Microsoft

A playbook for monolith decomposition

Amit Sinha, Chief Technology Officer, Private Equity

Intended audience

This playbook has been written for C-suite executives in non-technical roles, of organisations with monolith applications that require decomposition. The playbook is designed to give a holistic view of why monolith decomposition is important, how to build the business case and the implementation options available.

It is recommended that this playbook be discussed with the executive(s) accountable for technology such that an informed decision can be made on the optimal path forward.

Table of Contents

Introduction

Over the past few months, I have sat down with several Private Equity (PE) Operating Partners and CTOs, talking through business and technology challenges faced internally and with the portfolio. The same topics surfaced consistently, regardless of fund size, technology maturity or portfolio size. One of these themes is migrating and modernising a monolithic application into a de-coupled, cloud native architecture.

This paper has been written to try and help structure the thinking which needs to go into a monolith decomposition programme.

For folks who are not in the Private Equity world, this playbook could still be of interest to you, as the principles, business case creation, patterns and strategy would largely remain the same.

This whitepaper has been split into 3 broad sections:

- Building the business case
- Creating a monolith decomposition strategy
- Patterns and implementation approaches

Whitepaper sections

This whitepaper has been split into three distinct sections:

- **Overview of monoliths:** This section assumes no/little prior knowledge of monolith applications. It will take you through what monoliths are, why they're a problem, how to compose a business case for decomposition and what the benefits of a decomposition programme are.
- **Technical overview:** Provides a deeper understanding of the technical considerations behind monolith decomposition as well as strategies for decomposing the monolith itself.
- **Conclusion:** Concludes the whitepaper and any areas of follow-up.

Naturally the technical overview is a little more detailed and hence isn't mandatory for all parties to read.

Overview of monoliths

Firstly, are monoliths inherently bad?

Before answering this question, I think it's important we agree what a monolith is. For this purpose we refer to a monolith as a fully functional application bundled into a single package and is deployed as a single unit.

So, do we believe monoliths are inherently bad? The simple answer is no. The term monolith is often used to refer to legacy applications which is both inaccurate and unfair. Depending on the type of monolith and the use-case, a monolith architecture could enable organisations to fulfil their business needs whilst ensuring that they don't have to deal with the complications of managing and operating other architectural approaches (e.g. a suite of microservices) – which bring their own challenges and necessary cultural and process transformations.

With this in mind, the rest of the paper will talk through monolith decomposition with the view that the target solution doesn't necessarily have to be microservices[1].

There are different types of monoliths which reflect years of application design shifts and approach changes. The three broad monolith types are:

- **Traditional monolith:** where everything is bundled together and there isn't a clear demarcation between aspects of the application. Trying to separate out the application is difficult because deep application knowledge and history of development is required
- **Modular monolith:** unlike a traditional monolith where everything is bundled together without a clear demarcation, a modular monolith has 'some' demarcations in the design of the application, known as 'modules'. Inter-module communications might still be complex with no clear understanding, but the modules themselves allow for a start on decomposition
- **Distributed monolith:** is a more modern form of monolith. The monolith has taken some core concepts of modern software design (i.e. microservices) but with traditional application principles. This causes an issue because the application will likely inherit the disadvantages of both approaches without necessarily realising the full extent of the benefits of either approach

[1] Microservices refers to an approach to application design and organisational design, known as a microservice architecture. In a microservice architecture, software is designed and composed of small independent services (or small applications) which can be developed and deployed on their own. Microservices will then use pre-defined 'contracts' to communicate with each other. Often Microservices are managed by small, independent teams.

