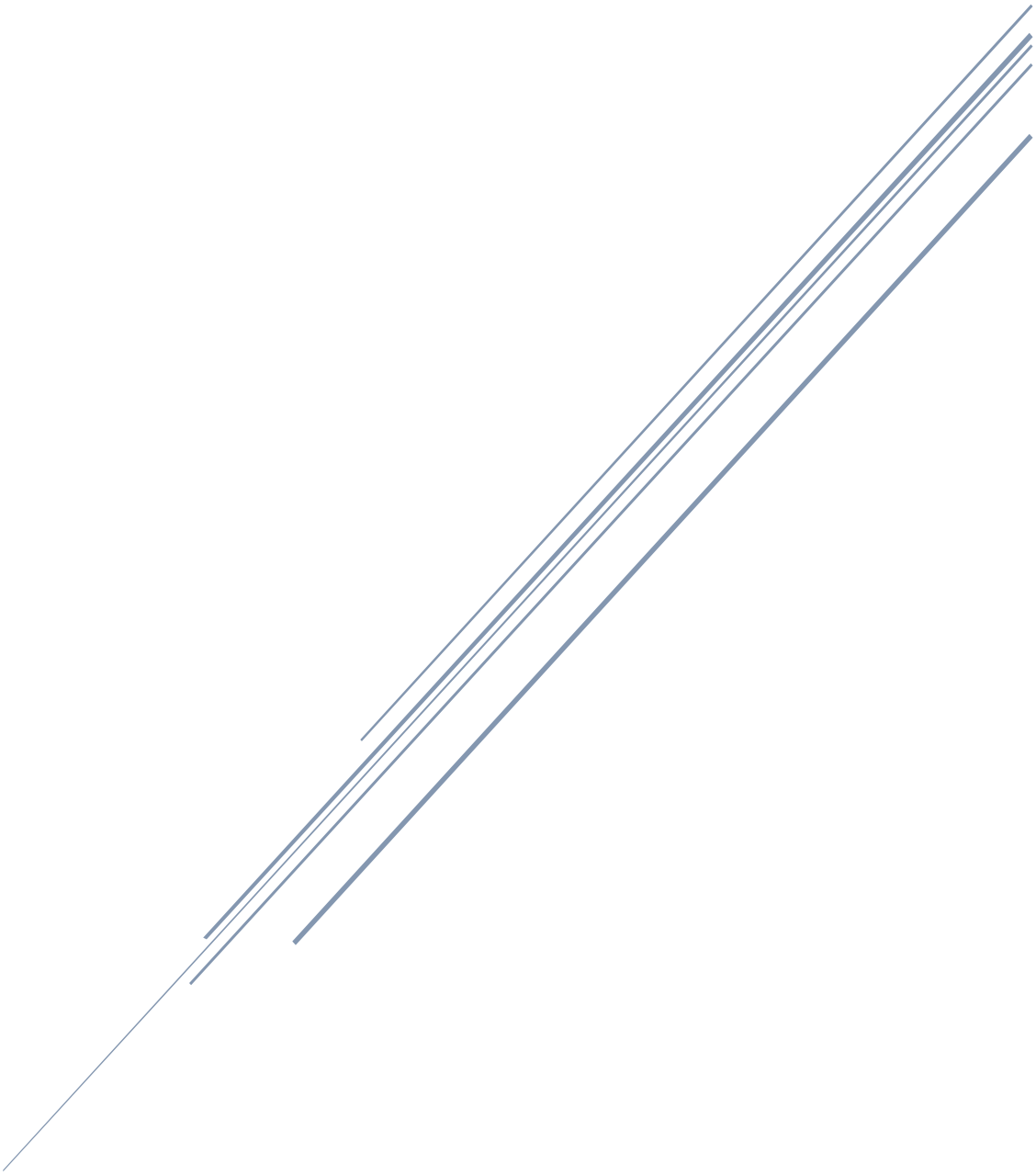
# MODERN LAPS

A custom solution to manage local administrator password rotation across Azure AD and Active Directory joined devices, utilizing Microsoft Endpoint Configuration Manager and Microsoft Azure



Trevor Jones
smsagent.blog

# Table of Contents

# Overview

Modern LAPS is custom solution for managing password rotation for local administrator accounts. It has no dependency on Active Directory or even corporate network connectivity, so it can be used to manage the local administrator passwords for both Azure AD joined and Active Directory joined devices. The solution utilizes existing technologies in your environment including Microsoft Endpoint Configuration Manager as well as services from Microsoft Azure in the cloud. It also includes a custom application for secure retrieval of passwords.

This document contains a step-by-step guide for how to set up the required components in Configuration Manager and Azure.

Although this solution is intended to be configured as described, the source code for the scripts used as well as the custom app is available in GitHub and these, together with the Azure and Configuration Manager components can be customised or reworked to fit your own environment and requirements, or simply used as a reference for developing your own solution.

# Features

- Local administrator account password rotation across any Azure AD or Active Directory joined device
- Secure password storage and retrieval
- Azure-AD authentication and RBAC security for Azure components
- Encryption key rotation supported
- SAS key rotation supported
- Client-side logging
- Custom app for password retrieval and manual password rotation
- Password history (useful for the scenario where a system restore is performed and the local administrator password is reverted to a previous one)

# Requirements

- Microsoft Endpoint Configuration Manager. All devices targeted by this solution must be using MECM
- An Azure subscription
- Azure Active Directory authentication – this can be cloud or federated
- Windows 10 workstations
- Familiarity with Configuration Manager administration
- Familiarity with Azure administration

# Components

The following components are used in this solution:

- Microsoft Endpoint Configuration Manager Configuration Baselines and Collections
- Azure Active Directory
- Azure Event Grid Service
- Azure Function App
- Azure SQL Database (serverless)
- Azure System Managed Identity
- Azure Storage Account
- Azure Key Vault

- Azure Enterprise Applications
- Azure App Registrations

## Cost

There is a cost associated with the Azure services, but for the most part this is relatively low. The big exception is the Azure SQL Database. Using the Azure Price Calculator and excluding Azure Active Directory, the monthly estimate for a basic configuration is around US $120.

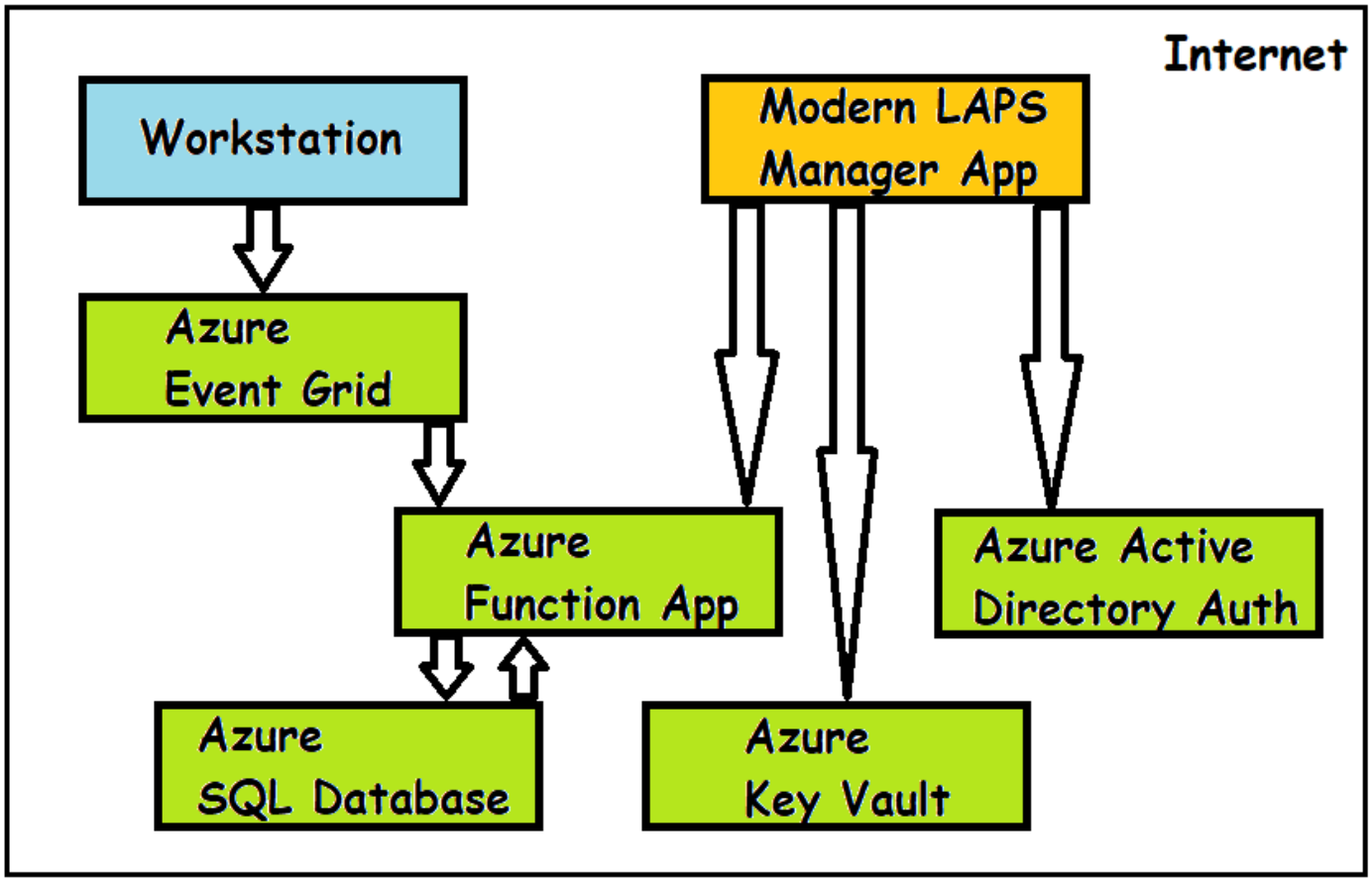
## Support

I am unable to provide support for this solution. Please use it at your own risk. I highly recommend testing it in a lab and understanding the various components before implementing it in a production environment.

## How it Works – The Short Version

The solution uses a Configuration Baseline in Configuration Manager to regularly run a PowerShell script on client workstations. This script periodically rotates the local administrator account password with a strong password, encrypts it and hands it off to the Azure Event Grid service with an http request. The Event Grid passes the event to an Azure Function via an event subscription. The Azure Function sends the password change details to an Azure SQL Database.

To retrieve a password, I created a custom standalone Windows application. The app authenticates you with Azure AD, retrieves decryption keys from Azure Key Vault, retrieves local password data from the Azure SQL Database via the Azure Function app then displays the local administrator password for any managed workstation.



# How it Works – The Long Version

## Password Rotation Script

Using a Configuration Baseline in Configuration Manager, a PowerShell script is run on a schedule on a managed workstation. Using a frequency that you decide, the script will rotate the password for the local administrator account, even if the account has been renamed. The password is a strong password and you can specify the length and complexity of the password with a regular expression. Once the password has been successfully changed, the script will encrypt the password with a 256-bit AES symmetric encryption key before attempting to send it to the Azure Event Grid service with an http request using TLS1.2.

If the encrypted password cannot be sent at that time, it will be temporarily stamped to the registry as an encrypted string. Each time the baseline evaluates it will attempt to send the password again and once successful, will delete it from the registry.

The actions of the script are logged both to the registry and to the application event log.

## Azure Event Grid

I chose to use the Azure Event Grid service to receive the password change event details to add some resilience and efficiency into the process of sending the password to the SQL database. The Event Grid service receives the event which is then sent to an Azure function via an event subscription. Sending to the Event Grid first avoids any delay in waiting for the Azure function to execute, cold start or scale and means the password rotation script can execute very quickly in just a couple of seconds or less. The Event Grid can handle large numbers of events and will retry sending the event for up to 24 hours if it cannot be received immediately by the Azure function. It can also handle so-called 'dead lettering', where any failed events can be placed in Azure storage for review.

## Azure Function

An Azure function is used to send the password change details to the Azure SQL database. The function receives the event from the event grid subscription, decrypts the password, encrypts the password again using a different key, then runs an SQL query to add the details the database. The Azure function uses a system managed identity to access the database so no credentials need to be stored in code.

## Azure SQL Database

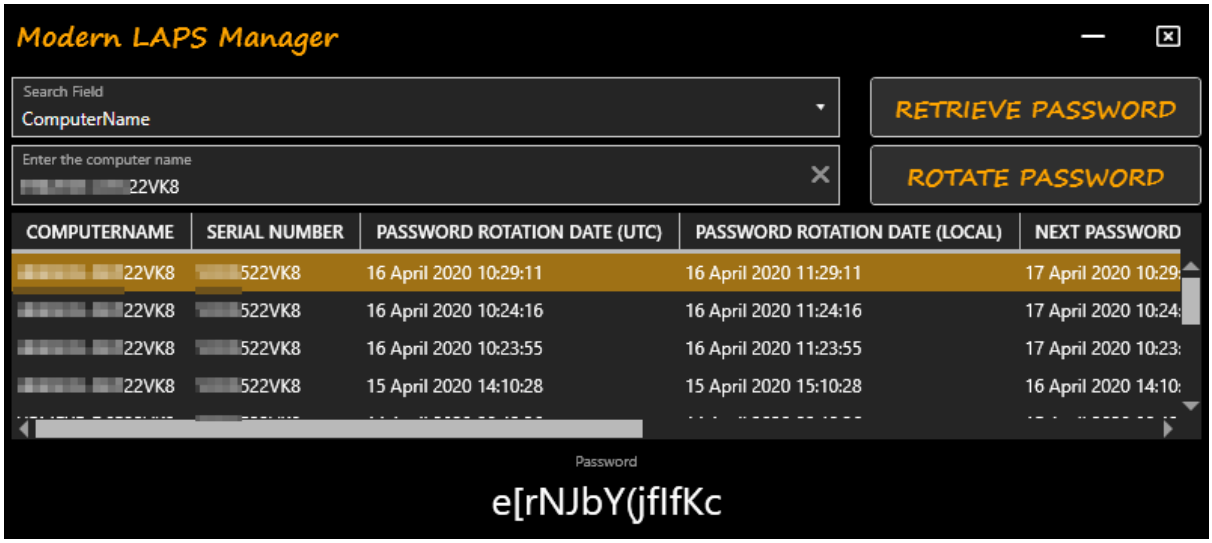
A serverless Azure SQL database is used to store the password data. The endpoint of the SQL database does not need to be exposed publicly or even to any Azure vNet as only Azure services are used to communicate with it. Azure SQL database supports "Transparent data encryption" which encrypts the database, backups and logs with a service-managed key. Access to the database is controlled by Azure AD group membership.

## Password Retrieval

To retrieve a password, the custom app should be used. The app contains the code to perform the necessary actions.

Here's how it works:

The user must first be a member of the Azure AD group that grants permissions to the relevant resources in Azure. The user runs the application, enters a computer name or serial number and clicks "Retrieve password". The user will then be prompted to authenticate with Azure AD using the usual Sign-in window. Once authenticated, authentication tokens will be obtained for the Azure Key Vault and Azure SQL Database audiences. The app will connect to the Key Vault and retrieve the encryption key used to encrypt the local administrator password. It will then send an http request using TLS1.2 to an Azure function, passing the auth token for the SQL database. The function is responsible for authenticating with the user token against the database and running the query to return the data. After the client app receives the data, it displays it and decrypts the local administrator password.




## Manual Password Rotation

The app also supports manually rotating the password on an online client. For this to work, the user must be an administrator on the target workstation or provide credentials that are, and be able to establish both a CIM session and a PowerShell session on the default ports. The app will connect to the remote workstation and rotate the password by triggering an evaluation of the compliance baseline. It is also possible to manually rotate the password using the Run Script feature in Configuration Manager which bypasses the above requirements.

# Security

Creating a completely secure, unhackable solution is extremely difficult, however I have put particular effort into making this solution as secure as possible.

- Where used, encryption keys are 256-bit AES encryption keys using CBC mode.
- The Configuration baseline that runs on the client does contain an encryption key to encrypt the local administrator password during transit, however this key is used only for transit. Before adding the password to the SQL database, it is encrypted again with a different key that is only known to the Azure function. This key is also stored in the Azure Key Vault.
- When the Configuration baseline runs, the PowerShell script is temporarily created in a location that only a local administrator has access to and is removed again immediately after execution (which only take a couple of seconds), so there is no permanent cache of the script on the client.
- The Configuration baseline script is stored in Configuration Manager, so you must be using RBAC to secure access to the Configuration Manager console, as well as ensuring the ConfigMgr SQL database is secured appropriately.
- When the client sends the password to the Event Grid service, it uses TLS1.2. The password itself is encrypted and not transmitted in plain text to protect it from packet sniffers.
- If the password cannot be uploaded immediately it is temporarily stored in the registry until it can be uploaded. The password is doubly encrypted - once using the encryption key and again by converting it to an encrypted string.
- The function app that sends password data to the SQL database stores encryption keys and the SQL connection string in an encrypted form both at rest and during transmission. No encryption keys are exposed in the function code.
- The function app also communicates with the SQL database as a system assigned managed identity. This means that no credentials are stored in code.

- The SQL database is encrypted using transparent data encryption which encrypts the database, backups and logs. Access to the database is only via the Function app managed identity, or via membership of an Azure AD group. The SQL database is not accessible publicly unless you add a public IP range firewall rule to the SQL server. Azure SQL database also has some nice security features including vulnerability assessments and Advanced Threat Protection.
- Encryption keys are stored in Azure Key Vault. Access to the vault is controlled by Azure AD group membership.
- The client application can only access the vault and the SQL database if the user is successfully authenticated and can obtain the required tokens. Communication to Azure endpoints uses TLS1.2. The client app does not keep encryption keys or sensitive data anywhere on disk, only in memory.
- The endpoints for the Azure Key Vault and its secrets are stored in the current user registry hive as encrypted strings. These can only be decrypted on the same machine and by the same account that encrypted them.

To enhance security, I recommend the following:

- Rotate the encryption keys periodically. See the Operations guide for more detail.
- Rotate the access key used for the Event Grid endpoint periodically. See the Operations guide for more detail.
- Rotate the Function key periodically. See the Operations guide for more detail.
- Review the built-in security recommendations for the Azure SQL database and set baselines.
- Enable auditing on the Azure SQL server.
- Review the Azure AD group memberships periodically.
- Review RBAC to the relevant Azure services periodically.

# Sections

The rest of this document contains the following four sections:

1. **Setup guide**. This contains step-by-step instructions for creating and configuring the components of this solution.
2. **Operations guide**. This contains information on using the Modern LAPS Manager app to retrieve and rotate passwords as well as info on client-side logging.
3. **Maintenance guide**. This contains information on how to maintain the solution, for example performing key rotations.
4. **Monitoring guide**. This focuses on things you can do to monitor the solution including the Azure resources and some tips for reporting.

**Important: if you are already using Microsoft LAPS or another solution to manage local administrator passwords, be sure to disable that before deploying this solution to avoid conflict.**

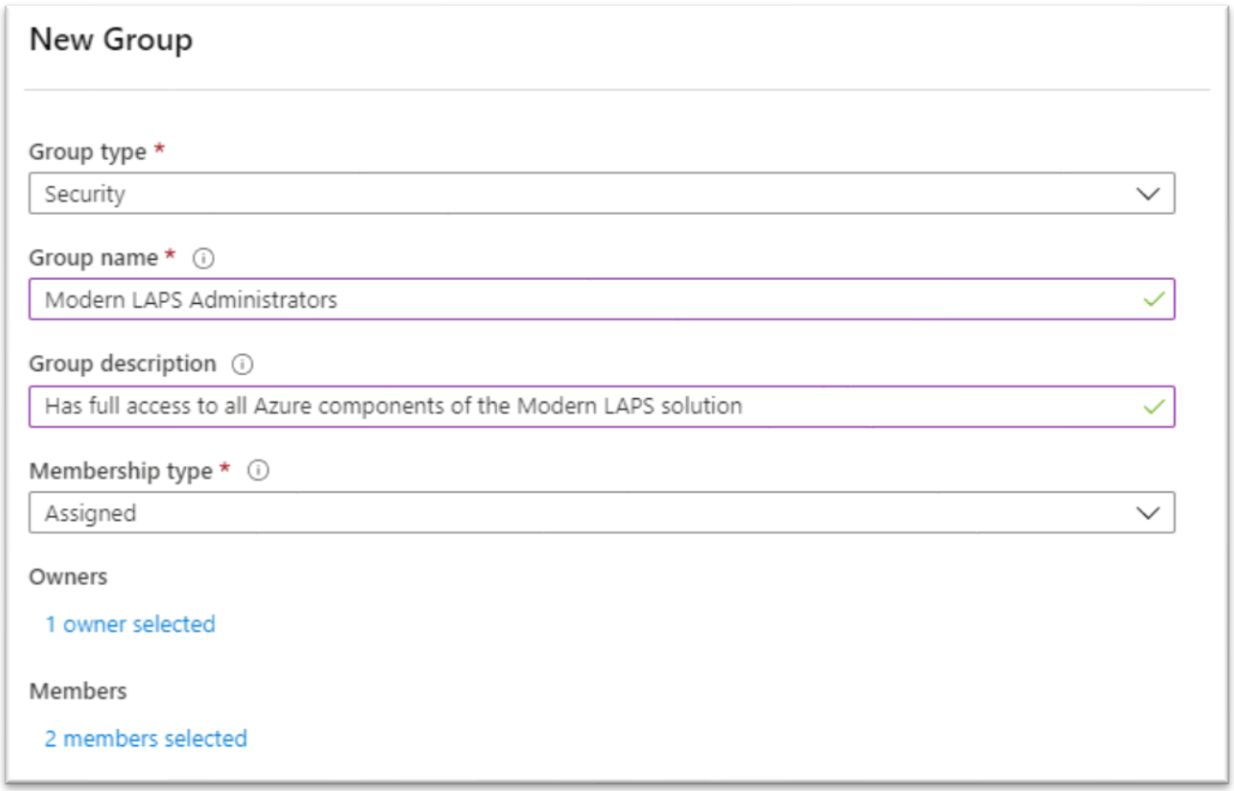
# Setup Guide

In this section, we cover the actions required to configure the Azure services, Configuration Manager baselines, and packaging the client app.

## Create Active Directory Groups

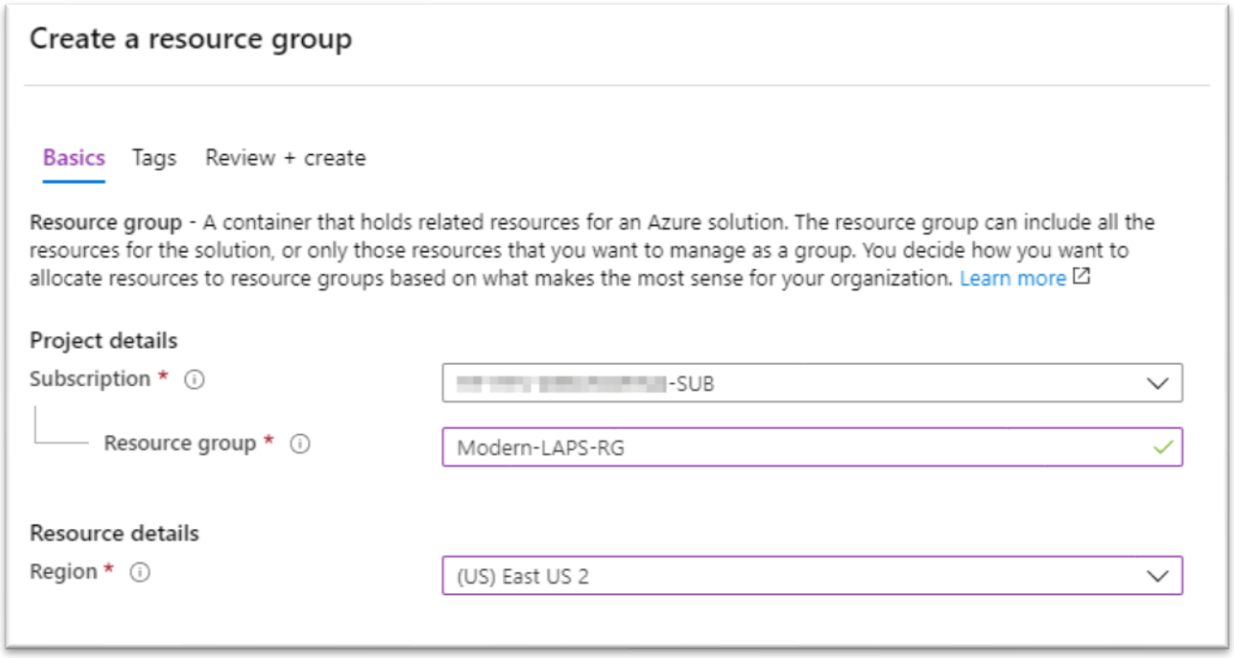
In the Azure portal, create 2 Azure AD Security groups as follows:

| Name | Role |
|---|---|
| **Modern LAPS Administrators** | Has full access to all Azure components of the Modern LAPS solution. Cannot retrieve passwords using the custom app. |
| **Modern LAPS Users** | Has read access to the relevant Azure components of the Modern LAPS solution. Can retrieve local administrator passwords using the custom app. |



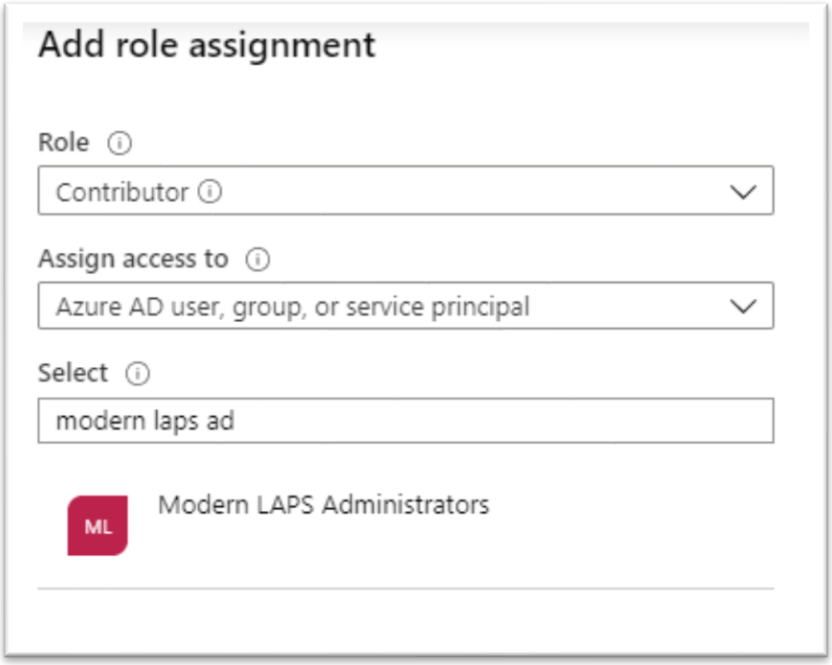

## Create a Resource Group

This is optional but strongly recommended. Create a resource group in your Azure subscription to contains the components of this solution.



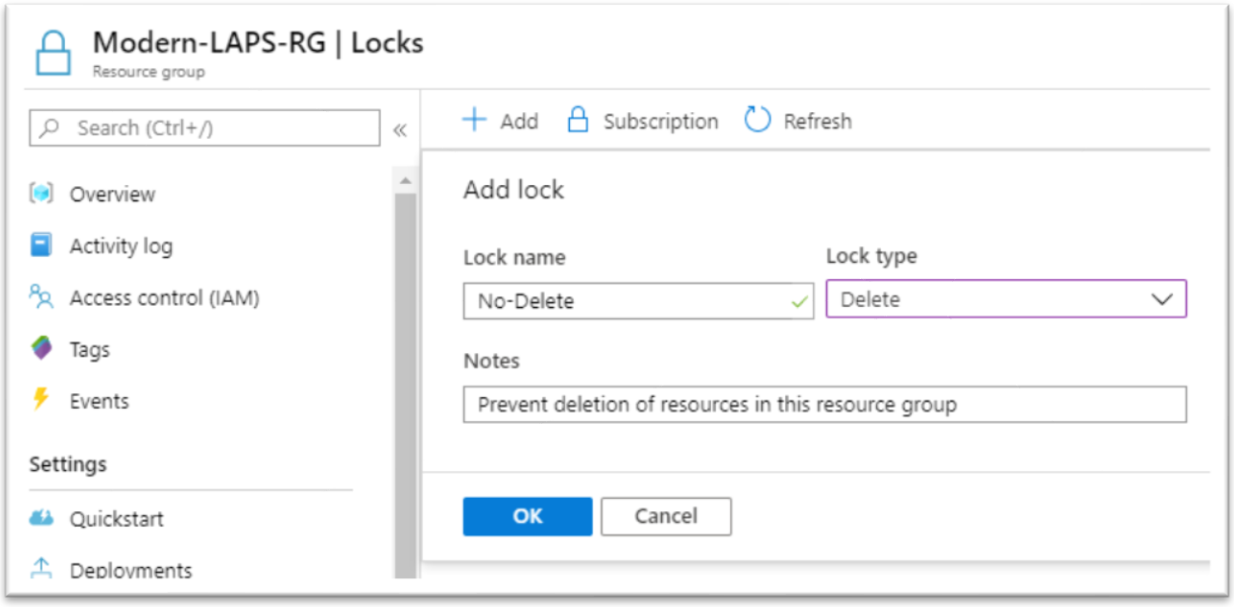

## Assign Role Permissions

In the **Access control** pane, click **Add** to add a role assignment. Add the Azure AD group **Modern LAPS Administrators** to the **Contributor** role. This grants them permissions to manage all resources in this solution in the resource group.




## Optional: Add a Deletion Lock

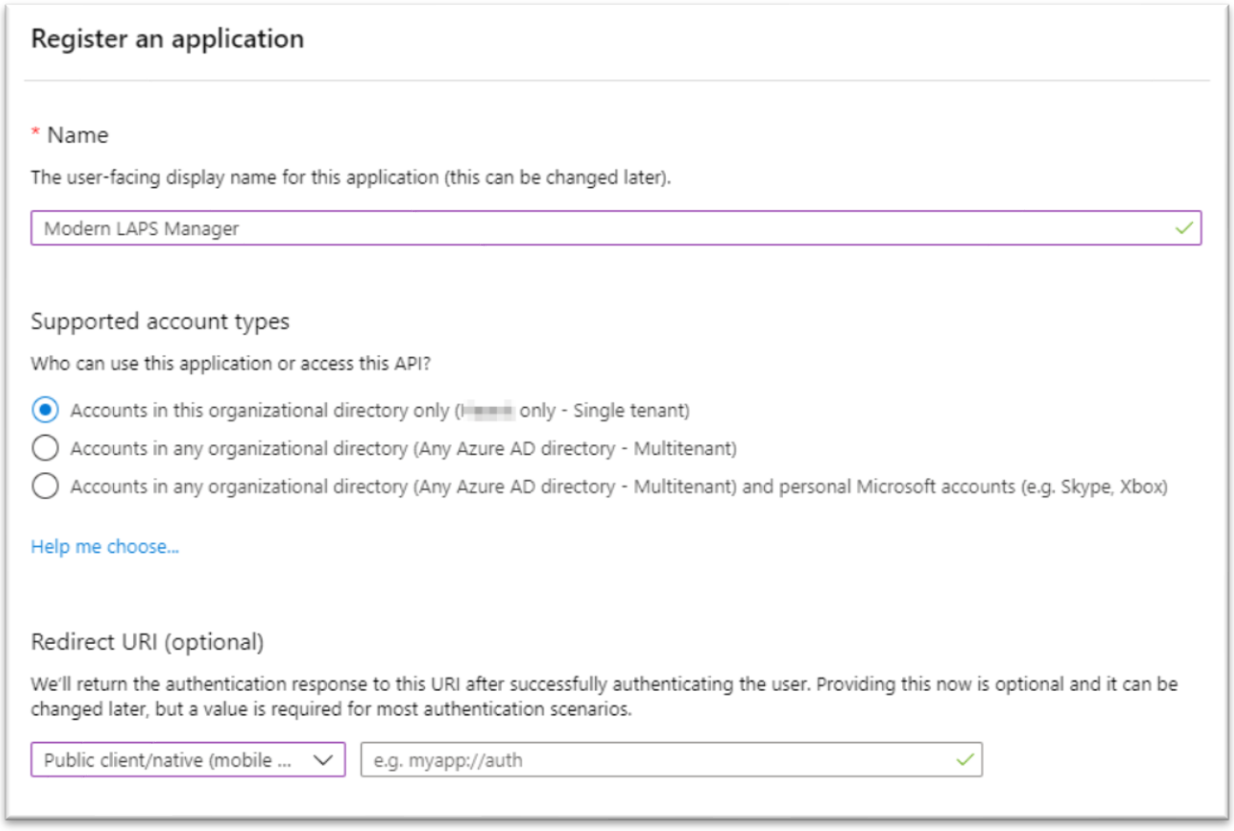
Add a lock to prevent deletion of resources in the resource group.




## Create a New App Registration

An app registration is needed to represent the **Modern LAPS Manger** custom app and act on behalf of the user in the tenant.

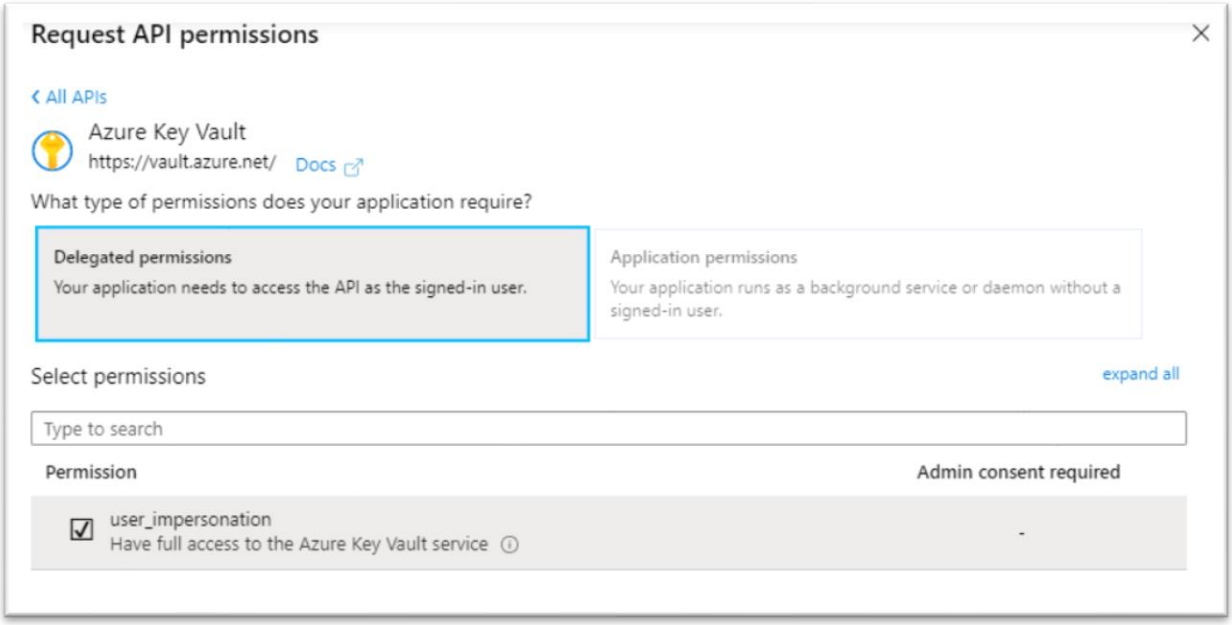
- Navigate to **App registrations** and click **New registration**.
- In the **Register an application** pane, enter **Modern LAPS Manager** as the app name.
- Select **Accounts in this organization directory only**
- For the **Redirect URI**, select **Public/client native**. There is no need to enter a redirect URI at this point.
- Click **Register**



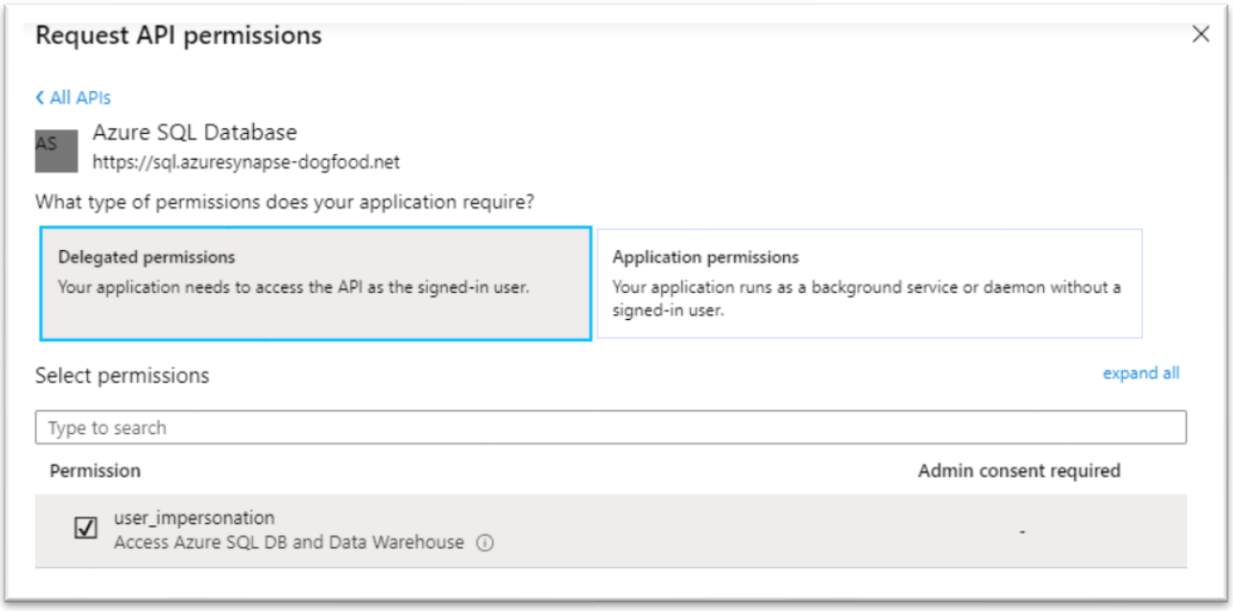
## Configure the App registration

- In the **Branding** pane, upload the included logo icon file if desired, or choose one of your own.
- In the **Authentication** pane, click **Add a platform**, then select **Mobile and desktop applications**. In the **Redirect URIs**, select **https://login.live.com/oauth20_desktop.srf** then click **Configure**.


