Microsoft Core Services Engineering and Operations (CSEO) recently partnered with Microsoft Finance to migrate a mission-critical financial system named Mercury to Microsoft Azure. CSEO migrated an on-premises infrastructure of more than 150 servers in a staged process to a cloud-native configuration hosted in Azure virtual machines (VMs). The new, cloud-based Mercury solution in Azure enabled Microsoft Finance to avoid significant capital expenditure, gain greater visibility into financial activity, and create a more agile, resilient, and efficient solution in the cloud.

Examining financial reporting at Microsoft

Microsoft Finance provides management of and support for all Microsoft financial systems. Microsoft Finance uses a profit and loss reporting platform to connect and aggregate financial data across the company. Internally referred to as Mercury, this platform is the official book of record at Microsoft. It’s used by thousands of Microsoft employees to gather financial data and drive data-driven processes across Microsoft departments, including Sarbanes-Oxley (SOX) compliance, financial accuracy requirements, and official earnings reporting.

Mercury is a massive data warehouse solution that manages terabytes of transactional data such as expenses, revenue, hierarchy, budget, and forecast data. It compiles the data from 10 different financial data sources at Microsoft. Over 5,000 users leverage Mercury for financial functions and decision making, and it sends data to more than 50 downstream services and applications. Mercury is a highly available application with more than 99.99% uptime.

Hosting Mercury in the datacenter

Previously, Mercury was hosted in on-premises datacenters on physical servers running Windows Server. In the data center, the Mercury footprint spanned more than 150 physical servers across test and production environments. Microsoft SQL Server provided the base for Mercury functionality, including SQL Server Integration Services. The platform environment also included high-performance flash memory cards, storage area networks (SANs), and
network load balancers to support SQL Server cluster configurations. These clusters provided high availability and increased performance in the on-premises version.

In the datacenter, Mercury required high-performance hardware to manage millions of ongoing transactions. The extraction, transformation, and loading (ETL) nature of Mercury transactions created a large amount of sequential data processing, which was managed by high-performance server hardware with high performance memory cards connected to SAN storage. This hardware combination created an adequate environment in which the platform could operate, but several different factors led to the planned migration to Azure.

Evaluating cloud migration

We evaluated migration to Azure based on several factors, including the state and condition of the on-premises environment, potential improvements presented by the Azure environment, and broader enterprise considerations that affected the migration. The most important factors that led us to evaluate migration to Azure for Mercury included:

- Support for ongoing digital transformation. Azure is now the default platform upon which we build our IT infrastructure. Several years ago, CSEO created a vision for moving from on-premises data centers to Azure, as the "first and best customer" of our cloud services. We’ve moved more than 93 percent of our on-premises infrastructure to the cloud, and we’re assessing our strategic initiatives around our cloud efforts. Moving Mercury to Azure was one piece of a broader digital transformation happening in CSEO.
- Hardware requirements for the on-premises environment. The high-performance hardware required to run Mercury meant significant capital investment. We were running the platform on high-performance computing platforms that were provisioned to meet the considerable demands of the solution. Several components of the solution—including several servers—were either scheduled for a hardware refresh or already weren’t meeting performance standards. This refresh would require purchasing new server hardware and significant capital investment in the data center-based environment from which Microsoft is moving away.
- Lack of visibility into solution costs. In the on-premises solution, we didn’t have visibility into solution costs and effectiveness. This made it challenging to ascertain expenses and appropriate chargebacks to business groups, or understand the costs for other aspects of Mercury.

Migrating high-performance data processing to Azure

Based on our migration goals and pain points from our on-premises implementation, we established guiding principles that we used to lead us through the migration process:

- Build automation into the solution wherever possible. To make the most of the Azure environment’s agile and dynamic nature, we set out to build automation into all aspects of our solution, including infrastructure deployment, management, and monitoring. As such, we established the following performance goals for Mercury in Azure:
  - The infrastructure provisioning and system validation process must take less than 15 hours.
  - The post-deployment validation and cutover process must take less than 10 hours.
  - A live switchover from on-premises systems to Azure cloud systems must take less than 10 seconds.
  - Infrastructure VM scaling must occur within less than 10 minutes of a demand surge.
- Provide granular component selection. We wanted solution components to function interchangeably, thereby allowing us to add, remove, or modify individual VM, application, and network components without negatively impacting the rest of the environment.

Migration planning and implementation

We planned a lift-and-shift approach to migration, which meant taking components from the datacenter and migrating them directly to Azure without having to refactor or change the functionality or relationship of solution components. We intentionally chose to migrate Mercury to the infrastructure as a service (IaaS) model in Azure, which
is typically used in lift-and-shift scenarios. Using the IaaS model and the lift-and-shift approach helped us migrate Mercury as quickly as possible and minimize user impact and potential setbacks from re-engineering or required modifications. It also enabled us to run on-premises and Azure versions of Mercury in parallel, which made our testing and fallback processes more robust. We performed the migration process in three stages:

1. Evaluation of on-premises environment and performance. In the first stage, we established a clear understanding of the on-premises environment performance demands. We created key metrics to use for performance evaluation, and gathered data based on those metrics from our on-premises environment. The key metric benchmarks included:
   - Average CPU utilization for 64 cores should be at least 50 percent.
   - Physical memory size for a server is at least 2 TB.
   - Disk I/O operations per second (IOPS) required is at a minimum, 80,000.
   - Network speed is at least 2 GB/sec.

