Extracción de datos de revisión sistemática de literatura blanca 2014-2019

Patrones relacionados a la arquitectura de microservicios

Valdivia Ruiz José Alí

Directores
Dr. Limón Riaño Héctor Xavier
Dra. Cortés Verdín María Karen

Licenciatura en Ingeniería de Software

Universidad Veracruzana

Tabla de contenido

Tablas de datos generales

Documento E1

<table>
<tr><th colspan="2">DATOS GENERALES</th></tr>
<tr><td>Título</td><td>A case-based reasoning approach to reuse quality-driven designs in service-oriented architectures</td></tr>
<tr><td>Autores</td><td>Rodríguez, G
Díaz-Pace, J A
Soria, Á</td></tr>
<tr><td>Año</td><td>2018</td></tr>
<tr><td>Fuente</td><td>Elsevier Ltd</td></tr>
<tr><td>Tipo de publicación</td><td>Article</td></tr>
<tr><td>Referencia o detalles de la publicación</td><td>https://doi.org/10.1016/j.is.2018.06.003</td></tr>
<tr><td>Palabras clave</td><td>Case-based reasoning, Tool support, Object-oriented design, Quality attributes, Service-oriented architecture, Web Services.</td></tr>
<tr><td>Resumen</td><td>Service-Oriented Architecture (SOA) has become a dominant approach for developing distributed enterprise-wide applications. Most organizations capitalize on SOA by discovering and reusing services already accessible over the Internet. In addition to functional requirements, the implementation of a SOA design must consider quality-attribute properties (e.g., performance, interoperability or security, among others), which require developers to explore and assess candidate solutions fulfilling the same functional requirements. This exploration is usually driven by architectural knowledge and SOA principles, but it can be a time-consuming and error-prone process, even for expert developers. To deal with this issue, we present a case-based reasoning approach called AWESOME to assist developers in exploring differ- ent development alternatives, by modeling quality-attribute aspects and SOA design patterns as cases. Our approach has been evaluated with four case-studies, and the results have shown that the solutions generated by AWESOME are judged as satisfactory by a number of SOA experts.</td></tr>
<tr><td>Respuesta a preguntas de investigación</td><td>
<table>
<tr><th>Referencia a pregunta de investigación</th><th>Responde a la pregunta</th><th>Referencia a tabla de análisis</th></tr>
<tr><td>PI1</td><td>✓</td><td>PI1-02</td></tr>
<tr><td>PI2</td><td>✓</td><td>PI2-02</td></tr>
<tr><td>PI3</td><td>✗</td><td></td></tr>
</table>
</td></tr>
<tr><td>Notas</td><td>Es un interesante trabajo sobre una propuesta case para sistemas SOA, que se basa en patrones y atributos de calidad. Responde a las preguntas, sin embargo, todo el documento tiene el contexto de SOA. Esto quiere decir que es necesario identificar si estos patrones también se replican en</td></tr>
</table>

	microservicios. La identificación se ha hecho con ayuda de un experto en microservicios.

Documento E3

DATOS GENERALES			
Título	**An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS)**		
Autores	**Hecht, Geoffrey** **Jose-Scheidt, Benjamin** **Figueiredo, Clement De** **Moha, Naouel** **Khomh, Foutse**		
Año	**2014**		
Fuente	**IEEE**		
Tipo de publicación	**Conference proceedings**		
Referencia o detalles de la publicación	**http://ieeexplore.ieee.org/document/7037678/**		
Palabras clave	**Cloud Patterns, Replication, Sharding, Priority Queue, QoS**		
Resumen	**Cloud patterns are described as good solutions to recurring design problems in a cloud context. These patterns are often inherited from Service Oriented Architectures or Object- Oriented Architectures where they are considered good practices. However, there is a lack of studies that assess the benefits of these patterns for cloud applications. In this paper, we conduct an empirical study on a RESTful application deployed in the cloud, to investigate the individual and the combined impact of three cloud patterns (i.e., Local Database proxy, Local Sharding- Based Router and Priority Queue Patterns) on Quality of Service (QoS). We measure the QoS using the application's response time, average, and maximum number of requests processed per seconds. Results show that cloud patterns doesn't always improve the response time of an application. In the case of the Local Database proxy pattern, the choice of algorithm used to route requests has an impact on response time, as well as the average and maximum number of requests processed per second. Combinations of patterns can significantly affect the QoS of applications. Developers and software architects can make use of these results to guide their design decisions.**		
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación**	**Responde a la pregunta**	**Referencia a tabla de análisis**
	PI1	✓	**PI1-01**
	PI2	✓	**PI2-01**

	PI3	✓	**PI3-01**
Notas	**La respuesta a la PI3 solo se considera tiempo como métrica. Sin embargo, utiliza múltiples escenarios.**		

Documento E4

DATOS GENERALES			
Título	**Poster: Exploration of Academic and Industrial Evidence about Architectural Tactics and Patterns in Microservices**		
Autores	**Osses, F** **Márquez, G** **Astudillo, H**		
Año	**2018**		
Fuente	**IEEE**		
Tipo de publicación	**Conference proceedings**		
Referencia o detalles de la publicación	**DOI: 10.1145/3183440.3194958**		
Palabras clave	**Architectural patterns, architectural tactics, microservices, taxon- omy, systematic literature review, academy, industry**		
Resumen	Microservices are quickly becoming an outstanding architectural choice in the service-oriented software industry. This approach pro- poses to develop each application as a collection of small services, each running on its process and inter-communicating with light- weight mechanisms. Currently, there is still no clear perspective of emerging recurrent solutions (architectural patterns) or design decisions (architectural tactics) in microservices both in industry and academia. This article describes a systematic review of the academic and industrial literature on architectural patterns and tactics proposed for microservices. The study reported: 44 architec- tural patterns of microservices in academia and 80 in the industry; architectural tactics related to microservices dependent on other disciplines; and it was also found that most of architectural patterns and tactics are associated to ve quality attributes: scalability, exi- bility, testability, performance, and elasticity. Added to that results, it was noticed that most microservices in the academic area are reported in evidence related to DevOps and IoT, but the industry is not interested in associating disciplines. Finally, a new proposal of microservices pattern taxonomy is suggested.		
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación**	**Responde a la pregunta**	**Referencia a tabla de análisis**
	PI1	✓	**PI1-10**
	PI2	✓	**PI2-09**
	PI3	✗	
Notas	**Lamentablemente este documento hace referencia a una RSL, la cual no se tuvo acceso.**		

	El documento analizado es muy resumido y carece de detalles.

