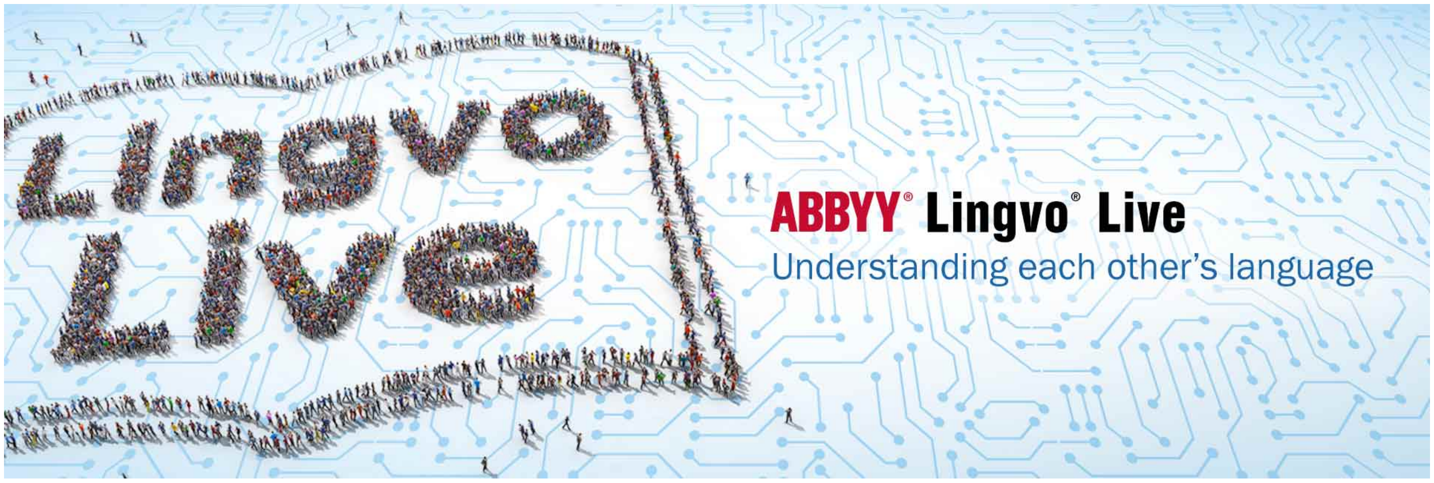
# ABBYY's social networking service evolves on Azure Service Fabric

April 26, 2018



*Authored by Eugene Agafonov from ABBYY, in conjunction with Alex Belotserkovskiy from Microsoft.*

This post is part of a series (https://blogs.msdn.microsoft.com/azureservicefabric/tag/customer-profile/) about customers who've worked closely with Microsoft on Azure Service Fabric (https://azure.microsoft.com/en-us/services/service-fabric/) over the last year. We look at why they chose Service Fabric, and we take a closer look at the design of their application.

In this installment, we profile ABBYY and their Lingvo Live social networking service, which was redesigned to run on Azure using Service Fabric and a microservices-based architecture.

ABBYY is a software company that provides optical character recognition, document capture, and language software for PCs and mobile devices. More than 40 million people from over 200 countries use ABBYY products, technologies, solutions, and services.

In 2015, ABBYY went social with the launch of Lingvo Live, a cross-platform social networking service for language learners. Lingvo Live provides free access to more than 130 dictionaries for 14 languages and other useful tools for people interested in languages and cultures, including students, teachers, business travelers, and professional translators.

# The challenge of legacy code

To get Lingvo Live to market quickly, the ABBYY developers relied on many existing ABBYY technologies, some decades old. The code base included DCOM components, native libraries, and .NET services written in the past by different teams, but it worked fine, and the developers didn't want to refactor it without knowing whether the market would embrace their idea. They launched as is with a website, mobile clients, and back-end services, hoping to gather user feedback about the product and validate their idea. At the time, Lingvo Live was hosted using their own IT infrastructure. The limitations of this setup soon became apparent.

The development team created the Lingvo Live website and the HTTP API using ASP.NET Web API, a familiar web platform with support for mobile apps. The goal was to make Lingvo Live accessible from any device, and for the first implementation, the team targeted iOS and Android. Users quickly began consulting dictionaries from their mobile devices, posting translation questions to a growing community, and sharing their experiences with other language learners.

When Lingvo Live launched, it consisted of multiple language and search components, some of which were more than 20 years old and very difficult to update or maintain. Windows Server Network Load Balancing (NLB) was used to enhance the service's availability and scalability. Key components included:

- **Lingvo Engine**: The platform that supports ABBYY's existing dictionaries for a large number of languages via a COM interface.
- **Translation Memory**: Custom logic that enables applications to use aligned text fragments with translation examples. Text fragments are aligned when they have the same meaning, and great examples gives context to the user who wants a word or phrase translated.
- **Full-text search service**: Based on internally developed technologies, including a morphology engine for many different languages.

