Microservice Architecture for Insurers

Solutions for building flexible and nimble IT Architecture using microservices

Abstract

Most insurers are going through a "digital transformation" phase and, as a result, have similar competing challenges i.e. to support the needs and the ever-changing demands of a customer centric and customer-first strategy. To achieve that there has been a significant push to move away from a traditional monolithic architecture while ensuring the integrity and consistency of data across all systems.

Insurance IT systems were designed to be monoliths either due to the interwoven nature of insurance business processes or because they were built across many years or gathered as part of mergers and acquisitions over time.

This paper highlights some of the common challenges and impediments across insurers and how they are being solved pragmatically. While keeping in mind some pitfalls and anti-patterns that arises commonly across and Insurer's IT landscape.

Author

Sabyasachi Chowdhury, *Principle Architect Technology with Cognizant's Insurance vertical, has vast experience architecting and implementation large legacy transformation initiatives for insurers across the globe. Have spent considerable amount to time solving current architecture impediments and business process roadblocks to help Insurers march towards a more digital focused ecosystem along with maintaining their proven and stable legacy IT landscape.*

Sabyasachi.Chowdhury@cognizant.com

Thanks to Mr Rajesh Shastri – CTO & Head of Digital Engineering Insurance – Cognizant Technology Solutions for inputs and suggestions.

Impediments

Traditionally, insurance carriers were mostly focused towards providing marginally better service to their insured's and agents by adding features to existing products, through premium optimization or by streamlining claims and other business processes.

But there is a fundamental shift towards engaging with customers and channel partners by building user centric system of engagements, reacting to continuous feedbacks and the ability for customers, agents and employees to interact across different channels and devices.

A state of art customer experience is not only a function of a human centric front end layer, but are built upon the foundations of degree of responsiveness, availability and consistency of data and the ability to gather feedback and respond to those feedbacks quickly.

Most insurers are challenged with a host of impediments, which can be classified into 2 broad categories -

The ability to react fast to changing business demands or Speed to market

Below are some of the key impediments towards achieving speed to market -

- ✓ **monolithic** policy and claims systems, which evolved over years
- ✓ **manual processes and interventions** through multiple actors
- ✓ **constantly changing rules** that govern business processes, resulting in significant automation challenges
- ✓ **complex and interwoven business processes and lifecycle management** spanning across multiple domains causing systems to sit as close as possible, adding up to the size and complexity of these systems and increases the need of data flow in every direction.
- ✓ **dependency on batch** oriented systems operating on data feeds
- ✓ **heterogeneous systems** landscape due to mergers and acquisitions over years which also led to accumulation of technical debt
- ✓ **lack of enterprise agility** towards adoption of newer technologies and tools, technology evaluation and adoption takes months.
- ✓ **lack of agile and DevOps** practices, mostly targeted towards small to medium initiatives in silos, primarily towards mobile and web based developments.
- ✓ **lack of test driven development** and **continuous Integration** are not popular due to challenges posed by demands for changing requirements and to speed to market.
- ✓ **large costly release cycles** spanning over weeks - due to ever-increasing demand towards speed to market and changes to underlining rules and regulations, most of the changes are often collected over time and distributed through infrequent software release cycles.

Seamless end user experience

Below are some of the key impediments towards a better user experience -

- ✓ **strong LOB alignment** and ownership leading to implementation of singular executable or applications leading to LOB centric portals
- ✓ **manual and disjoint processes** for FNOL, 3rd Party Claimants & Injured Workers as they are still done over phone

- ✓ **services** were mostly focused around what the system can provide rather than what the customer wants and relegated to software professionals seeking to resolve interface and data sharing problems associated with incompatible software systems.
- ✓ **lack of API strategy** and API centric architecture and platforms
- ✓ **legacy business services** causing too much dependence on other system availability
- ✓ **low availability** due to traditional long maintenance cycles and lack of continuous delivery mechanism.
- ✓ **lack of cloud adoption** and dependence on infrastructure which are on premise.

