Microsoft

Managing Microsoft 365 with Microsoft365DSC and Azure DevOps

Date

January 15, 2024

Version

v3.0 - Draft

Prepared by

Yorick Kuijs (yorick.kuijs@microsoft.com)
Cloud Solution Architect

Contributors

Andras Varga (avarga@microsoft.com)
Senior Consultant

Changelog

Version	Date	Author	Changes
1.0	Nov 1, 2020	Yordan Bechev Yorick Kuijs	First release
1.0.1	Nov 3, 2020	Yorick Kuijs	Updated incorrect links
1.1	Dec 2, 2020	Yorick Kuijs	Incorporated feedback from Zaki Semar Shahul Added Azure Conditional Access for the used service account
1.2	Oct 1, 2021	Yorick Kuijs	Corrected issues Added Certificate authentication scenario
1.21	Dec 23, 2021	Yorick Kuijs	Corrected download link to scripts after migration to new website
2.0	Nov 23, 2022	Yorick Kuijs	Major update: Combining scenarios, demonstrating new flexible setup. Reviewed by Brian Lalancette, Andi Krüger, Jeffrey Rosen, Andrew Piskai, Dean Sesko and Albert Boland.
3.0	Dec 1, 2023	Andras Varga	Major update: Complete restructuring and extension, support for both Microsoft-hosted and self-hosted VMs, support for data file split, added module centralization and caching, transforming from screenshots to descriptions

MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, our provision of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The descriptions of other companies' products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Microsoft. Microsoft cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

© 2013 Microsoft Corporation. All rights reserved. Any use or distribution of these materials without express authorization of Microsoft Corp. is strictly prohibited.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Table of Contents

1 Introduction

Microsoft 365 is a very popular productivity cloud solution. Each customer has their own tenant which stores their data, applications and configuration. Using the Microsoft 365 admin center (https://admin.microsoft.com) and other admin centers, customers can configure and manage their tenants.

Many companies are adopting DevOps practices and are interested in applying them against Microsoft 365 as well. Infrastructure as Code and Continuous Deployment/Continuous Integration (CD/CI) are important concepts in DevOps.

Microsoft365DSC is a PowerShell Desired State Configuration (DSC) module that can configure and manage Microsoft 365 in a true DevOps style: "Configuration-as-Code".

1.1 Microsoft 365 and DevOps

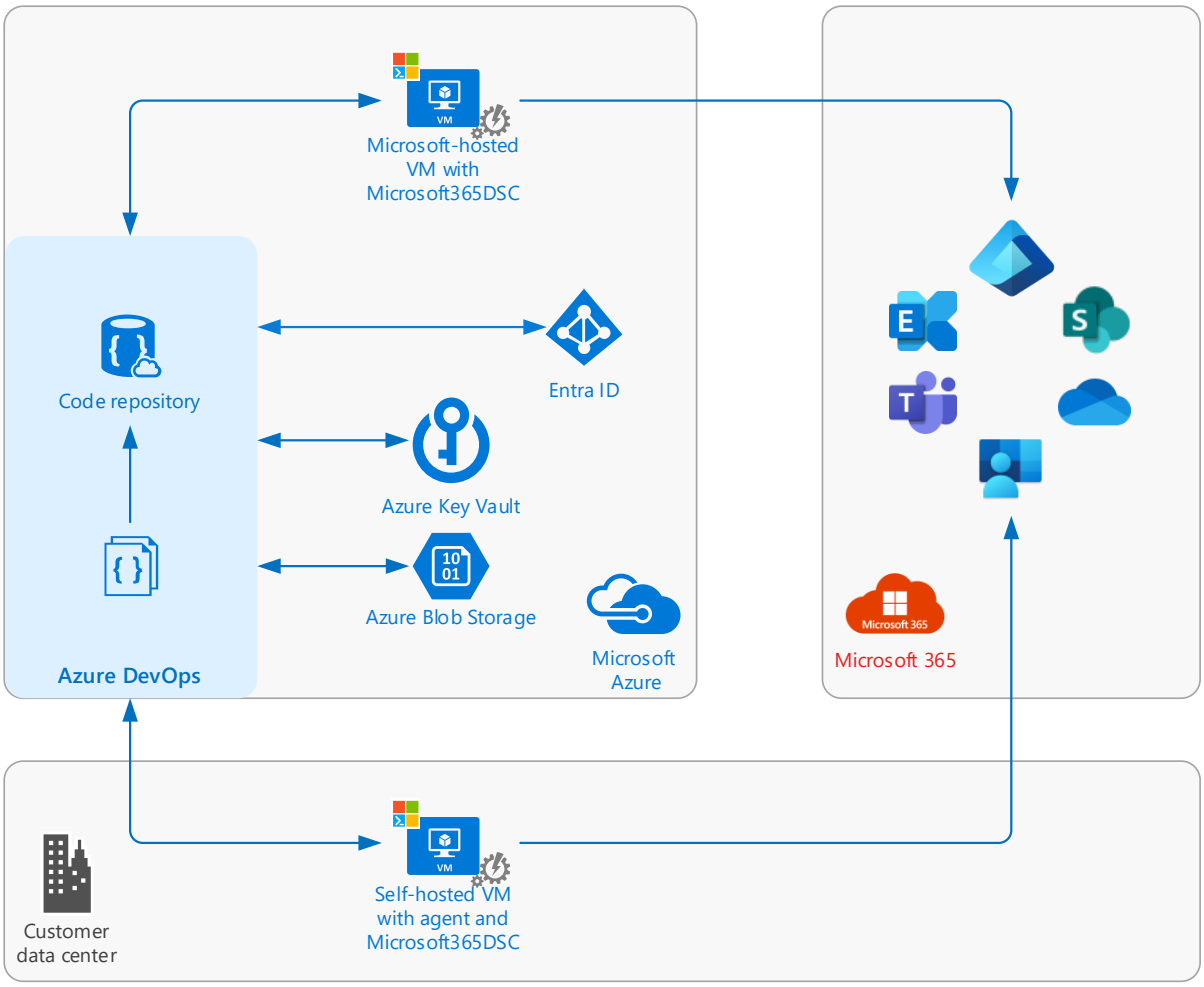
When you perform management of your Microsoft 365 tenant manually, there is no way to consistently deploy changes and to monitor for changes. By using "Configuration-as-Code" principles, you document/define the configuration of your tenant in code. You can then deploy this configuration programmatically to your tenant and periodically check if the defined/intended configuration still matches the actual configuration. The tool that allows you to do this is Microsoft365DSC (https://microsoft365dsc.com).

By adding CD/CI capabilities, for example by using Azure DevOps, you can also add additional quality gates making sure changes to your configuration are deployed in a controlled and consistent way.

1.2 Architecture

In this document we are going to describe the process and steps required to implement a basic Configuration-as-Code setup using Microsoft365DSC, Azure DevOps, Azure Key Vault, and Azure Blob Storage. Changes to Microsoft 365 are made within a Git repository in Azure DevOps and then fully and automatically deployed to a Microsoft 365 tenant.

The setup we are using looks like this:



1.3 Assumptions

This document assumes you are familiar with creating and deploying PowerShell Desired State Configurations (DSC) and have some DevOps background, as well.

Important:

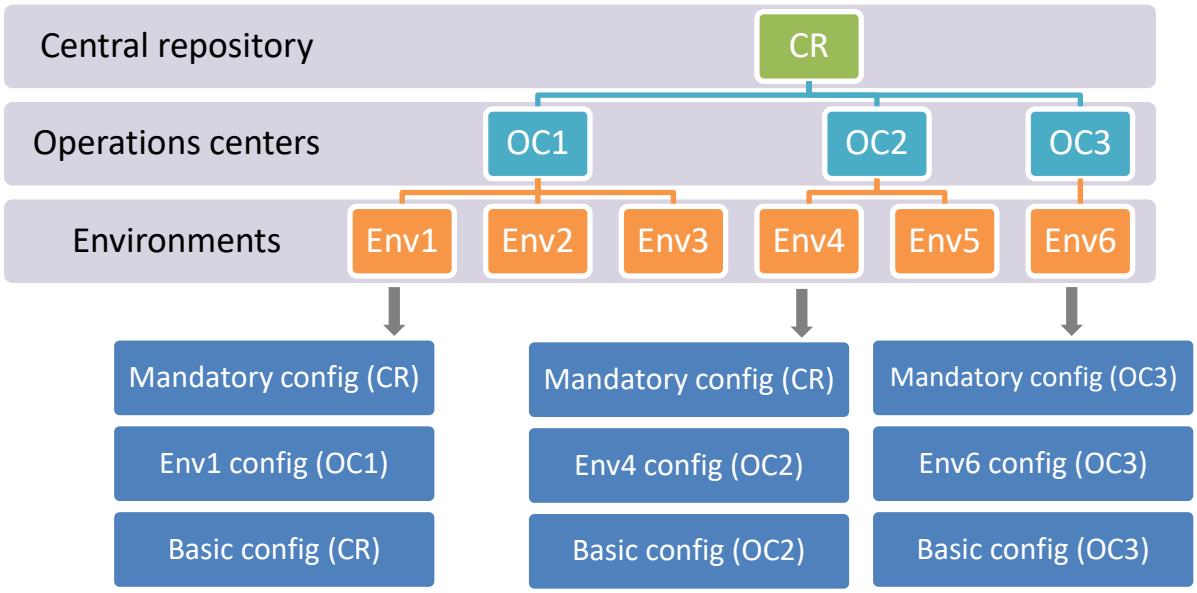
If you are new to PowerShell DSC, please first check out the first two links in section 9.1. These are recordings of two very good PowerShell DSC training courses that will give you a good understanding of what PowerShell DSC is and how it works. A good foundation for the skills you need when working with PowerShell DSC.

2 Solution Description

This solution consists of multiple components. In this chapter, the solution is described in more detail.

2.1 Model and Definitions

This solution follows a multiple operations center, multiple environment approach, which means that you can have multiple operations centers, each of which will be responsible for the configuration management of certain environments (Microsoft 365 tenants). Each environment can only belong to a single operations center, as shown in the following diagram.



- **Configuration layering model** – There are three layers of configuration you can include in your model:
  - **Mandatory settings** (priority 0)

    Mandatory settings override any other settings defined elsewhere.
  - **Environment-specific settings** (priority 1)

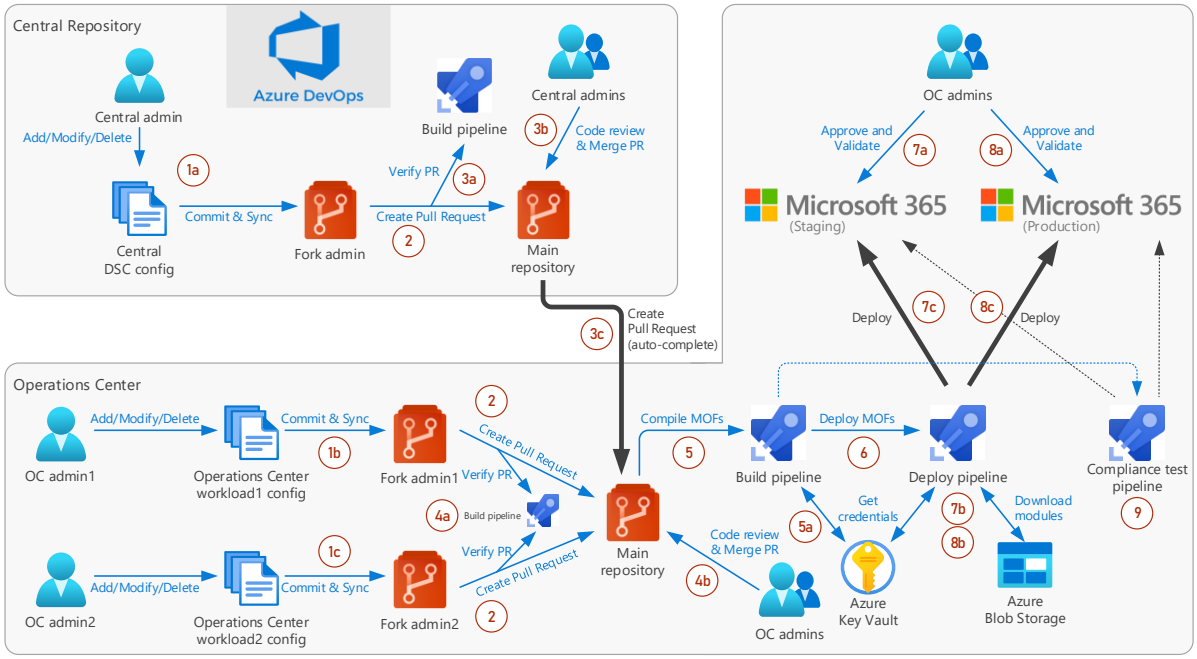
    Environment-specific settings take precedence over Basic settings but are overwritten by Mandatory settings.
  - **Basic settings** (priority 2)