- In the **API permissions** pane, click **Add a permission**. Select **Azure Key Vault**. Select **Delegated permissions** and check **user_impersonation**. Click **Add permissions**.




- In the **API permissions** pane, click **Add a permission** again. Under **APIs my organization uses**, search for and select **Azure SQL Database**. Select **Delegated pemissions** and check **user_impersonation**. Click **Add permissions**.




- Under the **Owners** pane, click **Add owners**. You can't add the Modern LAPS Administrators AAD group, but you can add the individual members here as owners.

## Configure the Enterprise application

- Search for **Enterprise applications**, then search for the **Modern LAPS Manager** app.
- On the **Properties** pane, change **User assignment required** to **Yes**.[1]
- Under the **Owners** pane, click **Add owners**. You can't add the Modern LAPS Administrators AAD group, but you can add the individual members here as owners.
- Under **Users and groups**, click **Add user**. Under **Users and groups**, search for and select the **Modern LAPS Users** group. Click **Assign**.
- Under the **Permissions** pane, click **grant admin consent** to the app for your organisation.[1]

[1]*About user assignment and admin consent*: If you do not use user assignment, then the app is open to anyone to access. Their access to Azure services is still limited by their own user permissions. If you do not grant admin consent, then the user is prompted for consent to use the application the first time. The preferred option is to require user assignment and control who can use the app in the User and groups pane. This option requires that you grant admin consent; the user not be able to provide their own consent in this case as the API permissions assigned to the app require admin consent.

# Create an Azure Key Vault

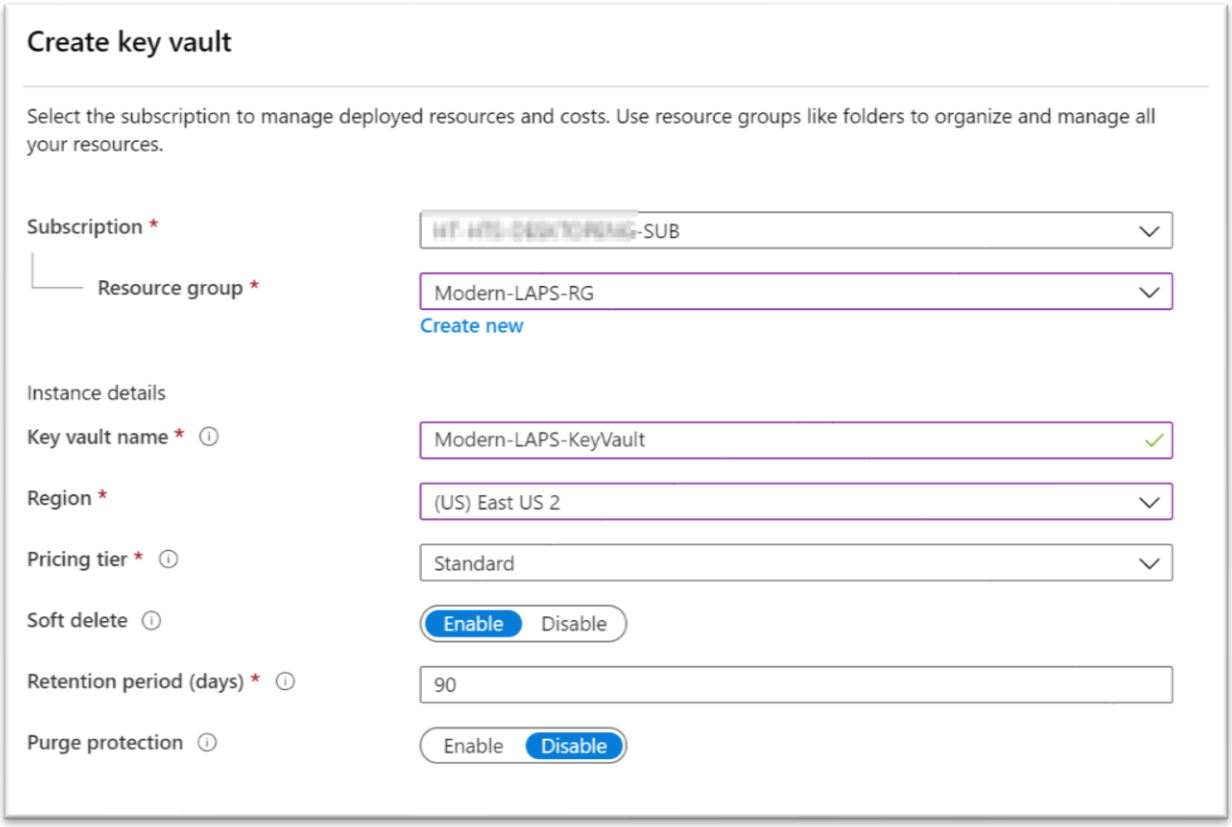
Here we will create a Key Vault to store credentials and encryption keys. You may have existing key vaults, but create one specifically for this solution because it is not possible to grant access only to specific secrets in the vault. We can only grant general access at the data-plane level to all secrets in the vault, and you don't want to grant users of this solution access to secrets other than those needed by the solution.

We will grant access to the Key Vault to the Modern LAPS AAD groups using access policies.

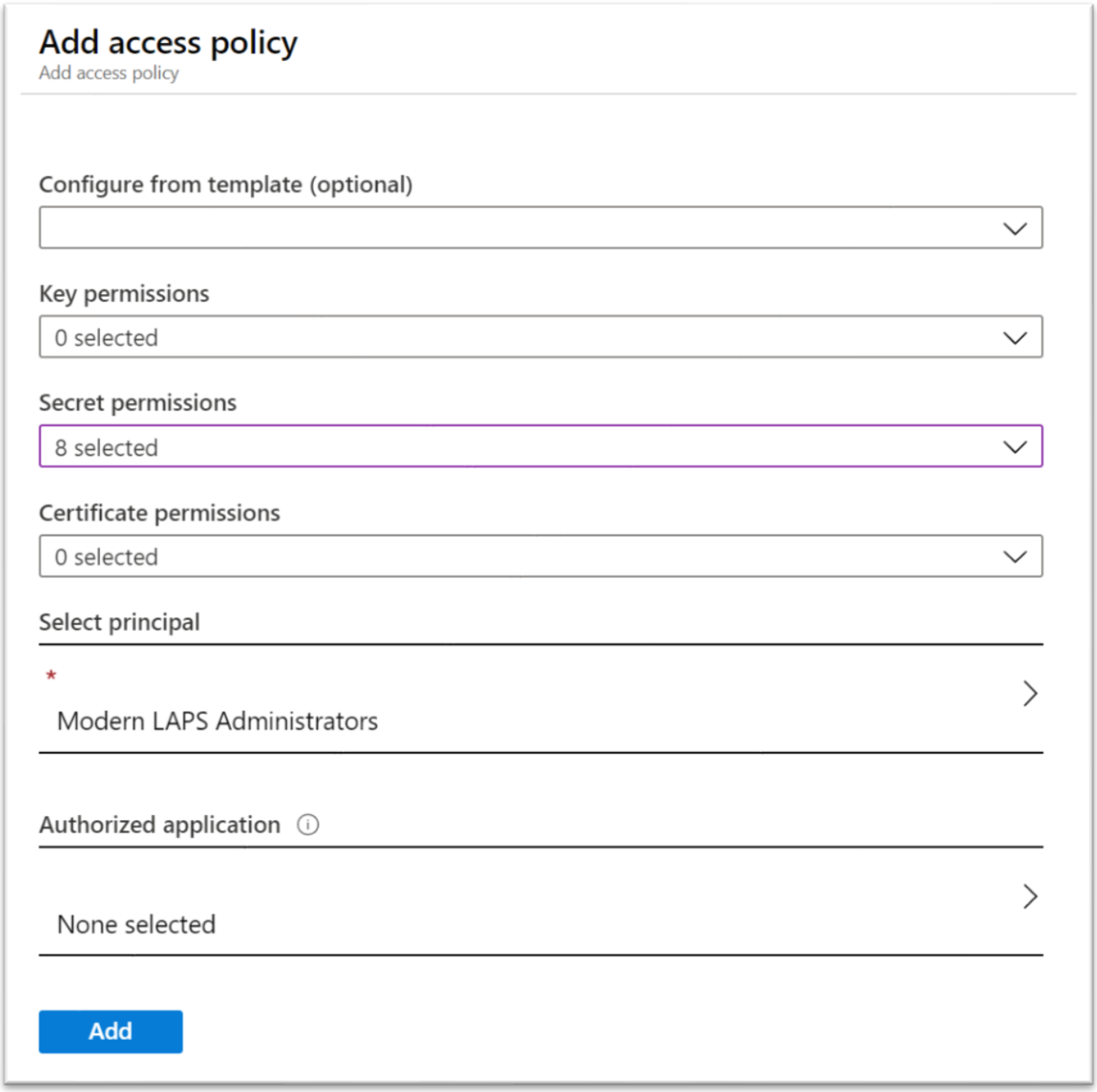
In the Azure portal, search **Key Vaults**.

- Click **Add**
- Select your **subscription**
- Select your **resource group**
- Give the key vault a **name**

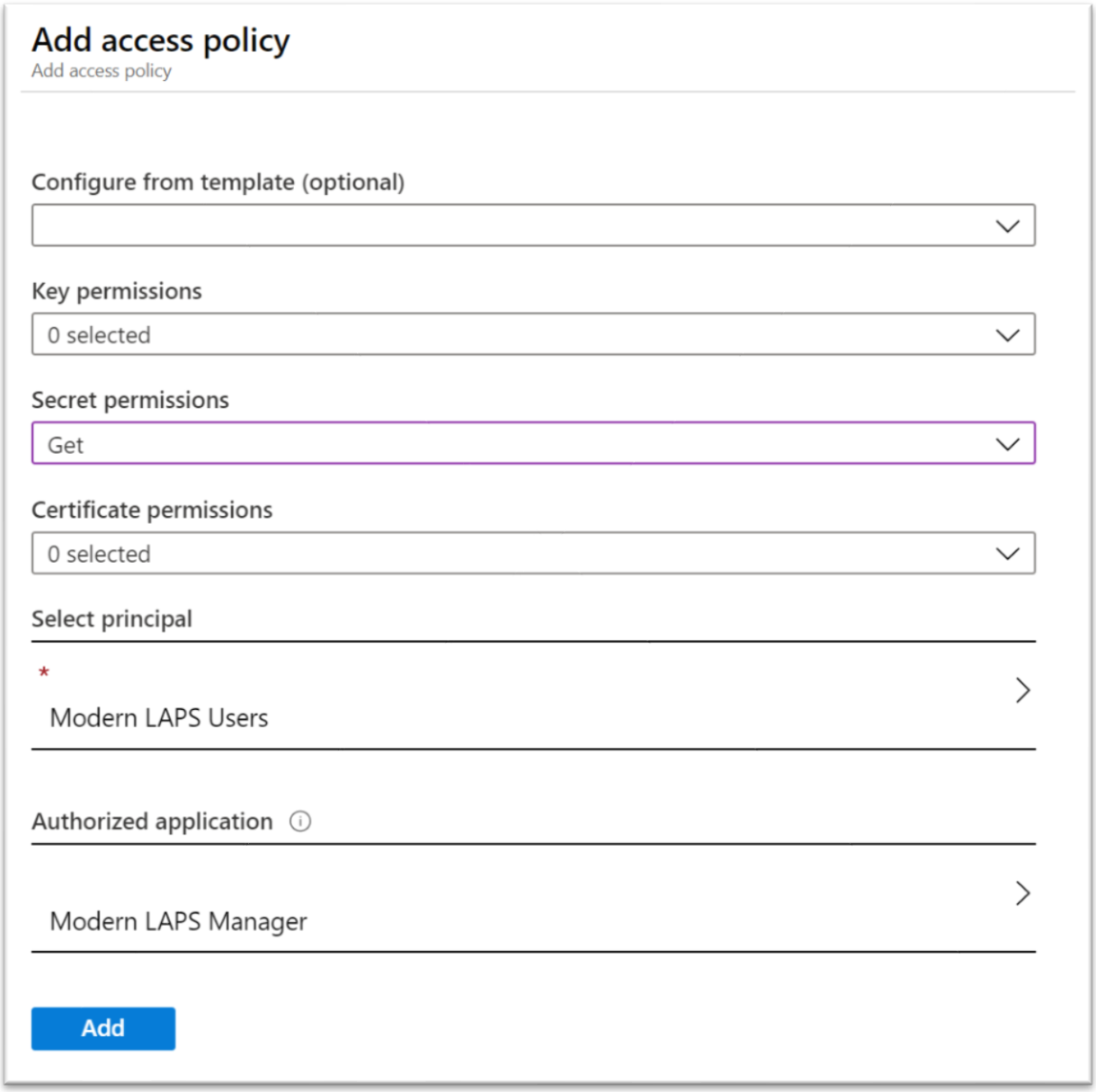
- Select your **region**
- For **pricing tier**, **Standard** is sufficient
- The other settings can be adjusted according to your requirements

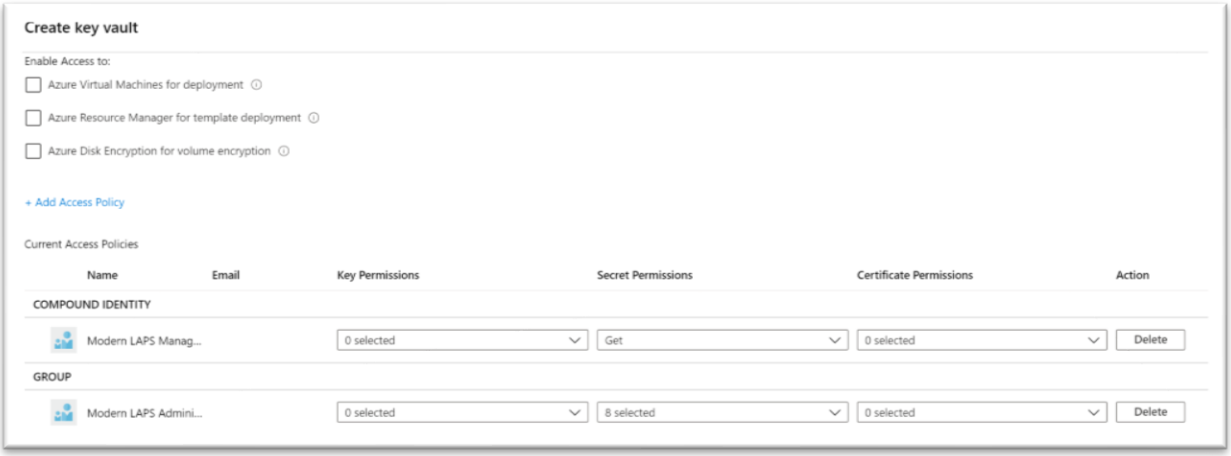



- Click **Next: Access policy >**
- Click **Add Access Policy**
- Under **Secret permissions**, click **select all**
- Under **Select principal**, search for and select the AAD group **Modern LAPS Administrators**
- Click **Add**

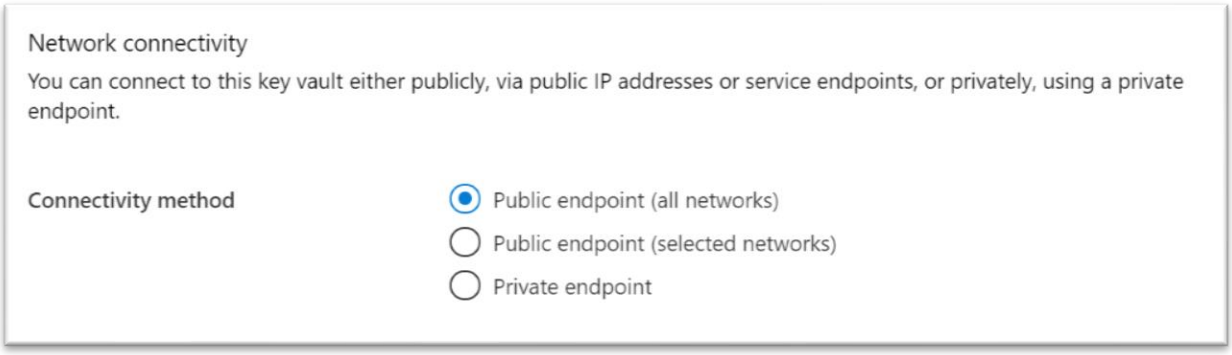



- Click **Add Access Policy** again
- Under **Secret permissions**, select **Get**
- Under **Select principal**, search for and select the AAD group **Modern LAPS Users**
- Under **Authorized application**, search for an select **Modern LAPS Manager**
    - This will create a compound identity that allows members of the group to access the vault via the Authorized application
- Click **Add**




- Click **Next: Networking >**
- Allow connectivity via a **Public endpoint (all networks)**




- Click **Review + create**, then **Create**

## Create Encryption Keys

Create some encryption keys. These will be used for encrypting the local administrator passwords. We need to create two sets of keys. One will be used to encrypt the local administrator password during transit (Client encryption key). The other will be used to encrypt the password at rest (Active encryption key).

To create an encryption key we will use PowerShell and the System.Security.Cryptography.Aes .Net class. Run the following code, which will create a 256-bit AES encryption key using the default CBC mode.

```
$aes = [System.Security.Cryptography.Aes]::Create()
[System.Convert]::ToBase64String($aes.Key)
[System.Convert]::ToBase64String($aes.IV)
```

The first value is the encryption key as a base 64 string, the second value is the initialization vector as a base 64 string.

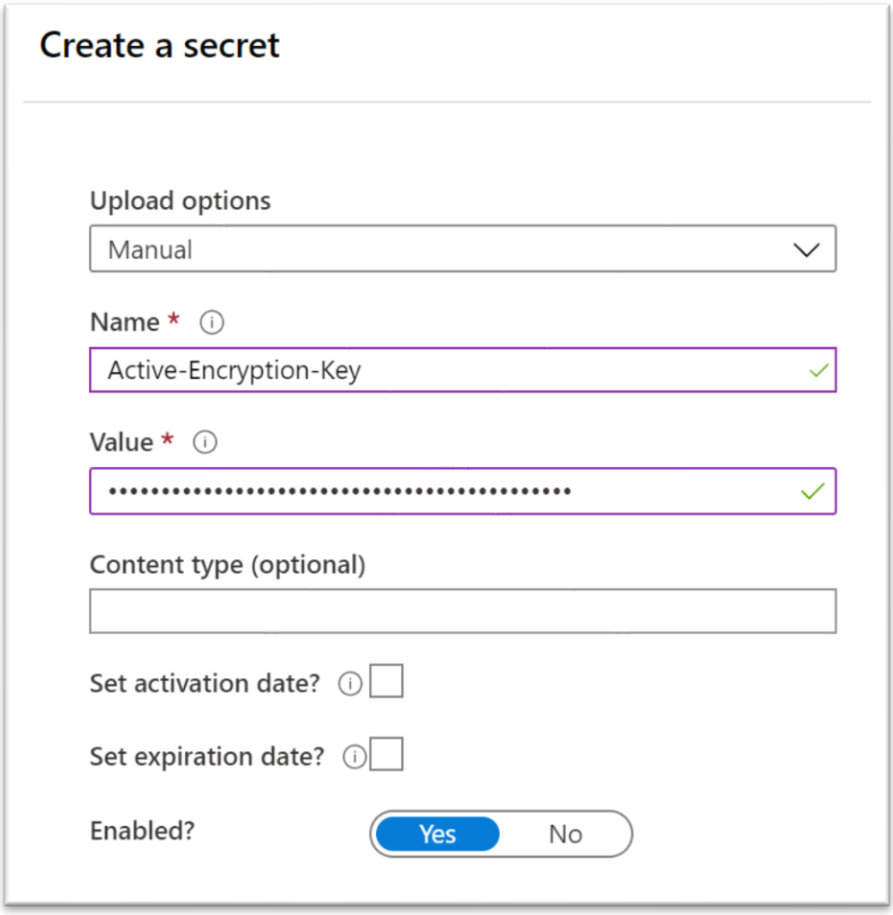
## Save the Encryption Keys to the Vault

The following 6 secrets must be created in the vault. You must create the Active and Client keys. The Previous key secrets will just be placeholders for now. They are used for key rotation where the active key becomes the previous key and a new active key is generated.

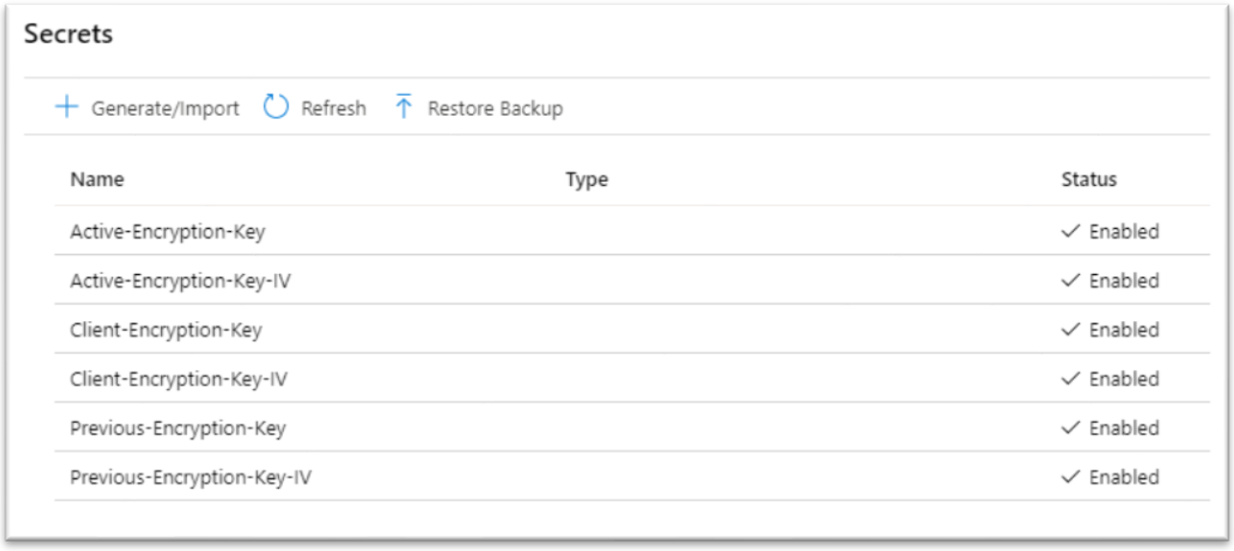
| Secret Name | Purpose |
|---|---|
| **Active-Encryption-Key** | This key is used to encrypt the password at rest. It is the currently active key. |
| **Active-Encryption-Key-IV** | This is the initialization vector for the above key. |
| **Previous-Encryption-Key** | This key will be the previous encryption key for the password at rest, used during key rotation. |
| **Previous-Encryption-Key-IV** | This is the initialization vector for the above key. |
| **Client-Encryption-Key** | This key is used to encrypt the password during transit. |
| **Client-Encryption-Key-IV** | This is the initialization vector for the above key. |

To add a secret to the vault:

- In the Azure Key Vault, click **Secrets**.
- Click **Generate/Import**
- Under **Upload options**, select **Manual**
- Enter the name: **Active-Encryption-Key**
- Under value, paste the encryption key string
- Click **Create**




Repeat the process for the other 5 secrets. For the **Previous-Encryption-Key** and **Previous-Encryption-Key-IV** secrets, simply enter "." (dot, full stop) for now as a placeholder. These secrets will not be populated until key rotation is used for the first time.

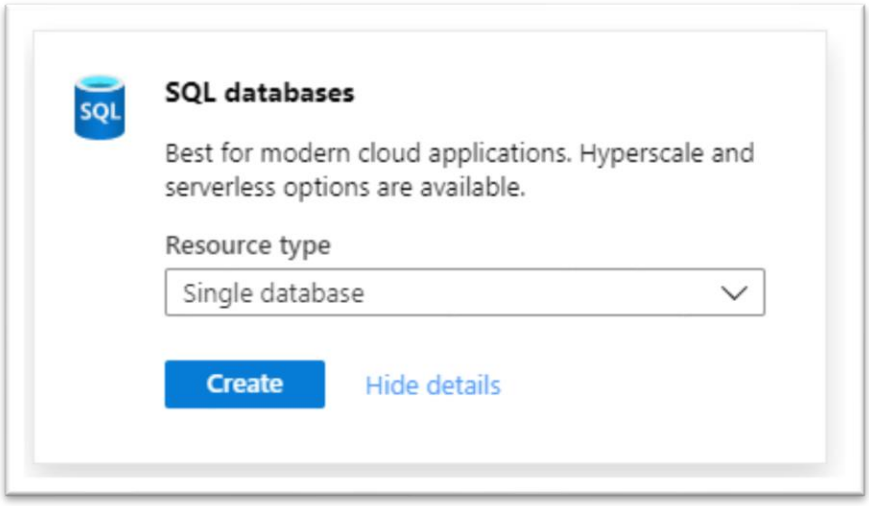



# Create an Azure SQL Database

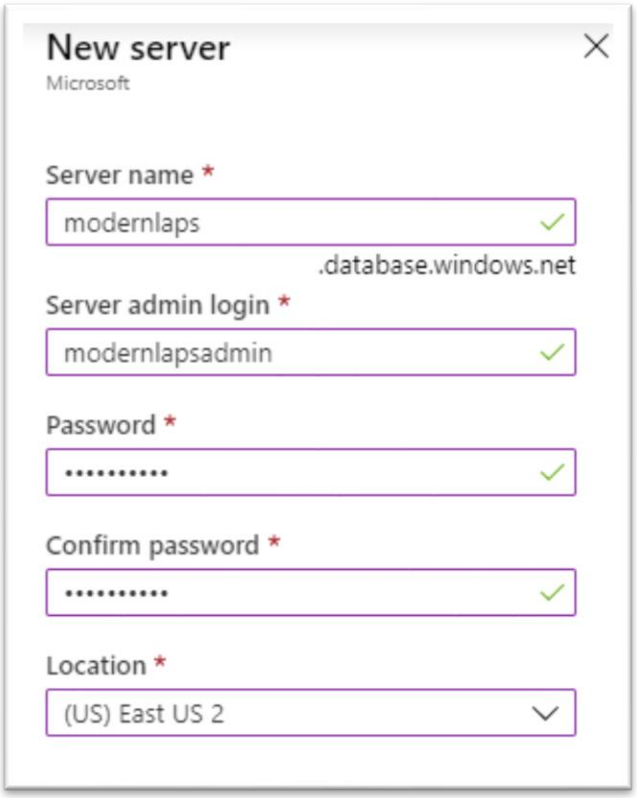
Search for **Azure SQL** in the Azure portal.

## Create a Database

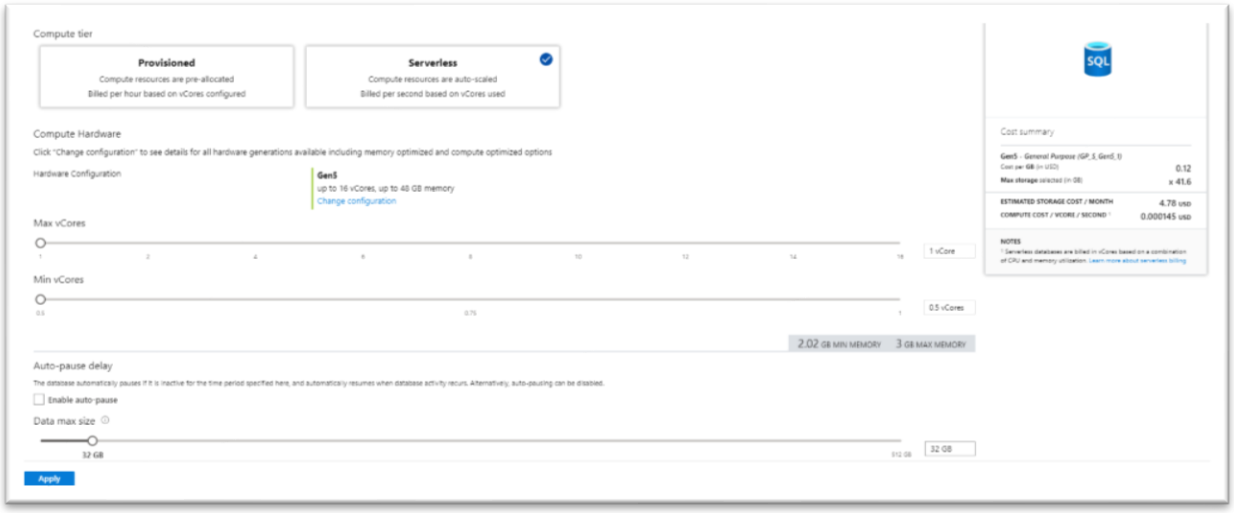
- Click **Add**
- Under **SQL databases**, select the **Single database** resource type and click **Create**.

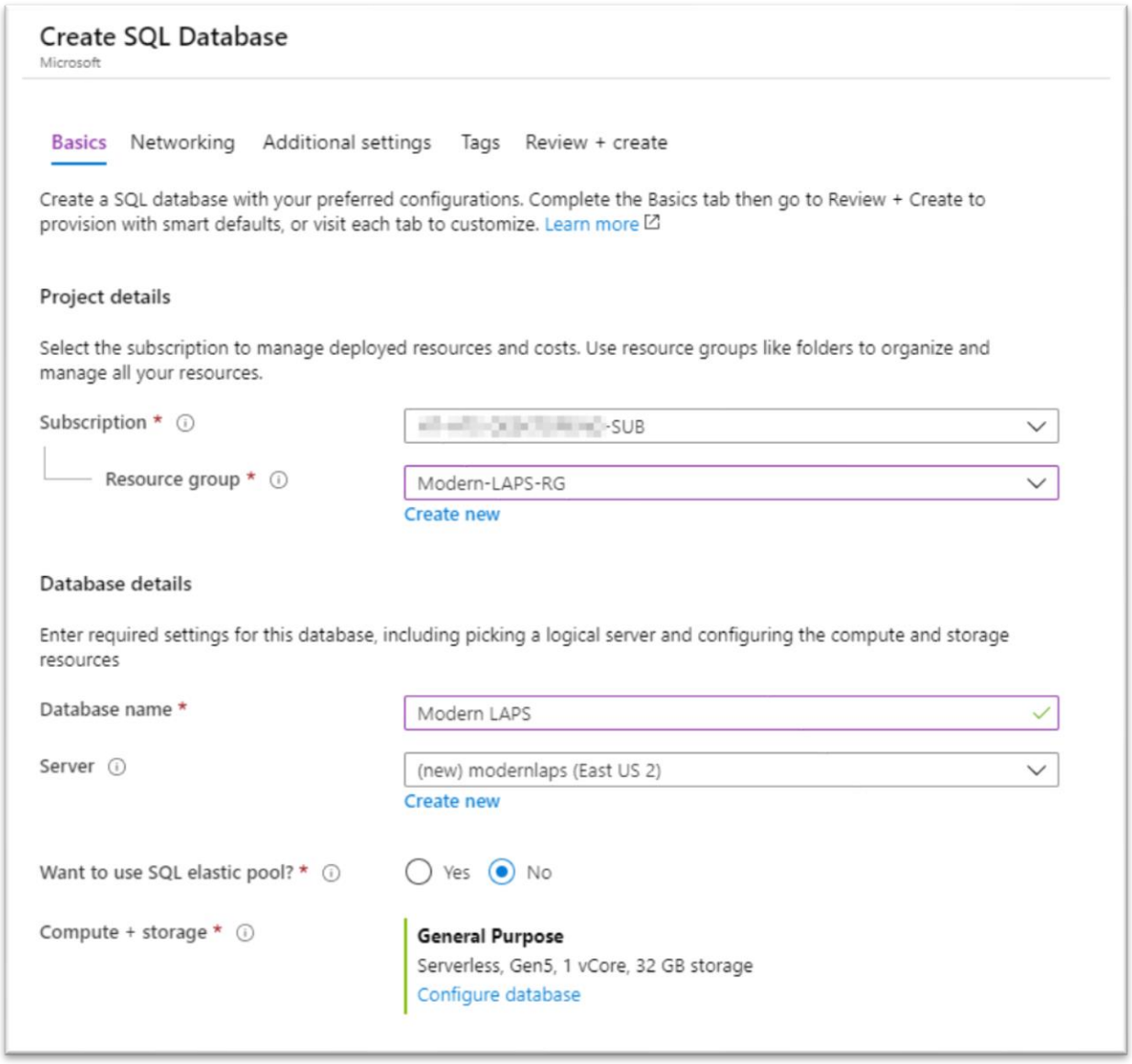



- On the **Basics** page, select your **subscription**
- Select the **Resource group** you created earlier
- Enter the **Database** name
- For **Server**, if you have an existing server you can use it, or create a new one as follows:
    - Click **Create new**
    - Enter the **Server name**
    - Enter a **Server admin login**
    - Enter a **password**
    - **Confirm** the password
    - Select the **Location**
    - Click **OK**

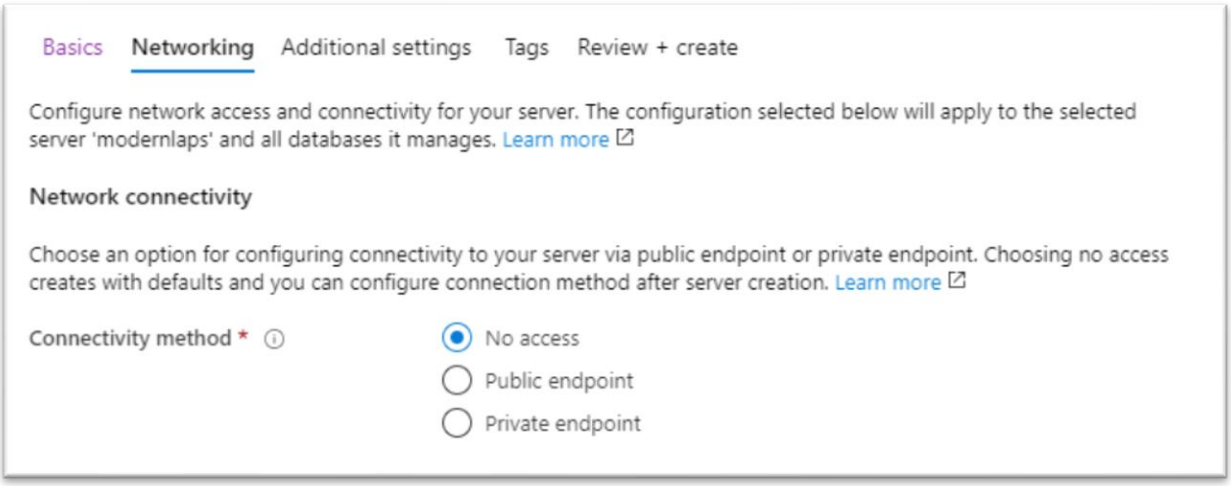



- Select **No** to use a **SQL elastic pool**
- Under **Compute + storage**, select **Configure database**
- Under the **General Purpose**, I recommend to choose **Serverless** so that you can start with the minimum configuration and it can be auto-scaled if needed. Serverless databases are billed on the consumption model. You can change this configuration after creation of required.
    - Under **Hardware Configuration**, you can leave the default select, ie Gen5.
    - You can leave the **Max vCores** set at 1 and the **Min vCores** at 0.5
    - Under **Auto-pause delay**, **deselect** the **Enable auto-pause** option. **This is important** because if the database is paused due to inactivity, the first attempt to connect to it will fail. We don't want any failures or delays in adding data to or retrieving data from the database.
    - You can adjust the **Data max size** as required.
    - Note the **cost summary** on the right. You can use this as a guide for how much the selected resources will cost you.
    - Click **Apply**

