2018 25th Asia-Pacific Software Engineering Conference (APSEC)

Actual Use of Architectural Patterns in Microservices-based Open Source Projects

Gastón Márquez and Hernán Astudillo
Toeska research team, Universidad Técnica Federico Santa María
Valparaíso, Chile
Email: gaston.marquez@sansano.usm.cl, hernan@inf.utfsm.cl

***Abstract*—Microservice-based systems instantiate an architectural style that conceives of systems as sets of modular, customer-centric, independent, and scalable services. These systems express a similar essential structural organization and seems appropriate to design them using architectural patterns because these combine an understanding of the system domain and good practices. Code repository platforms provide the developer community with ideas and examples about microservice systems, but since they are in early adoption, there is still no clear notion of which actual microservice systems incarnate architectural patterns (if any), reducing the use of frameworks and the achievement of quality attributes. This paper extends a previous study on architectural patterns for microservices in academic and industry sources. We explored which architectural patterns for microservices are used in actual microservice-based open source systems, by subjecting thirty well-known open source projects to a comprehensive multi-criteria code and design review. We found that (1) open source projects use only a few architectural patterns broadly; (2) most projects use the same few frameworks; (3) there are very few microservice architectural patterns as such; and (4) what most projects use (what was previously called) are SOA patterns. This study shows that microservice systems builders do use architectural patterns, but only a few of them. It remains to be determined whether additional patterns would be productively used to build microservice systems, or the few ones currently used are the only ones actually necessary.**

***Index Terms*—Microservices, architectural patterns, open source projects, quality attributes, framework**

I. Introduction

The *microservices* architectural style is an approach to develop single applications as suites of small services, each running in its process and communicating with lightweight mechanisms [1]. Microservices have become enormously popular because traditional monolithic architectures no longer meet the needs of scalability and rapid development cycle[2], and the success of large companies (mainly Netflix[1]) in building and deploying services is a strong motivation for other companies to consider making the change. This movement provides fertile ground to identify recurring problems and solutions, which eventually translates into design and architectural patterns.

The knowledge acquired in industrial and academic experiences [3] [4] shows that using patterns (specifically, architectural patterns) in systems development allows (1) discerning the domain where the system is being developed, (2) satisfying systemic properties ("quality attributes" or QAs [5]), and (3) creating large-scale reuse design techniques (frameworks [6]).

[1]https://www.netflix.com/

We have already conducted studies [7] [8] to determine whether architectural patterns have been used in the development of microservices-based systems, and we found that some well-known architectural patterns are indeed used to develop microservices architectures, but also new patterns are emerging as alternatives and several patterns have been "imported" from other fields. Thus, article expands our previous study to explore whether microservices architectural patterns are actually used to build open source systems, focusing on three aspects: microservices architectural patterns, QA's, and frameworks.

The main **contributions** of this article are:

- A catalog of microservices architectural patterns reported in academic and industrial literature.
- A correlation between QA's and microservices architectural patterns.
- A list of technologies, deployed as frameworks, used in the construction of microservices-based systems with microservices architectural patterns.
- A comparative analysis of SOA and microservices architectural patterns.

The remainder of this article is structured as follow: Section II provides background; Section III illustrates the research process; Section IV describes the results; Section V discusses the threats to validity; Section VI reviews related work; and Section VII summarizes, concludes, and describe the future work.

II. Background

Two main concepts are key for this study: microservices-based systems, and architectural patterns.

A. Microservices-based systems

The microservices architectural approach derives from the service-oriented architecture (SOA) approach [9]. SOA is a software architectural pattern where application components provide services to other components via a communications protocol over a network, and communication may involve simple data passing or include several coordinating services connecting to each other.

Many large systems evolve from self-contained *monolithic* applications built of interconnected, interdependent components (see Figure 1-a) to collections of large services (i.e. traditional SOA), and further to collections of small, autonomous, lightweigth-connected services (see Figure 1-c)).

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00017

The market's high pace of demand for new application features requires changes both in the applications themselves (loose coupling and high scalability) and in the way they are built (loose team dependencies and fast deployment). Microservices address both concerns since small services can be built and deployed by independent development teams; the concomitant freedom allows teams to focus on improving each service and increase business value. Hence, in practice, DevOps (Development and Operations) and Continuous Delivery [10] are a close fit for microservice architectures.

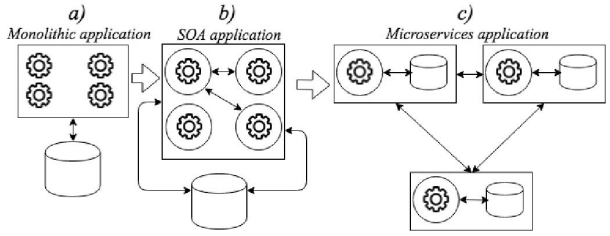


Fig. 1. From monolithic applications to microservices-based systems

B. Architectural patterns

Architectural patterns [11] have three elements:

- *Context*: A recurring, common situation in the world that gives rise to a problem.
- *Problem*: The situation that arises in the given problem, appropriately generalized.
- *Solution*: A successfully architectural resolution to the problem, appropriately abstracted.

Architectural patterns allow reusing knowledge gained by practitioners addressing some recurring situation. This reuse implies a description of the solution, making explicit what QAs are provided by the static and runtime configuration of elements which compose the architectural pattern.

As an example, the well-known Broker architectural pattern (Figure 2) has a *context* of distributed environments, where a recurring *problem* is how to structure the system to facilitate service invocations among remote components, and a *solution* is to introduce a broker component that decouples clients and servers by making the latter's services available to the former via requests to the broker itself.

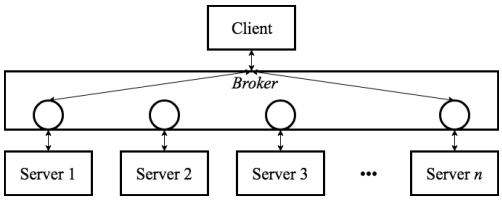


Fig. 2. The Broker architectural pattern

III. Research Methodology

In this section, we describe the research methodology followed in this study, which includes the following stages: the research scope; the overall strategy; the goals and research questions; and (in some detail) the study steps.