    Basic settings only take effect if the same setting is not defined elsewhere.
- You may want to use a single layer only (any of the three), two layers (basic and environment-only or environment-only and mandatory), or all three layers.
- **Configuration definition model** – You can configure the above layers of configuration at two levels:
  - In a **Central Repository (CR)**
  - In the **Operations Center (OC)** an environment belongs to

- If used, mandatory configurations usually come from a central repository in a multiple operations center environment (see Env1 and Env4 in the diagram). In certain situations, with more independent operations centers, it can be defined in the OC, as well (see Env6).
- Environment-specific configuration is always defined at the OC level.
- If used, basic configurations usually come from a central repository in a multiple operations center environment (see Env1 in the diagram). When there is no need to define basic settings centrally, they can be defined in the OC (see Env4 and Env6).

2.2 Deployment Process

The following diagram shows the steps of the configuration change and deployment process.



The steps on the diagram are as follows:

1. An administrator edits the configuration in their own personal copy (fork). This can be either of the following:
   a. Central admin updates the Basic or Mandatory configuration data files in the central repository
   b. OC admin1 updates the data files for their own workloads in the operations center repository
   c. OC admin2 updates the data files for their own workloads in the operations center repository
2. When done, the admin creates a Pull Request to have his changes merged into the main repository at the given location (CR or OC)
3. In the central repository, the quality assurance process starts:
   a. An automated process runs certain quality checks against the Pull Request (optional)
   b. Other admins validate the changes via a peer review process, and when quality checks succeed, they merge the Pull Request
   c. After a successful code change, an auto-complete pull request to the OCs is also initiated

4. In each operations center, the quality assurance process starts:
    a. An automated process runs certain quality checks against the Pull Request (optional)
    b. Other admins validate the changes via a peer review process, and when quality checks succeed, they merge the Pull Request
5. The data file merge initiates a build pipeline at the OCs that retrieves the credentials from Azure Key Vault (5a) and compiles the so called *MOF files*
6. Once the build pipeline completes successfully, the deployment pipeline starts with sending a notification to the admins, and waits for their approval to all configured environments
7. When the OC admins are ready for deployment:
    a. They approve the deployment for the first environment (e.g., Staging)
    b. The deployment pipeline downloads the cached modules from an Azure Blob Storage, retrieves the credentials from Azure Key Vault, and finally,
    c. Deploys the generated MOF file to the first environment
8. After a successful deployment, the admins check if the change has been deployed successfully and if the desired result has been achieved. When they are ready to continue with the deployment:
    a. They approve the deployment to further environments (one by one or parallel – e.g., Production)
    b. The deployment pipeline downloads the cached modules from an Azure Blob Storage, retrieves the credentials from Azure Key Vault, and finally,
    c. The deployment pipeline performs the deployment to the other environments
9. The change is now automatically and consistently deployed to all environments, and can be checked with the Compliance test pipeline on a scheduled basis

2.3 Azure Components

2.3.1 Azure Key Vault

Azure Key Vault is used to store all service account and application credentials for a given operations center that are required for successful Microsoft365DSC deployments. These credentials are protected with access control and are downloaded by the pipelines only when they are required. There is a one-time upload process, and thereafter, whenever a pipeline in Azure Pipelines runs, it downloads these credentials. Access to Azure Key Vault is provided via an app registration associated with a service principal. Azure DevOps then uses a Service Connection to access the secrets stored In Azure Key Vault.

2.3.2 Azure Blob Storage

Azure Blob Storage is used to cache all PowerShell modules that Microsoft365DSC is depending on – including the required version of the Microsoft365DSC module itself – to speed up the pipeline execution times, especially when using Microsoft-hosted agents. The zipped modules are stored in an Azure Blob Storage which is accessed by Azure DevOps via the same service principal and Service Connection as the Key Vault.

2.3.3 Azure DevOps

In Azure DevOps, you can use various components:

- Azure Pipelines for data validation, compilation, deployment and compliance check
- Microsoft-hosted agents and self-hosted agents
- Environments with approval workflows

This solution uses all these components.

2.3.3.1 Pipelines

There are four pipelines included in this solution:

- **Preparation**

  This pipeline downloads all PowerShell modules that the given DSC version depends on (including Microsoft365DSC itself), compresses these into a zip file, and uploads this package to the previously created Azure Blob Storage.

  It will only run automatically, when there is a change in the required Microsoft365DSC version.

- **Build**

  This pipeline compiles the DSC configurations into MOF files and create a deployment package. One MOF file per environment is created. The result of the pipeline is an output folder in which all components are placed that are required for the deployment of the MOF files. These are:

  - The MOF files themselves
  - The deployment script and the compliance check script
  - The DSCResources.psd1 file, to determine which version of Microsoft365DSC and generic modules must be used
  - The PsExec.exe tool, to import the authentication certificate into the LOCAL SYSTEM's personal certificate store for the deployment of certain resources

  At the end of the pipeline, the entire Output folder is packaged as a zip file and attached to the build pipeline as an artifact.

  The build pipeline is automatically triggered when there's a change in you resource definition data files.

- **Deployment**

  This pipeline deploys new configurations to the target environments. It uses the build artifacts to deploy the generated MOF file to the corresponding environment. The sample code only deploys to one environment (Production), but you can easily add additional environments via a configuration file. Each environment has its own stage in the pipeline, which enables you to define individual approval workflows to the environments.

  A new run of the deployment pipeline is automatically triggered after every successful build.

- **Compliance test**

  This pipeline can be scheduled to periodically check if the environments are still in the desired state and send a notification of the results to either an email address or a Teams channel. It uses the MOF

files included in the build artifacts to compare them one-by-one with the current online state of configuration.

2.3.3.2 Virtual Machines

Any of the above pipelines can use either **Microsoft-hosted** or **self-hosted** virtual machines (agents on a virtual machine) to run the corresponding scripts. Depending on your personal preferences and security requirements, you can run all pipelines online or all on-premises on your VMs but you can also follow a mixed approach. In this guide, you will find execution and configuration steps for both.

2.3.3.3 Environments

Environments correspond to Microsoft 365 tenants, each of which is bound to one of your operations centers. As already described, the deployment and compliance test pipelines use Azure DevOps environments to represent your tenants. For your convenience, you can define these environments in a configuration file, and Azure Pipelines will create them automatically for you.

The only thing you have to do manually, is to add approval workflows to your environments. This is described later in the document.

2.3.4 Entra ID

Microsoft Entra ID is the identity store for this solution. It stores your admin accounts, app registrations, service principals, app permissions, and role assignments.

Although there are many possibilities to authenticate to Microsoft 365 services (see Authentication and Permissions - Microsoft365DSC - Your Cloud Configuration), in this solution we only use Service Principals with certificate authentication to access your Microsoft 365 workloads. This is one of the most secure ways of authentication and also supports all workloads that are implemented by Microsoft365DSC.[1]

Similarly, all other cloud access (access to Azure resources and email services) is made available via service principals, by use of client secrets.

The following table provides an overview of all app registrations and their purpose.

Name	Description	Quantity and Location
Microsoft365DSC Deployment	This app registration is used by Microsoft365DSC to authenticate towards the Microsoft 365 tenant using a certificate secret.	One per managed environment, in the corresponding tenant
Microsoft365DSC Azure Access	The Azure DevOps project is using this app registration to authenticate towards Azure, retrieve credentials from the Azure Key Vault, and download cached PowerShell modules from Azure Blob Storage.	One per operations center, in a freely selected tenant
Microsoft365DSC Email Notification	If you choose to use email to send status reports, you need an app registration to authenticate against Microsoft 365 so you can use Exchange Online as the SMTP server.	One per operations center, in a freely selected tenant

[1] Currently, there are two individual Teams resources that are not supported with service principals

2.4 Certificates

The following table provides an overview of all certificates registrations and their purpose.

Name	Description	Quantity
Microsoft365DSC Authentication Certificate	This certificate is used by Microsoft365DSC running in Azure Pipelines to authenticate towards the Microsoft 365 tenant when accessing it via service principal.	One per managed environment
Encryption Certificate	If you choose to leverage service account-based authentication for any of the workloads instead of service principals, this certificate is used to encrypt credentials in compiled MOF files.	One per operations center

2.5 Microsoft365DSC Configuration

The Microsoft365DSC configuration uses so-called Composite Resources, which are a way to structure DSC resources into separate configurations. So instead of creating one huge DSC configuration file with all DSC resources for all workloads, which will become very hard to read and maintain, you now have multiple smaller and dedicated composite resources and one main DSC configuration (M365Configuration.ps1) which is responsible for calling each of the composite resources.

This is made possible by two solution components: the M365DSC.CompositeResources module and the split data file logic.

- **M365DSC.CompositeResources module** – This is a dynamically generated module for every new Microsoft365DSC version that is published in PSGallery, and can be downloaded by Azure Pipelines. It contains a composite resource for each workload. Each composite resource contains all DSC resources for that workload, which makes it much easier for you to compose your own data files.
- **Split data file logic** – This logic is built into the build script, and enables you to split your resources into multiple data files based on workloads, operational responsibilities, transparency, etc., instead of a single file that contains all your settings. With the split approach you can easily set up an edit control model for your individual data files. (E.g., when Exchange admins can only edit the Exchange data files, while SharePoint admins can only edit the SharePoint data files.)

Note:

You can find examples for this in chapter 5.

There is one more module to mention that is used by the solution: **M365DSCTools**. This public module is downloaded by the pipelines from PSGallery, and it contains functions that support pipeline script activities.

2.6 Customize the Solution

This solution supports an easy customization of the included code package to fit your specific situation. Better yet, you should update the factory configuration with your own settings!

Customizations are supported by the following components:

- **DSC version definition** (DscResources.psd1)

  It allows you to specify the Microsoft365DSC version you want to use for your deployments. Furthermore, it lists the required generic modules, for which you can also modify the version to use (for advanced use cases only).

Important:

You should be careful when changing to a new version because it may contain changes in the resource definitions that may make you review and modify your data files.

- **Global pipeline variables** (Pipelines\variables.yaml)

  It's a configuration file for almost all customizable settings for a specific operations center. The behavior of the pipelines will depend of the values you provide here. For most situations, simply editing this configuration file will eliminate the need to touch the pipeline definitions.

Note:

There are some operations center-specific secrets that would belong to this configuration file, however, storing secrets in clear text is not a recommended way to go. This is why these secrets are stored in Azure Key Vault, and you will find hints in *variables.yaml* on how to populate these.

- **Pipeline definitions** (Pipelines*-pipeline.yaml)

  Although they are written as universal and adaptive, based on the variables, there are specific settings that can only be adjusted in these files (e.g., scheduler settings and the choice between Microsoft-hosted and self-hosted VMs).