Solutions

It is imperative to focus on improved customer engagement models, continuous innovations, digitization or automation (like 0 touch and state through processing) and improved decision making by leveraging data, but the scope should not only be restricted to digital front ends alone, but more towards building an efficient and nimbler IT ecosystem/architecture from backend up to the system of engagement layer.

Abstraction from core legacy system by moving towards a microservices based architecture, practicing domain driven design principles and operating in a fully agile model is the only recipe to solve some the current problems around speed to market and end user experience.

There may not be a direct correlation between the solutions this papers talks about with any of the impediments listed above, mostly because they complement each other and cannot be fully achieved or implemented without the existence of the other.

Migrate towards a Microservices based architecture

A good example in the context of an insurer is the rating service, because they depend on just rates and rules and have minimum to no dependency on any external or parallel service the rating service can be defined within a bounded context. Hence it becomes one of the prime candidates across insurers for a first step towards microservices adoption and provides the ability to deploy and scale independent of the policy systems.

There is no definite standard which defines the size of a Microservices, hence there can be other candidates for Microservices within an Insurers landscape such as account management, commission calculation systems etc. But systems for claims and document management are ideally bad candidates to be defined as a Microservice because Claims is a long running process which interacts and depends on data and actions across multiple actors and systems. Similarly the document management system is heavily dependent on policy and claims date for generation of correspondence and also on customer data for preferences.

James Lewis and Martin Fowler describe microservices as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery"

Hence it is important to understand the principles of domain driven design and importantly the most common patterns like bounded context, circuit breakers, anti-corruption etc to design a Microservice.

Leveraging the existing capabilities, take an API first strategy

The term API economy refers to the opportunities associated with productizing the exposure of your business functions mostly developed as microservices through APIs. Consider that your API is a consumable product, and an insurer should market and position their products correctly for maximum profit.

For insurers embarking on the digital transformation journey it is imperative to offer a richer set of data-driven, customer-facing services, enabled by a customer-centric approach of doing business or risk of being marginalized. Below are some of the key areas an insurer should focus to define the overall digital and API strategy -

- an omni channel enabled buying journey, that augments traditional channels with robust self-service options, direct purchasing and a single customer experience across online, mobile and now social channels.
- ability to leverage data and analytics across the entire value chain, including product innovation, marketing and sales, new business, servicing, claims, and operations
- APIs to make it easy to do business with partners(providing APIs to partners allows insures to offer insurance options to their customers directly)
- Get quote or rate APIs exposed for 3rd party aggregators
- First notice of loss (FNOL) automation and self-service options for claims and claims status
- Get quote or rate for external comparative rater integrations like PL Rater and EZLynx

Walgreens is a good example. Like many retailers, the company provides a photo printing service that takes digital photos and turns them into physical photographs. *https://developer.walgreens.com/*

APIs have a deep design relationship with microservices, in nutshell the business APIs exposed are developed as microservices to provide scalability, responsiveness, management of deployment, monitoring and most importantly communication between every other Microservice in the architecture as well as with the applications and Web sites they power and the databases from which they draw real-time information, essential to their functioning.

Agility towards changes and new opportunities through the adoption of CI/CD and PaaS

Most Insurers are already performing Continuous Integration (CI) in some form or the other, those adopting CI have further leveraged agile ideologies to use microservices as a driver to achieve more frequent software releases, which has led to the practice of continuous delivery (CD). CD uses a quality focused ideology to build potentially shippable product increments, which speeds the deployment pipeline, achieving a result that culminates in bringing changes to production as quickly as possible. CD is a business function and too many frequent releases to production may cause disruptions and confusion, hence many Insurers still follow a stringent process of approvals and gatekeeping for any CD pipeline.

Development of a microservices based application is always a decentralized effort, allowing smaller teams (famously referred to as “2 pizza team” – a team which can be fed by 2 pizzas) internal or otherwise – to oversee individual segments of the application through their development lifecycle and through release. This helps development of applications around singular business processes rather than overwhelmingly large development efforts.

