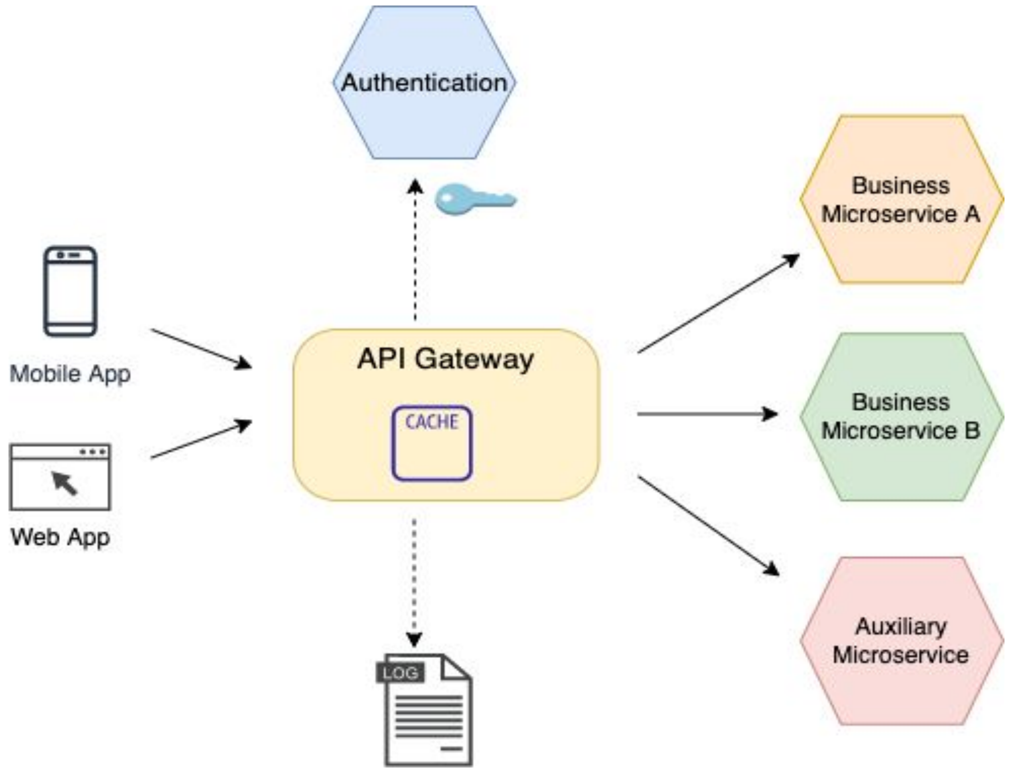
- Application domain divided into smaller domains
- One domain → one team → one microservice
- Independent services in
  - Used technology
  - Implementation
  - Deployment



State of the Art:

Design Patterns of Microservices

- Techniques to solve recurrent microservices problems
- Examples:
  - API gateway (right)
  - Event sourcing
  - Backends-for-frontends
  - Distributed tracing
  - Log aggregator
  - …



State of the Art:
Anti-Patterns of Microservices

- Poor and suboptimal solutions
- Bad practices
- Examples:
  - No API gateway
  - No health check
  - Wrong cut
  - Shared persistence
  - Local logging
  - …

Research Methodology:
Research Questions

- Research Question 1:

  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 structure those design patterns and anti-patterns?

- Research Question 2:

  Which of these design patterns and anti-patterns exist in popular open source microservices applications?