3 Prerequisites

3.1 Virtual Machine

No matter if you go for a pure Microsoft-hosted solution, a self-hosted solution or a mixture of these two, you will require computers for deploying the solution and run your pipelines on.

Here is an overview of what you need depending on the solution you choose. In all cases, the computers can be either physical or virtual machines. For simplification, we assume the use of a virtual machines for self-hosted agents and physical machines for deployment.

Purpose Type of Solution	Deploying the Solution	Running Azure Pipelines
Microsoft-hosted	One deployment workstation per operations center	N/A
Self-hosted	One virtual machine per operations center to act as both: ▪ Deployment workstation and ▪ Azure Pipelines agent	
Mixed	One virtual machine per operations center to act as both: ▪ Deployment workstation and ▪ Azure Pipelines agent	

The requirements for the physical or virtual machines are:

- Windows Server 2016 / Windows 10 or above
  - Recommended to have at least 2 CPUs and 8 GB of memory
  - x64 version is required

  Note:

  Using the ARM version of Windows is **not** supported
- .Net Framework 4.7 or higher
  - Download .NET Framework | Free official downloads (microsoft.com)
- PowerShell v5.1
  - Installed by default on all current versions of Windows Server

  Note:

  Later PowerShell versions aren't supported at this time, because some modules used by Microsoft365DSC don't support those PowerShell versions yet.

3.2 Azure

This solution uses various Azure Components. You must have all required Azure components listed in section 2.3 available and functioning to deploy this solution successfully. Moreover, you will need to create an Azure DevOps organization and permissions to configure this organization.

3.3 Microsoft 365

You will also need at least one Microsoft 365 tenant, that can be managed using Microsoft365DSC, and that hosts all required solution components.

In all your tenants that you plan to manage or where you plan to host solution components, you need an Entra ID account with administrative privileges. Although more limited permissions may suffice, we assume you have an account with Global Administrator permissions.

3.4 Licenses

You can either use a fully licensed or a trial version of the above-mentioned products.

Microsoft365DSC, M365DSC.CompositeResources, and M365DSCTools are open-source modules, available under an MIT license (https://github.com/microsoft/Microsoft365DSC/blob/master/LICENSE), which means that you do not need to purchase any license and can use it for free.

4 Deployment

The preparation steps are divided into three categories, that are also indicated in the section title:

- **Common** (without any indication): to be executed for all types of deployments
- **For Microsoft-hosted solution**: to be executed only if you're solution is purely cloud-based, that is, it only relies on Microsoft-hosted virtual machines or if you use a mixed approach
- **For self-hosted solution**: to be executed only if you're solution only relies on self-hosted virtual machines or if you use a mixed approach

At the end of each section, we also indicate whether the described steps should be executed once, once for every operations center, or per environment.

When you see text highlighted with light blue color, it indicates that you can freely choose your string value for it, or substitute your previously chosen value. In most cases, we also provided recommended values which you can either accept or make up your own.

Note:

Because of the frequent development of the solution components included in this document, you may notice slight changes in the management UI at the time of deployment, compared to this document. If you notice changes that make it hard to follow the described steps, don't hesitate to report it to the authors.

4.1 Preparing the Deployment Workstation (for Microsoft-hosted Solution)

4.1.1 Configure PowerShell Requirements

This solution needs a few components to be installed on a deployment workstation that will be used to configure the necessary cloud components. You will need one deployment workstation per operations center (only one center described in this whitepaper).

In this step we are going to install these components:

- Log on to the deployment workstation with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Update PowerShellGet by executing the following commands:

```
Install-PackageProvider NuGet -Force
Install-Module -Name PowerShellGet -Force
```

Note:

If you run into issues downloading these updates, check out the following article: PowerShell Gallery TLS Support - PowerShell Team (microsoft.com)

It is possible that the PowerShell Gallery isn't registered correctly in your installation. In that case *Get-PSRepository* will not return any results. If so, run the following command:

```
Register-PSRepository -Default
```

- Install all necessary modules by executing the following command:

```
Install-Module Microsoft.Graph, Microsoft365DSC, Az.Resources,
Microsoft.PowerApps.Administration.PowerShell -Force
```

- Install the dependency modules for the previously deployed Microsoft365DSC version:

```
Update-M365DSCDependencies
```

4.1.2 Create the Microsoft365DSC Authentication Certificate

To authenticate against Microsoft 365, we need a self-signed certificate. In this section, we are going to create this certificate:

- Log on to the deployment workstation with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Create and export a new self-signed authentication certificate by running the following PowerShell commands:

Note:

Update the <password> parameter to your own password and define your own path to export the certificate to

```
$tempPath = "$($env:USERPROFILE)\Downloads"
$DSCCertPwd = "<password>"
$DSCCertName = "Microsoft365DSC"

$DSCCert = New-SelfSignedCertificate -Subject "CN=$DSCCertName" -CertStoreLocation
"Cert:\LocalMachine\My" -KeyExportPolicy Exportable -KeySpec Signature
$password = ConvertTo-SecureString -String $DSCCertPwd -AsPlainText -Force
Export-PfxCertificate -Cert $DSCCert -FilePath "$tempPath\M365ClientCert.pfx" -Password
$password
Export-Certificate -Cert $DSCCert -FilePath "$tempPath\M365ClientCert.cer"
```

Note:

You can get the certificate on the deployment workstation at a later time with this command:

```
$DSCCert = Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object {$_.Subject -eq
"CN=$DSCCertName"}
```

For more information on creating a certificate for application authentication, see: Create a self-signed public certificate to authenticate your application | Microsoft Learn

Note:

Repeat these steps for each environment you are going to manage.

Important:

When your certificate expires, you should execute these steps again, and also update your configuration at all places, where your certificate file or certificate thumbprint was used (see sections 4.7.1 and 4.7.2).

4.1.3 Create the Microsoft365DSC Encryption Certificate

If you chose to use service account-based authentication for any of the workloads, another certificate may be required for the Local Configuration Manager to encrypt credentials in compiled MOF files. In this section, we are going to create this certificate:

- Log on to the deployment workstation with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Create and export a new self-signed encryption certificate by running the following PowerShell commands:

Note:

Update the <password> parameter to your own password and define your own path to export the certificate to

```
$tempPath = "$($env:USERPROFILE)\Downloads"
$encryptionCertPwd = "<password>"
$encryptionCertName = "Microsoft365DSC Node Document Encryption"

$encryptionCert = New-SelfSignedCertificate -Type DocumentEncryptionCertLegacyCsp -
DnsName $encryptionCertName -HashAlgorithm SHA256 -NotAfter (Get-Date).AddYears(10)
$password = ConvertTo-SecureString -String $encryptionCertPwd -AsPlainText -Force
Export-PfxCertificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.pfx"
-Password $password
Export-Certificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.cer"
```

Note:

You can get the certificate on the deployment workstation at a later time with this command:

```
$encryptionCert = Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object
{$_.DnsName -eq $encryptionCertName}
```

Note:

Repeat these steps for each of your operations centers.

Important:

When your certificate expires, you should execute these steps again, and also update your configuration at all places, where your certificate file or certificate thumbprint was used (see sections 4.7.1 and 4.7.2).

4.2 Preparing the Virtual Machine (Phase 1 – for Self-hosted Solution)

4.2.1 Configure PowerShell Requirements

This solution needs a few components to be installed for it to work. In this step we are going to install these components:

- Log on to the virtual machine with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Update PowerShellGet by executing the following commands:

```
Install-PackageProvider NuGet -Force
Install-Module -Name PowerShellGet -Force
```

Note:

If you run into issues downloading these updates, check out the following article: PowerShell Gallery TLS Support - PowerShell Team (microsoft.com)

It is possible that the PowerShell Gallery isn't registered correctly in your installation. In that case *Get-PSRepository* will not return any results. If so, run the following command:

```
Register-PSRepository -Default
```

- Install the required PowerShell modules (Az.KeyVault, Az.Storage, and Pester) by executing the following command:

```
Install-Module Az.KeyVault, Az.Storage, Pester -Force
```

- (Windows client versions only) Enable Windows Remote Management by executing the following command:

```
Enable-PSRemoting -Force
```

Note:

Repeat these steps for each of your operations centers.

4.2.2 Create the Microsoft365DSC Authentication Certificate

To authenticate against Microsoft 365, we need a self-signed certificate. In this section, we are going to create this certificate:

- Log on to your virtual machine with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Create and export a new self-signed authentication certificate by running the following PowerShell commands:

Note:

Update the <password> parameter to your own password and define your own path to export the certificate to

```
$tempPath = "$($env:USERPROFILE)\Downloads"
$DSCCertPwd = "<password>"
$DSCCertName = "Microsoft365DSC"

$DSCCert = New-SelfSignedCertificate -Subject "CN=$DSCCertName" -CertStoreLocation
"Cert:\LocalMachine\My" -KeyExportPolicy Exportable -KeySpec Signature
$password = ConvertTo-SecureString -String $DSCCertPwd -AsPlainText -Force
Export-PfxCertificate -Cert $DSCCert -FilePath "$tempPath\M365ClientCert.pfx" -Password
$password
Export-Certificate -Cert $DSCCert -FilePath "$tempPath\M365ClientCert.cer"
```

Note:

You can get the certificate on your virtual machine at a later time with this command:

```
$DSCCert = Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object {$_.Subject -eq
"CN=$DSCCertName"}
```

For more information on creating a certificate for application authentication, see: [Create a self-signed public certificate to authenticate your application | Microsoft Learn]

Note:

Repeat these steps for each environment you are going to manage.

Important:

When your certificate expires, you should execute these steps again, and also update your configuration at all places, where your certificate file or certificate thumbprint was used (see sections 4.7.1 and 4.7.2).

4.2.3 Configure the Local Configuration Manager

If you chose to use service account-based authentication for any of the workloads, another certificate may be required for the Local Configuration Manager to encrypt credentials in compiled MOF files. In this section, we are going to create this certificate:

- Log on to your virtual machine with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Create and export a new self-signed encryption certificate by running the following PowerShell commands:

Note:

Update the <password> parameter to your own password and define your own path to export the certificate to

```
$tempPath = "$($env:USERPROFILE)\Downloads"
$encryptionCertPwd = "<password>"
```

```
$encryptionCertName = "Microsoft365DSC Node Document Encryption"

$encryptionCert = New-SelfSignedCertificate -Type DocumentEncryptionCertLegacyCsp -
DnsName $encryptionCertName -HashAlgorithm SHA256 -NotAfter (Get-Date).AddYears(10)
$password = ConvertTo-SecureString -String $encryptionCertPwd -AsPlainText -Force
Export-PfxCertificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.pfx"
-Password $password
Export-Certificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.cer"
```

Note:

You can get the certificate on your virtual machine at a later time with this command:

```
$encryptionCert = Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object
{$_.DnsName -eq $encryptionCertName}
```

- Run the following code in your PowerShell window to deploy the Local Configuration Manager configuration:

```
Configuration ConfigureLCM {
    Import-DscResource -ModuleName PsDesiredStateConfiguration
    node localhost {
            LocalConfigurationManager {
                    ConfigurationMode = "ApplyOnly"
                    CertificateId     = $encryptionCert.Thumbprint
            }
    }
}

$LCMConfig = ConfigureLCM

Set-DscLocalConfigurationManager -Path $LCMConfig.Directory
```

- To validate a successful configuration of the thumbprint, run the command below:

```
Get-DscLocalConfigurationManager | Format-Table -Property
CertificateID,ConfigurationMode -AutoSize
```

The **CertificateID** parameter should now show the thumbprint of your certificate and the **ConfigurationMode** should show **ApplyOnly**.

Note:

We configure the *ApplyOnly* setting because we will use a pipeline to implement the monitoring functionality, later in this document.

