Migration of Monoliths to Microservices

This survey is part of ongoing research on how refactoring towards a microservice architecture is currently done by professionals. We are studying what processes are followed, what tools are used and how decomposition results are evaluated.

Please answer it if you have been involved in at least one migration of a monolith to a microservices architecture. The expected time to answer is under 30 minutes.

* Indica uma pergunta obrigatória

Confidentiality Statement

By collaborating on this research, you understand that the data collected through this form:
1. Includes no personally identifying information.
2. Will be used by the researchers exclusively for the purpose of scientific inquiry.
3. May be published by the researchers (e.g., in journals, conferences, or blog posts).
4. Will go through additional anonymization and aggregation steps before being published.
5. Will be kept by the researchers in perpetuity and can used for future academic studies.
6. Is collected using Google Forms and, therefore, its collection and use is subject to Google's Privacy Policy (policies.google.com/privacy).

Authors

The research is being conducted by Rita Peixoto, Filipe Correia and Tiago Boldt Sousa (University of Porto), Jonas Fritzsch and Justus Bogner (University of Stuttgart), Nour Ali (Brunel University London), Eduardo Guerra (Free University of Bozen-Bolzano) and Cesare Pautasso (University of Lugano). If you have any questions, please get in touch with Rita Peixoto and Filipe Correia (up201806257@g.uporto.pt, filipe.correia@fe.up.pt).

Sources

Some of the materials used in this survey are based on (and sometimes paraphrase) the book "*Monolith to Microsevices: Evolutionary Patterns to Transform Your Monolith*", by Sam Newman. O'Reilly Media, 2019.

1. Experience and Background

1. 1.1 What areas have you been working on these past 5 years? *
   Tick all that apply.

   Marcar tudo o que for aplicável.

   - [ ] Software Development
   - [ ] Software Architecture
   - [ ] Operations
   - [ ] Quality assurance
   - [ ] Product Management
   - [ ] Coaching
   - [ ] Teaching
   - [ ] Scientific Research
   - [ ] Outra: _____

2. 1.2. What is your title? *
   E.g., Senior Developer, Architect, Principal Software Engineer, Tester, etc.

   _____

3. 1.3. Which country are you working from? *

   _____

4. 1.4. What is your professional experience in Microservices (in years)? *

   _____

5. 1.5. How many projects that involved the migration of monoliths to microservices have you been involved in? *

   _____

6. 1.6. What were the domain areas of these projects? *

Marcar tudo o que for aplicável.

- [ ] Healthcare
- [ ] Finance
- [ ] Retail
- [ ] Social Media
- [ ] Security
- [ ] Manufacturing
- [ ] Education
- [ ] Travel and Tourism
- [ ] Insurance
- [ ] Construction
- [ ] Real Estate
- [ ] Supply Chain Management
- [ ] Automotive
- [ ] Outra: _____

7. 1.7. Roughly how many monthly active users did these systems serve when the process of migrating to microservices was started? *

Marcar apenas uma oval.

- ◯ < 100
- ◯ 100 – 1K
- ◯ 1K – 10K
- ◯ 10K – 100K
- ◯ 100K – 1M
- ◯ > 1M

8. 1.8. Roughly how many people were working on these systems when the process of migrating to microservices was started? *

Marcar apenas uma oval.

- < 10
- 10 – 50
- 50 – 200
- 200 – 600
- > 600

2. Strategies and Processes

9. 2.1. Where do you look for guidance when migrating a monolith system to microservices? *
Tick all that apply

Marcar tudo o que for aplicável.

- [ ] Scientific articles
- [ ] Books
- [ ] Conference presentations
- [ ] Web resources, blogs
- [ ] Other practitioners' experiences
- [ ] External consulting
- [ ] Internal consulting
- [ ] Outra: _____

10. 2.2. How do you often plan the migration of a monolith to microservices in regards to the evolution of the product? *
Tick all that apply

Marcar tudo o que for aplicável.

- [ ] Rewrite/rebuild the entire system from scratch
- [ ] Stop product evolution, refactor the system to microservices, and then proceed with evolving the product
- [ ] Continuous refactoring interspersed with product evolution
- [ ] Outra: _____

11. 2.3. How likely are you to consider these data sources when deciding how to decompose a monolith into different services? *

Marcar apenas uma oval por linha.

	Very unlikely	Unlikely	Neutral	Likely	Very likely
The output of static analysis tools (e.g., based on source code repository analysis)					
The output of dynamic analysis tools (e.g., based on runtime data, such as logs or traces)					
Development process data (e.g., from version-control, project management tools)					
Software Documentation (e.g., business logic/capabilities/objects, data flow) and related artifacts (e.g., source code, configuration files)					

12. 2.4. Which criteria do you use often for deciding the new service boundaries when decomposing a monolith into different services? *
Tick all that apply

Marcar tudo o que for aplicável.