Research Methodology:
Adopted Methodology for RQ1

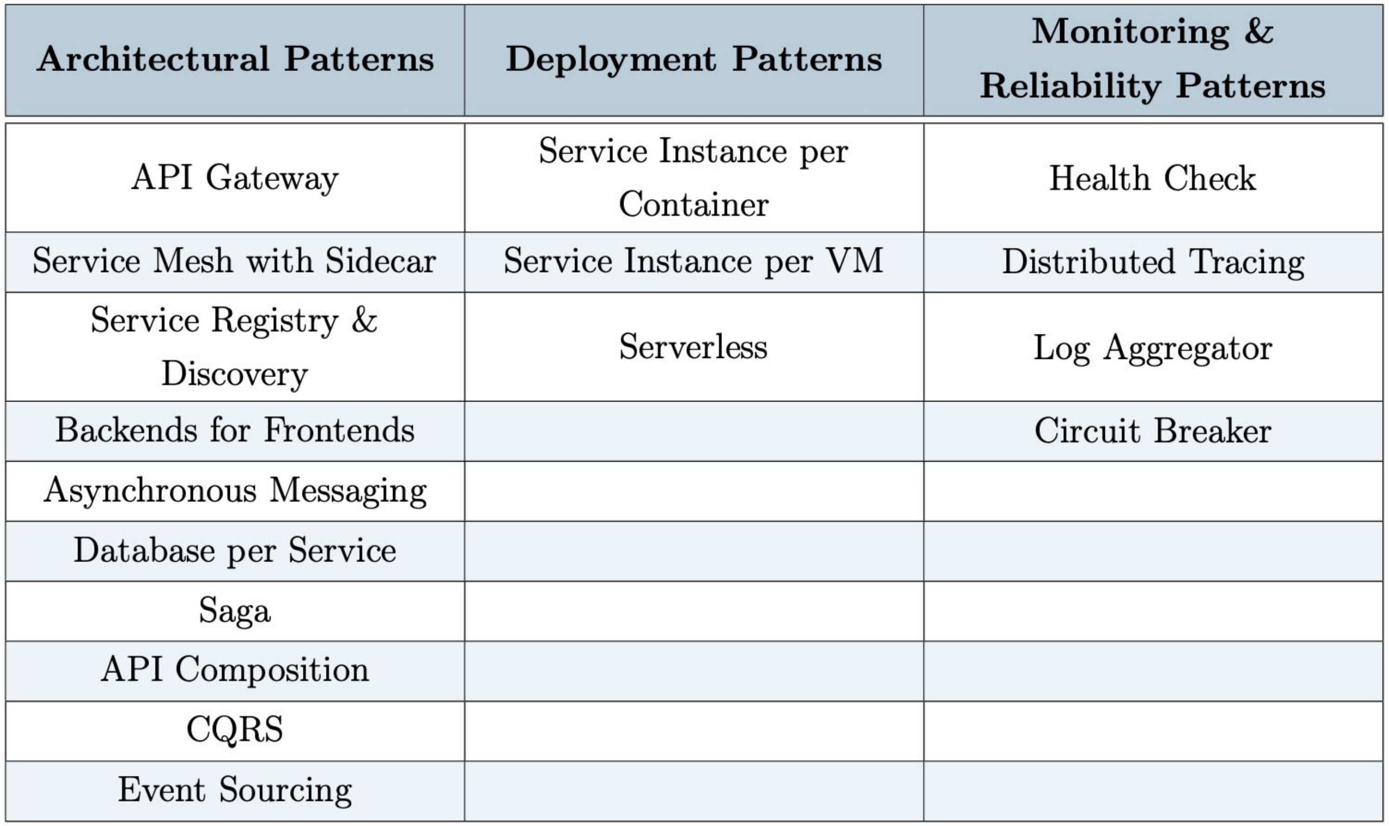
- Research Question 1:

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 structure those design patterns and anti-patterns?

1. Querying digital libraries such as IEEE Explore, ACM Digital Library, Springer, Scopus, Google Scholar
2. Applying snowballing and omitting studies without classification
3. Analysing classifications
4. Consulting systematic mapping studies
5. Developing classification proposal