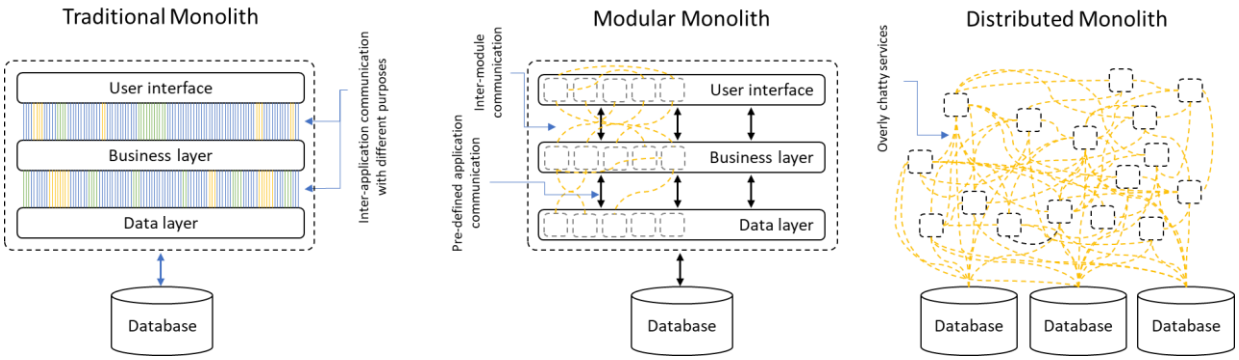

Figure 1 Depiction of different kinds of monoliths

You can find more detail about the monolith types in the Know the monolith type section of this whitepaper.

Building the business case for monolith decomposition

Building the business case requires a clear articulation of the problem currently at hand, what the benefits are of tackling the problem (i.e. benefits of the solution) and how to go about solving the problem.

What problems do monoliths pose?

When talking to teams about the issues posed by monoliths, they usually fall into one (or more) of the following categories:

- Application instability
- Application performance
- Slow development
- Limited / No innovation

Naturally, these issues are felt across multiple parts of the organisation. Often the areas which are really impacted are business, operations, technical and talent.

Business

The business is affected because the service or offering is not fit for purpose and might not always be available, ultimately increasing costs and decreasing potential revenue realisation.

Operations

With a failing monolith, Operations teams often find themselves having to react to unforeseen, high priority incidents related to the application. The monolith might become a key offender of high priority tickets on its own as well as causing upstream / downstream issues.

Technical

The technical team developing on the monolith can often find themselves resistant to change due to application complexity and maybe less satisfied in their job. This can cause tension between technology and business teams, further amplifying the discomfort.

Talent

Talent is often one of the hardest pieces of the puzzle. A monolith often houses complexity in implementation as well as business logic. In some cases, the monolith might be implemented in legacy technology for which skills are hard to find. This causes tension with existing personnel (across multiple areas of the organisation) who don't feel they can grow and then bring fresh talent into the organisation.

Articulating the business problem

When creating your business case, ensure you take time to understand the value lost due to the monolith across these key areas. This loss can be monetary, market related, vision related or talent. Articulating a clear need for change will be key to a cohesive business case.

Benefits of monolith decomposition

Undertaking a decomposition programme could have many business and technical benefits. It can also benefit personnel. We have highlighted some of the key ones here with a focus specifically on the opportunity for private equity portfolio companies.

Business

- **Decreased time to innovation:** Through functionality isolation, teams can focus on key business functionality and innovate without needing to deal with the implications of a monolithic architecture
- **Availability to talent:** With legacy applications, often the changes require deep business knowledge and (depending on the age of the monolith) the technology skills might not be as readily available in the market. With a modern architecture, on a modern technology stack, new talent can be more easily found on the market and training programmes created to upskill existing personnel

- **Adaptability to changing expectations:** Through a modern architecture which allows new functionality to be easily added and deployed to serve customers
- **Exit strategy:** Having a modern, resilient, scalable solution for key business applications can result in higher exit valuations

Technical

- **Scalability:** With modern architecture, functionality which is more highly used can scale seamlessly and independently of the rest of the application
- **Availability and operations of application:** If designed correctly, automation can respond to operational events and can ensure that only parts of the application which are required are available, saving cost and ultimately reducing operations efforts
- **Adoption of new technology:** Allowing for innovation and differentiation in customer offerings, providing a competitive edge

Personnel

In a couple of cases, I've seen technical teams embark on a programme of change to adopt new technologies for new opportunities, whilst simultaneously supporting legacy systems. With the right framework, this can be quite effective at retaining technical teams and supporting their growth. However, it is important that the legacy system still receives the necessary attention and is not further neglected.

Creating a monolith decomposition strategy

Having a clear strategy to tackle the monolith decomposition will often be a key difference between a successful programme and one which fails. No two decomposition programmes will be the same, as the monolith being decomposed is unlikely to be exactly the same.

To ensure the strategy covers all the necessary areas, consider the key principles listed below.

