Migrating System Center Configuration Manager from on-premises infrastructure to Microsoft Azure

For 25 years, System Center Configuration Manager has run using on-premises infrastructure. Now, Microsoft and other global companies face the challenges of scarcity and rising prices of server hardware needed to maintain physical datacenters. Companies managing essential services like Configuration Manager with on-premises infrastructure face continued hidden and explicit datacenter costs, fewer options for scaling services based on demand, and less real-time insight into app and data security.

Until recently, Microsoft ran Configuration Manager using a mixture of on-premises virtual machines (VMs) and physical servers as the primary change-management tool. Microsoft’s ongoing effort to move mission-critical services to the cloud meant reevaluating this configuration. As a result, Microsoft Core Services Engineering and Operations (CSEO), formerly Microsoft IT, did a lift-and-shift migration of Configuration Manager infrastructure to Microsoft Azure, improving uptime, reliability, and scalability. Running Configuration Manager on Azure frees it from underlying hardware limitations, essentially future-proofing the service.

Assessing the challenge of migrating on-premises Configuration Manager to Azure

It doesn’t make a lot of sense, financially or otherwise, to keep running mission-critical services using on-premises datacenters. When you build out datacenter hardware, you purchase equipment for surge, not baseline usage. On-premises infrastructure hardware offers limited scalability. Once you buy hardware, even if you’re not using it at capacity, you still have the associated costs of ownership, such as electricity, cooling, fire suppression, security, and building space.

To make things worse, the costs for acquiring, maintaining, and upgrading infrastructure hardware are increasing, and the total cost of ownership is hidden in other budget-line items. These factors make it difficult to know the total cost for an additional server as well as the total cost for providing a specific service.

Microsoft, like a growing number of cloud-first companies, now limits its purchase of physical datacenter infrastructure hardware to maintain products that don’t support the cloud or Infrastructure as a Service (IaaS). Shifting Configuration Manager to Azure removes our hardware constraints, allows us to reduce and more accurately track operating costs, and increases our flexibility in both scaling up and down to meet usage needs. It also enables us to provide the Configuration Manager development team feedback for product improvements.

Formulating a migration strategy

We started building our migration strategy just over a year ago with a three-person team experienced in Configuration Manager, SQL, PowerShell, and Azure. Our first decision: whether to create a new architecture or move the architecture as-is.

Many IT pros dream of starting from scratch, but our services are highly complex and contain many interdependencies. Creating a new architecture would increase both the total cost and the length of the migration. That course of action also held the risk of an unintentional loss of service. The answer: a lift-and-shift strategy. In other words, move existing Configuration Manager VMs and physical servers to Azure, which provides the same functionality as the on-premises infrastructure, without changing their structure or functionality.
The second question was whether to migrate by site-system role or by site. After an in-depth examination of the architecture, we found our roles were too interdependent to make a role-based migration viable. That meant performing the migration by site.

**Measuring network connectivity and latency**

As one of the first companies to migrate Configuration Manager from on-premises VMs to Azure, we had no data or pre-existing how-to resources to reference. We determined that a phased approach would let us learn on the job and apply our experience from each migration to those that followed.

Our first priority was to ensure that service to our nearly 400,000 clients worldwide wouldn’t degrade. Prior to the migration, a Microsoft employee who wanted to install software went to the Software Center on their laptop or PC and clicked a button. The request was sent to the on-premises Configuration Manager, which then retrieved the appropriate files and installed them on the employee’s computer. We had to ensure the latency a user experienced from an on-premises VM didn’t radically increase with our migration to Azure.

We also performed manual and automated tests in our preproduction environment to compare latencies between different Azure regions for our on-premises client base. Most Azure geographies contain multiple Azure regions, but not all Azure regions were available to us because of the Microsoft Azure ExpressRoute connection points that CSEO had already chosen. New Azure customers could pick ExpressRoute connection points that best fit their needs, but we needed to use our existing endpoints. This restriction meant that Azure regions yielded varying latency, and not necessarily in a predictable way. For instance, the Microsoft employee trying to install software might have experienced only five minutes of latency using on-premises Configuration Manager. Running the same installation on Configuration Manager in one Azure region within the Azure geography could take 20 minutes, but running it in another region could take only six minutes. Our testing let us pick the best Azure region within an Azure geography to minimize the latency impact for users of each site.