A. Scope

Our research approach is architectural pattern identification through a systematic literature review process [8] [7]. On the other hand, we limited the study to the Github repository because it is one of the primary platforms for open source projects, but future studies may include other repositories.

B. Strategy

We defined three phases, each one containing several activities:

a) Phase I - Exploration: We explored and synthesized existing characterizations of microservices architectural patterns from academic and industry sources since the variety in proposed patterns suggests that there is not yet an agreed upon definition of *microservices architectural pattern*.

b) Phase II - Refinement: We established a set of criteria to filter microservices architectural patterns and establish the patterns corpus as the basis for this study.

c) Phase III - Verification: We analyzed a set of microservices-based open source projects to (1) identify the architectural patterns used in their development and (2) compare these patterns with those found in our previous study [8] [7].

C. Goals and research questions

We established the goal of this study, based on the Goal-Question-Metric [12] approach, which is *distinguish which microservice architectural patterns identified in academic and industry sources are actually used in well-known microservices-based open source systems.*

To achieve this goal, we defined four research questions:

- **RQ1**: *Which microservice architectural patterns have been proposed in academic and industry sources?* Rationale: This question enables us to obtain the initial set of reported architectural patterns for microservices in industry and academy sources. In this way, we can know if there are trends, factors, concepts, others; that characterize architectural patterns in microservices-based systems.
- **RQ2**: *Which quality attributes are addressed by these patterns?* Rationale: QAs are requirements that specify criteria to evaluate the operation of systems. Also, they are part of the (non-functional) requirements of the system and are characteristics that must be expressed quantitatively, such as performance, security, availability, among others [5]. The intention of using architectural patterns in the project development is to satisfy these QAs either partially or entirely. Therefore, this RQ seeks to illustrate which QAs address the architectural patterns reported in RQ1.
- **RQ3**: *Which of the microservice and architectural patterns reported in academic and industry sources are actually used in microservices-based open source systems?* Rationale: This RQ illustrates the correlation between the results of the RQ1 versus the architectural patterns identified in microservices-based open source projects,

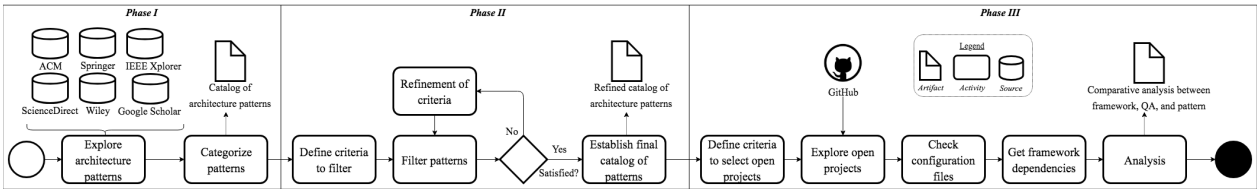


Fig. 3. Research process

through an empirical comparison. This RQ intends to verify if the developers of open source projects use some of the patterns reported in RQ1.

- **RQ4**: *Which of the microservice and architectural patterns reported in academic and industry sources are "really" patterns for microservices (and not just recycled from other architectural styles)?* Rationale: This RQ intends to establish what is understood as a microservice architectural pattern. For this purpose, will be made a comparative analysis on whether the architectural patterns reported in RQ1 are effectively microservices architectural patterns or are reused patterns of other disciplines.

D. Process

The research process (see Figure 3) is divided into three phases, each one with a set of activities and an artifact as a final result. In Phase I, most activities focused on carrying out an exploratory study in several electronic databases to gather an initial set of reported microservices architectural patterns. The search string used to obtain the set of patterns was: "architectural" AND ("micro service" OR "microservi" OR "micro-servi") AND "pattern" AND "design", and the search period was between September 2017 and January 2018. In this same phase, we categorized the patterns defining their context, problem, and solution.

In Phase II, we performed the refinement tasks, and filtered the architectural patterns using several criteria:

- **C1**: Patterns to migrate from monolithic to microservices This criterion is to filter those patterns that represent solutions to migrate from monolithic architectures to microservices. Our research is focused on architectural patterns for systems already built using the microservices approach.
- **C2**: Redundancy In Phase I, we realized that some patterns with different names describe the same context, problem, and solution; some cases, such as the Containerize the Services [13] and Container [14], are quite similar indeed. In such cases, we gathered identical patterns into a single one.
- **C3**: Poor documentation All reported architectural patterns have descriptions, but sometimes it is quite weak. In such cases, we omitted these patterns from the analyzed corpus since the available documentation did not allow an adequate study.

These criteria were executed sequentially. Finally, the remaining architectural patterns were subjected to quality assessment using three quality criteria (QC) questions:

- **QC1**: Does the architectural pattern describe the *context* in which it is applied? Y (yes), it clearly describes the context; P (partly), it partially describes the context; N (no), it vaguely describes the context
- **QC2**: Does the architectural pattern describe the *problem* it is trying to solve? Y, it clearly describes the problem; P, it does not directly describe the problem; N, we could not determine which problem it tries to solve
- **QC3**: Does the architectural pattern describe the *solution* for this context and problem? Y, it clearly describes the solution; P, it partially describes the solution; N, we could not determine what solution it proposes

Each pattern was evaluated with the QCs using the scores $Y = 1$, $P = 0.5$, $N = 0$.

Finally, in Phase III we explored the open source projects to obtain the architectural pattern used in them. In GitHub, we used the following string: ("micro services" OR "microservices" OR "micro-services") AND ("system"). Then, we used these inclusion and exclusion criteria for each project:

- *Inclusion criteria*: Benchmark requirements for microservices projects proposed by Aderaldo et al. [15], and projects with source code available.
- *Exclusion criteria*: Projects with no solid information (basic examples, projects in process, others), tools to build microservices (instead of frameworks), component-based projects to build microservices (e.g., projects just for gateway components to microservices), and projects used as an example for lectures or talks.