- Optional: Secure your certificate
  - Export the certificate to PFX format (already done)
  - Delete the certificate from the certificate store
  - Re-import the certificate from the PFX file but do not select the option to make the private key exportable
  - Import the PFX file into Azure Key Vault for secure backup

Note:

Repeat these steps for each of your operations centers.

Important:

When your certificate expires, you should execute these steps again, and also update your configuration at all places, where your certificate file or certificate thumbprint was used (see sections 4.7.1 and 4.7.2).

4.2.4 Create Azure DevOps Agent Service Account

The Azure DevOps agent needs a service account with the correct permissions. In this step we are going to create this account and assign local Administrator permissions:

- Log onto the virtual machine
- Open **Computer Management**
- Create a local service account, for example: **DevOpsAgent**

Note:

Make sure you use a long and complex password.

This account will be used to run the Azure DevOps agent with, which is used by Azure DevOps to deploy configurations to Microsoft 365.

- Add this account to the local **Administrators** group

Note:

Repeat these steps for each of your operations centers.

4.3 Preparing the Microsoft 365 Tenant

4.3.1 Connect to Microsoft Graph

This chapter uses Microsoft Graph PowerShell SDK calls for any configuration actions. Therefore, you should connect to Microsoft Graph first.

- Log on to the deployment workstation with Administrative credentials
- Open an **elevated** Windows PowerShell window
- Connect to Microsoft Graph with the following commands, providing your global admin credentials when prompted:

```
Connect-MgGraph -Scopes Application.ReadWrite.All, AppRoleAssignment.ReadWrite.All,
Directory.ReadWrite.All, RoleManagement.ReadWrite.Directory -NoWelcome

$tenantName = (Get-MgDomain | Where-Object {$_.isInitial}).Id
```

Note:

This command also saves your tenant ID in a variable that is used in further steps. In the next steps, it is recommended to use this PowerShell session to preserve your variables.

4.3.2 Create a Service Principal for Workload Access

Some of the Microsoft 365 workloads also support authentication using application credentials. To use this feature, an app registration associated with a service principal must be created in Microsoft Entra ID, with the correct permissions granted.

Use the following steps to configure your new app registration in Entra ID.

- Create the app registration by running the following PowerShell commands:

Note:

You can use your own app name if needed

```
$DSCAppName = "Microsoft365DSC Deployment"
$DSCApp = New-MgApplication -DisplayName $DSCAppName
```

Note:

You can get the application at a later time with this command:

```
$DSCApp = Get-MgApplication -Filter "displayName eq '$DSCAppName'"
```

- Remove any existing secrets:

```
$DSCApp.PasswordCredentials | ForEach-Object {
    Remove-MgApplicationPassword -ApplicationId $DSCApp.Id -KeyId $_.KeyId
}
```

- Add the previously created certificate to the app:

```
$DSCCertName = "Microsoft365DSC"
$DSCCert = Get-ChildItem -Path "Cert:\LocalMachine\My" | Where-Object {$_.Subject -eq
"CN=$DSCCertName"}
$keyCredential = @{
    DisplayName = $DSCCertName
    EndDateTime = $DSCCert.NotAfter
    Key         = $DSCCert.GetRawCertData()
    Type        = "AsymmetricX509Cert"
    Usage       = "Verify"
}
Update-MgApplication -ApplicationId $DSCApp.Id -KeyCredential $keyCredential
```

- Assign a service principal to the app:

```
$DSCServicePrincipal = New-MgServicePrincipal -AppId $DSCApp.AppId
```

Note:

You can get the service principal at a later time with this command:

```
$DSCServicePrincipal = Get-MgServicePrincipal -Filter "displayName eq '$DSCAppName'"
```

- Display app parameters:

```
Write-Host "--- Microsoft365DSC application ---"
Write-Host "Application name: " $DSCApp.DisplayName
Write-Host "Application ID: " $DSCApp.AppId
Write-Host "Certificate name: " $DSCCertName
Write-Host "Certificate thumbprint: " $DSCCertThumb
```

- Take note of the above details

Note:

Repeat these steps for each environment you are going to manage.

4.3.3 Grant Permissions for the Workload Access Service Principal

To configure a workload, the service principal must have the proper workload permissions present. The next steps describe all possibly required permissions but you may choose to select only a subset of it in your environment.

Some workloads require Microsoft Graph app permissions, some the workload's REST API's app permissions while others Entra ID admin roles.

- First of all, load the following functions into your PowerShell session to support programmatic permission assignment:

```
function New-ServicePrincipalAdminRoleAssignment {
    [cmdletbinding()]
    param
    (
            [Parameter(Mandatory)]
            [System.String]$ServicePrincipalId,
            [Parameter(Mandatory)]
            [System.String]$Role
    )

    if ($null -eq (Get-MgDirectoryRole -Filter "displayName eq '$Role'")) {
            $template = $null
            $template = Get-MgDirectoryRoleTemplate | Where-Object {$_.DisplayName -eq
$Role}
            New-MgDirectoryRole -RoleTemplateId $template.Id
    }
    $AADRole = $null
    $AADRole = Get-MgDirectoryRole -Filter "displayName eq '$Role'"
    $params = @{
            "@odata.id" =
"https://graph.microsoft.com/beta/directoryObjects/$ServicePrincipalId"
    }
    Invoke-MgRestMethod -Method POST -Uri
"https://graph.microsoft.com/beta/directoryRoles/$($AADRole.Id)/members/`$ref" -Body
$params
}
```

```
function Grant-ServicePrincipalAppPermission {
```

```
    [cmdletbinding()]
    param
    (
            [Parameter(Mandatory)]
            [System.String]$ApplicationId,
            [Parameter(Mandatory)]
            [System.String]$ServicePrincipalId,
            [Parameter(Mandatory)]
            [System.Object]$Resource,
            [Parameter(Mandatory)]
            [System.String]$Role
    )

    $resourceSp = Get-MgServicePrincipal -All -Filter "displayName eq '$($Resource)'"
    $resourceAppId = $resourceSp.AppId
    $resourceObjectId = $resourceSp.Id
    $roleId = ($resourceSp.AppRoles | Where-Object {$_.Value -eq $Role}).Id
    $existingResourceAccess = (Get-MgApplication -ApplicationId
$ApplicationId).RequiredResourceAccess
    $reqResourceAccess = @(@{
                    ResourceAccess = @(@{
                                    Id   = $roleId
                                    Type = "Role"
                            })
                    ResourceAppId  = $resourceAppId
            })
    foreach ($resAccess in $existingResourceAccess) {
            if ($resAccess.ResourceAppId -eq $resourceAppId) {
                    $reqResourceAccess[0].ResourceAccess += $resAccess.ResourceAccess
            }
            else {
                    $reqResourceAccess += $resAccess
            }
    }
    Update-MgApplication -ApplicationId $ApplicationId -RequiredResourceAccess
$reqResourceAccess
    New-MgServicePrincipalAppRoleAssignment -ServicePrincipalId $ServicePrincipalId -
PrincipalId $ServicePrincipalId -ResourceId $resourceObjectId -AppRoleId $roleId
}
```

- Configure permissions for Microsoft Entra ID (incl. Conditional Access):

```
# Directory.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role "Directory.ReadWrite.All"

# Policy.Read.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role "Policy.Read.All"

# Policy.ReadWrite.ConditionalAccess
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"Policy.ReadWrite.ConditionalAccess"
```

- Configure permissions for Microsoft Exchange Online (incl. Office 365):

```
# Exchange.ManageAsApp
```

```
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Office 365 Exchange Online" -Role
"Exchange.ManageAsApp"
```

- Configure permissions for Microsoft Intune:

```
# DeviceManagementConfiguration.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"DeviceManagementConfiguration.ReadWrite.All"

# DeviceManagementApps.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"DeviceManagementApps.ReadWrite.All"

# DeviceManagementManagedDevices.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"DeviceManagementManagedDevices.ReadWrite.All"

# DeviceManagementServiceConfig.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"DeviceManagementServiceConfig.ReadWrite.All"

# DeviceManagementRBAC.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role
"DeviceManagementRBAC.ReadWrite.All"
```

- Configure permissions for Microsoft Planner:

```
# Tasks.ReadWrite.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Microsoft Graph" -Role "Tasks.ReadWrite.All"
```

- Configure permissions for Microsoft Power Platform:

```
# Login interactively with a tenant administrator for Power Platform
Add-PowerAppsAccount -Endpoint prod -TenantID $tenantName

# Register a new application, this gives the SPN / client application same permissions
as a tenant admin
New-PowerAppManagementApp -ApplicationId $DSCApp.AppId
```

- Configure permissions for Security and compliance:

```
# Compliance Administrator
New-ServicePrincipalAdminRoleAssignment -ServicePrincipalId $DSCServicePrincipal.Id -
Role "Compliance Administrator"
```

- Configure permissions for Microsoft SharePoint Online and OneDrive:

```
# Sites.FullControl.All
Grant-ServicePrincipalAppPermission -ApplicationId $DSCApp.Id -ServicePrincipalId
$DSCServicePrincipal.Id -Resource "Office 365 SharePoint Online" -Role
"Sites.FullControl.All"
```

- Configure permissions for Microsoft Teams:

```
# Teams Administrator
New-ServicePrincipalAdminRoleAssignment -ServicePrincipalId $DSCServicePrincipal.Id -
Role "Teams Administrator"
```

Note:

Repeat these steps for each environment you are going to manage.

4.3.4 Create a Service Principal for Azure Key Vault and Blob Storage Access

This solution stores all service account and application credentials in Azure Key Vault. There is a one-time upload process, and thereafter, whenever a pipeline in Azure Pipelines runs, it downloads these credentials. Access to Azure Key Vault is provided via an app registration associated with a service principal.

The solution also caches all PowerShell modules that Microsoft365DSC is depending on – including the required version of the Microsoft365DSC module itself – to speed up the pipeline execution times, especially when using Microsoft-hosted agents. The zipped modules are stored in an Azure Blob Storage which is accessed by the same service principal as the Key Vault.

Use the following steps to configure your new app registration in Entra ID.

- Create the app registration by running the following PowerShell commands:

Note:

You can use your own app name if needed

```
$azureAppName = "Microsoft365DSC Azure Access"
$azureApp = New-MgApplication -DisplayName $azureAppName
```

Note:

You can get the application at a later time with this command:

```
$azureApp = Get-MgApplication -Filter "displayName eq '$azureAppName'"
```

- Add a client secret to the app:

Note:

You can change the name and expiration period if needed

```
$passwordCredential = @{
    DisplayName = "Microsoft365DSC-DevOps Service Connection"
    EndDateTime = (Get-Date).AddYears(2)
}
$azureClientSecret = (Add-MgApplicationPassword -ApplicationId $azureApp.Id -
PasswordCredential $passwordCredential).SecretText
```

- Assign a service principal to the app:

```
$azureServicePrincipal = New-MgServicePrincipal -AppId $azureApp.AppId
```

Note:

You can get the service principal at a later time with this command:

```
$azureServicePrincipal = Get-MgServicePrincipal -Filter "displayName eq
'$azureAppName'"
```

- Display app parameters:

```
Write-Host "--- Azure access application ---"
Write-Host "Application name: " $azureApp.DisplayName
Write-Host "Application ID: " $azureApp.AppId
Write-Host "Client secret name: DevOps Pipelines"
Write-Host "Client secret value: " $azureClientSecret
```

- Take note of the above details, and save the client secret value to a safe place

Note:

Repeat these steps for each of your operations centers.

