Microsoft

Managing Microsoft 365 in True DevOps Style with Microsoft365DSC and Azure DevOps

Date

October 24, 2024

Version

v3.0 - Final

Prepared by

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

Contributors

Andras Varga

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	October 24, 2024	Yorick Kuijs Andras Varga	Implemented many new features. Full description of all changes in paragraph 1.3

Contributors

Version	Changes
1.0	Yordan Bechev, Yorick Kuijs
1.0.1	Yorick Kuijs
1.1	Yorick Kuijs, Zaki Semar Shahul
1.2	Yorick Kuijs
1.21	Yorick Kuijs
2.0	Yorick Kuijs, Brian Lalancette, Andi Krüger, Jeffrey Rosen, Andrew Piskai, Dean Sesko and Albert Boland
3.0	Yorick Kuijs, Andras Varga, André Kamman, Kelly den Haan, Harry van den Brink, Robert Koers, Ronald Bode, William Francillette, Thierry Eppner, Jordy Blommaert, Andreas Krüger

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[1]: "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 Setup

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.

1.3 Details about all updates in current version

This version of the whitepaper contains the following changes in comparison to the previous version:

- Separate Data and Scripts Azure DevOps projects
    - The current solution is split into two different projects: Scripts and Data files.
    - This means that you can grant Microsoft 365 administrators access to just the Data project, making sure they cannot change any of the scripts to deploy the changes.

[1] As long as endpoint are made available by the Microsoft 365 Product Group for the component you want to manage, we can create resource to manage that component.

- Multiple levels of data files: Basic (Generic) and Tenant specific
    - In the previous version of the whitepaper, each tenant had its own data files. So, if you have multiple tenants and want to update a setting on each tenant, you have to update all data files.
    - With this change, we have one Basic layer in which all the settings for all tenants live. Each tenant then has its own tenant specific data file. Both files are merged into the actual config that is being deployed to the tenant.

- Mandatory settings check
    - There can be settings that should not be overridden. The basic layer defines these settings and the tenant layer should not override these settings.
    - The new solution will make sure these Mandatory settings are configured and not changed.

- Split data files into workloads
    - To make the data files better readable, we have split the data files into a data file per workload.
    - This is implemented for all layers, Basic, Mandatory and Tenant specific.

- Quality Assurance tests
    - Since we are all human and can make simple mistakes that can cause issues, we have added Quality Assurance tests to test the data for accuracy. If issues are found, the configuration won't be deployed.

- Microsoft Hosted Agent support
    - The previous solution used Self-Hosted agents for various reasons. This did mean you had to install and manage this virtual machine yourself. We received feedback if it would be possible to use Microsoft Hosted Agents.
    - We implemented solutions to mitigate the reasons why we earlier chose for Self-Hosted agents. You can now use Microsoft Hosted Agents if you want (we assume this as the default in this version).

- Tokenizing
    - If you are using group and/or policy names that include tenant specific text, you cannot add these to the basic settings since these aren't generic. That means these specific items have to be added to the tenant specific data file. This can be avoided when you can somehow use a naming standard and tokenize these names.
    - This update includes the possibility to use generic tokens, which then are replaced by tenant specific values. That way you can add the token to the Basic file and by replacing the token during the Build process to make it tenant specific.

- Replacing Credentials with Certificate Thumbprint
    - When the previous whitepaper was released, Teams and Security and Compliance only supported Credential authentication. Since then, these two workloads received support for service principal authentication using certificate thumbprints.
    - The solution now only support service principals so any MFA requirements won't be an issue anymore.

- Generated Composite Resources module that support all workloads
    - The previous version of the whitepaper contained a template of a Composite Resources module that you had to extend and maintain yourself.
    - Right now, we have created a Composite Resource module generator that generates and publishes a new version to the PowerShell Gallery for each version of Microsoft365DSC. This generator also implements support for splatting, which is not possible out-of-the-box with PowerShell DSC. This ensures you can omit parameters when you are not using them, without the code throwing errors.

- Generic functions are implemented via a public module called M365DSCTools, which is available in the PowerShell Gallery (https://www.powershellgallery.com/packages/M365DSCTools)
    - Improvements to that module are now automatically available to everyone
    - You can add new features and fix bug by submitting a pull requests here: https://github.com/ykuijs/M365DSCTools

- Improved troubleshooting information
    - We have updated our logging function, which outputs better information.
    - We have added as much troubleshooting information in the scripts to make troubleshooting issues easier.
    - Since we are merging data files, we are also including a copy of the merged data in the pipeline artifacts. Both a tokenized and non-tokenized version.

1.4 Feedback

If you have any feedback or are running into issues, you can create an issue in the Issue list of the M365DSC_CICD project on GitHub: https://github.com/ykuijs/M365DSC_CICD/issues

All feedback is welcome:

- New feature suggestions
- Technical or documentation issues
- Improvements
- Etc., etc.

2 Solution Description

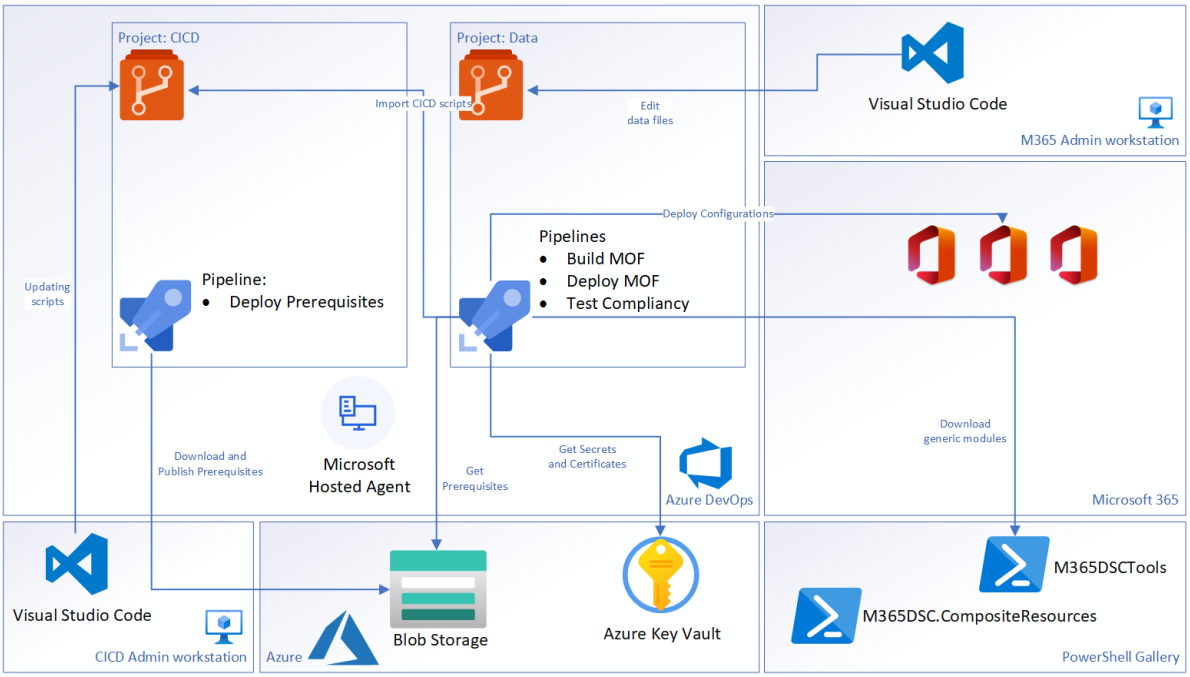
This solution consists of multiple components. In this chapter, the solution is described in more detail.