**Deciding what to migrate**

Given our test results and the complexity of our setup, we knew that we couldn’t migrate all Configuration Manager components to Azure without causing service degradation. To support our clients worldwide, we had configured a Central Administration Site (CAS) with five primary sites, 12 secondary sites, and load balancers—all running on-premises, as seen in Figure 1. We decided to migrate our five primary sites and CAS but leave the secondary sites on-premises. As much as we wanted to move primary sites and discontinue secondary sites, removing the secondary sites altogether would have to wait for future upgrades to our on-premises network infrastructure. Load balancers would remain on-premises.
Determining the order of migration

After we decided what to migrate, we needed to figure out the migration sequence. Given our structure, we planned to move the primary-site servers first and the CAS last. Each primary-site migration started with the primary-site server, followed by the primary server’s SQL clusters, and finished with site systems. By design, our first migration would serve as a learning experience and blueprint for the primary sites that followed, so we put each part of the first primary-site migration into its own phase. The automation team used these phases to observe the manual processes so that we not only could evaluate our results but create resources to automate later site migrations.

The CAS ranked as a higher risk move because of its central role in communication among all the other sites. When a primary site has an outage, the outage has a great impact on the site’s users but presents minimal risk beyond that. The site’s users are stranded, but users of other primary sites are unaffected. Conversely, if the CAS has an outage, the impact on users is minimal because they can still access their primary sites. The risk of further problems, however, is high because the primary sites can no longer be administered centrally. We use the CAS to change a policy or push a critical software update to all primary sites with a minimum risk of errors. Without the CAS, we’d need to make the change manually on each primary site in violation of our established procedures, increasing the possibility of inconsistently deploying the change or making other mistakes.

Migrating Configuration Manager to Azure

Our phased plan was to migrate one site per month and build in evaluation and stabilization periods between each part of the migration. That plan would give us time to validate our results and resolve any service-disrupting issues before we moved to the next step.
Migrating primary sites

A primary-site migration started with backing up and restoring the primary-site server. We used the same server name and drive layout in Azure as we did on-premises. The Azure VM name differed, but the NetBIOS server name remained the same. When we were ready, we turned off the on-premises site VM and turned on the Azure VM and performed a Configuration Manager site recovery.

Next came SQL. We’d been using SQL clustering rather than SQL Server Always On Availability Groups. If we’d been using SQL Server Always On, we could’ve just added SQL nodes in Azure and then migrated directly. But Azure prevents the use of SQL clustering.

As a result, we needed to move to using SQL Server Always On. First, we created a new two-node SQL Server Always On setup in Azure. Because of the size of our databases and our desire to minimize downtime for the migration, we started with a backup and copied it to Azure. We then followed with deltas until we were ready to switch over. After completing the last delta, we turned on the Azure SQL VMs and ran a Configuration Manager site reset to point at the new SQL database location.

Next, we moved site services. After we set up our distribution points (DPs), management points (MPs), and fallback status points (FSPs) in Azure, we turned off the roles on-premises and validated proper functionality and normal business processes over time to ensure that there was no interruption to services. Then we started decommissioning the on-premises VMs.

Understanding SQL Server Always On Availability Groups drive performance in Azure

Migrating SQL from on-premises traditional disks to Azure VMs and disks produced some unexplained metrics. As part of our premigration investigation, we examined our SQL performance on the on-premises cluster to measure our IO per second (IOPS). We then architected our SQL Azure VMs to handle that level of IOPS. We used p30 disks with striping to link and spread the reads and writes across multiple disks so that we could achieve the IOPS we had determined were necessary.