Since each selected project may use one or more frameworks, we gathered three types of configuration files to determine frameworks and dependencies:

- `docker-compose.yml`: configures the application's services through Docker[2], allowing work in all environments: production, staging, development, testing, and continuous integration workflows.
- `POM.xml`: is the fundamental unit of work (Project Object Model, POM) in Maven[3]; this XML file contains project information and configuration details to build it.
- `build.gradle`: used by Gradle[4] to define compilation settings that apply to all project modules.

To automate our search for frameworks, we developed a Java parser that reads the files and shows framework dependencies, using the following

[2]https://www.docker.com/
[3]https://maven.apache.org/
[4]https://gradle.org/

libraries: `jackson-dataformat-yaml`[5] to read `docker-compose.yml` files, `MavenXpp3Reader` [6] to read `POM.xml` files and a custom tool (developed by us) to read `build.gradle` files.

IV. Results

This section describes the results from our research process, broken down in RQ's, and concludes with a discussion about key findings.

A. *RQ1: Which microservice architectural patterns have been proposed in academic and industry sources?*

We found 164 architectural patterns proposed for microservices in academic and industry sources (see Table I).

TABLE I
ARCHITECTURAL PATTERNS IN ACADEMIA AND INDUSTRY

***Academia* 32%**			
Ref.	Year	Author(s)	# of architectural patterns
[13]	2015	Balalaie et al.	15
[14]	2016	Butzin et al.	7
[16]	2016	Qanbari et al.	4
[17]	2016	Brown et al.	17
[18]	2016	Messina et al.	1
[19]	2018	Taibi et al.	8
***Industry* 68%**			
[20]	2017	C. Richardson	42
[21]	2017	N. Shalom	1
[22]	2017	Mulesoft	6
[23]	2015	A. Gupta	6
[24]	2017	P. Calçado	3
[25]	2017	L. Majerowicz	3
[26]	2016	R. Dhall	4
[27]	2017	M. Wasson	9
[28]	2017	M. Churchman	6
[29]	2018	Arcitura	32
Total			**164**

We are struck by the significant number (and redundancy) of patterns found. We initially expected to see a few patterns since the field is still immature; however, we found a lot of evidence, but redundant. This finding was the primary motivation to create Phase II. Regardless of the above, given the large volume of patterns, we found necessary to create a classification to categorize the patterns [7]. Consequently, Figure 4 summarizes the name of the 17 patterns obtained in Phase II.

To create the classification, we held team meetings to identify trends described in the architectural patterns, through (1) brainstorming sessions and (2) following the strategy proposed by Velardi et al. [30], which is an automated procedure to achieve a significant speed-up factors (in our case, categories) in the development of resources (architectural patterns) with human validation and refinement (see the categories in Table II).

Figure 5 shows the results of the refinement. We define the values "yes" and "no" to evaluate the patterns. One of the most significant phenomenon we observe is that eighty-four patterns ($\approx$ 51% of the initial set of 164 patterns) are repeated (Figure 5-*b)*). Finally, from this refinement (corresponding to the $\approx$ 14% of the initial set), we obtained a set of 23 patterns that were submitted to the quality assessment (Figure 5-*c)*).

The results of our quality assessment show that 17 architectural patterns get the full score (see Figure 6). We decided to use only the patterns that meet the maximum score. The detailed description of the context, problem, and solution of each of these microservices architectural patterns is available in https://goo.gl/QggVn6.

B. *RQ2: Which quality attributes are addressed by these patterns?*

To identify the QA's addressed by each architectural pattern, we made a thorough reading of the literature of each pattern. In some cases, we found more than one pattern that satisfied a QA. Figure 4 shows the QAs summary in the y-axis.

Figure 7 represents the distribution of QA's in the architectural patterns, revealing that *availability* is one of the most-addressed QA's. Remembering that [31] something is available if it is accessible and usable when an authorized entity demands access, we infer that a critical concern of the found architectural patterns is to keep the system accessible at all times. This matches S. Newman's observation [2] that microservices need to be resilient to failures and able to restart often on another machine for availability.

Although the other QA's reported in Figure 7 are also relevant, we notice that *observability* is gaining importance in microservices-based development. There are opposite opinions about associating monitoring with observability (AMP17), since *monitoring* is a verb (something that is done) but *observability* is an adjective (something that somebody has). However, from a control theory perspective, the AMP17 pattern satisfies the *observability* QA because it is a system property concerning how well the internal states of a system can be inferred from its external outputs [32]. Since microservices-based systems are in constant evolution, they fit well with practices like continuous integration (CI) [33], whose key goals are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates. This feature is framed in the description of observability.

C. *RQ3: Which of the microservice and architectural patterns reported in academic and industry sources are actually used in microservices-based open source systems?*

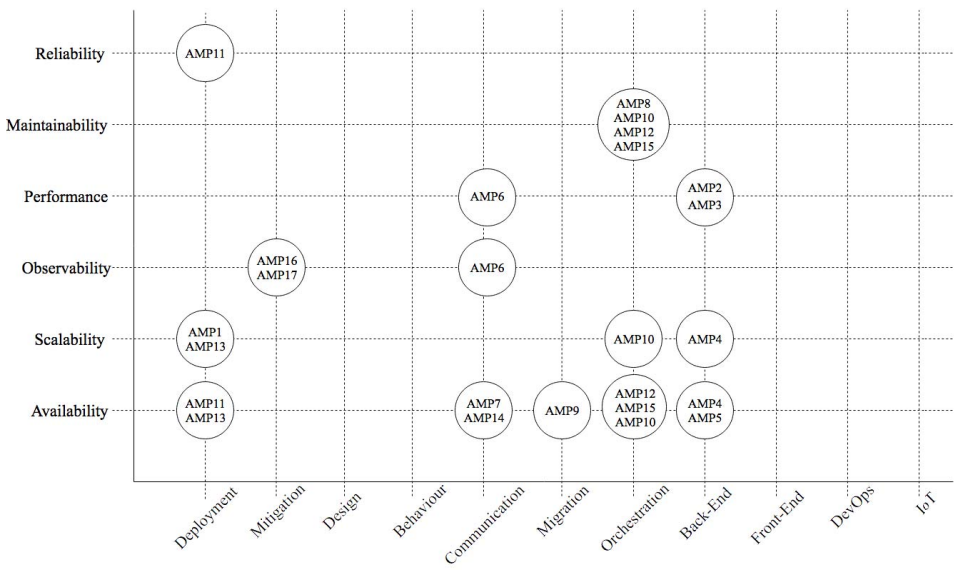
Table III shows the summary of the thirty projects and their respective configuration files checked. For this study, we first selected distributed feature frameworks, where in most projects, the `POM.xml` and `docker-compose.yml` files were analyzed. To complete the frameworks extraction, we also study the source code through a manual process. The goal to perform this task is to analyze how these frameworks work (functionality and dependency) at source code level. This analysis allowed us to refine the framework extraction from