2.1 Solution Setup

This solutions allows Microsoft 365 to be managed using a Configuration-as-Code approach. In this version, the data files that contain the settings and the logic that is used to deploy the settings are split into separate projects:

- Data files: Data project
- Deployment logic: CICD project

By using this method, you prevent that Microsoft 365 administrators are able to (accidentally) change the deployment logic.



2.1.1 Data Project

The data project contains all data files that are used by the solution. The solution is using a layered configuration:

- Basic layer: These settings apply to all tenants, but can be overridden in a tenant specific data file.
- Tenant layer: These settings apply to an individual tenant.

- Mandatory layer: These settings are mandatory and are not allowed to be overridden. This means that these settings must be configured in the Basic layer and cannot be configured in the Tenant layer. This is enforced by running unit tests over the data files.

To improve readability and make reviews easier, the data files for each layer are split into a file per workload.

To add a new tenant to the solution, a template is provided which can be deployed by running a script. See paragraph 5.1.3, for instructions.

For more information on the data files, see paragraph 5.1.1.

The Data project also contains pipeline definitions, but these are basic files that link to the full files in the CICD project. That way a pipeline can only be updated by the solution administrators.

Both the Microsoft 365 and Solution administrators should get access to this repository.

2.1.1.1 Change reviews

To make sure changes are reviewed before they are merged into the repository, quality gates will be configured. A Pull Request is required to change files in the main branch, a review is required before a Pull Request can be merged and users cannot review their own Pull Request. This way a so-called "four-eyes principle" is enforced.

2.1.2 CICD Project

Microsoft365DSC and all of its prerequisites are required with each deployment. Installing all required modules from scratch can take more than 10 minutes. To speed up that process, we have implemented a solution that caches these modules and places them on an Azure Blob Storage. When a new version of Microsoft365DSC is specified in the CICD repository, a pipeline will automatically make sure the prerequisites are downloaded, packaged and uploaded to the specified Blob Storage.

All certificates that are needed for authenticating against Microsoft 365 are securely stored in an Azure Key Vault. Since the solution uses Azure DevOps Hosted Agents, which are public agents, it is required to configure your Key Vaults with public access. If your organization has the requirement to limit access to Key Vault to private networks only, consider using "Managed DevOps Pools" (for more info, see paragraph 2.3.3.2).

Just the Solution administrators should get access to this repository.

2.1.2.1 Deployment

A deployment of the settings to the tenants consists of several steps:

Build pipeline

When a PR is merged, the Build pipeline is triggered. This pipeline performs these steps:

1. **Prepare the agent**: Install all prerequisite modules and configure settings on the agent.

2. **Validate secrets**: Check if all secrets used in the data files actually exist in the Azure Key Vault.
3. **Pre build**: Run several unit tests making sure all data is valid. This step also merges the data files in the Basic and Tenant layers.
4. **Build**: Compile the merged data files into a MOF file, which can then be deploy
5. **Post build**: This step updates configuration in Azure DevOps making sure the deployment matches the configuration specified in the General files.
6. **Publish artifacts**: The Build process not just generates the MOF files, but also saves the merged data files. These are also included in the artifacts, so they can be used for troubleshooting.

Release pipeline

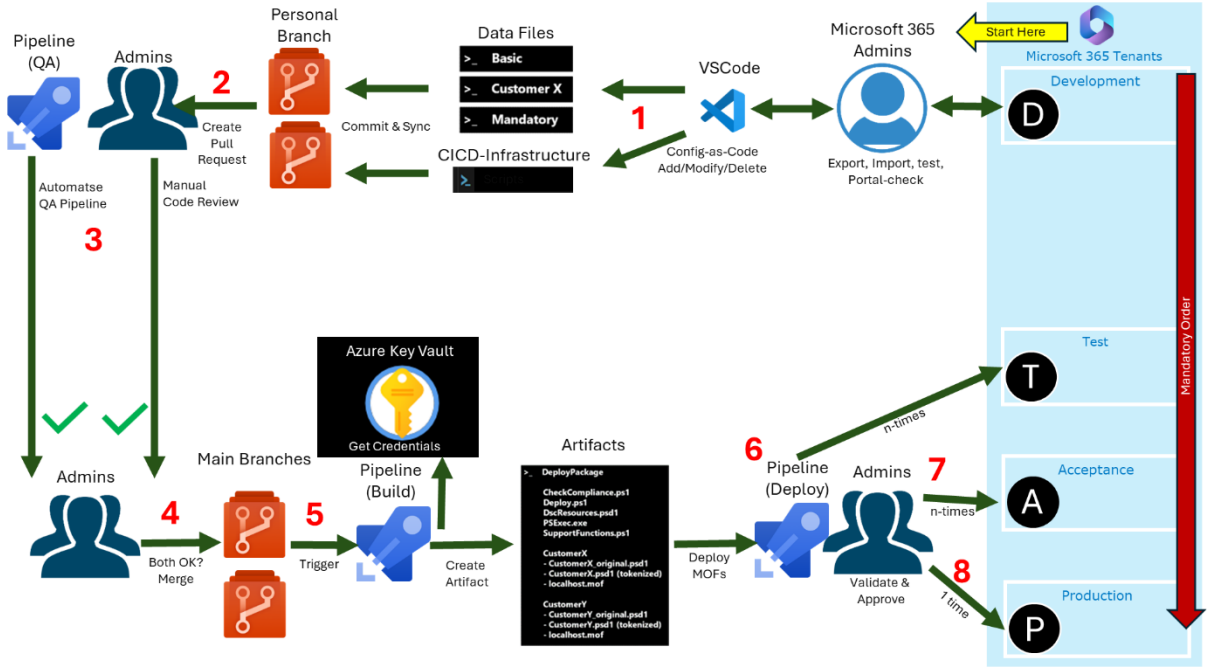
After a successful build, a Release pipeline is triggered. Based on the deployment order defined in the data files, the Release pipeline executes the following steps for each tenant:

1. **Configure the agent**: Configure required settings on the agent.
2. **Prepare modules**: Download all required modules from the Azure Blob Storage.
3. **Download secrets**: Download all required secrets from the Key Vault, so they can be used by the agent.
4. **Deploy configurations**: Deploy the MOF file to the Microsoft 365 tenant.

After a successful deployment, approval has to be provided before the process continues on the next tenant.

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 (branch or fork). This can be either of the following:
   a. Admin1 updates the data files for their own workloads (data files)
   b. Admin2 updates the data files for their own workloads (data files)
   c. The CICD admin updates the scripts in the CICD repository
2. When done, the admin creates a Pull Request to have his changes merged into the main repository at the given location
3. The quality assurance process starts:
   - An automated process runs certain quality checks against the Pull Request
   - Other admins validate the changes via a peer review process
4. When quality checks succeed, the Pull Request is merged
5. The data file merge initiates a build pipeline that retrieves credentials from Azure Key Vault and compiles the so called *MOF files*
6. Once the build pipeline completes successfully, the deployment pipeline starts deploying the configuration to the Test environment
7. Once the deployment to the Test environment completes successfully, a notification is send to admins to validate and approve the deployment. When the admins approve the deployment, the deployment pipeline deploys the generated MOF file to the next environment
8. After a successful deployment, the admins check if the change has been deployed successfully and if the desired result has been achieved, they approve the deployment to further environments
9. The change is now automatically and consistently deployed to all environments

2.3 Azure Components

2.3.1 Azure Key Vault

Azure Key Vault is used to store all service account and application credentials 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 five 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.