Microservices are an attractive CI/CD pattern because of their enablement of speed to market. With each microservice being developed, deployed and run independently microservices allow organizations to “divide and conquer”, and scale teams and applications more efficiently.

Many Insurers are adopting agile and CI/CD in targeted areas such as mobile and web. Those areas, however, typically depend on core back-end systems for data and business logic. If those systems are not also nimble then the agility of the business will always remain limited. Insurers must therefore bring Agile and CI/CD to all their platforms— including the mainframe—if that’s where their core data and business logic reside.

Key Considerations

Like SOA, companies planning towards microservices are finding themselves struggling with mainly defining the approach, where to start, service granularity, data migration challenges, organizational and cultural change, and distributed processing challenges. Hence, understanding the future needs, current business drivers, overall organizational structure and technology environment is important before jumping on to the bandwagon due to some of the antipatterns and pitfalls associated with this architecture pattern.

Some of the key antipatterns and pitfalls that are encountered with the insurance ecosystem due to the level of complexity within the current landscape are –

Level of Service Granularity (“Grains of Sand” Pitfall)

Choosing the right level of granularity for your services is critical to the success of any microservices effort. Service granularity can impact performance, robustness, reliability, change control, testability, and even deployment.

For example, if you have 3 services each for add, update and delete customer, it may be justified to club them to a single service called manage customer.

Data migration antipatterns

When insurer plan to migrate from a monolithic application to microservices architecture the first goal is to split the functionality of the monolithic application into small, single-purpose services and the second goal is to then migrate the monolithic data into small databases (or separate schemas) owned by each service.

Most of the insurers believe that data should be a corporate and not an application asset. Given that, you can appreciate the risk involved and the concerns crop up with continually migrating data. Data migrations are complex and error-prone—much more so than source code migrations. Understanding the risks involved with data migration and the importance of "data over functionality" is the first step in avoiding this antipattern.

Example, insurers are embarking onto the journey of microservices by carefully selecting use cases, which can be broken into smaller more independent components like rating and rules components. But when it comes to breaking a monolith like a claims processing system or for that matter a policy administration system, it is advisable to follow the path of functionality first and data last.

Reporting and data consistency antipatten

With microservices the services and corresponding data are contained within a single bounded context. How to obtain reporting data in a timely manner and still maintain the bounded context between the service and its data is the challenge. Remember, the bounded context within microservices includes the service and its corresponding data, and it is critical to maintain it.

At the same time maintaining data consistency across these systems not only for the purpose of reporting but also for supporting end customer queries and needs is also of upmost importance.

The REST pitfall

REST is by far the most popular choice for exposing APIs and accessing microservices and communicating between microservices. The REST pitfall is about using REST as the only communication protocol and ignoring the power of messaging to enhance your microservices architecture. With messaging and asynchronous requests the service caller does not need to wait for a response from the service when making a request, referred to as "fire-and-forget" processing.

Examples can be submission of a final and accepted quote, FNOL, policy submission and also for communication between microservices like the rating and policy admin system, policy and payment system etc.