Result of Research Question 1

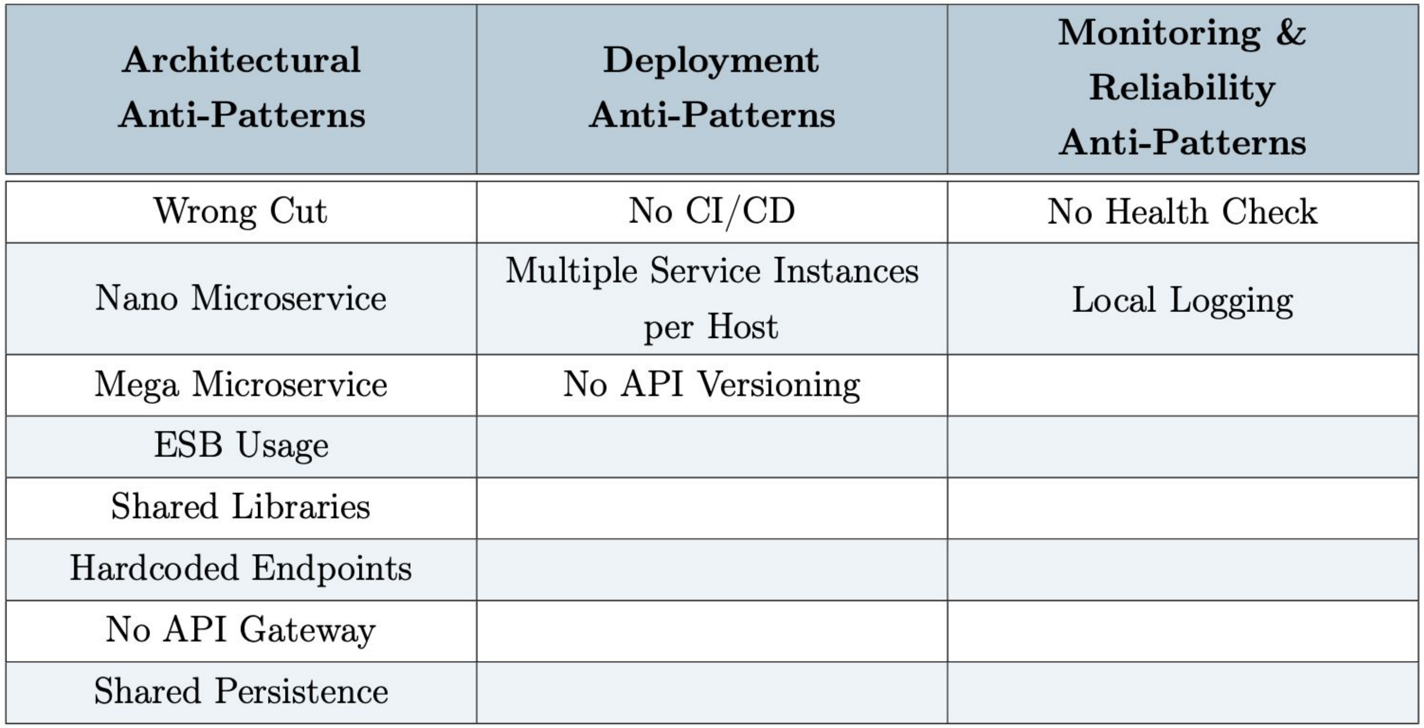
- Proposed classification of design patterns



Architectural Patterns	Deployment Patterns	Monitoring & Reliability Patterns
API Gateway	Service Instance per Container	Health Check
Service Mesh with Sidecar	Service Instance per VM	Distributed Tracing
Service Registry & Discovery	Serverless	Log Aggregator
Backends for Frontends		Circuit Breaker
Asynchronous Messaging		
Database per Service		
Saga		
API Composition		
CQRS		
Event Sourcing		

Result of Research Question 1

- Proposed classification of anti-patterns



Architectural Anti-Patterns	Deployment Anti-Patterns	Monitoring & Reliability Anti-Patterns
Wrong Cut	No CI/CD	No Health Check
Nano Microservice	Multiple Service Instances per Host	Local Logging
Mega Microservice	No API Versioning	
ESB Usage		
Shared Libraries		
Hardcoded Endpoints		
No API Gateway		
Shared Persistence		

Research Methodology:
Adopted Methodology for RQ2

- Research Question 2:

> Which of these design patterns and anti-patterns exist in popular open source microservices applications?