- [ ] Decomposition by business capability (i.e., so that the decomposition reflects the structure of the organization, its teams and business units)
- [ ] Decomposition by subdomain (i.e., so that the decomposition reflects an analysis of the processes and information flows of the business. e.g., using Domain-Driven Design)
- [ ] Based on coupling and cohesion of the business logic of the monolith (i.e., keep strongly-coupled elements of the implementation in the same services)
- [ ] Based on coupling and cohesion of the data owned by the monolith (i.e., keep strongly-coupled data entities managed by the same services)
- [ ] Based on lexical similarity (e.g., joining in the same service API endpoints with similar names)
- [ ] Outra: _____

3. Tools

13. 3.1. Thinking of the migration from monolith to microservices projects you have been involved in, have you been assisted by any automated or semi-automated tools? *

Marcar apenas uma oval.

- ◯ Yes
- ◯ No

14. 3.2. If yes, which tools were used and for which purpose?

_____

15. 3.3. Which type of tools did you find the most useful?

_____

16. 3.4. In which of these activities would you most appreciate additional tool support? *
Tick at most 5 options.

Marcar tudo o que for aplicável.

- [ ] Understanding the monolith
- [ ] Deciding services boundaries
- [ ] Refactoring code
- [ ] Evaluating a decomposition result
- [ ] Regression testing
- [ ] Deployment automation
- [ ] Analysis of organizational restructuring needs (team size, team organization, etc.)
- [ ] Planning and managing the refactoring process
- [ ] Microservice API design
- [ ] Outra: _____

17. 3.5. What characteristics would you find most important in a tool for refactoring towards a microservice architecture? *
Tick at most 8 options.

Marcar tudo o que for aplicável.

- [ ] Ease of use
- [ ] Versatility – usable via command line, within IDEs, independently, etc.
- [ ] Support for various programming languages
- [ ] Requiring minimal manual inputs
- [ ] Doing the refactoring quickly
- [ ] Being actively maintained
- [ ] Having successful case studies reported
- [ ] Ability to apply restrictions to the decomposition result
- [ ] Easy modifications on restrictions without having to start over
- [ ] Multiple decompositions alternatives
- [ ] Provide an explanation of candidate decompositions
- [ ] Provide a visualization of candidate decompositions
- [ ] Edit/model a candidate decomposition
- [ ] Provide comprehensive decomposition guidance
- [ ] Provide an analysis of the changes of quality attributes and metrics
- [ ] Outra: _____

4. Refactoring Techniques - Splitting the Monolith

In this section, we explore a selection of refactoring techniques for splitting a monolith. The goal is to understand how often these techniques are used, how they are used, their challenges and which techniques are used together.

Please take the next ~5min to read about each one of the refactoring techniques below.

Strangler Fig Application

When to use: When we have decided to evolve a system to a microservices architecture, and we want to take incremental steps toward the new architecture but also ensure that each step is easily reversible, reducing risks.

How to apply it: Make minimal changes to the existing system and gradually move functionality over to the new microservices architecture. Do this by replacing or rewriting existing features in parallel to the old architecture, one at a time, until the old architecture has been fully replaced. Usually, we will need to create a proxy or façade that provides a stable API for old clients throughout the migration.

UI Composition

When to use: As we incrementally migrate a monolith to microservices, the functionality the user-interface provides needs to be served partly by an existing monolith and partly by the new microservice architecture.

How to apply it: Different variants of *UI Composition* may be used: *Page-based Composition* consists of having different services serve different pages; *Widget Composition* consists of embedding a piece of user-interface provided by a new service into the monolith-provided user interface; and *Micro Frontends* consist of taking a microservice-based approach to frontend development.

Branch by Abstraction

When to use: When changes to the existing codebase will likely take time to carry out, and we want to avoid any disruption. It assumes we can change the code of the current system.

How to apply it: Develop, in the monolith, an alternative implementation for a module so that the new and old implementations comply with the same interface/abstraction. The new implementation will typically call a new external service, whereas the old would implement the functionality internally. At some point, switch from the old implementation to the new implementation.

Parallel Run

When to use: When the failure of the functionality being worked implies a high risk for the business.

How to apply it: With a parallel run, rather than calling either the old or the new implementation, we call both, allowing us to compare the results to ensure they are equivalent. Despite calling both implementations, only one is considered the source of truth at any given time. Typically, the old implementation is considered the source of truth until the ongoing verification reveals that we can trust the new implementation.

Decorating Collaborator

When to use: When we want to trigger some behavior based on something happening inside the monolith, but we want to avoid changing the monolith itself. This technique assumes we can intercept responses returned by the monolith before they reach the clients.

