Migrating the products content management system to Microsoft Azure

For more than a decade, Microsoft has been using an internally developed content management application to create and store product content and publish it to Microsoft Support, Premier Contract Support, and other internal and partner channels. Built internally and managed by the Microsoft Support Engineering group, the Microsoft products content management application is the single “golden” source of all information for each product Microsoft builds and sells.

The products content management application was built long before engineering teams across Microsoft began to develop modern app experiences in Microsoft Azure. The application relied on a physical infrastructure that became increasingly difficult to manage and to protect against modern threats. Despite these limitations, the products content management application supports a critical business function and needs to remain operational for all content channels and apps that pull product information from it.

To make the business-critical application more manageable, and keep it secure and compliant in the face of increasingly sophisticated threat vectors, we decided to migrate the application and its supporting infrastructure to Azure.

Additionally, migrating to cloud services in Azure provided immediate benefits in the following areas:

- Scalability
- Security
- Performance
- Reliability

The business challenges of maintaining legacy apps in the environment

In the past, one trend at Microsoft was to build standalone internal applications to automate and facilitate individual business processes and workflows. This practice resulted in a great number of applications, supporting important business functions, built to work on the technology and infrastructure that was available at the time they were created. It also produced many instances where applications built by one group were similar in function to solutions built by other business groups.

Microsoft has become a cloud-first organization. Now engineering teams build new application experiences to take advantage of the power of the Azure service cloud. Engineering groups across the company look toward longer-term goals of moving their individual apps to centralized cloud-based solutions. As this transition takes place, numerous aging applications built on physical infrastructures still need to be moved off noncompliant hardware without disrupting the business.

Adding physical infrastructure introduces complexity

Like almost any internal application built on-premises and remaining in operation for many years, the products content management system’s architecture had grown quite complex. As the service’s capacity grew, we added physical infrastructure. As business processes evolved, we added on features and functionality. As illustrated in Figure 1, the products content management system grew to include 44 physical servers and spanned multiple domains.
Managing aging applications is time-consuming

Because we added and replaced resources at different times during the lifecycle of the content management application, there was little consistency. The servers varied in age and their configurations differed, making it difficult and time consuming for us to manage them.

Adding to the challenge of managing our complex infrastructure, we had no documentation or code library to refer to. We understood how the application worked, but we lacked key details about what was built, when it was built, and in what order things were added. Whenever a problem occurred, we spent a great deal of time and effort investigating and fixing the issue. Anything that needed a code fix or change required backward engineering. Another area in which the application was lacking was business continuity. Any type of update to any part of the infrastructure resulted in downtime. Also, the application was designed with very little failover to keep it operational in the event of a natural disaster or widespread outage.

Facing modern threats

In June 2017, the Spectre threat emerged. Spectre is a sophisticated attack designed to infiltrate hardware vulnerabilities. Microsoft and its OEM partners worked diligently in the face of that threat to release security patches to protect client and server devices. However, many servers within our application’s infrastructure were simply too old to be patched. We were facing a compliance deadline and needed to make a choice. We needed to either replace all the noncompliant hardware or migrate the application to cloud services in Azure.

Deciding to migrate to Azure

We needed to make a few key improvements, but we didn’t want to build an entirely new cloud application while we’re a little more than a year away from moving to a centralized content management system. We also didn’t want to disrupt the business or the content channels that pull products data from our system. We decided to do a “lift and shift” migration, meaning that we moved the existing system’s infrastructure and functionality into Azure.
Our migration project took five months to complete. A solution engineer worked with a couple of developers to reverse-engineer the solution code, and then we began moving the solution’s physical infrastructure to the cloud resources.

Migration to Azure addressed our pain points. Just as importantly, migration to Azure provided a near-seamless transition experience for end users and the channels that depend on the content managed by the system. It also offered us an opportunity to consolidate some of the infrastructure, streamline parts of the service, improve performance, and provide business continuity through high-availability services, such as Azure Virtual Machine (VM) scale sets and Azure Traffic Manager.