  This pipeline is the only pipeline that exists in the CICD DevOps project.

- **PR Validation**

  The main branch of the Data DevOps project deploys the configurations to the various tenants. That means you should protect that branch and prevent users from making changes directly to the main branch. By configuring **Branch Protection** rules, any changes directly to the main branch are prohibited and changes are only allowed via Pull Requests.

  To make sure all submitted changes in these Pull Requests are technically correct, the **PR Validation** pipeline will run several unit tests. These tests will make sure all data files have the correct syntax, all required properties exist, properties have the correct data type, etc.

  A new run of the deployment pipeline is automatically triggered with the creation or update of a pull request.

  This pipeline exists in the Data DevOps project.

- **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 data files, both the tokenized and non-tokenized versions.
  - The deployment script and the compliance check script
  - The DSCResources.psd1 file, to determine which version of Microsoft365DSC 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.

This pipeline exists in the Data DevOps project.

- **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.

  This pipeline exists in the Data DevOps project.

- **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.

  This pipeline exists in the Data DevOps project.

2.3.3.2 Virtual Machines

Any of the above pipelines can use either **Microsoft-hosted** (recommended) 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.

Agent Type	Pros	Cons
Microsoft-Hosted	• No maintenance or upgrade required • Agent has many components default installed • Each run gets a clean new machine • One free agents, fixed costs for each additional agent	• Required to install required non-default components during each job • No control over the default installed components • No possibility to log onto the agents, e.g. for troubleshooting • Does not allow access to internal resources • Uses the Azure global network, configuring Conditional Access Policies allow access from all VMs in that network

Managed DevOps Pools (recently announced)	• All benefits from Microsoft-Hosted agents
	• But with adding more flexibility by running your own pool of agents, which allow:
	  ○ Configuring the use of private IPs and therefore use of Conditional Access Policies
	  ○ Specifying your own VM sizes
		• None

Self-Hosted	• Allows more flexibility • Can run the agent wherever you want: Azure, on-prem, etc. • Allows access to internal resource, by using an on-prem VM • Can log onto the agents, e.g. for troubleshooting • Possible to configure fixed IP, to allow Conditional Access Policies or private endpoints. • Use Managed Identity for authentication	• Additional costs due to VM running constantly. • Responsible for maintenance and upgrade (patching, monitoring, troubleshooting) • Multiple runs can impact each other • Required to harden security yourself: https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/windows-agent?view=azure-devops#information-security-for-self-hosted-agents

Note:

Each DevOps organization have one free Microsoft-Hosted agent included. Unfortunately, because of mis-use of these agents, these are not automatically assigned to a new organization. In order to have this assigned, you now have to submit a request here: https://aka.ms/azpipelines-parallelism-request

2.3.3.3 Environments

Environments correspond to Microsoft 365 tenants. As already described, the deployment and compliance test pipelines use Azure DevOps environments to represent your tenants.

The Post-Build step of this solution manages and configures these environments. It uses the information in the Generic environment file as input. Therefore you have to make sure those files are configured correctly. See paragraph 5.1.4.1 for more information.

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.[2]

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 solution installation, 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 solution installation, in a freely selected tenant

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 solution installation

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.

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

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

- **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.)

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.

2.5.1 Composite Resources

Composite Resources is a feature in PowerShell Desired State Configuration that allows you to define a layer between a PowerShell Configuration (defined in a PS1 file) and DSC Resources, like Microsoft365DSC. By using Composite Resources, you can add additional logic to DSC Configurations or add additional grouping, of which we are using both in this solution.

For example:

- **Adding additional logic**

  By default PowerShell DSC provides the Group and the GroupSet resources. The Group resource allows you to add a user to a local Computer group and the GroupSet allows you to add a user to a set of groups. In this example, the GroupSet resource is a Composite Resource that contains additional logic to iterate through the provides set of groups and call the Group resource for each of them. When reviewing the generated MOF file, you will see that a Group resource is indeed generated for each of the groups in the provides set.

  In this solution we are using the Composite Resources for some additional logic: PowerShell DSC unfortunately does not support Splatting for DSC configurations. Splatting is a method that allows you to dynamically create an hashtable with parameter information and then insert that data into a PowerShell cmdlet. This method makes code much more efficient and easier to read.

  Since Splatting cannot be used out-of-the-box in DSC Configurations, creating flexible and manageable DSC Configurations is impossible. If we want to create two policies where in Policy1 we want to configure Setting1 and Setting2, but for Policy2 we want to configure Setting3 and Setting4, that is not possible. In order to accomplish this scenario, we have to implement templates for all possible combinations of parameters, which can grow very large and become unmanageable quickly. That is where Composite Resources come into play.

  In this solution we are using the DscBuildHelpers module, which offers a function to dynamically generate resources, basically achieving the same as Splatting.

More information about Splatting and Composite Resources can be found in the links in paragraph 10.1.

- **Add Grouping of resources**

  The amount of possible settings for all Microsoft 365 is enormous. When creating one single DSC Configuration and placing all resources for all settings you want to configure in this one configuration, you can imagine that that configuration will become very big very quickly. Second, since all resources are in a single file, it can become difficult to find the correct resource you want to configure. It is possible to agree on grouping resources on workloads, but if someone does not adhere to that agreement it will become an even bigger mess.

  To prevent this from happening, this solution is using a Composite Resource per workload. Each resource contains all possible resources for that specific workload.

And since we have created a generator for the module that contains these Composite Resources (M365DSC.CompositeResources), you do not have to manage any logic or additional resources or settings anymore.

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 enables you to specify the Microsoft365DSC version you want to use for your deployments. This file exists in the CICD project repository, so can only be changed by the CICD admins.

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.

Especially when you cross the Breaking Changes release scheduled. More information on those can be found here: https://microsoft365dsc.com/concepts/breaking-changes/

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

  It's a configuration file for almost all customizable settings. 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.

  This file exists in the CICD project repository, so can only be changed by the CICD admins.

- **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).

- **Environment generic settings files** (DataFiles\Environments\<env_type>\<environment_name>\<environment_name>#Generic.psd1)

  Each environment has its own Generic file. In this file, generic information is configured like Certificate Thumbprint, environment names, tokens, CICD settings, etc. It is therefore important to make sure this file is configured correctly for the environment.

2.7 Tokenization

To allow more flexibility, this solution supports the use of tokens in the data files. This means that you can use tokens which are replaced with a value that you can specify. For example:

You are using an abbreviation of the logical name of your tenant, like TST for Test, ACC for Acceptance and PRD for Production. In your group names, you include the abbreviation. If you then want to use those group in a resource, for example for applying a policy to that group, you can then include the group name in the Basic data files even though the group name is tenant specific.

Instead of specifying the actual group name, you replace the tenant abbreviation with the token:

The group **GRP-TST-Administrators** becomes **GRP-{{TenantAbbreviation}}-Administrators**. Then in the Generic data file for the tenant you specify that the TenantAbbreviation token has to be replaced with the value **TST**. During the MOF compilation, the code then replaces the {{TenantAbbreviation}} token with the specified value, resulting in the tenant specific group name.

