The Journal of Systems and Software 146 (2018) 215–232

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

The pains and gains of microservices: A Systematic grey literature review

Jacopo Soldani[a,*], Damian Andrew Tamburri[b], Willem-Jan Van Den Heuvel[b]

[a] Dipartimento di Informatica, Universitá di Pisa, Pisa, Italy
[b] Jheronimus Academy of Data Science, TU/e, Universiteit van Tilburg, NL,the Netherlands

ARTICLE INFO

Article history:
Received 7 May 2018
Revised 17 September 2018
Accepted 26 September 2018
Available online 27 September 2018

Keywords:
Microservices
Microservices design
Microservices development
Microservices operation
Systematic grey literature review
Systematic literature review

ABSTRACT

The design, development, and operation of microservices are picking up more and more momentum in the IT industry. At the same time, academic work on the topic is at an early stage, and still on the way to distilling the actual "Pains & Gains" of microservices as an architectural style. Having witnessed this gap, we set forth to systematically analyze the industrial *grey* literature on microservices, to identify the technical/operational pains and gains of the microservice-based architectural style. We conclude by discussing research directions stemming out from our analysis.

1. Introduction

Microservices are gaining more and more momentum in enterprise IT Thönes (2015). Prominent examples are Amazon, Netflix, Spotify and Twitter, which are already delivering their core business through microservice-based solutions.

Microservices were first introduced in 2014 by Lewis and Fowler in their famous blog post [S22]. They define an architectural style for developing applications as suites of small and independent (micro)services. Each microservice is built around a business capability, it runs in its own process, and it communicates with the other microservices in an application through lightweight mechanisms (e.g., HTTP APIs). To some extent, the microservice architectural style can be seen as a natural extension of SOA (*Service-Oriented Architecture*), which puts emphasis on the independent/self management of services, and on their lightweight nature (Zimmermann, 2017). Microservices are loosely coupled architectural elements that can be independently deployed by fully automated machinery. Such independent deployment is often achieved by exploiting lightweight, container-based platforms (Pahl et al., 2017). As such, microservices offer by design a scalable solution for service-oriented operation (Fountain, 2003).

There exist secondary studies analyzing research trends on microservices in the academia, all concluding that academic research on microservices is still in its early stage. At the same time, companies are working day-by-day on the design, development and operation of microservices, as also witnessed by the huge amount of grey literature on the topic. We can hence observe a sort of gap between academic research and industry practices, especially in an effort to figure out which are the technical/operational "pains" and "gains" of microservices.

In this paper, we try to fill the above mentioned gap by complementing the academic state-of-the-art on microservices with a systematic analysis of the industrial *grey* literature on the topic (as recommended by Garousi et al. (2016)). More precisely, we aim at identifying, taxonomically classifying, and systematically comparing the existing grey literature on pains and gains of microservices, from their design to their development. We conducted a systematic review of 51 selected industrial studies, published since 2014 (when the microservice-based architectural style was first defined by Lewis and Fowler) till the very end of 2017. Following the guidelines by Garousi et al. (2017) (and those by Petersen et al. (2008)), we first excerpted two taxonomies from the selected industrial studies, one for the pains of microservices, and one for their gains. We then exploited such taxonomies to classify and compare the selected industrial studies, in order to distill the actual recognition of pains and gains of microservices by the IT industry.

The results of our study show that the pains of microservices are mainly due to the intrinsic complexity of microservice-based applications. For instance, while designing a microservice-based application, the primary pains are determining the "right" granu-

* Corresponding author.
E-mail address: soldani@di.unipi.it (J. Soldani).

https://doi.org/10.1016/j.jss.2018.09.082
0164-1212/

larity of its microservices and the design of its security policies. Managing distributed storage and application testing are instead the primary pains at development time. At operation time, the primary pain of microservices is their consumption of network and computing resources, which is recognized as much higher with respect to that of other architectural styles (e.g., monoliths, SOA).

At the same time, microservices bring various, widely recognized gains. At design time, the primary gains of microservices those related to the exploitation of design patterns, which are widely recognized to help mitigating/solving some pains of microservices. Development emerges by far as the stage gaining most from microservices, as different microservices can be independently developed by different developers, who have all the freedom in choosing the technology stacks and data stores that best fit the microservices they are developing. Operation is also gaining from microservices, which can be independently deployed and scaled whenever needed.

Our systematic analysis of grey literature on microservices can provide benefits to both researchers and practitioners. A systematic presentation of the industrial state-of-practice on microservices provides a body of knowledge to develop new theories and solutions, to analyze and experiment research implications, and to establish future research dimensions. At the same time, it can help practitioners to understand the currently recognized pains and gains of microservices, and their maturity. This can have practical value for practitioners, e.g., as a starting point for microservices experimentation or as a guideline for day-by-day work with microservices.

The rest of this article is organized as follows. Section 2 discusses related work. Section 3 illustrates the methodology, research questions and scope of our systematic study. Section 4 presents the results of our study. Section 5 discusses results and limitations of our study. Section 6 concludes this article, by discussing findings, implications and directions for future work.

2. Related work

Besides the seminal paper by Sill (2016), only a handful of other efforts provide empirical evidence on effective/efficient exploitation of microservices. For instance, Balalaie et al. (2016) illustrate experiences and lessons learned from the migration and architectural refactoring of a commercial application to microservices. Hassan and Bahsoon (2016) provide a first investigation on design trade-offs, when trying to partition applications in independent microservices. Haselböck et al. (2017) offer an overview of the microservice design space, by proposing a strategic model for instrumenting microservice design decisions. Erl et al. (2017) and (Zimmermann, 2017) instantiate SOA notions in the world of microservices. A key observation turning out from above mentioned research efforts (and from closely related efforts, as well) is that academic research on microservices — from their design to their development and operation — is still at an early stage.

The early stage of academic research on microservices also impacts on surveys on the state-of-the-art or state-of-practice on microservices. State-of-the-art surveys focus on classifying and comparing the existing research body on microservices (including some non peer-reviewed content), in order to determine the maturity of research on microservices, and on establishing the research challenges emerging from the analyzed literature. Concrete examples are the systematic analyses provided by Di Francesco et al. (2017), by Pahl and Jamshidi (2016) and by Vural et al. (2017). They all conclude that academic research on microservices is still immature, by also outlining various research directions.

State-of-practice analyses are instead yet to be provided, as (to the best of our knowledge) the only available efforts are those by Ghofrani and Lübke (2018) and by Taibi and Lenarduzzi (2018). Ghofrani and Lübke (2018) provide a preliminary analysis on state-of-practice on microservices, based on an online survey (with 3 questions, answered by 25 interviewed practitioners). The results presented by Ghofrani and Lübke (2018) provide a high-level overview of industry-oriented challenges on microservices, without delving into the details of their actual pains and the gains. Our study instead aims at providing a deeper analysis on which are the technical/operational pains and gains of microservices recognized by industrial researchers and practitioners working day-by-day with microservices.

Taibi and Lenarduzzi (2018) illustrate 11 microservice-specific bad practices, resulting from interviews of developers experienced with microservice-based systems. Such bad practices are reflected by some of the pains that we identify in our study, whose objective is however different from that by Taibi and Lenarduzzi. Instead of trying to identify bad practices by conducting interviews, our aim is to elicit the main technical/operational pains and gains of microservices by systematically analysing the grey literature on the topic.

In summary, we perceive a sort of gap between the industrial understanding and state-of-practice on microservices and the related academic literature on the topic. The key motivation of our study is to try to fill this gap, by eliciting the current state-of-practice on microservices through a systematic grey literature review.

3. Research design

A major intrinsic difficulty of our study is our necessary reliance over what is called *grey* literature (Garousi et al., 2016), intended as materials and research produced by organizations outside of the traditional commercial or academic publishing and distribution channels. Common grey literature publication types include reports (annual, research, technical, project, etc.), working papers, government documents, white papers, videos and evaluations. On the one hand, the use of grey literature is risky since there is often little or no scientific factual representation of data or analyses presented in grey literature itself (Farace and Schöpfel, 2010). On the other hand, a growing interest around using grey literature for software engineering practitioner benefit as well as combining it to determine the state of the art and practice around a topic is gaining a considerable interest in many fields (Farace and Schöpfel, 2010; Stempfhuber et al., 2008), including software engineering (Garousi et al., 2016).

For the scope of this study, and in an effort to maximize its validity, we followed a systematic approach based on that by Petersen et al. (2008) for conducting systematic literature reviews in software engineering. More precisely, following the guidelines by Garousi et al. (2017), we varied the standard approach by Petersen et al. (2008) as follows:

- We exploited general web search engines for searching for grey literature (instead of indexing databases).
- We employed “saturation” as stopping criteria (i.e., we stopped our search when no new relevant results/concepts were emerging from search results).
- We combined inclusion/exclusion criteria with quality assessment control factors.
- We fixed the type of relevant grey literature to blog post, whitepapers, industrial magazines and videos.

We hereafter outline the systematic approach we followed, by starting from problem definition and by also describing the triangulation and inter-rater reliability assessment trials we ran to enforce the validity of our findings.

(microservice* ∨ micro-service*) ∧ (pain* ∨ disadvantage* ∨ cons* ∨ drawback* ∨ pitfall* ∨ antipattern* ∨ code smell*)
(a)

(microservice* ∨ micro-service*) ∧ (gain* ∨ advantage* ∨ pros* ∨ benefit* ∨ ease* ∨ pattern* ∨ best practice*)
(b)

Fig. 1. Strings employed for searching the technical/operational (a) pains and (b) gains of microservices. In both cases, "* " matches lexically related terms (e.g., plurals, verb conjugations).

3.1. Research problem definition and research questions

The problem we aim to address is twofold. On the one hand, we aim to elicit pains, gains, best practices, and fallacies over using microservices in practice, by looking at the reported failure and success experiments directly from the industrial literature. This information can have clear practical value since other practitioners can use the results captured in this manuscript as a starting point for microservices experimentation in their own practice.

On the other hand, we aim to elicit the gaps, limitations, and practical directions experimented up to this point. This can help researchers investigating microservices, who can start from our findings to proceed in their research from a basis of synergy with industry itself (as we believe that the results captured in this manuscript are one such basis of synergy).

Stemming from the above research problems, we formulate the following research questions:

- Q_1 *How much evidence of microservices experimentation from industry is available online?* We aim to provide a high-level overview of the involvement of industrial researchers and practitioners in microservices, by analyzing how much industrial studies have been published in the recent past (i.e., since the beginning of 2014 till the end of 2017), as well the types of contributions, the sources of publications, and their contents.
- Q_2 *What are the technical and operational "pains" of microservices?* We aim to elicit the technical and operational difficulties intrinsic to the design, development and operation of microservice-based applications, which industrial researchers/practitioners have been experimenting in the recent past.
- Q_3 *What are the technical and operational "gains" of microservices?* We aim to elicit the technical and operational benefits that are provided by microservices, which industrial researchers/practitioners have been appreciating during the design, development and operation of microservice-based applications.

3.2. Search for industrial studies

As recommended by Garousi et al. (2017), industrial studies can be identified by exploiting search strings on search engines (e.g., Google, Bing). Following the guidelines provided by Petersen et al. (2008), we identified the search string by structuring them guided by our research questions. More precisely, we defined the search strings based on the PICO terms of our question (Kitchenham and Charters, 2007), by exploiting only the terms *Population* and *Intervention*. The keywords were taken from each aspect of a research question. Differently from Petersen et al. (2008), we did not restrict our focus to specific outcomes or experimental designs in our study, as we wished to have a broader overview of the industrial state-of-practice on microservices. By restricting ourselves to certain types of studies, we could have obtained a biased/incomplete analysis, as some sub-topics might have been over-/under-represented for certain types of study. The above explained difference is also reflected by the strings we employed for searching for the pains of microservices (Fig. 1.(a)) and for searching for the gains of microservices (Fig. 1.(b)).

We exploited the above indicated search strings to look for industrial studies (e.g., blog posts, whitepapers, industry-oriented magazines, videos) that were published since the beginning of 2014 (when microservices were first proposed by Lewis and Fowler) until the end of 2017. The search engines we employed are Google (primary), Bing, Duck Duck Go, Yahoo! and Webopedia. Since engines look for search strings over the whole pages they index, our search resulted in a high number of irrelevant studies, which were further refined with a secondary search and manual screening, based on the inclusion/exclusion criteria and control factors discussed in the following section.