Documento E6

DATOS GENERALES			
Título	**Guidelines for adopting frontend architectures and patterns in microservices-based systems**		
Autores	**Harms, H** **Rogowski, C** **Lo Iacono, L**		
Año	**2017**		
Fuente	**ACM**		
Tipo de publicación	**Conference proceedings**		
Referencia o detalles de la publicación	https://doi.org/10.1145/3106237.3117775		
Palabras clave	**Frontend architecture, Design patterns, Microservices**		
Resumen	Microservice-based systems enable the independent development, deployment, and scalability for separate system components of enterprise applications. A signi cant aspect during development is the microservice integration in frontends of web, mobile, and desktop applications. One challenge here is the selection of an adequate frontend architecture as well as suitable patterns that satisfy the application requirements. This paper analyses available strategies for organizing and implementing microservices frontends. These approaches are then evaluated based on a quality model and various prototypes of the same application implemented using the distinct approaches. The results of this analysis are generalized to a guideline that supports the selection of a suitable architecture.		
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación**	**Responde a la pregunta**	**Referencia a tabla de análisis**
	PI1	✓	**PI1-03**
	PI2	✓	**PI2-03**
	PI3	✓	**PI3-02**
Notas	**Todo lo relacionado de este documento es sobre la integración de microservicios en el front-end o interfaz de usuario. Además, que su principal tema son arquitecturas de front-end no directamente patrones**		

Documento E7

DATOS GENERALES	
Título	**Highly Scalable Microservice-based Enterprise Architecture for Smart Ecosystems in Hybrid Cloud Environments**

Autores	**Müssig, Daniel** **Stricker, Robert** **Lässig, Jörg** **Heider, Jens**
Año	**2017**
Fuente	**Scitepress**
Tipo de publicación	**Journal article**
Referencia o detalles de la publicación	**http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006373304540459**
Palabras clave	**Cloud, IT-Infrastructure, Scaling, Microservices, Application Container, Security, Authorization Pattern.**
Resumen	Conventional scaling strategies based on general metrics such as technical RAM or CPU measures are not aligned with the business and hence often lack precision flexibility. First, the paper argues that custom metrics for scaling, load balancing and load prediction result in better business-alignment of the scaling behavior as well as cost reduction. Furthermore, due to scaling requirements of structural – non-business– services, existing authorization patterns such as API-gateways result in inefficient scaling behavior. By introducing a new pattern for authorization processes, the scalability can be optimized. In sum, the changes result in improvements of not only scalability but also availability, robustness and improved security characteristics of the infrastructure. Beyond this, resource optimization and hence cost reduction can be achieved.
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ✓ **PI1-04** **PI2** ✓ **PI2-04** **PI3** ✗
Notas	**Directamente solo habla de dos patrones, pero se podrían considerar a otros dos como patrones.** **Fue suprimido API-Key-Distribution por ser considerado muy general. Validado con experto.**

Documento E9

DATOS GENERALES	
Título	**Implementation Patterns for Microservices Architectures**
Autores	**Brown, Kyle** **Woolf, Bobby**
Año	**2016**
Fuente	**ACM**
Tipo de publicación	**Conference proceedings**

Referencia o detalles de la publicación	**https://dl.acm.org/citation.cfm?id=3158170**
Palabras clave	**Microservices, Agile Development, Pattern Languages**
Resumen	**We describe a set of implementation patterns for building applications using microservices. We discuss the application types and requirements that lead to the n eed for microservices, examine different types of microservices, and discuss patterns required for implementing data storage and devops in a microservices environment.**
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ✓ **PI1-05** **PI2** ✓ **PI2-05** **PI3** ✗
Notas	**Es un interesante documento que recolecta patrones para microservicios y patrones que benefician a la integración de un sistema de microservicios.**

Documento E10

DATOS GENERALES	
Título	**Incorporating Security Features in Service-Oriented Architecture using Security Patterns**
Autores	**Dwivedi, Ashish Kumar** **Rath, Santanu Kumar**
Año	**2015**
Fuente	**ACM**
Tipo de publicación	**Journal article**
Referencia o detalles de la publicación	**http://dl.acm.org/citation.cfm?doid=2693208.2693229**
Palabras clave	**Security Patterns, Service Composition, SOA, Web Services.**
Resumen	**Service-Oriented Architecture is an architectural style where dif- ferent heterogeneous components share information with each other by using special types of messages based on the protocol known as Simple Object Access Protocol. Various technologies, such as Common Object Request Broker Architecture, Java 2 Platform, Enterprise Edition, Java Message Service etc. are applied to re- alize Service-Oriented Architecture for different applications. Be- sides these approaches, two other techniques, REpresentational State Transfer, and web services are applied for the realization of Service-Oriented Architecture. Web services provide a plat- form independent communication scheme between applications. The security preservation among the composition of services is an important task for Service-Oriented Architecture. In this study, an attempt is made to incorporate security**

	features in Service- Oriented Architecture with the help of software security patterns. This scheme is described by developing an architectural model integrated with security goals and security patterns. The struc- tural and behavioral aspects of composition of web services in- corporated with security features are presented using a Unified Modeling Language class diagram and a sequence diagram re- spectively. At the end of this study, an evaluation is performed between identified security patterns and critical security proper- ties along with Service-Oriented Architecture design principles. A case study of an online banking system is considered to explain the use of security patterns.
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ✓ **PI1-06** **PI2** ✓ **PI2-06** **PI3** ✗
Notas	**Menciona 6 patrones específicos de seguridad, sin embargo, todo está en el contexto de SOA. Gracias a un experto del área se ha identificado solo a un patrón en el contexto de microservicios. Tendrán que ser validados para microservicios.**

