CAPCO

MICROSERVICES CODE FACTORY

GENERATIVE AI'S POTENTIAL FOR END-TO-END AUTOMATION OF BANKING LEGACY SYSTEMS REFACTORING

In this article, we evaluate the use of generative AI in supporting the complete process of refactoring monolithic applications into microservices-based solutions.

This is the third post in our Generative AI series. The first post considered usage of Gen-AI over the entire software delivery lifecycle for greenfield projects[1], and the second focused on using advanced knowledge management techniques towards soliciting better responses from Gen-AI[2].

INTRODUCTION

Many financial institutions are taking the route of splitting their legacy monolithic applications into microservices to achieve modernization through digitalization and migration to cloud. This process starts with refactoring, i.e. restructuring legacy monolithic applications into the optimal number of microservices to achieve the best results.

In this post, we describe our efforts and results of executing practically against our refactoring methodology, described in a recent paper[3]. The methodology uses machine learning to generate the ideal refactoring scenario (blueprint) for any legacy monolithic application, resulting in an optimized collection of microservices for superior system performance and with minimized cloud operational costs.

As a next step, our Microservices Code Factory uses the refactoring recommendation blueprint to create, test and document code for the entire microservices collection. We explore how Gen-AI can assist with this process, helping improve the code quality and accelerate the end-to-end automated refactoring process.

IMPORTANCE OF THE TOPIC

There are several ways in which monolithic applications (or any workloads) can be migrated to the cloud, such as rehosting, re-platforming, refactoring, re-architecting, and so on. Of these, refactoring allows to enjoy the benefits of a cloud native architecture, but without the (often significant) cost and time penalties that come with rebuilding the application from scratch. Automation around a refactoring approach is naturally of interest as this can help turn massive complexity into something that is manageable and viable.

HOW GEN-AI CAN HELP WITH TASKS THAT WOULD BE NEAR-IMPOSSIBLE TO ACHIEVE WITHOUT IT

Transforming a monolithic application into a microservices-based solution (using the target state blueprint our refactoring methodology provides) can largely be done using a deterministic approach that:

- Generates appropriate code repositories, and
- Lifts out existing code functions in the monolith and moves them to the appropriate microservices together with auxiliary assets such as unit tests and technical documentation, aiming for maximum reuse of existing assets.

There are areas where Gen-AI is especially well-geared to help improve quality and accelerate the code refactoring process, i.e. we use Gen-AI as a co-pilot, specifically for:

1. **Discovering external dependencies:** Monolith applications often have dependencies, such as external code libraries they need to function, database connections, external components, and so on. These dependencies need to be 'inherited' by the appropriate microservices.
2. **Creating unit test cases for the new service layer:** Legacy applications are often many years old and frequently have little or no unit test coverage, making it hard to maintain code quality.
3. **Auto-documenting the new solution:** Documentation of legacy applications is often very outdated and/or incomplete. Such documentation is invaluable to newcomers or to use as reference during production outages.
4. **Improving code quality:** As we are lifting the source code from the monolith to the new microservices, we have a great opportunity to use Gen-AI to improve the code without changing its functionality.

CASE STUDY

As a practical case study, we selected a monolithic application that manages the sales, production, and logistics of a product. This legacy system was poorly documented but had some pre-existing test coverage built in. The code stack uses Java v11, and employs SpringBoot (v2), MVC, and JPA.

We used the approach outlined in Figure 1 to execute the monolith refactoring process. Our end-to-end solution spans across six steps and comprises our refactoring methodology (code crawler plus clustering engine), our new Microservices Code Factory – a tool for generating microservices code, and Gen-AI.

Figure 1. Automating the end-to-end monolith refactoring process with Gen-AI

PROCESS STEPS COMPONENTS	1	2	3	4	5	6
Capco refactoring methodology (to create the optimized microservices stack recommendation)	'Crawl' the monolith source code and generate a list of code functions with detailed data profiles	Cluster code functions based on data similarity and generate a target microservice composition recommendation (blueprint)				
Capco Microservices Code Factory			- Use the blueprint to generate code repositories (folders) - Lift source code over from functions in the monolith into the appropriate microservices as per the blueprint - Migrate existing unit test cases from the monolith to the corresponding microservices	Generate an intra-service communications layer		Replicate the monolith configuration files to the microservices
Gen-AI (LLM - large language model) as a co-pilot			- Enhance (or create new if none exist) additional unit tests - Generate technical documentation (source code and API specifications) - Improve code quality based on given parameters	Generate documentation to account for the additional APIs	Evaluate external dependencies	

Once we've created the microservices stack recommendations (a blueprint that will guide the source code transformation process), and generated a code repository for each of the target-state microservices, we create a new intra-service communications layer among the new microservices stack that mimics the existing internal data read/write interactions in the monolith. This is where our Gen-AI co-pilot comes into play:

- We use AI to analyse the monolith source-code unit tests and create additional tests towards improving code coverage. If no unit test cases exist in the monolith, AI creates these tests from scratch, and this is integrated into the code repositories.

- Gen-AI generates technical documentation including:

    - JavaDocs for documenting classes, methods, fields, and other program elements in the newly created microservices

    - API specifications (OpenAPI) for each service. This includes all existing APIs as well as new API endpoints arising from the intra-service communications layer.

- As part of moving code over from the monolith to the microservices, code functions are sent to AI, with the purpose of improving code quality. Gen-AI is requested to apply the code quality rules we define (based on Sonar software quality management system) and adjust the code accordingly without changing the underlying functionality or method signature. By doing this, we ensure that the microservice code stack will be generated with Sonar quality improvement recommendations and save significant time in applying such code changes manually at a later stage.