3.3. Sample selection & control factors

Table 1 outlines the inclusion and exclusion criteria adopted in our sample selection. The inclusion criteria ($i_1 - i_4$) were designed to focus explicitly on the kind of practical grey literature that identifies the design, development and operation principles currently deemed efficient in industry based on their direct application. At the same time, the exclusion criteria permit disqualifying studies that do not offer the necessary design/implementation details (e_1), that refer to nonfactual or unquantifiable evidence (e_2 and e_3), and that do not discuss the limitations and practical impact for the proposed solutions or outlined issues (e_4 and e_5). An industrial study is to be selected if it satisfies *all* the inclusion criteria, while it is to be excluded if it satisfies at least one of the exclusion criteria.

In addition to the inclusion/exclusion criteria in Table 1, to ensure the quality of the selected grey literature, we selected only those industrial studies that were satisfying four additional control factors:

- *Practical experience* → A study is to be selected only if it is written by practitioners with 5+ experience in service-oriented design, development and operation, or if it refers to established microservices solutions with 2+ years of operation.
- *Industrial case-study* → A study is to be selected only if it refers to at least 1 industrial case-study where a quantifiable number of microservices are operated.
- *Heterogeneity* → The selected studies reflect at least 5 top industrial domains and markets where microservices were successfully applied.
- *Implementation quantity* → The selected studies refer to/show implementation details for the benefits and pitfall they discuss, so that other researchers and practitioners can use them in action.

The control factors *practical experience, industrial case-study* and *implementation quantity* were checked against each single study, whereas the control factor *heterogeneity* was checked against the overall set of industrial studies to be selected[1].

[1] The information for checking the inclusion/exclusion criteria and the control factors was excerpted from the selected industrial studies themselves, from the authors' LinkedIn pages, and from the website/portfolio of the company where each selected industrial study was carried out.

Table 1
Inclusion and exclusion criteria for sample selection.

Inclusion	i_1)	The study discusses the industrial application of microservices.
	i_2)	The study discusses the benefits or shortcomings of microservice design, development or operation.
	i_3)	The study reports on direct experiences, opinions or practices on microservices by educated practitioners.
	i_4)	The study refers to a practical case-study of design, development or operation of microservices.
Exclusion	e_1)	The study does not offer details on design or implementation of microservices.
	e_2)	The study is not referred to industrial cases or other factual evidence.
	e_3)	The benefits or pitfalls of microservices are not justified/quantified by the study.
	e_4)	The study does not provide scope and limitations of proposed solutions/patterns.
	e_5)	The study does not offer evidence of a practitioner perspective.

At the end of our screening, 51 industrial studies were selected based on the inclusion/exclusion criteria and on the additional control factors. The complete list of selected industrial studies is provided in Appendix A.

3.4. Analysis & inter-Rater reliability assessment

To attain the findings discussed in Section 4 we adopted thematic coding (Buder and Creß, 2003). The selected sample of articles were subject to annotation and labeling with the goal of identifying themes emerging from the analyzed text. This process of analysis was executed in parallel over two 50% splits of the entire dataset, to ensure avoidance of observer bias. The coders of the two splits (i.e., the first two authors of this study) were then inverted and an inter-rater evaluation was enacted between the two emerging lists of themes. To evaluate inter-rater reliability, we adopted the widely known and adopted Krippendorff α coefficient (or Kα), which measures the agreement between two ordered lists of codes applied as part of content analysis (Krippendorff, 2004). As part of our evaluation, Kα was applied twice.

We first applied the evaluation coefficient to measure the agreement between the two emerging lists of codes by the two independent observers who individually coded 100% of the dataset in 2 rounds. The result of applying Kα to measure the agreement between these two lists amounted to 0.76, slightly lower than the typical reference score of 0.80 (i.e., 80% agreement). A discussion and analysis of disagreement points revealed misalignment between the depth of the coding strategy, which was subsequently addressed. This modification brought the agreement between the emerging lists of codes to K$\alpha = 0.81$.

Kα was then applied again to triangulate the 0.81 score for the final list of themes with its identical counterpart, coded by the third author of this paper, who re-coded the entire dataset with the final coding strategy. The agreement between the final list A (obtained by the first 2 authors) and a final list B (obtained by the third author) was evaluated to K$\alpha = 0.92$.

3.5. Replication package

To encourage repeatability of our approach and verifiability of our findings, results, and discussion points, a replication package is freely available on GitHub[2] The replication package contains the intermediary artifacts and the final results of our study. The selected industrial studies can instead be freely accessed online, by following the bibliographic information reported in Appendix A.

4. Results

In this section we analyze the selected grey literature under three different perspectives, aligned with the research questions Q_1, Q_2 and Q_3. We first provide a general overview of the selected industrial studies (Section 4.1). We then analyze in detail the pains and gains of microservice-based applications emerging from the selected grey literature (Sections 4.2 and 4.3, respectively). We finally provide a summary of the findings of our analysis (Section 4.4).

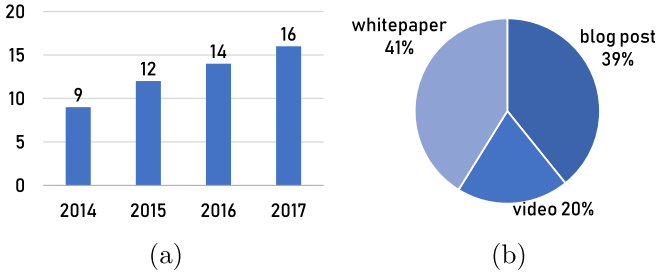


Fig. 2. Distribution of selected industrial studies (a) by year and (b) by contribution type.

4.1. Overview of the selected grey literature

In order to address Q_1, we hereby provide an overview of the selected studies [S1]-[S51] based on generic aspects, i.e., when and where they were published, in which format, and which was their content. Table 2 shows how we classified [S1]-[S51] based on their publication year, their source (i.e., *ad-hoc blog* maintained by experts, *community* of practitioners, *company* website, industry-oriented *magazine*, or industrial *summit*), the type of contribution (i.e., *blog post*, industrial *whitepaper*, or *video*), and their contents (description of *patterns* or *best practices, experience report*, or discussion of *pros/cons* of microservices). The table also shows the companies and industrial settings (i.e., *product, case study* or *consulting*) where the selected industrial studies were carried out.

Fig. 2 illustrates the distribution of the selected studies by year and type of contribution. While the distribution of selected studies by contribution type does not provide any particular insight, their distribution by year shows that the amount of industrial contributions on microservices is yearly increasing since 2014. This signals that microservices are steadily gaining attention in the IT industry since 2014, when James Lewis and Martin Fowler gave rise to the microservice-based architectural style with their famous blog post [S22].

Fig. 3 instead illustrates the distribution of the selected studies by source and contents. We can observe that the selected studies are mainly taken from community portals and company websites, where practitioners and researchers are mainly sharing their best practices, describing microservices-related patterns or just discussing advantages and disadvantages of adopting microservice-based solutions. The above clearly witnesses the interests of the IT industry in microservices, in their pains and gains, and in best practices and patterns that can be exploited to better leverage of microservices.

The above is even more evident if we restrict the visualization to the selected studies that were published last year (Fig. 4), where

[2] https://www.tinyurl.com/microservices-study.

Table 2
Classification of the selected industrial studies based on year of publication, type of contribution, source, contents, reference company and industrial setting.

Study	Year	Contribution type	Source	Content	Company	Industrial setting
[S1]	2015	Whitepaper	Community	Patterns	Asurion	Case study
[S2]	2014	Whitepaper	Community	Pros/cons	C4Media	Consulting
[S3]	2015	Blog post	Company	Pros/cons	IGATE	Case study
[S4]	2016	Whitepaper	Magazine	Exp. report	Comparethemarket	Case [S5]
[S5]	2017	Whitepaper	Company	Pros/cons	Chakray	Consulting
[S6]	2017	Video	Summit	Pros/cons	nearForm	Consulting
[S7]	2016	Video	Summit	Exp. report	SimplicityItself	Case study
[S8]	2015	Video	Company	Pros/cons	Dev2 Technologies	Case study
[S9]	2016	Whitepaper	Community	Patterns	HCL Technologies	Case study
[S10]	2016	Blog post	Ad-hoc blog	Best practices	JDriven	Case study
[S11]	2016	Whitepaper	Magazine	Patterns	Techworld	Consulting
[S12]	2017	Video	Summit	Exp. report	Netflix	Case study
[S13]	2015	Blog post	Company	Pros/cons	ThoughtWorks	Consulting
[S14]	2016	Whitepaper	Community	Best practices	Navica	Consulting
[S15]	2016	Whitepaper	Community	Patterns	DynaTrace	Case study
[S16]	2015	Blog post	Ad-hoc blog	Pros/cons	Spreadshirt	Case study
[S17]	2017	Blog post	Company	Best practices	LogicRoom	Case study
[S18]	2014	Whitepaper	Community	Best practices	Microsoft	Case study
[S19]	2016	Whitepaper	Community	Patterns	WSO2	Case study
[S20]	2017	Blog post	Community	Pros/cons	SAP	Case study
[S21]	2015	Blog post	Ad-hoc blog	Best practices	Tyro Payments	Case study
[S22]	2014	bBog post	Company	Patterns	ThoughtWorks	Consulting
[S23]	2015	Whitepaper	Community	Best practices	Pivotal	Product
[S24]	2015	Blog post	Company	Pros/cons	MadeTech	Consulting
[S25]	2017	Whitepaper	Magazine	Best practices	Nginx	Product
[S26]	2015	Blog post	Company	Exp. report	Netflix	Product
[S27]	2017	Whitepaper	Company	Patterns	Micro	Consulting
[S28]	2017	Video	Summit	Exp. report	Speedment	Case study
[S29]	2015	Blog post	Company	Best practices	Smartbear	Consulting
[S30]	2015	Video	Company	Pros/cons	Sam Newman	Consulting
[S31]	2017	Whitepaper	Community	Pros/cons	Apiumhub	Consulting
[S32]	2017	Blog post	Company	Patterns	Valtech	Consulting
[S33]	2016	Whitepaper	Community	Best practices	Avi Networks	Consulting
[S34]	2015	Blog post	Company	Best practices	Neotys	Consulting
[S35]	2014	Blog post	Ad-hoc blog	Patterns	Eventuate	Case study
[S36]	2014	Blog post	Ad-hoc blog	Patterns	Eventuate	Case study
[S37]	2014	Blog post	Ad-hoc blog	Patterns	Eventuate	Case study
[S38]	2016	Whitepaper	Company	Best practices	Nginx	Product
[S39]	2014	Video	Summit	Patterns	Pivotal	Case study
[S40]	2017	Blog post	Company	Pros/cons	Oracle	Consulting
[S41]	2017	Video	Company	Pros/cons	ThoughtWorks	Consulting
[S42]	2017	Video	Summit	Pros/cons	ThoughtWorks	Consulting
[S43]	2016	Whitepaper	company	Best practices	FacileLogin	Case study
[S44]	2014	Video	Summit	Pros/cons	Netflix	Case study
[S45]	2017	Blog post	Company	Best practices	Coding Sans	Case study
[S46]	2017	Whitepaper	Community	Best practices	ServisBOT	Consulting
[S47]	2017	Whitepaper	Community	Pros/cons	Nirmata	Consulting
[S48]	2017	Whitepaper	company	Patterns	Microsoft	Product
[S49]	2017	Whitepaper	Company	Exp. report	Microsoft	Case study
[S50]	2014	Blog post	Community	Best practices	Contino	consulting
[S51]	2016	Blog post	Community	Pros/cons	AND Digital	Consulting

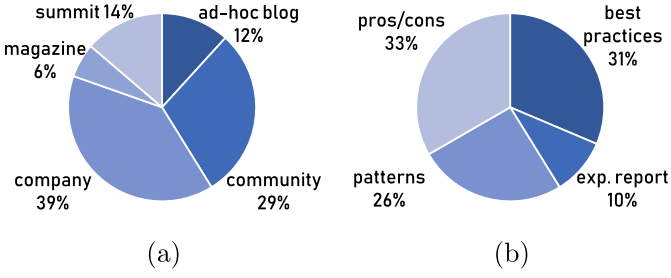


Fig. 3. Distribution of selected industrial studies (a) by source and (b) by contents.