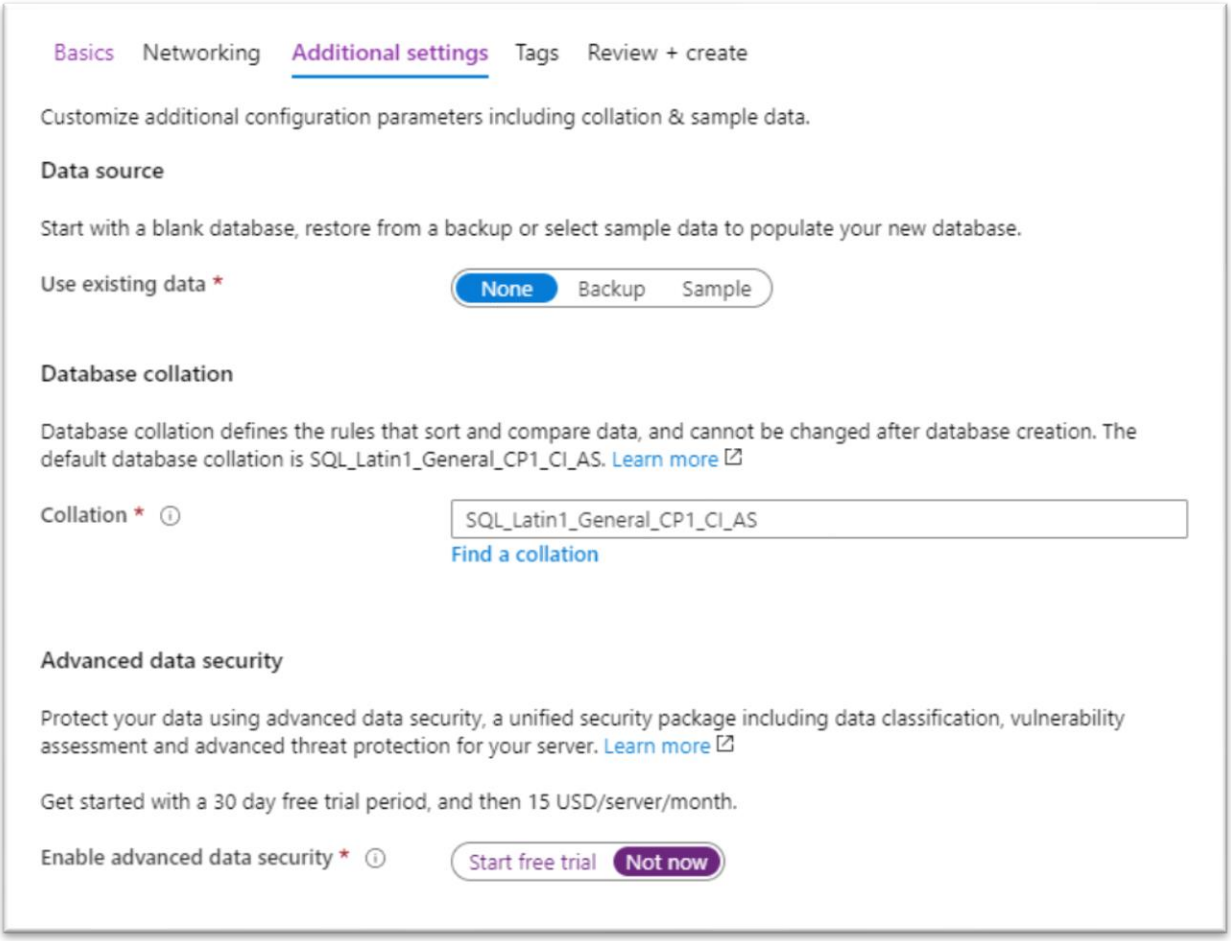



- Back in the **Basics** page, click **Next: Networking >**
- Under **Connectivity method**, select **No access**



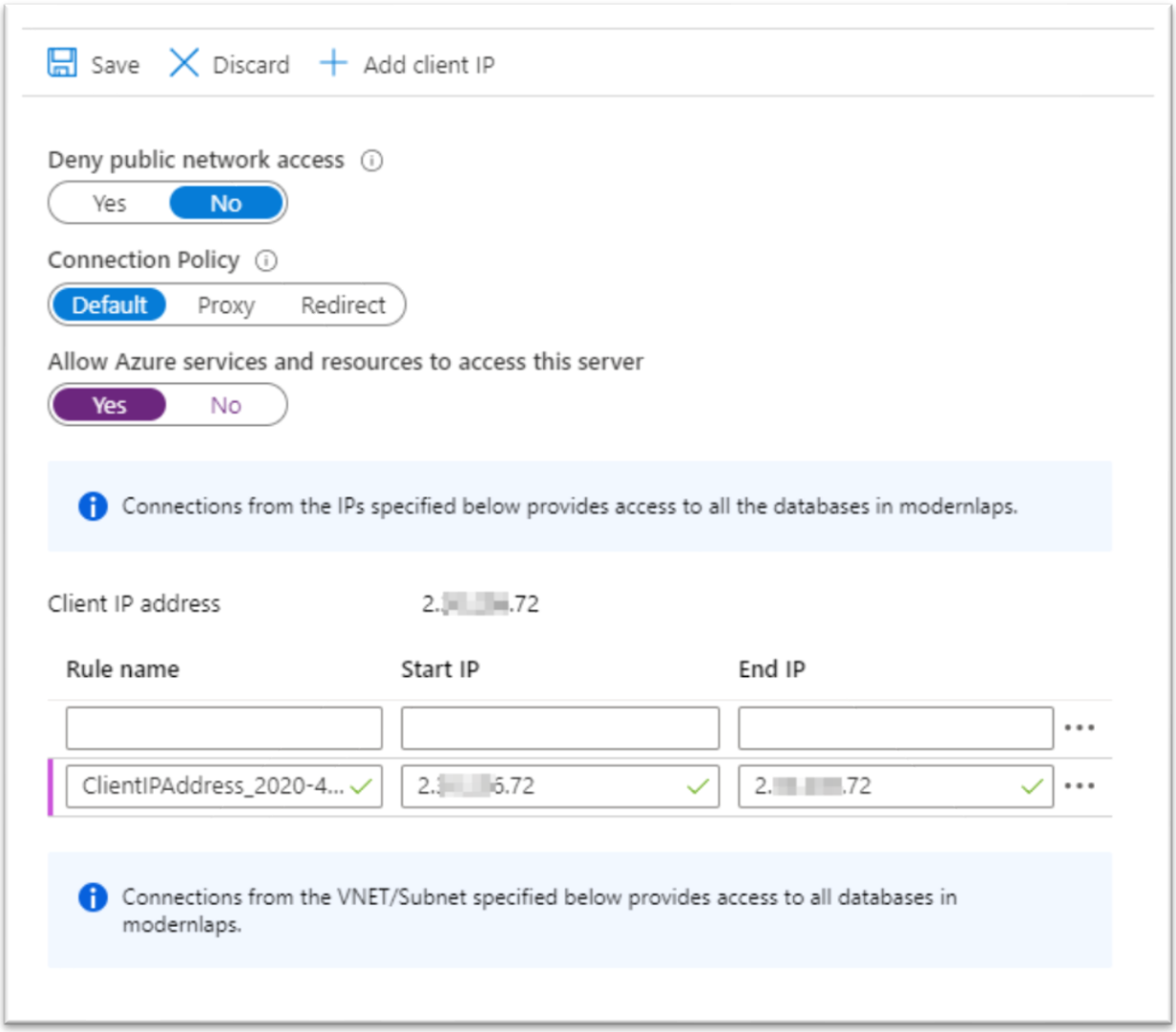

- Click **Next: Additional settings >**
- Under **Data source > Use existing data**, select **None**
- Under **Database collation** use the default collation
- Under **Advanced data security**, select **Not now**. I strongly recommend to use Advanced data security but we can configure it later




- Click **Review + create**, then **Create**.
- Continue once the deployment has complete.

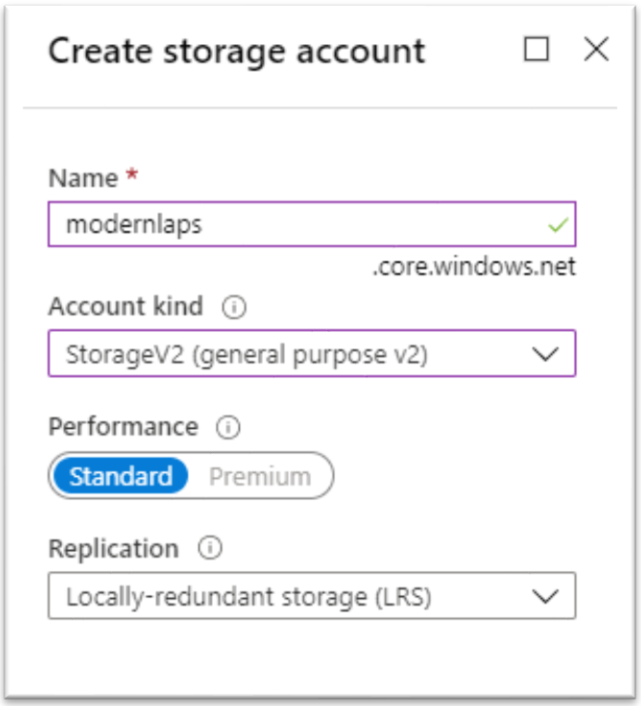
## Configure the SQL Server

- In the Azure portal, search **Azure SQL** again.
- Click on the **SQL server** you just created.
- On the **Manage Backups** pane, configure a database backup **retention policy** per your requirements.
- Under **Active Directory admin**, we will set an Azure AD group to allow Azure AD authentication.
    - Click **Set admin**
    - Search for and select the **Modern LAPS Administrators** group
    - Click **Save**
- Under **Transparent data encryption**, confirm that this is enabled with a Service-managed key.
- Under **Firewalls and virtual networks**, you will need to add a firewall rule to allow you to perform the initial configuration of the database from your local workstation. After the database is configured, you can remove this rule.
    - At the top, click **Add client IP**. This will create a rule for your current public IP address
    - While we are here, select **Yes** to **Allow Azure services and resources to access this server**. This is necessary to allow the Azure function app to communicate with the server.
    - Click **Save**.

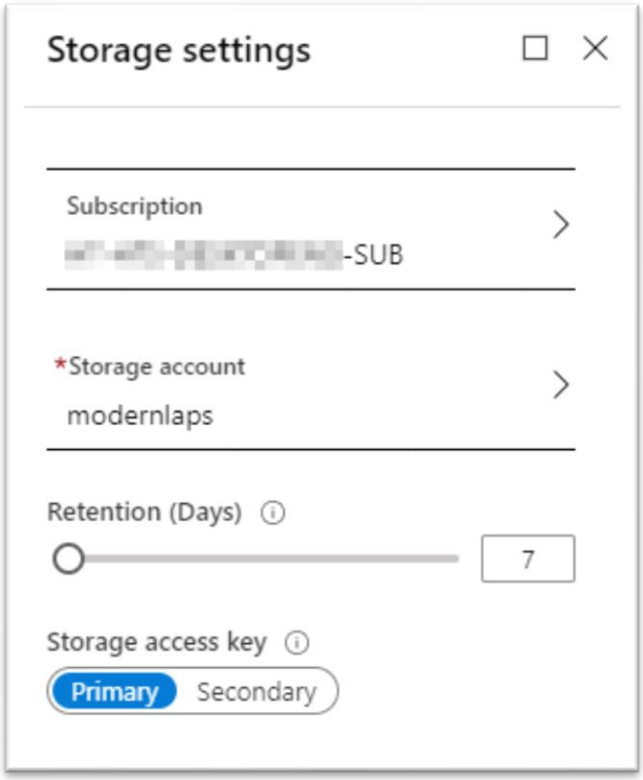



- Under **Auditing**, select **On**. I strongly recommend to enable auditing for security reasons.
    - You can select whichever log destination suits your needs. Here we will use a storage account
    - Check **Storage**, the click **Configure**

- Select your **subscription**, then click **Configure required settings**
- Create and configure a new storage account. I recommend StorageV2

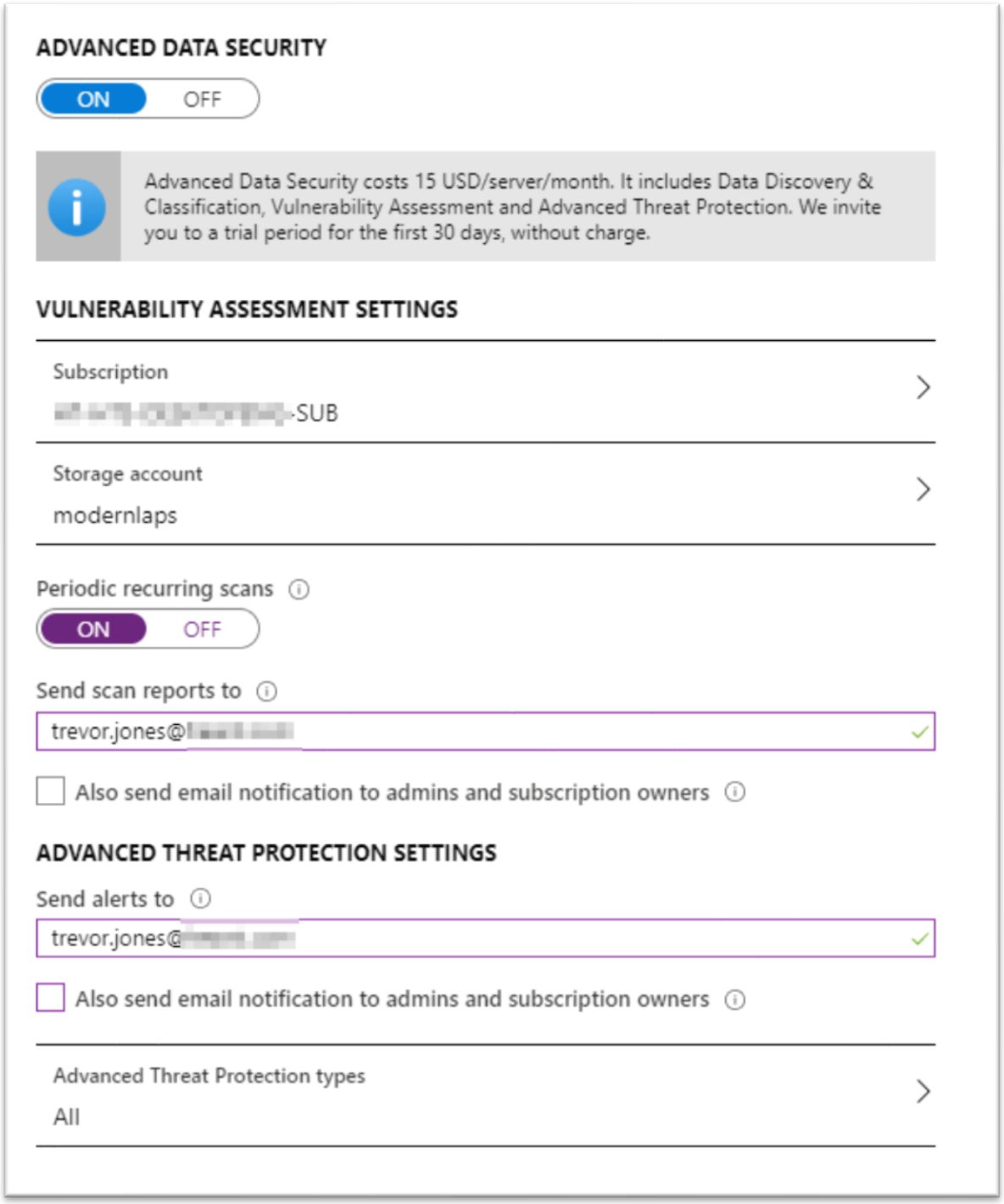



- Set a **retention period** for your logs



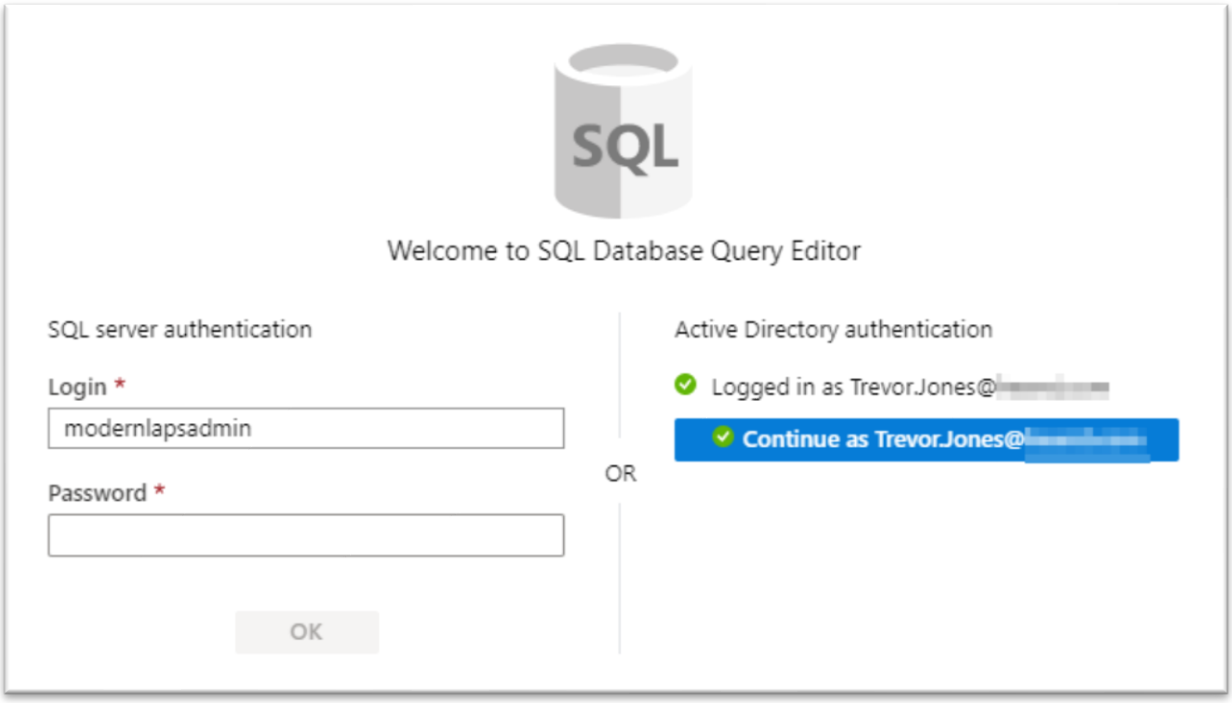

- Click **OK**
- Back in the **Auditing** pane, click **Save**.

- Under **Advanced data security**, I strongly recommend to turn it **ON**. There is a small cost associated with this service.
  - Select the **Storage account** you just created
  - Enable **periodic recurring scans** and send scan reports to an administrator



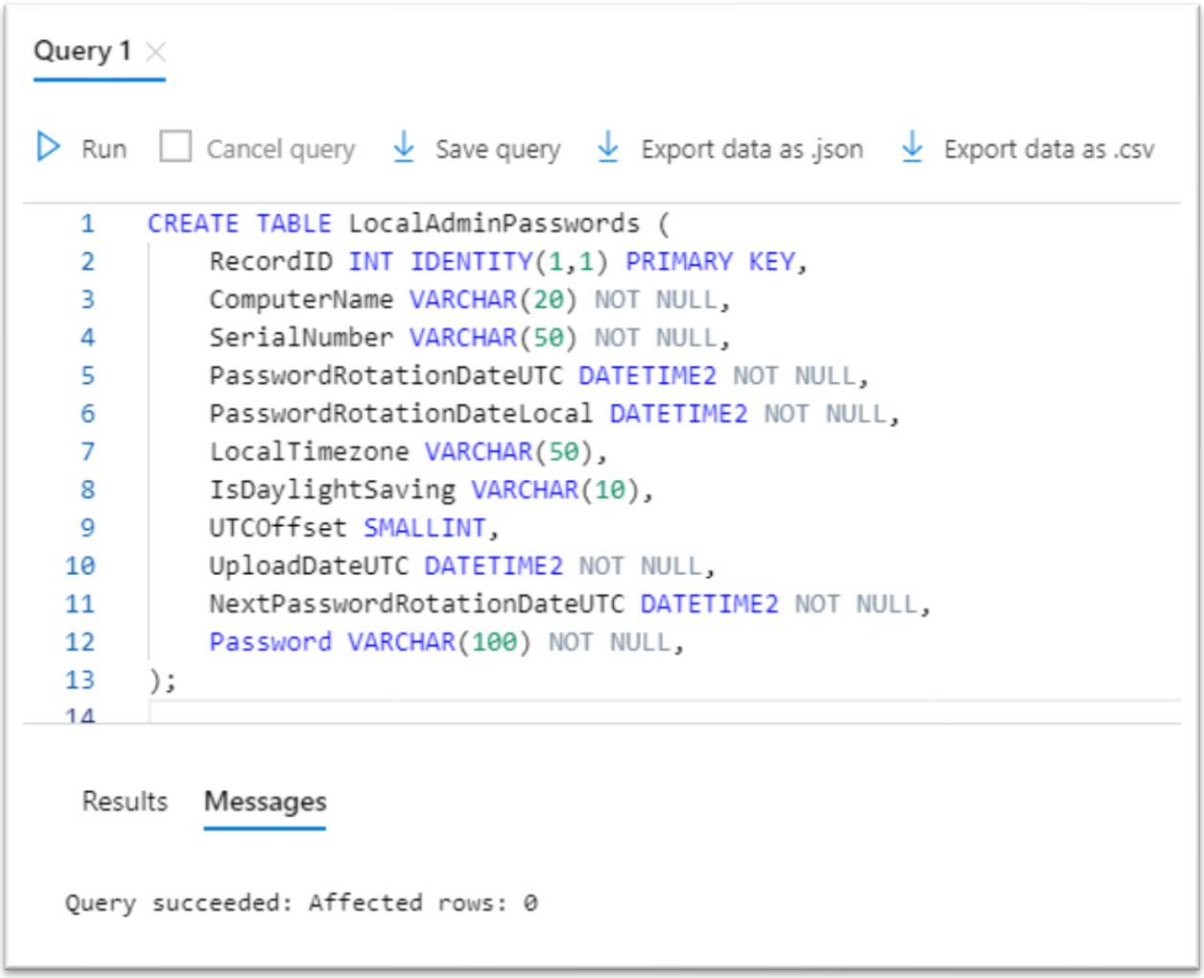

## Configure the SQL Database

- In the **SQL server** blade, select **SQL databases**, then select the database you created
- On the **Query editor** pane, you should be able to login with your Azure AD account if you are a member of the **Modern LAPS Administrators** group. Otherwise login with SQL authentication.

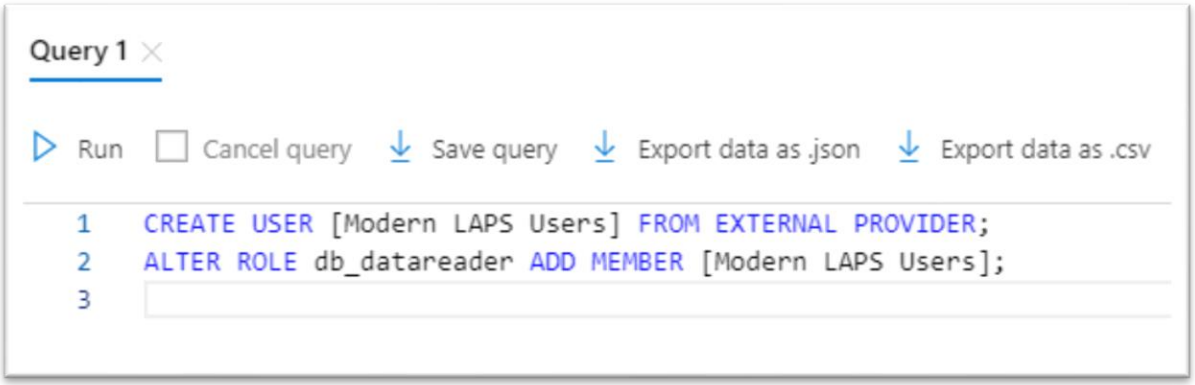



- First, lets create a table in the database to store the local admin password data.
  - In the Query editor window, paste and run the following code:

```
CREATE TABLE LocalAdminPasswords (
    RecordID INT IDENTITY(1,1) PRIMARY KEY,
    ComputerName VARCHAR(20) NOT NULL,
    SerialNumber VARCHAR(50) NOT NULL,
    PasswordRotationDateUTC DATETIME2 NOT NULL,
    PasswordRotationDateLocal DATETIME2 NOT NULL,
    LocalTimezone VARCHAR(50),
    IsDaylightSaving VARCHAR(10),
    UTCOffset SMALLINT,
    UploadDateUTC DATETIME2 NOT NULL,
    NextPasswordRotationDateUTC DATETIME2 NOT NULL,
    Password VARCHAR(100) NOT NULL,
);
```




- Next we need to create a **contained database user** for the **Modern LAPS Users** AAD group. This will grant them read access to the database for password retrieval.
    - Paste and run the following code into the Query editor:
      **CREATE USER [Modern LAPS Users] FROM EXTERNAL PROVIDER;**
      **ALTER ROLE db_datareader ADD MEMBER [Modern LAPS Users];**




- Later we will grant access to managed identity of the Azure function app once we have created it.

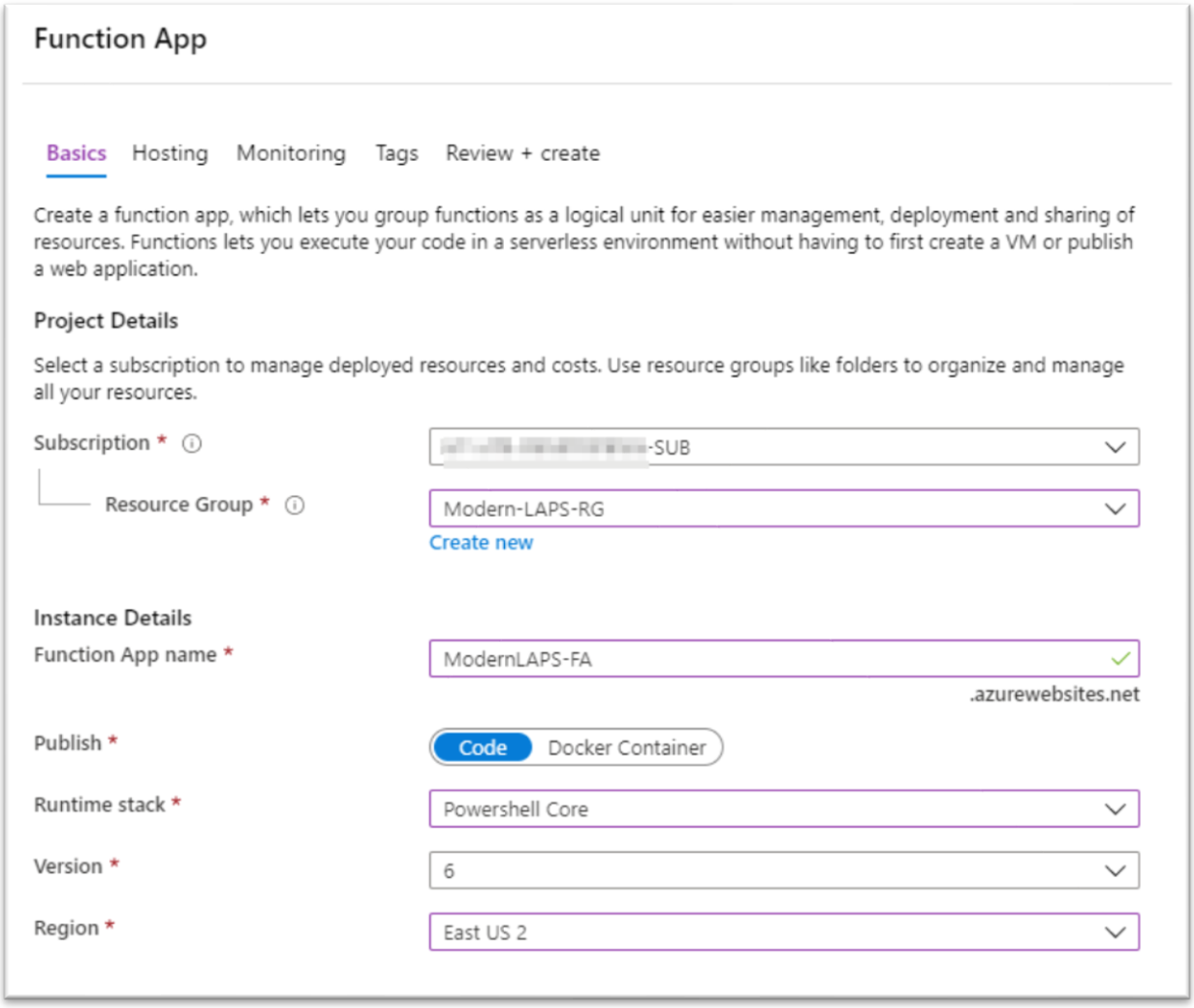
# Create an Azure Function app

## Create the Function app

Here we will create a function app which will be used to host our serverless functions. These functions are responsible for interacting with the SQL database, securely adding and retrieving records.

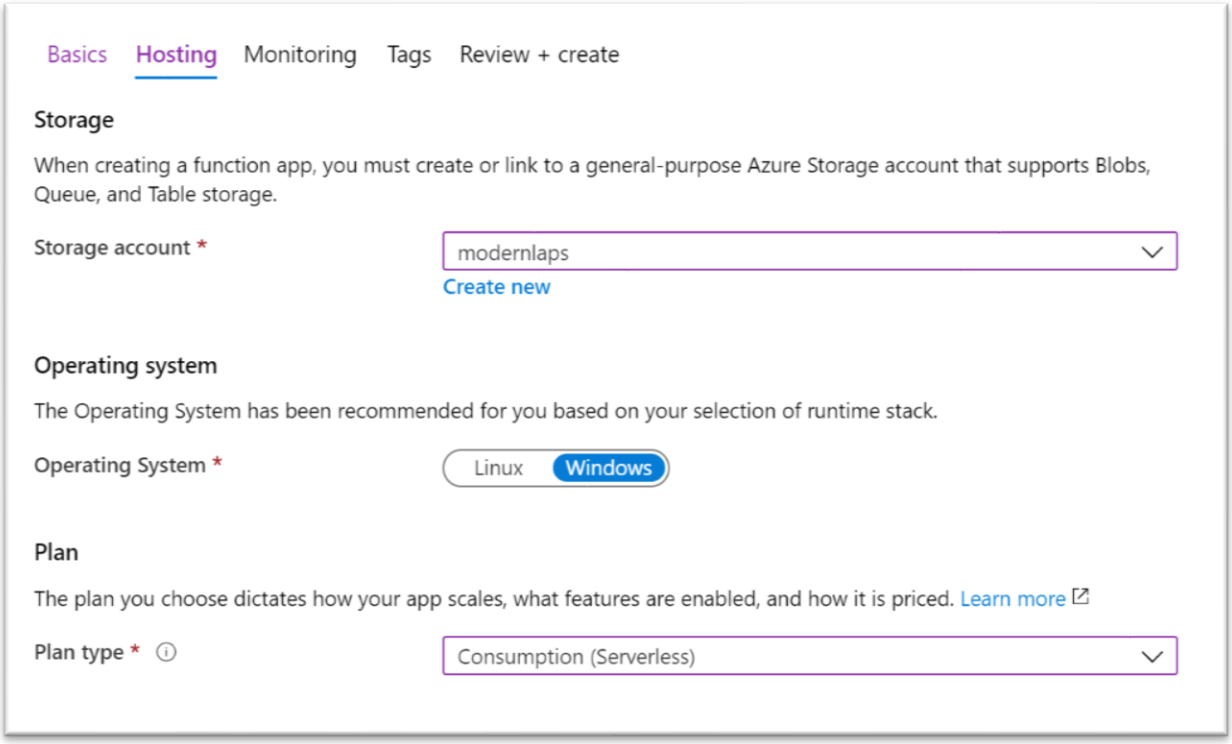
In the Azure portal, search for **Function app**.

- Click **Add**
- On the **Basics** page, select your **subscription**
- Select the **resource group** you created earlier
- Give the function app a **name. This name must be unique in your tenant.**
- For **Publish**, choose **Code**
- For **Runtime stack**, choose **PowerShell Core**
- For **Version**, **6**
- Select your **region**

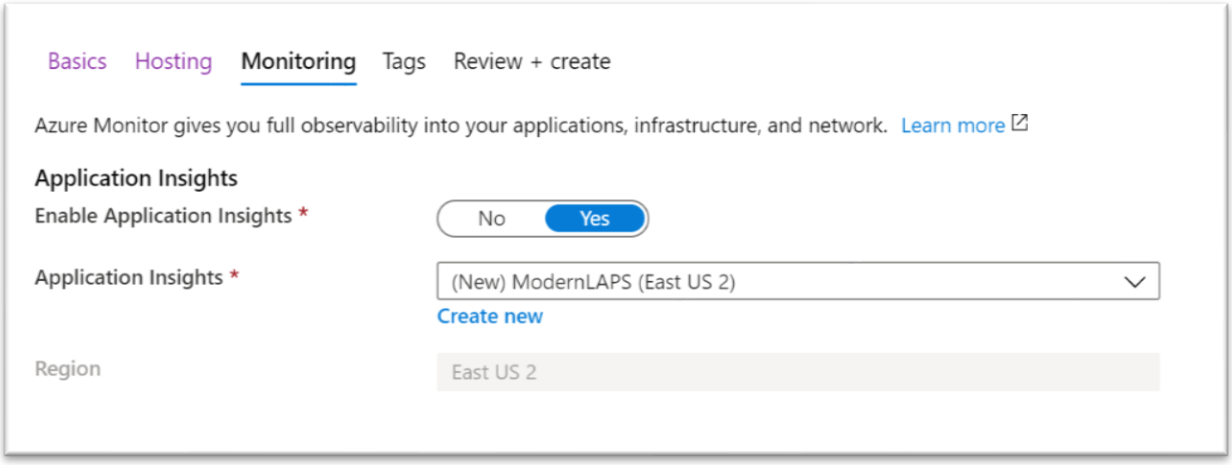



- Click **Next: Hosting >**
- On the **Hosting** page, choose the **storage account** you created earlier
- Choose the **Windows** Operating system

- For the plan, I recommend to choose **Consumption**. This should be sufficient for the solution. If you require the additional features of a Premium plan such as warm instances and vNet connectivity, choose that, but bear in mind there will be a cost.



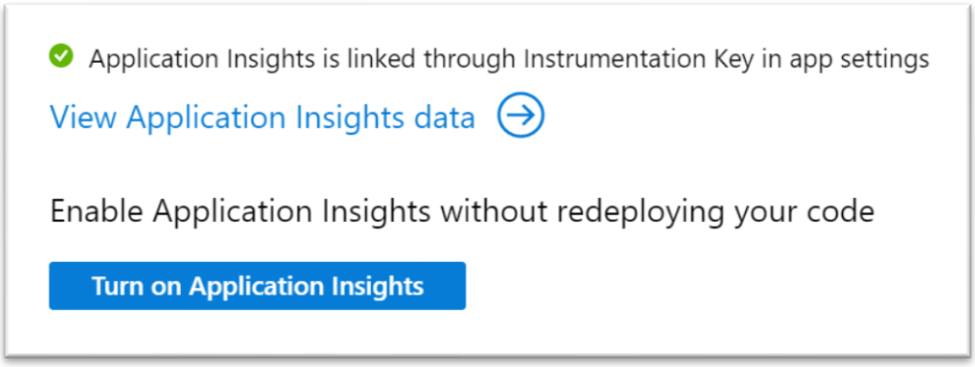

- Click **Next: Monitoring >**
- On the **Monitoring** page, I recommend to enable **Application Insights**. This exposes useful information for your function app including live monitoring and analytics.

