Deloitte.

Mainframes to Microservices

Lessons Learned in Modernizing High-Demand Applications

Lori Olson
Deloitte Consulting LLP

Agenda

1 Introduction

2 Getting to Microservices

- A Lessons Learned During the Requirements Activities
- B Where Should We Start with the Design?
- C Determining the Priority Quality Attributes & Maintaining the Contracts
- D Strangling the Mainframe & Applying the Patterns
- E Organizing Around the Architecture & Achieving Functional Parity

What if you have to replace an application processing hundreds of millions of documents per day for a government agency or healthcare provider?

Is it possible to choose microservices and still achieve the same stability as software that has been continually optimized over a 25-year lifespan?

What factors should you consider when defining the architecture to meet this challenge?

What are Microservices?

Microservices are an architectural style that structures an application as a **collection of loosely coupled services**, to implement business capabilities

Demonstrated Benefits of Microservices Architecture

Continuous deployment of large, complex capabilities

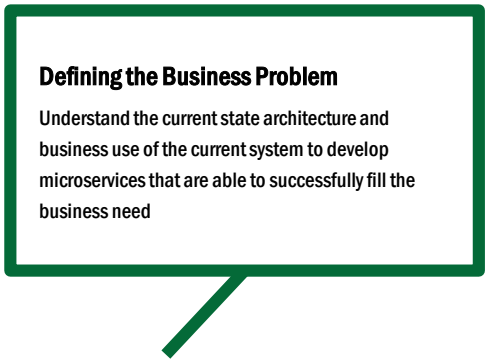
Enables an organization to evolve its technology stack

Allows for scalability and improved maintenance

Lessons Learned During the Requirements Activities

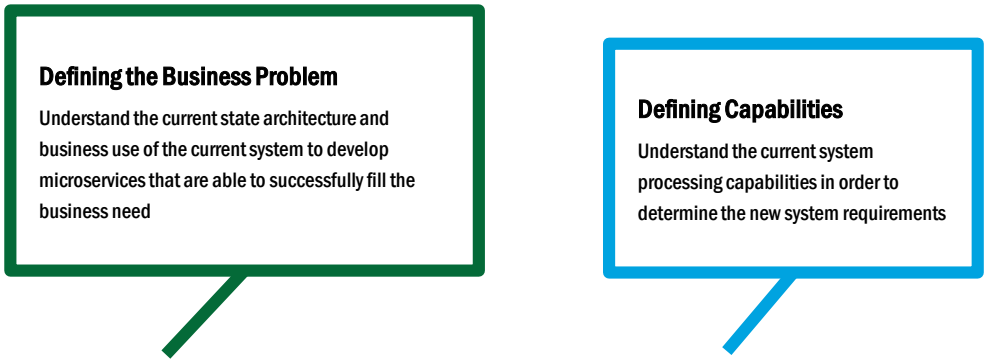
How do you **know what you should build**?

Lessons Learned During the Requirements Activities



How do you **know what you should build**?

Lessons Learned During the Requirements Activities



How do you **know what you should build**?

Lessons Learned During the Requirements Activities

Defining the Business Problem

Understand the current state architecture and business use of the current system to develop microservices that are able to successfully fill the business need

Defining Capabilities

Understand the current system processing capabilities in order to determine the new system requirements

How do you **know what you should build**?

Using Documentation and Process Definitions

Review any available documentation, source code, and system process flows to gain an understanding of the current system. Reverse engineer the source code to diagram a process flow tied to the **business need**

Lessons Learned During the Requirements Activities

Defining the Business Problem

Understand the current state architecture and business use of the current system to develop microservices that are able to successfully fill the business need

Defining Capabilities

Understand the current system processing capabilities in order to determine the new system requirements

Working with the Subject Matter Expert

Engage with the current subject matter expert if available. Develop a strawman process flow based on what you know and confirm with the SME to identify gaps in your understanding

How do you **know what you should build**?