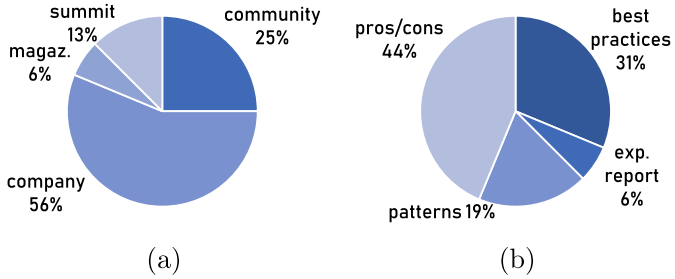


Fig. 4. Distribution of selected industrial studies (a) by source and (b) by contents (restricted to the studies published in 2017).

companies are involved in the majority of the contributions, and where the contributions are mainly focused on the pros and cons of microservices, and on the best practices and patterns that can be exploited to better leverage of microservices.

Finally, Fig. 5 plots the distribution of the selected studies by industrial setting. While the distribution over the period 2014–2017 (Fig. 5(a)) does not provide any particular insight, it is interest-

Table 3
A taxonomy for pains of microservices.

Stage	Concern	Pain
Design	Architecture	API Versioning, Communication heterogeneity, Service contracts, Service dimensioning, Size/complexity
	Security	Access control, Centralised support, CI/CD, Endpoint proliferation, Human errors, Size/complexity
Development	Microservices	Microservice separation, Overhead
	Storage	Data consistency, Distributed transactions, Heterogeneity, Query complexity
	Testing	Integration testing, Performance testing, Size/complexity
Operation	Management	Cascading failures, Operational complexity, Service coordination, Service location
	Monitoring	Logging, Problem location, Size/complexity
	Resource consumption	Compute, Network

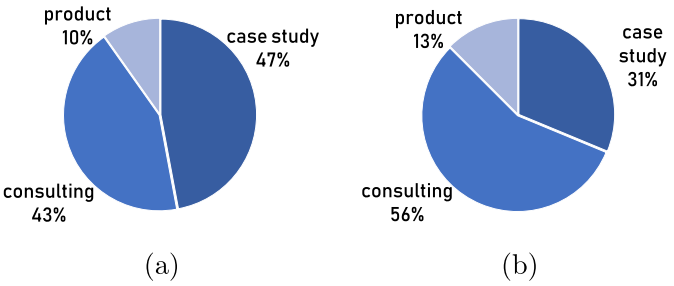


Fig. 5. Distribution of selected industrial studies by industrial setting (a) over the period 2014–2017, and (b) restricted to the studies published in 2017.

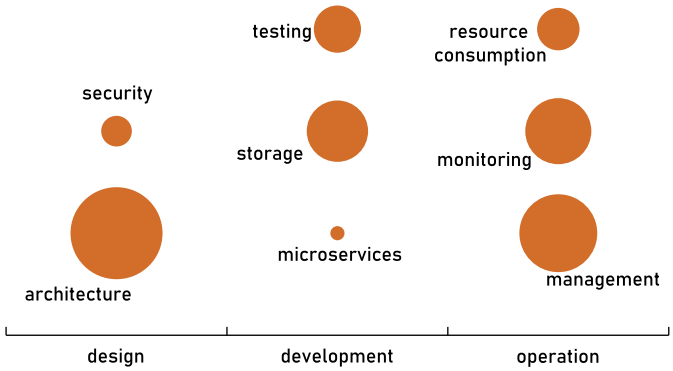


Fig. 6. Coverage of the concerns in the taxonomy for pains of microservices. The size of each bubble is directly proportional to the number of selected industrial studies discussing a pain pertaining to the corresponding concern.

ing to note that the industrial studies published in 2017 mainly focused on consulting. If we cross this data with the facts that microservices gained momentum, and that the technology around microservices has matured (Jamshidi et al., 2018), we can conclude that IT companies started focusing on increasing their consultancy portfolio on microservices.

4.2. Microservices pains

We hereby introduce a taxonomy for classifying the pains of microservices. We also show how we exploited such taxonomy to analyse current industrial perspectives on the pains of microservices.

4.2.1. A taxonomy for pains of microservices

Table 3 illustrates a taxonomy for the pains of microservices. We obtained such taxonomy by following the guidelines for conducting systematic reviews in software engineering proposed by Petersen et al. (2008):

- We first established the *stages*, by aligning Q_2 with common steps of the lifecycle of software (i.e., design, development and operation).
- We then identified the *concerns* by performing a first scan of the selected studies, which was focused on Q_2 to ensure the validity of the taxonomy.
- The concrete terms (i.e., the *pains*) were excerpted directly from the selected studies, and they were manually organized into the above mentioned sub-classes.

As we anticipated in Section 3, the obtained taxonomy undergone various iterations among the authors of this study, and it was submitted for validation to external experts from two organizations located in two different countries. This has resulted in some corrections and amendments to the first version of the taxonomy, which resulted in the taxonomy displayed in Table 3.

4.2.2. Pains in the selected industrial studies

Table 4 shows the classification of all selected industrial studies based on the taxonomy for pains of microservices introduced earlier. The table provides a first overview of the coverage of pains over the selected studies, despite (due to reasons of readability and space) it only captures the classification over the concerns listed in the taxonomy of pains[3] Such coverage is also displayed in Fig. 6, from which it is clear that the main concerns are the design of microservice-based architectures, the distributed and heterogeneous storage hampering the work of developers, and the management and monitoring of operating instances of microservice-based applications. Security, testing and resource consumption are also notable concerns, while the concrete development of each microservice is not significantly perceived as problematic.

We hereby discuss in detail the notable concerns listed above, stage by stage. We shall display the occurrences of the pains of each concern in a sorted table, and we will compare their weights[4] by exploiting %-based pie charts.

Design stage. According to Fig. 6, the main concern during the design of microservice-based applications is their architecture. This is also witnessed by the yearly coverage of design-related concerns (Fig. 7). The recognition of pains pertaining to the architecture of microservice-based application is steadily increasing since 2014, and it keeps higher with respect to that of security-related pains.

The coverage of the concrete pains due to the architecture of microservice-based applications are listed in Fig. 8, which also displays their weights in the selected industrial studies. The figure highlights what one can obviously expect, namely that industrial researchers and practitioners are mostly pained by the size and complexity of the architectures obtained when designing application according to the microservice-based architectural style.

[3] The detailed classification, displaying each occurrence of each pain, is publicly available in the replication package (see Section 3.5).

[4] We will measure the weight of a pain as the percentage of its occurrences among all occurrences of all pains pertaining to its same concern. This is analogous to what done by Pahl et al. (2017) to measure weights while classifying studies on cloud container technologies.

Table 4
Classification of the selected studies based on the concerns listed in the taxonomy for pains of microservices.

	Design		Development			Operation				Design		Development			Operation		
	Arc.	*Sec.*	*Mic.*	*Sto.*	*Tes.*	*Man.*	*Mon.*	*Res.*		*Arc.*	*Sec.*	*Mic.*	*Sto.*	*Tes.*	*Man.*	*Mon.*	*Res.*
[S1]	•					•	•		[S27]			•					
[S2]	•				•	•	•		[S28]	•			•				
[S3]	•			•	•	•			[S29]	•				•	•	•	•
[S4]	•			•		•	•		[S30]	•	•		•	•	•	•	
[S5]	•				•	•		•	[S31]		•						
[S6]					•	•	•	•	[S32]	•						•	•
[S7]					•	•			[S33]		•		•				
[S8]	•	•			•	•	•		[S34]					•		•	
[S9]	•					•	•		[S35]	•					•		
[S10]	•			•					[S36]		•		•				
[S11]	•					•			[S37]	•			•	•	•		•
[S12]				•	•	•		•	[S38]	•			•	•	•	•	
[S13]	•	•		•		•	•	•	[S39]	•		•			•	•	•
[S14]	•			•			•		[S40]	•			•		•	•	•
[S15]	•							•	[S41]	•				•	•	•	
[S16]	•	•	•	•		•	•	•	[S42]	•			•		•	•	
[S17]	•				•	•		•	[S43]		•						
[S18]	•								[S44]	•				•	•	•	•
[S19]	•	•		•		•	•		[S45]	•			•	•	•	•	•
[S20]	•			•	•	•			[S46]		•					•	
[S21]		•							[S47]	•	•			•			
[S22]	•			•			•	•	[S48]	•			•	•	•	•	•
[S23]		•		•		•	•		[S49]	•			•		•		•
[S24]	•		•			•	•		[S50]	•			•	•	•	•	
[S25]	•	•		•				•	[S51]	•		•	•		•	•	
[S26]			•	•													

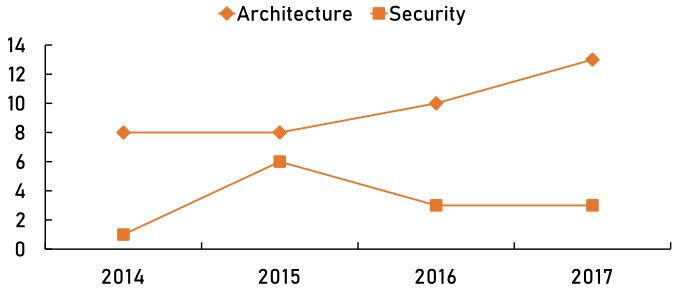
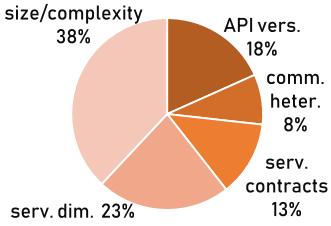


Fig. 7. Yearly evolution of the coverage of the concerns pertaining to the design stage.

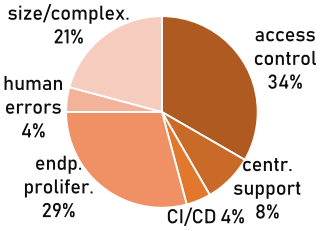


Pain	#
size/complexity	27
service dimensioning	16
API versioning	13
service contracts	9
comm. heterogeneity	6

Fig. 8. Weights and occurrences of the pains pertaining to the architecture concern during the design stage.

Service dimensioning is also significantly recognized as a pain, with 16 of the selected industrial studies recognizing difficulties in determining how much "micro" a microservice should be. All such studies agree that it is often difficult to identify the business capability/bounded context that should be assigned to each microservice, as the boundaries among different capabilities/contexts are usually not that sharp.

Another observation worth doing is about the pains due to exploiting programmable APIs to allow microservices to intercommunicate. The group composed by the pains *API versioning* and *service contracts* is also significantly covered by the selected industrial studies. Since communications among microservices is based on APIs, the API provided by a microservice becomes a sort of contract between such microservices and those consuming its API. Such a kind of contracts should not be violated, if we wish to allow microservices to continue to intercommunicate. This of course impacts on the versioning of the APIs provided by microservices, hence paining the design of microservice-based architectures.



Pain	#
access control	8
endpoint proliferation	7
size/complexity	5
centralised support	2
CI/CD	1
human errors	1

Fig. 9. Weights and occurrences of the pains pertaining to security during the design stage.

Fig. 9 instead displays the weights and occurrences of the pains related to securing a microservice-based application. Other than the intrinsic complexity of securing microservice-based architectures (obviously due to their size and complexity), the most recognized pains concerning security are due to the access control and to the endpoint proliferation. Access control is recognized as critical in 8 out of the selected industrial studies, all pointing out that the microservices in an architecture should be able to quickly and consistently ascertain the provenance and authenticity of a request. The industrial studies also agree that this is currently far from being easy, mainly because of the distributed nature of microservice-based architectures, and since their microservices evolve asynchronously. The pains due to controlling accesses in microservice-based architectures could be mitigated if designers would be provided with tools supporting a consistent, decentralized access control. However, as highlighted by [S46] and [S23], this is currently not the case, as existing tools provide a centralized support for securing microservices.