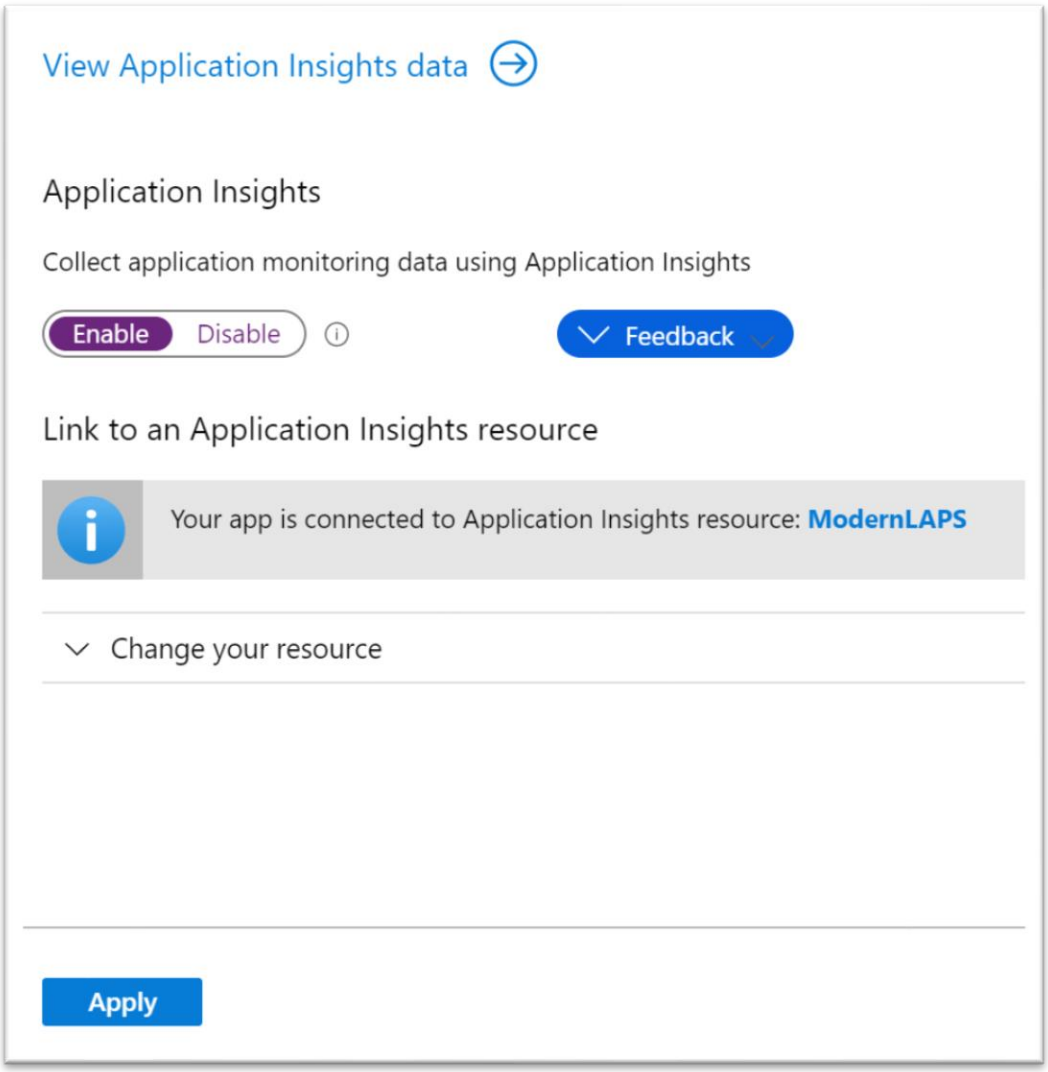



- Click **Review + create**, then **Create**.
- When the deployment is complete, click **Go to resource**

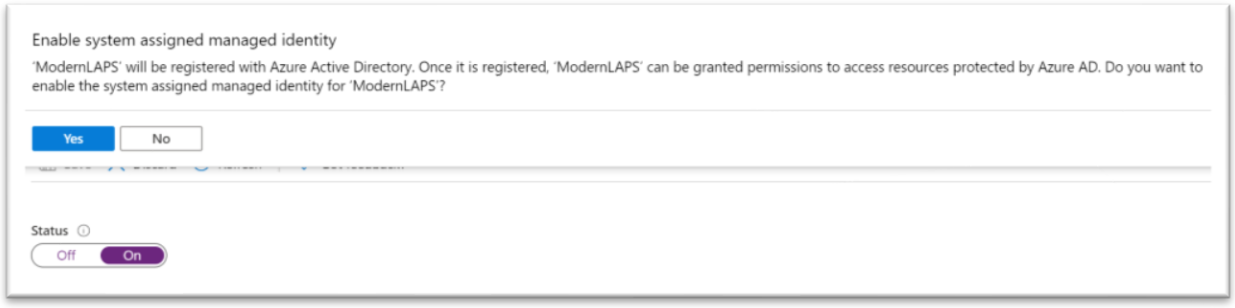
## Configure the Function app

- On the **Application Insights** pane, click **Turn on Application Insights**, then select **Apply**

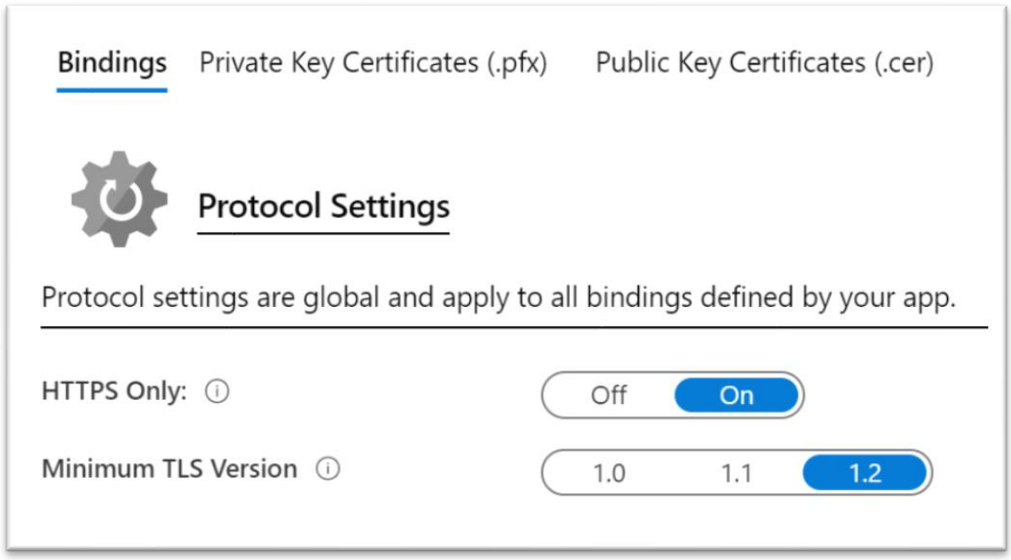



- On the **Identity** pane, under **System assigned**, turn the status to **On**. Click **Save**, then **Yes** at the prompt. This will create a managed identity – a service principal in Azure Active Directory – that can be used to access the SQL database without credentials.

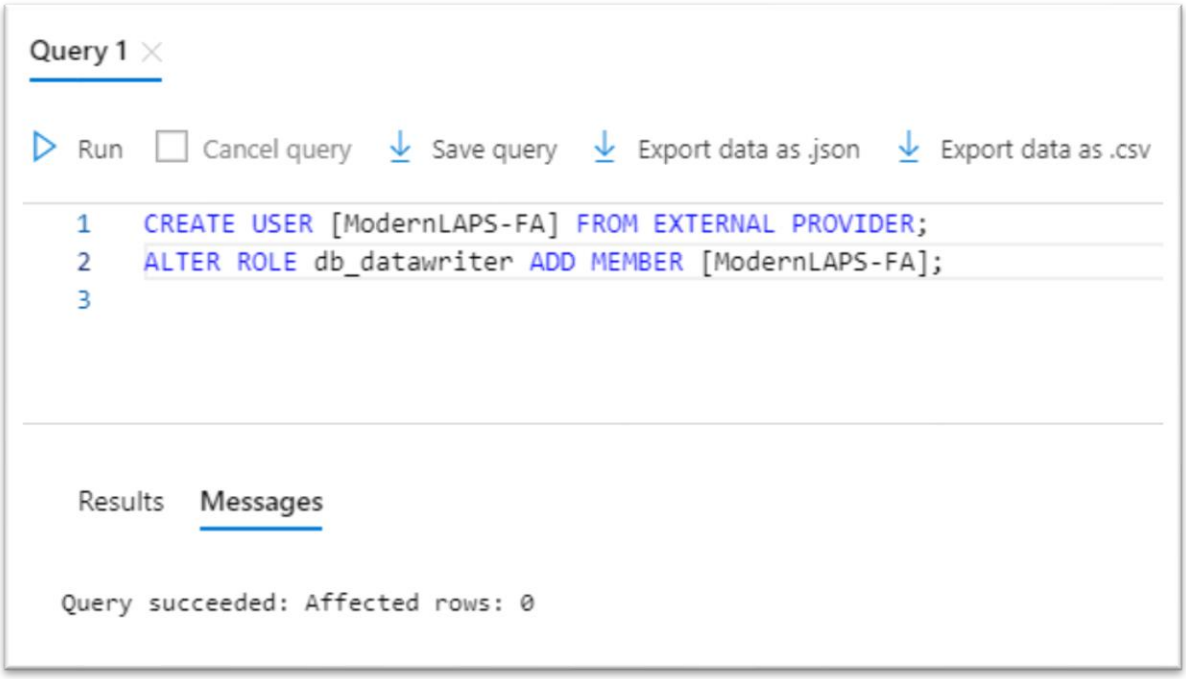



- On the **Custom domains** pane, turn **HTTPS Only** to **On**
- On the **TLS/SSL** pane, on the **Bindings** tab, ensure that **HTTPS Only** is **On** and **Minimum TLS Version** is set to **1.2**.




## Grant the Managed Identity Write Access to the SQL Database

- In the Azure portal, search **Azure SQL**
- Select the **database** you created earlier
- Select **Query editor** and sign in
- Paste and run the following code, changing the user name to the name of your function app
  **CREATE USER [ModernLAPS-FA] FROM EXTERNAL PROVIDER;**
  **ALTER ROLE db_datawriter ADD MEMBER [ModernLAPS-FA];**




## Create Application settings

Create some applications settings in the function app to contain sensitive strings required by the functions. The application settings are encrypted at rest and in transit and are exposed as environment variables at runtime. We can simply use the environment variables instead of storing sensitive information in the function code.

Note it is possible to call the Key Vault directly at runtime to retrieve the keys, but this means additional authentication must take place and will lengthen the execution time of the function.

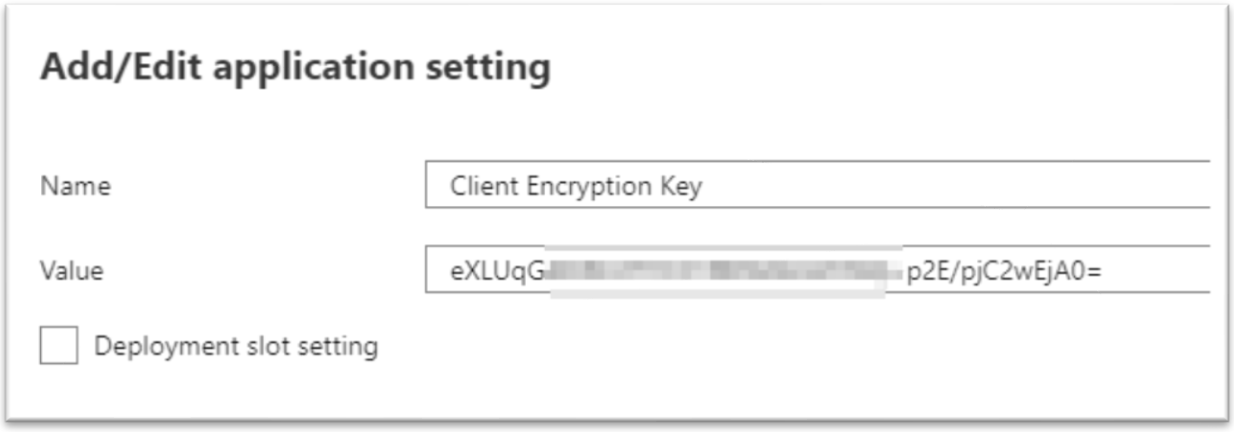
It is also possible to use Key Vault references however at the time of writing there is an issue where secrets are not updated in the function app when updated in the Key vault.

We need to create 5 applications settings:

1. Client Encryption Key
2. Client Encryption Key IV
3. Server Encryption Key
4. Server Encryption Key IV
5. SQL Connection String

To create an application setting:

- In the function app, click **Configuration**
- Click **New application setting**
- Enter the **name** and **value**. The values can be retrieved from the Key vault.
- Click **OK**

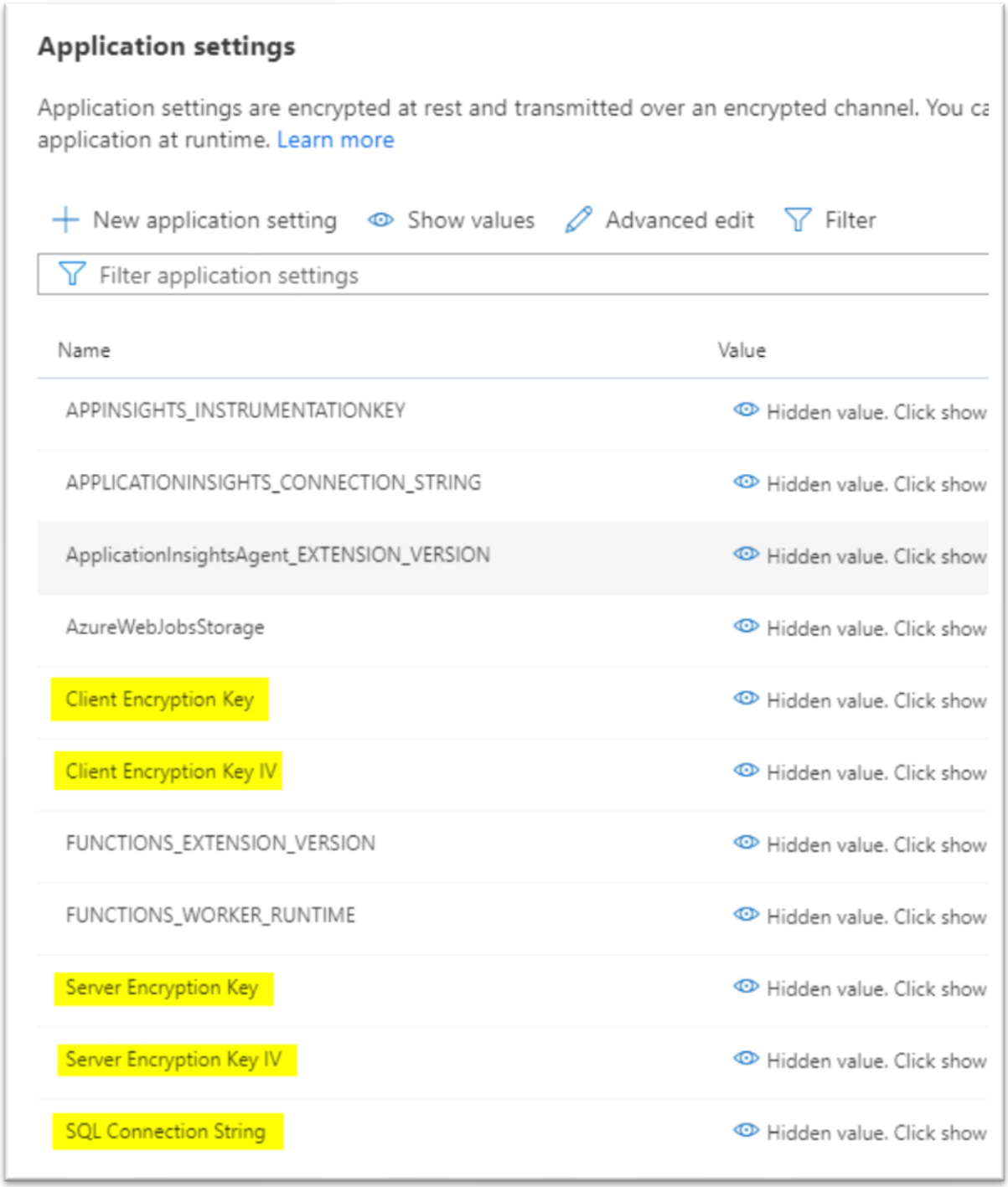



Create all 5 application settings. The "**Server**" encryption key settings are the "**Active**" encryption key secrets in the vault.

For the **SQL Connection string**, open **Azure SQL** in the portal. Click on the **database** you created. Click on **Connection strings**. Copy one of the **ADO.NET** connection strings, but **remove** any reference to a **User ID**, **Password** or **Authentication**. For example:

*Server=tcp:modernlapsdb.database.windows.net,1433;Initial Catalog=Modern LAPS;Persist Security Info=False; MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;*

When all the application settings are created, click **Save**.




## Create Functions

We will create two functions in the function app. The first function has an Event Grid trigger, and will be used to add local admin password changes to the database. The second has an HTTP trigger and will be used by the custom app to retrieve passwords from the database.

### Create the Event Grid Trigger function

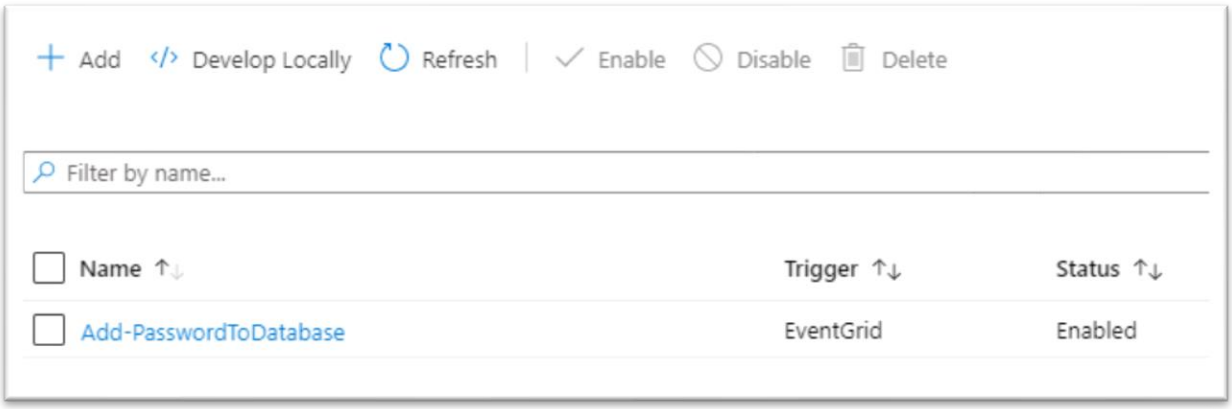
This function does the following:

1. Receives local admin password change data from the Event Grid service
2. Decrypts the local admin password using the Client key
3. Encrypts the local admin password using the Server key
4. Gets an access token for the database audience for the managed identity

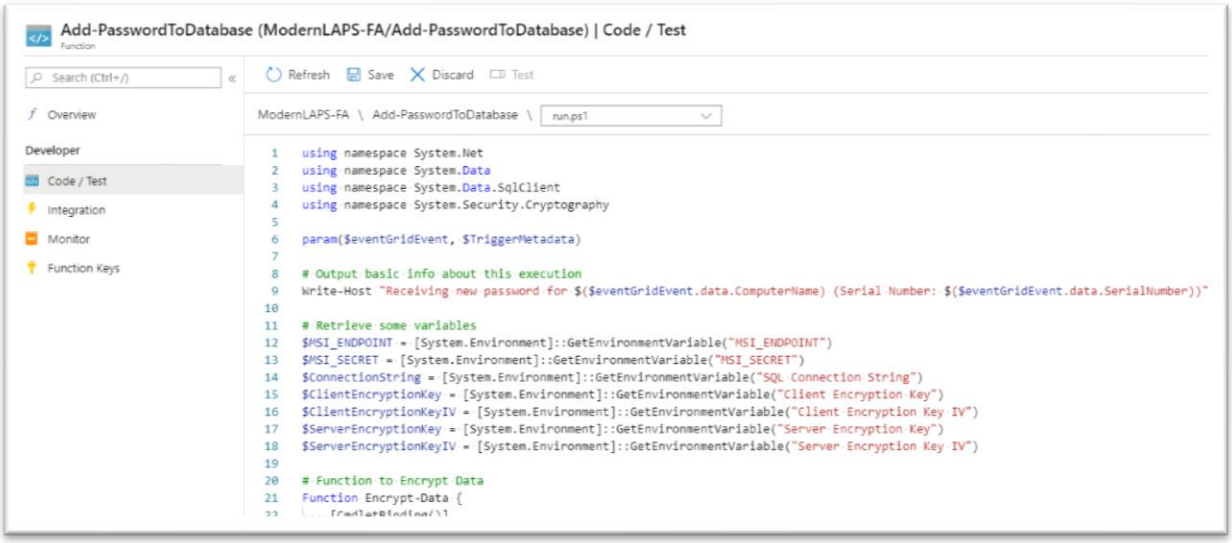
5. Inserts the data into the SQL database

To create the function:

- In the function app, click **Functions**
- Click **Add**
- Under **Templates**, select **Azure Event Grid trigger**.
- Give the function a **name** and click **Create function**




- Click on the function, then click **Code / Test**
- Paste the code from the script **Modern LAPS Function Update SQL Database** in the GitHub repo into the **run.ps1** file, overwriting any existing code.
- Click **Save**



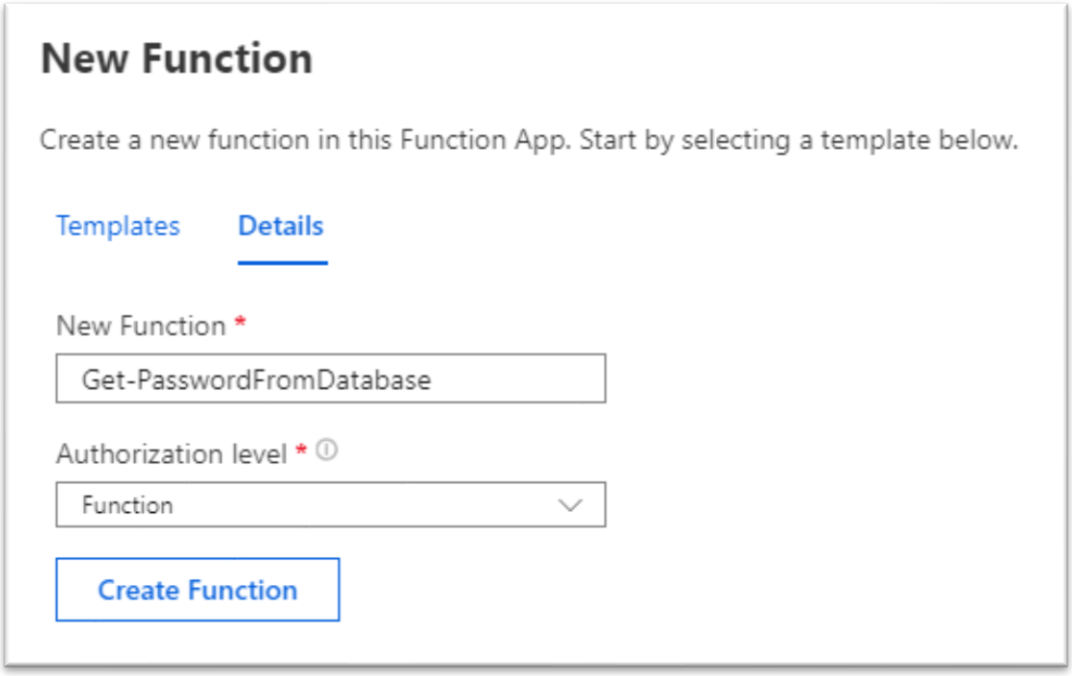

## Create the Http Trigger function

This function does the following:

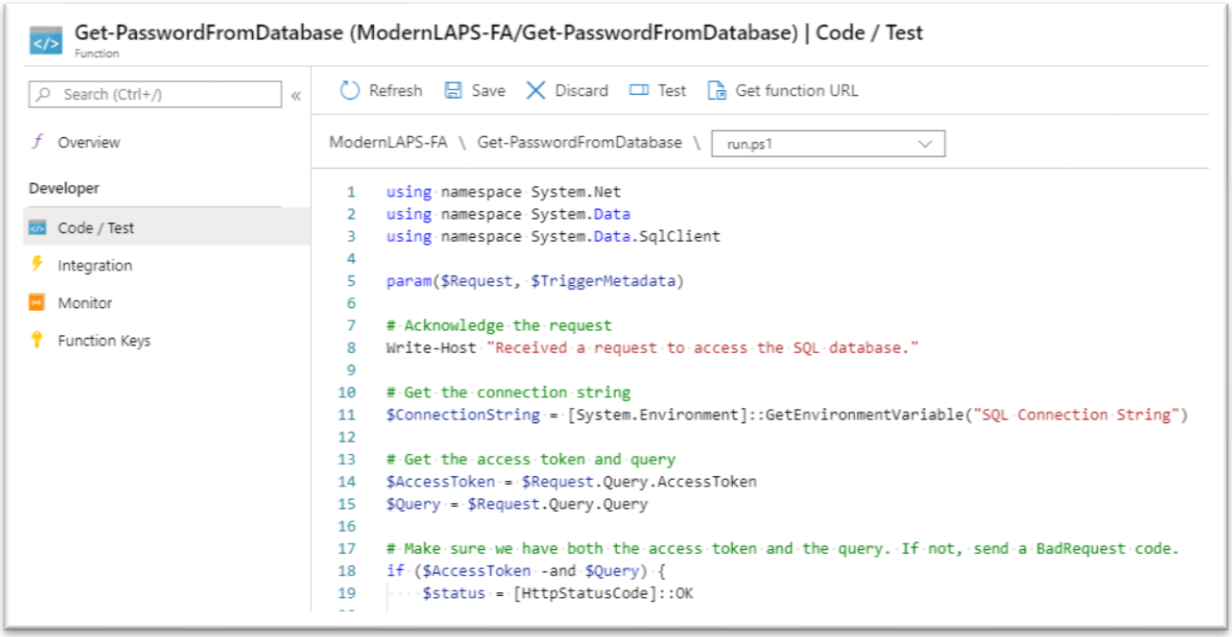
1. Receives an Http request from the custom app
2. Checks that the request contains a SQL authentication token for the user, and the query to run against the SQL server
3. Queries the database
4. Returns the result in the response

To create the function:

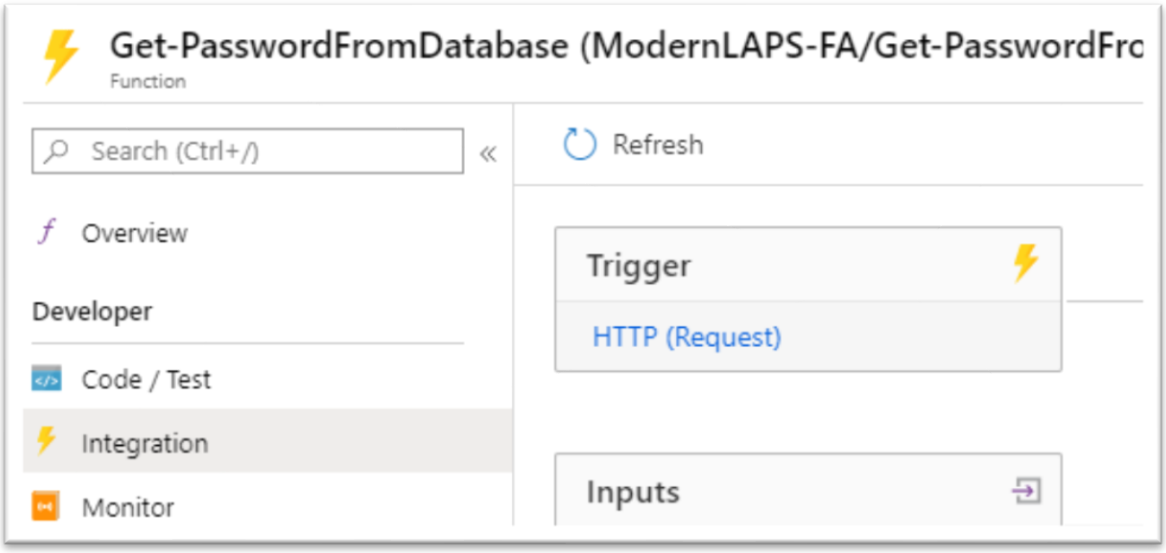
- In the function app, click **Functions**
- Click **Add**
- Under **Templates**, select **HTTP trigger**.
- Enter a **name** for the function
- Set the **Authorization level** to **Function**
- Click **Create Function**



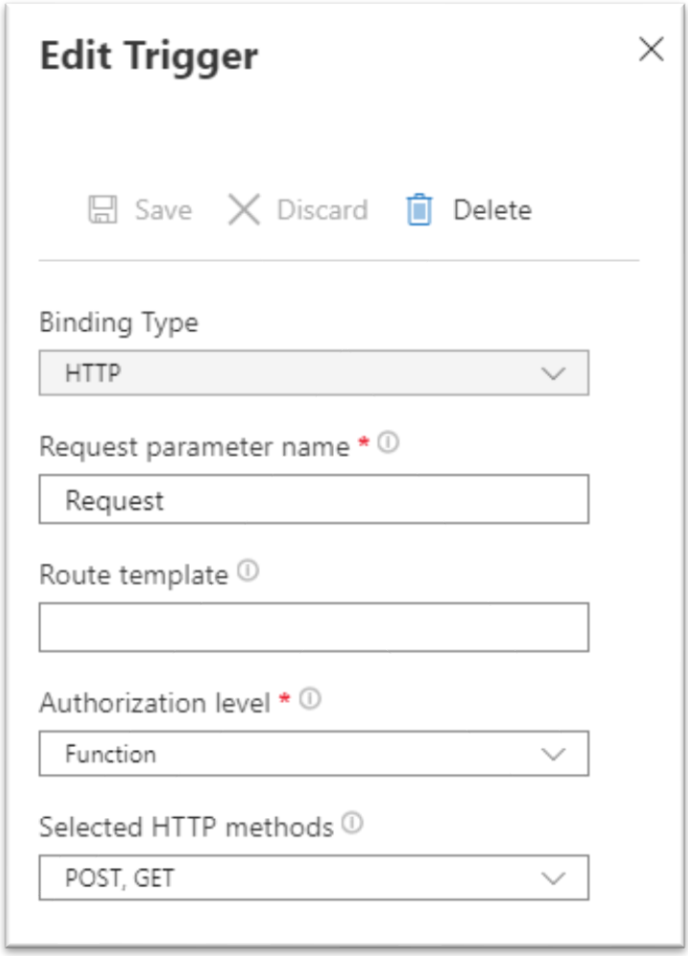

- Click on the function, then click **Code / Test**
- Paste the code from the script **Modern LAPS Function Retrieve SQL Data** in the GitHub repo into the **run.ps1** file, overwriting any existing code.
- Click **Save**




- Click on **Integration**, the click on the **HTTP (Request) Trigger**




- In the **Selected HTTP Methods**, make sure **POST** is selected.
- Click **Save**.



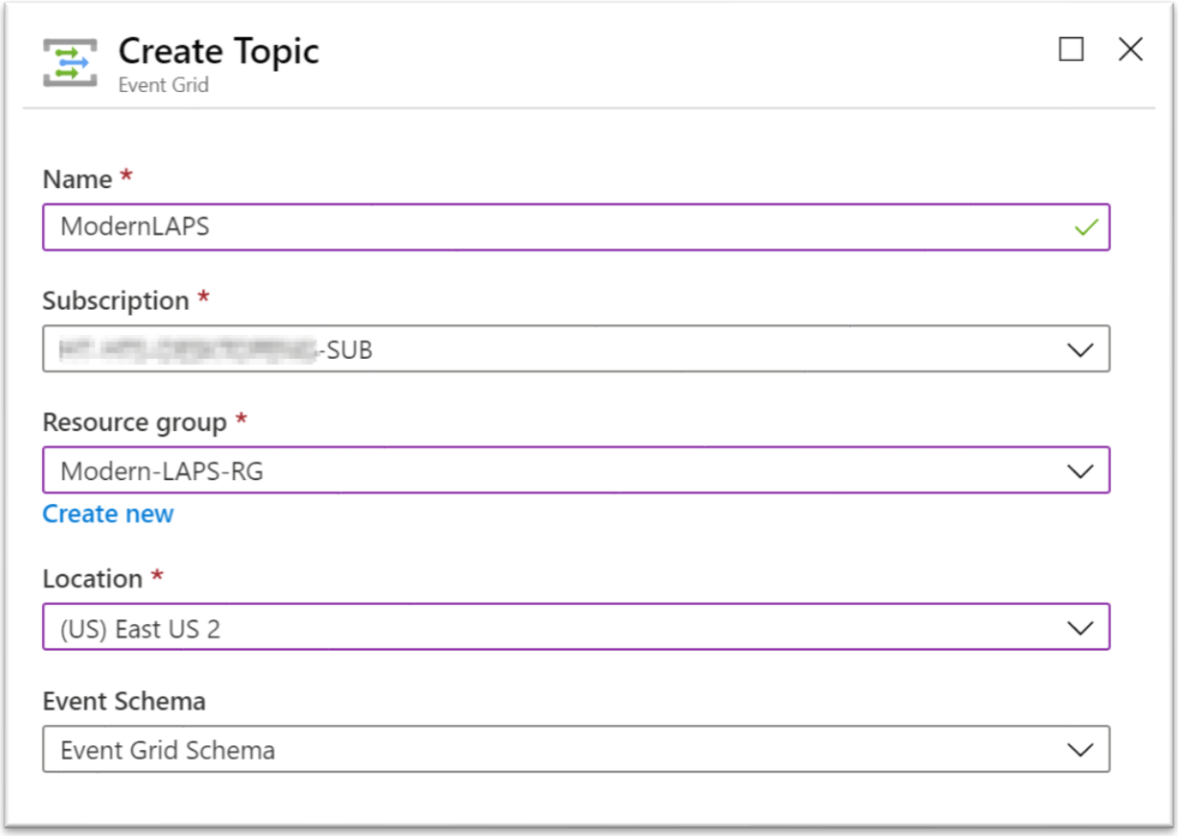
# Create an Event Grid Topic and Subscription

An Event Grid topic is an endpoint to which clients can send local admin password change data. A subscription routes the event data to an event handler, which in this case is an Azure function.

## Create the Topic

In the Azure portal, search **Event Grid Topics**

- Click **Add**
- Enter a **name**
- Select your **subscription**
- Select your **resource group**
- Select the **location**
- For the **event schema**, use the **Event Grid Schema**
- Click **Create**

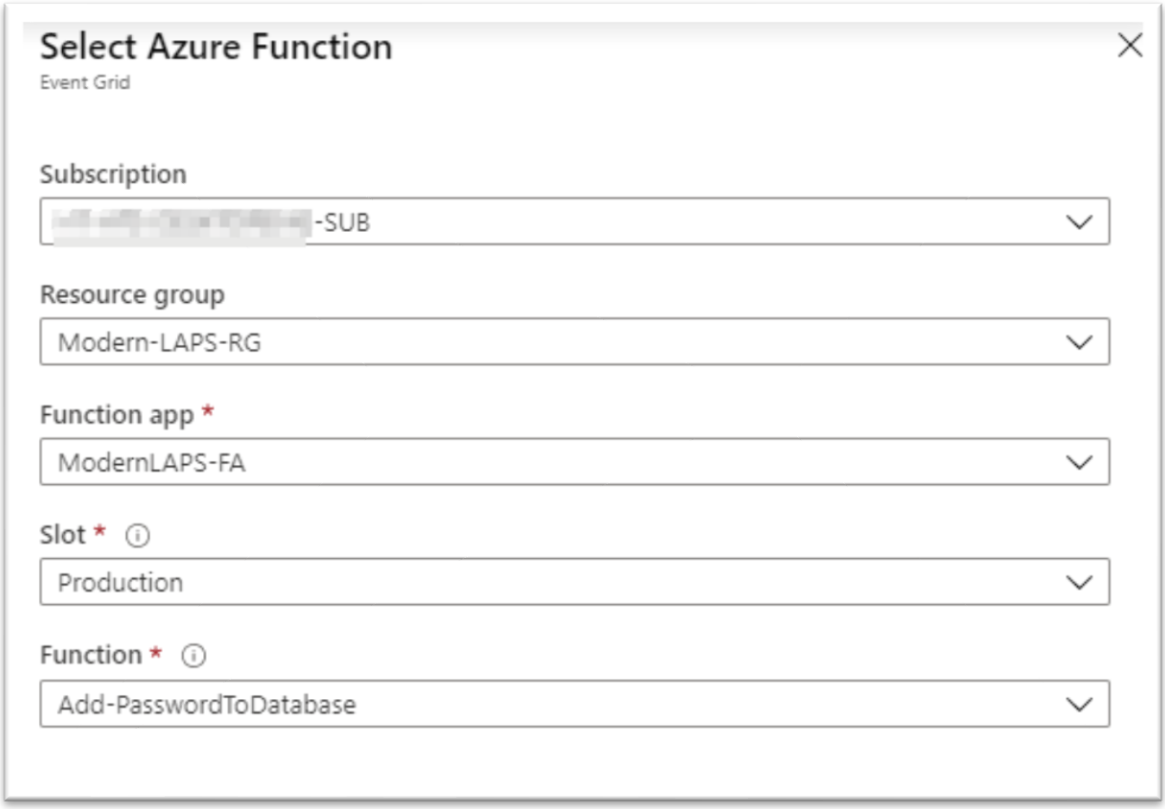



- Once created, select the Event Grid Topic

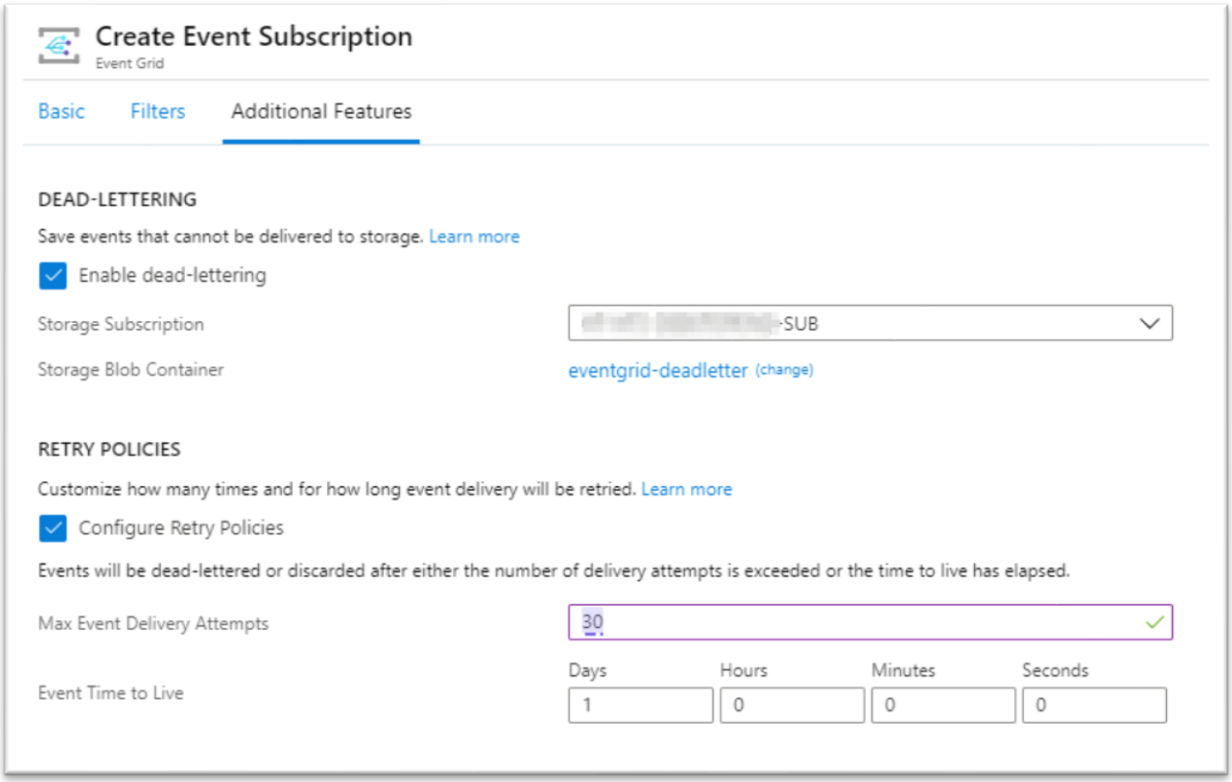
## Create the Subscription

- In the Event Grid Topic, click **+ Event Subscription**
- On the **Basic** page, enter a **name**
- Use the **Event Grid Schema**
- The **Topic details** are populated for you
- Under **Event Types**, do nothing
- For the **Endpoint Type**, select **Azure Function**
- Click **Select an endpoint**

- You should find all the parameters are populated for you. If not, select them being sure to choose the Azure function with the Event Grid trigger.
- Click **Confirm Selection**




- On the **Additional Features** page, you can optionally enable **dead-lettering**. This allows you to save any events that couldn't be delivered to a storage account. This means if a password change could not be delivered to the Azure function for whatever reason, you still have a record of it in storage.
    - You can also configure a **Retry policy**. You can set a maximum number of delivery attempts and how long before the undelivered event will expire.
    - You do not need to configure **Batching** unless you have a high number of workstations that may send data to the Event Grid at the same time.