[5]https://github.com/FasterXML/jackson-dataformat-yaml

[6]https://maven.apache.org/ref/3.3.1/maven-model/apidocs/org/apache/maven/model/io/xpp3/MavenXpp3Reader.html

TABLE II
CATEGORIES DESCRIPTIONS

Category	Description
IoT patterns	Patterns that describe solutions for systems of interrelated computing devices, mechanical and digital machines
DevOps patterns	Patterns to help practices that increases an organization's ability to deliver services at high velocity
Front-End patterns	Patterns that support functionalities that are part of the presentation layer
Back-End patterns	Patterns that support functionalities that are part of the data access layer
Orchestration patterns	Pattern for communication between point-to-point connections
Migration patterns	Patterns that help the decomposition of a system into small services are among the other
Communication patterns	Patterns that contribute to the communication processes bases on exchanging messages
Behaviour patterns	Behavioral patterns identify common flexibility forms between services
Design patterns	Object oriented patterns for microservices
Mitigation patterns	Patterns to take preventive measures to reduce the likelihood of the risk
Deployment patterns	Patterns to building and managing the deployment pipeline in software teams



ID	Architecture pattern
AMP1	Backend for Frontend
AMP2	Results Cache
AMP3	Page Cache
AMP4	Scalable Store
AMP5	Key Value Store
AMP6	Log Aggregator
AMP7	Service Registry
AMP8	Enable Continuous Integration
AMP9	Service Discovery
AMP10	Container
AMP11	Circuit Breaker
AMP12	Load Balancer
AMP13	Database is the Service
AMP14	Messaging
AMP15	API gateway
AMP16	Health check
AMP17	Monitor

Fig. 4. Architectural patterns and their corresponding relation regarding to QAs and categories

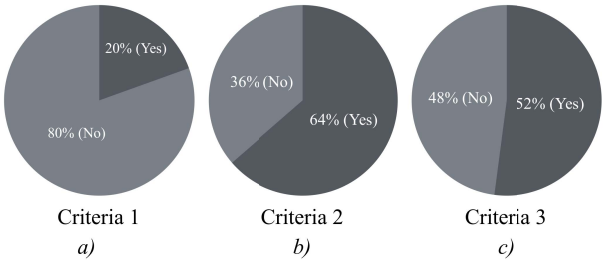


Fig. 5. Quality assessment score distribution. Criteria 1 (Yes:32, No:132), Criteria 2 (Yes:84, No:48), and Criteria 3 (Yes:25, No:23)

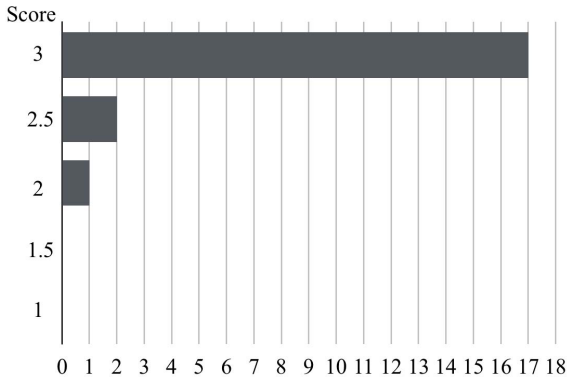


Fig. 6. Distribution of patterns per criteria.

configuration files (see Figure 8). For this purpose, we used the tools Understand[7]and Cytoscape[8] in four steps:

1) Generate the architectural dependency diagram of the project, allowing to identify clusters (as potential microservices)
2) Identify the files related to each cluster
3) Code analysis of these files to find their frameworks
4) Traceability analysis to locate framework references in the source code

The other frameworks that we set aside, such as security and testing frameworks, will be considered in a future study. Sub-

[7]https://scitools.com/features/
[8]http://www.cytoscape.org/

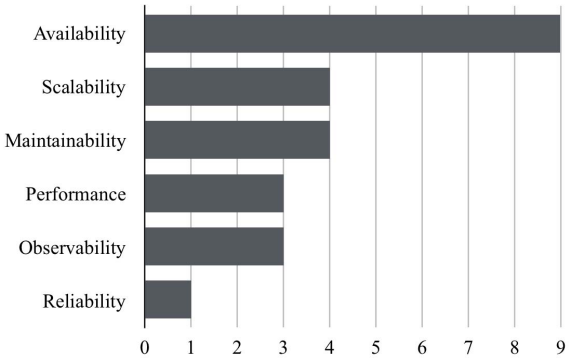


Fig. 7. Quality attributes distribution

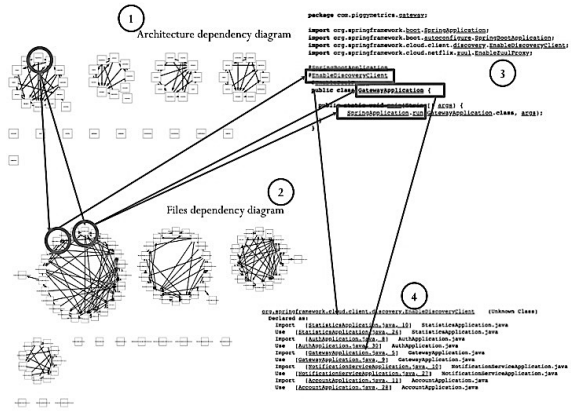


Fig. 8. Manual steps overview

sequently, Figure 9 illustrate which microservice architectural pattern satisfies each framework. To make this linkage, we accessed the documentation of each framework and took a final decision on the framework-pattern relationship in a consensual manner by the research team and other colleagues. The details of the references consulted regarding the frameworks are available at https://goo.gl/QggVn6.