Documento E12

DATOS GENERALES	
Título	**Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications**
Autores	**Torkura, Kennedy A.** **Sukmana, Muhammad I.H.** **Cheng, Feng** **Meinel, Christoph**
Año	**2017**
Fuente	**IEEE**
Tipo de publicación	**Conference proceedings**
Referencia o detalles de la publicación	**http://ieeexplore.ieee.org/document/8118424/**
Palabras clave	**Cloud-Security, Security-as-a-Service, Vulnera- bility Assessment, Cloud Native Applications**
Resumen	This paper discusses a new approach for designing and deploying Security-as-a-Service (SecaaS) applications using cloud native design patterns. Current SecaaS approaches do not efficiently handle the increasing threats to computer systems and applications. For example, requests for security assessments drastically increase after a high-risk

	security vulnerability is disclosed. In such scenarios, SecaaS applications are unable to dynamically scale to serve requests. A root cause of this challenge is employment of architectures not specifically fitted to cloud environments. Cloud native design patterns resolve this challenge by enabling certain properties e.g. massive scalability and resiliency via the combination of microservice patterns and cloud-focused design patterns. However adopting these patterns is a complex process, during which several security issues are introduced. In this work, we investigate these security issues, we redesign and deploy a monolithic SecaaS application using cloud native design patterns while considering appropriate, layered security counter-measures i.e. at the application and cloud networking layer. Our prototype implementation out-performs traditional, monolithic applications with an average Scanner Time of 6 minutes, without compromising security. Our approach can be employed for designing secure, scalable and performant SecaaS applications that effectively handle unexpected increase in security assessment requests.
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ☑ **PI1-07** **PI2** ✗ **PI3** ✗
Notas	**La información es pobre con respecto a patrones.** **Se podría asumir que todos los patrones usados atacan a seguridad ya que es la única prioridad**

Documento E14

DATOS GENERALES	
Título	**Microservices migration patterns**
Autores	**Balalaie, Armin** **Heydarnoori, Abbas** **Jamshidi, Pooyan** **Tamburri, Damian A.** **Lynn, Theo**
Año	**2018**
Fuente	**Wiley**
Tipo de publicación	**Journal article**
Referencia o detalles de la publicación	**http://doi.wiley.com/10.1002/spe.2608**
Palabras clave	**cloud-native architectures, cloud computing, microservices, migration patterns**

Resumen	**Microservices architectures are becoming the defacto standard for building con- tinuously deployed systems. At the same time, there is a substantial growth in the demand for migrating on-premise legacy applications to the cloud. In this context, organizations tend to migrate their traditional architectures into cloud-native architectures using microservices. This article reports a set of migration and rearchitecting design patterns that we have empirically identified and collected from industrial-scale software migration projects. These migration patterns can help information technology organizations plan their migration projects toward microservices more efficiently and effectively. In addition, the proposed patterns facilitate the definition of migration plans by pattern com- position. Qualitative empirical research is used to evaluate the validity of the proposed patterns. Our findings suggest that the proposed patterns are evident in other architectural refactoring and migration projects and strong candidates for effective patterns in system migrations.**

Respuesta a preguntas de investigación	**Referencia a pregunta de investigación**	**Responde a la pregunta**	**Referencia a tabla de análisis**
	PI1	✓	**PI1-08**
	PI2	✓	**PI2-07**
	PI3	✗	

Notas	**Un interesante documento. Sin embargo, está orientado a migraciones de monolíticos a microservicios, varios patrones su intención es solo el proceso de migración.**

Documento E27

DATOS GENERALES	
Título	**Understanding the impact of cloud patterns on performance and energy consumption**
Autores	**Khomh, Foutse** **Abtahizadeh, S. Amirhossein**
Año	**2018**
Fuente	**Elsevier**
Tipo de publicación	**Journal article**
Referencia o detalles de la publicación	**https://doi.org/10.1016/j.jss.2018.03.063**
Palabras clave	**Cloud patterns Energy consumption Performance optimization Energy efficiency**
Resumen	**Cloud patterns are abstract solutions to recurrent design problems in the cloud. Previous work has shown that these patterns can improve the Quality of Service (QoS) of cloud applications but their impact on en- ergy consumption is still unknown. In this work, we conduct an empirical study on two multi-processing and multi-threaded**

	applications deployed in the cloud, to investigate the individual and the combined impact of six cloud patterns (Local Database Proxy, Local Sharding Based Router, Priority Queue, Compet- ing Consumers, Gatekeeper and Pipes and Filters) on the energy consumption. We measure the energy consumption using Power-API; an application programming interface (API) written in Java to monitor the energy consumed at the process-level. Results show that cloud patterns can effectively reduce the energy consumption of a cloud-based application, but not in all cases. In general, there appear to be a trade-off between an improved response time of the application and the energy consumption. Moreover, our find- ings show that migrating an application to a microservices architecture can improve the performance of the application, while significantly reducing its energy consumption. We summarize our contributions in the form of guidelines that developers and software architects can follow during the implementation of a cloud-based applicaton.
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ✓ **PI1-09** **PI2** ✓ **PI2-08** **PI3** ✓ **PI3-03**
Notas	**Interesante documento sobre el consumo de energía y su relación con el uso de patrones. Además, describe cómo fueron implementados los patrones y las reglas que ajustaron.** **Sobre el consumo de energía:** • Gatekeeper, the Competing Consumers pattern improves performance and energy efficiency only when the application follows a microservices architectural style • The Pipes & Filters pattern can have a positive impact on both performance and energy consumption • Local Database Proxy patten with Priority Message Queue has a negative impact on the average response time • Local Sharding-Based Router, Pipes and Filters, and Competing Consumers patterns […] with little impact on energy efficiency, the per- formance can be penalized • Gatekeeper is added to an appli- cation, both response time and energy efficiency can be affected • Migrating an application to a microservices architecture can improve the performance of the application, while significantly reducing its energy consumption

Documento E28

DATOS GENERALES	
Título	**Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges**

Autores	**Justus Bogner,** **Jonas Fritzsch,** **Stefan Wagner,** **Alfred Zimmermann**
Año	**2019 (September 30-Oct 4)**
Fuente	**IEEE**
Tipo de publicación	**Conference paper**
Referencia o detalles de la publicación	**DOI 10.1109/ICSME.2019.00089**
Palabras clave	**Microservices, interviews, industry, evolvability, assurance**
Resumen	**While Microservices promise several beneficial char- acteristics for sustainable long-term software evolution, little empirical research covers what concrete activities industry ap- plies for the evolvability assurance of Microservices and how technical debt is handled in such systems. Since insights into the current state of practice are very important for researchers, we performed a qualitative interview study to explore applied evolvability assurance processes, the usage of tools, metrics, and patterns, as well as participants' reflections on the topic. In 17 semi-structured interviews, we discussed 14 different Microservice-based systems with software professionals from 10 companies and how the sustainable evolution of these systems was ensured. Interview transcripts were analyzed with a detailed coding system and the constant comparison method.** **We found that especially systems for external customers relied on central governance for the assurance. Participants saw guidelines like architectural principles as important to ensure a base consistency for evolvability. Interviewees also valued manual activities like code review, even though automation and tool support was described as very important. Source code quality was the primary target for the usage of tools and metrics. Despite most reported issues being related to Architectural Technical Debt (ATD), our participants did not apply any architectural or service-oriented tools and metrics. While participants generally saw their Microservices as evolvable, service cutting and finding an appropriate service granularity with low coupling and high cohesion were reported as challenging. Future Microservices research in the areas of evolution and technical debt should take these findings and industry sentiments into account.**
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** / ✓ / **PI1-11**