“ We could not afford to rewrite the application to leverage modern technologies and a microservices approach all at once, so we needed an evolutionary approach. Azure and Service Fabric more than fulfilled our requirements--and solved our performance and availability problems.”

—Eugene Agafonov: Development Manager
ABBYY

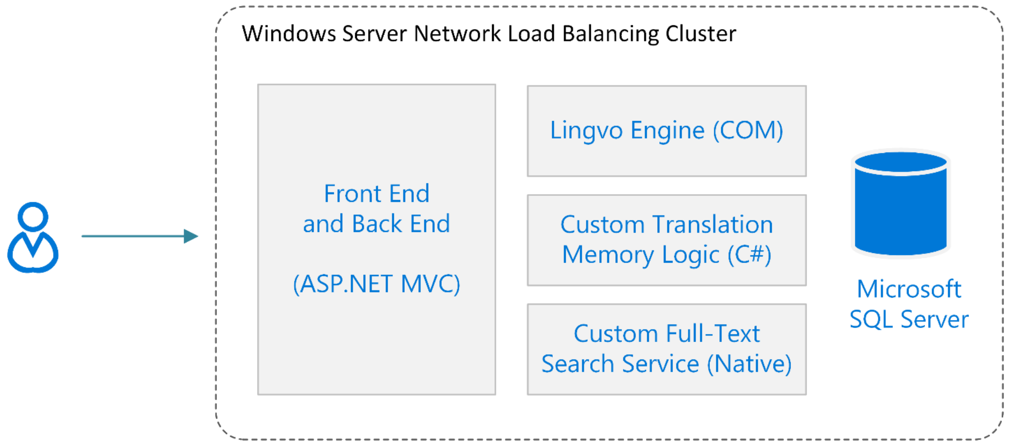

Windows Server Network Load Balancing Cluster
Front End and Back End (ASP.NET MVC)
Lingvo Engine (COM)
Custom Translation Memory Logic (C#)
Custom Full-Text Search Service (Native)
Microsoft SQL Server

*Figure 1. ABBYY Lingvo Live conceptual architecture at launch.*

After deploying the first version into production, the team at ABBYY ran into a few maintenance issues. For example, even a minor change to the website, such as editing text or updating HTML and CSS required them to redeploy the entire website.

In addition, deploying the application meant deploying multiple components in a multitude of ways. Many of the legacy components had dependencies and couldn't be easily changed. Some components required manual installation and configuration. Even if the team wanted to deliver quick updates and new features, they couldn't do so easily. Too many unexpected problems would show up in production after deployment.

Lingvo Live was also implemented initially as only a local service. When the company wanted to increase their reach beyond Russia, they met several new problems. For example, when ABBYY began promoting Lingvo Live in South America, the team began seeing issues with the Internet traffic to Russia. To support workload spikes during launch events, the server infrastructure was overprovisioned out of necessity, wasting capacity most of the time. When a marketing campaign triggered an especially large spike in user activity, the engineers had to manually add more servers to make sure they could handle the increased load.

The expense of managing and maintaining the legacy code base in a traditional datacenter was simply too high. So the ABBYY team looked to the cloud.

## A new vision: Service Fabric and microservices

The team knew that modern technologies and a microservices architecture could ease many of their issues, but they couldn't afford to rewrite the Lingvo Live application all at once. They wanted to take an evolutionary approach: host the application as is, gradually improve it over time, refactor the old code piece by piece, and move it to the new platform.

The first step toward the goal was to decouple the web front end from the HTTP API, so development teams could work on them side by side. One team would use the latest front-end development technologies and frameworks, which would enable them to develop the UI much faster and set up a continuous delivery pipeline that could support zero downtime upgrades. The other team would begin splitting up the monolithic back-end logic into microservices, making sure the new services continued to work with the legacy code they weren't yet ready to migrate.

They considered running containers and orchestrators, but the choices at the time were available only on Linux. Another option was a pure IaaS approach, where they could set up the exact environment they needed. They just didn't want to take on the added responsibility for organizing deployment, health checking, and hosting.

The ideal scenario would include at least some PaaS components to free the team to concentrate on the application instead of the infrastructure. But the platform had to support low-level operations like configuring and installing the DCOM components used in the older code base. It had to provide enough elasticity to dynamically allocate and return resources. The wish list also included support for DevOps, upgrade management with zero downtime, and diagnostic and self-healing features that eased management.

Service Fabric fulfilled all the requirements.