Using Documentation and Process Definitions

Review any available documentation, source code, and system process flows to gain an understanding of the current system. Reverse engineer the source code to diagram a process flow tied to the **business need**

Lessons Learned During the Requirements Activities

Defining the Business Problem

Understand the current state architecture and business use of the current system to develop microservices that are able to successfully fill the business need

Defining Capabilities

Understand the current system processing capabilities in order to determine the new system requirements

Working with the Subject Matter Expert

Engage with the current subject matter expert if available. Develop a strawman process flow based on what you know and confirm with the SME to identify gaps in your understanding

How do you **know what you should build**?

Using Documentation and Process Definitions

Review any available documentation, source code, and system process flows to gain an understanding of the current system. Reverse engineer the source code to diagram a process flow tied to the **business need**

A Picture is Worth More Than 1000 Words

The outcome should be a business process model that reflects the current state processing, the business rules for each step in the process, and the current system architecture. This helps you to identify the most appropriate candidates for modernization

Where We Started with the Design…

- **Determine the seams in the system:** Seams in the system are where processes or activities are integrated or have a change of control or responsibility
- **Use those seams** to translate multi-purpose programs into single purpose components, finding places where changes have the least potential risk to the processing pipeline
- **Prioritize changes** to the monolithic processes decomposed into individual components. Only those components are changed. It is important to maintain the inbound and outbound contracts so upstream and downstream systems are not impacted
- **Understand the mainframe logic** for solving the problem, understand the challenges the initial implementation faced and how those were addressed

How we learned from the Mainframe

Annually: ~3.5 Billion records

Daily Peak: ~625 Million records

- To process large volumes, the data was sorted in memory and processed in chunks
- Reference data was stored in memory to reduce time spent with read/write cycles
- Optimized the microservices to utilize memory for processing and avoid disk writes when possible

Determining the Priority Quality Attributes

What quality attributes were required?

Performance

The system needed optimal performance in order to process 3.5 billion records annually

Scalability

It had to be scalable to accommodate increased annual intake

Maintainability

It needed to be developed in a modern language to support ease of maintenance and a larger pool of available resources

How would we architect the solution?

1 A microservice component for each single purpose activity, allowing for concurrent throughput

2 Designing for horizontal scalability for independently sizing services based on process demand

3 Separating the activities also allows for concurrent, asynchronous processing and independent changes

Maintaining the Contracts

Discover the legacy APIs

- Break down the business components
- Find the seams in the technology and the handoffs between the different subsystems

Creating new, maintainable APIs

- Version APIs so that any breaking changes will be versioned
- Nest response objects to allow for updates to core response objects without needing special rules

The importance of documentation

- Keep the documentation as part of the code
- Generate documentation artifacts for distribution as part of the releases