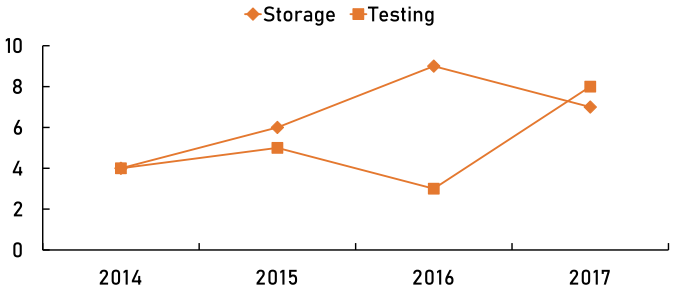
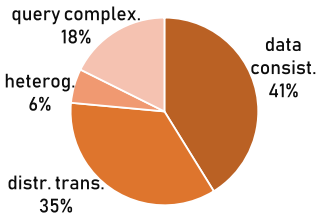
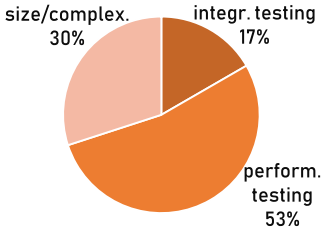


Fig. 10. Yearly evolution of the coverage of the concerns pertaining to the development stage.



Pain	#
data consistency	21
distributed transactions	18
query complexity	9
heterogeneity	3

Fig. 11. Weights and occurrences of the pains pertaining to storage during the development stage.



Pain	#
performance testing	16
size/complexity	9
integration testing	5

Fig. 12. Weights and occurrences of the pains pertaining to testing during the development stage.

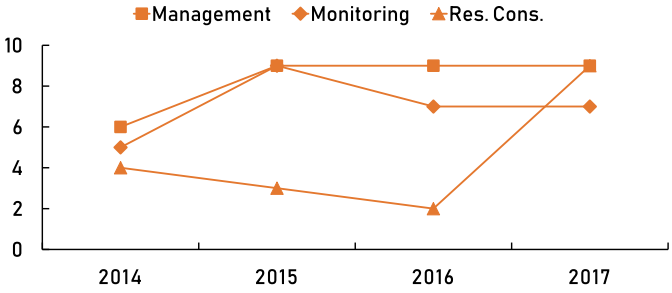


Fig. 13. Yearly evolution of the coverage of the concerns pertaining to the operation stage.

Endpoint proliferation is due to partitioning of architectures in suites of independent microservices. The attack surface in monoliths or service-oriented architectures was limited to few isolated servers. Microservice-based architectures instead require to open a lot more ports, to expose more APIs and to distribute authentication and access control (as already discussed above). The attack surface is hence much more extended, and this is the main reason why endpoint proliferation is significantly recognized as a pain by industrial researchers and practitioners.

Development stage. The selected industrial studies recognize storage and testing as significant concerns while developing microservice-based applications. The same does not hold for the development of the microservices forming an applications (Fig. 6). Given the above, we hereby focus on the concrete pains due to storage and testing concerns. The yearly coverage of such pains is displayed in Fig. 10. The recognition of both storage- and testing-related pains overall increased from 2014 to 2017, even if with a non-monotonic behaviour. It is worth observing that the recognition of storage-related pains started decreasing in 2017, while that of testing-related pains experienced an increase peak in the same year. If we cross this data with the technology waves illustrated by Jamshidi et al. (2018), we can observe that the technology for managing storage in microservice-based application is maturing, and this resulted in starting to reduce storage-related pains for developers of microservices. At the same time, the more microservices are widespreading, the more developers are pained by their testing [S6].

The pains due to storage concerns are listed in Fig. 11, together with their weights and occurrences in the selected industrial studies. From the figure it is clear how storage-related pains are caused by the distributed nature of storage in microservice-based applications. 21 of the selected industrial studies highlight that, by distributing the storage over the various microservices forming an application, it becomes a pain to ensure data consistency. They also highlight how eventual consistency may help mitigating this pain, even if it is not easy to ensure. Eventual consistency may also not be a viable option in some cases, especially when consistency must be ensured on sensitive data.

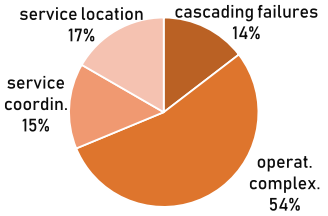
Operating with distributed data stores is another pain significantly recognized by industrial researchers and practitioners. The intrinsic complexity in implementing transactions over distributed data stores is pointed out in 18 of the selected industrial studies. 9 of them also highlight the complexity of building queries combining data stored in distributed and heterogeneous data stores.

Fig. 12 instead illustrates the weights and occurrences related to the concern of testing while developing microservice-based applications. The figure highlights how performance testing is by far the most challenging pain among those related to testing microservice-based applications. As discussed in [S34] and [S6], a microservice-based application is partitioned in a huge number of independently evolving services, and this makes it challenging to measure the performances of each of them and of the application itself. This especially holds when testers wish to measure user experience through the user interface of a microservice-based application.

Operation stage. As shown by Fig. 6, all three concerns of the operation stage are recognized as significant by the selected industrial studies, with management and monitoring being slightly more covered than resource consumption. This can be observed also in Fig. 13, which plots the yearly coverage of operation-related concerns in the period 2014–2017. The recognition of pains related to the management and monitoring increased in 2016 (if compared with that in 2014), while it kept stable from 2016 on. The recognition of pains related to resource consumption followed the opposite behaviour, as it decreased from 2014 to 2016, by then experiencing an increase peak from 2016 to 2017. The latter means that resource consumption became a first-class concern for the industrial researchers and practitioners authoring the selectedindustrial studies, and this may be due to the widespread and day-by-day work with microservice-based applications and tools (Jamshidi et al., 2018).

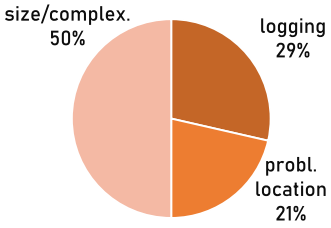
The concrete pains due to the management of microservice-based applications, their weights and occurrences are displayed in Fig. 14. The figure highlights how the intrinsic complexity of operating applications composed by distributed and heterogeneous microservices is the main challenge in the scope of managing microservice-based applications.

The above mentioned complexity is partly explained by the three other pains significantly discussed by the industrial researchers and practitioners authoring the selected industrial stud-



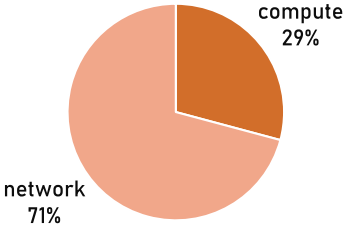
Pain	#
operational complexity	26
service location	8
cascading failures	7
service coordination	7

Fig. 14. Weights and occurrences of the pains pertaining to management during the operation stage.



Pain	#
size/complexity	21
logging	12
problem location	9

Fig. 15. Weights and occurrences of the pains pertaining to monitoring during the operation stage.



Pain	#
network	17
compute	7

Fig. 16. Weights and occurrences of the pains pertaining to resource consumption during the operation stage.

ies. The partitioning of an application in microservices can change over time, as well as the number of replicas of a microservice and their actual locations (i.e., the hosts where they are deployed, and the ports on which they are listening). This makes it challenging to locate and coordinate the microservices forming an application. Additionally, if not properly isolated, a failure in a microservices may cause the failure of another, and so on. Handling cascading failure is hence another factor hampering the operation of microservice-based applications.

Fig. 15 displays the weights and occurrences of the pains pertaining to monitoring microservice-based applications. Similarly to the case of management, the primary challenge is the intrinsic complexity of monitoring an application composed by a huge number dynamically evolving, heterogeneous microservices.

According to the industrial researchers and practitioners authoring the selected industrial studies, such a complexity is partly due to other two significantly discussed pains, i.e., *logging* and *problem location*. 12 of the selected industrial studies point out the pains due to handling a huge number of distributed logs, one for each microservice forming an application. This, along with the distributed nature of microservice-based application, makes it hard to identify the source of an issue in an application, as discussed in 9 of the selected industrial studies.

Fig. 16 illustrates the weights of the pains related to the resources consumed by a microservice-based application, during its operation stage. An increase in resource consumption (with respect to monolithic/service-oriented applications) is significantly recognized by the selected industrial studies. 7 out of the selected industrial studies highlight how the higher number of running services requires an increased amount of runtime environments,

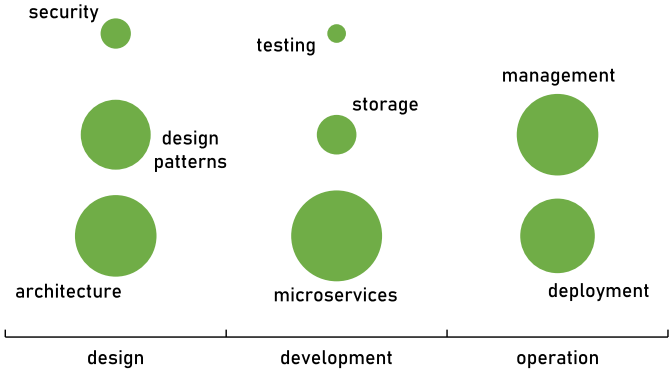


Fig. 17. Coverage of the concerns in the taxonomy for gains of microservices. The size of each bubble is directly proportional to the number of selected industrial studies discussing a gain pertaining to the corresponding concern.

which in turn results in an increased consumption of compute resources.

The increased consumption of network resources emerges to be by far more challenging. 17 out of the selected industrial studies highlight that, since the microservices in an application intercommunicate through remote API invocations, applications generate a much higher network traffic with respect to monoliths (where modules interact through in-memory calls) or service-based applications (composed by a lower number of services, hence reducing the amount of remote API invocations).

4.3. Microservices gains

In this section, we first present a taxonomy for classifying the gains of microservices, and we then illustrate how we exploited it to analyse current perpectives of industrial researchers and practitioners on the gains of microservices.

4.3.1. A taxonomy for gains of microservices

We hereby present a taxonomy for classifying gains of microservices (Table 5). It can be helpful to analyse the current state of the art on microservices in the IT industry, as well as to identify potential trends and research directions. The proposed taxonomy for gains of microservices was obtained with an approach similar to that for determining the taxonomy for pains (Section 4.2.1).

The obtained taxonomy undergone various iterations among the authors of this study, and it was also submitted to external experts from two different research institutions. This resulted in some corrections and amendments to the first version of the taxonomy, and the final outcome is the taxonomy displayed in Table 5. The latter was then validated through the inter-rater reliability assessment described in Section 3.4.

4.3.2. Gains in the selected industrial studies

We classified the selected industrial studies based on the taxonomy for gains of microservices (Table 5). The obtained classification is displayed in Table 6. Due to reasons of readability and space, the table only captures the classification over the concerns listed in the taxonomy of gains[5] It still provides a first overview of the coverage of gains over the selected studies, which is even more evident in the bubble plot in Fig. 17. The figure clearly highlights how biggest gains come from architecture concerns, from the exploitation of design patterns, from the development of microservices,

[5] The detailed classification, displaying each occurrence of each gain, is publicly available in the replication package (see Sect. 3.5).

Table 5
A classification framework for gains of microservices.

Stage	Concern	Gain
Design	Architecture	Bounded contexts, Cloud native, Decentralised governance, Fault tolerance, Flexibility
	Design patterns	API gateway, Circuit breaker, Database per service, Message broker, Service discovery
	Security	Automation, Fine-grained policies, Firewalling, Isolation, Layering
Development	Microservices	CI/CD, Loose coupling, Reusability, Service size, Technology freedom
	Storage	Data persistence, Data isolation, Microservice-orientation
	Testing	Automation, Rollback, Unit testing, Updates
Operation	Deployment	Containerisation, Independency, Reliability, Speed
	Management	Fault isolation, Scalability, Updateability

Table 6
Classification of the selected studies based on the concerns listed in the taxonomy for gains of microservices.

