Designing a modern service architecture for the cloud

The digital transformation that many enterprises are undertaking has its benefits and its challenges: while it brings new opportunities that add value to customers and help drive business, it also places demands on legacy infrastructure, making companies struggle to keep pace with the digital world’s ever-increasing speed of business. Consider an enterprise’s line-of-business (LOB) systems, such as for finance in general, or procurement and payment in particular. These business-critical systems are traditionally based on-premises, can’t scale readily, and in many cases aren’t available to mobile devices.

As we continue along our digital transformation journey, Microsoft has been looking to the cloud to reinvent how we do business, by streamlining our operations and adding value to our partners and customers. This technical case study describes how the Microsoft Core Services Engineering and Operations (CSEO) team saw our move to the cloud as an opportunity to completely rethink how we architect and run our core finance processes when they’re built on a modern architecture. Here, we discuss the thought processes and drivers behind the approach that CSEO took to design a new service architecture for our Finance department’s Procure-to-Pay service.

Evolving from an apps-oriented to a services-focused architecture

Financial systems need to be secure by their nature. Moreover, their designs are typically influenced by an organizational culture that is understandably risk averse, so the concept of moving sensitive financial processes to the cloud can be especially challenging. The technical challenges are equally significant: Consider the transactional nature of financial systems, their real-time transactional data processing, auditing frequency and scale, and the numerous regulatory aspects that are associated with financial operations.

At Microsoft, many of our core business processes (such as procurement and payment) have traditionally been built around numerous monolithic, standalone apps. Each of these apps was siloed in its own on-premises environment, used its own copy of data, and presented one or more interfaces, often disconnected from each other. Without a unifying, overarching strategy, each of these apps evolved independently on an ad hoc basis, updating as circumstances required without considering impacts on other parts of the Procure-to-Pay process.

These complex and unwieldy apps required significant resources to maintain, and their redundant data led to inconsistent key performance indicators (KPIs) that were based on different underlying data sets. Furthermore, the user experience suffered because there wasn’t a single end-to-end process for Procure-to-Pay. Instead, people had to work within several different apps—each with its own interface—to complete a task, forcing users to learn to navigate through many different user experiences as they attempted to complete each step. The overall process was made even more cumbersome because people still had to complete manual steps in between certain apps. This in turn slowed completion of every Procure-to-Pay instance and was expensive to maintain.

CSEO’s ongoing efforts to shift services to the cloud gave the Microsoft Finance Engineering team within CSEO an opportunity to completely rethink how to approach Procure-to-Pay by designing a cloud-based, services-oriented architecture for the Finance department’s procurement and payment processes. This, modern cloud-based service, known as Procure-to-Pay, would focus on the end-to-end user experience and would replace the app-centric view of the legacy on-premises systems. Additionally, the cloud-based service would utilize Azure’s inherent efficiencies to reduce capital expenditure costs, scale dynamically, and promote referencing of certified master data instead of copying data sets as the legacy apps did.

In this part of the case study, we describe some key principles that we followed when designing our new service-based architecture, and then provide more insight into the architecture’s data, API, and UI.

Principles of a service-based architecture

We started this initiative by defining the key principles that would guide our approach to this new architectural design. These principles included:
Focus on the end-to-end experience by creating an overarching user experience (UX) layer that developers can use to connect different services and present a unified user experience.

Design as cloud first, mobile first to gain the cost and scalability benefits associated with cloud-based services, and to improve end user productivity.

Maintain single master copies of data with designated data owners to ensure quality while reducing redundancy.

Develop with efficiency and cost-effectiveness at every step to reduce Microsoft Azure-based compute time costs.

Decouple UI from business functionality by defining separate layers for UI, business functionality, and data storage within each service.

Utilize flighting with early adopters and other participants to reduce change-management risk.

Automate as much as possible, identifying the manual steps that users had to take when working with the old on-premises apps and determining how to automate them as part of the new end-to-end user experience.

In the next few sections, we provide additional insights into how we applied these principles from UI, data, and API perspectives as we designed our new architectural model and used it to build our Procure-to-Pay service.

Emphasizing a holistic, end-to-end user experience

When we surveyed our legacy set of on-premises apps, we discovered a significant overlap of functionality between them. Our approach with the new architectural model was to break down the complete feature set within each app to separate core functionality from duplicated features.

We used this information to consolidate the 36 standalone legacy apps into an architecture that comprises 16 discrete services, each with a unique set of functionality, presentation layer, APIs, and master data. On top of these 16 unique services, we defined an overarching End-to-End User Experience layer that developers can use to create a singular, unified experience that can span numerous services.

“Building meaningful experiences for people is what we do now. To truly empower people and to deliver value, we had to move away from focusing on experiences mapped to capabilities.”