Figure 10 shows the distribution of which patterns, reported in Figure 9, were found in open source projects. One of the most referenced patterns is API Gateway, which plays an important role in acting as a single entry point. Similar to the Façade pattern from object-oriented design, API Gateway encapsulates the internal system architecture and provides an API that is tailored to each client.

Regarding AMP1 and AMP16, these architectural patterns were detected in the project infrastructure (we examined the configuration files and corroborated this using the available documentation), so they did not use frameworks to build them.

TABLE III
PROJECTS SELECTED BY OUR RESEARCH PROCESS

Name	URL	File checked
Gizmo	https://goo.gl/c49vSH	*Manually*
Genie	https://goo.gl/1QpFi5	build.gradle
Graph Processing	https://goo.gl/vV2bGR	POM.xml
Acme air	https://goo.gl/MGK2ur	*Manually*
Movie recommendation	https://goo.gl/yw3NNa	POM.xml
Sock Shop	https://goo.gl/81BdzL	*Manually*
Microservices book	https://goo.gl/mNCZym	POM.xml
Piggy Metrics	https://goo.gl/jLdCih	POM.xml
Netflix microservice	https://goo.gl/NvCjw8	docker-compose.yml
microService	https://goo.gl/snWKck	POM.xml
Share bike	https://goo.gl/i1wehd	POM.xml
Lelylan	https://goo.gl/ySbY54	docker-compose.yml
E-Commerce App	https://goo.gl/y9sqXR	*Manually*
Task track support	https://goo.gl/chwBTh	POM.xml
CAS Microservice	https://goo.gl/hqgeWr	POM.xml
Warehouse microservice	https://goo.gl/CtZ9qr	POM.xml
Microservices Reference	https://goo.gl/VBjGER	POM.xml
Vehicle tracking	https://goo.gl/HqbqR3	*Manually*
EnterprisePlanner	https://goo.gl/wjzoYd	docker-compose.yml
Micro company	https://goo.gl/MmbQCv	POM.xml
Freddy's bbq joint	https://goo.gl/69LW7h	POM.xml
Photo uploader	https://goo.gl/3SHrdZ	docker-compose.yml
Delivery system	https://goo.gl/1oZ67r	POM.xml
Service Commerce	https://goo.gl/KRP8vq	*Manually*
Kenzan Song Library	https://goo.gl/iCxaJE	POM.xml
Blog post	https://goo.gl/18RYyK	POM.xml
Tap-And-Eat	https://goo.gl/JmLoAp	POM.xml
WeText	https://goo.gl/GfYtVd	*Manually*
Pitstop	https://goo.gl/47neif	docker-compose.yml
SiteWhere	https://goo.gl/VD1onH	POM.xml

D. ***RQ4****: Which of the microservice and architectural patterns reported in academic and industry sources are "really" patterns for microservices (and not just recycled from other architectural styles)?*

Table IV shows which architectural patterns found in this study are proper microservices patterns or are borrowings from other fields; indeed, most patterns are SOA patterns [34].

TABLE IV
SOA AND MICROSERVICES ANALYSIS

architectural pattern	SOA	Microservices	References
API Gateway		✓	[35]
Load balancer	✓	✓	[13] [36]
Container		✓	[37]
Key-value store	✓	✓	[38]
Log aggregator		✓	[17]
Messaging	✓	✓	[39] [40]
Service registry	✓	✓	[41] [35]
Data base is the service		✓	[18]
Enable Cont. Integration		✓	[42]
Circuit breaker	✓	✓	[35]
Results cache		✓	[17]
Monitor	✓	✓	[43] [13]
Service discovery	✓	✓	[35] [36] [44]
Page cache		✓	[17]
BackEnd for FrontEnd		✓	[17] [2]
Scalable store		✓	[17]
Health check	✓	✓	[20] [45]

This similarity is based on the next concept: microservices architectural is a share-as-little-as-possible approach that places a heavy emphasis on the idea of a bounded context, whereas SOA is a share-as-much-as-possible approach that