4.3.5 Configure Azure Key Vault

Open the Azure Portal (https://portal.azure.com), and execute the following steps:

- Add role assignment to the subscription:
  - Open **Subscriptions**, and select your subscription
  - In the left navigation pane, click **Access Control (IAM)**
  - Click **Add role assignment**
  - On the **Role** tab, under **Job function roles**, search for and select **Reader**
  - On the **Members** tab, select **User, group, or service principal**, and then search for and select **Microsoft365DSC Azure Access** (or your own application's name that you created before)
  - Click **Review + Assign**
  - Take note of the subscription name and ID and the tenant ID
- Create an Azure Key Vault:
  - Open **Key vaults**, and click **Create**
  - On the **Basics** tab, fill in the following details:
    - **Subscription**: choose your subscription
    - **Resource group**: M365DSC (or you can choose your own resource group name)
    - **Key vault name**: M365DSC-AKV (or you can choose your own name)
    - **Region**: West Europe (or you can choose your own region)
    - **Pricing tier**: Standard
  - On the **Access configuration** tab, fill in the following details:
    - **Permission model**: Vault access policy

        - **Access policies**:
            - Secret permissions: Get, List
            - Certificate permissions: Get, List
            - Principal: Microsoft365DSC Azure Access (or your own application's name that you created before)
    - Click **Review + Create**

 **Note:**

Repeat these steps for each of your operations centers.

4.3.6 Configure Azure Blob Storage

Open the Azure Portal (https://portal.azure.com), and execute the following steps:

- Create a storage account:
    - Open Storage accounts, and click Create
    - On the Basics tab, fill in the following details:
        - **Subscription**: choose your subscription
        - **Resource group**: M365DSC (the same resource group where your Key Vault is)
        - **Storage account name**: m365dscblobstorage (or you can choose your own unique name)
        - **Region**: West Europe (or you can choose your own region)
        - **Performance**: Standard
        - **Redundancy**: Locally-redundant storage (LRS)
    - Click **Review**, and then **Create**
    - When done, open the storage account, and in the left navigation pane, click **Access Control (IAM)**
    - Click **Add role assignment**
    - On the **Role** tab, under **Job function roles**, search for and select **Storage Account Contributor**
    - On the **Members** tab, select **User, group, or service principal**, and then search for and select **Microsoft365DSC Azure Access** (or your own application's name that you created before)
    - Click **Review + Assign**
- Create a new container within the storage account:
    - Open the storage account, and in the left navigation pane, click **Containers**
    - Click **+ Container**, and fill in the following details:
        - **Name**: dependency-modules (or you can choose your own name)
        - **Anonymous access level**: Private (no anonymous access)
    - Click **Create**
    - When done, open the container, and in the left navigation pane, click **Access Control (IAM)**

- Click **Add role assignment**
- On the **Role** tab, under **Job function roles**, search for and select **Storage Blob Data Contributor**
- On the **Members** tab, select **User, group, or service principal**, and then search for and select **Microsoft365DSC Azure Access** (or your own application's name that you created before)
- Click **Review + Assign**

Note:

Repeat these steps for each of your operations centers.

4.3.7 Create a Service Principal for Email Notifications (Optional)

This solution can check the compliance state of your settings and send you notifications about the results either via email or Teams message. Email notifications require an Entra ID app registration associated with a service principal.

Use the following steps to configure your new app registration in Entra ID.

- Create the app registration by running the following PowerShell commands:

Note:

You can use your own app name if needed

```
$emailAppName = "Microsoft365DSC Email Notification"
$emailApp = New-MgApplication -DisplayName $emailAppName
```

Note:

You can get the application at a later time with this command:

```
$emailApp = Get-MgApplication -Filter "displayName eq '$emailAppName'"
```

- Add a client secret to the app:

Note:

You can change the name and expiration period if needed

```
$passwordCredential = @{
    DisplayName = "Microsoft365DSC-DevOps Pipelines"
    EndDateTime = (Get-Date).AddYears(2)
}
$emailClientSecret = (Add-MgApplicationPassword -ApplicationId $emailApp.Id -
PasswordCredential $passwordCredential).SecretText
```

- Assign a service principal to the app:

```
$emailServicePrincipal = New-MgServicePrincipal -AppId $emailApp.AppId
```

Note:

You can get the service principal at a later time with this command:

```
$emailServicePrincipal = Get-MgServicePrincipal -Filter "displayName eq
'$emailAppName'"
```

- Display app parameters:

```
Write-Host "--- Email application ---"
Write-Host "Application name: " $emailApp.DisplayName
Write-Host "Application ID: " $emailApp.AppId
Write-Host "Client secret name: Microsoft365DSC"
Write-Host "Client secret value: " $emailClientSecret
```

- Take note of the above details, and save the client secret value to a safe place

Note:

Repeat these steps for each of your operations centers.

4.3.8 Grant Permissions for the Email Notification Service Principal (Optional)

Compliance email notifications require proper app permissions. The next steps describe how you can programmatically apply these permissions.

- First of all, load the functions already defined in section 4.3.3 into your PowerShell session to support programmatic permission assignment
- Configure permissions for the email notification service principal:

```
# Mail.Send
Grant-ServicePrincipalAppPermission -ApplicationId $emailApp.Id -ServicePrincipalId
$emailServicePrincipal.Id -Resource "Microsoft Graph" -Role "Mail.Send"
```

Note:

Repeat these steps for each of your operations centers.

4.3.9 Create a Teams Incoming Webhook (Optional)

This solution can check the compliance state of your settings and send you notifications about the results either via email or Teams message. Notifications to Microsoft Teams channels require an Incoming Webhook. The webhooks are used as tools to track and notify. The webhooks provide a unique URL, to send a JSON payload with a message in card format.

Refer to this article to configure a new Webhook for your selected Teams channel: [Create an Incoming Webhook - Teams | Microsoft Learn]

4.4 Preparing Azure DevOps (Phase 1)

4.4.1 Create a New Project in Azure DevOps

We need a new project in Azure DevOps for each operations center in which the DSC configurations will be stored and from where the deployments will be executed. These projects can either belong to the same Azure DevOps organization or to their own dedicated organizations.

- Create a new project using the following steps:
  - Log into the Azure DevOps portal
  - Click the **New project** button in the upper-right corner, and fill in the following details:
    - **Project name**: M365Automation (or use your own name)
    - **Visibility**: Private
    - Leave all other settings as default
  - Click **Create**
  - Once the project is created, it is opened automatically
- Configure your project's basic settings:
  - Click **Project settings** in the lower left corner
  - In the **Overview** section, under **Azure DevOps services**, ensure that the following options are selected (in most configurations, it is recommended to de-select all others for simplicity):
    - Repos
    - Pipelines
  - In the **Permissions** section, adjust your project access settings
  - Adjust any other settings that are required by your organization
- Initialize the first repository:
  - In the left pane, click **Repos**
  - Under **Initialize main branch with a README or gitignore**, click **Initialize**
- Configure repository permissions:
  - Click **Project settings** in the lower left corner
  - In the **Repositories** section, select your repository, and then click on the **Security** tab
  - Select the user **<project_name> Build Service (Organization)**, and allow the **Contribute** permission

Note:

Repeat these steps for each of your operations centers.

4.4.2 Create a Service Connection to Azure

For the project to utilize the Key Vault and storage account you created earlier you have to create a service connection in the Azure DevOps project.

Create a new service connection using the following steps:

- Click **Project settings** in the lower left corner
- In the **Service connections** section, click **New service connection**. Click through the wizard using the data provided here:
  - **Choose a service or connection type**: Azure Resource Manager
  - **Authentication method**: Service principal (manual)
  - **Environment**: Azure cloud
  - **Scope Level**: Subscription
  - **Subscription Id**: The subscription id you saved previously, in section 4.3.5
  - **Subscription Name**: The subscription name you saved previously, in section 4.3.5
  - **Service Principal Id**: The *AppId* of the Azure access app you saved previously, in section 4.3.4 (should be in $azureApp.AppId)
  - **Credential**: Service principal key
  - **Service principal key**: The client secret to the Azure access app you saved previously, in section 4.3.4 (should be in $azureClientSecret)
  - **Tenant ID**: The tenant id you saved previously, in section 4.3.5
  - **Service connection name**: AzureConnection
  - Don't check **Grant access permission to all pipelines**
  - Click **Verify and save**

 **Note:**

Repeat these steps for each of your operations centers.

4.5 Preparing Azure DevOps (Phase 2 – for Self-hosted Solution)

4.5.1 Create an Agent Pool in Azure DevOps

The Azure DevOps agents will perform the actual deployment. Each self-hosted agent needs to be placed in its own Agent Pool. In this step, we will create a dedicated Agent Pool for this solution:

- Log into the Azure DevOps portal
- Click **Project settings** in the lower left corner
- In the **Agent pools** section, click the **Add pool** button in the upper right corner
- Fill in the wizard with the following details:
  - **Pool type:** Self-hosted
  - **Name**: Microsoft365DSC (or use your own name)
  - **Description**: Agent pool used for deploying DSC configurations to Microsoft 365 (or use your own description)

- Check **Grant access permission to all pipelines**
- Click **Create**
- Click the newly created pool to open the pool
- Click the **New agent** button to open the required information to add a new agent
- Copy the link under **Download the agent** for use later in this document

Note:

Repeat these steps for each of your operations centers.

4.5.2 Create a Personal Access Token

The Azure DevOps agent needs to be able to connect to Azure DevOps with the correct credentials. It is using a Personal Access Token (PAT) to do this. In this step we will create a new PAT to be used by the Azure DevOps agent:

- Log into the Azure DevOps portal
- Click the user icon in the upper-right corner and select the **Personal access tokens** menu item
- Click **New Token** to create a new token
- Fill in the wizard with the following details:
  - **Name:** DevOpsAgent (or use your own name)
  - **Expiration**: Select your desired expiration date (max. one year)
  - **Scopes**: Custom defined
  - In the bottom, click **Show all scopes**
  - Select **Agent Pools** > **Read & manage**
  - Click **Create** to create the token

Important:

Copy and store the generated token in a secure place. You cannot retrieve the token at a later point in time.

  - Click **Close** to close the wizard. Your token is now created.

Note:

Repeat these steps for each of your operations centers.

4.6 Preparing the Virtual Machine (Phase 2 – for Self-hosted Solution)

4.6.1 Install and Configure the Azure Pipelines Agent on the Virtual Machine

All Azure DevOps agent prerequisites have now been configured. In this step we will install the agent on the virtual machine:

- Install the agent:
  - Connect to your virtual machine with administrative credentials
  - Download the **Azure Pipelines Agent** using the download link from the last step of section 4.5.1.
  - Create a new folder e.g. C:\Agent and extract the downloaded zip to that folder
- Configure the agent:
  - Open an **elevated** Command Prompt
  - Browse to the folder you created and run config.cmd:

```
cd C:\Agent
config.cmd
```

  - Fill in the wizard using the following data:
    - **Enter server URL**: https://dev.azure.com/M365Automation (or your custom organization name) and press [Enter]

Note:

The agent will be unable to register if you specify the organization name including the project name (https://dev.azure.com/<org_name>/<project_name>).

    - **Enter authentication type**: Press [Enter] to use the Personal Access Token for authentication
    - **Enter personal access token**: Paste the Personal Access Token you saved in section 4.5.2 and press [Enter]
    - **Enter agent pool**: Microsoft365DSC (or use the name specified earlier) and press **[Enter]**
    - **Enter agent name**: Press [Enter] to use the server name (or use your own name – max fifteen characters)
    - The agent checks some prerequisites
    - **Enter work folder**: Press [Enter] to use the default work folder

Note:

If prompted, press [Enter] to acknowledge **N** for **Perform an unzip for each step**

    - **Enter run agent as a service**: Y and press [Enter]

    - **Enter User account to use for the service**: Enter the service account you created in section 4.2.4 (use the format ComputerName\AccountName) and press [Enter]
    - **Enter Password for the account**: Enter the service account password
    - The agent is being configured. Press [Enter] to start the service automatically
- Verify the agent service on the virtual machine:
  - Open the local **Services** console, and verify if you have the **Azure Pipelines Agent** running.
- Verify the agent is successfully registered in Azure DevOps:
  - Log into the Azure DevOps portal
  - Click **Project settings** in the lower left corner
  - In the **Agent pools** section, click your custom agent pool
  - Select the **Agents** tab, and validate that your agent is present and is online – the name of the agent will be the host name of your virtual machine

Note:

Repeat these steps for each of your operations centers.