	PI2	☑	**PI2-10**
	PI3	✗	
Notas	**Relaciona parcialmente a todos los patrones con «evolución». Sí menciona los patrones pero describe muy poco sobre ellos. No son muchos.**		

Documento E29

DATOS GENERALES	
Título	**Actual Use of Architectural Patterns in Microservices-Based Open Source Projects**
Autores	**Gastón Márquez** **Hernán Astudillo**
Año	**2018 (December)**
Fuente	**IEEE**
Tipo de publicación	**Conference paper**
Referencia o detalles de la publicación	**DOI 10.1109/APSEC.2018.00017**
Palabras clave	**Microservices, architectural patterns, open source projects, quality attributes, framework**
Resumen	**Microservice-based systems instantiate an architec- tural style that conceives of systems as sets of modular, customer- centric, independent, and scalable services. These systems express a similar essential structural organization and seems appropriate to design them using architectural patterns because these com- bine an understanding of the system domain and good practices. Code repository platforms provide the developer community with ideas and examples about microservice systems, but since they are in early adoption, there is still no clear notion of which actual microservice systems incarnate architectural patterns (if any), reducing the use of frameworks and the achievement of quality attributes. This paper extends a previous study on architectural patterns for microservices in academic and industry sources. We explored which architectural patterns for microservices are used in actual microservice-based open source systems, by subjecting thirty well-known open source projects to a comprehensive multi-criteria code and design review. We found that (1) open source projects use only a few architectural patterns broadly; (2) most projects use the same few frameworks; (3) there are very few microservice architectural patterns as such; and (4) what most projects use (what was previously called) are SOA patterns. This study shows that microservice systems builders do use architectural patterns, but only a few of them. It remains to be determined whether additional patterns would be productively used to build microservice systems, or the few ones currently used are the only ones actually necessary.**

Respuesta a preguntas de investigación	Referencia a pregunta de investigación	Responde a la pregunta	Referencia a tabla de análisis
	PI1	✓	PI1-12
	PI2	✓	PI2-11
	PI3	✗	
Notas	Es un buen trabajo, aborda de forma eficiente los otros dos puntos. No aporta nada con métricas.		

Documento E30

DATOS GENERALES	
Título	Identifying Availability Tactics to Support Security Architectural Design of Microservice-based Systems
Autores	Gastón Márquez Hernán Astudillo
Año	2019 (September 9-13)
Fuente	ACM
Tipo de publicación	Conference paper
Referencia o detalles de la publicación	https://doi.org/10.1145/3344948.3344996
Palabras clave	Microservices, availability, architectural tactics, frameworks, patterns
Resumen	Microservices is an architectural style that considers systems as modular, customer-centric, independent, and scalable suite of services. In order to address security requirements in microservices- based systems, architects often must focus on critical quality at- tributes, such as availability, aiming at employing architectural solutions that provide design decisions that address key security concerns (also known as architectural tactics). Although current architectural tactics for availability offer an extensive catalog of al- ternatives to improve availability and security factors, new availabil- ity concerns (emerging from security microservices requirements) demand new or improved architectural tactics. In this article, we examined the source code and documentation of 17 open source microservices-based systems, identified 5 uses of availability tac- tics, and characterized them using a newly introduced descriptive template. We found that almost all (4 out of 5) tactics did focus on preventing faults rather than detecting, mitigating or recovering from them (which are the traditional tactics taxonomies' branches). This approach can be further used to systematically identify and characterize architectural tactics in existing

	microservices-based systems in other critical quality attributes concerning security, such as confidentiality and integrity.
Respuesta a preguntas de investigación	**Referencia a pregunta de investigación** / **Responde a la pregunta** / **Referencia a tabla de análisis** **PI1** ✓ **PI1-13** **PI2** ✓ **PI2-12** **PI3** ✗
Notas	**Solo menciona 3 directamente como patrones. Menciona otros pero como TÁCTICAS.** **Dirigido solo a disponibilidad.**

Tablas de PI1

¿Qué patrones se identifican en los sistemas de microservicios?

PI1-01

PI1- Análisis de la información	
Identificador: PI1-01	
¿Cuáles son los patrones identificados?	**Local Database Proxy, Local Sharding-Based Router and Priority Queue Patterns.**
¿De qué manera fueron identificados los patrones?	**identified in the literature**
¿Qué ventajas se describen de dichos patrones?	**Local Database Proxy:** ▪ provides a read scalability on a relational database ▪ Microsoft provided guidelines for the replication in a cloud application ▪ Is a good design solution for applications experiencing heavy loads of read requests **Local Sharding-Based Router:** ▪ Recommended when the need for scalability concerns read and write operations ▪ Is more adequate for applications handling huge write requests loads **Priority Queue Patterns:** ▪ to allow asynchronous communications between components

	▪ to design loosely coupled components and to improve scalabilit ▪ A moderate effect on both (Proxy and Sharding)
¿Qué desventajas se describen de dichos patrones?	**Local Database Proxy:** ▪ components must use a local proxy whenever they need to retrieve or write data **Local Sharding-Based Router:** ▪ each split must be independent as much as possible ▪ The response time is not lower on read requests **Priority Queue Patterns:**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**A combination of the Priority Message Queue pattern with Local Database proxy or Local Sharding- Based Router patterns can improve the QoS of an application experiencing heavy loads of read and write requests.** **-Sin embargo, no tienen dependencia**

PI1-02

PI1- Análisis de la información	
Identificador: PI1-02	
¿Cuáles son los patrones identificados?	**Asynchronous Query, service locator, asynchronous completion token, event notification**
¿De qué manera fueron identificados los patrones?	**our expertise and to literature sources**
¿Qué ventajas se describen de dichos patrones?	**Solo se describen los patrones en cuestión de atributos de calidad. Ver PI2-02**
¿Qué desventajas se describen de dichos patrones?	**Solo se describen los patrones en cuestión de atributos de calidad. Ver PI2-02**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No documentado de manera explícita**