	Design			Development			Operation			Design			Development			Operation	
	Arc.	*Des.*	*Sec.*	*Mic.*	*Sto.*	*Tes.*	*Dep.*	*Man.*		*Arc.*	*Des.*	*Sec.*	*Mic.*	*Sto.*	*Tes.*	*Dep.*	*Man.*
[S1]	•	•		•					[S27]	•			•			•	•
[S2]	•	•		•					[S28]	•			•			•	•
[S3]	•			•		•			[S29]	•			•			•	•
[S4]	•	•	•	•			•	•	[S30]	•	•	•	•	•		•	•
[S5]	•	•		•	•		•	•	[S31]	•		•	•			•	•
[S6]		•						•	[S32]	•	•		•			•	•
[S7]						•	•		[S33]			•					
[S8]	•	•		•	•		•	•	[S34]						•		•
[S9]		•							[S35]		•	•					
[S10]	•								[S36]		•			•			•
[S11]	•			•			•	•	[S37]	•	•		•	•	•	•	•
[S12]	•	•		•			•	•	[S38]	•	•	•	•	•		•	•
[S13]	•	•	•	•			•	•	[S39]		•		•			•	
[S14]	•				•			•	[S40]	•	•		•		•	•	•
[S15]	•			•					[S41]	•			•	•		•	•
[S16]		•		•	•	•	•	•	[S42]	•			•	•		•	•
[S17]	•			•			•	•	[S43]	•	•		•	•		•	•
[S18]	•			•					[S44]			•	•			•	
[S19]	•	•	•	•	•		•	•	[S45]		•		•				•
[S20]	•			•			•	•	[S46]		•	•			•		
[S21]		•	•						[S47]	•			•				•
[S22]	•	•		•	•		•	•	[S48]		•	•	•	•		•	•
[S23]	•	•	•						[S49]		•		•			•	•
[S24]				•			•	•	[S50]	•	•		•	•			
[S25]	•			•			•	•	[S51]	•	•		•	•	•	•	•
[S26]	•	•		•	•		•	•									

and from application peculiarities simplifying their actual deployment and management. Security design and storage implementation also provide notable gains, while application testing is not significantly impacting on the advantages of developing microservice-based applications.

In the following, we provide details on the notable concerns highlighted above. We shall proceed concern-wise, by listing the occurrences of gains in sorted tables, and by comparing their weights through %-based pie charts.

Design stage. The main sources of gains during the design stage are the architecture peculiarities and the exploitation of design patterns (Fig. 17). This can be observed also in Fig. 18, which shows the evolution of the coverage of design concerns in the period 2014–2017. The figure also shows that the interests for gains pertaining to the architecture concern is steadily increasing, while that for design patterns instead keeps stable. The same does not hold for security-related gains, which (after a period of slight increase) are starting to loose their appeal for the industrial researchers and practitioners authoring the selected industrial studies.

The concrete gains due to microservice-based architectures, their weights and occurrences in the selected industrial studies are illustrated in Fig. 19. Bounded contexts (which are an intrinsic peculiarity of microservice-based architectures) emerge as the primary architectural gain, with 24 out of the selected industrial studies highlighting their benefits. The notion of bounded contexts comes from (Evans, 2003), and it makes microservices selfcontained. A microservice can be analysed, understood and updated without knowing anything about the internals of the other microservices in an architecture. This impacts on the governance of an application, which can be decentralised by assigning the governance of microservices to different groups (as pointed out by 7 out of the 24 studies discussing the benefits of bounded contexts).

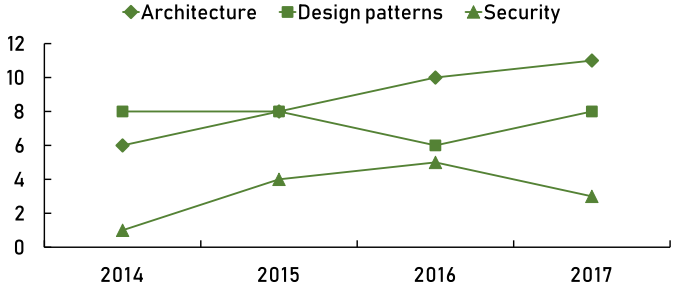
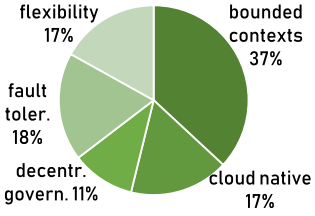


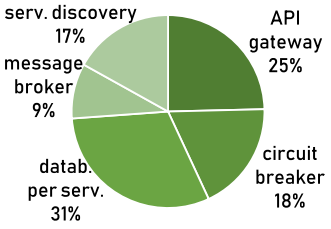
Fig. 18. Yearly evolution of the coverage of the concerns pertaining to the design stage.

Industrial researchers and practitioners are also highlighting that microservice-based architectures are fault tolerant by design, that microservices can be flexibly added and removed from an architecture, and that microservice-based architectures are cloud native. Fault tolerance and cloud nativeness of microservice-based architectures are two of the main reasons why the IT industry is



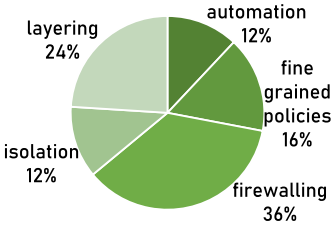
Gain	#
bounded contexts	24
fault tolerance	12
cloud native	11
flexibility	11
decentralised governance	7

Fig. 19. Weights and occurrences of the gains pertaining to the architecture concern during the design stage.



Gain	#
database per service	20
API gateway	16
circuit breaker	12
service discovery	11
message broker	6

Fig. 20. Weights and occurrences of the gains pertaining to design patterns during the design stage.



Gain	#
firewalling	9
layering	6
fine-grained policies	4
automation	3
isolation	3

Fig. 21. Weights and occurrences of the gains pertaining to security during the design stage.

strongly interested in them, with Netflix being one of the most prominent examples [S26],[S12].

The coverage of design patterns is instead displayed in Fig. 20. The design pattern *database per service* is the most discussed in the selected industrial studies, with 20 out of them highlighting that each microservice should be equipped with its own database (if it needs a persistent storage, of course). *API gateway* and *circuit breakers* are also significantly covered. 16 of the selected studies explain how the API gateway design pattern permits decoupling the clients of a microservice-based application from its internals, while 12 studies discuss how circuit breakers can help mitigating issues due to failures (such as cascading failures, one of the pains displayed in Fig. 14). *Service discovery* and *message brokers* are also discussed, to mitigate the pains of service location and service coordination, and to allow asynchronous service intercommunications, respectively. More generally, 30 of the selected studies were discussing at least one of the above listed patterns, by also showing how designers exploit them to mitigate some of the pains discussed in Section 4.2. This highlights the importance of design patterns, and the need for additional patterns allowing to further mitigate the pains of microservice-based architectures [S37].

Microservice-based architectures are also bringing some security-related gains, whose coverage in the selected industrial studies is displayed in Fig. 21. Firewalling emerges as the primary gain, with 9 of the selected industrial studies discussing how API gateways can simplify it in microservice-based architectures.

Other gains worth mentioning are due to the support of layered and fine-grained security policies. By defining hierarchical groupings of the microservices forming an architecture, designers can apply differently grained security policies to the different layers in a hierarchy. The partitioning of applications into small, independent microservices also simplifies the design of automated security policies and it favours a better isolation among the different components in an application (with respect to monolithic/service-oriented architectures).

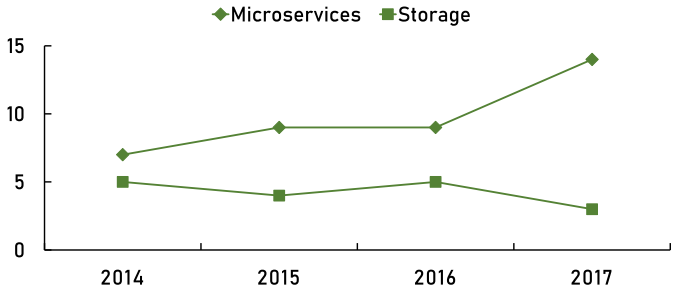
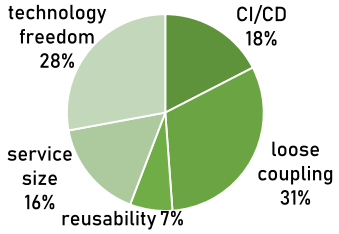


Fig. 22. Yearly evolution of the coverage of the concerns pertaining to the development stage.



Gain	#
loose coupling	27
technology freedom	24
CI/CD	15
service size	14
reusability	6

Fig. 23. Weights and occurrences of the gains pertaining to the concrete development of microservices.

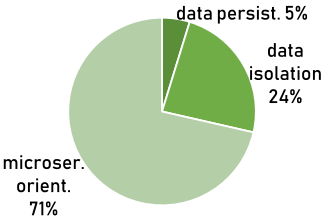
Development stage. According to Fig. 17, the development of microservices is the main source of gains in microservice-based architectures. This is also witnessed by the yearly coverage of development-related concerns (Fig. 22). The recognition of gains pertaining to the development of microservices is steadily increasing since 2014, and it keeps much higher with respect to that of the gains pertaining to storage.

The coverage of the concrete gains due to the development of microservices is displayed in Fig. 23, which also displays their weights in the selected industrial studies. The loose coupling (which allows to independently develop the microservices forming an application) and the freedom in choosing the technology stack for implementing each microservices are the most attractive gains. They are strictly related to one another, and their grouping is by far more covered with respect to the other gains pertaining to the development of microservices.

Another significant gain emerging from the selected industrial studies is related to CI/CD, a DevOps practice enabling the continuous release of updated versions of a software (Nygard, 2007). As pointed out by 15 of the selected industrial studies, microservices natively support CI/CD. The latter can be directly operated on microservices, hence allowing to continuously and independently release updated versions of the microservices forming an application.

The actual size of microservices and their reusability are also discussed in the selected industrial studies. As reusability is peculiar to many service-based architectural style, it is more interesting to comment on how the actual size of microservices impacts on their development. Microservices focus on doing one thing well, and their business logic turn out to be small and self-contained [S31]. 14 of the selected studies highlight how this makes them quick to develop and easy to understand for new personnel assigned to their development and maintenance.

Microservices have also some positive impact on storage concerns (Fig. 17). The coverage of the storage-related gains in the selected industrial studies is reported in Fig. 24. The only gain significantly covered is that related to the microservice orientation of



Gain	#
microservice orientation	15
data isolation	5
data persistence	1

Fig. 24. Weights and occurrences of the gains pertaining to storage during the development stage.

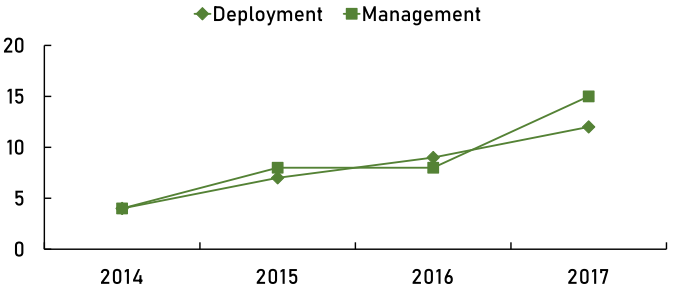
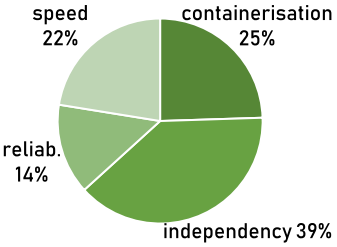


Fig. 25. Yearly evolution of the coverage of the concerns pertaining to the operation stage.



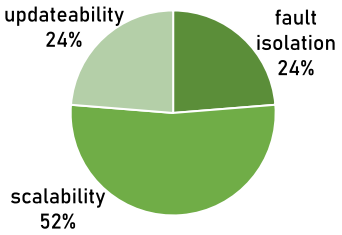
Gain	#
independency	19
containerisation	12
speed	11
reliability	7

Fig. 26. Weights and occurrences of the gains pertaining to deployment during the operation stage.

storage. 15 of the selected industrial studies indeed highlight that, when a microservice must be equipped with a backend database, its developer can freely choose the database type (e.g., SQL or NoSQL) and how to structure the data in it. This is because the only service accessing to such database will be the microservice she is developing (following the *database per service* design pattern).