Some important principles

Whilst not an exhaustive list, these principles are key to any successful strategy. Consider these key areas as part of the overall strategy, ahead of any programme implementation work.

Think process, people and technology

Transformation goes beyond just the technology. As the monolith is being decomposed, consider if there are changes required to the business processes to support the intended outcome.

Workforces considerations are also key:

- Are the technical teams trained up on the new IT processes?
- Are the teams structured in a way which enables close collaboration?

It's important to ensure all people-related matters have been considered and any necessary changes are made ahead of a programme of work being undertaken.

Bring the business along on the journey

Often decomposing a monolith means taking time away from bug fixes, new feature developments and discussions about how processes are going to change. Ensuring that the business is onboard with the programme, support and embrace the transformation, and understand the benefits will be vital to a successful transformation.

Know the starting point

Decomposing an application which has hundreds of developers working on it, is business critical and has been in production for years will be a significant challenge. Knowing the target and working back to the required understanding needed from the current application will support gaining initial momentum.

Often the required knowledge can be siloed in teams and amongst individuals. Consider holding workshops and ensuring shared team knowledge transfer sessions to obtain an overall shared understanding, de-risking the issue of knowledge on the programme.

Be clear on the intended outcome

Understanding the target solution, even at a high level will help to define where the programme can start. Given that these programmes can be large and complex, iteratively building on a skeletal target solution can help the programme get started and incremental value be delivered as the fuller vision is developed.

Decide whether the entire monolith is being replaced

Whilst appearing counter-intuitive, it is not necessary for the whole monolith to be replaced. If clearly defined business functionality needs to be decomposed and modernised, providing clear value, then consider a target in which the business functionality is decomposed from the original monolith but with the overall monolith remaining a part of the overall solution.

This does not need to be an initial programme decision, rather it could be iterative. Employing an approach of "I'm trying too hard, this isn't worth it", with a clear view on where the line is will help ensure the programme does not come unstuck in analysis paralysis.

Microservices are not the only option

Microservices are not a silver bullet. The architectural approach requires a significant amount of business cultural and process change to realise its potential value. These cultural and processes changes include (but are not limited to): changing team cultures to allow for independence amongst teams; smaller team sizes focused on individual areas; lifecycle ownership within teams; changes to deployment processes to allow for smaller, more frequent production releases; allowing (some) heterogeneity within technology; etcetera.

Often microservices are adopted without addressing key areas including poor coding standards; no DevSecOps[2] processes; lack of automation; and traditional process gates could result in a more complex environment. Consider whether all aspects are being addressed through the transformation to realise the true value potential.

If adopting microservices, do it right

Microservices require significant cultural and processes changes, as mentioned in Microservices are not the only option principle. If the target solution includes microservices, ensuring key considerations on the architecture are made:

- How granular will these microservices be?
- Have all technical considerations been made?
- What DevSecOps processes will be put in place?
- Who will own the microservices?

This strategic thinking and planning with the right representation (including technical, operational and business teams) is important to ensure overall programme success.

What if the monolith cannot be decomposed immediately?

Sometimes monoliths can't be decomposed because the skill isn't available in-house or the budget to undertake such a programme of work isn't available.

In this scenario, there is an option known as 'digital decoupling' which can help yield some value in the interim.

Coined by Accenture, it's an approach where applications are built around a monolith to help continue revenue growth and outward innovation. The application being built around the monolith will communicate with the monolith application but will be independent of the actual application.

The approach allows organisations to respond positively to market changes and customer demands whilst not having to embark on a full-scale monolith decomposition.

[2] DevSecOps is an amalgamation of development, security and operations. It's an approach to design, process and organisational culture which embeds automation and security into the entire IT lifecycle.

Often the areas of focus will be areas that are valuable to the organisation. This results in one of three areas being looked into:

1. **Decoupling customer channels:** from any system or process complexity which allow for a more modern and agile offering to customers
2. **Decoupling data:** unlocking the power of data from legacy systems by adopting big data technologies to garner insights and aid in decision making
3. **Decoupling from central systems:** surfacing information and functionality which will still feed into central systems but will be separate, allowing for flexibility and scalability