PI1-03

PI1- Análisis de la información	
Identificador: PI1-03	
¿Cuáles son los patrones identificados?	**API Gateway** **Backend for frontends**
¿De qué manera fueron identificados los patrones?	**De manera implícita de la literatura**
¿Qué ventajas se describen de dichos patrones?	**API Gateway:**

	• request shaping, caching, authentication, monitoring, and load balancing • changes in one microservice can easily lead to redeployments of all client types **Backend for frontends:** • prevent the gateway from having too much logic to handle request shaping for di erent client types, it can be divided into mul- tiple gateways • avoids a general middleware and development teams can implement the frontend connections on this application level more independently
¿Qué desventajas se describen de dichos patrones?	**API Gateway:** • a single entry point to systems • It is not the task of the gateway to integrate the microservices in the frontend • the test results also show that a high load on a gateway can have a negative impact **Backend for frontends:** **No menciona nada de manera explícita**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No de manera explícita, sin embargo, fueron probados para frontend**

PI1-04

PI1- Análisis de la información	
Identificador: PI1-04	
¿Cuáles son los patrones identificados?	**Load balancer, API Gateway, auth-service**
¿De qué manera fueron identificados los patrones?	**We investigate some common authorization principles concerning the compatibility with our pre- sented infrastructure. *** ***No especifican si fue mediante un experto o literatura**
¿Qué ventajas se describen de dichos patrones?	**Load balancer:** • can handle requests with different durations much better than other approaches such as round robin. **API Gateway:** • The real interface addresses (URLs) of the microservices can be hidden, injection inspection or input validation

	• The real interface addresses […] can be realized equivalently for each service. **Auth-service:** • separation of sensitive user data from the open interfaces • the less exten- sive functionality of the auth-service compared to an API Gateway • improves the authorization pro- cess concerning response time
¿Qué desventajas se describen de dichos patrones?	**Load balancer:** • With increasing number of requests the load balancer has to be scaled. **API Gateway:** • But for the authorization process this service requires also a database con- taining data for all services, which creates vulnerabilities. **Auth-service:** • The generation of traffic from the service • If an opponent sends many requests during a (D)Dos attack, the pat- tern supports him by multiplying each request. • The worst aspect of these designs is the dependency between the microservices and the management ser- vices from the count of requests
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No**

PI1-05

PI1- Análisis de la información	
Identificador: PI1-05	
¿Cuáles son los patrones identificados?	**Backend for Frontend, adapter microservice, results cache, page cache, Key-Value store, log aggregator, correlation ID, service registry.**
¿De qué manera fueron identificados los patrones?	**No especificado, pero se podría asumir que mediante la literatura**
¿Qué ventajas se describen de dichos patrones?	**Backend for Frontend:** • Implement different BFFs for different types of clients • It can orchestrate several calls to business microservices that result from a single client action • It can translate the results of a microservice into a channel-specific representation that

	more cleanly maps to the needs of the user experience of that client • It can filter results from a business microservice that are not needed by a particular client type **Adapter microservice:** • converts the existing service's non-microservice API to an API that client microservies will expect **Results cache:** • can be as simple as a Key-Value Store (either in-memory or in a Scalable Store). **Page cache:** • For very long datasets, a Page Cache is preferable to fetching all the data **Key-Value store:** • is its simplicity **Log aggregator:** • will "listen for" or tail each individual log file and forward the log entries to an aggregated collection point as they are made. **Correlation ID:** • is a simple identifier • allows you to match or correlate specific requests to one service to other service requests in the same call chain. **Service registry:** • to map between a unique identifier and the current address of a service instance in order to decouple the physical address of a service from the identifier
¿Qué desventajas se describen de dichos patrones?	**Backend for Frontend:** • there is a risk that business logic becomes embedded in the BFF instead of within the underlying business microservices where it should be implemented **Adapter microservice:** • only last until the underlying existing services can be replaced by a natively implemented microservice. **Results cache:** • how to keep it from becoming stale **Page cache:** • particularly on a mobile application **Key-Value store:**

	• more complex store type such as a Document store for storing cache entries, then you would find that the performance of such solutions is often not as good **Service registry:** • the setup and management of the registry infrastructure is often more trouble than it's worth
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**Backend for Frontend*:** • often use Page Caches to store long results obtained • Service Registry may help the client code that makes up the bulk of a Backend for Frontend to be resilient in the face of changes to the physical address **Adapter microservice*:** • often use Results Caches in order to reduce the number of times they have to invoke the underlying SOA service that they convert. **Results cache*: key-value- store, scalable store** **Page cache: Backend for Frontend** **Key-Value store*: Correlation ID** **Correlation ID*: Log aggregator** ***Sin embargo, no son obligatorios**

PI1-06

PI1- Análisis de la información	
Identificador: PI1-06	
¿Cuáles son los patrones identificados?	**Secure Channel**
¿De qué manera fueron identificados los patrones?	**Literatura e identificación en aplicaciones**
¿Qué ventajas se describen de dichos patrones?	• security goals are achieved • security properties violate SOA design principles, such as loose coupling, service contract, abstraction, reusability, composability, discoverability, granular- ity, extendability, vendor diversity, statelessness and autonomy. Hence, those patterns need to be considered that can be used to minimize these limitations.
¿Qué desventajas se describen de dichos patrones?	**performance may be degraded**

¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No de manera explícita. En el documento se encuentran relacionados porque han sido analizados para cumplir una meta. Por lo tanto, la combinación de ellos se ajusta. Sin embargo, los patrones podrían ser intercambiables**

PI1-07

PI1- Análisis de la información	
Identificador: PI1-07	
¿Cuáles son los patrones identificados?	**load-balancing, service discovery, externalized configuration and API gateway**
¿De qué manera fueron identificados los patrones?	**No lo dice de manera explícita, pero parece que fueron identificados en la literatura**
¿Qué ventajas se describen de dichos patrones?	**Service Discovery:** • enables deployed service instances locate themselves. • register specific metadata • performs periodic health checks on registered services **API Gateway:** • receives all incoming requests and forwards them to the appropriate services
¿Qué desventajas se describen de dichos patrones?	**El documento habla de los inconvenientes de su propuesta, pero no de los patrones**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No de manera explícita. El documento explica su interacción, pero no describe una dependencia.**

PI1-08

PI1- Análisis de la información	
Identificador: PI1-08	
¿Cuáles son los patrones identificados?	**Service registry, service registry client, internal load balancer, external load balancer, circuit breaker, edge server**
¿De qué manera fueron identificados los patrones?	**Literatura y 3 aplicaciones en donde los autores estuvieron involucrados**
¿Qué ventajas se describen de dichos patrones?	**Service registry:** • Store service instance's addresses • Locate each other (services) dynamically **Service registry client:** • periodic heartbeat should be sent […] to keep the instance in the available instances list