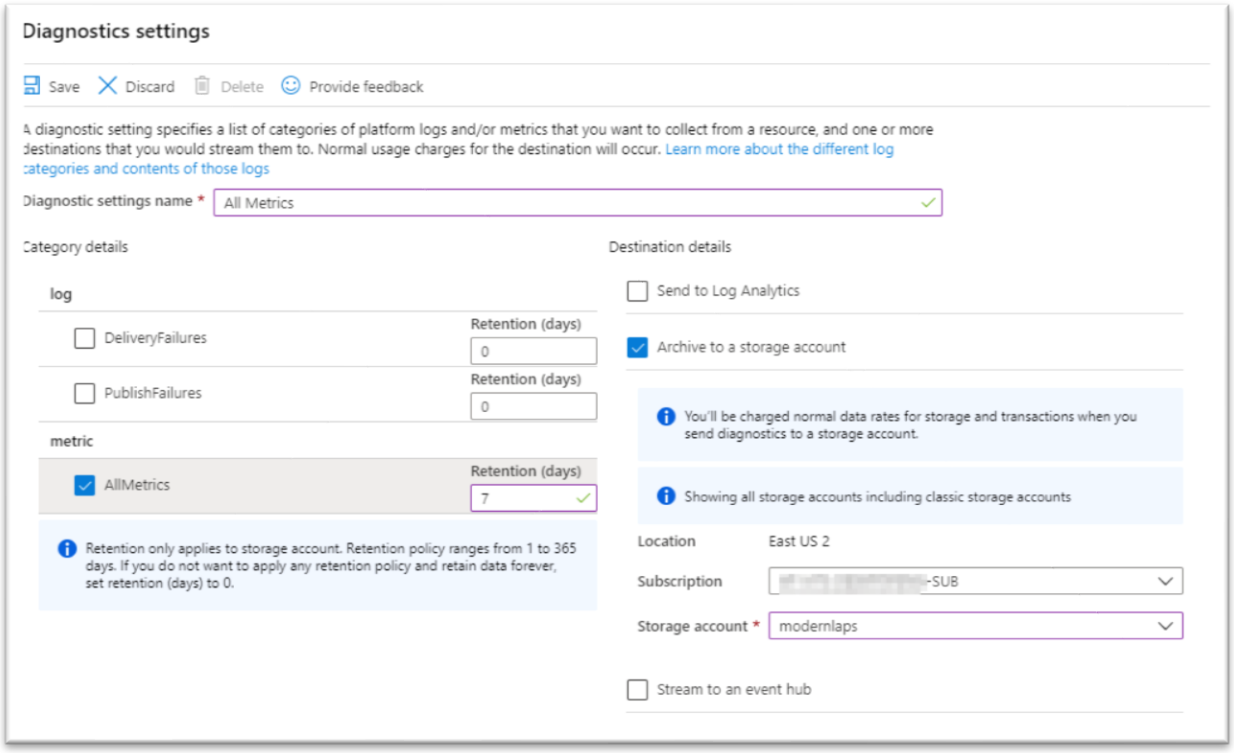



- Click **Create**

## Configure Diagnostic Settings

I highly recommend to add diagnostics settings so you can track delivery events.

- In the Event Grid Topic, click **Diagnostic settings**
- Click **Add diagnostic setting**
- Enter a **Diagnostics settings name**
- Select which **Categories** you want
- Select a **Destination**
- In the example below I have configured all metrics to go to a storage account with a 7-day retention period.




# Create Configuration Manager Baselines

We need to create two baselines in Configuration Manager. One will be used to push the password rotation script. The other will be used to stamp the current user registry with data required by the custom app for password retrieval, such as the Key vault endpoint, the secret names, the URL of the function app. These will be stored in the registry as encrypted strings for the user of the custom app. The required values are not hard-coded into the app, therefore they need to be retrieved at runtime.

## Create the Password Rotation Baseline

### Prepare the remediation script

From the GitHub repo, open the script **Modern LAPS Password Rotator** for editing. You will notice at the top of the script is a **Parameters** section. Let's go through the parameters you need to populate.

***LocalAccountName***

This is the name of the local account you wish to rotate the password for. The default value in the script gets the name of the built-in local administrator account using its SID. Even if the account has been renamed, the SID will be the same. You could, if you wish, enter the name of another local account that has administrator privileges, however only one account can be entered.

***PasswordRotationFrequency***

This is how frequently in days you wish to rotate the local administrator password.

***EventLogSource***

This is the name of the source that will be used to log events to the application event log. Change the name if you wish.

***EventGridTopicEndpoint***

This is the URL of the Event Grid Topic endpoint that you created earlier. To retrieve it, in the Azure portal, search for Event Grid Topics. Select the topic you created. In the **Overview** pane you will find the **Topic Endpoint**. Copy and paste this into the script.

***EventGridTopicKey***

This is the access key used to authenticate with the topic endpoint. You can find this in the **Event Grid Topic** under **Access Keys**. Use **Key 1**. Access Keys can be periodically regenerated to increase security as described in the Operations guide. Copy and paste the key into the script.

***ClientEncryptionKey***

This is the Client Encryption Key you created and stored in the Azure Key Vault. Open the **Key Vault** and go to **Secrets**. Click on the secret **Client-Encryption-Key**. Open the current version. Copy and paste the secret value into the script.

***ClientEncryptionKeyIV***

This is the initialization vector for the Client encryption key. Retrieve it from the vault in the same was as described above.

***PasswordLength***

This is the number of characters that the new password should contain.

***PasswordRegex***

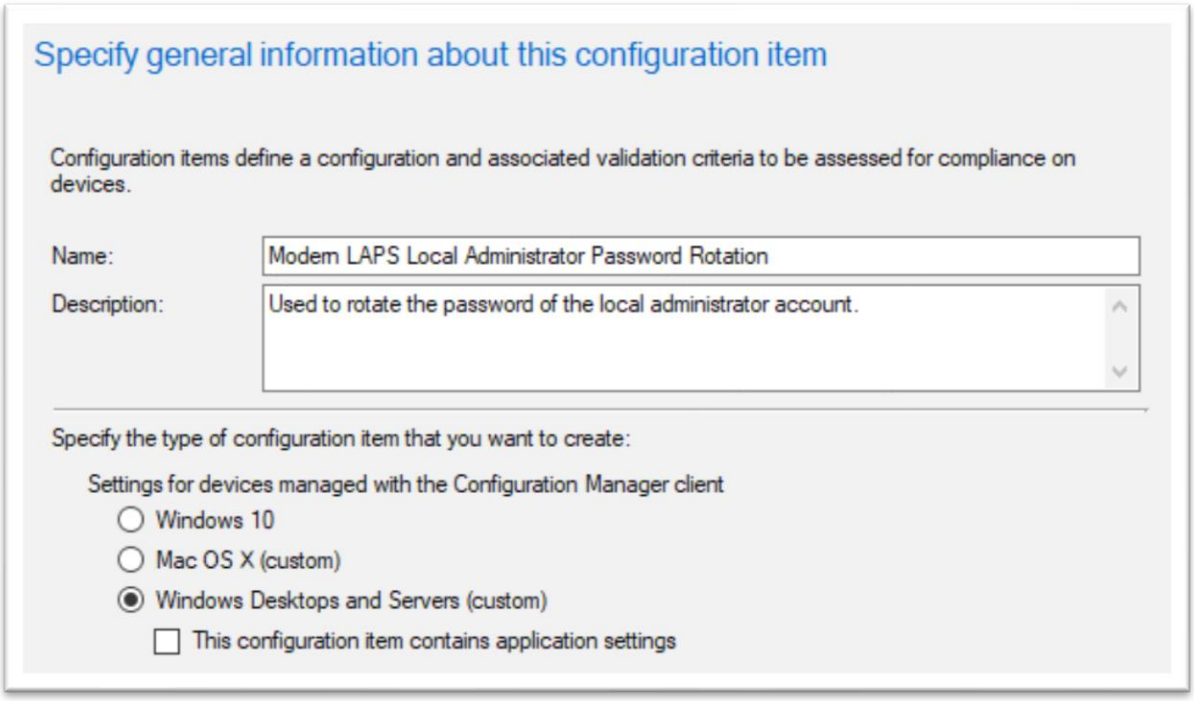
This is a regular expression that defines the complexity of the password. The default expression uses the following criteria:

- At least 1 upper case letter
- At least 1 lower case letter
- At least 1 number or special character
- At least as long as the defined password length

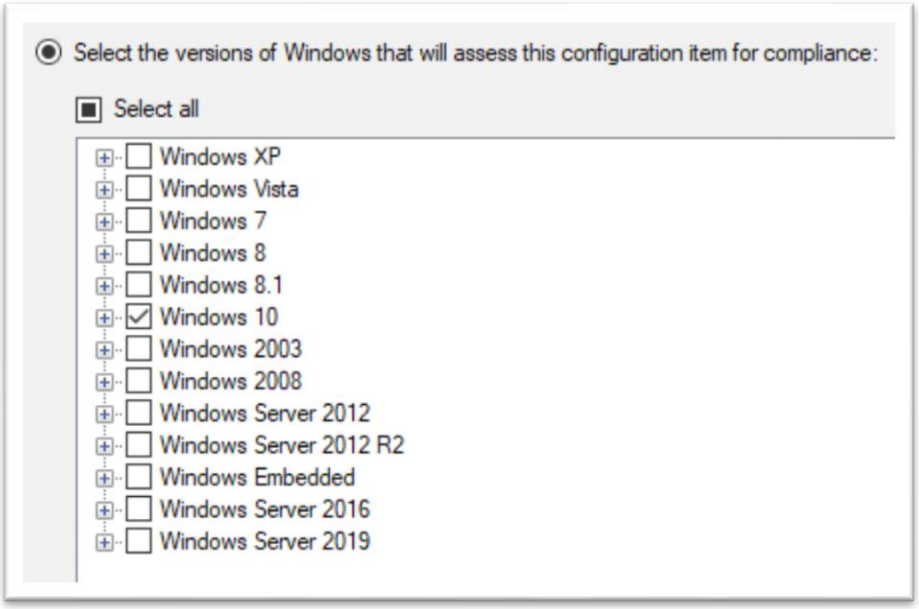
The remaining parameters are for the location in the registry where script execution data will be stored. You do not need to change these values.

## Create a Configuration Item

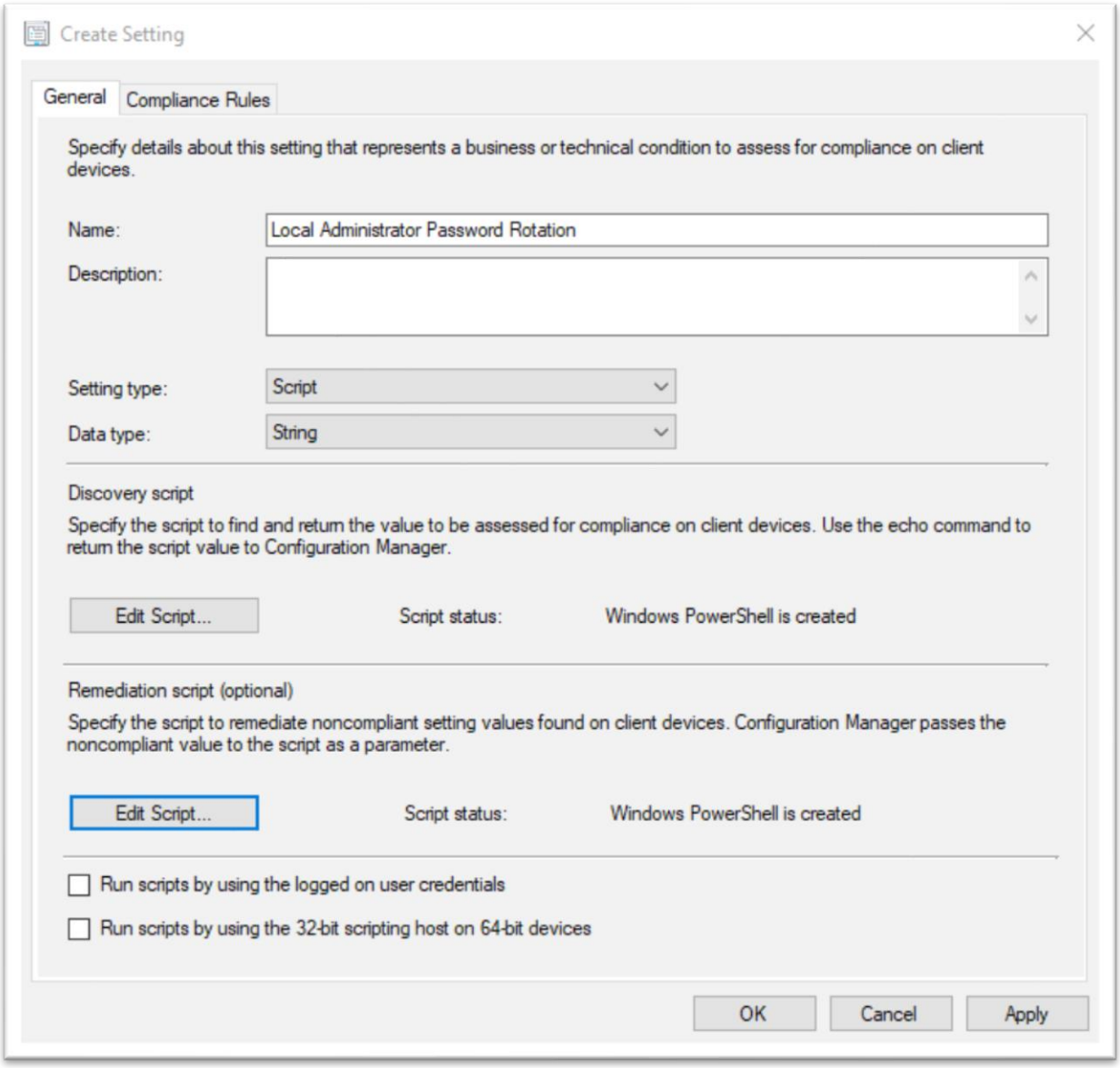
- In the **Configuration Manager console**, create a new **Configuration Item**
- Give it a **name**, **description** and select **Windows Desktops and Servers** as the type. Click **Next**



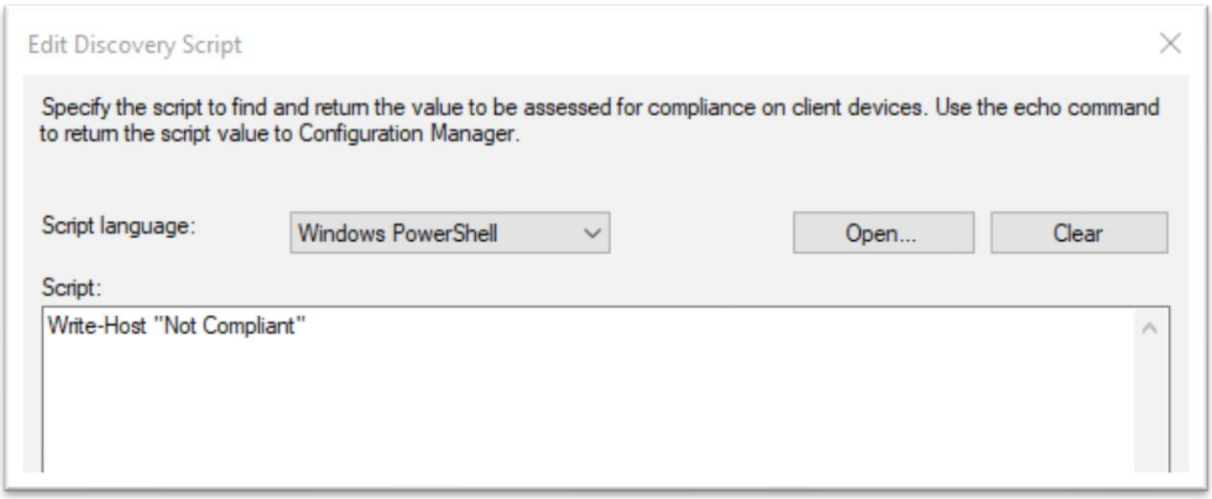

- Select only **Windows 10** for the **Windows version**. Click **Next**



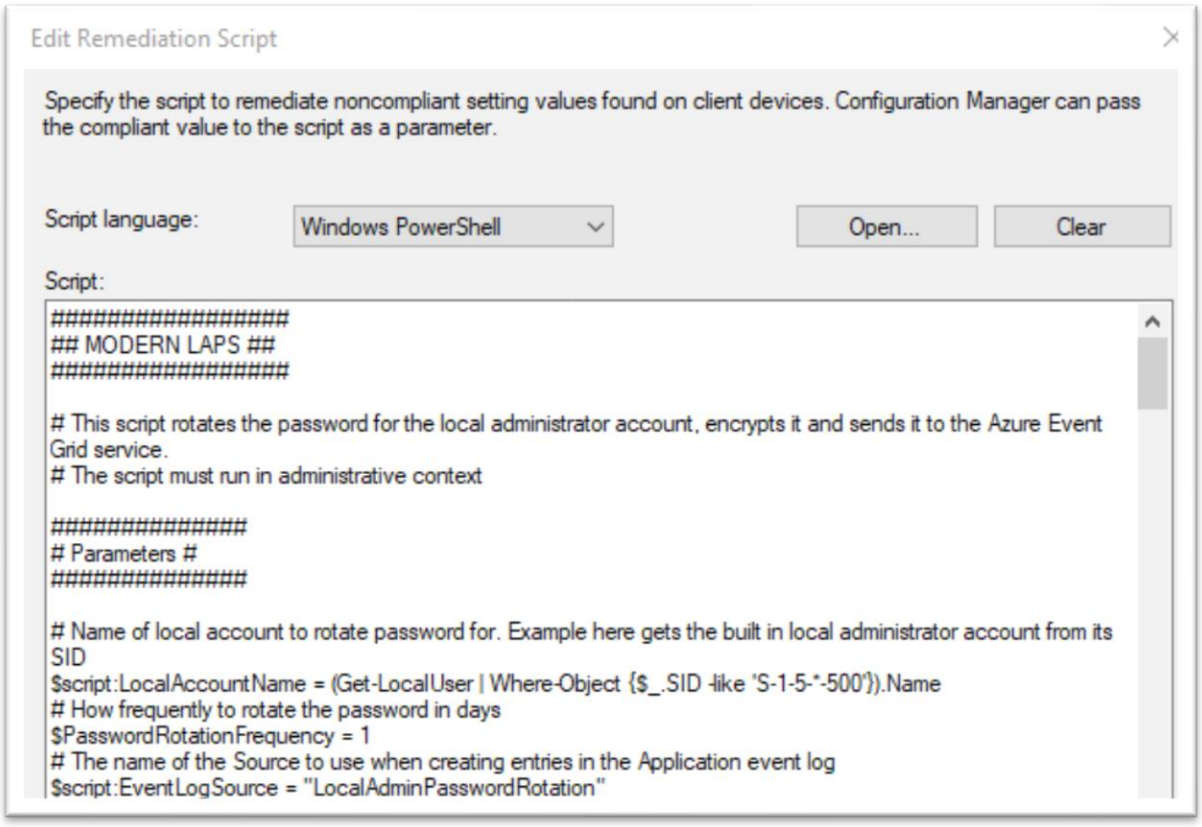

- On the **Settings** page, click **New**
- Enter a **name**
- For the **Setting type**, choose **Script**
- For the **Data type**, choose **String**



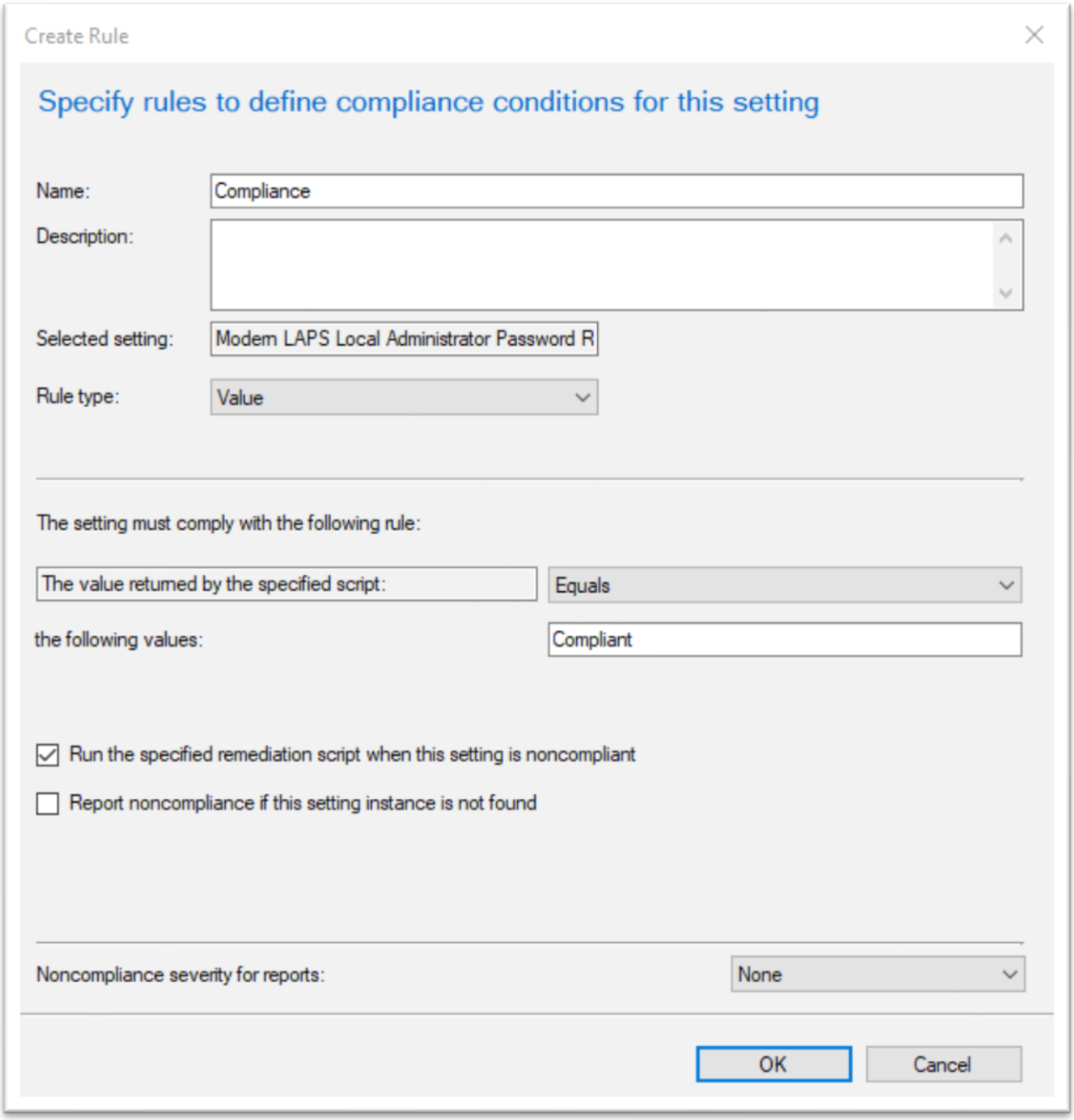

- For the **Discovery script**, click **Edit Script…** and simply enter **Write-Host "Not Compliant"** as a PowerShell script.



- Click **OK** to close.
- For the **Remediation script**, click **Edit Script…** and paste the content of the **Modern LAPS Password Rotator** script that you edited in the previous section. Click **OK**.



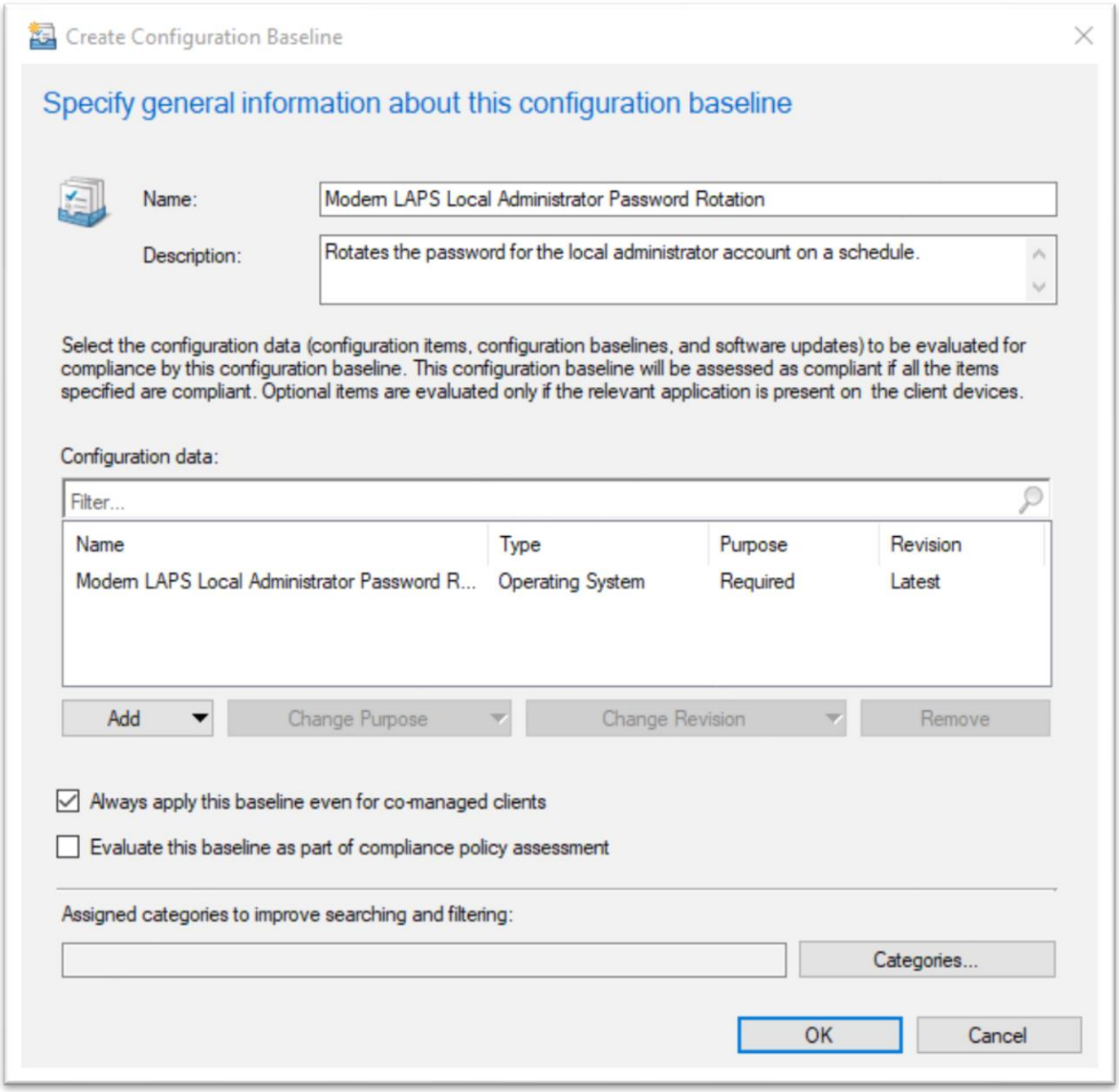

- Click on the **Compliance Rules** tab and click **New**
- Enter a **name**
- Select **Rule type**: **Value**
- Enter **Compliant** in the value box
- Check the option **Run the specified remediation script when this setting is noncompliant**.
- Click **OK**




- Click **OK** again to save the setting
- Click **Next, Next, Next** and **Close** to save the item.

## Create a Configuration Baseline

- In the **Configuration Manager console**, create a new **Configuration Baseline**
- Give it a **name** and **description**
- Under **Configuration data**, click **Add > Configuration items**
- Select the Configuration item you created before
- Check the box **Always apply this baseline even for co-managed clients**
- Click **Ok** to save the baseline.

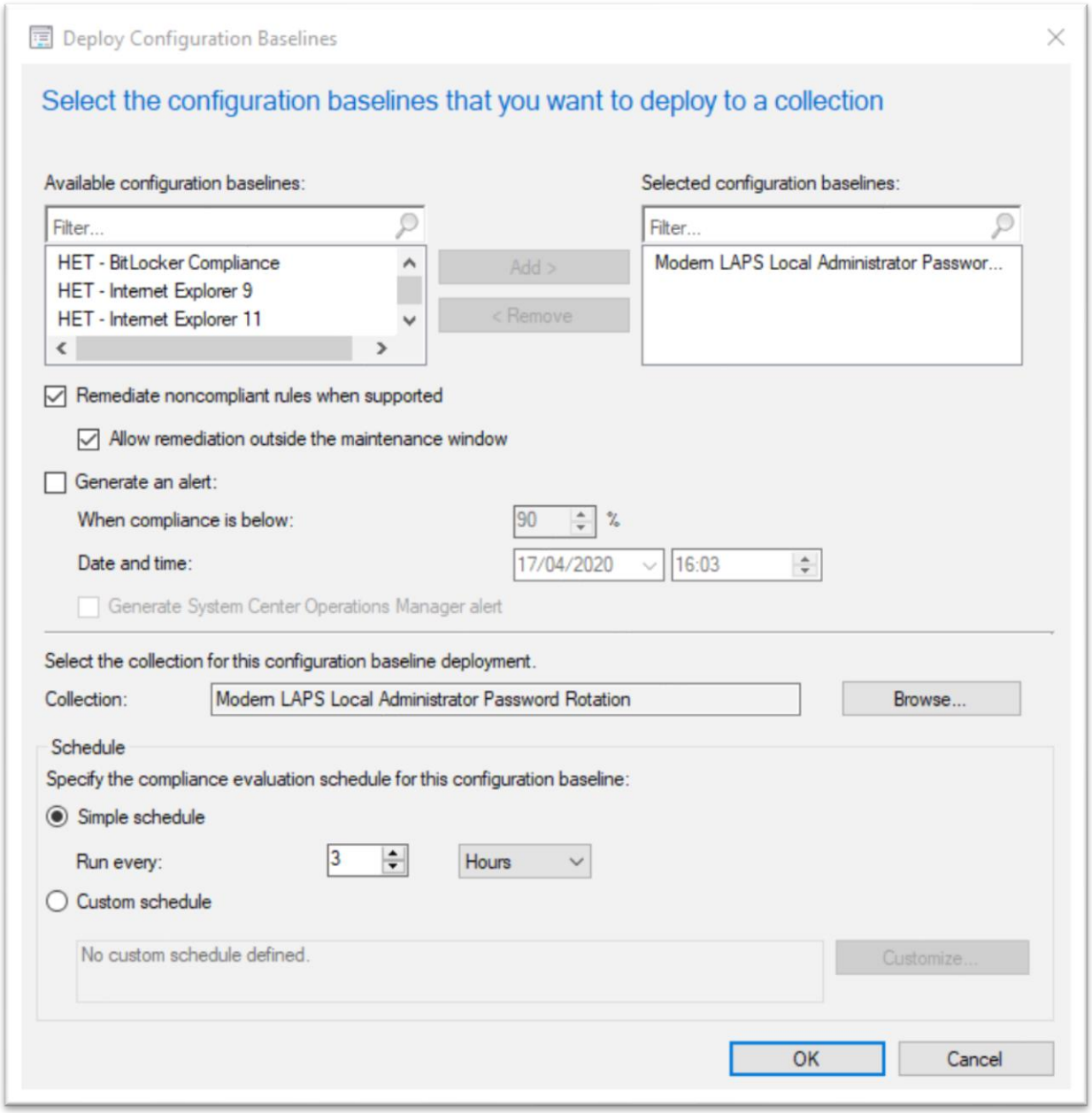



## Deploy the Configuration Baseline

- Right-click the baseline and **Deploy**
- Check the option **Remediate noncompliant rules when supported**
- Check the option **Allow remediation outside the maintenance window**
- Select the **device collection** you want to target
- Set the **schedule** – I recommend every 3 hours
- Click **OK** to save.

Note that the schedule the baseline evaluates with is **not** the schedule that the password will be rotated with – that schedule is set in the remediation script with the **PasswordRotationFrequency** parameter. The evaluation schedule should run more frequently than that. The reason is that if a password is rotated but it couldn't be uploaded at the same time (perhaps because of no internet connectivity, for example), the script will try to upload it again then next time the baseline evaluates.

Also note that baselines do not evaluate exactly on schedule but use a randomization window.

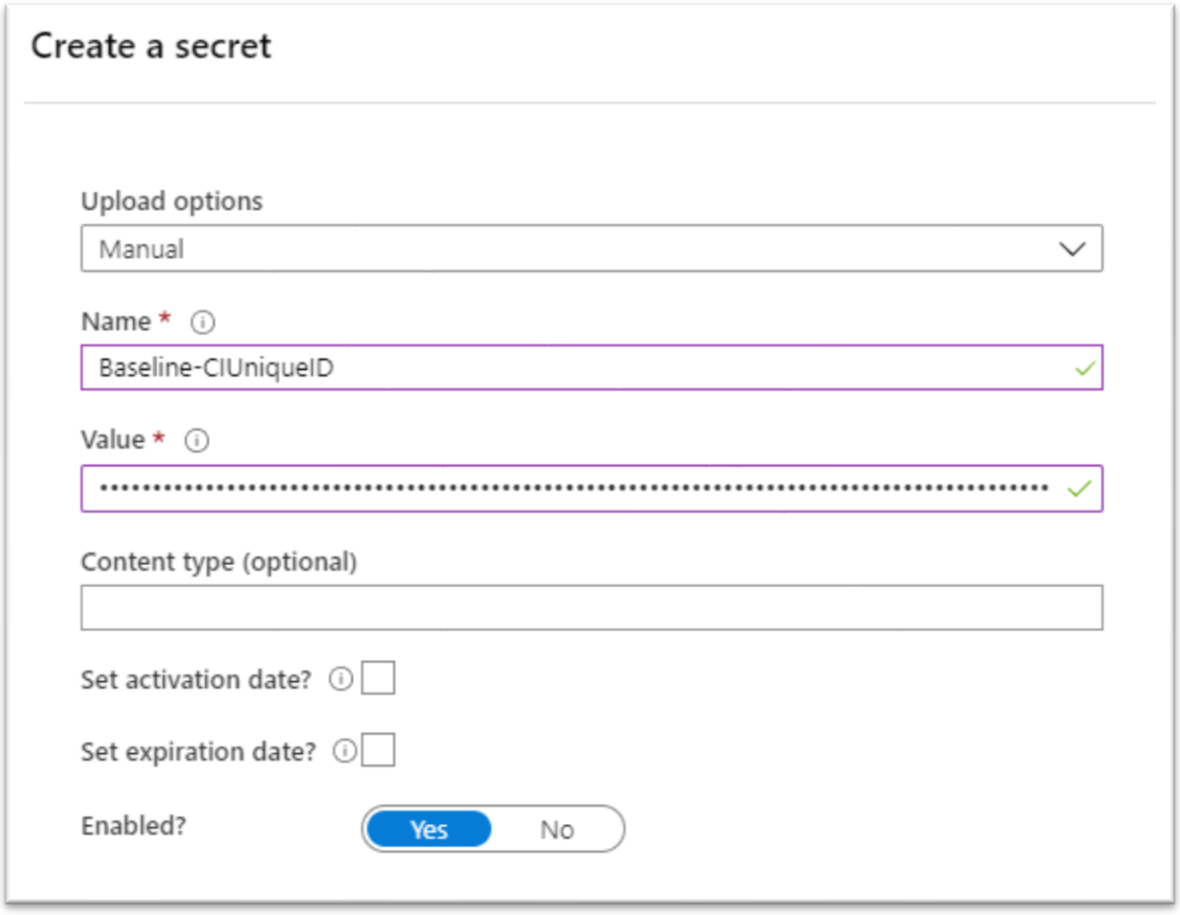



## Add the Baseline CI Unique ID to the Key Vault

The Modern LAPS Manager app will use the **CI Unique ID** of the baseline when the option to manually rotate the password is used.

To retrieve this value, navigate to the baseline in the ConfigMgr console. **Right-click** any column header and add the **CI Unique ID** column. Select the baseline and **Ctrl-C** to copy. Paste the text somewhere and copy just the **CI Unique ID**.