—Sahil Garg, Principal Software Engineering Manager, Finance Procure-to-Pay, Microsoft

As Figure 1 illustrates, our modern architecture utilizes a modular approach to services that promotes interconnectivity. Because users interact with the services at the End-to-End User Experience layer, they experience a consistent and unified sequence of events in a single interface. Behind the scenes, developers can connect APIs in one service to another to access the functionality they require, or transparently pass the user from one service’s UI to another as needed to complete the Procure-to-Pay process.
Figure 1. Our modern architecture model defines 16 unique vertical services, each with its associated UI, API, and data layers. It also provides an overarching user experience layer that developers can use to connect any combination of the services into a seamless, end-to-end user experience.

Another critical aspect of providing an end-to-end experience is automating the front- and back-office operations (such as support) as much as possible. To support this automation, our architecture incorporates a Procure-to-Pay Support layer underneath all the services. Developers can integrate support bots into their Procure-to-Pay services to monitor user activity and proactively offer guidance when deemed appropriate. Moreover, if the support bot can’t quickly resolve the issue, it will silently escalate to a human supervisor who can interact with the user within the same support window. Our objective is to make the support experience so seamless that users don’t recognize when they are interacting with a bot vs. a support engineer.

All these connections and data referencing are hidden from the user, resulting in a seamless experience that can be expressed as a portal, a mobile app, or even as a bot.

Consolidating data to support end-to-end experiences

One ongoing challenge that we experienced with our siloed on-premises apps was how each app utilized its own copy of data, resulting in wasted storage space and inconsistent analytics due to the variances between data sets. In contrast, the new architectural data model had to align with our principle of maintaining single, master copies of data that any service could reference. This required forming a new Finance data lake to store all the data.

The decision to create a data lake required a completely new mindset. We decided to shift away from the traditional approach, in which we needed to understand the nature of each data element and how it would be implemented in a solution. Today, our strategy is to place all data into a single repository where it can be available for any potential use—even when the data has no current apparent utility. This approach recognizes the inherent value of data without having to map each data piece to an individual customer’s requirements. Moreover, having a large pool of readily available, certified data was precisely what we needed to support our machine learning (ML) and AI-based discovery and experimentation—processes that require large amounts of quality data that had been unavailable in the old siloed systems.
“Data lakes and the elastic cloud are helping us scale and innovate, generating data insights and analytics at a much faster pace than ever before. Improving compliance by re-examining processes and reducing the number of data copies are excellent side benefits.”

— Vivek Chauhan, Principal Software Engineering Lead, Finance Procure-to-Pay, Microsoft

After we formed the Finance data lake, we defined a layer in our architecture to support different types of data access:

- **Hot access** is provided through the API layer (described later in this case study) for transactional and other situations that require near real-time access to data.
- **Cold/warm access** is used for archival data that is one hour old or older, such as for machine learning or running analytics reports. This is a hybrid model, where we can access data that is as close to live status as possible without accessing the transaction table, but also perform analytics on top of the most recent cold data.

By offering these different types of access, our new architectural model streamlines how people can connect data sources from different places and for different use scenarios.

**Designing enterprise services in an API economy**

In the older on-premises apps, the tight coupling of UI and functionality forced users to go through each app’s UI just to access the data. This type of design provided a very poor and disjointed user experience because people had to navigate many different tools with different interfaces to complete their Procure-to-Pay task.

One of the most significant changes that we made to business functionality in our new architectural model was to completely decouple business functionality from UI. As Figure 1 illustrates, our new architectural model has clearly defined layers that place all business functionality in a service’s API layer. This core functionality is further broken down into very small services that perform specific and unique functions; we call these **microservices**.

With this approach, any microservice within one service can be called by other services as required. For example, a link-validation microservice can be used to verify employee, partner, or supplier banking details. We also recognized the importance of making these microservices easily discoverable, so we took an open-source approach and published details on Swagger about each microservice. Internal developers can search for internal APIs for reuse, and external developers can search for public APIs.

“Modernizing our service architecture has empowered our engineers to achieve digital transformation at an industry pace. They’re now able to innovate, iterate, and leverage APIs across the service line with reduced risk and technical debt.”

— James Gagnon, Principal Software Engineering Lead, Finance Procure-to-Pay, Microsoft

As an example, Figure 2 illustrates the usage scenario for buying a laptop, where the requester works through the unified User Experience layer. What is hidden to the user is how multiple services including Catalog Management, Purchase Experience, and Purchase Order interact as needed to pass data and hand off the user transparently from service to service to complete the Procure-to-Pay task.
Figure 2. An example usage scenario for buying a laptop, illustrating how the person requesting a new computer works through the unified End-to-End User Experience layer while multiple services work transparently in the background to complete the end-to-end Procure-to-Pay task

When defining our modern architecture, we wanted to minimize the risk that an update to microservice code might impact end-to-end service functionality. To achieve this, we defined service contracts that map to each API, and how the data interfaces with that API. In other words, all business functionality within the service must conform to the contract’s terms. This allows developers to stub a service with representative behaviors and payloads that other teams can consume while the service code is being updated. Provided the updates are compliant with the contract, the changes to the code won’t break the service.