	• Instance removal from the service registry can be done through either not sending the heartbeat anymore **Internal load balancer:** • fetches the list of available instances of a desired service • balance the load between the available instances using local metrics • the possibility of having different load balancing mechanisms in different clients **External load balancer:** • retrieves the list of available instances from the service registry • uses a centralized algorithm for balancing the load between instances **Circuit breaker:** • a system fail fast and not wait until reaching a service call timeout • more resilient when an unavailable service is called **Edge server:** • internal service structure complexity and evolution be hidden • can do dynamic routing • the internal structure changes would not affect them and will be handled through new routing rules • best place to monitor the overall usage of services
¿Qué desventajas se describen de dichos patrones?	**Service registry:** • The rest of the system is coupled to this component • could become a single point of failure **Service registry client:** • the client should be implemented for all of the programming languages **Internal load balancer:**

	• need to create an internal load balancer for different programming languages in use • load balancing mechanism is not centralized **External load balancer:** • local metrics, eg, the response time of the instances, cannot be used to improve the load balancing **Circuit breaker:** • Recognizing the appropriate response in the open circuit state **Edge server:** • single point of failure • this layer should be replicated through load balancing mechanisms
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No de manera directa. Algunos se pueden beneficiar pero no son dependencias.**

PI1-09

PI1- Análisis de la información	
Identificador: PI1-09	
¿Cuáles son los patrones identificados?	**Local Database Proxy, Local Sharding- Based Router, Priority Queue, Competing Consumers, Gatekeeper y Pipes and Filters**
¿De qué manera fueron identificados los patrones?	**Literatura**
¿Qué ventajas se describen de dichos patrones?	**Local Database Proxy:** • Slaves may be added or removed during the execution to obtain elasticity • uses data replication between master/slave databases and a proxy to route requests • Apto para múltiples consultas de lectura **Local Sharding- Based Router:** • useful when an application needs scalability both for read and write operations • The sharding logic is applicable through multiple strategies; a range of value, a specific shard key or hashing can be used to distribute data among the databases • balancing the workload across shards

	Priority Message Queue: • allow asynchronous communications between components • is recommended when there are different types of messages **Competing Consumers:** • Allows applications to handle fluctuating workloads (from idle times to peak times), by deploying and coordinating multiple instances of the consumer service • It guarantees that a failed service instance will not result in blocking a producer • Instances of a consumer service can be dynamically added or removed **Gatekeeper:** • It processes and directs requests to trusted messages on another in- stance(s) called Trusted Host • decouple application instances from storage, ensuring that trusted hosts connect only to the gatekeeper(s) and not directly to clients • suitable for applications that handle sensitive/protected information **Pipes and Filters:** • recommends to decompose the processing into a set of discrete components • improves resiliency because if a task is failed, it can be rescheduled
¿Qué desventajas se describen de dichos patrones?	**Local Database Proxy:** • must use a local proxy whenever they need to retrieve or write data • could be a risk of bottleneck on the master database when there is a need to scale with write requests **Local Sharding- Based Router:** • each split must be inde- pendent as much as possible to avoid joins **Priority Queue:** **Competing Consumers:**

	Gatekeeper: • The Trusted Host holds the neces- sary code and security measures required • A secure communication channel (HTTPS, SSL, or TLS) must be employed **Pipes and Filters:** • the fact that the time required to pro- cess a single request depends on the speed of the slowest filter in the pipeline • a risk that one or more filters form a bottleneck • Deployment automation and testing of pipe and filters architectures can be complex
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**Competing consumers: message queue as the communication channel**

PI1-10

PI1- Análisis de la información	
Identificador: PI1-10	
¿Cuáles son los patrones identificados?	**Backend for Frontend, Microservices DevOps, Service Registry, Change code dependency to service call, Deploy cluster and orchestrate containers, Microservices Architecture*, Self-containment of services, Container, and REST Integration** ***Los autores lo clasifican como patrón, difiero.**
¿De qué manera fueron identificados los patrones?	**In academia and in industry (literatura)**
¿Qué ventajas se describen de dichos patrones?	**Backend for Frontend:** • Acts as a single API for a client • Different BFFs for differents types of cliens • API customized to what the client type needs **Microservices DevOps:** • Allows you to isolate each microservice as much as posible • Able to easily and quickly identify and resolve issues • To set guidelines on how microservices interrelate and interact **Service Registry:**

	• Decouple the physical address of a service from the ideitifier **Change code dependency to service call:** • Share that piece of funcionality as a service, could be either a completely isolate service or a part of one of the dependent • They could be scaled independent of each other **Deploy cluster and orchestrate containers:** • Can manage a cluster of computing nodes • Should be able to deply the services container images on-demand • It should handle the failure of instances and restart the failed nodes • It should provide a mean for autoscaling **Microservices architecture:** **Self-containment of services:** • Providing the required libraries together with the service makes the deployment much easier • Better decoupling between services • Increase in autonomy • Reduces the amount of requiered coummunication **Container:** • Enclose the microservice itself (libraries and data) • Supports the requierement of self-containment • Better testability, eas of service deployment, better scalability **REST Integration:** • Dedicated REST endpoint that services can use to notify them of the death of a patient
¿Qué desventajas se describen de dichos patrones?	**No descritas**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**Deploy cluster and orchestrate containers: could be a good feature […] service Discovery*** ***Sin embargo, no dice que dependa**

PI1-11

PI1- Análisis de la información
Identificador: PI1-01

¿Cuáles son los patrones identificados?	**Event-driven messaging** **Service registry** **Strangler** **Backend for frontend** **Consumer-driven contracts** **Tolerant reader** **API Gateway** **Request-reaction** **Self-contained systems** **Event sourcing**
¿De qué manera fueron identificados los patrones?	**Semi-structured interviews to profesionals (industry)**
¿Qué ventajas se describen de dichos patrones?	**Event-driven messaging** **-Decouple services** **-Implement reliable asynchronous and long-running communication** **Service registry** **-For dynamic service discovery** **Strangler** **-Extend an existing monolith with new microservices until its final replacement** **Backend for frontend** **-To place an intermediary between service consumers and producers** **-To prevent too many concurrent long-running http requests** **Consumer-driven contracts** **-To make service interface evolution more robust and to prepare consumers for future changes** **Tolerant reader** **-To make service interface evolution more robust and to prepare consumers for future changes** **API Gateway** **-Improve security** **Request-reaction** **-Sometimes paired with Event-driven messaging** **Self-contained systems**