In the Azure portal, navigate to the **Key Vault** you created earlier. Add a new **secret** called **Baseline-CIUniqueID** and paste in the **CI Unique ID value.**




## Create the Custom App Settings Baseline

This baseline will be used to ensure that the parameters required for the **Modern LAPS Manager app** are present on the local system. These parameters are unique to your environment. Note that before a user can use the Modern LAPS Manager app, this baseline needs to have been deployed and run so the app can retrieve what it needs from the Azure services.

### Prepare the Discovery and Remediation Scripts

In the GitHub repo there are two scripts for this baseline:

1. **Modern LAPS Manager User Registry Keys Detection**
2. **Modern LAPS Manager User Registry Keys Remediation**

In each script is a set of values from your Azure environment in the hash table **$DataHash**. These values must be retrieved and populated from your Azure environment. The values are the same in each script, and no other values need to be changed in the script.

```powershell
#########################
## MODERN LAPS MANAGER ##
#########################

# This script checks that the HKCU registry keys required by the Modern LAPS Manager app are present with the correct values
# It is intended to be used a a detection script for a Configuration item in Configuration Manager
# This script must run in the user context


# Reg Key Values
$DataHash = @{
    KeyVaultURL = "https://[redacted]-keyvault.vault.azure.net/"
    KeyVaultAPIVersion = "api-version=7.0"
    ActiveEncryptionKeyName = "Active-Encryption-Key"
    ActiveEncryptionKeyIVName = "Active-Encryption-Key-IV"
    PreviousEncryptionKeyName = "Previous-Encryption-Key"
    PreviousEncryptionKeyIVName = "Previous-Encryption-Key-IV"
    BaselineCIUniqueIDName = "Baseline-CIUniqueID"
    ClientAppID = "[redacted]-46dd-a05b-41222f15e24f"
    ClientAppRedirectURI = "https://login.live.com/oauth20_desktop.srf"
    FunctionURL = "[redacted].azurewebsites.net/api/Get-PasswordFromDatabase?code=rdYNdfi4Sg3w3[redacted]JO2Yx3YS4w=="
}

# Reg key names
$RegBase = "HKCU"
$RegBranch = "Software"
$AppVendor = "SMSAgent"
$AppName = "Modern LAPS Manager"
```

## SCRIPT VALUES

***KeyVaultURL***

This is the **URL** of you **Azure Key Vault**. In the Azure portal, navigate to your Key Vault. On the **Overview** pane, copy the **DNS Name** into the script (with or without the trailing slash).

***KeyVaultAPIVersion***

This is the current version of the API which at the time of writing is "**api-version=7.0**". See https://docs.microsoft.com/en-us/rest/api/keyvault/getsecret/getsecret.

***ActiveEncryptionKeyName***
***ActiveEncryptionKeyIVName***
***PreviousEncryptionKeyName***
***PreviousEncryptionKeyIVName***
***BaselineCIUniqueIDName***

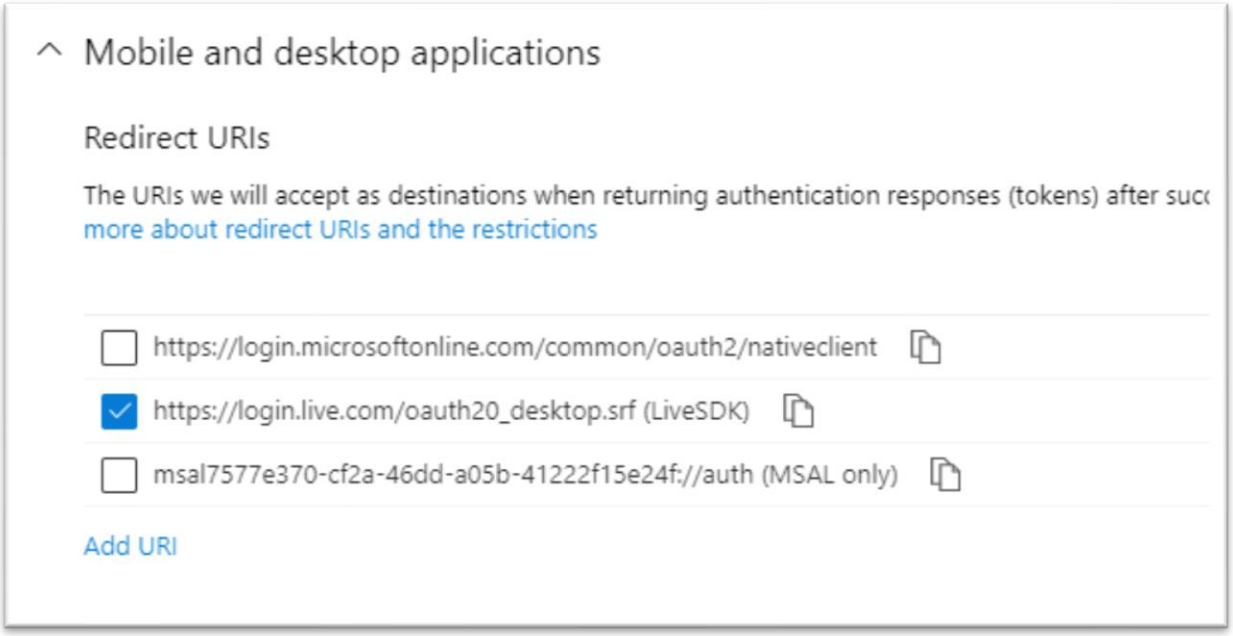
These values are the **names** of the secrets in your Key Vault. You do not have to use the same names for the secrets as I have, but you must provide whichever names you use to the script.


***ClientAppID***

In the Azure portal, search for **App registrations**. Locate and open the app registration you created earlier (eg **Modern LAPS Manager**). In the **Overview** pane, copy and paste the **Application (client) ID** value (GUID) into the script.

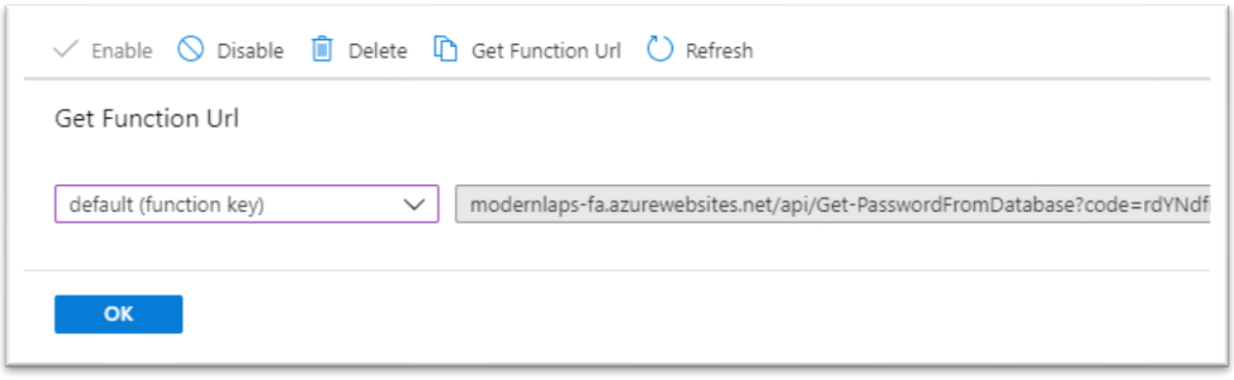
***ClientAppRedirectURI***

If you created the app registration as detailed earlier in this guide, the **Redirect URI** will be **https://login.live.com/oauth20_desktop.srf**. You can verify this in the app registration and the **Authentication** pane. The URI should be selected.



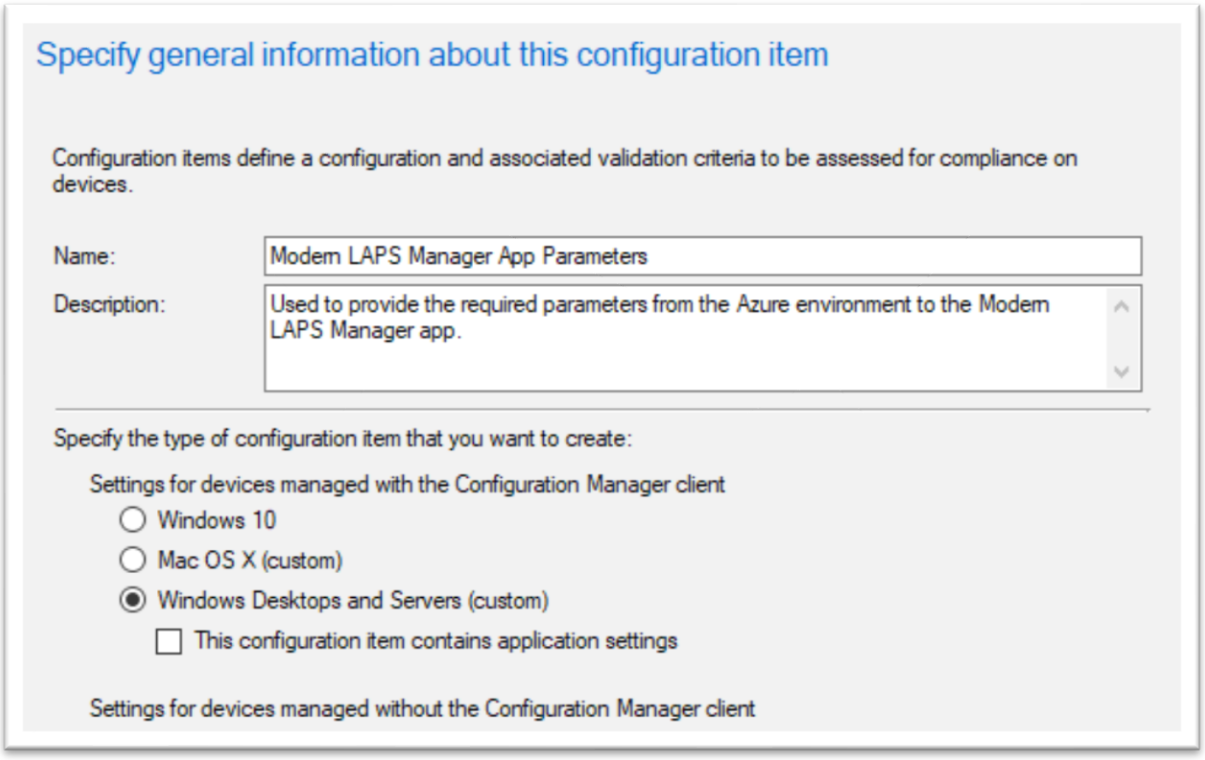

***FunctionURL***

In the Azure portal, search for **Function app**. Open the function app you created earlier. Click on **Functions** and click on the function with the **HTTP trigger** (eg **Get-PasswordFromDatabase**). Click on **Get Function Url** and be sure to select the **default (function key)** (it is not the default option). Copy and paste the URL into the script. Note that the URL contains an access key that is unique to that function.

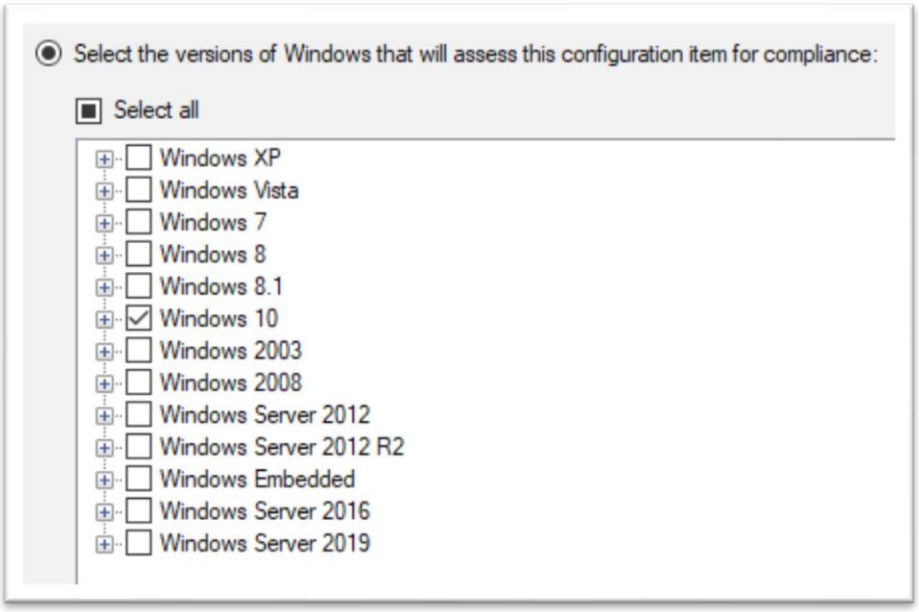



## Create a Configuration Item

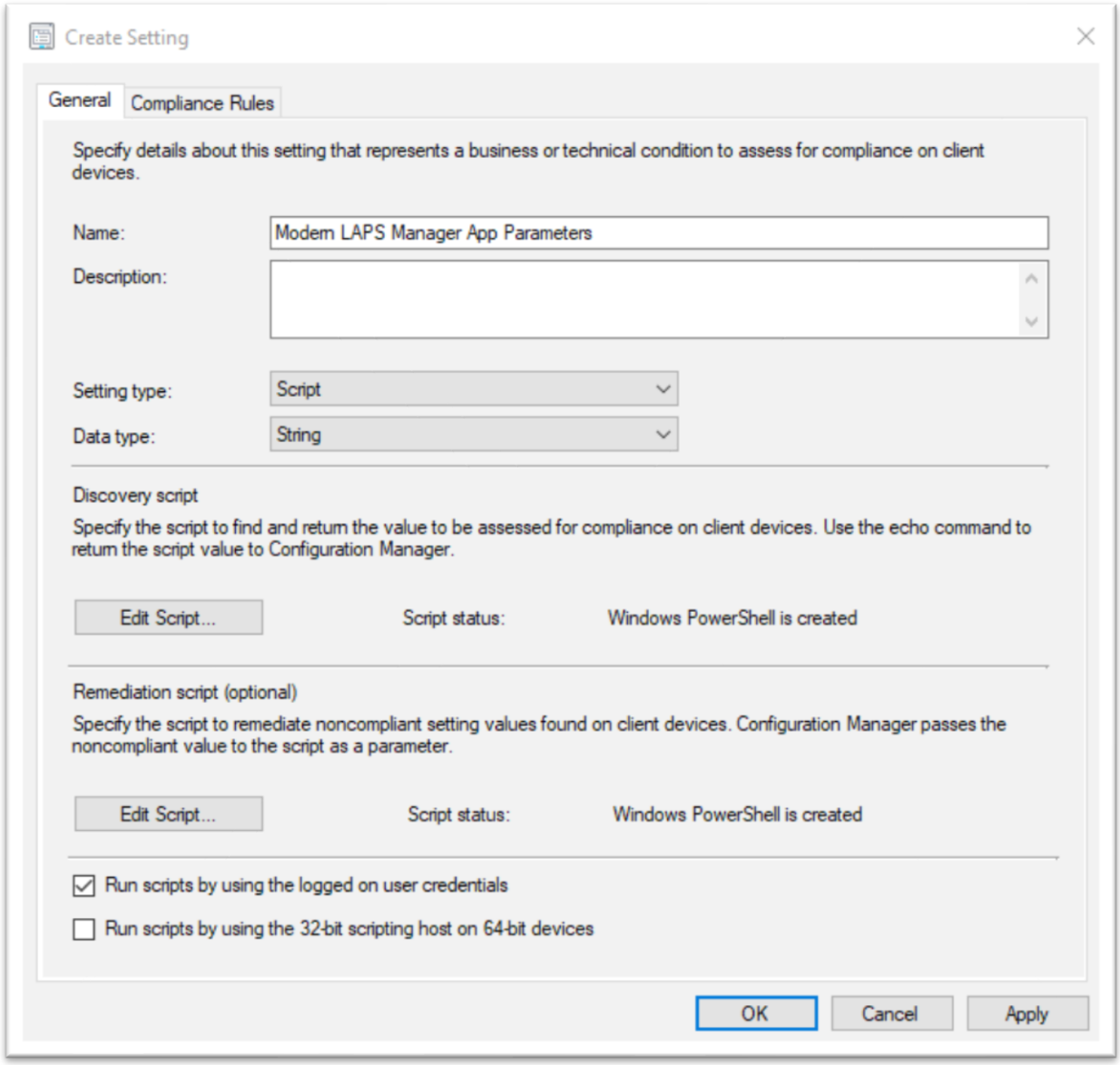
- In the **Configuration Manager console**, create a new **Configuration Item**
- Give it a **name**, **description** and select **Windows Desktops and Servers** as the type. Click **Next**



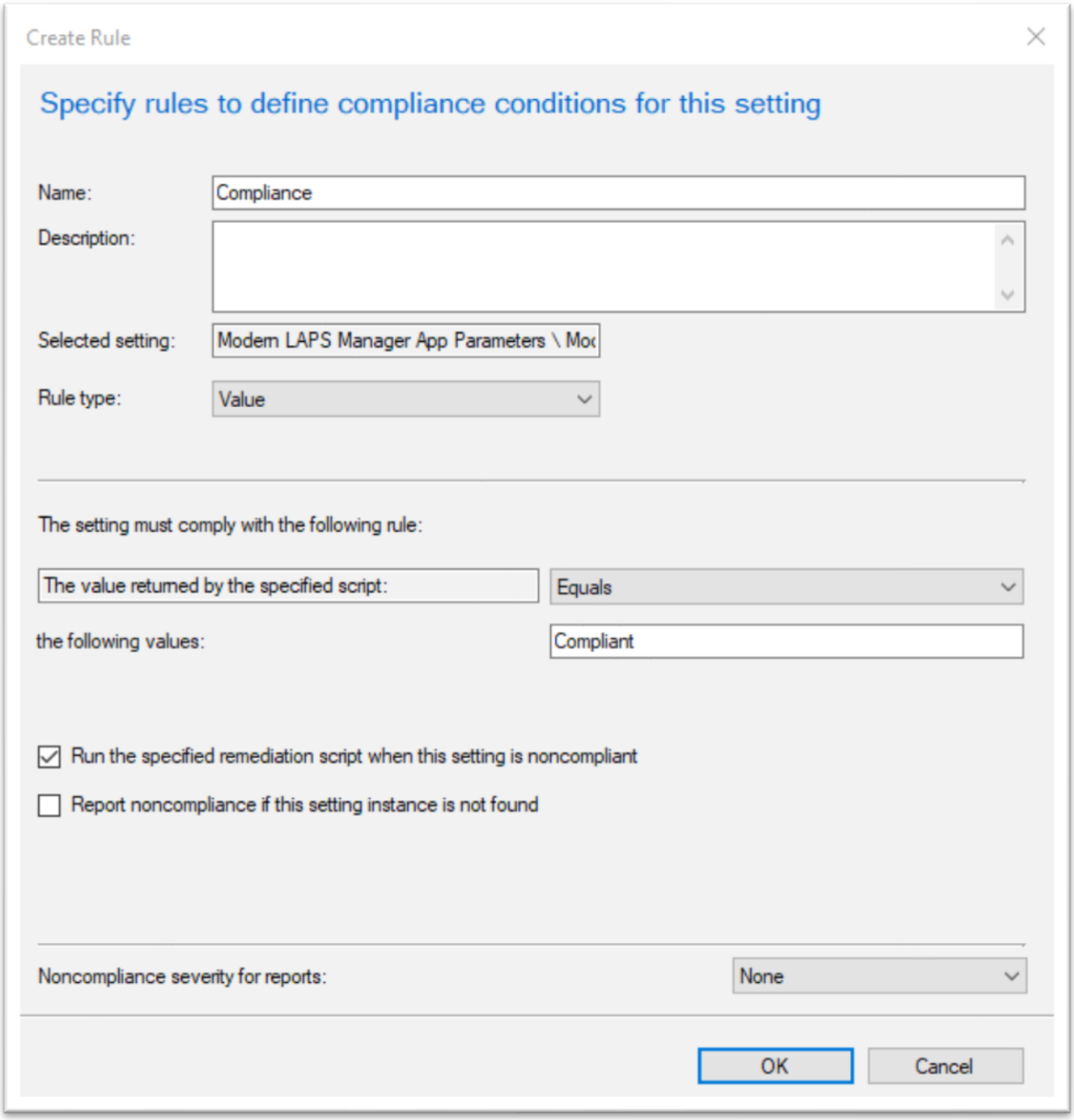

- Select only **Windows 10** for the **Windows version**. Click **Next**




- On the **Settings** page, click **New**
- Enter a **name**
- For the **Setting type**, choose **Script**
- For the **Data type**, choose **String**
- Be sure to check the box **Run scripts by using the logged on user credentials**



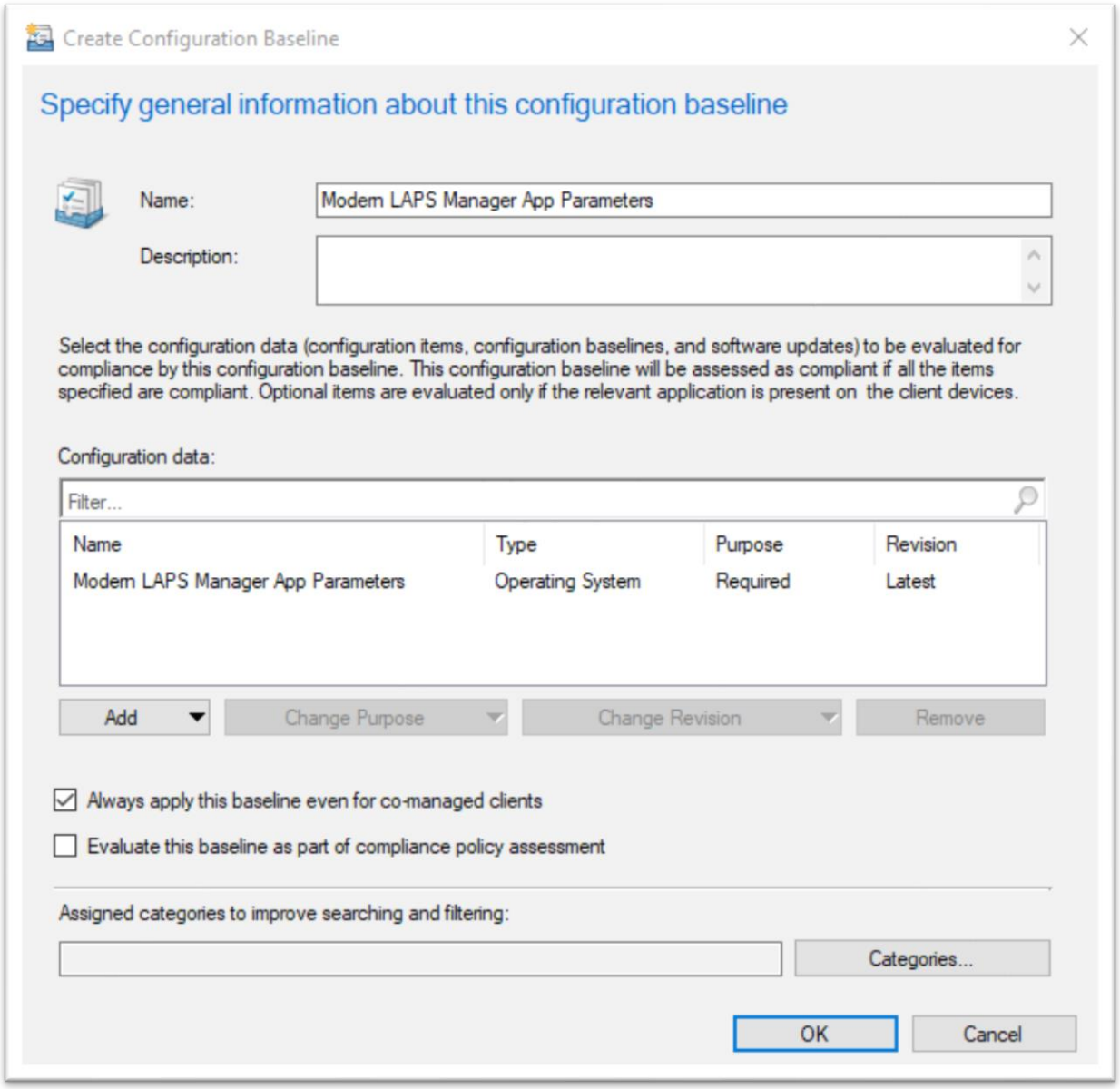

- For the **Discovery script**, click **Edit Script…** and paste the content of the edited **Modern LAPS Manager User Registry Keys Detection** script as a PowerShell script.
- For the **Remediation script**, click **Edit Script…** and paste the content of the edited **Modern LAPS Manager User Registry Keys Remediation** script as a PowerShell script.
- Click on the **Compliance Rules** tab and click **New**
- Enter a **name**
- Select **Rule type**: **Value**
- Enter **Compliant** in the value box
- Check the option **Run the specified remediation script when this setting is noncompliant**.
- Click **OK**




- Click **OK** again to save the setting
- Click **Next, Next, Next** and **Close** to save the item.

## Create a Configuration Baseline

- In the **Configuration Manager console**, create a new **Configuration Baseline**
- Give it a **name** and **description**
- Under **Configuration data**, click **Add > Configuration items**
- Select the Configuration item you created before
- Check the box **Always apply this baseline even for co-managed clients**
- Click **Ok** to save the baseline.

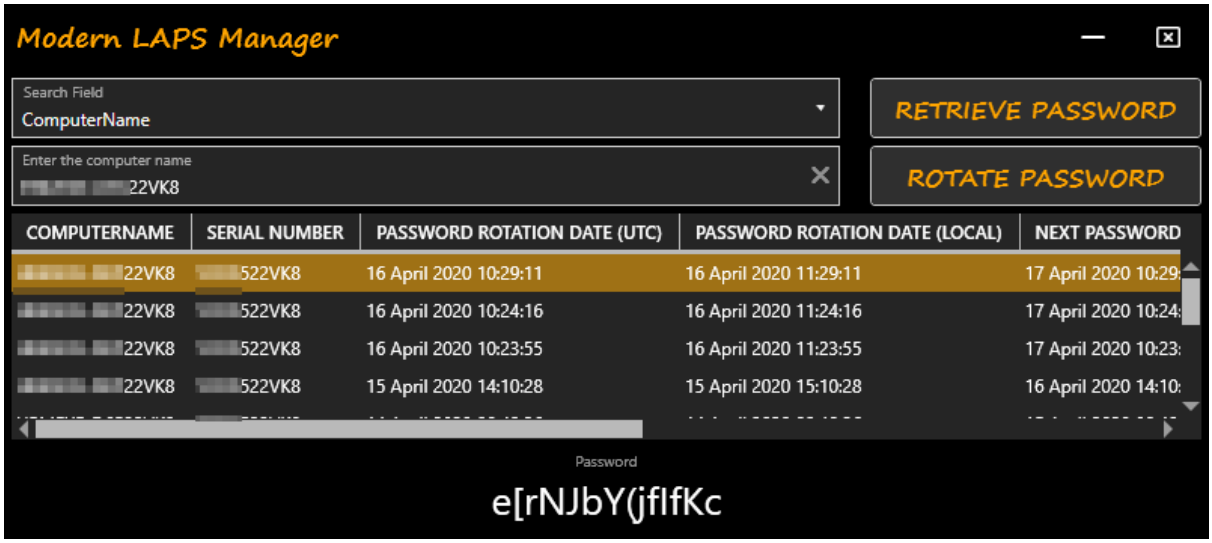



## Deploy the Configuration Baseline

- Right-click the baseline and **Deploy**
- Check the option **Remediate noncompliant rules when supported**
- Check the option **Allow remediation outside the maintenance window**
- Select the **device collection** you want to target
- Set the **schedule**
- Click **OK** to save.

This baseline should be targeted at a collection containing the devices or users that you want to use the Modern LAPS Manager app.

# Prepare the Modern LAPS Manager Custom App




The **Modern LAPS Manager** application is a custom application designed to work with this solution. The app is provided as a set of source files. I strongly recommend packaging the app before distribution to protect the code and shield the runtime environment from any tampering.

I recommend to package the application in MSIX format and deploy it to your target audience with Microsoft Endpoint Configuration Manager. Below is a guide for how to package it in MSIX format using the Microsoft MSIX Packaging tool.

*Note: remember to also deploy the Configuration baseline that contains the required parameters for the app to the same target audience.*

The source files for the application are provided so that if you customise this solution in anyway and you have the relevant skillset, you can make your own edits to the app to fit your changes before packaging.

## Package the App with the MSIX Packaging tool

To package the app in MSIX format, you will need a **code-signing certificate**. This can be from a trusted 3rd party provider, or internal PKI. In either case, the certification authority must be trusted by the client where the app is installed. You can also create self-signed certificate, in which case the certificate must be distributed to and trusted by your clients.

### Install the MSIX Packaging tool

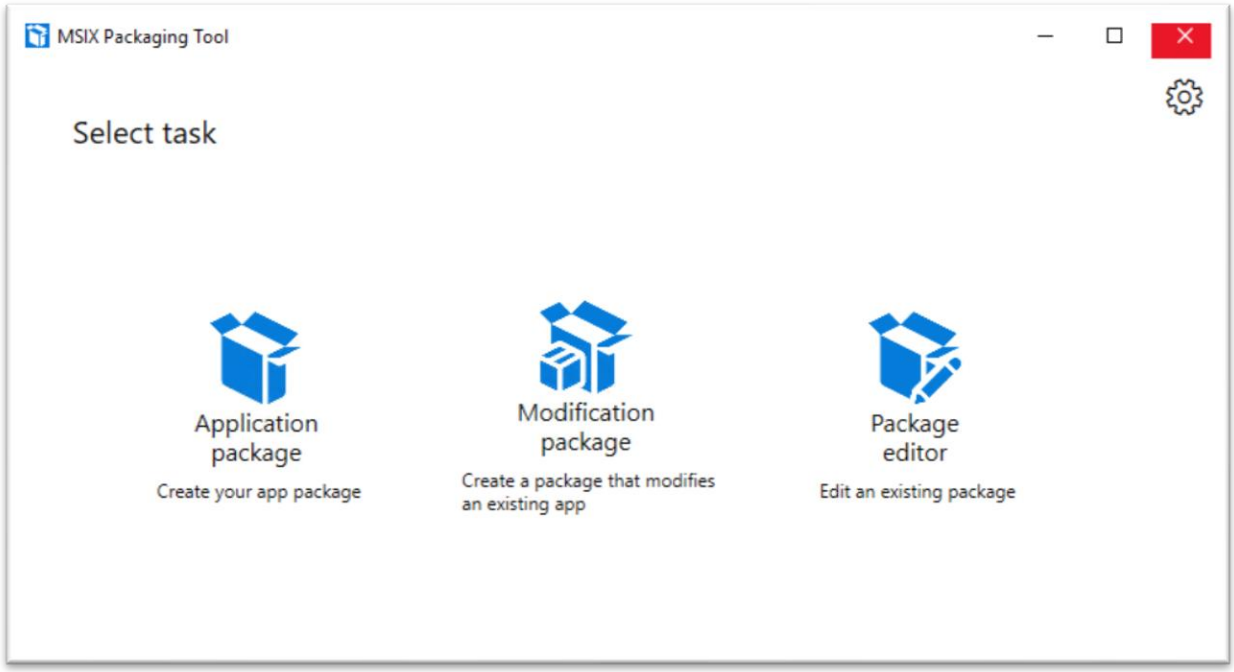
Install the MSIX Packaging tool. It is recommended to do this in a clean environment, such as a virtual machine. If using Hyper-V, there is a Quick Create VM available that contains the MSIX Packaging tool pre-installed.

### Copy the Application files

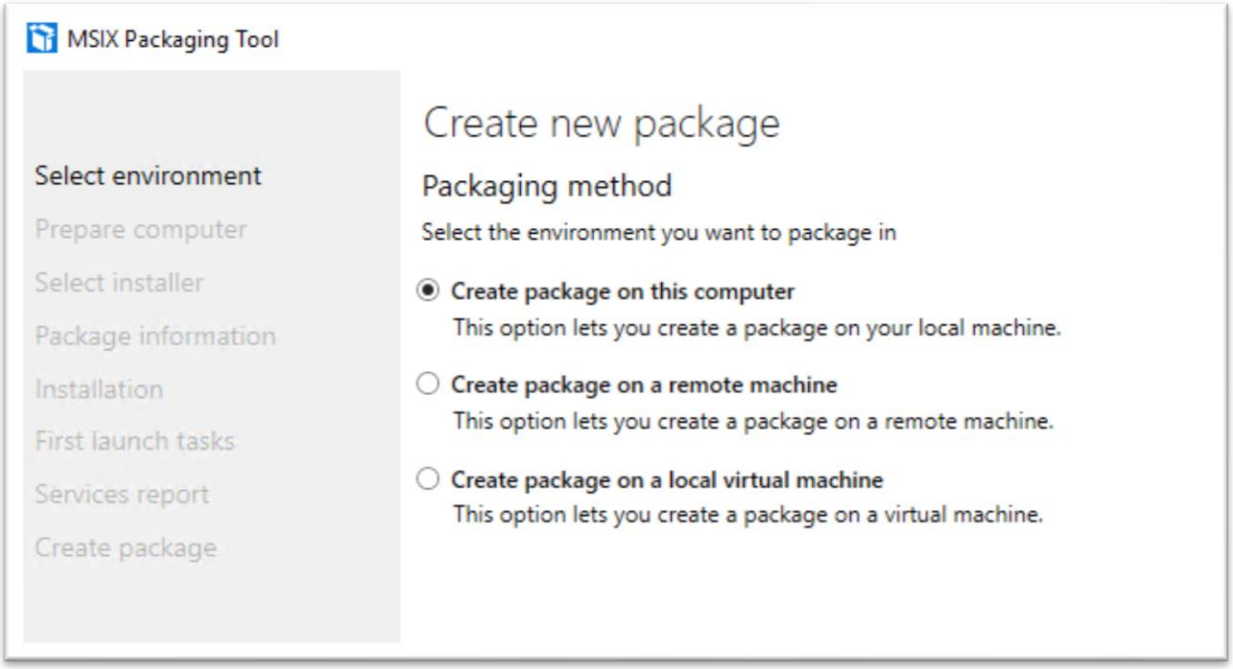
From the GitHub repo, copy all the files from the **Modern LAPS Manager app** folder to the packaging environment. The **Modern LAPS Manager.ps1** script included for reference and is not needed - it can be deleted as the executable file will be used to start the app.