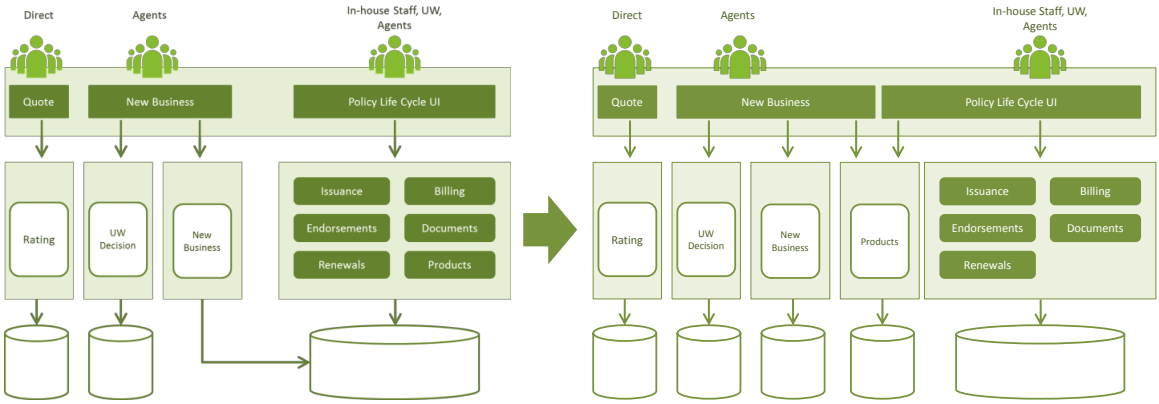
Adding to the maintenance nightmare

Microservices is about creating lots of small, distributed single-purpose services each owning their own data. As these services supports the notion of bounded context and share nothing architecture, where each service is compartmentalized and completely independent, hence adding to maintenance overhead.

A pragmatic approach towards Microservices.

Marching towards a Microservices based architecture helps an Insurer move away from some of the existing impediments and challenges with current systems and processes, but a methodical and pragmatic approach is what is most needed.

As most of the current Insurance systems are designed as monoliths, migration to microservices based architecture is ideally done in a progressive manner by adopting the functionality first and data last approach as depicted in the diagram below -



Best Practices

- Agile application development with 2 pizza teams and continuous Integration (CI)
- Ability to perform Continuous Delivery (CD).
- Adapt to the principles of cloud native and 12 factors apps.
- Adapt to the principles of Domain Driven Design, and patterns like the CQRS, event sourcing and bounded context.
- Identify the right candidates to be built as Microservices and move to cloud
    - What architecture characteristics are most important?
    - Can it function as an independent component within a bounded context?
    - Does that component have direct dependency with other components or do they share persistence?
    - Do these components change frequently?
    - Is there a need to scale and build elasticity for an application?
    - What are the primary business drivers?
- Assemble a "modern stack" from existing capabilities (cloud first, scale, working software, CI/CD, eliminate existing technical debt)
- Opportunistically modernize existing capabilities and incrementally move to the new stack
- Build new components as microservices for the capabilities that are deemed non-go-forward, which means contain existing legacy applications and make changes or enhancements to them only if not possible as Microservices.
- Expose the right grained APIs as products and independent of consumers' implementations or in other words build them generic and self-contained to the extent possible.
    - Start with more coarse-grained and then split it further if needed.
- Reuse existing services if they already meet performance SLAs and are already right grained; wrap them as REST APIs if needed, use REST for all new services.
- Microservices are not an alternate to SOA, continue investing in SOA and build upon it. Which means continue to build on the SOA patterns and encourage the culture of externalizing and reuse within the enterprise.
    - Examples of rules and rating components which can be externalized and built as microservices.
- Components which are dependent on external data feeds and requires critical processing should be treated asynchronously like batches, also move to cloud as that will help with elasticity.
- Design you applications and Microservices independent of any native cloud platform following the 12 factor app principles- this will help migrate to cloud and also reduce vendor locking from a cloud perspective.
- Initiatives like ODS, MBM and RDM should continue, if implemented on a RDBMS database, layer them with the likes of a MongoDB or any other NoSQL DB for faster queries by flattening the data
    - but balance between data availability and too many data migration between systems and databases
    - data migrations are error prone and complex and will become more complex over time.
    - treat data as a corporate and not an application asset

Some of the ready candidates to be moved to microservices based architecture as seen in the industry are

- premium generation or rating services
- FNOL and APIs for claims status
- product engines
- rule based services for rating, new business or underwriting

- or for that matter the whole Quote or New Business component
- account management

Summary

In order to overcome some of the key impediments, it is important to start small and experiment, instead of going with a mindset of breaking a monolith to smaller apps, identify the right candidate which are self-contained and migrate progressively.

Understand and take into consideration the common pitfalls and antipatterns that may arise due to implementation of new patterns like Microservices.

An overarching enterprise strategy and vison supported by architecture guidelines and principles towards microservices based architecture, APIs and data centricity is essential. It is also important to welcome the right level of changes to be more agile in order to reduce the size of the failures.