## Service Fabric benefits

Service Fabric was a perfect fit for the ABBYY team's evolutionary approach. It provided the flexible, scalable compute resources they wanted along with an SDK for microservices and state management. Service Fabric also supports containers and guest executables, something the team found especially useful when migrating their existing services and components to the new environment.

To begin, the development team split up the web portion of the application into a front end and an HTTP API that could be implemented separately. Decoupling the user interface from the API meant they could immediately take advantage of the latest front-end development technologies and frameworks such as React with Redux. They rewrote the interface based on Node.js and hosted it as a stateless service within a Service Fabric cluster.

With the front end in place, ABBYY began modernizing their existing technologies by moving the code from their monolithic back-end systems to microservices. The goal was to create several microservices that can be updated independently, each with its own code base if needed. ABBYY can develop new features quickly and scale only those parts of the application that required more resources. Service Fabric was key to enabling this step-by-step approach to migration.

Service Fabric eased ABBYY's infrastructure issues as well. Service Fabric can host different types of applications, such as .NET, Java, and PHP, and host as many applications as needed. ABBY can use a single cluster for all company services, saving on costs and maintenance time. Service Fabric clusters can be created on any virtual machines or physical computers running Windows Server or Linux. ABBYY deploys and runs their Service Fabric applications on Azure and uses their on-premises infrastructure for development and debugging.

“ Service Fabric supports Windows containers and guest executables, which is very useful when migrating existing services and components. That flexibility was the key to our migration strategy. We wanted to go to a modern platform, but we needed to go there step by step.”

—Eugene Agafonov: Development Manager
ABBYY

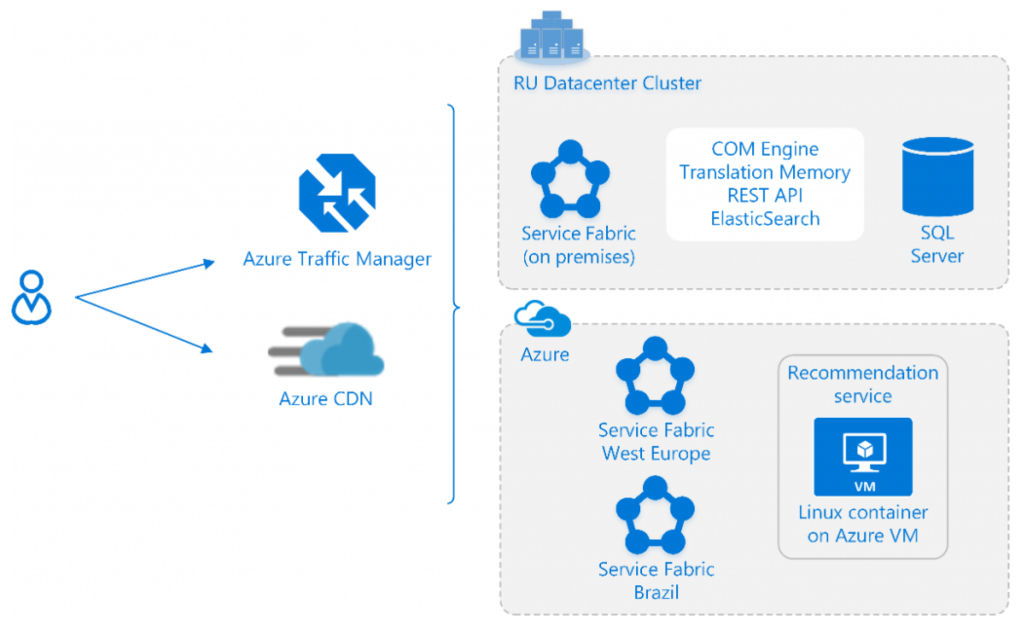


*Figure 2. ABBYY rearchitected the Lingvo Live social networking platform based on Service Fabric.*

The new architecture includes several Azure services:

- **Service Fabric** supports the composite nature of the Lingvo Live application. Service instances can be created across servers, virtual machines, and containers, and ABBYY doesn't need to standardize the tech that goes the components.
- Azure Traffic Manager (https://azure.microsoft.com/en-us/services/traffic-manager/) routes user requests to the nearest datacenter, Azure or on-premises, for optimal performance. Users no longer complain about the interminably slow response times.
- Azure Content Delivery Network (https://azure.microsoft.com/en-us/services/cdn/) (CDN) caches static web content at strategically placed locations so regional users receive speedy response times to queries.
- Azure virtual machines (https://azure.microsoft.com/en-us/services/virtual-machines/) are now used to host the Recommendation service, a new machine learning component that recommends words for users to study based on their search history. The service is written in Python and runs separately on Linux frameworks and dependencies.
- **SQL Server** stores state information associated with incoming user requests.