Planning the migration

Because we had no existing documentation or code library to reference, our plan included spending time identifying and documenting the existing application. We needed full documentation to help us design a simplified architecture and to better prepare us for future modernization efforts.

We also needed to reverse-engineer the code for the Authoring UI and the Build processor to include it in a new code library in Microsoft Visual Studio. When we were done planning the migration, creating comprehensive documentation, and building a code library, we began designing the simplified architecture.

Simplifying the architecture

To simplify the architecture, we decided to consolidate the primary infrastructure into two region-specific domains. We used the primary regions in which our content managers reside. We also reduced the number of development environments from four (integration, user-acceptance testing, staging, and production) to two (preproduction and production). We were able to reduce the total number of servers from 44 to 20—a 54 percent reduction. Figure 2 depicts the current cloud infrastructure after migrating all of the physical assets to cloud services in Azure.

![Diagram of cloud infrastructure](image)

*Figure 2. The products content management system after migrating the infrastructure to cloud services in Azure.*

We migrated the following functional components in the content management system to Azure:

- **Authoring UI.** The servers and web services that host the UI in which content managers author their content.
• **Database.** The on-premises database was moved to Microsoft Azure SQL running on Azure VMs. Previously, we had only one physical server running SQL Server. By migrating to Azure we were able to add three VMs in different regions to enable disaster recovery.

• **Build processors and file servers.** Build processors are servers that host the logic to process and move content from its source (the content management system) to its destinations (the support and partner sites). We reduced the number of build processor servers from ten to five. The file servers parse the XML files from the source to the destination through the PubWiz web service. Because of the improvements we saw when moving to the latest VM server configurations and the simplified architecture, we were able to reduce the number of file servers from eight to four.

**Planning the capacity**

As we planned for baseline capacity, we were primarily concerned with right-sizing our solution, with a focus on cost savings. We didn’t want to provision more resources than we needed to run under a normal resource load and then have them sit idle.

Azure’s scalability inherently improves how we plan for surge capacity. Before the migration to Azure, we spent several months each year preparing for the surge in growth during the holiday season and estimating resource requirements for it. Then we’d spend months offloading those resources. Now we can spin up new resources within a few hours and take them offline as soon as we no longer need them.

**Provisioning Azure resources**

We started with standard Microsoft CSEO [Azure Resource Manager](https://www.microsoft.com/en-us/azure/resource-manager) (ARM) templates created for provisioning. We customized and configured them in PowerShell for all our server types (SQL servers, web servers, and the file server).

Using Microsoft ARM templates reduced the time we spent adding new servers to the system infrastructure from months to a few hours. With ARM templates, we can also create multiple instances of a resource, which we can use to quickly ramp up different regions using identical resources. The only difference in the process to add a region would be to change region-specific code in the Visual Studio code library.

**Configuring the VM servers**

We built out the system’s infrastructure in the latest Windows Server and SQL versions—replacing all the physical servers with Azure IaaS 2016. We created the website, installed software configured services, and installed the code executables on all the servers.

For SQL, we configured each server to run four VM instances of SQL AlwaysOn. AlwaysOn helps protect the SQL Server back end of an application using a combination of SQL Server business continuity and disaster-recovery (BCDR) technologies, including high-availability and Failover Clustering.

**Deploying the code**

To deploy the code on the VM servers, we simply migrated the code files from Visual Studio to the newly provisioned servers and set up the web services. Code deployment can be automated through PowerShell to save time. However, because we didn’t have source code to include in the PowerShell automation script, we moved the code from the existing production servers onto the VMs and didn’t use automation for this step.

**Testing the servers and code**

We performed functional testing of all the migrated servers and the supporting code and services, including Windows Services, Azure VM, and Cloud Services. We validated the target-build processor, publishing service, and code compatibility with the latest versions of the operating system, .NET Framework, and Azure SQL.
We did see some code-compatibility issues because the original code was an earlier version of .NET Framework than the current server that SKUs use. We rewrote the code to be compatible with Microsoft .NET 4.0 and later. We also verified that service-account permissions were set up properly to ensure that we wouldn’t break content connections for partners and systems that connect to product content.