Apply these practices when creating APIs for legacy contracts and to new APIs

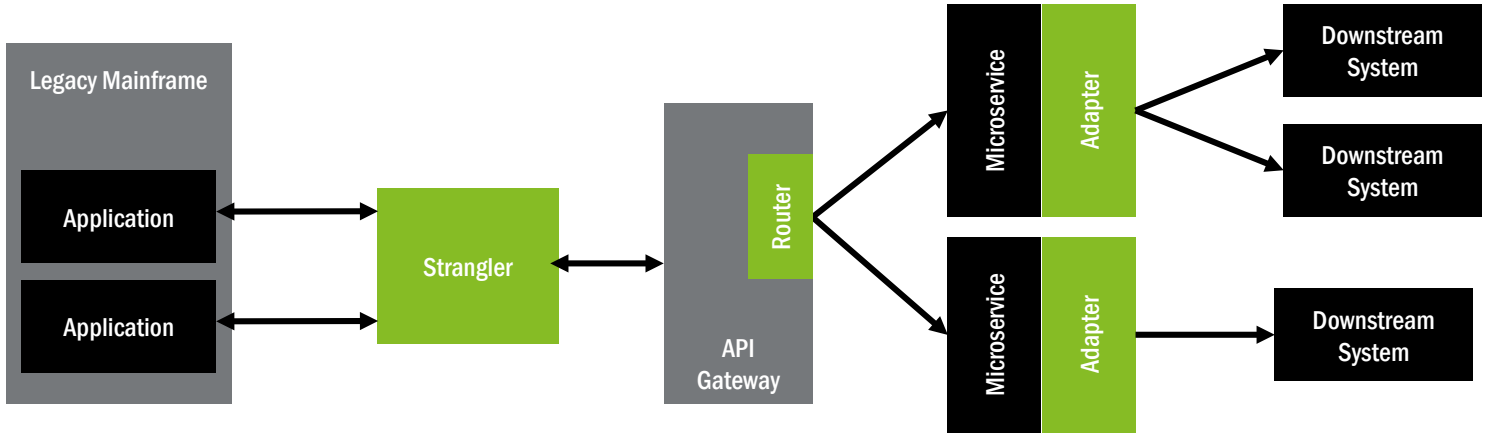
Strangling the Mainframe

Patterns we used and why

Strangler
Modernize the system by strangling out component by component of the legacy system

Router
Orchestrate the monolith processes across multiple microservices

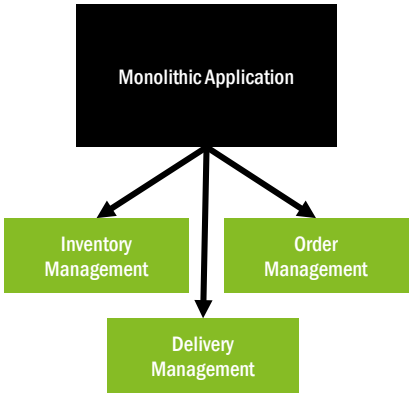
Adapter
Maintain the contracts in the legacy environment with an adapter to the new services



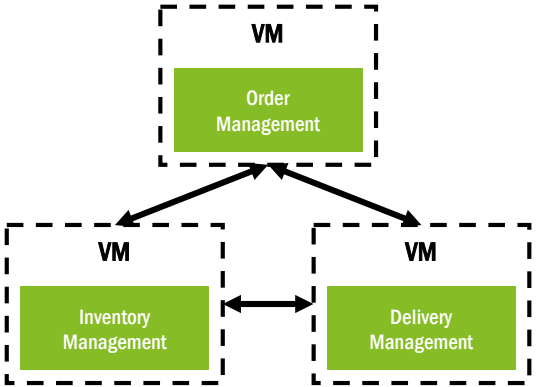
Applying the patterns

Patterns we used and why

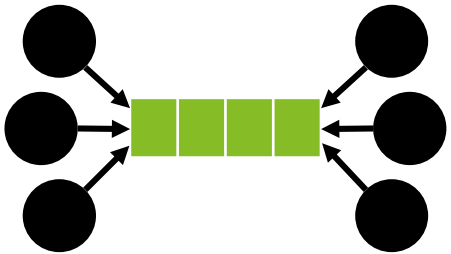
Decomposition
Separate services by business capabilities and subdomains



Service Independence
Deploy each service independently



Messaging
Use asynchronous messaging for inter-service communication



Organizing Around the Architecture

"Any piece of software reflects the organizational structure that produced it"
– Melvin Conway

Microservices are not only about delivery, they are also about ownership

Builds, Deployments and Transmittals	Data Access	Performance & Load Testing	Testing and Test Tooling	Production Support and Monitoring

Not every organization is ready to change

- Not very organization is ready to organize this way, so it may create redundancy in a project and external teams
- Foster these ideals to reduce the risk due to missed requirements or misunderstanding when there is handoff to another team

Achieving Functional Parity with the Quality Attributes

What is functional parity?

- Providing the same outcomes from the same inputs
- Necessary to maintain the contracts with legacy components

Testing Strategy

- Side-by-side functional testing with the legacy system using the same inputs for both systems and then comparing the outcomes
- Performance testing with the same request volume for normal and production peaks to validate and tune the modernized system
- Highly risk adverse organizations can do a full side-by-side production comparisons to verify parity between legacy and modernized systems before retiring the legacy system

Thank You

Contact Information

Lori Olson – Specialist Leader

Systems Integration – Application Architecture

Deloitte Consulting LLP

loolson@deloitte.com

LinkedIn: https://www.linkedin.com/in/lorio/

Deloitte.