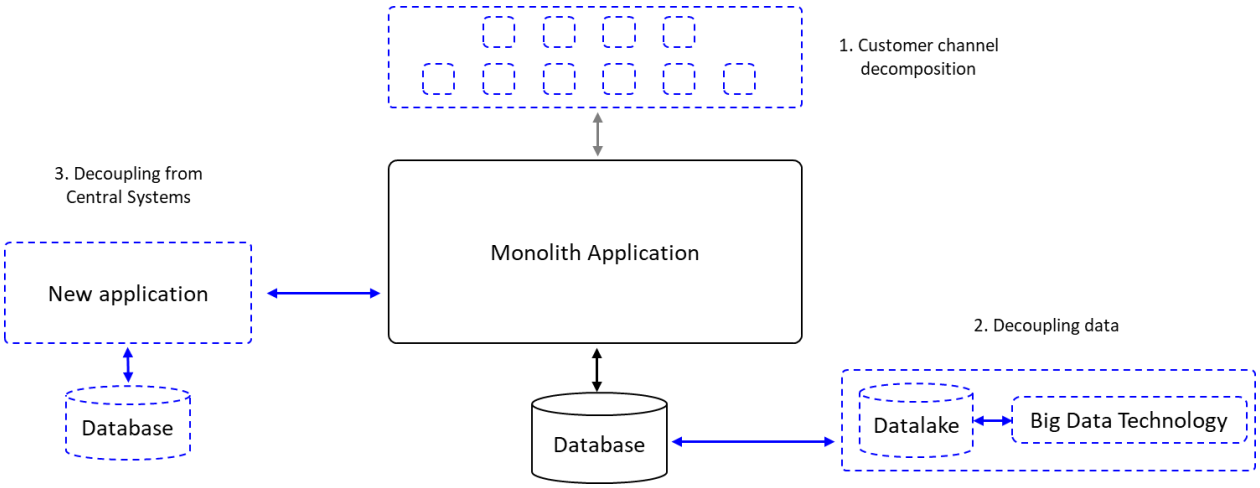

Figure 2 Depiction of different digital decoupling options

Whilst digital decoupling is a way of attaining some value in the interim, it is ***not*** a replacement for monolith decomposition. It can however be used as a starting point to help with the ultimate decomposition of a monolith.

Technical overview

Know the monolith type

Not all monoliths are the same. Understanding the kind of monolith helps decide the optimal course of action to support the intended outcome. It could also directly impact on the options available for the target solution in-line with other programme factors (e.g. timescales).

Traditional monolith

A traditional monolith application is one where the user interface, business logic and data access layer are bundled into a single tier and deployment unit. The application will have several, tightly coupled functions and often perform multiple related tasks. Libraries within the code will be shared and functions inside of the overall code base will communicate through abstractions layers.

Modular monolith

A modular monolith is a monolith which has a modular build with clearly defined functional boundaries and clear dependencies. Each module is independent and can be worked on independently without affecting any other modules.

The deployment of a modular monolith will remain a single unit.

Distributed monolith

A distributed monolith is a monolith that is deployed like a (micro)service architecture but is built with monolith principles in mind. It leverages a distributed systems architecture but is not designed to do so efficiently or reliably. The result is that the overall application has the inflexibility of monoliths coupled with the complexity of microservices. This particular form of monolith is particularly painful as the application does not enjoy the benefits of either architectural pattern.

Distributed monoliths are often the result of attempting to build a (micro)services architecture without considerations for key architectural components or the necessary cultural or processes changes.

There are a few key indicators that you have a distributed monolith:

- Services are tightly coupled
- Services are overly chatty (i.e., there is a strong and consistent dependency between functionality)
- Services deployments are in sequence and need to be choreographed precisely
- Scaling individual services is a challenge

If one or more of these indicators can be used to accurately describe the application then it is likely a distributed monolith.

Patterns for monolith decomposition

There are some accepted patterns to approach the Monolith decomposition.

Strangler Fig

Coined by Martin Fowler, Strangler Fig ('Strangler') refers to incrementally replacing specific pieces of functionality with a new system alongside the legacy system. As the functionality is slowly replaced and client traffic re-directed, the old application is "strangled", allowing it eventually to be decommissioned.

Often when employing this approach, a façade is created to shield incoming requests to the application.

Strangler is a pattern when a significant portion of the application will be replaced.