When we moved to Azure and monitored our SQL performance, however, our metrics suggested we weren’t getting the performance we’d expected. Our performance-monitoring metrics indicated that performance appeared to be poor. We’d see a huge spike in read-write disk delays, but after a couple of seconds or a minute everything would be fine.

After some investigation, we learned that although IO per second is a standard measurement, Azure does some blocking in 50-millisecond increments. In the end, this behavior wasn’t a performance problem. Configuration Manager never noticed the difference, and it ran with the load as it always had.

Assessing WSUS and load balancers

Configuration Manager doesn’t recommend using load balancers. We have some existing services, however, that require them. Azure’s internal load balancing (ILB) functionality provided us the option to keep our current design without a service interruption, but we’ll eventually discontinue ILB use with Configuration Manager in a separate, future project.

Addressing security concerns

Concurrent with the Configuration Manager migration to Azure, we worked on security initiatives. Originally, we planned on using one subscription for all of Configuration Manager, but we were changing from traditional internet-based MPs to cloud-management gateways (CMGs). Putting MPs and CMGs into one Azure subscription set off alarms for Microsoft security groups.

Here’s the concern: a single subscription on ExpressRoute has corporate connectivity to everything inside Microsoft. Even though Azure uses different subnets for internet-facing roles, and those subnets don’t have any link to our ExpressRoute subnets that connect to the corporate site, we decided that running both on the same subscription was too high a security risk. Instead, we created a second subscription that isn’t connected to any ExpressRoute—a sort of Azure perimeter network—where we put all our internet-facing roles.
Introducing site server high availability
We used site server high availability, a new feature in Configuration Manager, to migrate the CAS. This allowed us to avoid the risks associated with a CAS outage while migrating the CAS from on-premises infrastructure to Azure.

While we were migrating our primary sites, the Configuration Manager product team introduced site server high availability. Now available for standalone primary-site servers, site server high availability allows you to install a site server in passive mode in addition to your existing site server in active mode. In other words, site server high availability provides the same functionality for your primary site that SQL Server Always On provides for your SQL database. If your active site server has an outage, the passive site server is available for immediate use, providing users with uninterrupted service.

At this time, site server high availability is enabled only for standalone primary-site servers. The Configuration Manager product team, however, asked us to test the new functionality in our complex hierarchy. We used this functionality to migrate the CAS from using on-premises infrastructure to Azure.

Normally, if you were using site server high availability, you’d create two VMs. One would host your site server in active mode, and the other would host your site server in passive mode. We used this functionality in the short term in our migration.

We migrated the CAS to Azure while the existing CAS was still running on-premises, but we didn’t create it as a new site server. We created it as a clone of the original. The on-premises CAS was in active mode, and the Azure CAS ran in passive mode. During the transition, the passive Azure CAS stayed in sync with the active on-premises CAS. When all aspects of the CAS were migrated and in sync, we swapped the active and passive modes. The Azure CAS began running in active mode, and the on-premises CAS switched to passive mode. After we validated the migration, we uninstalled the passive on-premises version. The active version on Azure took over with only a short 10-minute downtime caused by the migration.

Optimizing with Peer Cache and BranchCache
In migrating Configuration Manager to Azure, we used Peer Cache and BranchCache to reduce traffic on the ExpressRoute network pipe. Peer Cache and BranchCache provided lower latency file copies and a better user experience. As a result, we reduced our Azure costs by reducing the number of DPs from 94 to 52, as shown in Figure 2.
Automating to improve migration efficiency and duration

From the start, we planned to automate migration and configuration processes to provide faster, more consistent, and error-free migrations. By the end of the project, automating migration and configuration processes yielded faster, error-free deployment of Configuration Manager site systems—30 minutes with all organizational requirements compared to the original several hours. It will also reduce Azure costs by allowing for easy scaling up or down of VMs and roles.

Automation started with investigating the manual work involved in building and configuring VMs and installing Configuration Manager site roles. We soon found that internal process documentation for creating and migrating resources wasn’t complete and, for some tasks, didn’t exist. So, we needed to fully document these manual processes before we could automate anything.