4.7 Configuring Azure DevOps

Now that all prerequisites have been created, we can fully configure the solution in Azure DevOps. You need to do this in each of your operations centers.

4.7.1 Prepare Your Repository

In each operations center, the newly created Azure DevOps project contains a Git repository to which all scripts of this solution must be added. In this step we will upload the scripts of the solution to the repository in Azure DevOps. You can use the deployment workstation, the virtual machine, or your own computer to perform the following steps.

- Download and install Visual Studio Code from https://code.visualstudio.com
- Download and install Git from https://git-scm.com
  - Download the most recent version of Git by clicking the **Download** button
  - Run the downloaded installer and use the default settings
- Before initializing the first sync from VS Code, run the following git commands from the VS Code terminal:

```
git config --global user.email "<your_email_address>"
git config --global user.name "<your_name>"
```

- Download the DSC scripts from https://aka.ms/M365DSCWhitepaper/Scripts

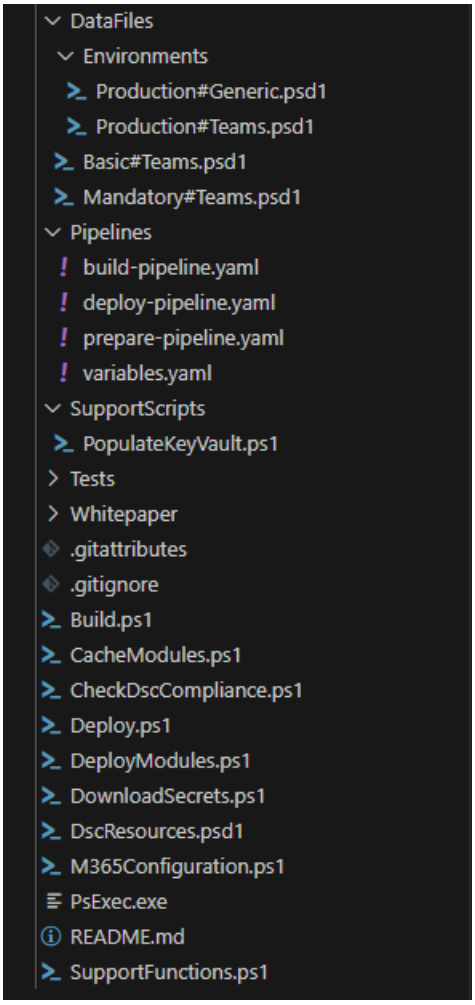
Note:

This package contains several scripts, check chapter 8 for more details

- Clone your DevOps repository:
  - Log into the Azure DevOps portal, and browse to your project
  - In the left pane, click **Repos**
  - Click on the **Clone in VS Code** button (acknowledge any browser notifications for opening any files)
  - Acknowledge that Visual Studio Code can open the external URL by clicking **Open** when prompted
  - Select %USERPROFILE%\Git as the source folder (create it if it does not exist) and click **Select Repository Location**
  - Login with your Microsoft 365 admin account
  - When asked: **Would you like to open the cloned repository**, click **Open**
  - The repository is now available (but still empty) in Visual Studio Code
  - Run the following git command from the VS Code terminal for the repo you're synchronizing:

```
git config core.ignorecase false
```

- Do an initial upload of your locally added content to the DevOps repository:
  - Open Windows Explorer and browse to the **%USERPROFILE%\Git\M365Automation** folder (or use the custom project name specified earlier)
  - Unzip the downloaded script package to this folder
  - Copy the DSCEncryptionCert.cer file that you created in section 4.1.2 or 4.2.2 to the folder
  - You will see the following file listing:



- Click on the Git **Source Control** icon in the left pane, type a commit message (e.g. **Initial upload**), expand the **Commit** button with the arrow on its right, and select **Commit & Sync** to synchronize your local changes with Azure DevOps
- If you get the message that **There are no staged changes to commit**, select **Always**
- Validate a successful sync by opening the Azure DevOps Portal, browsing to Repos and validating that all files have been uploaded

Note:

Repeat these steps for each of your operations centers.

4.7.2 Customize Your Solution

You can and should customize your repository to suit your environmental and operational requirements. Before doing this, think about and plan your operations and configuration definition model based on section 2.1. Once you know your operations staff, your operations centers and which environment will belong to which OC, you can easily do the first steps to customize your solution.

This can be done by editing the following configuration files in your repo using VS Code:

- **DataFiles\Environments\<environment_name>#Generic.psd1** (one file per environment, e.g., Production#Generic.psd1):

```
@{
    AllNodes    = @(
            @{
                    NodeName        = 'localhost' # For Microsoft-hosted solutions, leave
it as 'localhost'; for self-hosted or mixed solutions, enter the 'agent name' you
specified in section 4.6.1 for the given operations center
                    CertificateFile = '.\DSCEncryptionCert.cer' # The file path in the
repo to the encryption certificate of the given operations center
                    CertThumbprint  = '?' # The thumbprint of the encryption certificate
in the given operations center (should be in $encryptionCertThumb)
            }
    )
    NonNodeData = @{
            Environment     = @{
                    Name             = 'Production' # Environment name, e.g.,
'Production'
                    ShortName        = 'PROD' # Environment short name, e.g., 'PROD'
                    TenantId         = '?.onmicrosoft.com' # The name of the tenant that
belongs to the given environment
                    OrganizationName = '?.onmicrosoft.com' # The same as TenantId
            }
            Accounts       = @{} # Leave it empty if you're using service principals
            AppCredentials = @(
                    @{
                            Workload       = 'AzureAD'
                            ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                            CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                    }
                    @{
                            Workload       = 'Exchange'
                            ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                            CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                    }
                    @{
                            Workload       = 'Intune'
                            ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                            CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                    }
                    @{
                            Workload       = 'Office365'
                            ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                            CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                    }
                    @{
                            Workload       = 'OneDrive'
                            ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                            CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
```

```
                }
                @{
                        Workload       = 'Planner'
                        ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                        CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                }
                @{
                        Workload       = 'PowerPlatform'
                        ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                        CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                }
                @{
                        Workload       = 'SecurityCompliance'
                        ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                        CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                }
                @{
                        Workload       = 'SharePoint'
                        ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                        CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                }
                @{
                        Workload       = 'Teams'
                        ApplicationId  = '?' # The AppId of the DSC app for the given
environment (should be in $DSCApp.AppId)
                        CertThumbprint = '?' # The thumbprint of the encryption
certificate for the given environment (should be in $DSCCertThumb)
                }
            )
    }
}
```

- **DscResources.psd1** (enter the latest Microsoft365DSC version number, or any older version you would like to use; in certain situations, you may want to edit the version of the required generic modules – for advanced use cases only)

```
@{
    'M365DSC.CompositeResources' = 'latestMatchingMicrosoft365DSC'
    'M365DSCTools'               = 'latest'
    'Microsoft365DSC'            = '1.23.1129.1'
}
```

- **Pipelines\variables.yaml** (the pipeline configuration file):

  Follow the inline instructions in the

```
# ======== MODIFY VALUES IN THIS SECTION IF NEEDED =========
# ==========================================================
```

section of the file to fill all applicable values. You may need to use some of the previously noted values from this guide. You can also find examples for populating your operations center-specific secrets to Azure Key Vault if needed.

- **Pipelines\test-pipeline.yaml** (the compliance test pipeline)

  The scheduler is disabled by default in this pipeline, so only manual runs are allowed. If you wish to run the compliance checks regularly, uncomment the **schedules** section in the file, and define your own schedule, referring to the cron syntax. E.g., the below code part will schedule a run everyday at a six-hour frequency.

```
schedules:
- cron: "0 0,6,12,18 * * *"
  displayName: "Scheduled export"
  branches:
    include:
    - main
  always: true
```

- **Pipelines*-pipeline.yaml** (all pipeline definitions – five files)

  By default, all pipeline definitions are using Microsoft-hosted VMs. For each pipeline that you want to run on a self-hosted VM, you need to edit the yaml file, and replace the **pool** section, as shown below:

  - Original entry:

```
pool:
  vmImage: windows-latest
```

  - New entry (use the name of your agent pool you specified in section 4.5.1):

```
pool:
  name: <name_of_your_agent_pool>
```

- When done, save all your files
- Click on the Git **Source Control** icon in the left pane, type a commit message (e.g. **Initial customizations**), expand the **Commit** button with the arrow on its right, and select **Commit & Sync** to synchronize your local changes with Azure DevOps

Note:

Repeat these steps for each of your operations centers.

4.7.3 Add Your Secrets to Key Vault

All the secrets and certificates used by the solution need to be added to the Azure Key Vault. Don't store secrets in clear text! The solution contains a script that simplifies this process.

For your environments connected to a given operations center, it reads all used accounts and certificates from the PowerShell data files you updated in the previous section (DataFiles\Environments\<environment_name>#Generic.psd1) and asks for the corresponding passwords. It then adds these to Azure Key Vault, using a specific naming standard.

For your operations centers, it reads the encryption certificate from one of the PowerShell data files and asks for the corresponding password. It also asks you for password secrets if applicable in your setup, like the email application client secret or the PAT token for private package feed access.

In this step we are going to use this script to populate all required Key Vault items for a single operations center and all environments within (e.g., Production).

Note:

If you have multiple environments to be managed, run the same script for each environment with the appropriate parameters after creating the necessary data files.

- Log on / connect to the machine where you cloned your repository
- Open an **elevated** Windows PowerShell window
- Install the Az.KeyVault PowerShell module, if not already present:

```
Install-Module Az.KeyVault
```

- Switch locations to the SupportScripts folder:

```
cd $env:USERPROFILE\Git\M365Automation\SupportScripts
```

- Then run the following commands:
  - To configure operations center-specific secrets:

```
.\PopulateKeyVault.ps1 -OCConfiguration -VaultName <name_of_your_keyvault>
```

  - To configure environment-specific secrets (run once for each of your environments):

```
.\PopulateKeyVault.ps1 -Environment <name_of_your_environment> -VaultName
<name_of_your_keyvault>
```

- The script will read the data file and ask for all passwords and certificates it finds. If a secret is already present in the Key Vault, you are asked if you want to overwrite it or not.

Note:

Repeat these steps for each of your operations centers.

4.7.4 Configure Azure Pipelines

In this section, we will show you how to create and configure your pipelines for running Microsoft365DSC in your previously created Azure DevOps project.

4.7.4.1 Create the Preparation Pipeline

The **Prepare Dependencies** pipeline will only run automatically, when there is a change in the required Microsoft365DSC version (reflected in the DscResources.psd1 file). It then downloads all PowerShell modules that the given DSC version depends on (including Microsoft365DSC itself), compresses these into a zip file, and uploads this package to the previously created Azure Blob Storage.

This is how you can create and immediately run your new preparation pipeline:

- Log into the Azure DevOps portal, and browse to your project

- In the left pane, click **Pipelines**, and then click **Create Pipeline**
- Select **Azure Repos Git**, and then click the name of your project
- Select the **Existing Azure Pipelines YAML file**, and fill in the following data:
  - **Branch**: main
  - **Path**: /Pipelines/prepare-pipeline.yaml
- Click **Continue**
- Now you have the chance to review the yaml file
- In the upper right corner, click **Run** to start the pipeline
- The pipeline is created and started. On the page that is opened, you see the details of the pipeline run. However, it will not yet start.
- If you wait a couple of seconds, the page is refreshed, and you can see that **This pipeline needs permission to access a resource before this run can continue**. So, you first need to provide permissions to the AzureConnection Service Connection.
- Click on the **View** button, and then click **Permit**
- In the dialog that appears, click **Permit** once more
- After a couple of seconds, the pipeline will start running and the jobs are executed
- Check if the pipeline has completed successfully
- When you click the pipeline, you can see the history of all runs
- When you click on a specific run, you can see the logging and other details
- In the left pane, click **Pipelines** again, and then click the three dots at the far right of your new pipeline (the pipeline with the project name)
- From the menu, select **Rename/move**
- Change the **Name** to 'Prepare Dependencies', and click **Save**
- Your pipeline is ready to use now

Note:

Repeat these steps for each of your operations centers.