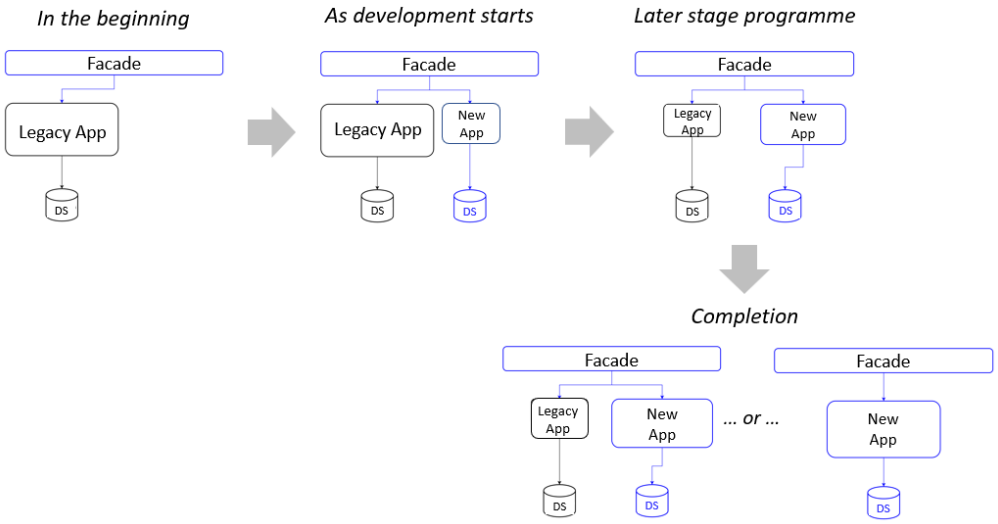

Figure 3 Depiction of the Strangler approach to monolith decomposition

Considerations

- Ensure the stranger façade does not become a single point of failure or cause performance issues
- Structure new applications and services in a way that they can easily be intercepted and replaced in future Strangler migrations
- Post-migration decides whether the Strangler façade will be removed or evolve into a legacy application adaptor

- Consider how to handle services and data stores that are potentially used by both new and legacy systems. Make sure both can access these resources side-by-side

Branching by abstraction

This approach uses the creation of an abstraction layer to help segment the application, allowing for the components to be replaced incrementally. Client traffic is initially directed to the abstraction layer allowing for changes to be made to functionality behind the abstraction layer. Once the functionality has been replaced, clients can continue to interface with the abstraction layer if required.

Branching by abstraction works at a lower level of abstraction to Strangler. It focuses on component replacement rather than full application replacement.

Figure 2 below depicts this in evolutionary terms for the component and client traffic.

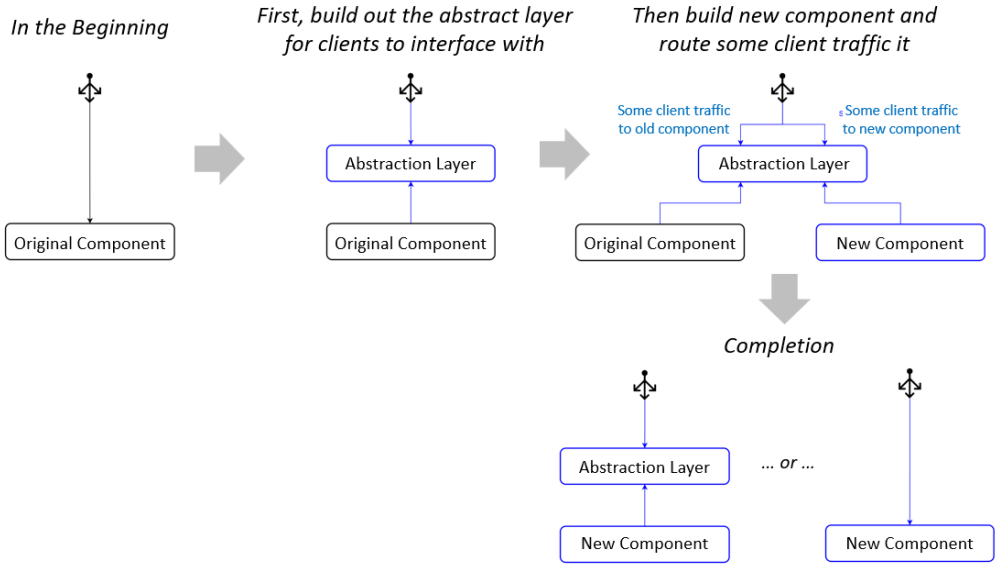

Figure 4 Depiction of branching by abstraction approach to monolith decomposition