Finally, our new cloud-based modern architecture gave us an opportunity to improve the user experience by specifying a single sign-on (SSO) event throughout the day, irrespective of how many services a user touches during that time. The key to supporting SSO was to leverage the authentication and authorization processes and protocols that are built into Azure Active Directory.

Benefits

Following are some of the key benefits that CSEO is experiencing by building our Procure-to-Pay service on our modern cloud-based architecture.

- **Vastly improved user experience.** The new Procure-to-Pay service has streamlined the procurement and payment process, providing a single, end-to-end user experience with a single sign-on event that replaces 36 legacy apps and automates many steps that used to require manual input. In internal surveys, employees are reporting a significant improvement in satisfaction scores across the enterprise: users are happier working with the new service, engineers can more easily troubleshoot issues, and feature updates can be implemented in days instead of months.

- **Better compliance.** We now have full governance over how our data is being accessed and distributed. The shift to a single Finance data lake with single copies of certified master data and clear ownership of that data, ensures that all processes are accessing the highest-quality data—and that the people accessing that data are authorized to do so.
Better insights. Now that our KPIs are all based on the certified master data, we’ve improved our analytics accuracy by ensuring that all analysis is based on the same master data sets. This in turn enables us to ask the big questions of our collective data, to gain insights and help the business make appropriate data-driven decisions.

On-demand scaling. The natural rhythm of Finance operations imposes high demand during quarterly and annual report periods, while requiring fewer resources at other times. Because our architecture is based in the cloud, we utilize Azure’s native ability to dynamically scale up to support peaks in processing and throttle processing resources when demand is low.

Significant cost and resource savings. Building our new Procure-to-Pay service on a modern, cloud-based architecture is resulting in cost and resource savings through the following mechanisms:

- Decommissioned physical on-premises servers: We’ve decommissioned the expensive, high-end physical and virtual servers that used to run the 36 on-premises apps and replaced them with our cloud-based Procure-to-Pay service. This has reduced our on-premises virtual machine footprint by 80 percent.
- Reduced code maintenance costs: In addition to decommissioning the on-premises apps’ servers, we no longer need to spend significant development time maintaining all the brittle custom code in the old siloed apps.
- Drastic reduction of compute charges: Our cloud-based Procure-to-Pay service has several UIs that can be parked and stored very cost effectively as BLOBs until the UIs are needed. This completely avoids any compute-based charges until a UI is required and is then launched on demand.
- Reduction in support demand: Our bot-driven self-serve model automatically resolves many of our users’ basic support issues, freeing up our support engineers to focus on more critical issues. We estimate a 20 percent reduction in run cost by decommissioning our Level 3 support line, and a 40 percent reduction in overall Procure-to-Pay related support tickets.
- Better utilization of computing resources: Our old on-premises apps incurred huge capital expenditure costs when purchasing their high-end hardware and licenses for servers such as Microsoft SQL Server. With a planning and implementation period that might take months, machines were typically overbuilt and underutilized because we would plan for an approximate 10 times capacity to account for growth. Later, the excess capacity wouldn’t be sufficient, and we would have to repeat this process to purchase newer hardware with even greater capacity. The new architecture has eliminated capital expenditures for Procure-to-Pay, favoring the more efficient, scalable, and cost-effective Microsoft Azure cloud environment. We’re also utilizing our data storage more efficiently. It is less costly to store data in the cloud, and storing a single master copy of data in our Finance data lake removes all the separate copies of the same data that each legacy app would maintain.
- Better allocation of personnel: Previously, our Engineering team had to review the back-end systems and build queries to cater to each team’s needs. Consolidating all data to the Finance data lake in our new system enables people to create their own Microsoft Power BI reports on top of the data, modify their analyses to form new questions, and derive insights that might not have appeared otherwise. As a result, our engineering resources can be reallocated to support more strategic functions.

Simplified testing and maintenance. We use Azure’s out-of-the-box synthetics to test each function within our microservices programmatically, which is a much easier and more comprehensive approach than physically testing each monolithic app in a reactive state to assess its health. Similarly, Azure’s service clusters greatly streamline our maintenance efforts, because we can deploy many instances of different services to achieve a higher density. Moreover, we now utilize a single cluster for all our preproduction environments. We no longer need to maintain separate development, system test, staging, and production environments.