Operation stage. As shown by Fig. 17, microservices provide significant benefits also during the operation stage. The recognition of gains pertaining to the deployment and management of microservices is steadily increasing since 2014 (Fig. 25). By crossing this data with the technology waves highlighted by Jamshidi et al. (2018), we can observe that the more technology for supporting microservices is getting mature, the more microservice become attractive for the gains they bring while being operated.

The coverage and weights of the gains pertaining to the deployment of microservices are illustrated in Fig. 26. The primary gain is independency. 19 of the selected industrial studies highlight that microservices can be deployed and undeployed independently from one to another. Microservices are designed and developed to be self-contained in a bounded context. Even if a microservice is exploiting (at runtime) some functionalities provided by other microservices, it can be deployed without requiring their availability. Once running, if one of the required microservices is not available (because it has not been deployed yet, or because it failed), the deployed microservice continue to be running, even if partly working [S31].



Gain	#
scalability	31
fault isolation	14
updateability	14

Fig. 27. Weights and occurrences of the gains pertaining to management during the operation stage.

Other significant gains are related to containerisation and speed of deployment. Containers (and Docker, in particular) emerge as the natural way of packaging and shipping microservices, hence fostering the build of portable, cloud-based deployments. This, along with the fact that microservices are small sized, makes microservices very fast to get up and running.

Fig. 27 instead displays the coverage and weights of the gains that microservices bring to the operation of the management of applications. The figure clearly shows tha scalability emerges as the primary gain for the industrial researchers and practitioners authoring the selected studies. 31 of the selected industrial studies indeed highlight how microservice applications naturally support the horizontal scaling of their microservices, as well as how microservices scale quickly and reactively (also thanks to their independence and speed of deployment).

Fault isolation and updateabilty are also significantly recognised as gains pertaining to the management of microservice-based applications. 14 of the selected studies point out that the partitioning of application in bounded microservices simplifies the operation of fault isolation, also thanks to the exploitation of the design pattern called circuit breaker (discussed earlier). 14 of the selected studies instead point out that, thanks to the independence among the microservices in an application, live upgrades and live updates can be applied to a running microservice, without requiring to touch the other microservices of an application.

4.4. Summary

Since 2014, when James Lewis and Martin Fowler gave rise to the microservice-based architectural style with their famous blog post [S22], microservices are steadily attracting increasing attention by the IT industry. This is supported by the trends discussed in Section 4.1, which answered to our Q_1 by illustrating how communities of practitioners and IT companies are more and more involved in industrial research on microservices. The growing interest in microservices by the IT industry is even more evident if we focus on the contributions published last year, whose vast majority was directly provided by IT companies.

The contents presented in the selected industrial studies span from discussing advantages and disadvantages of microservices, to proposing design patterns and sharing best practices to better leverage of microservices. This is in line with our research questions Q_2 and Q_3, which were answered in Sections 4.2 and 4.3, respectively. In the following, we report the main findings resulting from the analysis of the selected industrial studies, based on our taxonomies for pains and gains of microservices (Tables 3 and 5, respectively).

4.4.1. Summary of pains

The pains of microservice-based applications are mainly due to their intrinsic complexity. As recognized by almost all the selected industrial studies, the design, the development and/or the operation of microservice-based applications is hampered by the

fact that the business logic in such applications is heavily distributed over many independent and asynchronously evolving microservices. This generates various design, development and operational challenges, which are recapped below.

At design time, the primary pain of the design of microservice-based applications is the dimensioning of services. It is indeed often difficult to determine the partitioning of an application in bounded contexts, as the boundaries among the different business capabilities of an application are usually not sharp. The fact that intercommunications are purely based on remote API invocations is also creating concrete pains. The API provided by a microservice becomes a sort of contract between such microservice and the microservices consuming its API. This directly impacts on the versioning of APIs, as new API version must always be retro-compatible to avoid violating the contracts among microservices, hence allowing them to continue to intercommunicate.

Security is also generating pains at design-time, mainly due to access control and endpoint proliferations. Access control is one of the main challenges, as the design of microservice-based application should allow to the microservices forming an application to quickly and consistently ascertain the provenance and authenticity of a request (which is far from being easy due to the heavily distributed nature of microservice-based applications). Additionally, endpoints proliferate in microservice-based applications, as all their microservices are exposing remotely accessible APIs exposed. The attack surface to be secured is hence much larger with respect to classical monolithic/service-oriented applications.

At development time, the primary pains come from storage-related issues. By distributing the storage over the many microservices forming an application, it becomes challenging to ensure data consistency. Eventual consistency is an option to mitigate this pain, even if not always viable and not easy to implement too. At the same time, the heavy distribution of data makes it challenging also to implement distributed transaction, and even to query the data (also because of the heterogeneity of the data stores to be queried).

Testing is also paining the job of developers. Microservice-based application partition their business logic over independently evolving services, and this makes it challenging to measure the performances of the application itself. This especially holds when developers are required to test the user experience of a microservice-based application [S34].

The operation of microservice-based applications is also pained by their distributed and dynamic nature. Microservices can flexibly be added or removed from an application, they can be scaled in and out, or they can be migrated from one host to another. This, along with the huge number of microservices forming an application, makes it challenging to locate and coordinate the concrete instances of the microservices in an application. Additionally, if not properly isolated, a failing instance of a microservice can generate a cascade of failure in the microservice instances depending on it.

At the same time, the distribution of the instances of the microservices in an application, results in the distribution of their logs. Delving through many distributed logs is of course a pain, especially when the operators of a microservice-based application are required to identify the reasons/problem causing an issue in such application.

The primary pain during the operation of microservice-based applications is however given by their resource consumption. The require to run more services with respect to monolithic and service-based application, which require more runtime environments to be distributed, and which interact based on remote API invocations. This increases their consumption of compute and network resources, with the latter being the primary worry of the industrial researchers and practitioners authoring the selected studies.

4.4.2. Summary of gains

The primary gains of microservices are due to peculiar properties of microservice-based architectures, to design patterns allowing to better exploit them, to the ease of development of the microservices in an application, and to the possibility of independently deploying and managing the microservices in an application.

Consider the design of microservice-based architectures. Despite identifying the partitioning of applications in bounded contexts is not easy, microservices actually gain from the boundedness of their context. A microservice is self-contained, and it can be analyzed, understood and updated without knowing anything about the rest of the architecture it is part of. This positively impacts on the governance of applications, which can be decentralized by assigning the governance of different microservices to different groups. Additionally, bounded contexts enforce fault tolerance by design, and they permit flexibly adding and removing microservices to and from an architecture. The resulting architectures are also cloud-native, as their bounded contexts/microservices can be independently managed over different cloud platforms.

The mostly recognized gains at design-time are however those related to the exploitation of design patterns. The design pattern *database per service* permits enforcing the independence among microservices, by equipping each microservice with its own data store (if needed). *API gateway*s permit decoupling the clients of a microservice-based application from its internals, while *circuit breakers* and *service discovery* to mitigate the pains related to the failure and service coordination/location of microservices. The importance of design patterns for mitigating the pains of microservices is also explicitly highlighted by Richardson [S37].

The distributed nature of microservices can also provide some security-related gains, even the pains it brings are more significantly recognized in the selected grey literature. Microservice-based architecture indeed naturally support the possibility of defining hierarchical security policies, with differently grained security policies associated with different levels in a hierarchy. Design patterns also help securing microservices, with the API gateway emerging as the most prominent example, as it thoroughly simplifies firewalling microservice-based architectures.

Development emerges by far as the stage gaining most from microservices. In this perspective, the most important gains recognized by the industrial researchers and practitioners authoring the selected studies are the loose coupling among the microservices in an application (which permits independently developing them), the freedom in choosing the technology stack for developing a microservice, and the possibility of choosing the data store that best suits the needs of a microservice.

Other important gains come from the ease of use of DevOps techniques in microservice-based applications, and from the actual size of the microservices forming an application. On the one hand, CI/CD can be directly operated on each microservice in an application, hence allowing to continuously release updated/upgraded versions of such microservice, independently from the rest of an application. On the other hand, microservices focus on doing on thing and on doing it well, with a business logic that turns out to be small and self-contained [S31]. This makes it quick their development, as well as their understanding by new personnel assigned to their development and maintenance.

Finally, consider the operation of microservice-based applications. The possibility of independently deploying the microservices in an application is recognized as one of the most important gains due to microservices. A microservice can indeed be deployed without requiring the availability of other microservices, even if it exploits their functionalities at runtime. Once running, if one of the required microservices is not available (because it has not been deployed yet, or because it failed), the deployed microservice con-

Table 7
Coverage of the identified (a) pains and (b) gains in previous surveys on microservices. (Cell *i, j* shows the fraction of pains or gains pertaining to concern *j* that are recognized by survey *i*.).

	Design		Development		Operation		
	Arc.	*Sec.*	*Sto.*	*Tes.*	*Man.*	*Mon.*	*Res.*
Di Francesco et al. (2017)	1/5	1/6	2/4	1/3	2/4	1/3	0/2
Ghofrani and Lübke (2018)	2/5	1/6	1/4	0/3	1/4	1/3	1/2
Pahl and Jamshidi (2016)	2/5	1/6	0/4	2/3	3/4	1/3	0/2
Taibi and Lenarduzzi (2018)	5/5	0/6	0/4	1/3	2/4	0/3	0/2
Taibi et al. (2018)	2/5	1/6	2/4	1/3	3/4	1/3	0/2
Vural et al. (2017)	1/5	1/6	0/4	0/3	1/4	1/3	0/2

(a)

	Design			Development		Operation	
	Arc.	*Des.*	*Sec.*	*Mic.*	*Sto.*	*Dep.*	*Man.*
Di Francesco et al. (2017)	5/5	2/4	0/5	2/5	0/4	3/4	1/3
Ghofrani and Lübke (2018)	3/5	0/4	0/5	3/5	0/4	1/4	3/3
Pahl and Jamshidi (2016)	4/5	4/4	0/5	3/5	0/4	3/4	1/3
Taibi and Lenarduzzi (2018)	1/5	2/4	0/5	2/5	0/4	1/4	2/3
Taibi et al. (2018)	3/5	4/4	1/5	5/5	2/4	3/4	2/3
Vural et al. (2017)	4/5	1/4	0/5	2/5	0/4	2/4	2/3

(b)

tinue to be running, even if partly working [S31]. This simplifies the orchestration of the deployment of microservices, and it is the source of other related gains, e.g., reliability, fault isolation, and the possibility of applying live updates/upgrades to a running microservices, without being required to touch the other microservices in an application.

Independence of deployment is second only to scalability, which is by far recognized as the most important gain of microservices during the operation stage. Microservice-based applications naturally support the horizontal scaling of their microservices, and their scaling is quick and reactive (also thanks to their independence and speed of deployment).

Microservices also benefit from their natural integration with container-based platforms, as each microservice can be packaged and shipped in its own container. This makes microservice-based applications easy to be deployed on and migrated over multiple cloud platforms, as containers (and Docker containers, in particular) are nowadays widely supported by cloud providers (Pahl et al., 2017).

5. Discussion

5.1. Relation with previous surveys and the road ahead

We hereby discuss the relation with the results presented in this study and those illustrated by previous surveys on microservices. The coverage of the identified pains and gains in such surveys is illustrated in Table 7.

We can observe that the recognition of architecture-related pains and gains in previous surveys is aligned with that in our study. The same applies to the recognition of the gains pertaining to design patterns, to the development of microservices, and to the deployment and management of microservice-based applications.

On the other hand, the recognition of pains and gains pertaining to other concerns in previous surveys is not aligned with that in our study. This confirms that there is a gap between the industrial understanding and state-of-practice on microservices and the state-of-the-art of academic research. As highlighted by Jamshidi et al. (2018), one possible reason is that academia have limited access to industry-scale microservice-based application. This makes it difficult to academic researchers to concretely experience the technical and non-technical issues that might occur in real-world microservice production environments.