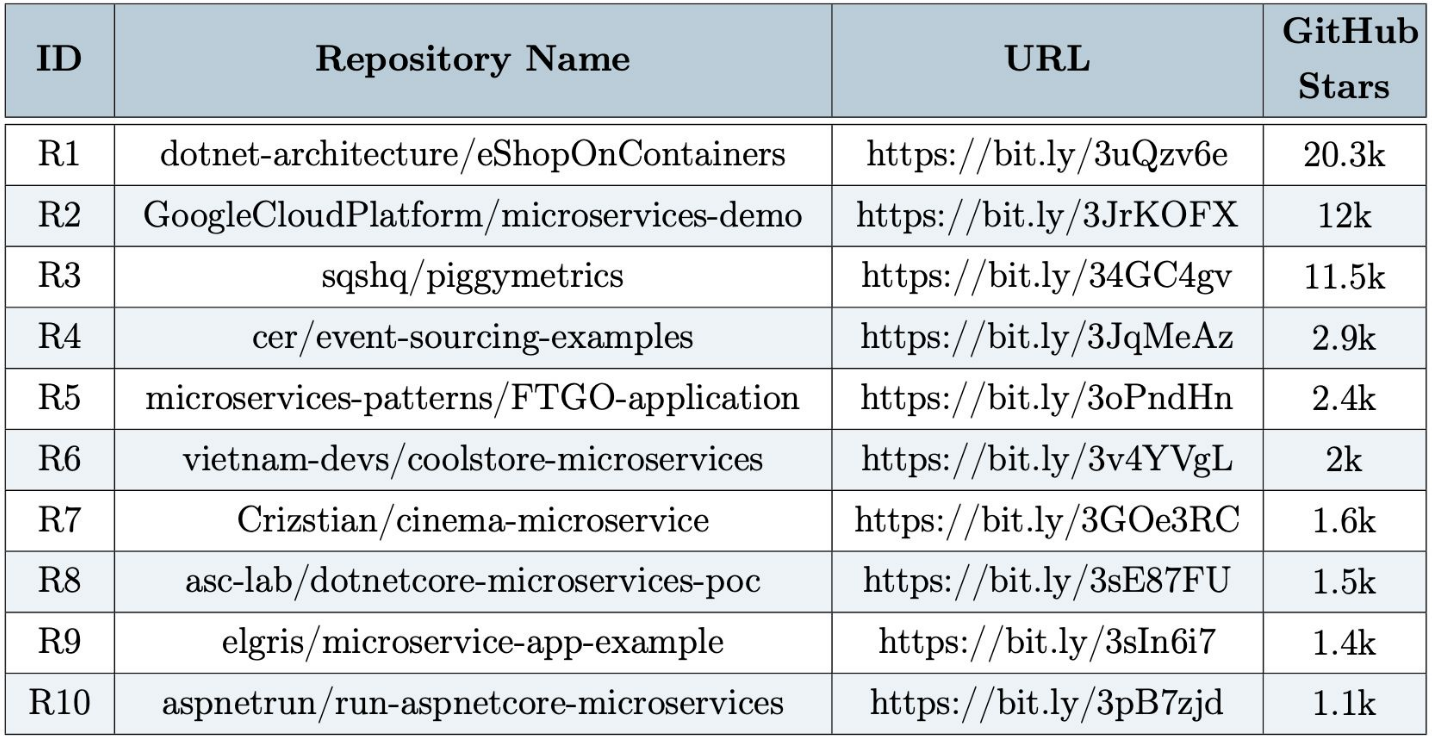
1. Querying GitHub
2. Selecting 10 applications with most GitHub stars
3. Excluding
   a. "saga" pattern → business logic
   b. "shared libraries" anti-pattern → automated tool for different languages
4. Detecting remaining 28 patterns and anti-patterns