Considerations

- Consider the application architecture and whether the branching by abstraction approach will be successful or not
- Consider the different implementation approaches for the abstraction layer based on your application as well as the fate of the interface on migration completion
- Post-migration decides whether the abstraction later will remain or be removed
- Ensure application performance is still acceptable with the interface in place
- Consider development processes or approaches which might need changing to ensure you can deploy changes and redirect client traffic in a timely manner

Implementation approaches

There are multiple ways of practically approaching the decoupling of a monolith. When considering options, it is important to take into account the organisation's structure and consider which teams will be managing the new application going forward.

Decomposition by domain

In this approach, the monolith will be broken into domains based on the organisation's individual business units. Once decomposed, the domains are assigned to the business units to own and move forward.

This approach allows for the decomposed domains to be loosely coupled and gives business units the opportunity to integrate the functionality into a service portfolio.

Example

A retail organisation has a customer team for sales, a marketing team for content management and a warehouse team for inventory. If the retailer had a monolith application which handles the e-commerce website, its content management and also inventory delivery in a single deployable unit, then the application could be split into 3 distinct parts. The e-commerce portion of the application would be managed by the customer team in sales, the content management functionality split out and managed by marketing, and inventory delivery managed by the warehouse team.

Considerations

- Ensure an in-depth understanding of the business (as without this, the application cannot be split and assigned accordingly)
- Ensure any major business model changes are accounted for when considering the split of the application
- Consider the modularity of the application and whether it allows for this form of approach
- The architecture of future applications will need to be considered such that benefits can be gained from alignment without being highly coupled to business model
- The splitting of any existing data alongside master copies of data will need to be clearly defined

Decomposition by user journey

In this approach, a thorough understanding of the monolith and user experiences, journeys are split out and developed individually. Each journey can be developed upon individually and subsequently assigned to a team or business unit.

Example

For example a bank has a legacy mainframe which handles current account originations, loan originations and account management for current accounts. The mainframe can be split into three based on the user journeys: current account originations; loan originations; and current account management.

Considerations

- Ensure an in-depth understanding of the monolith to allow for user journeys to be split out
- Consider the modularity of the application and whether it allows for this form of approach
- Clearly define ownership of the applications which will arise from the splitting of the application
- Consider any coupling issues which arise from the splitting of user journeys

Decomposition by capabilities

Decompose the application by capabilities offered by the application and group accordingly. Once grouped, assign capability groups to a team to own and develop. Please note, this will be similar to decomposition by domain if a decision is made to align capabilities to business units.

Once assigned, this will allow the individual teams to manage those applications with separate codebases and loose coupling between applications.

Example

For example, a media company has an all-in-one application which provides a CRM, customer portal for collaboration; customer profile management, payments system for subscriptions and governance for regulatory purposes. This application can be split based on its broad capabilities: CRM; customer collaboration portal; customer profile management; payment system; and regulatory governance. Some of these capabilities can be grouped together and managed by a single team (e.g. CRM, customer profile management).

Considerations

- Ensure an in-depth understanding of the monolith to allow for capabilities to be split out
- Consider team ownership for each of the new capabilities
- Consider approach for common functionality

Implementation advice

Choosing the right approach is important and often relies on the capabilities available to the organisation and programme team. Do note that when approaching a decomposition, more than one pattern can be used to support the overall decomposition.

Technologies which can support monolith decompositions include:

- Virtual Machines technology (Azure Virtual Machines)
- Container technology (Azure Container Instances, Azure Kubernetes Service)
- Serverless compute technology (Azure Functions, Azure Container Apps)

Please note that other supporting services not listed above, will be required to ensure a working solution.

Conclusion

Monolith decomposition is hard and there is a lot of preparation which needs to go into any programme of work to tackle it. It is crucial that ahead of starting any work the following is understood by all parties:

- A clear vision backed by a strategy
- A shared understanding of the problem statement
- A clear view of the benefits

Use this whitepaper to help guide that thought process and support any / all necessary conversations to achieve an optimal outcome.

If there is a desire to use Microsoft technology to help deal with a monolith, please get in touch with a Microsoft representative.