The above especially holds where the higher gap is registered, namely (i) for security-related pains and gains, (ii) for the difficulties in testing microservice-based applications (and especially their performances), (iii) for the pains in monitoring microservices, and (iv) for the issues due to the fact that microservices are highly resource consuming. All such issues are significantly recognized by industrial researchers and practitioners working with industry-scale application, while the academia still needs to concretely experience and work on them. This in turn means that solutions for (i) enforcing security policies in a decentralised manner, (ii) supporting performance testing of microservice-based applications, (iii) easing their monitoring, and especially logging and problem location, and (iv) optimizing their resource consumption are (and must be) research directions for future work by both industrial and academic researchers.

Other interesting research directions are related to concrete pains that still need to be addressed. For instance, there is a need for methodologies and techniques easing the dimensioning and versioning of microservices, and simplifying the execution of transactions/queries on distributed and heterogeneous data stores.

Nonetheless, even if some of the primary pains and gains have already been investigated, there is still a lot of work to do on them. A concrete example is the study of microservice-oriented design patterns, as industrial researchers and practitioners strongly believe that they are key to mitigate/nullify the pains of microservices. While various design patterns have already been proposed (e.g., those listed in the blog http://microservices.io by Chris Richardson) many other are yet to be investigated to further mitigate the pains of microservices [S37]. A first research effort in this direction is given by Taibi et al. (2018), which further witnesses the need for investigating and formalizing design patterns allowing to better exploit microservices.

5.2. Observations on systematic grey literature reviews

Our work led to two major observations on difficulties and potentials of conducting systematic grey literature reviews, which could be of help for researchers willing to perform them. We discuss them hereafter:

- We observed that the combination of systematic literature review protocols with grey literature studies can be valuable to shed light on yet uncharted areas of software engineering research, especially when such areas are seeing massive industrial adoption. This is the case of microservices, where we observed a massive proliferation of grey literature, with more than 10,000 articles on disparate sub-topics arranged between blogs, posts, comments, articles, white-papers, technology evaluations, technical reports and videos. A systematic review protocol allows to distill the essential state-of-practice on a subtopic (such as the pains and gains of microservices, in our case), which can be of help for researchers and practitioners.
- At the same time, we discovered that it is very difficult to uniquely measure the quality of grey literature when conducting a systematic, controllable and replicable secondary study. This is mainly because grey literature lacks an unique format acknowledged across all sources and available data. This lacking inspired us to build a rudimentary quality control framework (see Section 3.3), which can be of help to researchers willing to perform systematic grey literature reviews in software engineering. Anyway, strategies and approaches for multivocal literature research should be explored further, especially concerning quality control of grey literature, both standalone and in combination with white literature.

It is also worth highlighting that we observed intrinsic differences in classifying videos with respect to the other types of industrial studies (i.e., blog posts and whitepapers). We were able to evaluate the inclusion/exclusion criteria on blog posts and whitepapers by giving them a first, light read. With a second, deeper read, we were then able to classify them by applying our taxonomies of pains and gains. The same did not happen with videos, which we were required to carefully follow from the very first time. As a consequence, the search and classification of videos was significantly more time-consuming with respect to that of blog posts and whitepapers.

At the same time, videos gave us less room for interpretation, also thanks to the fact that the contributions contained in videos are actually presented by the authors of the studies themselves (who decide what to focus on, by discussing it more in detail, and how). This resulted in us being more aligned when classifying videos with respect of when we were classifying blog posts and whitepapers.

5.3. Threats to validity

Based on the taxonomy developed by Wohlin et al. (2000), four potential threats of validity may apply to our study, i.e., threats to *external* validity, threats to *construct* and *internal* validities, and threats to *conclusions* validity.

5.3.1. Threats to external validity

External validity concerns the applicability of a set of results in a more general context Wohlin et al. (2000). Since our primary studies are obtained from a large extent of online sources, our results and observations may be only partly applicable to the broad area of practices and general disciplines of microservices practices and general discipline, hence threatening external validity.

To reinforce the external validity of our findings, we organized 4 feedback sessions during our systematic analysis of grey literature on microservices. We analyzed the discussion following-up from each feedback session, and we exploited this qualitative data to fine-tune both our research methods and the applicability of our findings. We also prepared a bundle of all artifacts we produced during our analysis, so as to make it available to all who wish to deepen their understanding on our data (see Section 3.5). We are confident that this can help in making our results and observations more explicit and applicable in practice.

On another front, there is a risk of having missed relevant industrial studies, due to the fact that concepts related to those included in our search strings are differently named in such studies (e.g., a study discussing some pains of microservices may not employ the terms “pain” or “gain”, but rather some synonyms). This holds especially for videos, as our search strings were matched only against their title and description. To mitigate this threat to validity, we have explicitly included all relevant synonyms in our search strings (by looking among those listed in the Merriam-Webster’s dictionary), and we have also exploited the features offered by search engines, which naturally support considering related terms for all those contained in a search string.

5.3.2. Threats to construct and internal validities

According to Wohlin et al. (2000), construct validity concern the generalizability of the constructs under study, while internal validity concerns the validity of the methods employed to study and analyze data (e.g., the types of bias involved).

To mitigate these threats, we adopted various triangulation rounds and inter-rater reliability assessment methods (see Section 3.4), which were conceived to avoid bias by triangulation. To further reinforce internal and construct validity, the initially obtained taxonomies for pains and gains were submitted to two external domain-experts from two different research institutions, and the analysis results were checked by another external reviewer which is not among the authors and not belonging to the software engineering field.

5.3.3. Threats to conclusions validity

The validity of conclusions concerns the degree to which the conclusions of a study are reasonably based on the available data (Wohlin et al., 2000).

To mitigate this threat, we exploited theme coding and inter-rater reliability assessment to limit observer bias and interpretation bias, with the ultimate goal of performing a sound analysis of the data we retrieved. Additionally, the conclusions drawn in this article were independently drawn by each of us, and they were then double-checked against the selected industrial studies or related studies in a joint discussion session.

6. Conclusions

Microservices are nowadays adopted by many IT companies to deliver their business, with Amazon, Netflix, Spotify, and Twitter being the most prominent examples. Due to the huge investment by companies, the industrial state-of-practice on microservices is rapidly evolving and it has already reached some degree of maturity. At the same time, academic research efforts are still at an early stage, as also discussed by existing secondary studies, e.g., Pahl and Jamshidi (2016), Vural et al. (2017). This resulted in a sort of gap between academic state-of-the-art and industrial state-of-practice.

The objective of this article was to try to bridge the above mentioned gap by following the guidelines provided by Garousi et al. (2017). Our aim was indeed to complement academic state-of-the-art with a systematic analysis of the industrial grey literature on pains and gains of microservices. We followed a systematic approach based on that proposed by Petersen et al. (2008), which allowed us to select 51 industrial studies, and to analyse them to distill the pains and gains of designing, developing and operating microservices.

Our study showed that microservices are steadily gaining increasing attention by the IT industry, since 2014 (when they were

first defined by Lewis and Fowler). It also showed that the understanding of pains and gains of microservices is quite mature in the industry, hence meaning that the academia has much to learn from the industry on the topic. The identified pains are mainly related to the intrinsic complexity of microservice-based applications, while the gains relate to peculiar properties of microservice-based architectures, to design patterns allowing to better exploit them, and to the possibility of independently deploying and managing the microservices in an application. Both pains and gains can be useful to delineate research directions, as we discussed in Section 5.1.

We believe that both researchers and practitioners can benefit from our study. Our findings can indeed be exploited by researchers to develop new theories and solutions, to analyze and experiment research implications, and to establish future research dimensions. At the same time, our study summarizes the currently recognized pains and gains of microservices, and their maturity. Practitioners can hence exploit our findings as a starting point for microservices experimentation or as a guideline for day-by-day work with microservices.

In the scope of our future work, we plan to extend our analysis, by trying to identify which design principles for microservices stem from the industrial literature. We indeed plan to understand whether there already exist a number of design principles for microservices, intended as sound microservices design, development, and deployment decisions that lead to beneficial or more controllable microservice orchestrations.

We also plan to devise a methodology for metering "how much" each pain is actually paining a microservice-based application, and how intense are its gains. Indeed, even if the selected industrial studies recognise technical/operational pains and gains of microservices, they do not provide any quantitative information. This is mainly because a concrete solution for measuring the intensity of each pain/gain in a given microservice-based application is currently lacking (as pointed out by Zimmermann (2017), and by [S45] and [S6], for instance). Such a metering solution would permit tackling design decisions, by allowing to tune microservice-based architectures in order to maximize desired gains and to minimize unacceptable pains.

It is finally worth noting that our study focused on eliciting the *technical/operational* pains and gains of microservices. However, as highlighted by Jamshidi et al. (2018), *organisational* issues are also part of current and future challenges of microservices, from both the industrial and academic perspectives. This is why we plan to complement this study with a follow-up study, where we plan to systematically analyse grey literature to elicit the organisational benefits and drawbacks of microservices.

In this perspective, another (orthogonal) direction for our future work is the investigation and development of a systematic approach for *grey* literature reviews. To the best of our knowledge, ours is one of the first systematic grey literature reviews in the scope of software engineering, and there is a need for a protocol allowing to conduct systematic grey literature reviews (Garousi et al., 2016). This also follows from the lessons we learned with our study (see Section 5.2), where we observed that the combination of systematic literature review protocols (such as that by Petersen et al. (2008)) with grey literature studies can be valuable to shed light on yet uncharted areas of software engineering research, especially when such areas are characterized by a massive industrial adoption.

Appendix A. Selected Industrial Studies

[S1] Vijay Alagarasan. Seven Microservices Anti-Patterns. *InfoQ*, 2015. https://www.infoq.com/articles/seven-uservices-antipatterns.

[S2] Abel Avram. The Strengths and Weaknesses of Microservices. *InfoQ*, 2014. https://www.infoq.com/news/2014/05/microservices.

[S3] Vineet Badola. Microservices Architecture: Advantages and Drawbacks. *CloudAcademy's blog*, 2015. https://www.cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback.

[S4] Scott Carey. The Pros and Cons of Microservices. *ComputerWorldUK*, 2016. https://www.computerworlduk.com/applications/pros-cons-of-microservices-lessons-learned-by-comparethemarketcom-3642668/.

[S5] Chakray. What Are Microservices? Definition, Characteristics, Advantages and Disadvantages. *Chakray's blog*, 2017. https://www.chakray.com/en/what-are-microservices-definition-characteristics-and-advantages-and-disadvantages.

[S6] Clifton Cunnigham, Owen Garret, Matteo Collina, Mark Little. Pros/cons of highly scalable micorservice systems. *Microservices Day*, 2017. https://www.youtube.com/watch?v=w_0z_sKkul0.

[S7] David Dawson. Testing Microservices: Pain or Opportunity?. *Microservices Manchester*, 2016. https://www.youtube.com/watch?v=AjQlyzoFrsM.

[S8] Dev2. The pros and cons of microservices. *Dev2 Technologies*, 2015. https://www.youtube.com/watch?v=li2BSeNbo8c.

[S9] Rohit Dhall. Performance Patterns in Microservices-based Integration. *DZone*, 2016. https://www.dzone.com/articles/performance-patterns-in-microservices-based-integr-1.

[S10] Niels Dommerholt. Micro Services: Versioning Strategies. *Niels.nu*, 2016. http://niels.nu/blog/2016/microservice-versioning.html.

[S11] John E. Dunn. What are microservices? *ComputerWorldUK*, 2016. https://www.computerworlduk.com/applications/microservices-explained-is-this-just-tweaked-soa-or-something-much-bigger-3638372.

[S12] Josh Evans. Mastering chaos - A Netflix guide to microservices. *QCon, InfoQ*, 2016. https://www.youtube.com/watch?v=CZ3wIuvmHeM.

[S13] Martin Fowler. Microservices Trade-Offs. *MartinFowler.com*, 2015. https://www.martinfowler.com/articles/microservice-trade-offs.html.

[S14] Bernard Golden. 5 Fundamentals to a Succesful Microservices Design. *TechBeacon*, 2016. https://www.techbeacon.com/5-fundamentals-successful-microservice-design.