4.7.4.2 Create the Build Pipeline

The **Build MOF** pipeline will compile the DSC configurations into MOF files and create a deployment package. One MOF file per environment is created. The build pipeline is automatically triggered when there's a change in you resource definition data files under the **DataFiles** folder.

This is how you can create and immediately run your new build pipeline:

- Log into the Azure DevOps portal, and browse to your project
- In the left pane, click **Pipelines**, and then click **New Pipeline** in the upper right corner
- Select **Azure Repos Git**, and then click the name of your project
- Select the **Existing Azure Pipelines YAML file**, and fill in the following data:

  - **Branch**: main
  - **Path**: /Pipelines/build-pipeline.yaml
- Click **Continue**
- Now you have the chance to review the yaml file
- In the upper right corner, click **Run** to start the pipeline
- The pipeline is created and started. On the page that is opened, you see the details of the pipeline run. However, it will not yet start.
- If you wait a couple of seconds, the page is refreshed, and you can see that **This pipeline needs permission to access a resource before this run can continue**. So, you first need to provide permissions to the AzureConnection Service Connection.
- Click on the **View** button, and then click **Permit**
- In the dialog that appears, click **Permit** once more
- After a couple of seconds, the pipeline will start running and the jobs are executed
- Check if the pipeline has completed successfully
- When you click the pipeline, you can see the history of all runs
- When you click on a specific run, you can see the logging and other details
- In the left pane, click **Pipelines** again, and then click the three dots at the far right of your new pipeline (the pipeline with the project name)
- From the menu, select **Rename/move**
- Change the **Name** to 'Build MOF', and click **Save**
- Your pipeline is ready to use now

 **Note:**

Repeat these steps for each of your operations centers.

4.7.4.3 Create the Deployment Pipeline

This solution uses the **Deploy Configurations** pipeline to deploy new configurations to the target environments. The sample code only deploys to one environment (Production), but you can easily add additional environments in the **variables.yaml** configuration file (for more information, refer to section 4.7.2). A new run of the deployment pipeline is automatically triggered after every successful build.

This is how you can create and immediately run your new deployment pipeline:

- Log into the Azure DevOps portal, and browse to your project
- In the left pane, click **Pipelines**, and then click **New Pipeline** in the upper right corner
- Select **Azure Repos Git**, and then click the name of your project
- Select the **Existing Azure Pipelines YAML file**, and fill in the following data:
  - **Branch**: main
  - **Path**: /Pipelines/deploy-pipeline.yaml

- Click **Continue**
- Now you have the chance to review the yaml file
- In the upper right corner, click **Run** to start the pipeline
- The pipeline is created and started. On the page that is opened, you see the details of the pipeline run. However, it will not yet start.
- If you wait a couple of seconds, the page is refreshed, and you can see that **This pipeline needs permission to access 2 resources before this run can continue**. So, you first need to provide permissions to the AzureConnection Service Connection, and then to the environment you ran the deployment against.
- For each of these permission requests, do the following:
  - Click on the **View** button, and then click **Permit**
  - In the dialog that appears, click **Permit** once more
- After a couple of seconds, the pipeline will start running and the jobs are executed
- Check if the pipeline has completed successfully
- When you click the pipeline, you can see the history of all runs
- When you click on a specific run, you can see the logging and other details
- In the left pane, click **Pipelines** again, and then click the three dots at the far right of your new pipeline (the pipeline with the project name)
- From the menu, select **Rename/move**
- Change the **Name** to 'Deploy Configurations', and click **Save**
- Your pipeline is ready to use now

During the first run of the pipeline, one or more new environments have been created in Azure DevOps. To add a deployment approval workflow to these environments, repeat the steps below for each environment you've added:

- Log into the Azure DevOps portal, and browse to your project
- In the left pane, expand **Pipelines**, and then click **Environments**. You can see all your previously configured environments here.
- Open the environment you want to configure, and then select the **Approvals and checks** tab
- Click the **+** sign in the upper right corner, select **Approvals**, and click **Next**
- Enter the following details into the form:
  - **Approvers**: Add the users or groups you would like to designate as approvers
  - **Instructions to approvers**: Fill out if needed
  - Check **Advanced** > **Allow approvers** to approve their own runs
  - **Control options** > **Timeout**: 7 days
- Click **Create**

Note:

You have to repeat these steps whenever you create a new environment in Azure DevOps.

Note:

Repeat these steps for each of your operations centers.

4.7.4.4 Create the Scheduled Compliance Test Pipeline

The solution includes the **Check Compliance** pipeline that can be scheduled to periodically check if the environments are still in the desired state and send a notification of the results to either an email address or a Teams channel. To change the run schedule, refer to section 4.7.2.

This is how you can create and immediately run your new deployment pipeline:

- Log into the Azure DevOps portal, and browse to your project
- In the left pane, click **Pipelines**, and then click **New Pipeline** in the upper right corner
- Select **Azure Repos Git**, and then click the name of your project
- Select the **Existing Azure Pipelines YAML file**, and fill in the following data:
  - **Branch**: main
  - **Path**: /Pipelines/test-pipeline.yaml
- Click **Continue**
- Now you have the chance to review the yaml file
- In the upper right corner, click **Run** to start the pipeline
- The pipeline is created and started. On the page that is opened, you see the details of the pipeline run. However, it will not yet start.
- If you wait a couple of seconds, the page is refreshed, and you can see that This pipeline needs permission to access a resource before this run can continue. So, you first need to provide permissions to the AzureConnection Service Connection.
- Click on the **View** button, and then click **Permit**
- In the dialog that appears, click **Permit** once more
- After a couple of seconds, the pipeline will start running and the jobs are executed
- Check if the pipeline has completed successfully
- When you click the pipeline, you can see the history of all runs
- When you click on a specific run, you can see the logging and other details
- In the left pane, click **Pipelines** again, and then click the three dots at the far right of your new pipeline (the pipeline with the project name)
- From the menu, select **Rename/move**
- Change the **Name** to 'Check Compliance', and click **Save**
- Your pipeline is ready to use now

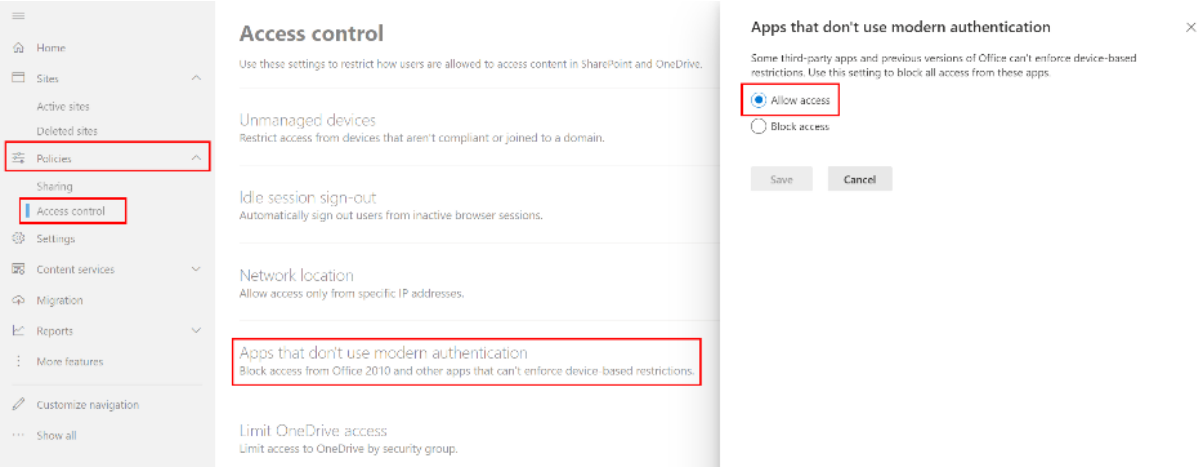
Note:

Repeat these steps for each of your operations centers.

4.7.5 Validate If Configuration Changes Are Deployed Successfully

- Make sure the following setting is configured:

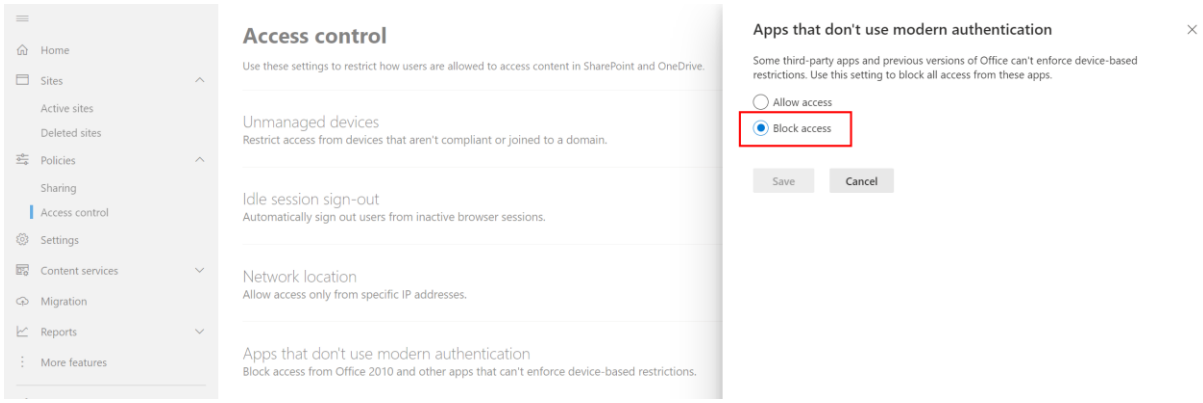
  SharePoint admin center > Policies > Access control > Apps that don't use modern authentication



- The above setting is configured by the **LegacyAuthProtocolsEnabled** DSC setting that can be found in **DataFiles\Environments\Production#SharePoint.psd1** in the repository:

```
SharePoint          = @{
    TenantSettings  = @{
            Ensure                      = "Present";
            LegacyAuthProtocolsEnabled = $True
    }
}
```

- Open the file with VS Code, and change this setting from $True to **$False**
- Save the file, click on the Git **Source Control** icon in the left pane, type a commit message, expand the **Commit** button with the arrow on its right, and select **Commit & Sync** to synchronize your local changes with Azure DevOps
- In the Azure DevOps portal, the **Build MOF** pipeline should have automatically started
- Once completed, the **Deploy Configurations** pipeline should automatically start, as well
- Here, you need to approve the deployment to the given environment
- When the deployment pipeline completes, the setting should have changed in the SharePoint admin center:



Note:

Repeat these steps for each of your operations centers.

5 Create Your Own Configuration Dataset

Once you verified that the solution you configured and customized works, you can create your own composite Microsoft365DSC configuration by using the files below. Here, we provide you with an example of two environments (Production and Development) to manage, but you can extend this further if needed.

First, you should decide if you'd like to use a combined resource definition approach (i.e. a single file contains all your settings) or a split resource definition approach (see section 2.5 for more information).

Note:

If you have to configure a large amount of settings, it can make sense to start right away with the split approach. It can make your settings more manageable.