For an example on how to use this feature:

In the Basic Azure AD data file, the Forest_Code token is used on line 147:

```
GroupsSettings                          = @{
    AllowGuestsToAccessGroups     = $True
    AllowGuestsToBeGroupOwner     = $False
    AllowToAddGuests              = $True
    EnableGroupCreation           = $False
    EnableMIPLabels               = $True
    Ensure                        = 'Present'
    GuestUsageGuidelinesUrl       = ''
    IsSingleInstance              = 'Yes'
    UsageGuidelinesUrl            = ''
    GroupCreationAllowedGroupName = 'grp-{{Forest_Code}}-rol-g-AddM365Groups'
}
```

https://github.com/ykuijs/M365DSC_Data/blob/11887b38cc2de5e863c05af651725085a9d86245/DataFiles/Templates/Basic/Basic%23AzureAD.psd1#L147

At the same time, this Forest_Code token is defined in the Generic data file of the EnvironmentTemplate, which will be used as a starting point for each new environment:

https://github.com/ykuijs/M365DSC_Data/blob/11887b38cc2de5e863c05af651725085a9d86245/DataFiles/Templates/EnvironmentTemplate/EnvironmentTemplate%23Generic.psd1#L45

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	N/A
Self-hosted	One virtual machine to act as both: ▪ Deployment workstation and ▪ Azure Pipelines agent	
Mixed	One virtual machine 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.

Note:

If you use a Development or MSDN tenant, it is possible that some features of Microsoft 365 are not available and therefore result in errors. Make sure you take this into account.

Note:

For Microsoft 365 different licensing packages can be purchased. If you try to configure components that are not part of your license, you will receive errors during deployment. Make sure you only use components that are part of your license.

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.

3.5 Skills and Experience

Important: This whitepaper describes step-by-step how to implement the solution and how to manage Microsoft 365 using Configuration-as-Code practices. It is however highly recommended that you also have a good experience with Microsoft365DSC, PowerShell Desired State Configuration, Azure DevOps, Git and the DevOps methodology in general. That way you won't just implement this solution as-is, but also understand what it is doing and being able to make changes to tailor the solution to your specific needs.

Please make sure you have the needed skills and experience when implementing this whitepaper. More resources can be found in chapter 10.

4 Deployment

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

- **Common** (without any indication): to be executed for all types of deployments
- **For Microsoft-hosted solution (Recommended)**: 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, or per environment.

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 solution installation.

Note:

It is possible to use an already existing workstation, but preferably use a new and clean Windows installation. That way you known for sure that the machine does not have any conflicting settings, package, etc. configured.

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 Microsoft365DSC, Az.Resources,
Microsoft.Graph.Identity.DirectoryManagement -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:

Important:

For recommendations on the use of Self-Signed certificates, please read paragraph 7.2.

- 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"
```

Important:

Make sure you periodically monitor the validity of the certificate and replace the used certificate when it is about to expire!

Important:

If you are using a central repository to store all certificates used in your organization, make sure these certificates are also added to that repository.

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"}
```

Note:

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

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 encrypting credentials in compiled MOF files. In this section we are going to create this certificate:

Important:

For recommendations on the use of Self-Signed certificates, please read paragraph 7.2.

- 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 "M365DSC" -AsPlainText -Force
Export-PfxCertificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.pfx"
-Password $password
Export-Certificate -Cert $encryptionCert -FilePath "$tempPath\DSCEncryptionCert.cer"
```

Important:

Make sure you periodically monitor the validity of the certificate and replace the used certificate when it is about to expire!

Important:

If you are using a central repository to store all certificates used in your organization, make sure these certificates are also added to that repository.

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 installation of this solution.

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 Az.KeyVault and Az.Accounts modules by executing the following command:

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

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

```
Enable-PSRemoting -Force
```

4.2.2 Configure the Local Configuration Manager

We need an encryption certificate to encrypt the credentials used in the DSC configuration. In this step we are creating this certificate:

Important:

For recommendations on the use of Self-Signed certificates, please read paragraph 7.2.

- Log onto your virtual machine with administrative credentials
- Open an **elevated** PowerShell ISE and run the following command:

```
$certForDSC = New-SelfSignedCertificate -Type DocumentEncryptionCertLegacyCsp -DnsName
'DSCNode Document Encryption' -HashAlgorithm SHA256 -NotAfter (Get-Date).AddYears(10)
```

Note:

This will create a self-signed signing certificate for the Local Configuration Manager to use.
You can also use a certificate created via a Certification Authority.

Important:

If you are using a central repository to store all certificates used in your organization, make sure these certificates are also added to that repository.

- Run the following command and document the value:

```
$certForDSC.Thumbprint
```

- Export the certificate to a CER file (required during the MOF compilation) by running the following command:

```
Export-Certificate -Cert $certForDSC -FilePath C:\DSCEncryptionCert.cer
```

- In the PowerShell window, browse to the folder C:\M365DSC
  - Create this folder if it does not yet exist
- Paste the following code in the white script pane:

```
Configuration ConfigureLCM
{
    Import-DscResource -ModuleName PsDesiredStateConfiguration

    node localhost
    {
        LocalConfigurationManager
        {
            ConfigurationMode = "ApplyOnly"
            CertificateId = $certForDSC.Thumbprint
        }
    }
}

ConfigureLcm
```

- Run the code (press F5 or click the green **Play** icon)

  A prompt will be shown indicating that the localhost.meta.mof has been created. Note the output path and replace the string <output_directory> with it below.

- Run the following command to deploy the Local Configuration Manager config:

```
Set-DscLocalConfigurationManager -Path <output_directory> -Verbose
```

- To validate a successful configuration of the thumbprint, run Get-DscLocalConfigurationManager. The **CertificateID** parameter should now show the Certificate 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
  - Delete the certificate from the certificate store
  - Reimport 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

4.2.3 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

4.2.4 Create the Microsoft365DSC Authentication Certificate

To authenticate against Microsoft 365, we need a certificate. In this step we are going to create a certificate:

Important:

For recommendations on the use of Self-Signed certificates, please read paragraph 7.2.

- Log onto the virtual machine with administrative credentials
- Open an **elevated** Windows PowerShell window
- Create and export a new authentication certificate by running the following PowerShell commands:

Note:

Update the <password> parameter to your own password

```
$clientCert = New-SelfSignedCertificate -Subject "CN=Microsoft365DSC" -
CertStoreLocation "Cert:\LocalMachine\My" -KeyExportPolicy Exportable -KeySpec
Signature

$password = ConvertTo-SecureString -String "<password>" -AsPlainText -Force

Export-PfxCertificate -Cert $clientCert -FilePath C:\M365ClientCert.pfx -Password
$password

Export-Certificate -Cert $clientCert -FilePath C:\M365ClientCert.cer
```

Important:

If you are using a central repository to store all certificates used in your organization, make sure these certificates are also added to that repository.

- Copy the created file **C:\M365ClientCert.cer** and store it for later use
- Run the following command, copy the displayed **Thumbprint** value, and document it for later use