How to apply it: Use the *decorator* pattern to make it appear from the outside that we have added new logic to the monolith without actually changing it. The technique consists of routing client requests to a proxy which will allow the call to reach the monolith, as it normally would, and based on the result of this call, call out to a new microservice, and possibly modify the result in some way before returning it to the client.

Change Data Capture

When to use it: When we need to react to a change in data in the monolith but cannot intercept this change at the perimeter of the monolith (e.g., using a decorator) or change its implementation.

How to apply it: Rather than trying to intercept and act on calls made into the monolith, we react to changes made in a datastore. The underlying capture system will be coupled to the monolith's datastore, and can rely on database triggers, transaction log pollers (usually a file, into which is written a record of all the changes that have been made) or batch delta copier (a program that regularly scans the database in question for what data has changed).

Change code dependency to service call

When to use it: When a software system is decomposed into a set of smaller services to use a microservices architectural style, some components in the system will act as dependencies to other services or components.

How to use it:

Try to keep the services' code as separate as possible. When services depend on the same piece of code, there is a chance that changes to that code motivated by the needs of one service might negatively affect the other service. If this is code shared as a library that rarely changes (e.g., a string manipulation library) it is reasonable to have different services depend on it. On the other hand, if this is code related to internal entities, or entails different scalability needs, we share it as a new service so that it can be changed independently and gradually, or scaled independently.

18. **4.1. Tell us to which degree you agree with the following statements in the scope of the techniques presented below:** *

"In the migration project(s) in which I have been involved the use of this technique was important"

Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Strangler Fig	◯	◯	◯	◯	◯
UI Composition	◯	◯	◯	◯	◯
Branch by Abstraction	◯	◯	◯	◯	◯
Parallel Run	◯	◯	◯	◯	◯
Decorating Collaborator	◯	◯	◯	◯	◯
Change Data Capture	◯	◯	◯	◯	◯
Change code dependency to service call	◯	◯	◯	◯	◯

19. 4.2. Which of these techniques do you often use together? Do you use them "before", "after" or "as a step of"?

In answering this question, consider the techniques above but also any other that might have been used in the migrations that you have been involved in.

20. 4.3. In your experience, what are the main challenges of applying these techniques?

5. Refactoring Techniques - Decomposing the Database

In this section, we explore a selection of refactoring techniques for decomposing the database. The goal is to understand how often these techniques are used, how they are used, their challenges and which techniques are used together.

Please take the next ~5min to read about each one of the refactoring techniques below.

Database View

When to use it: We want a single source of data for multiple services, but it is impractical to change all clients to point to the new service(s). For clients that will keep considering the monolith as their source of data, we want to mitigate concerns regarding coupling to specific parts of the data schema. There are more clients *reading* data than *writing* it.
How to use it: For clients that only read data, create a dedicated schema that hosts views looking like the old schema, and that are assembled from data now owned by the new service(s). Have clients that only read data point at those views instead of the original tables. We will need to change only the clients that need to write data so that they start using the new services directly, instead of the monolith.

Database Wrapping Service

When to use it: When multiple clients depend on the same datastore, but it is just too hard to consider pulling apart the underlying schema.
How to use it: Hide the database behind a service that acts as a thin wrapper and make clients depend on this new service instead of the database. Encourage developers writing the different client applications to think of this new service as someone else's and start storing their own data locally.

Database-as-a-Service Interface

When to use it: When we have clients that really just need a database to query. This could be because they need to query large amounts of data, or because external parties are already using toolchains that require a SQL endpoint to work against.
How to use it: Create a dedicated database to be exposed as a read-only endpoint, and have this database populated when the data in the underlying database changes. Take care to separate the database we expose from the database we use inside our service boundary. A mapping engine takes changes in the internal database, and works out what changes need to be made in the external database. When our internal database changes structure, the mapping engine will need to change to ensure that the public-facing database remains consistent.

Aggregate Exposing the Monolith

When to use it: As we extract services from the monolith, we often realize that some of the data should become part of the new services, and some of it should stay where it is. When the monolith still owns the data we want to access, we will need to allow the new services the means to access this data.
How to use it: Make explicit what information the new services need from the monolith by exposing it via dedicated service endpoints. The monolith still owns the data and decides

what state changes are valid; it is not just like a wrapper around a database. Beyond just exposing data, expose operations that allow external parties to query the current state and to make requests for new state changes.

Change Data Ownership

When to use it: As we extract services from the monolith, we often realize that some of the data should become part of the new services, and some of it should stay where it is. When the newly extracted service encapsulates the business logic that changes some data, that data should be under the new service's control.

How to use it: Move the data from the monolith over into the new service. Change the monolith to treat the new service as the source of truth for the data it now owns, calling it to read or change data when needed. When doing this, we may have to consider the consequences of breaking foreign-key constraints and transactional boundaries.

Synchronize Data in Application

When to use it: When we want to split the schema of the monolith into two separate datastores, in preparation for extracting a new service.