[S15] Andreas Grabner. Locating Common Micro Service Performance Anti-Patterns. *InfoQ*, 2016. https://www.infoq.com/articles/Diagnose-Microservice-Performance-Anti-Patterns.

[S16] Philipp Hauer. Microservices in a Nutshell: Pros and Cons. *Philipphauer.de*, 2015. https://www.blog.philipphauer.de/microservices-nutshell-pros-cons.

[S17] Pete Heard. What are Microservices? [Costs and Benefits]. *LogicRoom.co*, 2017. https://www.logicroom.co/what-are-microservices-costs-and-benefits.

[S18] Troy Hunt. Your API Versioning is Wrong. *DZone*, 2014. https://www.dzone.com/articles/your-api-versioning-wrong.

[S19] Kasun Indrasiri. Microservices in Practice: From Architecture to Deployment. *DZone*, 2016. https://www.dzone.com/articles/microservices-in-practice-1.

[S20] Siva Kumar. Advantages and Disadvantages of Microservices. *Linkedin Pulse*, 2017. https://www.linkedin.com/pulse/advantages-disadvantages-microservices-siva-kumar.

[S21] Graham Lea. Microservices Security: All the Questions You Should Be Asking. *Grahamlea.com*, 2015. http://www.grahamlea.com/2015/07/microservices-security-questions.

[S22] James Lewis, Martin Fowler. Microservices: A Definition of this New Architectural Term. *MartinFowler.com*, 2014. https://www.martinfowler.com/articles/microservices.html.

[S23] Joshua Long. The Power, Patterns and Pains of Microservices. *DZone*, 2015. https://www.dzone.com/articles/the-power-patterns-and-pains-of-microservices.
[S24] Rory MacDonald. Microservices: Pros & Cons of Using Microservices On A Project. *MadeTech.com*, 2015. https://www.madetech.com/blog/microservices-pros-and-cons.
[S25] Scott Matteson. 10 Tips for Securing Microservice Architectures. *TechRepublic*, 2017. https://www.techrepublic.com/article/10-tips-for-securing-microservice-architecture.
[S26] Tony Mauro. Adopting Microservices at Netflix: Lessons for Architectural Design. *Nginx's blog*, 2015. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices.
[S27] Micro Technologies. Microservices. *MicroDocs*, 2017. https://www.micro.mu/docs/microservices.html.
[S28] Per Minborg, Emil Forslund. Three Microservice Patterns to Tear Down Your Monoliths. *JavaOne*, 2017. https://www.youtube.com/watch?v=yxZm0Fhn9Tk.
[S29] John Mueller. Performance Issue Considerations for Microservices APIs. *Smartbear's blog*, 2015. https://www.blog.smartbear.com/software-quality/performance-issue-considerations-for-microservices-apis.
[S30] Sam Newman. The Principles of Microservices. *O'Reilly*, 2016. https://www.safaribooksonline.com/videos/the-principles-of/9781491935811.
[S31] Ekaterina Novoseltseva. Benefits of Microservices Architecture Implementation. *DZone*, 2017. https://www.dzone.com/articles/benefits-amp-examples-of-microservices-architectur.
[S32] Arvind Patil. Microservices Architectural Style. *Valtech's blog*, 2017. https://www.valtech.com/blog/microservices-architectural-style.
[S33] Ranga Rajagopalan. Rethinking Application Security with Microservices Architectures. *DarkReading.com*, 2016. https://www.darkreading.com/endpoint/rethinking-application-security-with-microservices-architectures/a/d-id/1325155.
[S34] Henrik Rexed. Testing & Monitoring the Performances of Microservices. *NeoTys' blog*, 2015. https://www.neotys.com/blog/testing-monitoring-the-performance-of-microservices.
[S35] Chris Richardson. API Gateway: Backend for Front-end. *Microservices.io*, 2014. http://microservices.io/patterns/apigateway.html.
[S36] Chris Richardson. Database per service. *Microservices.io*, 2014. http://microservices.io/patterns/data/database-per-service.html.
[S37] Chris Richardson. Microservice Architecture. *Microservices.io*, 2014. http://microservices.io/patterns/microservices.html.
[S38] Chris Richardson, Floyd Smith. Microservices -From Design To Deployment. *Nginx*, 2016. https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx.
[S39] Tammel Saleh. Microservices anti-patterns. *Nordic APIs Platform Summit*, 2014. https://www.youtube.com/watch?v=l56HzTKvZKc.
[S40] Boris Scholl. Getting Started with Microservices, Part 1: Advantages and Considerations. *Oracle's Blog*, 2017. https://www.blogs.oracle.com/developers/getting-started-with-microservices-part-one.
[S41] Cassandra Shum. Pros/cons of microservices. *Microservices Day*, 2017. https://www.youtube.com/watch?v=Fso7pTK7yT4.
[S42] Cassandra Shum, Rachel Laycok. Microservices: Pros and Cons. *O'Reilly Software Architecture Conference*, 2017. https://www.safaribooksonline.com/library/view/oreilly-software-architecture/9781491976142/video288556.html.
[S43] Prabath Siriwadena. Securing Microservices (Part I). *FacileLogin's blog*, 2016. https://www.medium.facilelogin.com/securing-microservices-with-oauth-2-0-jwt-and-xacml-d03770a9a838.
[S44] Sudhir Tonse. Effective IPC in the Cloud: The pros and cons of microservices architecture. *AWS re:Invent*, 2014. https://www.youtube.com/watch?v=CriDUYtfrjs.
[S45] Tamás Török. Microservice Architecture: All the Best Practices You Need to Know. *CodingSans' blog*, 2017. https://www.codingsans.com/blog/microservice-architecture-best-practices.
[S46] Marco Troisi.8 Best Practices for Microservices Security. *TechBeacon*, 2017. https://www.techbeacon.com/8-best-practices-microservices-security.
[S47] Manisha Verma. Microservices: Benefits and Challenges. *DZone*, 2017. https://www.dzone.com/articles/microservices-benefits-and-challenges.
[S48] Mike Wasson, Stuart Celarier. Microservices Architectural Style. *Microsoft*, 2017. https://www.docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.
[S49] Maira Wenzel, Cesar De La Torre, Luke Latham, Bill Wagner, Alma Jenks, Mike Jones. Designing a microservice-oriented application. *Microsoft*, 2017. https://www.docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/multi-container-microservice-net-applications/microservice-application-design.
[S50] Benjamin Wootton. Microservices: Not a Free Lunch. *HighScalability*, 2014. http://www.highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html.
[S51] Krzysztof Ziomek. Microservices: Pain and Gain. *Linkedin Pulse*, 2016. https://www.linkedin.com/pulse/microservices-pain-gain-krzysztof-ziomek.

References

Balalaie, A., Heydarnoori, A., Jamshidi, P., 2016. Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Softw. 33 (3), 42–52. doi:10.1109/MS.2016.64.
Buder, J., Creß, U., 2003. Manual or electronic? The role of coding in qualitative data analysis. Educ. Res. 45 (2), 143–154.
Di Francesco, P., Lago, P., Malavolta, I., 2017. Research on architecting microservices: trends, focus, and potential for industrial adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA). IEEE Computer Society, pp. 21–30. doi:10.1109/ICSA.2017.24.
Erl, T., Merson, P., Stoffers, R., 2017. Service-Oriented Architecture: Analysis and Design for Services and Microservices. Prentice Hall.
Evans, E., 2003. Domain-Driven Design: Tacking Complexity In the Heart of Software. Addison-Wesley Longman Publishing Co., Inc.
Farace, D., Schöpfel, J., 2010. (Eds.), Grey Literature in Library and Information Studies. K.G. Saur.
Fountain, T.C., 2003. Web service oriented architecture: "Smart operations" and IT strategy. In: Zhang, L.-J. (Ed.), ICWS. CSREA Press, pp. 481–486.
Garousi, V., Felderer, M., Mäntylä, M.V., 2016. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 26:1–26:6. doi:10.1145/2915970.2916008.
Garousi, V., Felderer, M., Mäntylä, M.V., 2017. Guidelines for including the grey literature and conducting multivocal literature reviews in software engineering. CoRR abs/1707.02553.
Ghofrani, J., Lübke, D., 2018. Challenges of microservices architecture: a survey on the state of the practice. In: Herzberg, N., Hochreiner, C., Kopp, O., Lenhard, J. (Eds.), Proceedings of the 10th Central European Workshop on Services and their Composition, Dresden, Germany, February 8–9, 2018. CEUR-WS.org, pp. 1–8.
Haselböck, S., Weinreich, R., Buchgeher, G., 2017. Decision models for microservices: design areas, stakeholders, use cases, and requirements. In: Lopes, A., de Lemos, R. (Eds.), Software Architecture, ECSA 2017. Springer, pp. 155–170.
Hassan, S., Bahsoon, R., 2016. Microservices and their design trade-offs: a self-adaptive roadmap. In: Zhang, J., Miller, J.A., Xu, X. (Eds.), 2016 IEEE International Conference on Services Computing (SCC). IEEE Computer Society, pp. 813–818.
Jamshidi, P., Pahl, C., Mendonca, N.C., Lewis, J., Tilkov, S., 2018. Microservices: the journey so far and challenges ahead. IEEE Softw. 35 (3), 24–35. doi:10.1109/MS.2018.2141039.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University.
Krippendorff, K., 2004. Content Analysis: An Introduction to its Methodology, (second edition) Sage Publications.
Nygard, M., 2007. Release It!: Design and Deploy Production-Ready Software. Pragmatic Bookshelf.
Pahl, C., Brogi, A., Soldani, J., Jamshidi, P., 2017. Cloud container technologies: a state-of-the-art review. IEEE Trans. Cloud Comput. doi:10.1109/TCC.2017.2702586.
Pahl, C., Jamshidi, P., 2016. Microservices: a systematic mapping study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2. SCITEPRESS, pp. 137–146. doi:10.5220/0005785501370146.
Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. BCS Learning & Development Ltd., pp. 68–77.
Sill, A., 2016. The design and architecture of microservices. IEEE Cloud Comput. 3 (5), 76–80.
Stempfhuber, M., Schaer, P., Shen, W., 2008. Enhancing visibility: Integrating grey literature in the sowiport information cycle. In: Ninth International Conference on Grey Literature: Grey Foundations in Information Landscape.
Taibi, D., Lenarduzzi, V., 2018. On the definition of microservice bad smells. IEEE Softw. 35 (3), 56–62. doi:10.1109/MS.2018.2141031.
Taibi, D., Lenarduzzi, V., Pahl, C., 2018. Architectural patterns for microservices: asystematic mapping study. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science. SCITEPRESS, pp. 221–232.
Thönes, J., 2015. Microservices. IEEE Software 32 (1), 113–116.
Vural, H., Koyuncu, M., Guney, S., 2017. A systematic literature review on microservices. In: Gervasi, O., Murgante, B., Misra, S., Borruso, G., Torre, C.M., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Stankova, E., Cuzzocrea, A. (Eds.), Computational Science and Its Applications – ICCSA 2017. Springer International Publishing, pp. 203–217.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers.
Zimmermann, O., 2017. Microservices tenets. Comput. Sci.– Res.Dev. 32 (3), 301–310. doi:10.1007/s00450-016-0337-0.

Jacopo Soldani is a post-doc researcher at the University of Pisa (Italy). He holds a PhD in Computer Science (2017, University of Pisa). His research interests include, but are not limited to, service-oriented and cloud computing, adaptation, coordination, and integration of software elements, and formal methods. He is involved in research projects on cloud and fog computing both at local and EU level.

Damian Andrew Tamburri is an assistant professor at the Jheronimus Academy of Data Science and Technical University of Eindhoven. Damian completed his Ph.D. at VU University Amsterdam, The Netherlands in March 2014 one year in advance. Though still in his very early career, he has published over 60+ papers in international journals and conferences. His current research interests lie mainly in advanced software architecture styles, advanced software architecting methods, and social software engineering.

Willem-Jan van den Heuvel is a full professor in Information Systems and managing director of the European Research Institute of Services Science (ERISS). His research interests are at the cross-junction of software service systems and business process management with an emphasis on (global) networked enterprises. In particular, his expertise revolves around the following major research themes: business process management, Big data analytics, service engineering (including service governance) and legacy system modernization.