We deployed three automation strategies during the third primary-site migration (AU1 in Figure 1):

- We used different Azure Resource Manager (ARM) templates for each site-system role because each has a different-sized VM. For this project, we automated three: MP and DP roles, which both use standard_F4 VMs, and software update point (SUP) roles, which use standard_F8 VMs.
- We used a custom script extension to install Configuration Manager site roles during deployment.
- We applied a Desired State Configuration (DSC) baseline at deployment that configures other organizational standards we have for IIS and VM configuration.

Using custom script-extension integration with Azure ARM templates

We used the Configuration Manager PowerShell Module to install Configuration Manager site roles during deployment. We installed Configuration Manager site roles using partial scripts implemented with our ARM templates. These templates let the user specify the site-role types during deployment and install each role with our
default configuration. We also used automation to deploy and configure Azure ILBs and to configure SQL Server Always On availability groups. We used a standard PS deployment script to build those resources and move databases into availability groups more quickly and efficiently than when we relied on a manual process. Using automation cut the time required to complete VM builds and site-role configuration from a couple of hours to 30 minutes and resulted in an error-free Configuration Manager deployment that met all organizational requirements.

In the future, we’ll move some of these workloads from custom scripts to Azure Runbooks, which offer one-touch deployment for a Configuration Manager site role. Azure Runbooks provide this feature because they contain the ARM template for the specified site role as well as the DSC configurations relevant to that role. Azure can not only scale VM size up and down, but also scale roles up and down. Azure Runbooks also will reduce costs because we can use it to quickly scale up MPs or DPs in response to high loads or quickly scale them down when the load is low. Finally, we can use the webhook feature in Azure Runbooks to kick off deployment of specific site roles through other processes—such as through alerting or using another front-end interface—with little time or manual effort.

We’ll also use DSC to install Configuration Manager site-system roles, transitioning us from using custom script extensions to DSC nodes for each Configuration Manager site role. Ultimately, this transition will enable DSC to continually check to ensure that our proper configurations are applied, which helps us avoid making inadvertent changes to our servers. If we detect issues when reading the logs, we can reinstall the role through DSC, using its functionality to achieve even higher availability and more uptime.

Using custom scripts for Configuration Manager: roles
We started the AU1 site migration with MP, DP, and SUP Configuration Manager site roles because they rely on IIS. We configured the main DSC baseline for these VMs using IIS organizational standards for ports, configuration log location, and log size. After we set up a PS drive to the CAS, we retrieved site-role input parameters through our deployment script, which determined the type of VM to build and site role to apply. We used different PowerShell cmdlets based on these site roles to install our organizational configurations. For AU1, we manually added FSPs and app catalogs to the Configuration Manager site-role installations within our scripts. After that, FSP and app catalog automation became part of the next primary-site migration automation.

Using custom scripts for Configuration Manager: WSUS
Each primary site has a shared database that’s also connected to a shared content directory. As a part of the WSUS post-configuration steps, we need to specify the Azure internal load balancer (ILB) that the site uses. Then we can apply the SUP role. To accomplish this, we installed WSUS and set the relevant post-configuration parameters as part of our custom script prior to applying the SUP role.

Deploying the DSC baseline configuration
The DSC baseline we applied to all deployed VMs included our VM standards, our IIS standards, and some scheduled tasks. Prior to using the DSC baseline to deploy configurations with our organizational policies, we performed that setup manually. That process not only took longer to implement, but it also increased the risk of errors.

Lessons learned
This migration didn’t occupy all our time over the past year. We still had ongoing responsibilities, concurrently running projects, and emerging high-priority issues. So, although we started planning this project more than a year ago, the migration itself consumed only about two weeks of our time every month for eight months. Here’s what we learned that can help others more efficiently migrate Configuration Manager from on-premises implementations to Azure:

- **SQL Server Always On Availability Groups vs. SQL clustering.** SQL Server Always On is a newer functionality than SQL clustering. If you use SQL clustering to provide redundancy and high availability, you might not have experience with SQL Server Always On. However, Azure doesn’t currently support SQL clustering. If you’re already using SQL Server Always On, you can just add SQL nodes in Azure and then migrate directly.
• **SQL drive performance.** Although IOPS is a standard measure, Azure performs some blocking in 50-millisecond increments. This difference causes some surprises in performance metrics. Ultimately, these metrics don’t matter. Configuration Manager runs as usual.