```
$clientCert.Thumbprint
```

Note:

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

4.2.5 Configure and harden virtual machine

When using a Self-Hosted agent, you are responsible for managing and securing the virtual machine. Therefore once you are done with configuring this solution, make sure you also:

- Implement server hardening
  - More information: Deploy an Azure Pipelines agent on Windows - Azure Pipelines | Microsoft Learn
- Monitor the machine for any issues
- Periodically install security updates

- Install and configure anti-virus applications
- Make sure only the correct administrators have access to this machine
- Etc, etc.

4.3 Preparing the Microsoft 365 Tenant

4.3.1 Connect to Microsoft Graph

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

Important:

If you are using Privileged Identity Management (PIM), make sure you activate your Global Admin role before executing these steps!

- 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 "Directory.ReadWrite.All"

$tenantName = (Get-MgDomain | Where-Object {$_.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

Microsoft 365 workloads 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.

To configure a workload, the service principal must have the proper workload permissions present. Some workloads require Microsoft Graph app permissions, some the workload's REST API's app permissions while others Entra ID admin roles.

The next steps describes creating the service principal, assigning the created authentication certificate and granting all required permissions. You may choose to select only a subset of it in your environment.

Note:

To see which permissions are required for which resource, use the [Get-M365CompiledPermissionList] cmdlet or review the documentation of each resource in the [Resources section] on Microsoft365DSC.com.

Important:

You have to execute the below steps in the Azure tenant in which your Microsoft 365 tenant have been created. This doesn't necessarily have to be the same tenant as your Azure resources. Make sure you connect to are connected to that tenant before executing these steps!

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

- First of all, download the following script to your machine:

https://github.com/ykuijs/M365DSC_CICD/blob/main/Supportscripts/CreateServicePrincipals.psm1

- Load the downloaded module into your PowerShell session to support programmatic creation of the service principal:

```
Import-Module <Script_Path>\CreateServicePrincipals.psm1
```

- Create the app registration and grant the correct permissions by running the following PowerShell commands:

Note:

Update the <DSCAppName> parameter to the name of the app that you want to use and the <tempPath> parameter to the path you have exported the M365 Authentication certificate to

```
$DSCAppName = "Microsoft365DSC Deployment"
$tempPath = "$($env:USERPROFILE)\Downloads"
New-M365DSCServicePrincipal -Credential (Get-Credential) -ServicePrincipalName
$DSCAppName -CertificatePath C:\M365ClientCert.cer
```

Note:

The script allows for creation of service principals for a specific workload. If you use the Workload parameter, the script will only grant the required permissions for that workload.

Important:

If during execution of the above step the process keeps waiting for creation of the principal, log onto the Entra ID portal, browse to **Applications** > **App registrations** and find the newly created service principal, click **API Permissions** and click on the **Grant admin consent for <OrgName>** button. This will manually provide consent for the added permissions and unblock the process.

- At the end of the script, the output shows details of the created app and certificate. Take note of these details. These are required in further steps.

Note:

Repeat these steps for each environment and/or workload you are going to manage.

4.3.3 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.

Important:

You have to execute the below steps in the Azure tenant in which you will create the Azure KeyVault and BlobStorage. This doesn't necessarily have to be the same tenant as your Microsoft 365 tenants. Make sure you connect to are connected to that tenant before executing these steps!

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 installation of this solution.

4.3.4 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**: Azure role-based access control (recommended)
- Click **Review + Create**
- Browse to the **Access control (IAM)** menu option and go to the **Role assignments** tab
- Add two new role assignments
  - Role: **Key Vault Reader** and **Key Vault Secrets User**
  - Member: Microsoft365DSC Azure Access (or your own application's name that you created before)

 **Note:**

Repeat these steps for each installation of this solution.

4.3.5 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 installation of this solution.

4.3.6 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 installation of this solution.

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

Compliance email notifications require proper app permissions. Make sure you add the **Microsoft.Graph >Mail.Send** permission via the Entra ID portal to the service principal created in the previous step.

Note:

Repeat these steps for each installation of this solution.

4.3.8 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 new Projects in Azure DevOps

We need two new projects in Azure DevOps in which the DSC configurations will be stored and from where the deployments will be executed.

- Create the new projects using the following steps:

- Log into the Azure DevOps portal
- Create two new projects named M365DSC_Data and M365DSC_CICD (or use your own names) by executing these steps twice:
  - Click the **New project** button in the upper-right corner, and fill in the following details:
    - **Project name**: Use the name specified above
    - **Visibility**: Private
    - Leave all other settings as default
  - Click **Create**
  - Once the project is created, it is opened automatically
  - Go back to your organizations main page by clicking on the Azure DevOps logo in the upper left corner

- 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 to add all users that will work this solution:
    - Add all solution administrators to the **Project Administrators** of both repositories
    - Add all M365 administrators to the **Contributors** group of the M365DSC_Data repository
  - 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**
- For the Data repository, configure the following 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 installation of this solution.

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 both new Azure DevOps project.

Create a new service connection using the following steps:

- Log into the Azure DevOps portal, and browse to your project
- 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 4.3.4
  - **Subscription Name**: The subscription name you saved previously, in section 4.3.4
  - **Service Principal Id**: The *AppId* of the Azure access app you saved previously, in section 4.3.3 (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 4.3.3 (should be in $azureClientSecret)
  - **Tenant ID**: The tenant id you saved previously, in section 4.3.4
  - **Service connection name**: AzureConnection
  - Don't check **Grant access permission to all pipelines**
  - Click **Verify and save**

 **Note:**

Repeat these steps for both newly created DevOps projects.

 **Note:**

Repeat these steps for each installation of this solution.

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 installation of this solution.

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 installation of this solution.

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 4.2.3 (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 installation of this solution.

4.7 Configuring Azure DevOps

Now that all prerequisites have been created, we can fully configure the solution in Azure DevOps.

4.7.1 Prepare Your Repository

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
  - After install, also install the PowerShell extension
- 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 and data files from:
  - https://github.com/ykuijs/M365DSC_CICD
  - https://github.com/ykuijs/M365DSC_Data

Note:

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

Important:

The new default branch for all Git repositories is called "main". This whitepaper assumes that you are using this default branch name. However, we have seen instances where the old default branch name 'master' was used. When that is the case, deployments will fail. So please make sure 'main' is your default branch.

- 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 C:\src as the source folder (create it if it does not exist) and click **Select Repository Location**

    Note:

    Use this default value as source folder or choose your own desired location.
  - Login with your account that has permissions in the Azure DevOps repository
  - 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 **C:\src\M365DSC_Data** folder (or use the custom project name or custom source folder specified earlier)
  - Copy the downloaded content from the M365DSC_Data repository to this folder
  - Browse to the **C:\src\M365DSC_CICD** folder (or use the custom project name or custom source folder specified earlier)
  - Copy the downloaded content from the M365DSC_CICD repository to this folder
  - Copy the DSCEncryptionCert.cer file that you created in section 4.1.2 or 4.2.2 to the folder
  - For both repositories: You will see all files listed as specified in chapter 8

- For both the CICD and Data repositories: 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 installation of this solution.

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. Once you know your operations staff and which environment you want to manage, 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:

- Edit the **Pipelines\testcompliancy.yaml** file to configure the schedule for the compliancy 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 every day at a six-hour frequency.

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

- If you are not using the default "M365DSC_CICD" name for the CICD repository: Edit all files in the **Pipelines** folder and update all "M365DSC_CICD" references into the name you used for the CICD repository.

```
resources:
  repositories:
  - repository: M365DSC_CICD
    type: git
    name: M365DSC_CICD/M365DSC_CICD
    ref: refs/heads/main
```

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

4.7.2.1 Customizing CICD repository

The CICD repository has a few items you can customize:

Used version of Microsoft365DSC

- Edit the **DscResources.psd1** in the root of the repository. Enter the latest M365DSC version number, or any older version.