### Package the Application Files

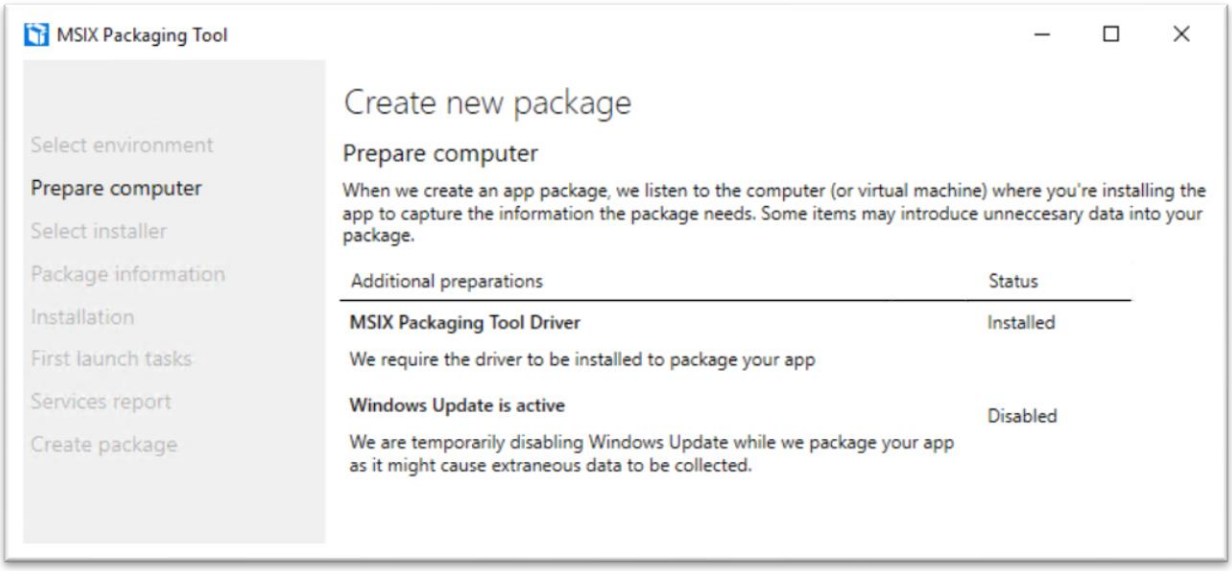
- Open the **MSIX Packaging tool**
- Click on **Application package**



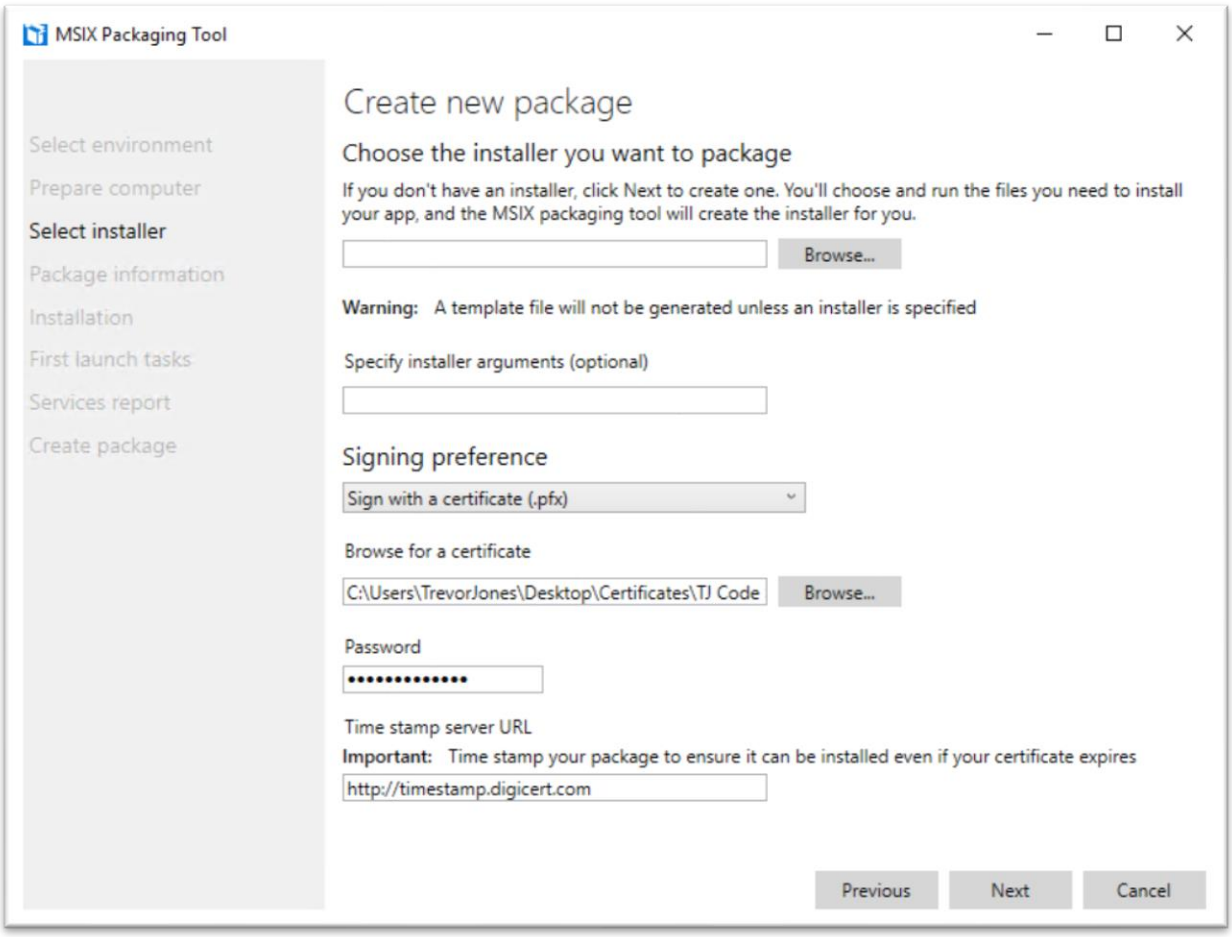

- Choose **Create package on this computer**




- Review the **preparations**

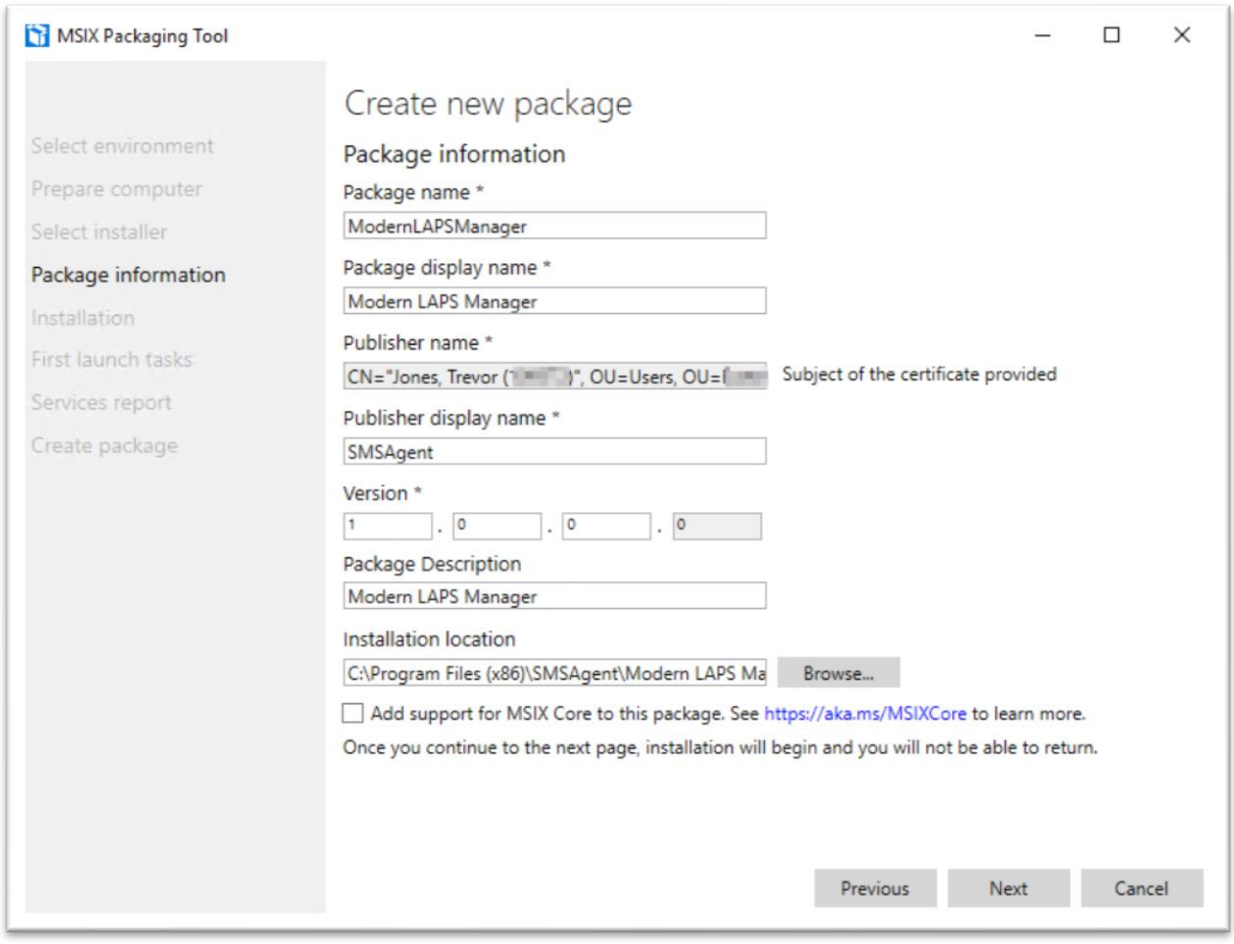



- Skip the **installer** options
- Select the **Signing preference**: **Sign with a certificate (pfx)**
- Browse to and select the **certificate pfx file**
- Enter the **password** if needed
- Enter the URL of a **time stamp authority**, such as http://timestamp.digicert.com

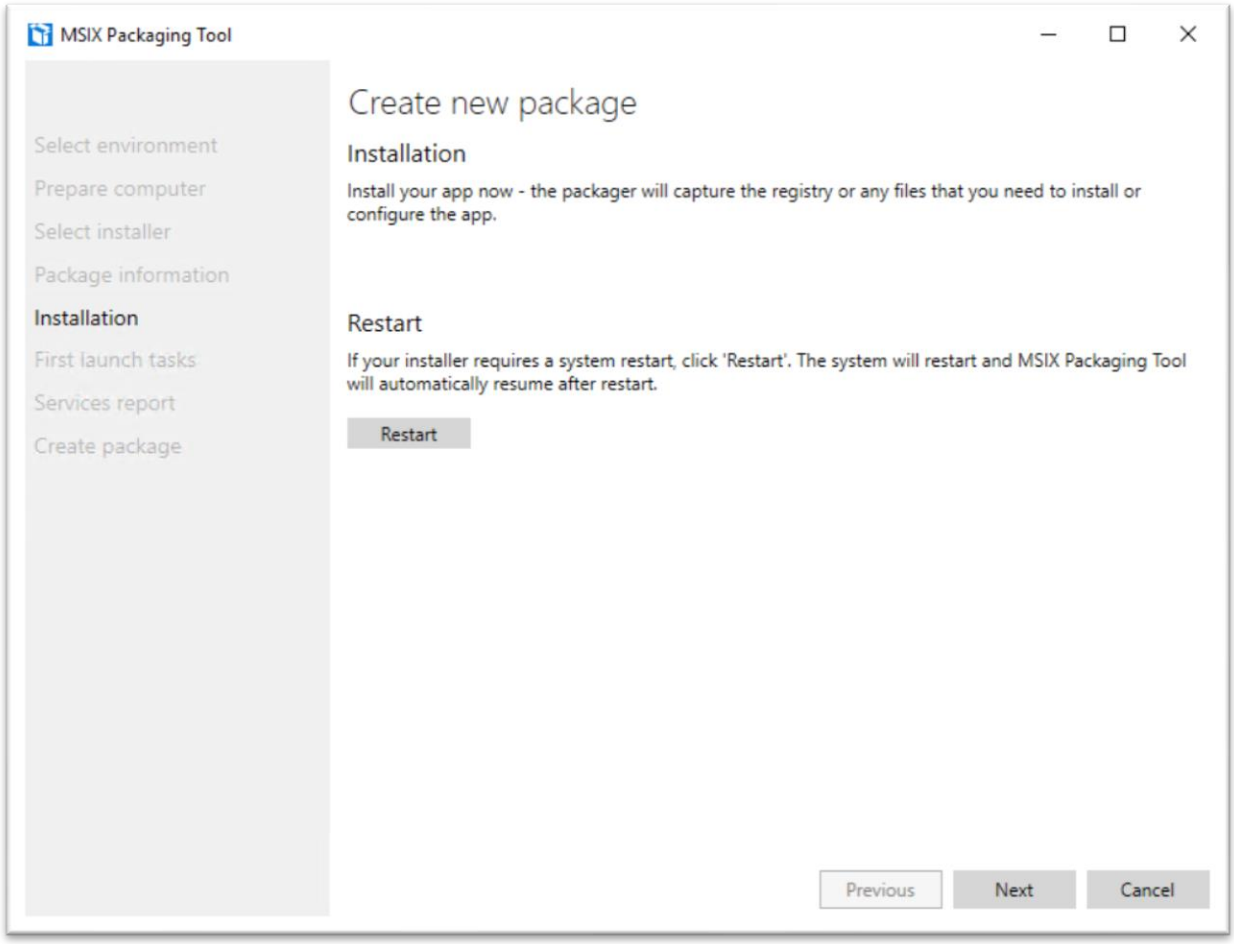



- Enter the **package name**, the **package display name**, the **publisher display name**, the **version** and the **package description**

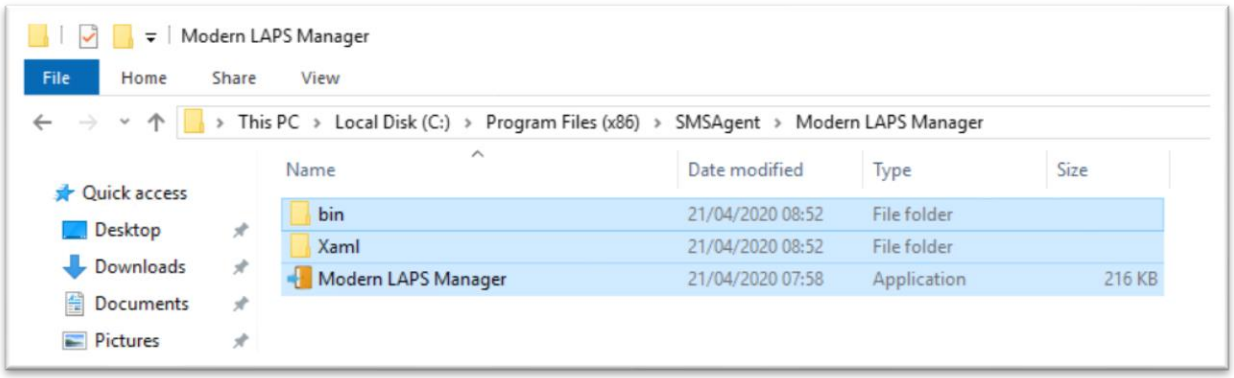
- For the **Installation location**, create a directory in the program files folder to contain the app files, for example **C:\Program Files (x86)\SMSAgent\Modern LAPS Manager**. Browse to and select this folder.



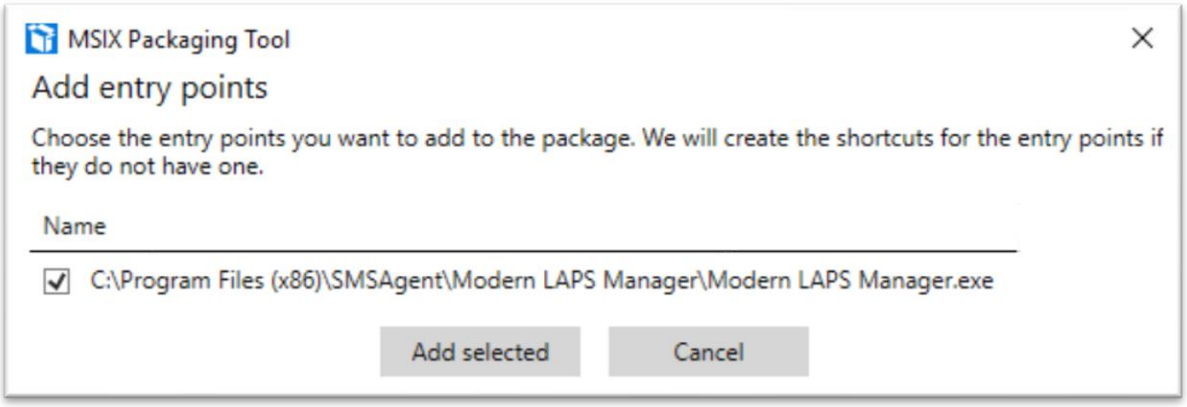

- Next you will get to the **Installation** page



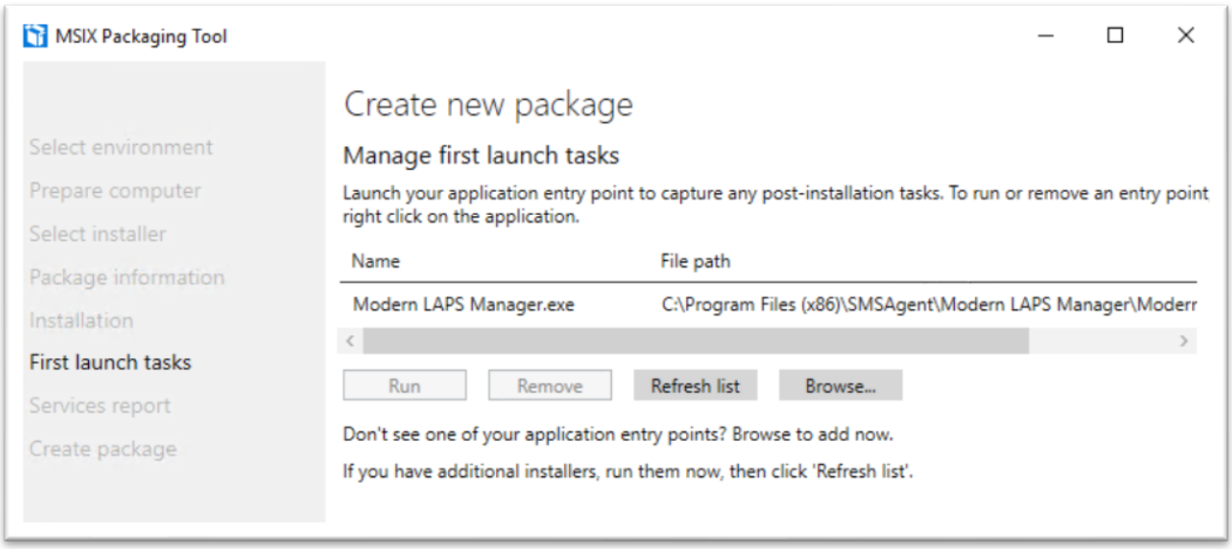

- At this point, copy all the **application files** to the **program files directory** you created:



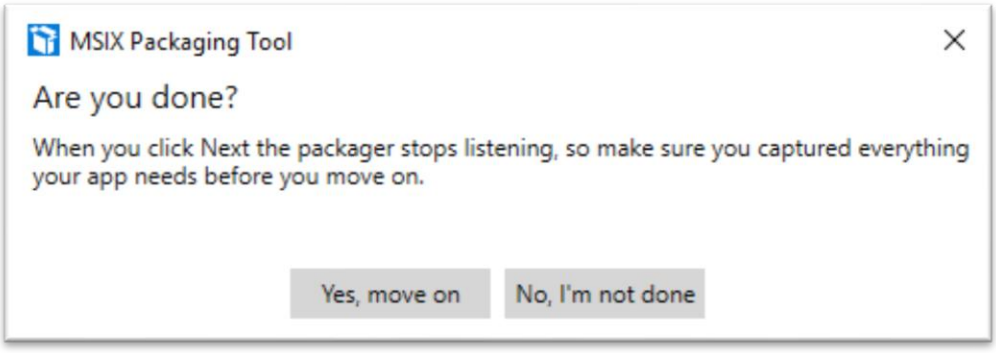

- There is no need to restart. After clicking **Next** you will be asked for an **entry point** for the application. Click **Browse** and check the **Modern LAPS Manager.exe** file



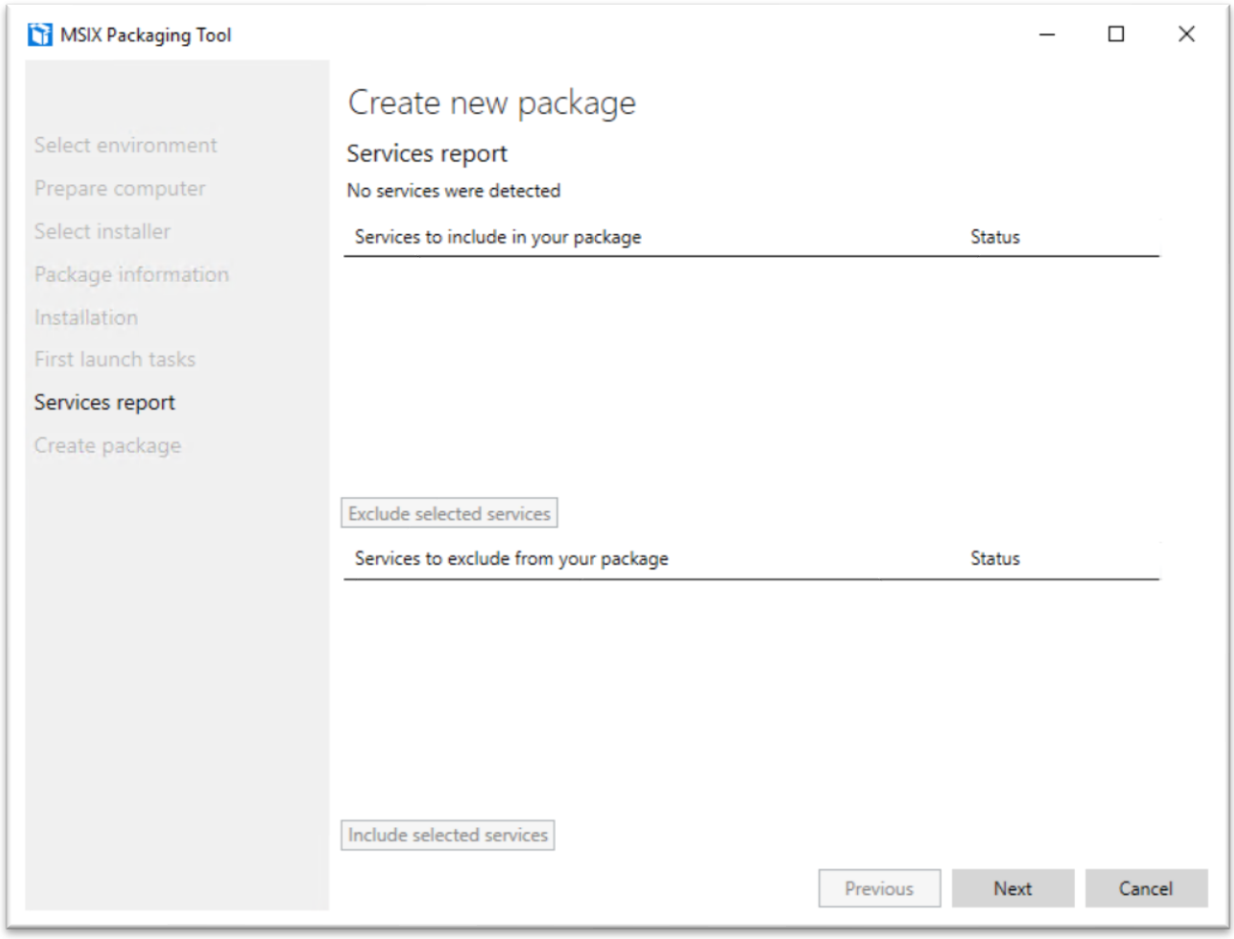

- There is no need to launch the entry point, simply click **Next**



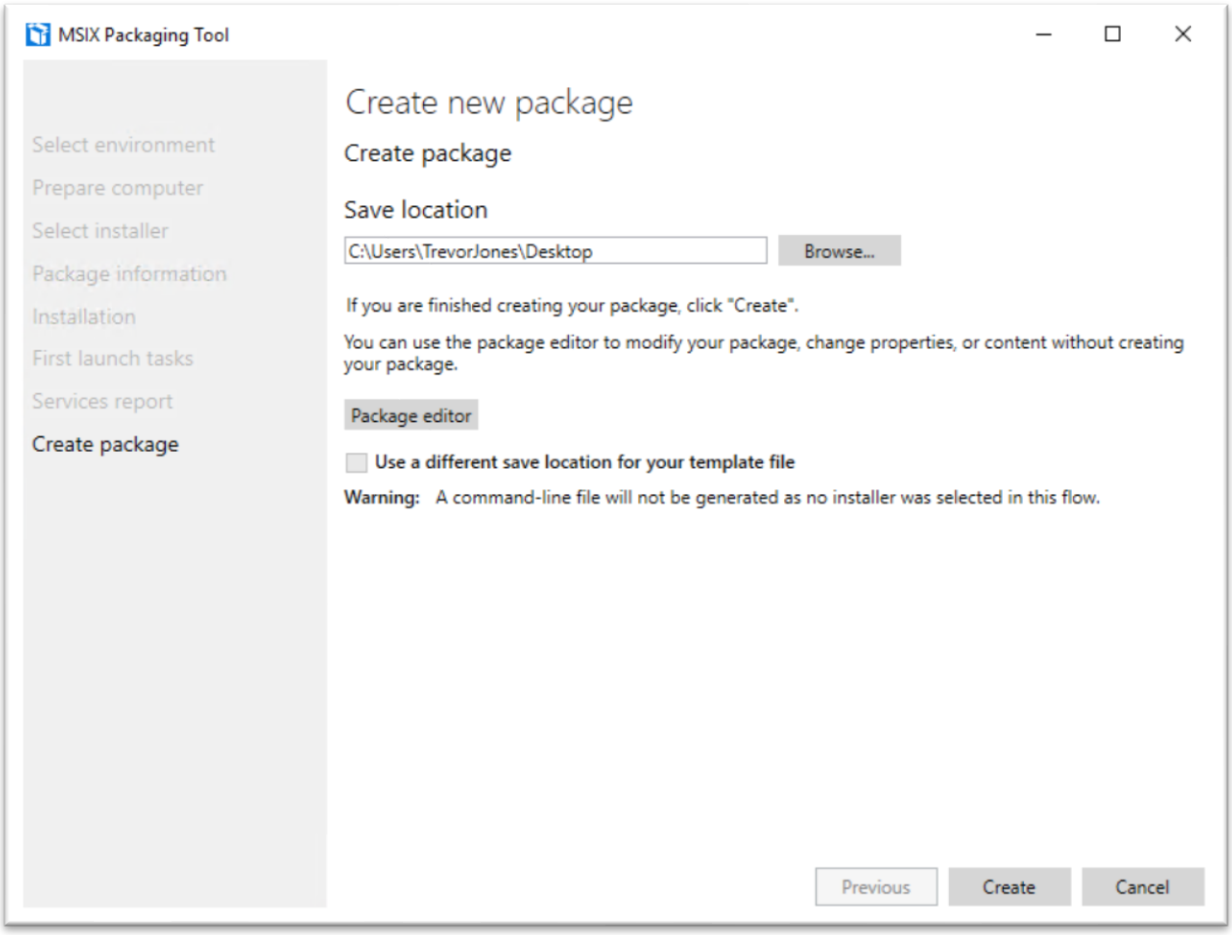

- Click **Yes, move on**



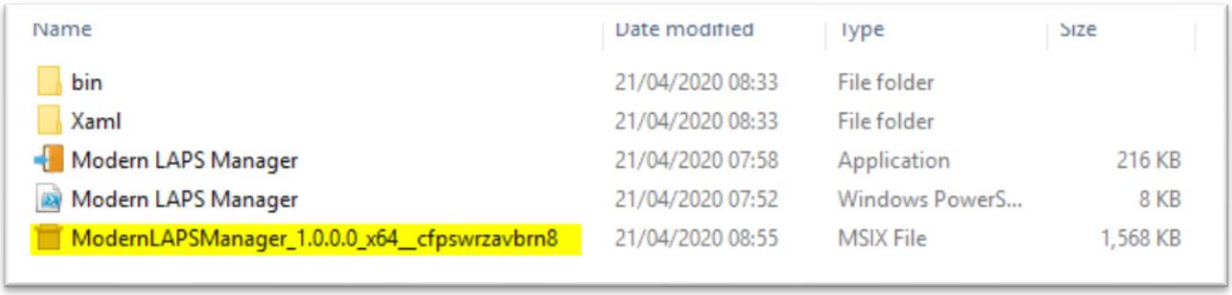

- Skip through the **Services** page




- Finally, click **Create**




You will now have an MSIX application that you can deploy to your clients.



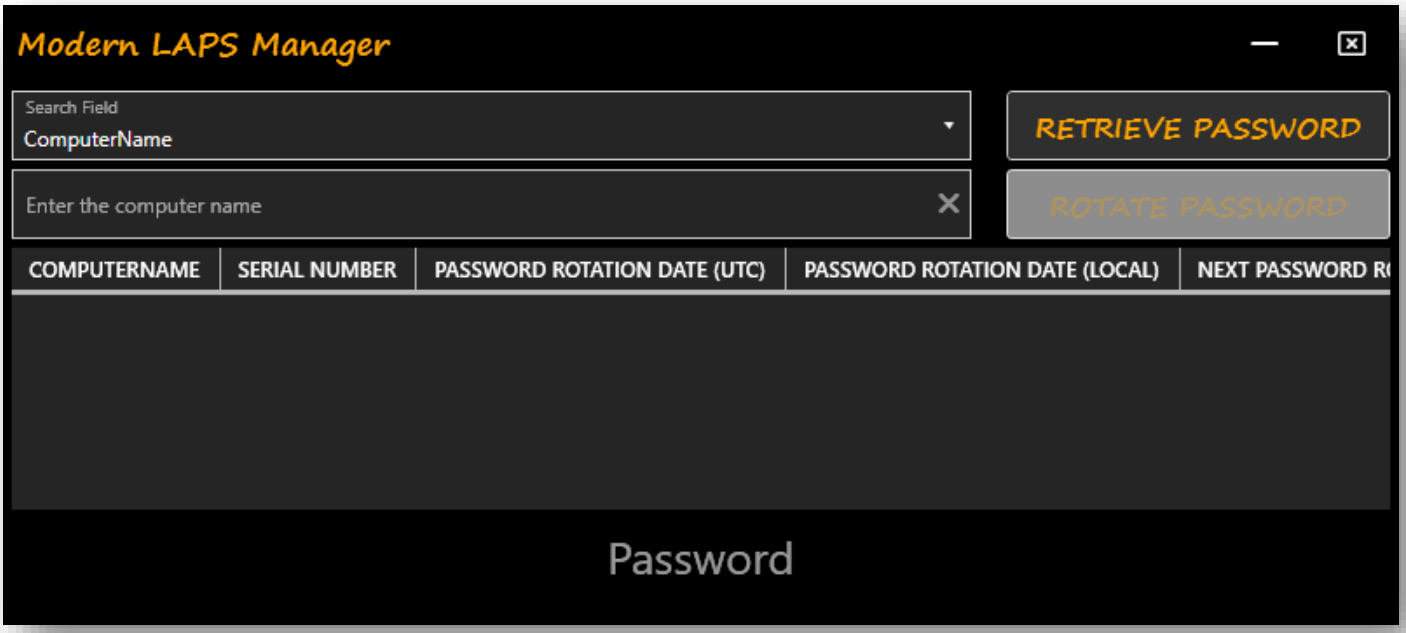
# Operations Guide

In this section, we will cover how to use the **Modern LAPS Manager** app.

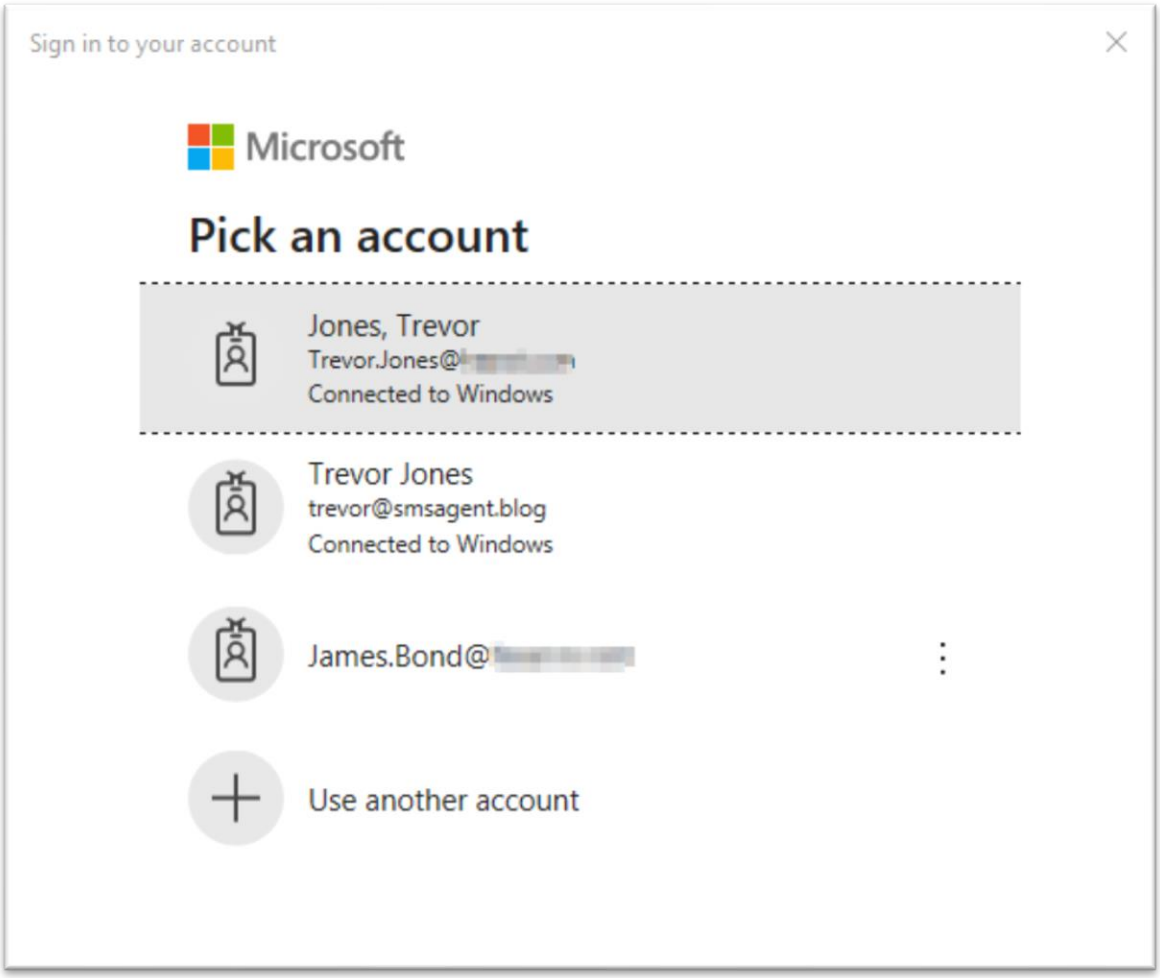
Before starting the app, make sure that the user is a member of the **Modern LAPS Users** AAD group, and that the **Configuration Baseline** containing the app parameters has been deployed and run.

## Retrieve a Password

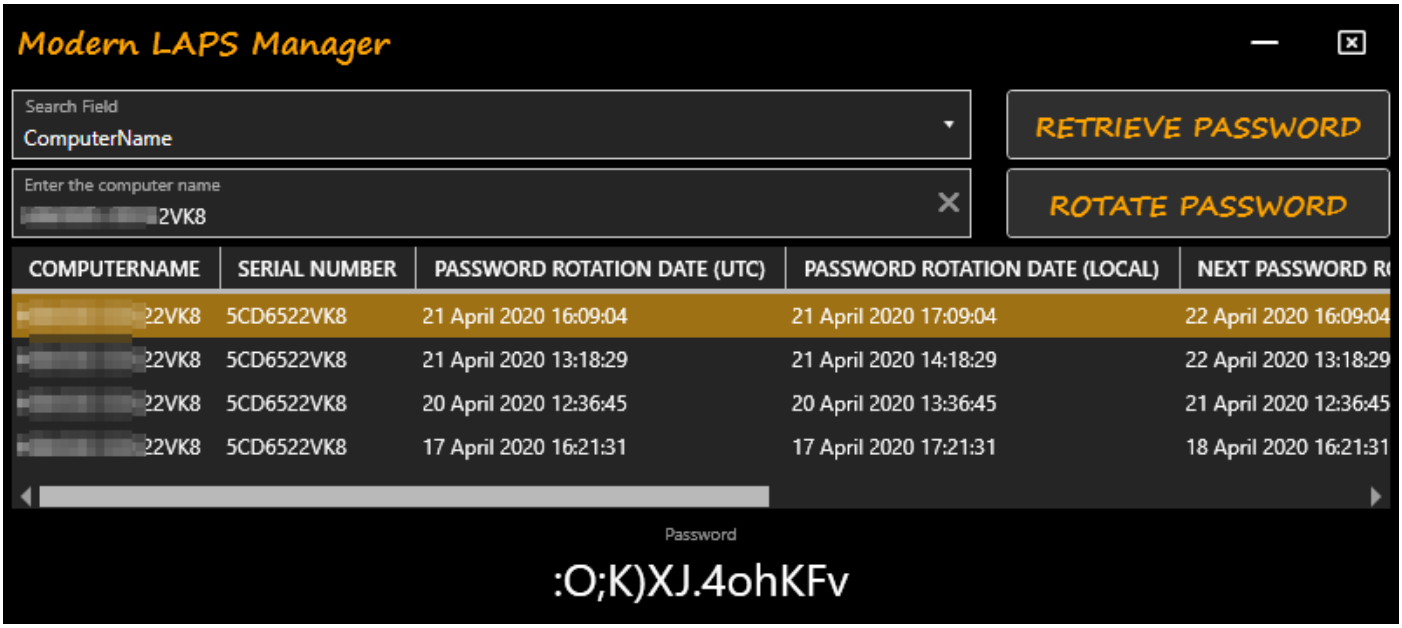
- Start the application




- In the **Search Field** box, you can select **ComputerName** or **Serial Number**. Using the ComputerName parameter should be easier, but Serial Number is included to provide a unique identifier in the case that the computer name has been changed or is unknown.
- In the box below, enter the computer name or serial number as appropriate. You can also enter just a part of the identifier to broaden a search (minimum 4 characters).
- Click **Retrieve Password**
- A **sign-in box** will appear requesting authentication with Azure AD




- After successful authentication, the app will connect to Azure Key Vault to retrieve encryption keys, then send a request to the Azure function to retrieve data from the Azure SQL server. The data is converted from JSON format into a data table and displayed in the app. The local administrator password is decrypted and displayed at the bottom.



- You'll notice that several results may be returned depending on how often the password is rotated and how much password history you are retaining in the SQL database. The app sorts the results in date descending order and selects the latest entry by default.
- There are several fields returned including the UTC date that the password was rotated, the local date, the local timezone and UTC offset of the client, when the next password rotation is scheduled etc.

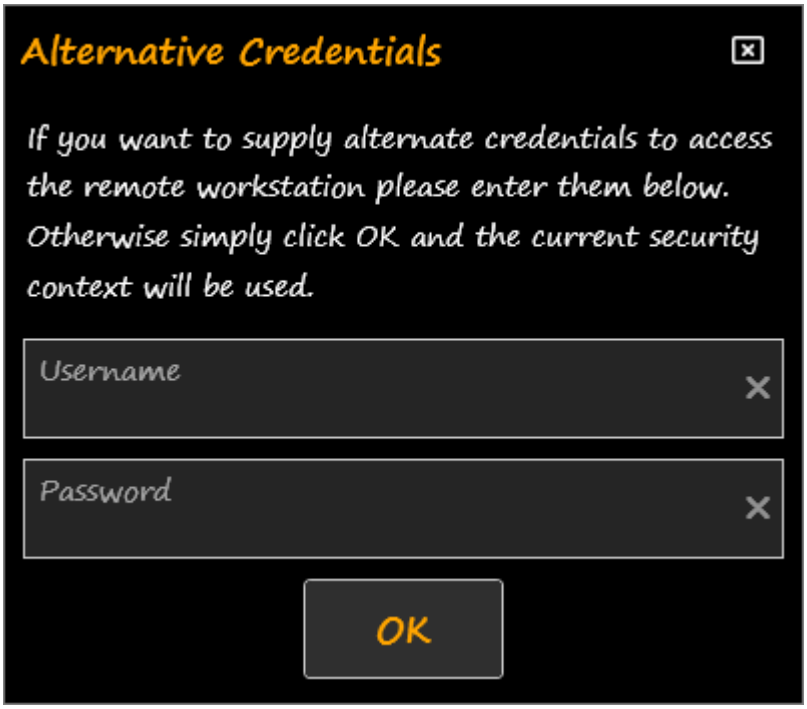
## Rotate a Password using the App

You can force an online client to rotate the local administrator password. This works by removing the current value of the **NextPasswordRotationDateUTC** from the registry and triggering an evaluation of the Configuration Baseline. The baseline PowerShell script detects that no date is set and forces a new password change.