• **Hybrid state latency.** During the migration, we saw increases in latency when we were in a hybrid state, with some services still running on on-premises infrastructure and some running in Azure. This latency can be minimized. For example, you can move MPs immediately after your SQL migration to avoid issues resulting in state message backlogs.

• **Network connectivity.** At Microsoft, we use ExpressRoute between various on-premises datacenters and Azure datacenters. ExpressRoute both control and limit network traffic, but Microsoft is putting all its services into Azure. The pipe, as big as it is, keeps getting filled. That reality affects speed and latency, and pipe upgrades aren’t seamless. Because of how we built ExpressRoute, we needed to implement rolling outages during a weekend for some recent upgrades.

• **Azure ILB.** Configuration Manager documentation recommends against using load balancers. If your company doesn’t have older services that require load balancers, Azure ILB isn’t something you’ll have to think about. You can use “SUP list” for WSUS, and MPs now provide app catalog functionality. Azure ILB worked out well for us with our older services. If, like us, you also have older services that rely on load balancers, you should determine which VMs you really need in order to properly configure them before adding them to the ILB. For instance, when you set up a load balancer for WSUS, you configure your VM to use port 80. If you want to use port 443, however, you need to tear down and rebuild the entire VM. You can address transitioning to using Configuration Manager functionality rather than Azure ILB after your migration.

• **Azure Just In Time (JIT).** Azure has its own JIT functionality, and as you move to more secure stances, it’s handy to have help in ensuring you have a secure infrastructure for Configuration Manager. As part of concurrently running security initiatives, we moved to JIT security. So when we arrive at work, our accounts don’t have administrative credentials on VMs. Everyone must go through a JIT process and request access to VMs only when it’s needed. Under JIT security, we receive elevated permissions to those VMs for a specific timeframe. Eventually those permissions age out, and JIT removes them. This approach ensures users have access only when necessary.

• **Security concerns.** Microsoft considers placing ExpressRoute and internet-facing subnets into one subscription to be a security risk. Some companies might not care about that and accept the cohabitation of those roles with one subscription with separate subnets. You’ll need to work out that issue with the security groups at your company.

• **Azure quotas.** The cloud isn’t endless. Be aware that Azure has quotas, both for overall VM count and per VM type. You can ask for those quotas to be increased—we’ve done so several times—but make sure the quotas in Azure will support whatever hierarchy you design.

• **Azure hands-on expertise.** More than a year ago when we started this project, fewer people had expertise in or familiarity with Azure. As Azure becomes more commonly used, this issue solves itself. Remember, however, that Azure is just a datacenter managed by other people. The hardware is extrapolated from you, but it’s still another VM running Windows in a datacenter with redundancy built into it and networking concerns you need to address.

• **Boundary groups.** Over time, boundary groups can get messy. Get the best result by cleaning up your boundary groups before implementing Peer Cache and BranchCache.

• **Migration and configuration documentation.** Before you start a migration, ensure that full documentation of the migration, creation, configuration, and management processes of resources is available. That documentation might not exist, and you can’t automate those processes without it.

### Next steps

Many companies are moving to a cloud-first strategy. Before you consider migrating Configuration Manager to Azure, consider these issues:
• Your total cost of ownership for on-premises infrastructure, including hidden costs such as electricity, cooling, and building space
• Your service-network consumption and layout
• Your SQL IOPS—specifically, what you need compared to what you now have

For more information

Microsoft IT Showcase
microsoft.com/its

Microsoft Azure
Moving the WSUS database to SQL Always On
BranchCache vs. Peer Cache
Site server high availability in Configuration Manager
Migrating mobile device management to Intune in the Azure portal
Securing the hybrid cloud network with Azure ExpressRoute

© 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.