Research Methodology:
Adopted Methodology for RQ2

- Context: Different technologies in 10 applications
- Problem: How to detect patterns and anti-patterns?
- Solution:
    - Manual inspection on the repository, source code, deployment files (Docker and Kubernetes .yaml) and dependency files (pom.xml)
    - Read documentation of used libraries and frameworks
    - Ad-hoc methods such as regular expression for “hardcoded endpoints” anti-pattern, substring search “\v” for “no API versioning” anti-pattern and “\hc” for health check pattern

Result of Research Question 2

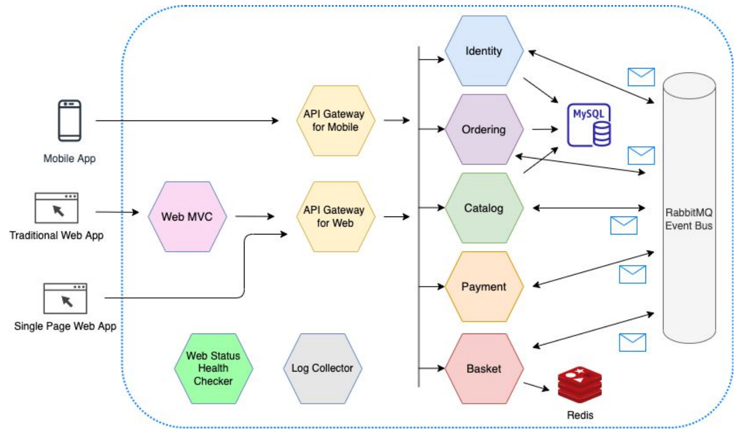
- Selected 10 open source projects



ID	Repository Name	URL	GitHub Stars
R1	dotnet-architecture/eShopOnContainers	https://bit.ly/3uQzv6e	20.3k
R2	GoogleCloudPlatform/microservices-demo	https://bit.ly/3JrKOFX	12k
R3	sqshq/piggymetrics	https://bit.ly/34GC4gv	11.5k
R4	cer/event-sourcing-examples	https://bit.ly/3JqMeAz	2.9k
R5	microservices-patterns/FTGO-application	https://bit.ly/3oPndHn	2.4k
R6	vietnam-devs/coolstore-microservices	https://bit.ly/3v4YVgL	2k
R7	Crizstian/cinema-microservice	https://bit.ly/3GOe3RC	1.6k
R8	asc-lab/dotnetcore-microservices-poc	https://bit.ly/3sE87FU	1.5k
R9	elgris/microservice-app-example	https://bit.ly/3sIn6i7	1.4k
R10	aspnetrun/run-aspnetcore-microservices	https://bit.ly/3pB7zjd	1.1k