- We use Gen-AI to evaluate all external dependencies of the monolith to ultimately enable replicating these references into the new service layer. These include items such as database configurations, required permissions, and so on.

RESULTS

Our automated, end-to-end refactoring process delivered well beyond expectations, and whilst not a magic bullet, Gen-AI is the best co-pilot you could wish for to support this process.

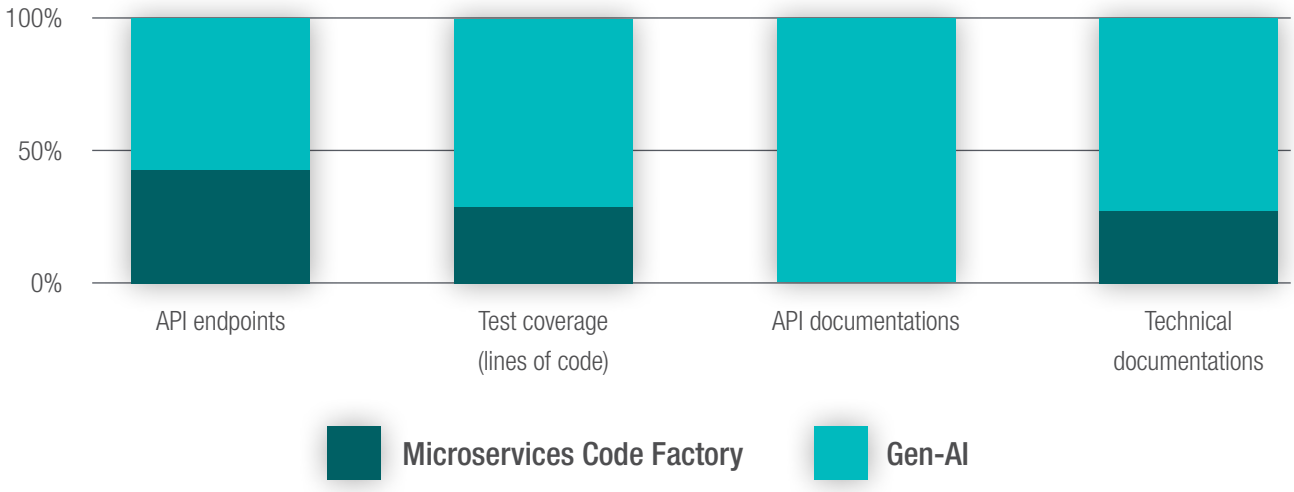
With Gen-AI's help, we were able to ensure that the blueprint created by our original methodology was followed precisely. AI models assisted in the areas of creating a complete test coverage and comprehensive technical documentation for the new code. Gen-AI particularly impressed us with its ability to improve code quality based on our parameters.

Figure 2 shows, in summary, the impact of involving Gen-AI in our automated end-to-end monolith refactoring process using our case study as a point of reference. In this chart, dark green denotes the percentage of activity performed directly by our Microservices Code Factory tool, and light green – the proportion done via Gen-AI. It is clear from the chart that Gen-AI made a significant contribution, automating aspects of the refactoring process that would be impossible to do at scale by human hand alone.

For example, for API endpoints, the code factory lifted approximately 50 percent of the API endpoint-related code from the monolith to the microservices stack. The other 50 percent was created using Gen-AI towards facilitating intra-service communication. As another example, where the original legacy system had no API documentation, AI significantly accelerated the process of creating comprehensive specifications, for both existing endpoints that were lifted from the monolith as well as the new intra-service endpoints.

AI also performed particularly well in terms of providing a detailed checklist of all jobs to be done as part of the refactoring process, such as database and email configuration.

Figure 2. Impact of involving AI in the automated monolith refactoring process in our case study



CONCLUSION

We were able to conclusively demonstrate that, through a combination of recommendations from our refactoring methodology, our microservices code generator and the power of Generative AI, legacy monolithic applications refactoring projects can be automated end-to-end. The resultant target state technology stack in our case study offered good test coverage, was well documented, and reduced the challenges around managing external dependencies. Not only was the code created, but it was also supportable and testable.

Importantly, automating the creation of microservices code against our blueprint would have taken approximately 3-4 months in our case study to do manually. Using AI this was achieved in 1.5 hours, with 100 percent accuracy and completeness and a significantly improved code quality compared to the original monolith code.

As always, there is room for enhancing several dimensions in this solution, which will be covered as part of our future research:

- Transformation of non-Java-based monoliths - we plan to use Gen-AI to transform legacy source code written in other programming languages such as COBOL or C#. It is expected that Gen-AI will be extremely helpful with code translation between monolith and target state.
- Gradual refactoring approach comprising intermediate states for very complex transformation initiatives - in these cases, the monolith will be systematically 'strangled' and the monolith–microservices co-existence will need to be managed over a period of time.
- Decomposition of multiple monoliths with partially overlapping features into a single microservices technology stack.
- Automated approaches to connect the microservices back-end code created by our solution to the front-end code.

REFERENCES

1. Generative AI's Potential For Industrializing And Scaling Software Delivery (capco.com)
2. Generative AI's Potential For Knowledge Management (capco.com)
3. Automation Of Legacy Banking Systems Refactoring (capco.com)

AUTHORS

Gerhardt Scriven, Managing Principal

Diego Sarai, Managing Principal

Rafael Maciel, Senior Consultant

Warley Rocha, Senior Consultant

CONTACTS

Alessandro Corsi, Partner, alessandro.corsi@capco.com

Luciano Sobral, Partner, luciano.sobral@capco.com

WWW.CAPCO.COM

CAPCO
a wipro company