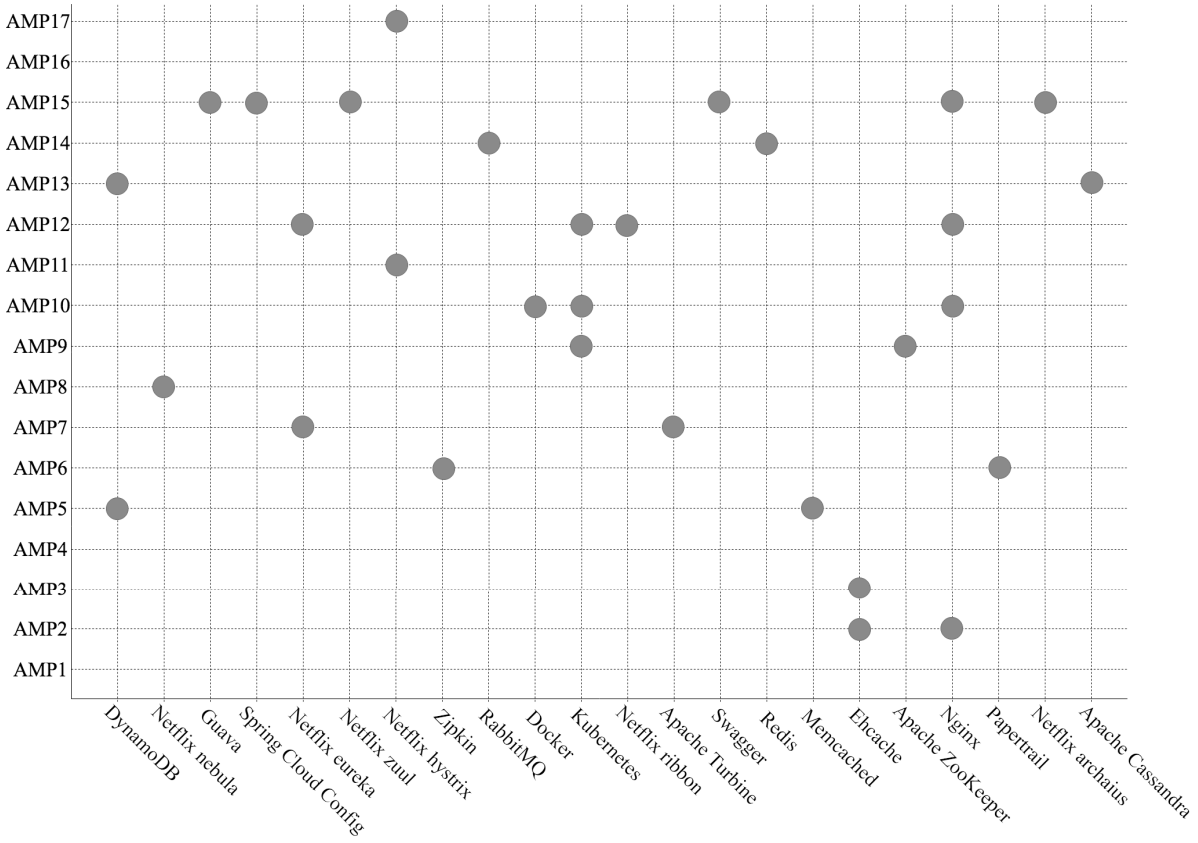


Fig. 9. Relationship between frameworks and microservices architectural patterns

TABLE V
SUMMARY OR MICROSERVICES ARCHITECTURE PATTERNS AND THEIR DESCRIPTIONS

Name	Category	QA(s)	Description
API gateway	Communication	Maintainability Availability	*Context:* Develop multiple versions of the product details user interface *Problem:* How do external clients communicate with the services? *Solution:* A service that provides each client with unified interface to services
Container	Orchestration	Scalability Maintainability	*Context:* Each service is deployed as a set of service instances *Problem:* How to simplify deploying applications? *Solution:* Containers enclose the microservice itself, including all required libraries and data
Log aggregator	Communication	Performance Observability	*Context:* Is required debug problems that cross different components *Problem:* How can view and search all log files of different distributed runtimes? *Solution:* Log Aggregator pulls all files together into a single, searchable database
Database is the service	Deployment	Scalability Availability	*Context:* Most services need to persist data in some database *Problem:* How to build the database architecture in a microservices? *Solution:* Each service has its own private database
Enable cont. integration	Orchestration	Maintainability	*Context:* The number of services is increasing *Problem:* How can always have available production-ready artifacts? *Solution:* Automate the process through pipelines with code, artifact, and servers
Result cache	Back-End	Performance	*Context:* Making network calls to remote databases or services are expensive *Problem:* How improve the performance when it makes many repeated calls to services? *Solution:* Results cache shortcuts the need to make repeated calls to the same service
Page cache	Back-End	Performance	*Context:* It's required use the Backend for Frontends Pattern to build dispatchers *Problem:* How to return more information that can be easily displayed? *Solution:* Page Ccache presents an interface that allows to request a limited subsets of data
BackEnd for FrontEnd	Deployment	Scalability	*Context:* The client server-side functionality needs should be accessible through a single API *Problem:* How represent a channel-specific service interface to a specific client type? *Solution:* Implement customized different single APIs for different types of clients
Scalable store	Back-End	Scalability Availability	*Context:* The application need persistence to represent current and previous user interaction *Problem:* How do you represent persistent state in an application? *Solution:* Put all state in a scalable store where it can be available to application runtimes

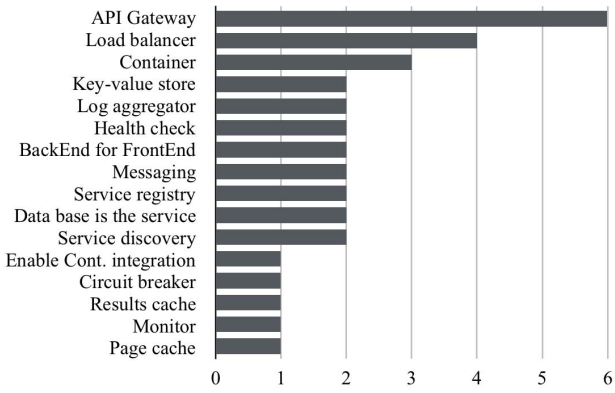


Fig. 10. Distribution of microservices architectural patterns in open source projects

puts a heavy focus on abstraction and business functionality reuse [46]. Some authors [47] assert that SOA and microservices have several elements in common, so it makes sense that some architectural patterns used in microservices are, consequently, patterns used in SOA. To establish this observation, we searched for academic evidence in each pattern. Table IV summarizes those patterns where there is evidence in both SOA and microservices. Finally, we characterized those patterns where we did not find evidence in SOA as *microservices architectural patterns*. To confirm this characterization, we consulted experts to complement the information described in Table IV. In general, experts agreed that some patterns that are reported in microservices are also used in SOA.

Table V describes the nine microservices architectural patterns that we identified in this study. With the columns category, QA(s), and description (context, problem, and solution). For each of these fields, we briefly detailed its description based on the pattern.

E. Discussion

The results obtained for RQ1 show that there is a wide range of proposed architectural patterns for microservices. We have no doubt that the authors who have contributed with patterns have done so with the aim of describing recurring solutions that they have used for a particular type of problems. However, we realize there is redundancy concerning the patterns found.