Result of Research Question 2

- Overview of the project

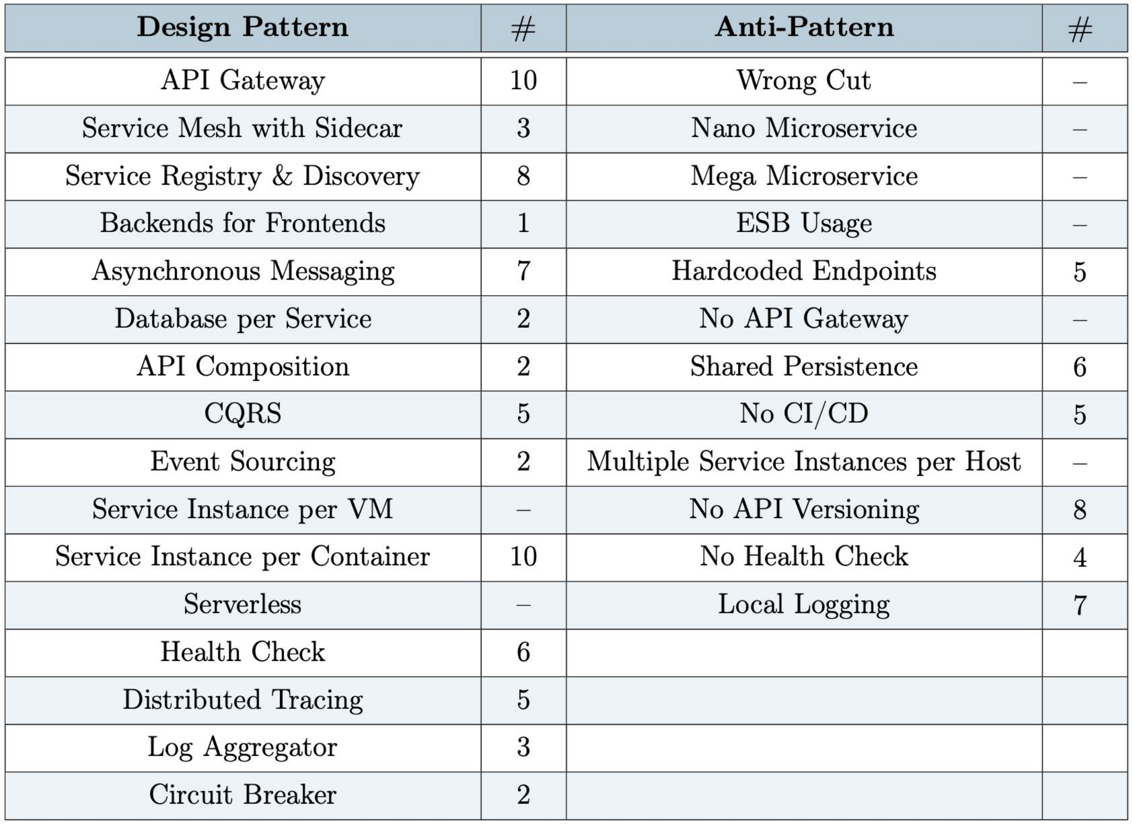


- Presence of patterns and anti-patterns

Design Pattern	✓\–	Anti-Pattern	✓\–
API Gateway	✓	Wrong Cut	–
Service Mesh with Sidecar	✓	Nano Microservice	–
Service Registry & Discovery	✓	Mega Microservice	–
Backends for Frontends	✓	ESB Usage	–
Asynchronous Messaging	✓	Hardcoded Endpoints	–
Database per Service	–	No API Gateway	–
API Composition	✓	Shared Persistence	✓
CQRS	✓	No CI/CD	–
Event Sourcing	–	Multiple Service Instances per Host	–
Service Instance per VM	–	No API Versioning	–
Service Instance per Container	✓	No Health Check	–
Serverless	–	Local Logging	–
Health Check	✓		
Distributed Tracing	–		
Log Aggregator	✓		
Circuit Breaker	–		

Result of Research Question 2

- The total number of detected patterns and anti-patterns



Design Pattern	#	**Anti-Pattern**	#
API Gateway	10	Wrong Cut	–
Service Mesh with Sidecar	3	Nano Microservice	–
Service Registry & Discovery	8	Mega Microservice	–
Backends for Frontends	1	ESB Usage	–
Asynchronous Messaging	7	Hardcoded Endpoints	5
Database per Service	2	No API Gateway	–
API Composition	2	Shared Persistence	6
CQRS	5	No CI/CD	5
Event Sourcing	2	Multiple Service Instances per Host	–
Service Instance per VM	–	No API Versioning	8
Service Instance per Container	10	No Health Check	4
Serverless	–	Local Logging	7
Health Check	6		
Distributed Tracing	5		
Log Aggregator	3		
Circuit Breaker	2		

Conclusion

- Summing up:
  - Only a few studies in the literature that classify patterns and anti-patterns of microservices, no consensus on classifications
  - We proposed “architectural”, “deployment” and “monitoring & reliability” categories for patterns and anti-patterns
  - Unequal distribution of patterns and anti-patterns in the examined open source microservices projects
  - API gateway and containers in all 10 projects
- For future work:
  - Consolidating the project analysis approach through metrics and automated tools
  - Focusing on detection of “saga” pattern and “shared libraries” anti-pattern
  - Inspecting more open source projects to generalise and validate the results

Thank you for listening,

Any Questions?