```
@{
    Microsoft365DSC = '1.23.1115.1'
}
```

Check-out the following page on the PowerShell Gallery for all available versions:

https://www.powershellgallery.com/packages/Microsoft365DSC

Solution variables

- Edit the **Pipelines\variables.yaml** file to configure the settings to be used in the pipelines:

Follow the inline instructions in the following section of the file:

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

Fill all applicable values. You may need to use some of the previously noted values from this guide.

- If you are not using the default "M365DSC_CICD" name for the CICD repository: Edit all files in the **Pipelines** folder and update all "M365DSC_CICD" references into the name you used for the CICD repository.

```
# Clone the CICD repository
- checkout: M365DSC_CICD
  clean: true
  fetchDepth: 1
  persistCredentials: true
  path: ./s/CICD
```

- 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 installation of this solution.

4.7.3 Configure required permissions

In this section, we will show you how to grant permissions to the various object which are required to successfully build and deploy configurations.

4.7.3.1 Grant Data Build Service account access to CICD project

In order to clone the CICD repository successfully, the Build Service account of the Data project needs to have access to the CICD repository.

This is how you can grant these permissions to the Build Service account:

- Log into the Azure DevOps portal, and browse to your **CICD** project
- Click **Project settings** in the lower left corner
- Click the **Permissions** option in the **General** category
- Click the **Contributors** group in the right window and select the **Members** tab
- Click **Add** and type the name of the Build Service account in your Data project, like <Data Project Name> Build Service. For example **M365DSC_Data Build Service**
- Click **Save** to add the account to the group

4.7.3.2 Grant Data Build Service account permissions in Data project

In order to manage the pipeline environments, the Build Service account of the Data project needs to have permissions granted to use the REST API.

This is how you can grant these permissions to the Build Service account:

- Log into the Azure DevOps portal, and browse to your **Data** project
- Click **Project settings** in the lower left corner
- Click the **Permissions** option in the **General** category
- Click the **New group** button in the upper right corner
- Name the group **Build Service Account Permissions** or any other name you want to use
- Add the Build Service account of your Data project, like <Data Project Name> Build Service to the Members field. For example **M365DSC_Data Build Service**
- Add a useful name to the **Description** field
- Click **Create**
- Click **Pipelines** in the left menu and then click **Environments**

- Create a temporary environment by clicking the **Create environment** button
  - Name: Temp
  - Click **Create**
- Go back to the environments page by clicking the back arrow to the left of the newly created environment
- Click the three vertical dots in the upper right corner and select **Security**
- Click **Add** and search for the newly created group **Build Service Account Permissions** or any other name you have used.
- Select **Administrator** as Role and click **Add**
- Click **Save** to store all changes
- Click the temporary environment, click the three vertical dots in the upper right corner and select **Delete**
- Then click **Delete** to delete this environment

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 projects.

4.7.4.1 Create the Preparation Pipeline (in CICD project)

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 **CICD** 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.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 the pipeline has completed successfully, you can check the folder in Azure Blob Storage in which a file called **M365DSC-<version>.zip** should have been created.
- 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 installation of this solution.

4.7.4.2 Create the Build Pipeline (in Data project)

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. It is created in the Data project.

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

- Log into the Azure DevOps portal, and browse to your Data 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.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 CICD repository.
- 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
- In the dialog that appears, click **Permit** once more
- Do this for both items that require permission
- 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 installation of this solution.

4.7.4.3 Create the Deployment Pipeline (in the Data project)

This solution uses the **Deploy Configurations** pipeline to deploy new configurations to the target environments. The sample code does not contain any environments yet, so you have to create one before you can deploy a configuration. For more information, refer to section 5.1.3.

A new run of the deployment pipeline is automatically triggered after every successful build. The pipeline is created in the Data project.

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

- Log into the Azure DevOps portal, and browse to your Data 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/deployment.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 3 resources before this run can continue**. So, you first need to provide permissions to the AzureConnection Service Connection, the CICD repository, 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
- To actually execute, the deployment needs approval. Click **View** and then click **Approve**
- The pipeline will start executing the deployment
- 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

 **Note:**

Repeat these steps for each installation of this solution.

4.7.4.4 Create the Pull Request validation Pipeline (in the Data project)

The solution includes the **PR Validation** pipeline that is ran every time a Pull Request is submitted. It validates if all changes in the pull request are valid.

This is how you can create and immediately run your new PR Validation 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/prvalidation.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 CICD repository.
- 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
- 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 'PR Validation', and click **Save**
- Your pipeline is ready to use now

 **Note:**

Repeat these steps for each installation of this solution.

4.7.4.5 Create the Scheduled Compliance Test Pipeline (in the Data project)

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 compliance test 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 2 resources before this run can continue**. So, you first need to provide permissions to the AzureConnection Service Connection, and then to the CICD repository.
- 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
- 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 installation of this solution.

4.7.5 Configure Branch Policies

In this section, we will show you how to configure branch policies to protect the **main** branch in your previously created **Data** Azure DevOps projects.

To configure branch policies, perform the following steps:

- Log into the Azure DevOps portal, and browse to your **Data** project
- In the left pane, click **Repos**, and then click **Branches**
- Hover over the **main** branch, click the three vertical dots that appear at the end of the line and select **Branch policies**
- Enable the setting **Require a minimum number of reviewers** and configure the following settings:
  - Minimum number of reviewers: **1** (or whatever value that is required in your organization)
  - Allow requestors to approve their own changes: **Disabled** (can be temporarily enabled for testing purposes)
  - Prohibit the most recent pusher from approving their own changes: **Disabled**
  - Allow completion even if some reviewers vote to wait or reject: **Disabled**

  - When new changes are pushed: **Reset all approval votes (does not reset votes to reject or wait)**
- Enable the setting **Check for comment resolution** and select the **Optional** value
- Under **Build Validation** click the + icon in the upper right corner of this section and enter the following values:
  - Build pipeline: **M365DSC PR Validation** (or your own name of the PR Validation pipeline)
  - Path filter (optional): **/DataFiles/***
  - Trigger: **Automatic (whenever the source branch is updated)**
  - Policy requirement: **Required**
  - Build expiration: **After 12 hours**
  - Display name: **PR Validation**
- Click **Save** to save the configured settings
- Make sure the toggle in front of the created Build validation is enabled
- (Optional) If you want to include specific approvers, click on the + icon in the **Automatically included reviewers** section and add the required reviewers, either are Required or Optional.

5 Create Your Own Configuration Dataset

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

5.1.1 How the data files work

When reviewing the Data repository, you see a folder called **DataFiles**. This folder contains the following folder structure:

Environments		Folder that contains all data files for all managed Microsoft 365 environments
	01_Test	A folder that contains all data files for all Test Microsoft 365 environments
	02_Acceptance	A folder that contains all data files for all Acceptance Microsoft 365 environments
	03_Production	A folder that contains all data files for all Production Microsoft 365 environments
Templates		Folder that contains all data file templates. There are three different templates.
	Basic	All data files that contain settings which apply to all environments. A so-called Baseline
	EnvironmentTemplate	A template folder structure for new environments. This is used by the provisioning script to create new environments in the Environments folder.
	Mandatory	All data files that contain settings that are mandatory for all environments. Settings in this file **must** be present in the Basic file and **cannot** be specified in any of the Environment specific files. These two requirement are validated using unit tests.

All folders in Environments contain placeholder files. These have to be populated with environment specific files, which we are going to do in one of the following steps.

5.1.2 Configuring the Basic settings

Before we are going to create a new environment, we first need to configure the Basic settings that apply to all environments. To do that, take the following steps:

- Open Visual Studio Code and open the **M365DSC_Data** repository.

- Browse to the **DataFiles\Templates\Basic** folder and make changes to the settings in the workload data files as desired.

The resource configuration structure should look like this:

```
@{
    NonNodeData = @{
        <workload>         = @{
            <single_instance_resource_name> = @{
                Ensure   = "String"
                Setting1 = <value1>
                Setting2 = <value2>
                SettingN = <valueN>
            }
            <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:

```
@{
    NonNodeData = @{
        SharePoint         = @{
            SharingSettings = @{
                Ensure                              = "Present"
                NotifyOwnersWhenItemsReshared       = $True
                PreventExternalUsersFromResharing = $True
                SharingDomainRestrictionMode        = "None"
            }
        }
    }
}
```

For example, this file configures two Teams Events policies:

```
@{
    NonNodeData = @{
        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.

To export the example data file to a location of your choice, take the following steps:

- Install the **M365DSC.CompositeResources** module
- Run the command `New-M365DSCExampleDataFile -OutputPath <folder>`

5.1.3 Creating a new environment

To create a new environment, take the following steps:

- Log onto your development machine where you have cloned the Data repository
- Open an elevated PowerShell window
- Browse to **C:\src\M365DSC_Data** (or the location you have cloned the repository in step 4.7.1)
- Browse to the folder **DataFiles\Templates**
- Run the command `.\ProvisionNewEnvironment.ps1`
- Type the name of your new environment and press **[ENTER]**

   Note: Use names and letters only
- Type the number of the type of environment you are creating, for example 1 for Test, and press **[ENTER]**
- The new environment is now created. A folder with the specified environment name should be created in the environment type folder and it should contain data files for each workload.

5.1.4 Configuring the new environment

Now that you have created a new environment, you have to update this new environment with the correct information.

- Open Visual Studio Code and open the **M365DSC_Data** repository.

5.1.4.1 Customizing Generic environment information

- Open the file marked Generic in the newly created environment folder, for example: **DataFiles\Environments\<env_type>\<environment_name>\<environment_name>#Generic.psd1** (e.g., Production#Generic.psd1):
- Update all items marked below to reflect the information from your environment

```
@{
    AllNodes    = @(
        @{
            NodeName        = 'localhost'
            CertificateFile = '.\DSCEncryptionCert.cer'
            CertThumbprint  = '<EncryptionCertificateThumbprint>'
        }
    )
    NonNodeData = @{
        Environment    = @{
            Name             = '{{Tenant_Name}}'
            ShortName        = '{{Tenant_ShortName}}'
            TenantId         = '{{TenantId}}'
            OrganizationName = '{{Tenant_Name}}'
            UsedWorkloads    = @{
                AzureAD            = $true
                Exchange           = $true
                Intune             = $true
                Office365          = $true
                OneDrive           = $true
                Planner            = $true
                PowerPlatform      = $true
                SecurityCompliance = $true
                SharePoint         = $true
                Teams              = $true
            }
            CICD             = @{
                DependsOn     = '<dependsOnEnvironment>'
                UseCodeBranch = 'main'
                Approvers     = @(
                    @{
                        Principal = 'test.user@domain.com'
                        Type      = 'User'
                    }
                    @{
                        Principal = 'test.group@domain.com'
                        Type      = 'Group'
                    }
                )
            }
            Tokens           = @{
```

```
            TenantGuid       = '3788f651-cd16-4482-b823-05c62208bc4b'
            Tenant_ShortName = 'TST'
            Tenant_Name      = 'TestEnv'

            Forest_Code      = 'TST'
            TenantId         = 'testenv.onmicrosoft.com'
        }
    }
    AppCredentials = @(
        @{
            Workload       = 'AzureAD'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'Exchange'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'Intune'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'Office365'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'OneDrive'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'Planner'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'PowerPlatform'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'SecurityCompliance'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'SharePoint'
            ApplicationId  = '<appid>'
            CertThumbprint = '<certThumprint>'
        }
        @{
            Workload       = 'Teams'
```

```
                ApplicationId  = '<appid>'
                CertThumbprint = '<certThumprint>'
            }
        )
    }
}
```

5.1.4.2 Customizing workload information

Each workload has its own data file per environment. This file can be found at the following location:

DataFiles\Environments\<env_type>\<environment_name>\<environment_name>#<workload>.psd1

5.1.5 Add Your Secrets to Key Vault

All the secrets and certificates used by the solution need to be added to the Azure Key Vault. The solution contains a script that simplifies this process. It reads all used accounts and certificates from the PowerShell data file you updated in the previous step (DataFiles\Environments\Production#Generic.psd1) and asks for the corresponding passwords. It then adds these to Azure Key Vault, using a specific naming standard.

In this step we are going to use this script to populate all required Key Vault items for a single environment (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, and run the following command:

```
cd C:\src\M365DSC_CICD\SupportScripts
.\PrepareKeyVault.ps1 -VaultName <name_of_your_keyvault> -DataFileFolder
<full_path_to_the_datafiles_you_want_to_use>
```

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

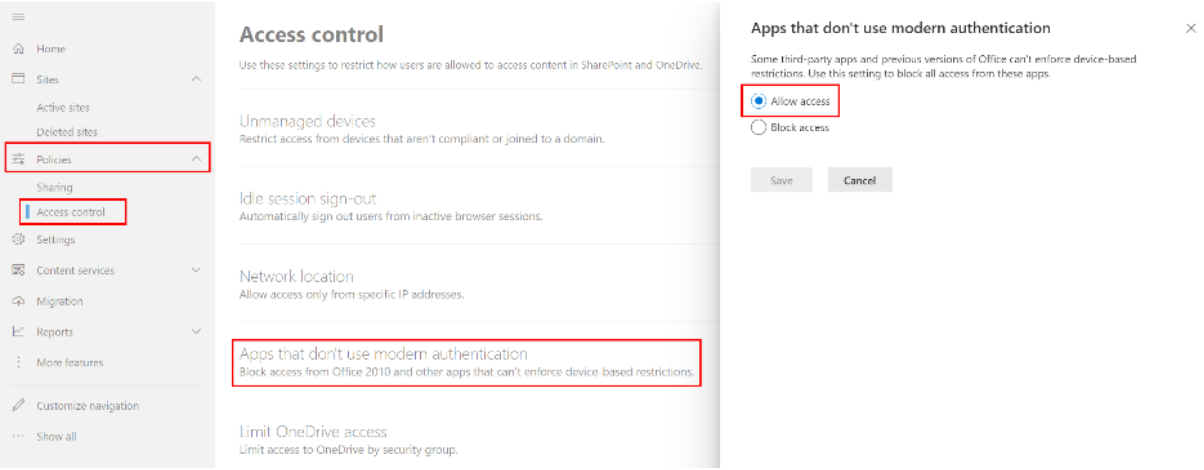
Note:

Repeat these steps for each installation of this solution.

5.1.6 Validate If Configuration Changes Are Deployed Successfully

- Make sure the following setting is configured:

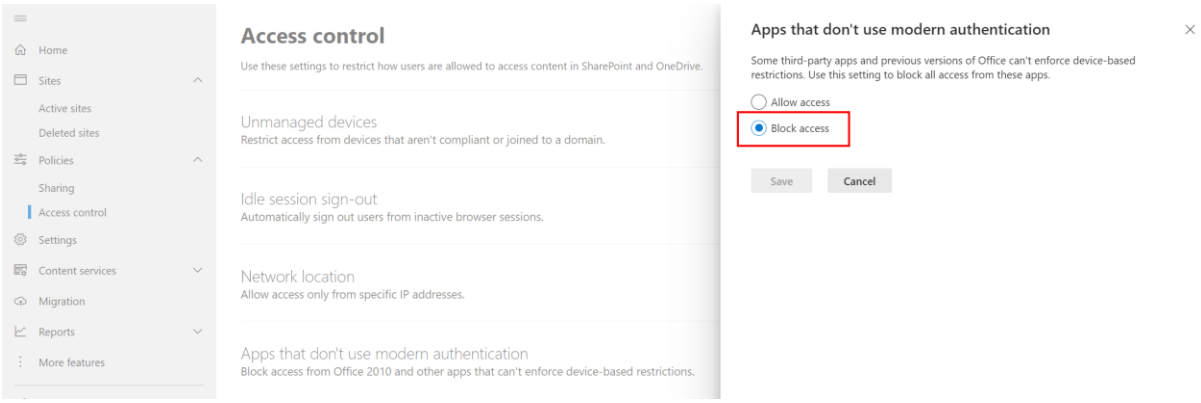
  SharePoint admin center > Policies > Access control > Apps that don't use modern authentication



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