Taking that observation, only a few patterns satisfy recurrent problems in microservices-based systems, and of these, some patterns are borrowed from other approaches, namely SOA (RQ4). We point this out not to make a gratuitous difference, but to illustrate new knowledge found about patterns. Our results show that we can characterize only nine patterns as correctly of microservices. We say "properly" because we could not find evidence about their use in SOA sources. Notwithstanding the above, this does not mean that in industrial experiences or papers related to distributing computing occur this trend. Therefore, we will use the data recapitulated in the RQ4 to conduct a systematic study on SOA and microservices as future work.

The QA's reported in Figure 7 (RQ2) match the academic literature on microservices. Features like availability are characteristic that sustain the initial conception of microservices. But, a point to consider is the fact that observability is emerging in this discipline. This QA is linked to new concepts that involve microservices, such as DevOps and CI. The essence of these concepts is the control that can exist at the moment of evolving a system. Exploring the contributions of this QA opens the possibility for new research approaches in other QA's such as evolvability and maintainability.

Finally, RQ3 reveals the great importance of the fact that Netflix has released its code, allowing for continuous build and integration into worldwide deployments. On the other hand, Spring Cloud has also had relevance in the development of open source projects. The advantages of using these technologies are their exceptional ability to adapt to new requirements through their frameworks. This adaptation and evolution allow the microservices-based system to satisfy new requirements.

V. Threats to validity

This section describes the threats to the internal and external validity of this study. We used the Threats/Influences/Strategies map proposed by Zhou et al. [48] to develop a mitigation threats strategy.

A. Internal validity

Internal validity is the extent to which a causal conclusion based on a study is warranted. We defined a strategy based on phases to structure a research process able to mitigating the following threats:

- *Incompleteness of search* To mitigate this threat, in Phase I we used highly referenced electronic databases to find proposed microservices architectural patterns; in Phase II we used qualitative and quantitative criteria to select architectural patterns; and in Phase III we used automatic tools (built by us) and semi-automatic (manual revision of configuration files) for frameworks identification.
- *Bias on open source project selection* To mitigate this threat, we opted for GitHub for this study, since it is a leading source code hosting platform, and its search engine allows to create strings search to refine the selection of projects. We used guidelines taken from other studies [15] and criteria to obtain a group of projects that were suitable for our research.
- *Bias on data extraction* The patterns information was discussed in meetings by researchers from our research team to identify other viewpoints that complemented our conclusions.
- *Bias on patterns categories and quality attribute selection* Brainstorming sessions, meetings with experts in academic conferences and the attendance at a Ph.D. summer school[9] allow us to mitigate this threat. In those gatherings, we discuss which QA's surround microservices environments and the relationship between architectural patterns and QA's with practitioners and experts. Finally,

[9]iSAPS - http://www.lorentzcenter.nl/lc/web/2017/875/info.php3?wsid=875

based on the feedback received, we refine the contents of Figure 4 and Table II.

B. External validity

External validity refers to how generalizable the results of the research are. To reduce threats to it, we have left the research package in https://goo.gl/QggVn6, including all artifacts necessary to replicate the study in other environments. We have also made available the dataset of open source projects, architectural patterns, and references.

VI. Related work

We focused the related work on those papers where architectural patterns for microservices are discussed.

Francesco et al. [49] reported a systematic mapping study to identify, classify, and evaluate the current state of the art on architecting microservices from publication trends, research focus, and potential for industrial adoption, defining a classification framework for categorizing the research on architecting microservices.

Taibi et al. [19] conducted a systematic mapping study to identify reported usage of microservices, and from these use cases extracted common patterns and principles. Their two key contributions are (1) identification of several agreed microservice architectural patterns that seem widely adopted and reported in the case studies identified, and (2) a catalog, in a standard format, of the advantages, disadvantages, and lessons learned for each pattern from the case studies. The same results are discussed in [35].

Alshuqayran et al. [50] presented a systematic mapping study of microservices architectures and their implementation, focusing on identifying architectural challenges, architectural diagrams/views, and quality attributes related to microservice systems.

We reached similar conclusions regarding the architectural patterns that contextualize microservices and the QA's that they support, but to the best of our knowledge, none of these studies has explored the use of architecture patterns in open source microservices-based systems, be it to complement/validate their results or to map their relationship with frameworks.

VII. Conclusions

This article presented a study of the actual use of microservices architectural patterns in open source microservices-based projects. The study was conducted through a research process with three steps: (1) gathering evidence on architectural patterns reported in academic and industrial sources, (2) filtering patterns through criteria, and (3) analyzing the use of these patterns in open source projects, to assess adoption of microservices patterns. We identified seventeen patterns that fit these criteria. In parallel, we associated the reported patterns with their quality attributes.

We wondered if the found architectural patterns are the entirely new knowledge that is being generated due to microservices adoption. We searched for evidence in academic sources, and from our initial set of seventeen patterns, we determined that just nine have been explicitly proposed for microservices architectures.

Our surmises that the growing use of microservices to build systems is increasing software architectural knowledge, some wholly new and "imported" from related fields. Individually, open source projects do combine "new" microservices architectural patterns with some already proposed for SOA systems.

There is no doubt that addressing additional quality attributes, like scalability and observability, will pose new problems and bring up new recurring solutions fit to be cataloged as microservice architectural patterns.

Future work will expand our previously proposed pattern language for scalable microservices [51] to include this article's findings and other quality attributes, and will try to identify recurrent design-time decisions in microservices architectures.

Acknowledgments

This work has been partially supported by CONICYT-PCHA (Doctorado Nacional/2016-21161005), CONICYT PIA (Basal FB0821 CCTVal), and by UTFSM DGIIP.

References

[1] J. Lewis and M. Fowler, *Microservices-A definition of this new architectural term*, 2014. https://martinfowler.com/articles/microservices.html.

[2] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] M. Kircher and J. Prashant, *Pattern-Oriented Software Architecture. Patterns for Resource Management*. West Sussex, England: Wiley series in software design patterns, 2004.

[5] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*, vol. 5. Springer Science and Business Media., 2012.

[6] R. E. Johnson, "Frameworks = (components + patterns)," *Commun. ACM*, vol. 40, no. 10, pp. 39–42, 1997.

