talentica
DREAM · BUILD · DISRUPT

# Modernize the product to improve scalability & reliability

## Contents

# Monolith to Microservice

# Accelerated feature development by 2x for a legacy channel marketing software

## BACKGROUND:

The channel marketing automation company collaborates with enterprises, accelerating channel-driven demand generation, strengthening partner engagement, and growing channel revenue.

It wanted to modernize its platform by moving from monolith to microservices architecture to improve the feature cycle time and increase user engagement.

## CHALLENGES:

The implementation involved the challenges below-

- Understanding existing legacy systems by analyzing codebase and architecture to learn about dependencies and potential areas for improvement
- Co-existence of legacy and new systems for live users
- Prioritizing the right features by developing a strategic roadmap guiding modernization efforts

## SOLUTION:

We developed a solution with-

- **Strangler fig design pattern**
  - Replaced functionalities of the legacy system with new microservices or modules.
  - Ensured seamless co-existence and gradual transition from the legacy system to the new architecture.
- **Anti-Corruption layer**
  - Implemented an anti-corruption layer to shield microservices from being influenced or corrupted by legacy system dependencies.
  - Defined clear interfaces and protocols to prevent contamination of new services by legacy code or data structures.
- **Micro frontend**
  - Integrated a React application as an iframe within the existing. NET application.
  - Enabled independent development and deployment of frontend modules while maintaining a cohesive user experience.
- **Microservice architecture with EKS (Elastic Kubernetes Service)**
  - Adopted a microservice architecture pattern leveraging EKS for container orchestration and management.
  - Enabled scalability, fault tolerance, and efficient resource allocation through Kubernetes-based infrastructure.
- **HTTPS and Queue (SQS and SNS) for Communication between Legacy and New System**
  - Implemented HTTPS for secure communication between the legacy and new systems, ensuring data integrity and confidentiality.
  - Utilized Amazon Simple Queue Service (SQS) and Simple Notification Service (SNS) for reliable and asynchronous messaging between components of both systems.

## RESULTS:

- Improved development speed and go-to-market strategy
- Simplified features with better user experience.
- Accelerated customer acquisition.
- Enabled direct integration to platform - New Business Offering.

# SOA to Microservices

## Handled 10K rows and 600 columns concurrently for a supply chain platform

### BACKGROUND:

We worked with a supply chain management software platform that has operations in 20 countries with 10K buyers and 200K suppliers on the platform.

Its existing solution was not suitable to handle scalability. With business expanding, the company decided to modernize the system to improve concurrency control.

### CHALLENGES:

- The existing solution was using an online excel to manage Bill of Material (BOM) from buyers and responses from sellers. But it could handle only 500 rows and 20 columns concurrently.
- The company had to process larger BOMs manually. It used to take 5-6 months to reward single BOMs, and they wanted to reduce it to 5-6 weeks.
- They wanted to modernize their system to handle 10K rows and 600 columns concurrently for 100 users.

### SOLUTION:

- We built a frontend like excel with responsive backend APIs, which could handle a concurrency of 3K requests/second. It allowed 100 users with different roles to make changes/format the sheet simultaneously.
- To support 10K rows and 600 columns for every BOM, we migrated existing SQL to scalable NoSQL (MongoDB) and migrated .NET monolith to node.js based microservices.
- For awarding, we created an online excel supporting over 10 million with search, filtering and sorting options using our own database on Lucene search.

### OUTCOME:

Scaled a platform to handle more than 10K rows and 600+ columns for 100+ concurrent users.

# Monolith to Modular Monolith

# Rearchitected a Recruitment Automation Software to Accelerate Feature Development by 2X

## BACKGROUND:

We teamed up with a recruitment automation platform that helps recruiters organize recruitment for fast growing lean organizations. The initial product was built 17 years ago. Since then, the client added many functionalities.

The existing on-premise version faced revenue growth challenges as many organizations opted for the pay-as-you-go model. Adding new functionality to the legacy product also consumed a lot of time and it inspired the client to think about re-architecture.

## CHALLENGES:

- **Architectural limitations**
  - Our client was spending too much time adding new features as the process was impacting existing ones. Fixing these issues was time-consuming and affected the developer morale.
  - It was built using monolith architecture and old technologies and was not supporting multiple databases. It had a lot of boilerplate code for security and session expiry, and comprehensive test cases were not written.
- **Poor user experience**
  - User interactions were not intuitive. Training recruiters used to take a lot of time. It was not supporting the custom reporting and analytics.
- **Difficult to upgrade product versions**
  - Upgrading to the latest version was difficult as multiple versions of the product were created to implement customer-specific features.

## SOLUTION:

- **Migrated to a single-page application.**
  - The tech stack was migrated from Java 7, JSP Servlets, SOAP services to a single-page app using Java 8 + Spring Boot,
  - Vue.js and Angular JS were used for the front-end and Solr-based search engine, Redis cache.
- **Migrated from monolith to a modular monolith**
  - Separate modules were implemented for admin, recruitment, chatbot, authentication, and integrations with Microsoft Teams, Google Meet and Hackerrank.
- **Implemented multi-tenancy**
  - To lower the cost of adoption and open new revenue streams from small and medium-sized companies.

## OUTCOME:

- **Re-architecture**
  - Increased market share by penetrating small and medium-sized companies.
  - Accelerated feature development by 2x.