Lessons learned and best practices

CSEO learned some valuable best practices as we designed our modern cloud-based architecture:

- Achieving a modern architecture starts with asking the big questions: Making the shift from large, unwieldy standalone on-premises apps to a modern, cloud-based services architecture requires some up-front planning. Assemble the appropriate group of stakeholders and gain consensus on the following questions: What type of architecture do we want? Where do we want to have global access to resources? What types of data should be stored locally, and under what circumstances? When and how do we programmatically access data that we don’t own to mitigate, minimize, or entirely remove data duplication? How can we ensure what we’re building is the most efficient and cost-effective solution?
- Identify where your on-premises apps are in their lifecycle when deciding whether to “lift-and-shift”: If you’re dealing with an app or service that is nearing its sunset phase and you only need to place it into the cloud for a short period while you transition to something newer, consider the “lift-and-shift” approach where your primary objective is to run the exact
same system in the cloud. For systems that are expected to have a longer lifecycle, you’ll reap greater rewards by rethinking your service architecture with a platform as a service (PaaS) mindset from the start.

- **Design your architecture for engineering rigor and agility.** Look for longer-term value based on strategic planning to make the most of your transition to the cloud. At Microsoft, this was the key determination that guided our new architecture’s development: *Reimagine how our core processes can be run when they’re built on a modern service architecture.* For us, this included being mobile first and cloud first, and shifting from waterfall designs to adopting agile practices. It also entailed making security a first thought in architectural design instead of a last thought, and designing the continuous integration/continuous deployment (CI/CD) pipeline.

- **Keep cost efficiency in mind.** From the very first line of code, everyone involved in developing your new services should strive to make each component as efficient and cost effective as possible. At Microsoft, this development principle is why we mandated a serverless compute model with no static environments that supported “parking” inactive code or UI inside BLOBs when they weren’t needed. This efficiency is also a key reasoning behind our adopting Azure resource groups to minimize the effort required to switch between stage and production environments.

- **Put everything into your data lake.** Cloud-based storage is inexpensive. When organizations look to the cloud as their primary storage solution, they no longer need to expend effort collecting only the data that they think everyone wants—especially because, in reality, everyone wants something different. At Microsoft, by creating the Finance data lake and shifting our mindset to store all master data there, irrespective of its anticipated use, we eliminated the resources we would traditionally spend to analyze each team’s data requirements. Today, we focus on identifying data owners and certifying the data. We can then address the data of interest when a customer makes a specific request.

- **Incorporate telemetry into your architecture to derive better insights from your data.** Your data-driven decisions are only as good as your data. In our old procurement and payment system at Microsoft, we didn’t know who was using the old data and for what reasons, or even how much it was costing us. With the new Procure-to-Pay service based on our modern architecture, we have telemetry capabilities inside everything we build. This helps with service health monitoring. We also incorporate this information into our feature and service decision-making processes as we continually improve Procure-to-Pay.

- **Promote your new architectural model to gain adoption.** You can define a new architectural design, but if you don’t promote it in a way that demonstrates its value, developers will hesitate to use it. At Microsoft, we published details about how developers could tap into this new architecture to create more intuitive and user-friendly end-to-end experiences that catered to their end users. This internal open-source approach creates a collaborative environment that encourages developers to join in and access the data they need, and then apply it to their own end-to-end user experience wrapper.

### Conclusions

At Microsoft, rethinking our approach to services with this cloud-based modern architecture is helping us become a data-driven organization. By consolidating our data into a single data lake and providing an API layer that enables rapid development of end-to-end procurement and payment services, we’ve created a self-serve platform where anyone can consume the certified data and present it in a seamless, end-to-end manner to the user, who can then derive insights and make data-driven decisions.

### Our next steps

The Procure-to-Pay service is just one cloud-based service that we built on top of our modern architecture. We’re continuing to mature this service, but we’re also exploring additional end-to-end services that can benefit other Finance processes to the same extent that Procure-to-Pay has modernized procurement and payment.

This new model doesn’t have to be restricted to Finance; our approach has the potential to benefit the entire company. The guiding principles we followed to define our Finance architecture align closely with our leadership’s digital transformation vision. That is why we’re also discussing how we might help other departments outside Finance adopt the same architectural model, build their own end-to-end user experiences, and reap similar rewards.

### For more information

**Microsoft IT Showcase**

[microsoft.com/itsshowcase](microsoft.com/itsshowcase)

*DevOps is sending engineering practices up in smoke*

[microsoft.com/itsshowcase](microsoft.com/itsshowcase)  November 2018
Meet the engineers: Building line-of-business apps at Microsoft with CI/CD

Designing a modern data catalog at Microsoft to enable business insights

Azure architecture guidance

© 2018 Microsoft Corporation. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.