[7] F. Osses, G. Márquez, and H. Astudillo, "Poster: Exploration of academic and industrial evidence about architectural tactics and patterns in microservices," *ICSE'18 Companion: 40th International Conference on Software Engineering Companion*, 2018. doi:{https://doi.org/10.1145/3183440.3194958}.

[8] G. Márquez, F. Osses, and H. Astudillo, "Review of architectural patterns and tactics for microservices in academic and industrial literature," *XXI Ibero-American Conference on Software Engineering*, 2018.

[9] T. Erl, *SOA: principles of service design*. Upper Saddle River: Prentice Hall, 2008.

[10] S. J. Fowler, *Production-ready Microservices: Building Standardized Systems Across an Engineering Organization*. O'Reilly Media, Inc., 2016.

[11] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (3rd Edition)*. SEI Series in Software Engineering, 2013.

[12] V. R. Basili, "Software modeling and measurement: the goal/question/metric paradigm.," 1992.

[13] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices migration patterns," *Technical Report No. 1, TR- SUT-CE-ASE-2015-01 Automated Software Engineering Group Sharif University of Technology*, 2015.

[14] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–6, 2016.

[15] M. Aderaldo, C. Mendonça, C. Pahl, and J. Pooyan, "Benchmark requirements for microservices architecture research," *In Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE'17)*, pp. 8–13, 2017.

[16] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "Iot design patterns: Computational constructs to design, build and engineer edge applications," *IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 277–282, 2016.

[17] K. Brown and B. Woolf, "Implementation patterns of microservices architectures," *Conference on Pattern Languages of Programs (PLoP)*, pp. 7:1–7:35, 2016.

[18] A. Messina, R. Rizzo, P. Storniolo, and A. Urso, "A simplified database pattern for the microservice architecture," *The Eighth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA)*, 2016.

[19] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: a systematic mapping study," *Proc. 8th Int. Conf. Cloud Computing and Services Science*, 2018.

[20] C. Richardson, *A pattern language for microservices*, 2017. http://microservices.io/patterns/.

[21] N. Shalom, "Building large scale services with microservices," *Cloudify*, 2017.

[22] MuleSoft, "The top six microservices patterns. how to choose the right architecture for your organization," 2017.

[23] A. Gupta, *Microservice Design Patterns*, 2015. http://blog.arungupta.me/microservice-design-patterns/.

[24] P. Calçado, "Patterns of microservices architecture," 2017. http://philcalcado.com/microservices-patterns.html.

[25] L. Majerowicz, *Integration Patterns for Microservices Architectures: The good, the bad and the ugly*, 2017. http://hecodes.com/2017/04/integration-patterns-microservices-architectures-good-bad-ugly/.

[26] R. Dhall, *Performance Patterns in Microservices based Integrations*, 2016. https://www.computer.org/web/the-clear-cloud/content?g=7477973\&type=blogpost\&urlTitle=performance-patterns-in-microservices-based-integrations.

[27] M. Wasson, *Design patterns for microservices*, 2017. https://azure.microsoft.com/en-us/blog/design-patterns-for-microservices/.

[28] M. Churchman, *Top Patterns for Building a Successful Microservices Architecture*, 2017. https://www.sumologic.com/blog/devops/top-patterns-building-successful-microservices-architecture/.

[29] Arcitura, 2018. http://microservicepatterns.org/.

[30] P. Velardi, A. Cucchiarelli, and M. Petit, "A taxonomy learning method and its application to characterize a scientific web community," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 2, 2007.

[31] I. O. for Standardization, 2016. https://www.iso.org/standard/66435.html.

[32] R. Kalman, "On the general theory of control systems," *IRE Transactions on Automatic Control*, vol. 4, no. 3, pp. 481–492, 1959.

[33] L. Bass, I. Weber, and L. Zhu, "Devops: A software architect's perspective," *Addison-Wesley Professional*, 2015.

[34] U. Zdun, C. Hentrich, and W. M. Van Der Aalst, "A survey of patterns for service-oriented architectures," *International journal of Internet protocol technology*, vol. 1, no. 3, pp. 132–143, 2006.

[35] F. Montesi and J. Weber, *Circuit Breakers, Discovery, and API Gateways in Microservices*, 2016. http://arxiv.org/abs/1609.05830.

[36] M. Stal, "Using architectural patterns and blueprints for service-oriented architecture," *IEEE Software*, vol. 23, no. 2, pp. 54–61, 2006.

[37] M. H. Syed and E. B. Fernandez, "The software container pattern," *Proceedings of the 22Nd Conference on Pattern Languages of Programs*, pp. 15:1–15:7, 2015.

[38] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud computing patterns: Fundamentals to design, build, and manage cloud applications," *Springer*, 2014.

[39] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 25–30, 2015.

[40] M. Keen, A. Acharya, S. Bishop, A. Hopkins, C. Milinski, S.and Nott, and P. Verschueren, "Patterns: Implementing an soa using an enterprise service bus," *IBM Redbooks*, 2004.

[41] Q. A. Liang, J. y. Chung, S. Miller, and Y. Ouyang, "Service pattern discovery of web service mining in web service registry-repository," *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)*, pp. 286–293, 2006.

[42] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

[43] R. T. Mannava V., "A novel event based autonomic design pattern for management of webservices," *Advances in Computing and Information Technology. ACITY 2011. Communications in Computer and Information Science*, vol. 198, 2011.

[44] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[45] IBM, *SOA Healthchecks*. http://www-01.ibm.com/software/solutions/soa/healthcheck.html.

[46] M. Richards, *Microservices vs. service-oriented architecture*. O'Reilly Media, 2015.

[47] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on soa and microservices," *2016 4th International Conference on Enterprise Systems (ES)*, pp. 60–67, 2017.

[48] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," *23rd Asia-Pacific Software Engineering Conference (APSEC)*, pp. 153–160, 2016.

[49] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 21–30, 2017.

[50] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51, 2016.

[51] G. Márquez, M. M. Villegas, and H. Astudillo, "A pattern language for scalable microservices-based systems," *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pp. 24:1–24:7, 2018.