AGILE AND AUTOMATION CONCLAVE 2018

Monolith to Microservices

Incremental & Iterative transformation

Mohammed Shuaib Mumtaz
Mohammed Shuaib  
Lead ATCI, Lean Architecture  
Md.Shuaib.Mumtaz@accenture.com

#Microservices,  
#Evolutionary Design,  
#DevOps
AGENDA

• Monolith Application Issues and Challenges
• Decomposition - Key Patterns and Concepts
• Systematic Refactoring Strategy
• Case Study
• Demo and Code Walk Through
• Q&A
The Need

• Why is there a need to convert a fully functional running monolithic application to Microservices?

• Is the conversion worth the pain and effort?

• Should I be converting all my applications to Microservices?
Issues with Monolithic Applications

- Gets bigger and bigger
- Everything is shared (coupled)
- Change is discouraged
- Side effect accompany change
- Scaling is challenging
- Architectural changes are impossible
- Steep Learning Curve
- Longer time-to-market
How to Decompose Big, Scary Monolith Application?

Using Refactoring approach:
- Branch by abstraction
- Strangulation
- Feature Toggles
- ...

Best Done Incrementally!
Branch By Abstraction

- Make a large-scale change to a software system in gradual way
- Release the system regularly while the change is still in-progress.
- Create an abstraction layer between the desired changes and the remainder of the application
- Enable evolutionary design of the application architecture while preserving the cadence of value delivery
- Decouple design lead time from release lead time
Branch By Abstraction Pattern

**STEP 1** – Identify the Component to Replaced

**STEP 2** – Create an Abstraction Layer & Refactor the System to use Abstraction Layer

**STEP 3** – Remove the old implementation

**Key benefit**: Code is working at all times throughout the re-structuring, thus enabling continuous delivery

Excerpt from Source at: https://martinfowler.com/bliki/BranchByAbstraction.html
Straggler Pattern

- Create a new system around the edges of the old one and letting it grow slowly until the old system is strangled

**Transform** – Create a parallel microservice

**Co-exist** – Incrementally redirect the traffic from the legacy to microservice

**Eliminate** – Eliminate the legacy module
Feature Toggles Pattern

Uses:

- Allows continuous delivery of small increments to production
- Reduces branching-merging overheads
- Helps deliver new functionality to users rapidly but safely
- Dynamically controlling system behavior
- With canary deployments which allow developers to have features incrementally tested by a small set of users
Steps to Take from Monolith to Microservices
Case Study Pet Clinic App
As-is Architecture review - Component Diagram
To Be Architecture - Component Diagram
Steps to Take from Monolith to Microservices

Step 1
- Identifying What Needs to be Migrated to Microservices
- Start with the least complex modules in the legacy system that will have the greatest benefits.

Step 2
- Apply Branch By Abstraction Pattern
- Introduce an abstraction layer in front of the old component
- Leave the old legacy code as is
Steps to Take from Monolith to Microservices

Step 3

- Apply Strangulate Pattern
- Create a parallel microservices implementation for the identified modules
- Create an independent Database for each Microservice
Steps to Take from Monolith to Microservices

Step 4

- Apply the toggle Pattern
- Introduce a run-time configuration toggle behind the abstraction layer
- Deploy the refactored code in production
- Incrementally redirect the traffic from the legacy module to the newly created microservice
- Sync the database of Legacy module and Microservice
- We can dynamically revert to the old component in the event of a failure
Steps to Take from Monolith to Microservices

Step 5

- When development and Testing is complete, and the traffic is completely redirected to the microservice, eliminate the legacy module and refactor code and Database to delegate to the new component.

Diagram:

- Front end <<consumer>>
- Proxy/Toggle
- Visit <<Abstraction Layer>>
- Clinic Service <<Flawed Supplier>>
- Visit Microservice <<Under Development>>
- DB
- Sync the Databases
Monolith to Microservices Journey

Sprint 1
- Identify the Affected Module/Service to be refactored as Microservice
- Decouple Visit Functionality
- Create a new Visit Microservice
- Configure the Microservice to do CRUD Operation on Monolith Centralized DB
- Implement Toolgiz to call Microservice for specific Users
- Create a new Interface and Implementation class for VisitService in Monolith Application
- Call the Microservice if the Toolgiz is

Sprint 2
- Configure Mongo DB for Visit Microservice
- Update Visit Microservice to implement Database Transactions using MongoDB
- Update Monolith Microservice to Sync Data between Centralized DB and Microservice DB
- + Add another card

Sprint 3
- Update Monolith Application and completely remove Visit functionality
- Remove Toolgiz functionality
- Add a API Gateway and Proxy server to redirect all Visit requests to New Visit Microservice
- Update Monolith Database to remove Visit functionality related tables
- + Add another card

Sprint 4
- Identify Other Candidate Services to be refactored as Microservices
- Decouple Vet and Customers Functionality
- Implement Stanagulate, Toggle and Branch by Abstraction Patterns
- Update API Gateway to send requests to respective microservices
- + Add another card

Sprint 5
- Implement Netflix OSS Stack Tools (Eureika, Zipkin, Admin Server, Hystrix) to Trace, Monitor and Debug and locate Microservices
- Update and configure Microservice with Netflix OSS Tools
- + Add another card

Agile and Automation Conclave 2018
Code Walk Through
References

• Pet Clinic App Source Code: https://github.com/spring-projects/spring-petclinic
• Branch By Abstraction Pattern: https://martinfowler.com/bliki/BranchByAbstraction.html
• Strangulation Pattern: https://www.martinfowler.com/bliki/StranglerApplication.html
• Feature Toggles Pattern: https://martinfowler.com/articles/feature-toggles.html
• https://thenewstack.io/from-monolith-to-microservices/
Q&A