How to use it: Do this in a sequence of three steps. The first step is to create the new datastore and bulk-import data from the monolith—only the part of the schema we are interested in splitting needs to be imported. The second step consists of making the monolith write the same data to both datastores, ensuring they are kept in-sync, and that writes to the new datastore are working well. In the third step, with a simple change to the monolith, we make the new datastore the source of truth and ensure that the reads also work. As we still write to both datastores, we can easily fallback to reading from the old datastore if any issue is found.

Tracer Write

When to use it: When we need to move the ownership of some data from the monolith to a new service in an incremental fashion, tolerating there being two sources of truth during the migration.

How to use it: First, identify the part of the monolith data schema whose ownership we want to change and the service that will host it. Then, ensure this data is kept in-sync between the monolith and the new service. This can be achieved in a number of ways: a) Allowing writes only on one of the sources of truth, and making it in-charge of replicating the change in the other one; b) Have clients send writes to both sources; or c) Allow clients to send writes to either source, and synchronize data behind the scenes. Once all clients use the new source of truth, the old source of truth can be retired.

Note: *Synchronize Data in Application* focuses on splitting the schema, and this technique focuses on moving a part of the schema to the ownership of a previously created service.

Split Table

When to use it: When we are using a relational database and we find that columns of a single table need to be split across two or more service boundaries.

How to use it: First, split the columns of the table apart in the existing schema, placing one (or more) of them in a new table. Then, move the new table to a different service. If there was code updating columns that are now owned by different services, the monolith will need to call the new service to update the split column.

Move Foreign-Key Relationship to Code

When to use it: When we are using a relational database, and we are moving some functionality to a new service but we want to keep some of the data used by that functionality in the monolith.

How to use it: With a monolithic system, joining different relational tables is often done by database queries, but when one of those tables is moved to a new service, this new service may need to fetch from the monolith any data that it requires that is still kept there, and join it "in memory" with the data it owns, rather than "in the database".

21. **5.1. Tell us to which degree you agree with the following statements in the scope of the techniques presented below:** *

"In the migration project(s) in which I have been involved the use of this technique was important"

Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Database View					
Database Wrapping Service					
Database-as-a-Service Interface					
Aggregate Exposing the Monolith					
Change Data Ownership					
Synchronize Data in Application					
Tracer Write					
Split Table					
Move Foreign-Key Relationship to Code					

22. 5.2. Which of these techniques do you often use together? Do you use them "before", "after" or "as a step of" each other?

In answering this question, consider the techniques above but also any other that might have been used in the migrations that you have been involved in.

23. 5.3. In your experience, what are the main challenges of applying these techniques?

6. Refactoring Techniques

The following questions are general to all refactoring techniques.

24. 6.1. What other *techniques* or *patterns* have you used in the migration process that were not mentioned in this survey?

25. 6.2. In your opinion, which are the most important challenges faced in the migration process? *
Tick at most 9 options.

Marcar tudo o que for aplicável.

- [ ] Database migration and data store splitting
- [ ] Dealing with data consistency
- [ ] Communication among services
- [ ] Ensuring code maintainability
- [ ] Decoupling services from the monolith
- [ ] Service orchestration
- [ ] Multitenancy
- [ ] Microservices statefulness
- [ ] Continuous deployment/integration
- [ ] Effort estimation
- [ ] Expected long-term return on investment (ROI)
- [ ] Team organization
- [ ] Infrastructure to support microservices
- [ ] Selecting infrastructure patterns
- [ ] Ensuring reliability
- [ ] Ensuring scalability
- [ ] Ensuring security
- [ ] Outra: _____

7. Evaluation

26. 7.1. Do you usually evaluate the result of a decomposition? How? *

_____

_____

_____

_____

_____

27. 7.2. What quality attributes do you assess when evaluating the decomposition result? *
Tick at most 4 options.

Marcar tudo o que for aplicável.

- [ ] Maintainability
- [ ] Performance, Efficiency
- [ ] Business-related indicators
- [ ] Reusability
- [ ] Security
- [ ] Scalability
- [ ] Availability
- [ ] Team Agility
- [ ] Outra: _____

28. 7.3. Why?

_____

_____

_____

_____

_____

29. 7.4. What *metrics* would you use to evaluate these quality attributes? Why? *
Example: I use the throughput (e.g., requests per second) to measure the efficiency.

_____

_____

_____

_____

_____

30. 7.5. In which environments do you evaluate a decomposition result? *
Tick all that apply

Marcar tudo o que for aplicável.

- [ ] Development
- [ ] Staging
- [ ] Production

31. 7.6. What kind of inputs do you use to evaluate a decomposition result? *
Tick all that apply

Marcar tudo o que for aplicável.

- [ ] Functional tests
- [ ] Simulation
- [ ] Production