```
SharePoint          = @{
    TenantSettings  = @{
            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 installation of this solution.

6 Troubleshooting

N/A

7 Security Enhancements

7.1 Using Azure Conditional Access to Secure Service Principal (for Self-Hosted Solution or Managed DevOps Pools Only)

Microsoft Entra Conditional Access[3] 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.

7.2 Self-Signed certificates or certificates created by a Certificate Authority

The steps for creating or requesting a certificate can be different in each organization. This is because an organization can use different products for Certificate Authorities, have different procedures for requesting certificates, have different requirements for certificates, etc.

This whitepaper therefore describes the use of Self-Signed certificates. Please only use these Self-Signed certificates in Test environments only and use a certificate issued by a Certificate Authority managed by your organization for Production environments.

See this article for more information: Create a self-signed public certificate to authenticate your application - Microsoft identity platform | Microsoft Learn

[3] 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 section describes what each item in the package is for.

8.1 CICD script repository

The template for the CICD script repository can be found on GitHub, location is described in paragraph 9.3. The following files are included in the template:

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.
DscResources.psd1	Data file that specifies the version of Microsoft365DSC to be used. If you want to use a different version of Microsoft365DSC, just update this file.
PsExec.exe	This tool is used to import a certificate into the LOCAL SYSTEM's personal certificate store.
ReadMe.md	Readme file in Markdown format. This file describes the details of the project, including changelog. It is displayed on the main page when opening the repository in Azure DevOps.
.vscode**	All files with settings for Visual Studio Code
Pipelines\build-pipeline.yaml	This is the template definition for the 'Build MOF' pipeline.
Pipelines\deploy-pipeline.yaml	This is the template definition for the 'Deploy Configurations' pipeline.
Pipelines\prepare.yaml	This is the definition for the 'Prepare Dependencies' pipeline.
Pipelines\prvalidation-template.yaml	This is the template definition for the 'PR Validation' pipeline.
Pipelines\testcompliancy-template.yaml	This is the template 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.
Scripts\Build.ps1	The script that is responsible for compiling the DSC MOF files (one file per environment).
Scripts\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.
Scripts\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.
Scripts\Deploy.ps1	The script that is responsible for deploying the resource settings defined in each DSC MOF file to the corresponding Microsoft 365 environment.

File name	Description
Scripts\DeployModules.ps1	The script that downloads and deploys the cached dependency PowerShell modules for the required Microsoft365DSC version from Azure Blob Storage.
Scripts\DownloadSecrets.ps1	The script that is responsible for retrieving the service account or applications credentials from Azure Key Vault.
Scripts\M365Configuration.ps1	The master DSC configuration file that orchestrates the various composite resources and passes the provided credentials/app registration info to those resources.
Scripts\PostBuild.ps1	The script that is responsible for preparing the Azure DevOps configuration once a Build process completes successfully.
Scripts\PreBuild.ps1	The script that is responsible for checking the validity of all data files and merging the various file into one environment specific file (one file per environment). This new file is used by the Build script/
Scripts\SupportFunctions.ps1	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.
Scripts\ValidateSecrets.ps1	The script that is responsible for validating if the required applications credentials exist in the Azure Key Vault.
SupportScripts\CreateServicePrincipals.psm1	A script that is used to create service principals in Entra ID, with the correct permissions. This script is used in this whitepaper.
SupportScripts\PopulateKeyVault.ps1	A script that must be run manually and that populates your Azure Key Vault with credentials for the specified environment 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!

8.2 Data files repository

The template for the Data files repository can be found on GitHub, location is described in paragraph 9.4. The following files are included in the template:

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.
ReadMe.md	Readme file in Markdown format. This file describes the details of the project, including changelog. It is displayed on the main page when opening the repository in Azure DevOps.
.vscode**	All files with settings for Visual Studio Code

File name	Description
DataFiles\Templates\Mandatory\Mandatory#<grouping>.psd1	PowerShell data files containing resource settings that are mandatory for all environments that are managed. 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 solution contains only one file but can be extended when required. See section 1.2 for more info.
DataFiles\Templates\Basic\Basic#<grouping>.psd1	PowerShell data files containing resource settings that act as default for all environments that are managed. 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 solution contains only one file but can be extended when required. See section 1.2 for more info.
DataFiles\Templates\EnvironmentTemplate\EnvironmentTemplate#<grouping>.psd1	PowerShell data files containing resource settings that act as a template for new environments. You can create new environments by executing the ProvisionNewEnvironment.ps1 script, which is using these files as the starting point. See paragraph 5.1.3 for more information.
DataFiles\Environments\<env_type>\<environment_name>\<environment_name>#Generic.psd1	PowerShell data files with environment-related, non-workload-specific information for the environment called *<environment_name>*. The solution contains only one file and you should create a new file for each additional environment you manage.
DataFiles\Environments\<env_type>\<environment_name>\<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 and tested against the Mandatory settings. You can split the resources into multiple data files by using a grouping tag after the # character. The solution contains only one file and you should create a new file for each additional environment you manage.
Pipelines/build.yaml	This is the definition for the 'Build MOF' pipeline.
Pipelines/deployment.yaml	This is the definition for the 'Deploy Configurations' pipeline.
Pipelines/prvalidation.yaml	This is the definition for the 'PR Validation' pipeline.
Pipelines/testcompliancy.yaml	This is the definition for the 'Check Compliance' pipeline.

9 Links

9.1 M365DSCTools

Module with several generic functions used in the scripts of this solution. This allows a bugfix or new feature to become available for everyone quickly.

Project site : https://github.com/ykuijs/M365DSCTools

PowerShell Gallery : https://www.powershellgallery.com/packages/M365DSCTools

9.2 M365DSC.CompositeResources

Module that is generated based on Microsoft365DSC and which contains Composite Resources for each Microsoft 365 workload. Using this module you can simply insert configuration data, which then is used to compile the MOF file.

Project site : https://github.com/ykuijs/M365DSC.CompositeResources

PowerShell Gallery : https://www.powershellgallery.com/packages/M365DSCTools

9.3 M365DSC CICD template

The template repository for all scripts that are used in this whitepaper.

Project site : https://github.com/ykuijs/M365DSC_CICD

9.4 M365DSC Data template

The template repository for all data files that are used in this whitepaper.

Project site : https://github.com/ykuijs/M365DSC_Data

10 Learning Materials

10.1 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
  - Splatting: About Splatting - 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

10.2 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

10.3 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

11 Acronyms

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