Going live

To switch from the physical servers to the Azure VMs, we needed to plan for downtime and content replication. We planned to perform the DNS cutover on a Friday during nonpeak business hours. This timing would ensure we had quick access to support resources. We communicated to content managers and dependent platform and service owners that the system would be in read-only mode for one hour so we could perform content replication and DNS switching. We informed external partners there would be a change in the connection string for the UI and the database.

After the migration went live, we performed an operational validation of all the system’s functionality.

Best practices and lessons learned

As we migrated the products content management system to Azure, we faced a few challenges and learned some things that might help other organizations thinking about migrating some of their on-premises apps and solutions to cloud services in Azure.

- **Understand and document your services.** Creating documentation and reverse engineering the app (because there was no code library) required a lot of time and effort during the planning phase of our migration. Beyond the migration, we needed that documentation to ensure that we had a clear understanding of our partners’ dependencies on the system so we could coordinate our changes with them. We recommend ensuring you have comprehensive documentation for your existing business applications, particularly any that you prioritized for modernization.

- ** Reserve your cloud resources in advance.** After you assess the environment, reserve your Azure resources in advance to get prioritized compute capacity. Also, you should reserve the required number of IP address blocks to ensure availability.

- **Make your code compatible with the latest infrastructure.** We encountered some compatibility issues between the app code and the latest infrastructure versions we used in Azure. The existing code in the app was .NET 2.0, while the current infrastructure standard is .NET 4.0 or 4.5. We could have downloaded .NET 2.0 to the current server types we used, but the better choice was to make the code compatible with the current version of .NET Framework. Taking time to ensure the code is compatible with the current framework can help prevent future compatibility issues between the code and product updates and patches.

- **Consolidate domains.** The original content management system spanned several domains. By consolidating the solution to two domains that can communicate directly, we were able to clear up complex trust issues that required the use of intermediate communications servers.

- **Use ARM templates and store them for reuse.** We saved time by starting with standard ARM templates, and customizing them with our own configurations and parameters. We stored our customized ARM templates for each server type in our Visual Studio code library.

- **Use Azure portal to create resource groups.** In Visual Studio, you can easily create and deploy Azure resource groups as a project. No ad-hoc resources are created when you deploy Azure resource groups, which can help reduce costs.

Improving management and agility

Migrating to Azure vastly improved our business agility in terms of supporting the products content management system, and it made it easier for us to manage risk. The system is more secure and regularly updated. Also, we can...
better manage security and secrets using certificate management in Azure Key Vault and the Managed Service Identity (MSI) service in Azure Active Directory (AD) for managed services.

Moving away from a physical infrastructure improved performance and increased reliability. Using Azure Traffic Manager makes it easier to load balance, because updates no longer require downtime.

ARM templates provide ecosystem consistency, and we now have an exceptionally uniform deployment experience. This additional consistency makes it easy and fast to scale resources up or down. We can now scale out to new geographic areas in ten minutes rather than in the three-month span the process used to involve.

Conclusion

Despite the age and complexity of the products content management system, we were able to successfully migrate it to Azure. Now the infrastructure is more manageable, and its application is more serviceable. We were able to provide a near-seamless transition for content managers and our partners that connect to the content.

As we assessed the products content management environment, we simplified the infrastructure and created the missing documentation set and code base. The documentation and code base will be key as we continue to sustain the migrated content management system that’s in use today. They’ll also be useful as we start looking at our longer-term goals, which include the eventual move to a modern, centralized solution that will manage content across all of Microsoft.

Are you ready to take the next step? Migrate your own applications to Azure by engaging with FastTrack, or start on your own with the migration center.

For more information

Microsoft IT Showcase

microsoft.com/itsshowcase

Azure Migration Center

Azure Resource Management Templates

© 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.