- **Combined approach** – In a combined approach, you have all your resource definitions in a single file for each configuration level. Your data file structure will look like this:
  - Basic configuration (applies to all environments)
    - DataFiles\Basic.psd1
  - Environment-independent configuration (applies to the <environment_name> environment only)
    - DataFiles\Environments\<environment_name>#Generic.psd1
    - DataFiles\Environments\<environment_name>.psd1
  - Mandatory configuration (applies to all environments)
    - DataFiles\Mandatory.psd1
- Split approach – In a split approach, you have your resource definitions split into multiple files for each configuration level, based on certain criteria. Let's say, the grouping you use is based on workloads, and you configure two workloads: Exchange and SharePoint. In this case, your data file structure will look like this:
  - Basic configuration (applies to all environments)
    - DataFiles\Basic#Exchange.psd1
    - DataFiles\Basic#SharePoint.psd1
  - Environment-independent configuration (applies to the <environment_name> environment only)
    - DataFiles\Environments\<environment_name>#Generic.psd1
    - DataFiles\Environments\<environment_name>#Exchange.psd1
    - DataFiles\Environments\<environment_name>#SharePoint.psd1
  - Mandatory configuration (applies to all environments)
    - DataFiles\Mandatory#Exchange.psd1
    - DataFiles\Mandatory#SharePoint.psd1

The **<environment_name>#Generic.psd1** data file has a specific structure that is described in section 4.7.2.

The resource configuration structure in any other data file should look like this:

```
@{
        NonNodeData = @{
                AzureAD             = @{}
                Exchange            = @{}
                Intune              = @{}
                Office365           = @{}
                OneDrive            = @{}
                Planner             = @{}
                PowerPlatform       = @{}
                SecurityCompliance  = @{}
                SharePoint          = @{
                        <single_instance_resource_name> = @{
                                Ensure   = "String";
                                Setting1 = <value1>;
                                Setting2 = <value2>;
                                Settingn = <valuen>;
                        }
                }
                Teams               = @{
                        <multi_instance_resource_name> = @(
                                @{
                                        Ensure   = "String";
                                        Identity = "<identity1>";
                                        Setting1 = <value1>;
                                        Settingn = <valuen>;
                                }
                                @{
                                        Ensure   = "String";
                                        Identity = "<identity2>";
                                        Setting1 = <value1>;
                                        Settingn = <valuen>;
                                }
                                @{
                                        Ensure   = "String";
                                        Identity = "<identityn>";
                                        Setting1 = <value1>;
                                        Settingn = <valuen>;
                                }
                        )
                }
        }
}
```

For example, this file configures a SharePoint resource and two Teams Events policies:

```
@{
        NonNodeData = @{
                AzureAD             = @{}
                Exchange            = @{}
                Intune              = @{}
                Office365           = @{}
                OneDrive            = @{}
```

```
            Planner            = @{}
            PowerPlatform      = @{}
            SecurityCompliance = @{}
            SharePoint         = @{
                SharingSettings = @{
                    Ensure                             = "Present";
                    NotifyOwnersWhenItemsReshared      = $True;
                    PreventExternalUsersFromResharing = $True;
                    SharingDomainRestrictionMode       = "None";
                }
            }
            Teams              = @{
                EventsPolicies                   = @(
                    @{
                        AllowWebinars   = $True;
                        Ensure          = "Present";
                        Identity        = "HR events policy";
                    }
                    @{
                        AllowWebinars   = $False;
                        Ensure          = "Present";
                        Identity        = "Sales events policy";
                    }
                )
            }
        }
}
```

You can find a complete reference of all resources and settings for a given version of Microsoft365DSC in the **M365DSC.CompositeResources** module. You can get it the following way:

- Install the module on your computer:
  - With local admin privileges:

```
Install-Module -Name M365DSC.CompositeResources -Force
```

  - Without local admin privileges:

```
Install-Module -Name M365DSC.CompositeResources -Scope CurrentUser -Force
```

- Browse to:
  - With local admin privileges:

```
%ProgramFiles%\WindowsPowerShell\Modules\M365DSC.CompositeResources\<version>\
```

  - Without local admin privileges:

```
%USERPROFILE%\Documents\WindowsPowerShell\Modules\M365DSC.CompositeResources\<versi
on>\
```

- Open the **M365ConfigurationDataExample.psd1** file.

6 Troubleshooting

N/A

7 Security Enhancements

7.1 Using Azure Conditional Access to Secure Service Principal (for Self-hosted Solution Only)

Microsoft Entra Conditional Access[2] can be used to prevent the created service principal login into Microsoft 365, except when coming from a specified location / IP address. This feature requires a Workload Identities Premium license in your tenant.

You can find detailed information on how to create such Conditional Access policies here: Microsoft Entra Conditional Access for workload identities | Microsoft Learn.

[2] At least Microsoft Entra ID P1 license is required

8 Package Details

This whitepaper uses a set of pre-created scripts. You can use these scripts as is or tailor them to your own situation. Usually, editing and customizing the configuration and pipeline files can cover all needs (see section 4.7.2), but in special cases, you can touch the scripts, as well. This chapter describes what each item in the package is for.

You can download the script package at: https://aka.ms/M365DSCWhitepaper/Scripts

The package contains the files listed in the below table:

File name	Description
.gitattributes	File used by Git, which specifies how each type of file should be handled. Usually there is no need to update this file.
.gitignore	File used by Git, which specifies all files and folders Git must ignore. Usually there is no need to update this file.
Build.ps1	The script that is responsible for preparing the data files and compiling the DSC MOF files (one file per environment).
CacheModules.ps1	The script that prepares and uploads the required dependency PowerShell modules to the Azure Blob Storage whenever the required Microsoft365DSC version changes in the *DscResources.psd1* file.
CheckDscCompliance.ps1	The script that used by the Check Compliancy pipeline to check all environment on compliance with the desired state and send the results via Email or Teams channel message.
Deploy.ps1	The script that is responsible for deploying the resource settings defined in each DSC MOF file to the corresponding Microsoft 365 environment.
DeployModules.ps1	The script that downloads and deploys the cached dependency PowerShell modules for the required Microsoft365DSC version from Azure Blob Storage.
DownloadSecrets.ps1	The script that is responsible for retrieving the service account or applications credentials from Azure Key Vault.
DscResources.psd1	Data file that specifies the version of Microsoft365DSC to be used. Furthermore, it lists the name and version of the required generic modules. If you want to use a different version of Microsoft365DSC, just update this file. In certain situations, you may want to edit the version of the required generic modules, as well (for advanced use cases only). Possible values for the Value field for each module: ▪ **latest**: Use the latest available version of the module ▪ **latestMatchingMicrosoft365DSC**: Use the latest available version of the module that was created for the specified version of Microsoft365DSC (currently only available for the *M365DSC.CompositeResources* module) ▪ **<version_number>**: Use the specified version of the module ▪ **$null** or '': Do not use the module as a prerequisite
M365Configuration.ps1	The master DSC configuration file that orchestrates the various composite resources and passes the provided credentials/app registration info to those resources.
PsExec.exe	This tool is used to import a certificate into the LOCAL SYSTEM's personal certificate store.

File name	Description
README.md	A project description file in Markdown format. This will be displayed when opening the repository in Azure DevOps.
SupportFunctions.psm1	A repository of PowerShell functions that is used by the other scripts in the repository and that enables the simplification of those scripts. Usually there is no need to update this file.
DataFiles/Mandatory#<grouping>.psd1	PowerShell data files containing resource settings that are mandatory for all environments that are managed through this operations center. Mandatory settings override any other settings defined elsewhere. You can split the resources into multiple data files by using a grouping tag after the # character. The script package contains only two files but this can be extended when required. See chapter 5 for more information.
DataFiles/Basic#<grouping>.psd1	PowerShell data files containing resource settings that act as default for all environments that are managed through this operations center. Basic settings only take effect if the same setting is not defined elsewhere. You can split the resources into multiple data files by using a grouping tag after the # character. The script package contains only two files but this can be extended when required. See chapter 5 for more information.
DataFiles/Environments/<environment_name>#Generic.psd1	PowerShell data files with environment-related, non-workload-specific information for the environment called *<environment_name>*. The script package contains this file for a single environment and you should create a new file for each additional environment you manage from the operations center. See chapter 5 for more information.
DataFiles/Environments/<environment_name>#<grouping>.psd1	PowerShell data files with environment-related resource configurations settings for the environment called *<environment_name>*. Environment-specific settings take precedence over Basic settings but are overwritten by Mandatory settings. You can split the resources into multiple data files by using a grouping tag after the # character. The script package contains two files for a single environment and you should create new file(s) for each additional environment you manage from the operations center. See chapter 5 for more information.
Pipelines/build-pipeline.yaml	This is the definition for the 'Build MOF' pipeline.
Pipelines/deploy-pipeline.yaml	This is the definition for the 'Deploy Configurations' pipeline.
Pipelines/prepare-pipeline.yaml	This is the definition for the 'Prepare Dependencies' pipeline.
Pipelines/test-pipeline.yaml	This is the definition for the 'Check Compliance' pipeline.
Pipelines/variables.yaml	This is the global pipeline configuration file that has both freely editable and protected variable definitions with detailed instructions.
SupportScripts/PopulateKeyVault.ps1	A script that must be run manually and that populates your Azure Key Vault with credentials for the specified environment or operations center before you first run your DevOps pipelines. Usually this is a one-time setup, and there is only need to run it again unless there is a change in your managed environments or credentials.
Tests/**	Quality assurance and data validation test definitions. Do not modify!

9 Learning Materials

9.1 Desired State Configuration

- Microsoft Learn: Getting Started with PowerShell Desired State Configuration (DSC) | Microsoft Learn
- Microsoft Learn: Advanced PowerShell Desired State Configuration (DSC) and Custom Resources | Microsoft Learn
- Desired State Configuration Overview for Engineers: Desired State Configuration Overview for Engineers - PowerShell | Microsoft Learn
- Creating configurations
  - Configurations: DSC Configurations - PowerShell | Microsoft Learn
  - Write, Compile, and Apply a Configuration: Write, Compile, and Apply a Configuration - PowerShell | Microsoft Learn
  - DependsOn: Resource dependencies using DependsOn - PowerShell | Microsoft Learn
  - DSC Resources: DSC Resources - PowerShell | Microsoft Learn
- Using configuration data in DSC
  - Using configuration data - PowerShell | Microsoft Learn
  - Separating configuration and environment data - PowerShell | Microsoft Learn
- Composite resources: Composite resources - Using a DSC configuration as a resource - PowerShell | Microsoft Learn
- Secure the MOF file
  - Securing the MOF File - PowerShell | Microsoft Learn
  - Credentials Options in Configuration Data - PowerShell | Microsoft Learn
- Local Configuration Manager
  - Configuring: Configuring the Local Configuration Manager - PowerShell | Microsoft Learn
  - Push/Pull model: Enacting configurations - PowerShell | Microsoft Learn
- Apply, Get, and Test Configurations on a Node: Apply, Get, and Test Configurations on a Node - PowerShell | Microsoft Learn
- Debugging DSC: Debugging DSC resources - PowerShell | Microsoft Learn

9.2 Microsoft365DSC

- microsoft365dsc.com: Introduction - Microsoft365DSC - Your Cloud Configuration
- Microsoft365DSC promotion video: Microsoft365DSC Promotional Video - YouTube
- GitHub repository: microsoft/Microsoft365DSC: Manages, configures, extracts and monitors Microsoft 365 tenant configurations (github.com)
- Microsoft365DSC YouTube channel: Microsoft365DSC - YouTube

9.3 Git

- Git manual: Git - Book (git-scm.com)
- PluralSight: "How Git Works" (subscription required)
  - https://app.pluralsight.com/library/courses/how-git-works/table-of-contents
- PluralSight: "Mastering Git" (subscription required)
  - https://app.pluralsight.com/library/courses/mastering-git/table-of-contents

10 Acronyms

Acronym	Meaning
CD/CI	Continuous Development / Continuous Integration
DSC	Desired State Configuration
LCM	Local Configuration Manager
MFA	Multi-Factor Authentication
MOF	Managed Object Format
OC	Operations Center
CR	Central Repository
VM	Virtual Machine
VS Code	Visual Studio Code