	-To achieve vertical isolation between subsystems **Event sourcing** **-No información fue encontrada. Sin embargo, se clasificó gracias a la descipción en que fue citada en E28 https://microservices.io/patterns/data/event-sourcing.html**
¿Qué desventajas se describen de dichos patrones?	**No descritas.**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No aunque describe que a veces se mezcla request-reaction con event-driven messaging**

PI1-12

PI1- Análisis de la información	
Identificador: PI1-02	
¿Cuáles son los patrones identificados?	<table><tr><th>architectural pattern</th><th>SOA</th><th>Microservices</th></tr><tr><td>API Gateway</td><td></td><td>☐</td></tr><tr><td>Load balancer</td><td>☐</td><td>☐</td></tr><tr><td>Container</td><td></td><td>☐</td></tr><tr><td>Key-value store</td><td>☐</td><td>☐</td></tr><tr><td>Log aggregator</td><td></td><td>☐</td></tr><tr><td>Messaging</td><td>☐</td><td>☐</td></tr><tr><td>Service registry</td><td>☐</td><td>☐</td></tr><tr><td>Database is the service</td><td></td><td>☐</td></tr><tr><td>Enable Cont. Integration</td><td></td><td>☐</td></tr><tr><td>Circuit breaker</td><td>☐</td><td>☐</td></tr><tr><td>Results cache</td><td></td><td>☐</td></tr><tr><td>Monitor</td><td>☐</td><td>☐</td></tr><tr><td>Service discovery</td><td>☐</td><td>☐</td></tr><tr><td>Page cache</td><td></td><td>☐</td></tr><tr><td>BackEnd for FrontEnd</td><td></td><td>☐</td></tr><tr><td>Scalable store</td><td></td><td>☐</td></tr><tr><td>Health check</td><td>☐</td><td>☐</td></tr></table>
¿De qué manera fueron identificados los patrones?	**Academy and industry**

¿Qué ventajas se describen de dichos patrones?	• **API Gateway**
	  ○ **Hide the service location**
	  ○ **Expose a unified endpoint for clients**
	• **Load balancer**
	  ○ **Distributes workload on a set of equal services**
	• **Container**

	○ **Simplify deploying applications** ○ **Enclose all required data.** ○ **Better testability and scalability** • **Key-value store** ○ **Simplicity. Works like a hashmap** • **Log aggregator** ○ **Merge different log file into one** • **Messaging** ○ **Asynchronous messaging for inter-service communication** • **Service registry** ○ **Pull all services locations in one place** • **Data base is the service** ○ **Each service has its own DB** • **Enable cont. integration** ○ **Automate the process. Introduces production ready artifacts. Pipeline from each service repository to test and to build.** • **Circuit breaker** ○ **Monitor the recent responses and will act when the number of failures passes a predefined threshold. At specific timeout it will check the services availability, in case of a successful the state will be changed** • **Results cache** ○ **Shortcuts the need of multiple calls** • **Monitor** ○ **A monitory facility to gather important information and send it monitoring "server"** • **Service discovery** ○ **Store service instances. Each service registers itself during initiation. Other instances can call the service discovery to get a services location.** • **Page cache** ○ **Allow the return to the client more information (subset of data). This information can be indexed and displayed later on the device** • **Back end for frontend** ○ **Implement customized different single API for different type of clients. Customized neeeds.**

	• Scalable store
	  ○ Horizontal scalability and survives to failures. Put all states in store and distribuite it. Related to NoSQL DB.
	• Health check
	  ○ Track service state implemented by a ping or related to a monitoring service. Exposed by an API that returns the health.

¿Qué desventajas se describen de dichos patrones?	**No se especifica.**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No se especifica.** **Load balancer can be improved by circuit breaker or health check, only services with good health are used.** **Backend for frontend (mobil) with page cache, return a larger subset of data improving the scrolling.**

PI1-13

PI1- Análisis de la información	
Identificador: PI1-03	
¿Cuáles son los patrones identificados?	**Circuit breaker** **Service registry** **Messaging**
¿De qué manera fueron identificados los patrones?	**Conducting literatura reviews in academic and industrial sources (open MSA systems)**
¿Qué ventajas se describen de dichos patrones?	**Circuit breaker** **-Checks health status by unsuccesful calls. Prevent additional requests to the broken service.** **Service registry** **-Maps between a unique identifier and the current address of the service, decouple the physical location.** **Messaging** **-Asynchronous messaging inter-service. Message queu allows the asynchronous. And reliably sent to diffent locations**
¿Qué desventajas se describen de dichos patrones?	**No mencionadas.**
¿El patrón necesita de otro patrón complementario? En caso de serlo, describir.	**No mencionado.**

Tablas de PI2

¿Qué atributos de calidad son beneficiados en los patrones utilizados en microservicios?

PI2-01

PI2- Análisis de la información	
Identificador: PI2-01	
¿Qué atributo de calidad se relaciona con el patrón?	**Local Database Proxy: scalability and availability** **Local Sharding-Based Router: scalability** **Priority Queue Patterns: scalability**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No de manera explícita**

PI2-02

PI2- Análisis de la información	
Identificador: PI2-02	
¿Qué atributo de calidad se relaciona con el patrón?	**Asynchronous Query:** • High performance **Service locator:** • Security **Asynchronous completion token:** • Performance **Event notification:** • Performance
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**Asynchronous Query:** • Low interoperability **Service locator:** • Performance **Asynchronous completion token:** • Modifiability **Event notification:** • Security

PI2-03

PI2- Análisis de la información	
Identificador: PI2-03	
¿Qué atributo de calidad se relaciona con el patrón?	**API Gateway: availability (load stages)**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo es**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**API Gateway: can have a negative impact on the interoperability and performance of the UI service.**

PI2-04

PI2- Análisis de la información	
Identificador: PI2-04	
¿Qué atributo de calidad se relaciona con el patrón?	**Auth-Service: Performance (response time)** **API Gateway: Seguridad**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**Auth-Service: Seguridad y modificabilidad**

PI2-05

PI2- Análisis de la información	
Identificador: PI2-05	
¿Qué atributo de calidad se relaciona con el patrón?	**Backend for frontend: Interoperabilidad** **Adapter microservices: Interoperabilidad** **Results cache: Performance** **Page cache: Performance** **Key-Value store: Consistency and Availability** **Correlation ID: Testabilidad** **Service registry: Portabilidad**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**Key-Value store: Partition tolerance**