2. Azure infrastructure planning. Using the on-premises performance data, we began assessing Azure resources that would provide adequate functionality and performance for the solution components. The corresponding components included:
   - Azure IaaS VMs. Physical Windows Servers would be migrated to Azure VMs running Windows Server.
   - Azure ExpressRoute. On-premises networking connectivity to both upstream providers and downstream consumers would be facilitated by ExpressRoute connections from Azure to the Microsoft corporate network.
   - Azure managed storage. The hardware SAN environment in our on-premises solution would be replaced with Azure-managed storage.

3. Azure infrastructure testing and implementation. The testing and implementation stage were the most intensive and time-consuming stage of the migration process. Within this stage, we made decisions about which specific Azure resource SKUs we would use, and then tested the implementation of those resources in preparation for live migration. The components of infrastructure testing and implementation included:
   - Azure resource SKU selection. We used the on-premises evaluation data to match our on-premises servers with corresponding Azure VM sizes. In some cases, the matching process was challenging. We navigated the wide array of Azure VM SKUs, and ensured through VM testing that the SKUs we selected provided adequate performance for the environment. We also needed to examine potential performance bottlenecks within specific SKUs. For example, many SKUs fulfilled CPU performance requirements but did not support the disk throughput performance that we required.
   - Microsoft SQL Server clustering. Clustering presented a challenge to our implementation team. Our on-premises environment used clustering extensively, and a high-performance SAN supported storage requirements for the cluster. We used Azure Storage Spaces in Azure VMs and managed Azure storage accounts to create aggregated Azure virtual disks in striped sets to provide the required disk performance from a single volume/connection point.
   - Infrastructure as code. With such a large infrastructure footprint and so many dependent services, migrating a live application with minimal downtime wasn’t possible without automating infrastructure as code. To address this, we developed an infrastructure as code implementation using Transact-SQL (T-SQL), Azure Resource Manager, and Azure PowerShell to manage infrastructure deployment and performance. We then tested the code using continuous integration (CI) through Azure DevOps build pipelines, and continuous deployment (CD) through Azure DevOps release pipelines. Infrastructure as code enabled a more granular approach to resource deployment and management. Using it, we were able to implement smaller parts of Azure infrastructure quickly and automatically. This combination of granularity and automation greatly improved our ability to modify and grow the Mercury infrastructure to meet demand and test new scenarios. The following figure depicts the release pipeline in Azure:
Figure 2. Azure DevOps deployment release pipeline for Mercury

The infrastructure automation enabled by Azure allows us to select, deploy, and manage individual components at a more granular level than on-premises. Granularity capabilities span VMs, applications, and network components for any part of the solution within the system boundary.

Figure 3. Granular component selection in the Azure DevOps release pipeline

- Data consistency testing. As Mercury is a data platform that includes sensitive data, we had to make sure there was no data loss in the platform migration process. Ensuring data parity and consistency was particularly important. However, checking it manually would be a tedious process, so all data parity testing was automated to certify data accuracy.
- Management and security. Along with network segmentation, we segregated our environments using Azure Resource Groups. We used Azure’s built-in role-based access control (RBAC) to create permissions and isolation for each environment. We also used RBAC as a cleanup tool by removing persistent access and using just-in-time (JIT) access through Azure Active Directory Privileged Identity Management (PIM). We then leveraged Azure Security Center for vulnerability and configuration scanning.
- Monitoring capabilities. We required multiple different monitoring systems on-premises to monitor the on-premises version of Mercury. In Azure, we now have a single monitoring platform that supplies comprehensive end-to-end monitoring for all solution components. Azure Monitor provides alert,
notification, and dashboard functionality that we integrated with ticketing tools for tracking, and with Azure Automation to self-heal.

**Figure 4. The Mercury system architecture in Azure**

**Benefits**

Migrating Mercury to Azure has resulted in several benefits, including:

- **Increased agility.** Azure’s inherent scalability has enabled us to create a much more agile and cost-effective environment for Mercury. We can scale VM performance and SKU to match application demand using (and paying for) only the resources that we need. Using infrastructure as code, we can provision and deploy entire environments with a single click.

- **Reduced effort for availability and reliability.** Azure’s native disaster-recovery and high-availability features create a simpler management environment for backups, failover, and availability. Azure provides a built-in globally available platform that’s always on. We no longer rely on infrastructure maintenance teams to make changes in the datacenter for items such as networking or hardware.

- **More straightforward processes and simpler operation.** Automation across the platform makes almost every process in Mercury simpler and easier. We can devote more engineering effort to solution improvements and innovation, and less effort to maintenance and upkeep.

- **Increased sense of ownership due to greater visibility.** With our on-premises solution, we didn’t have a clear overview of solution costs and effectiveness. In Azure, we can understand the benefits relative to the costs, and the system components’ performance relative to demand.

- **Greater insights into security and vulnerability.** With Azure Security Center, the security of our environment is more transparent to our administrators. In addition, the availability of security-related information means vulnerabilities are identified sooner.

- **Increased performance.** We’ve experienced a 20 to 40 percent improvement in daily data refresh processes, depending on variables. This improvement enables us to act on insights and move data down the pipeline faster and more efficiently.
Moving forward

Our migration to Azure has created enormous and tangible benefits for financial reporting at Microsoft. Mercury in Azure has given us a more agile, more available, and easier-to-administer environment that can adapt quickly and seamlessly to the changing needs of our business. We’re still working on the continued transformation of Mercury in Azure. We’re investigating options for refactoring components to platform as a service (PaaS) and software as a service (SaaS) models to streamline our services further. This will result in even better performance at lower costs and an even more agile platform on which to build our financial reporting solutions.

For more information

Microsoft IT Showcase

microsoft.com/itshowcase

Learn about Microsoft’s cloud centric architecture vision
Learn about how Microsoft is implementing modern engineering
Learn how Microsoft is designing a modern service architecture in Azure

© 2020 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.