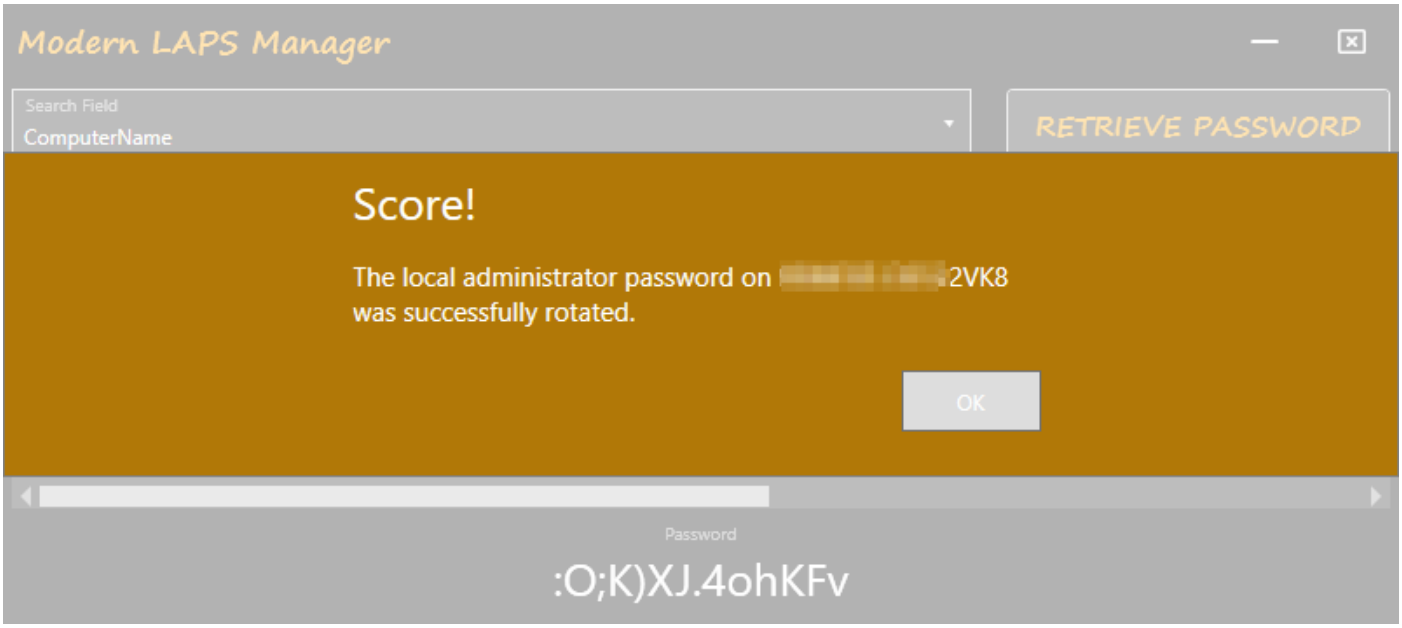
For this to work, you must have credentials with administrative privileges on the target device, as well as be able to establish a CIM session and a PowerShell session on the default ports.

Note: to change the local administrator password for the local workstation, you must run the app as administrator.

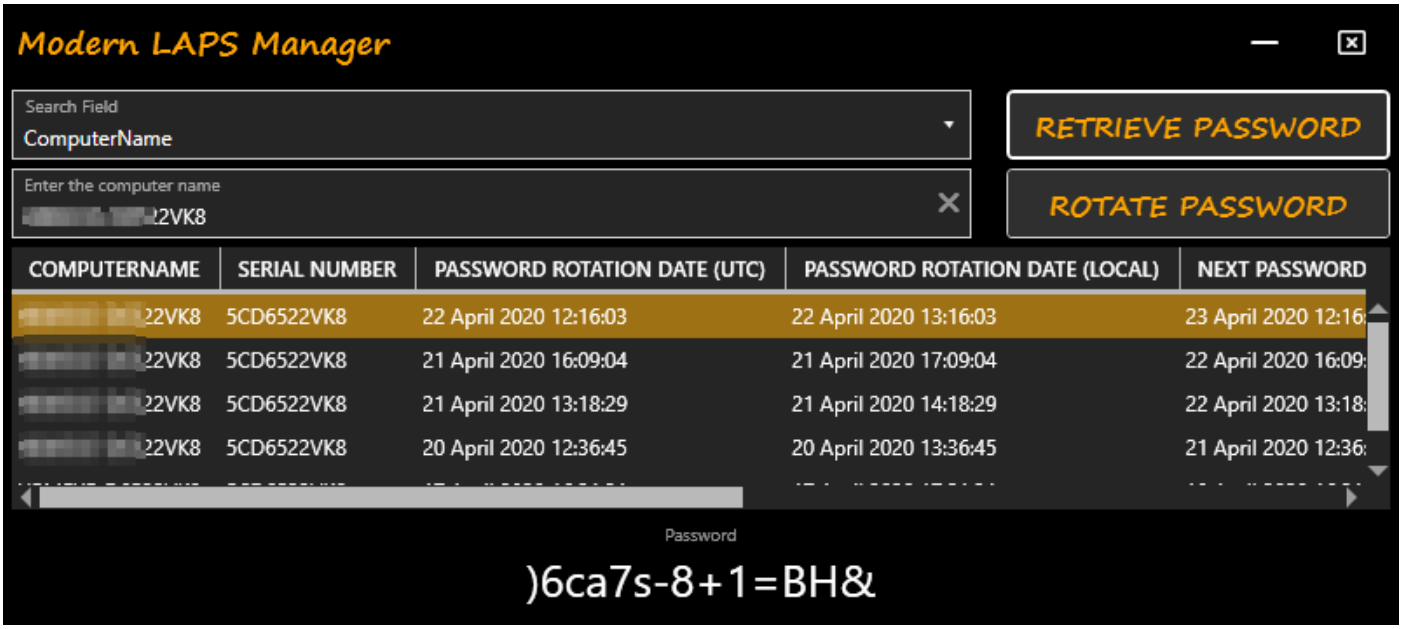
- First retrieve the password for a device using the app
- Select an entry in the table for that device
- Click **Rotate Password**
- A window will appear asking if you want to supply alternate credentials. Either enter credentials or just click **OK** to continue in the current user context




- You will be notified that the password change was successful




- To verify the change, wait a couple of seconds for the password change to route through the Azure services, then run **Retrieve Password** again. You'll see the new password that was generated




## Rotate a Password with Configuration Manager

You can also rotate a password using the **Run Script** feature of **Configuration Manager**. This method uses the client fast channel and doesn't require user-level local administrator privilege or WinRM access, and it will also work on an internet-only connected device via a Cloud Management Gateway.

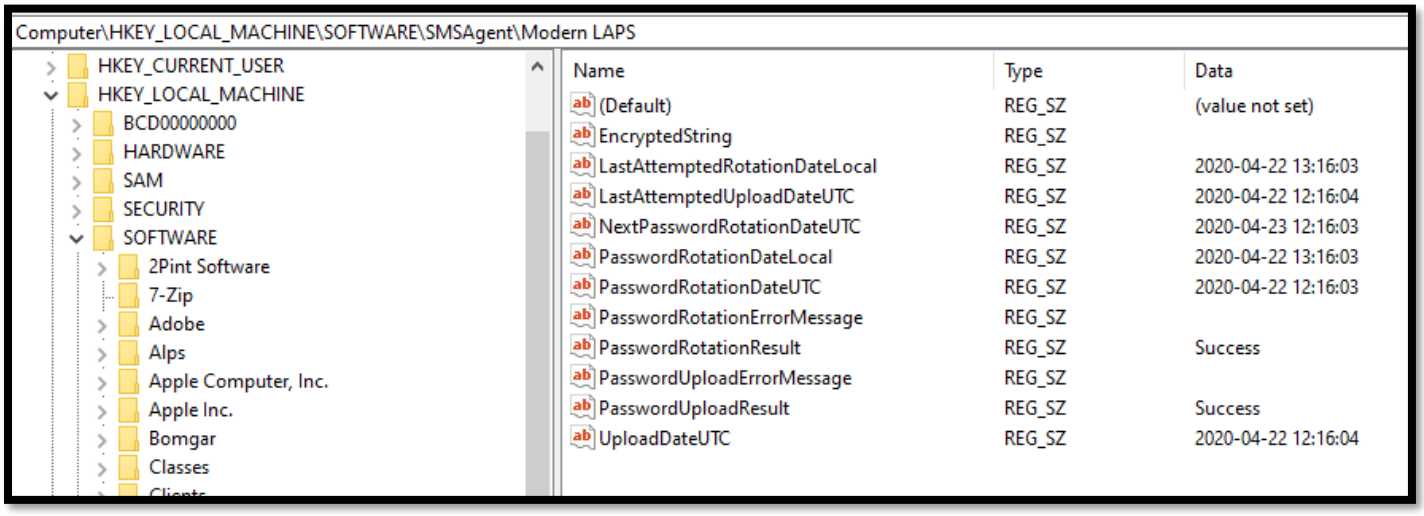
- Create and approve a new **Script** in **Configuration Manager** using the code from the **Modern LAPS Configuration Manager Manual Password Rotation Run Script** in the GitHub repo.
- **Update** the value of the parameter **$CI_UniqueID** with the unique ID of the configuration baseline that rotates the local administrator password. If you followed the instructions in this document, this will be in your Azure Key Vault.
- Locate a device (or use a collection) in the **Device** node in the console, right-click and **Run Script**.
- Select and run the script you created.
- After successful execution, you can verify the password change using the app.

# Client-Side Troubleshooting

The app logs activities both to the application event log and to the registry. If there was an error in either rotating the password or uploading the password, you'll find details in both places.

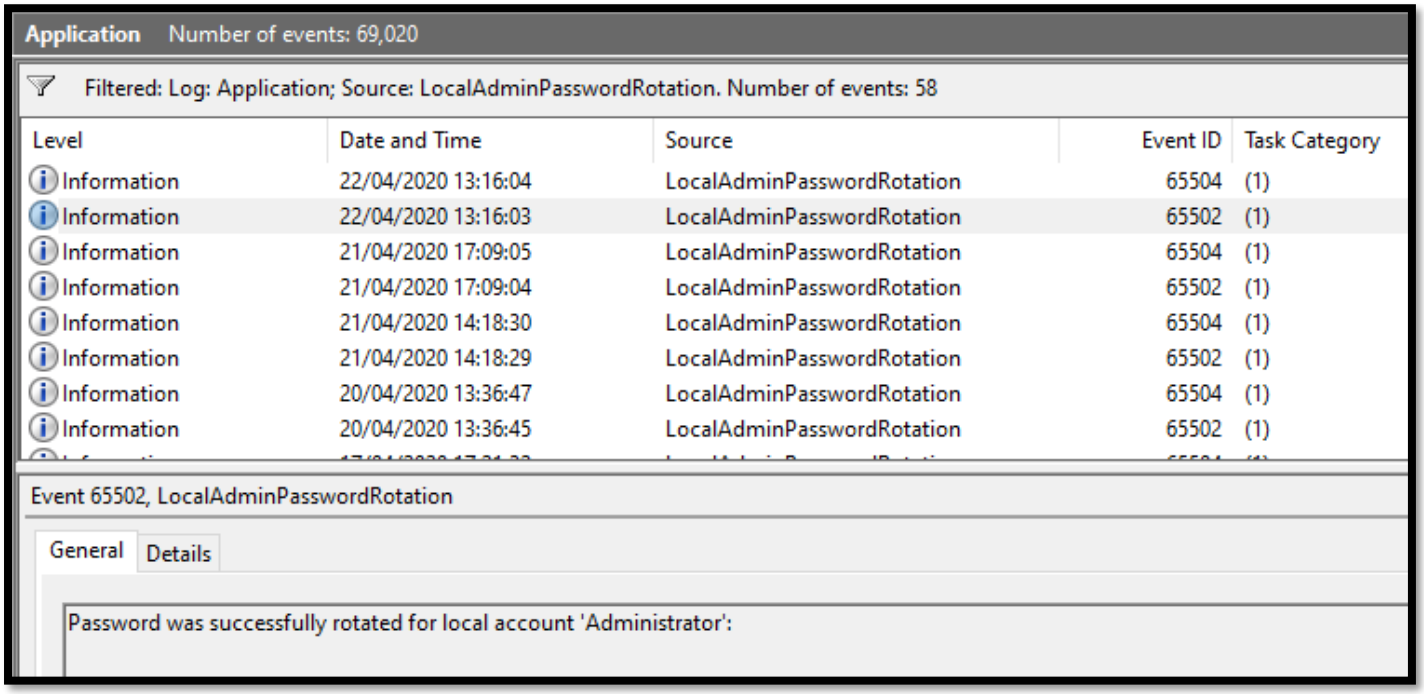
## Registry

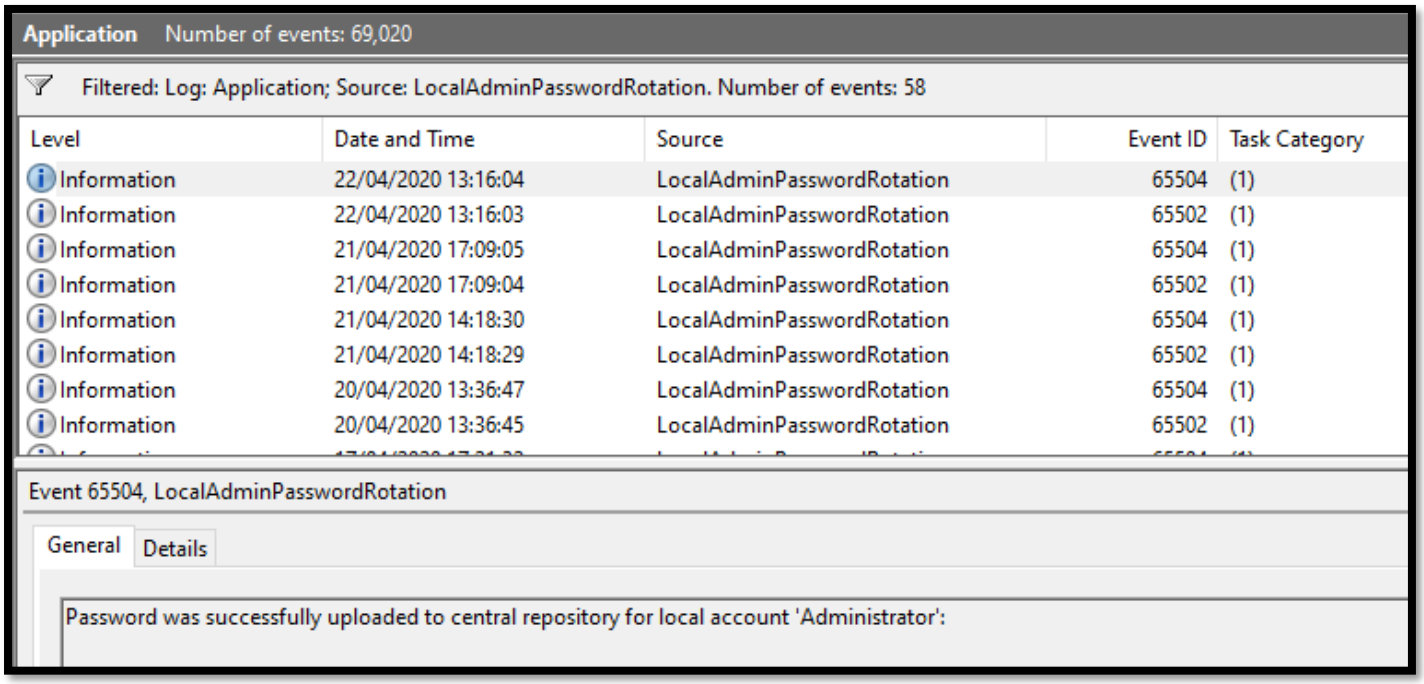
The registry keys are located at **HKLM:\SOFTWARE\SMSAgent\Modern LAPS**. Note that you could inventory this registry location with Configuration Manager to provide some insight on password changes across the organisation, as described in the Monitoring guide.

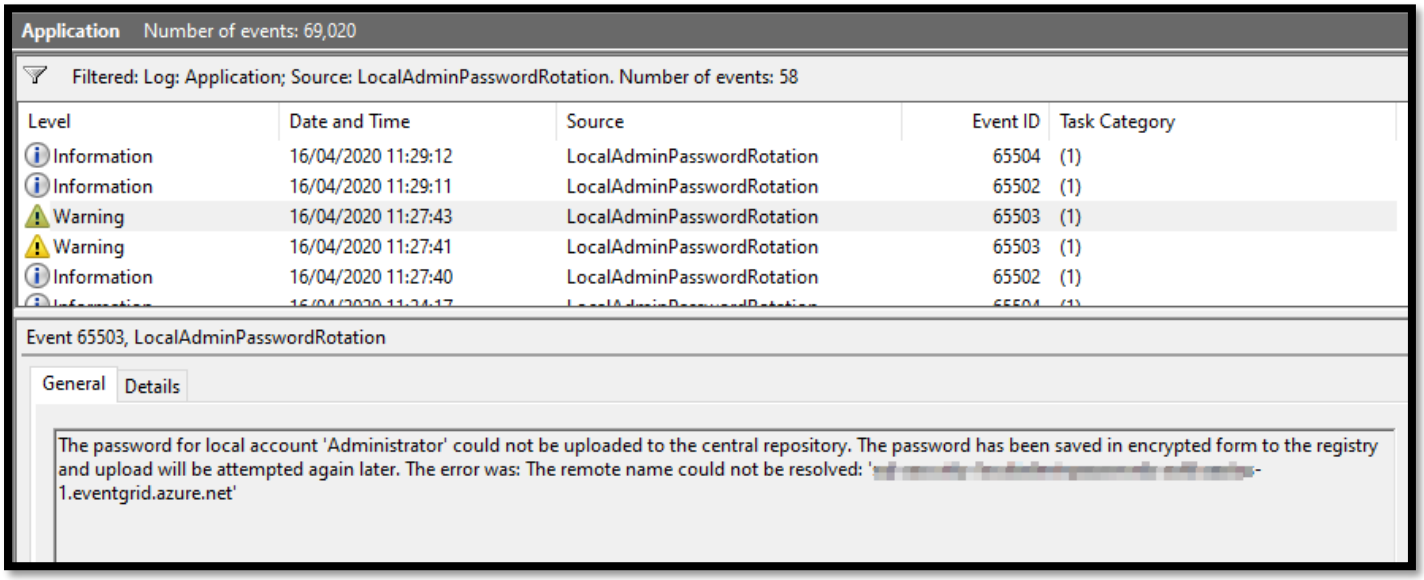



## Event Log

The app logs to the Application event log using the Source **LocalAdminPasswordRotation**. You can filter the event log by that source to find all password change events.




Application Number of events: 69,020
Filtered: Log: Application; Source: LocalAdminPasswordRotation. Number of events: 58
Level Date and Time Source Event ID Task Category
Information 22/04/2020 13:16:04 LocalAdminPasswordRotation 65504 (1)
Information 22/04/2020 13:16:03 LocalAdminPasswordRotation 65502 (1)
Information 21/04/2020 17:09:05 LocalAdminPasswordRotation 65504 (1)
Information 21/04/2020 17:09:04 LocalAdminPasswordRotation 65502 (1)
Information 21/04/2020 14:18:30 LocalAdminPasswordRotation 65504 (1)
Information 21/04/2020 14:18:29 LocalAdminPasswordRotation 65502 (1)
Information 20/04/2020 13:36:47 LocalAdminPasswordRotation 65504 (1)
Information 20/04/2020 13:36:45 LocalAdminPasswordRotation 65502 (1)
Event 65504, LocalAdminPasswordRotation
General Details
Password was successfully uploaded to central repository for local account 'Administrator':


Application Number of events: 69,020
Filtered: Log: Application; Source: LocalAdminPasswordRotation. Number of events: 58
Level Date and Time Source Event ID Task Category
Information 16/04/2020 11:29:12 LocalAdminPasswordRotation 65504 (1)
Information 16/04/2020 11:29:11 LocalAdminPasswordRotation 65502 (1)
Warning 16/04/2020 11:27:43 LocalAdminPasswordRotation 65503 (1)
Warning 16/04/2020 11:27:41 LocalAdminPasswordRotation 65503 (1)
Information 16/04/2020 11:27:40 LocalAdminPasswordRotation 65502 (1)
Event 65503, LocalAdminPasswordRotation
General Details
The password for local account 'Administrator' could not be uploaded to the central repository. The password has been saved in encrypted form to the registry and upload will be attempted again later. The error was: The remote name could not be resolved:
1.eventgrid.azure.net'

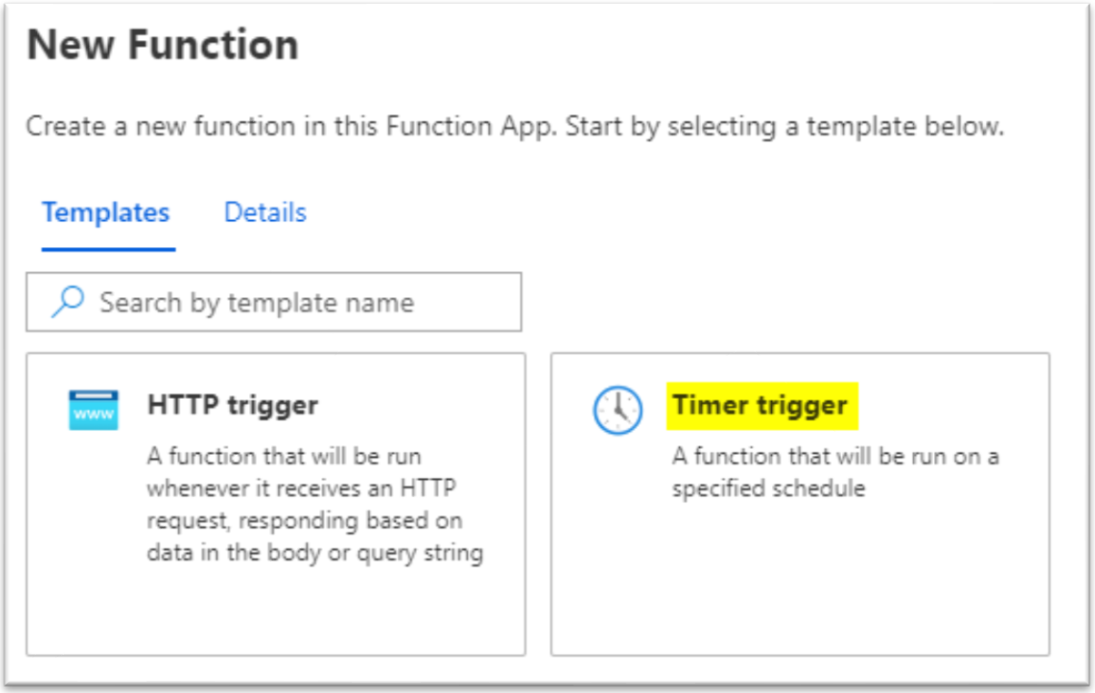
# Maintenance Guide

In this section, we will cover activities for maintaining the solution, such as performing encryption key rotation.

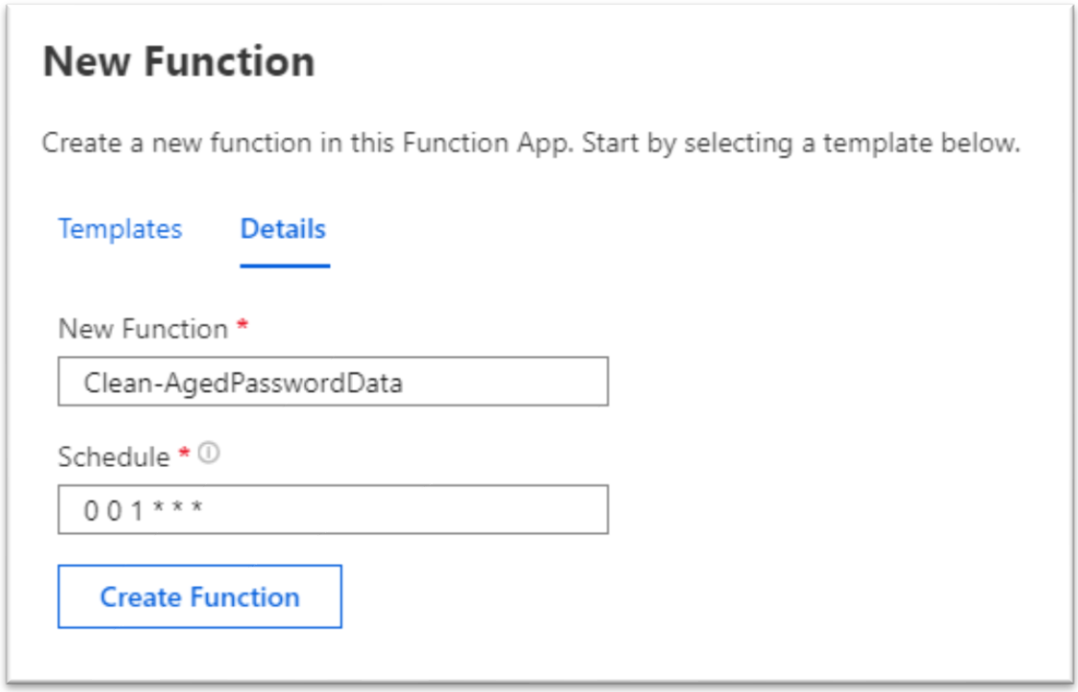
## Cleaning Out Aged Password Data from the SQL Database

Over time, the SQL database will fill up with historic password data which is undesirable, will increase storage requirement and could lengthen the time taken to run queries against the database. To clear out aged password data, you can create a function in your Function app with a Timer trigger. This could, for example, run every day and clean out historic password data older than x days. A PowerShell script is included in the GitHub repo to use for this purpose.

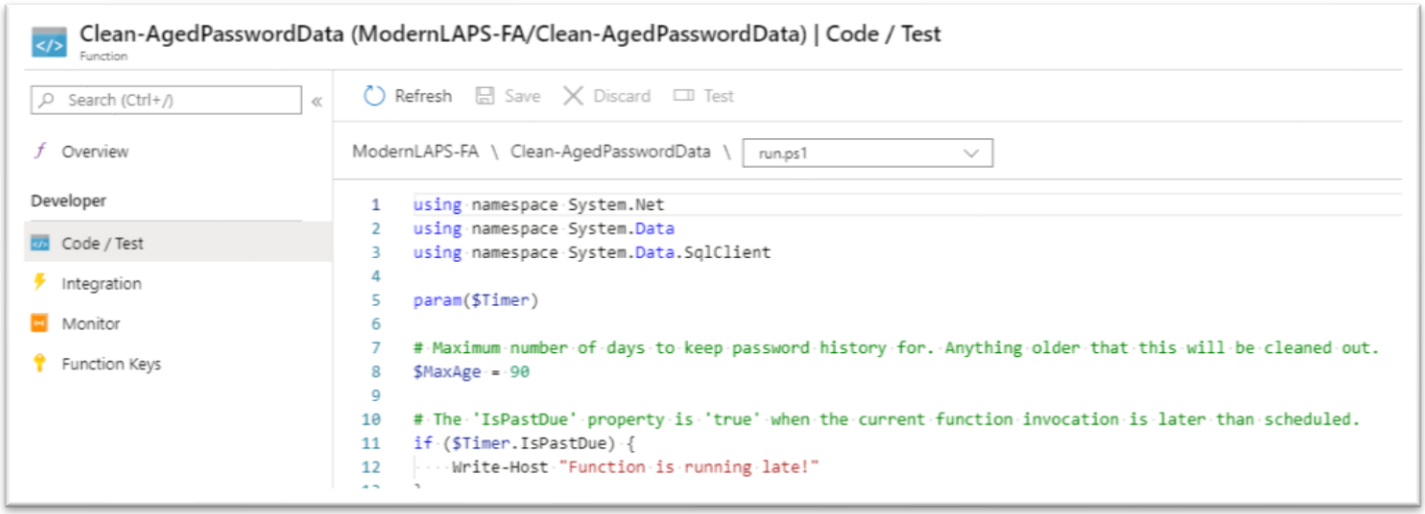
- In Azure portal, search for **Function app**
- Click on the function app you created in this guide
- Click on **Functions**
- Click **+ Add** to create a new one




- Choose the **Timer trigger** template
- Give the function a **name** and add a **Schedule** in cron format. The example schedule is every day at 01:00 AM. Click **Create Function**




- Once created, in the function click on **Code / Test**. Paste in the content of the **Modern LAPS Function Clean Aged Password Data** script, overwriting the existing content.
- Change the value of the **$MaxAge** parameter to the number of days you want to keep password history for
- **Save** the function.



# Monitoring Guide

In this section, I will provide some tips to monitor the solution, including monitoring the Azure resources as well as monitoring password rotation on the clients with ConfigMgr and PowerBI.

## Azure Monitoring

The various services and resources in Azure used by this solution should be monitored. For example, if events from the event grid are getting dropped, or the function app fails to update the database etc you need to know about it. Azure Monitor provides some great tools to help with this. I won't dive into the details of configuring Azure Monitor for your environment in this guide, but I'll give some suggestions of what you can do.

### Configure Diagnostic Settings

Using Azure Monitor you can enable **diagnostic settings** for the various resources and send log data to a destination such as a storage account for archiving and manual review, or to a Log Analytics workspace.

### Log Analytics

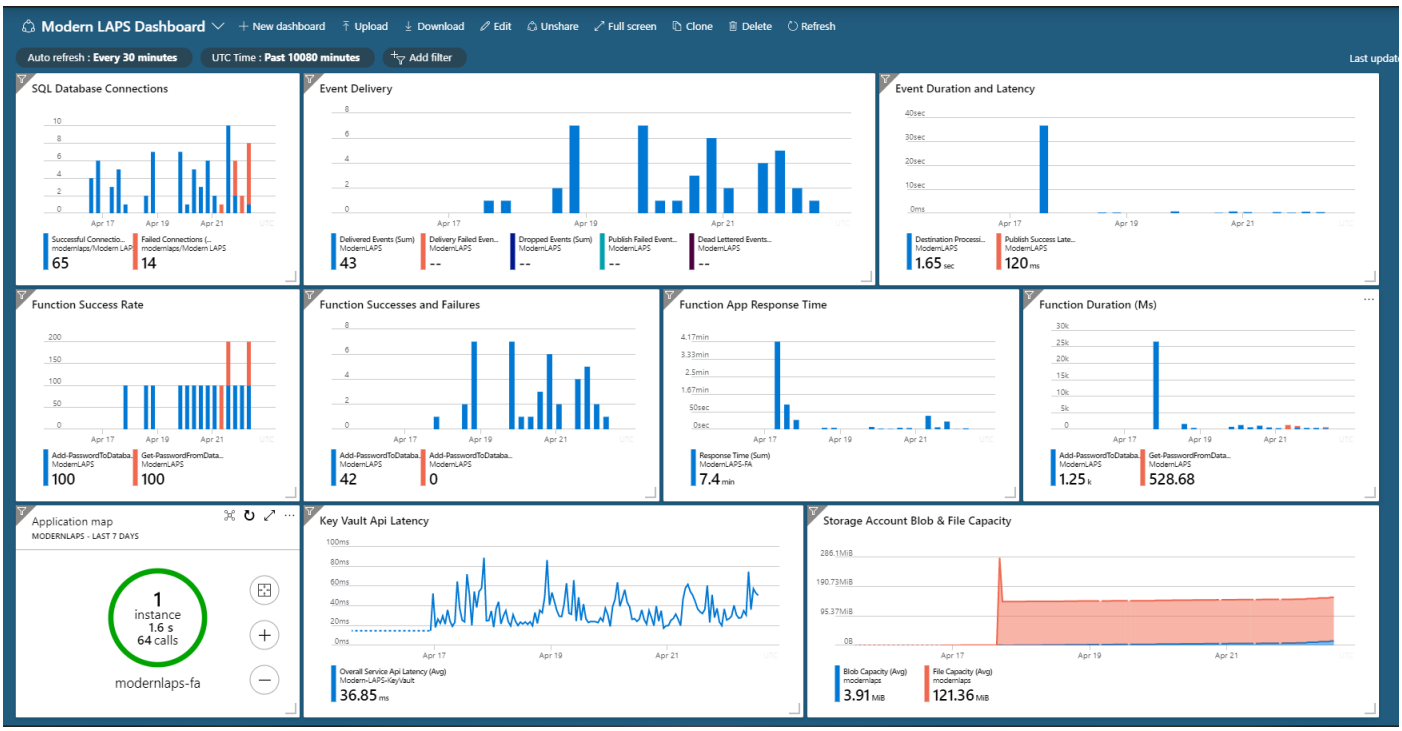
Consider creating a **Log Analytics** workspace in your resource group and sending logs there. This provides a rich analytics environment within Azure to analyse resource data.

### Alerts

Consider creating **Alert rules** to notify you of events such as failed connections to the SQL Server, or dropped events in the Event Grid topic.

### Create a Dashboard

A **dashboard** is very handy for a quick overview of key metrics in the solution. In the screenshot below, I created a dashboard to monitor key metrics from the SQL database, the Event Grid, the Function app, the Key Vault and the Storage account.



# Configuration Manager Reporting

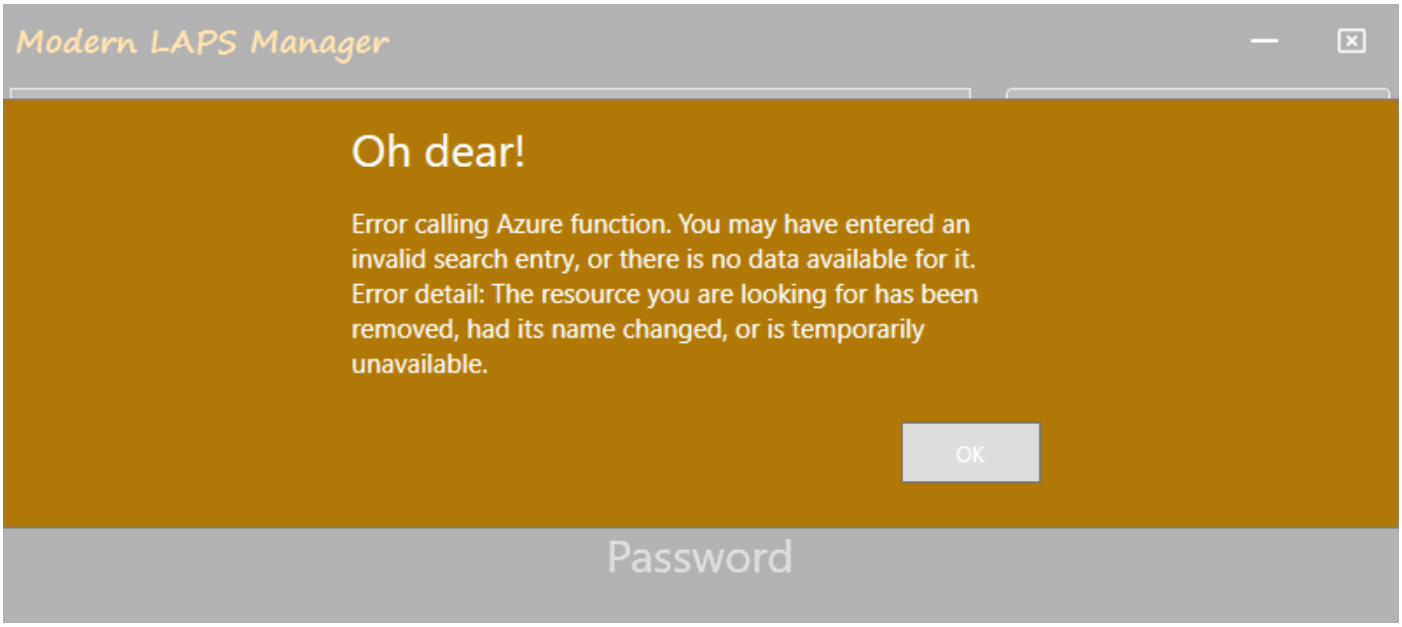
## Inventory the Registry Keys

Consider inventorying the registry keys used on the client under HKLM:\SOFTWARE\SMSAgent\Modern LAPS with the ConfigMgr client. This will give you a view in the SQL database that you can query for password change data such as the last time a password was changed for a device, the number of devices whose password change is older than the defined rotation schedule, any devices that are erroring in changing the password or uploading it etc. You can pull this data into a PowerBI report for a quick overview of the health of the solution on the client side.

# Troubleshooting

## The resource you are looking for has been removed..

When attempting to retrieve a password, the following error is returned in the app.




This is essentially an HTTP 400 error and can occur if the URL for the function app is incorrect, or if the authentication key in the URL is incorrect or expired. Check that the URL for the function app is correct in the Configuration Item that sets the registry keys required for the app, and / or update the authentication key in the URL as required.