## Deployment automation

Service Fabric rolling upgrades are a key feature that help ABBYY juggle their old code base alongside the new. An application remains available throughout the rolling upgrade. The team learned how to use the

Service Fabric features to update their front end seamlessly with zero downtime. New versions of the front end can be launched side by side with the previous version, and users can use either version.

For more complicated upgrades involving components that run on-premises, they emulate a virtual IP (VIP) swap using Azure Traffic Manager. They create a new Service Fabric cluster, test the deployment, create a new endpoint in Traffic Manager, then turn off the old one.

## Next steps

With the improved Lingvo Live running in Azure, the team continues to rewrite the back-end components using Service Fabric Reliable Actors, an implementation of the Virtual Actor pattern. This computation model for distributed systems enables a large number of actors to execute simultaneously and independently of each other. By using this model, the ABBYY team hopes to gain even greater flexibility and scalability in their social networking service.

Figure 3 shows the next phase of the architecture that ABBYY is working on. The front-end node is part of a Service Fabric cluster that hosts the two front-end components, the website and the API gateway. The API gateway will route requests to corresponding actors. The state is stored in SQL Database, which was already in use at ABBYY. The plan is to replace the full-text search service with ElasticSearch, customized to use ABBYY's morphology libraries. ElasticSearch will run as a guest service, and actors will run in parallel on the same Service Fabric node.

Containers are also planned for host-agnostic components such as the Recommendation service that currently runs on Linux. This service, a Python script and some libraries, doesn't need to know how it will be hosted, so containers are the perfect model. The developers can package the script, data, and libraries, and the Service Fabric runtime can orchestrate and host it.

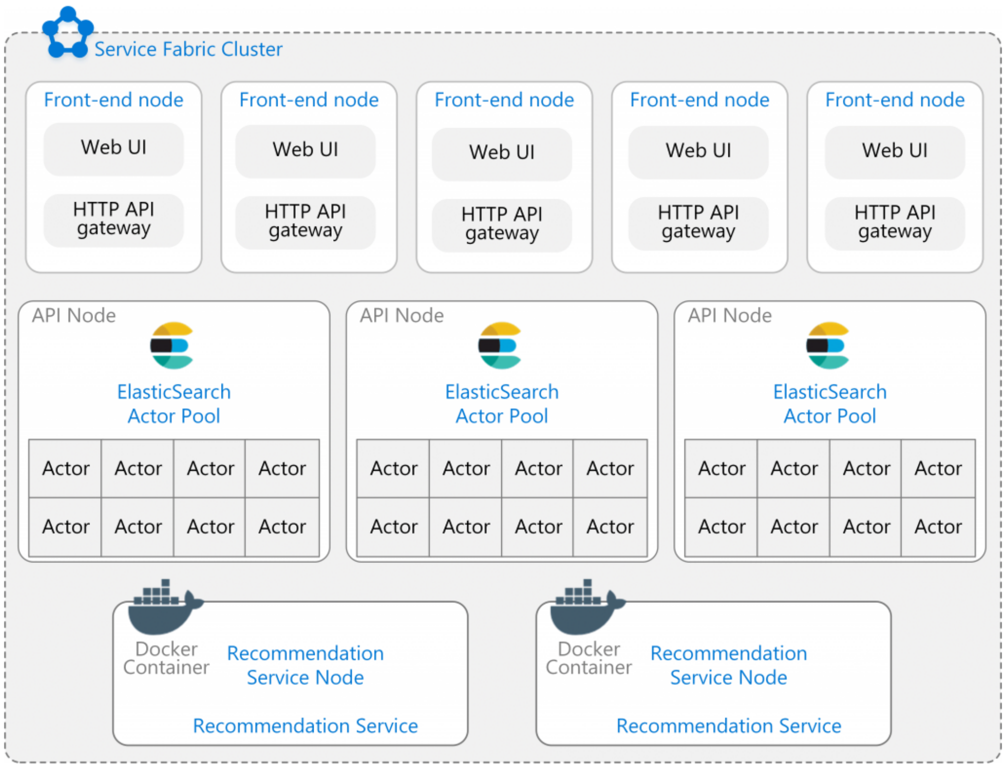


*Figure 3. Proposed use of actors and containers in Lingvo Live.*

## Summary

ABBYY continues to translate its social networking service into a modern, microservices-based platform that runs on Azure. The new architecture lets ABBYY take an evolutionary approach to product improvements. The product stays in production and continues to serve users while older components can be gradually replaced behind the scenes.

As they implement their architectural wish list, Service Fabric provides a reliable platform for orchestrating services, and Azure supplies the flexible, cost-effective infrastructure they need. With a service-driven platform managed by Service Fabric, ABBYY is creating the social dictionary of the future.