PI2-06

PI2- Análisis de la información

Identificador: PI2-06	
¿Qué atributo de calidad se relaciona con el patrón?	**Secure Channel:** • Confidentiality, Integrity
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo podría ser de manera específica**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**Performance**

PI2-07

PI2- Análisis de la información	
Identificador: PI2-07	
¿Qué atributo de calidad se relaciona con el patrón?	**Service registry: Scalability, high availability and dynamicity** **Service registry client: Scalability, high availability, and dynamicity** **Internal load balancer: Scalability, high availability and dynamicity** **External load balancer: Scalability, high availability and dynamicity** **Circuit breaker: Fault tolerance and high availability** **Edge server: Modifiability and dynamicity**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No está documentado**

PI2-08

PI2- Análisis de la información	
Identificador: PI2-08	
¿Qué atributo de calidad se relaciona con el patrón?	**Local Database Proxy: scalability (read)** **Local Sharding- Based Router: scalability (read and write) and performance** **Priority Queue: scalability** **Competing Consumers: maintainability** **Gatekeeper: security** **Pipes and Filters: performance and scalability**

¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No documentado de manera directa**

PI2-09

PI2- Análisis de la información	
Identificador: PI2-09	
¿Qué atributo de calidad se relaciona con el patrón?	**Container: testability, scalability**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo son**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No documentado de manera directa**

PI2-10

PI2- Análisis de la información	
Identificador: PI2-01	
¿Qué atributo de calidad se relaciona con el patrón?	**Parcialmente Evolvability en todos. (Maintainability and portability)**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Tal vez no es un atributo muy discutido.** **"However, problems related to architecture and the data model were reported as serious threats for long-term evolvability "** **"finding the appropriate service granularity was a prevalent theme and service cutting was by far named as the most challenging activity"**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No descrito.**

PI2-11

PI2- Análisis de la información	
Identificador: PI2-01	

¿Qué atributo de calidad se relaciona con el patrón?	• **API Gateway**
	  ○ **Maintainability**
	  ○ **Availability (Reliability)**
	• **Load balancer**

	Maintainability **Availability (Reliability)** • **Container** **Maintainability** **Scalability (Maintainability)** **Availability (Reliability)** • **Key-value store** **Availability (Reliability)** • **Log aggregator** **Performance** **Observability (Maintainability?)** • **Messaging** **Availability (Reliability)** • **Service registry** **Availability (Reliability)** • **Data base is the service** **Scalability (Maintainability)** **Availability (Reliability)** • **Enable cont. integration** **Maintainability** • **Circuit breaker** **Reliability** **Availability (Reliability)** • **Results cache** **Performance** • **Monitor** **Observability (Maintainability?)** • **Service discovery** **Availability (Reliability)** • **Page cache** **Performance** • **Back end for frontend** **Scalability (Maintainability)** • **Scalable store** **Scalability (Maintainability)** **Availability (Reliability)** • **Health check** **Observability (Maintainability?)**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Sí**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No documentado**

PI2-12

PI2- Análisis de la información	
Identificador: PI2-01	
¿Qué atributo de calidad se relaciona con el patrón?	**Availability (Reliability) (todos los mencionados en este artículo)**
¿El atributo de calidad no es propio de los sistemas de microservicios?	**Lo es.**
¿El uso de dicho patrón arquitectónico limita o afecta a otro atributo de calidad?	**No descrito.**

Tablas de PI3

¿Qué métrica es utilizada para cuantificar un atributo de calidad en un patrón dentro de un sistema de microservicios?

PI3-01

PI3- Análisis de la información	
Identificador: PI3-01	
¿Qué métrica fue utilizada en dicho atributo de calidad?	**Measured the response time (milliseconds) and the number of transactions per second**
¿La métrica sobre dicho atributo de calidad representa un alto grado de dificultad?	**Mann-Whitney U test to test H R_x^1 ...**
Describir la métrica empleada	**response time and amount of queries executed per second. The result is a tri-dimensional comparison between response time, average number of queries and maximum number of queries executed per second. These measures were taken by the test application itself during every experimentation. The response time measured in these experiments is the overall response time of the application when executing all the queries. This metric is measured in milliseconds. We choose these metrics because it reflects the capacity of the application to scale with the number of requests. We are only considering results where all the request are processed successfully.**
¿La evaluación del atributo de calidad difiere a una	**No**

métrica? Describir en caso de serlo.	
¿La métrica empleada es una adaptación de una métrica bien conocida en otra arquitectura? Describir en caso de serlo.	**No**

PI3-02

PI3- Análisis de la información	
Identificador: PI3-02	
¿Qué métrica fue utilizada en dicho atributo de calidad?	**Time, porcentaje de aceptación (interoperabilidad), número de archivos a modificar (modificabilidad)**
¿La métrica sobre dicho atributo de calidad representa un alto grado de dificultad?	**No**
Describir la métrica empleada	**The stimulus is a request from the UI service to the Customer microservice, which queries customer data. In the test scenario, the Customer microservice is not available and it is measured whether the UI service displays evasion content. The performance test cases determine the time to interact (TTI), i.e. the time span before the user can interact with the GUI.** **in relation to interoperability, only the SCS prototypes and the prototype for the UI monolith returned the expected result without timeouts in the low load scenario as well as in the high load scenario.** **illustrate that SCSs require the modi cation of fewer artifacts and les for a certain feature implementation.**
¿La evaluación del atributo de calidad difiere a una métrica? Describir en caso de serlo.	**No**
¿La métrica empleada es una adaptación de una métrica bien conocida en otra arquitectura? Describir en caso de serlo.	**No, sin embargo, se basa en ATAM**

PI3-03

PI3- Análisis de la información	
Identificador: PI3-03	
¿Qué métrica fue utilizada en dicho atributo de calidad?	**Performance (Response time) and energy consumption**
¿La métrica sobre dicho atributo de calidad representa un alto grado de dificultad?	**Performance= milliseconds** **Energy consumption= kilojoules**
Describir la métrica empleada	**Mann-Whitney U test y the Cliff's δ effect size.** **Para obtener los valores de los escenarios**
¿La evaluación del atributo de calidad difiere a una métrica? Describir en caso de serlo.	**No**
¿La métrica empleada es una adaptación de una métrica bien conocida en otra arquitectura? Describir en caso de serlo.	**No**