**DBLoader Utility**

The DBLoader can be used to load data from delimited text files into SQL Server. This Windows console utility uses the SQL Server native client bulk load interface, which works on all versions of SQL Server, including Azure SQL DB, Azure SQL MI and Azure SQL DW.

The files that DBLoader imports can be located on a windows file system, Azure Blob Storage or Azure Data Lake Storage. The files can be uncompressed or gzip compressed. The utility only supports the specification of a single destination table, so loading multiple tables will require multiple executions. Wildcards are supported, to enable the utility to load all the files in a folder to the same table. The schema of the file is inferred from the schema of the SQL Server table.

The DBLoader utility is not meant as a replacement for Polybase for loading data into Azure SQL DW. If optimized properly, Polybase will load files on all of the compute nodes in the DW cluster, while DBLoader uses the bulk load interface, which must first traverse the single control node.

DBLoader uses a combination of command line parameters and a configuration file to allow the user to control its execution.

**Command Line**

DBLoader <path><file name> -S<schema> -T<tablename>

**Parameters**

Note, there are no spaces between the option and the value

|  |  |
| --- | --- |
| **<path>** | optional path specification – otherwise the current directory is used  Prefixes;   * drive letter (local storage) * asb:/ (Azure Blob Storage) * adl:/ (Azure Data Lake) |
| **<file>** | mandatory file specification or wildcard specification |
| **<schema>** | optional schema specification – default is DBO |
| **<tablename>** | SQL Server table name to write the data into |

**Config File Settings**

|  |  |  |
| --- | --- | --- |
| **Key** | **Optional** | **Description** |
| ConnectionString | N | SQL Server connection string |
| StorageConnectionString | Y | Azure Blob Storage connection string with key from Azure portal |
| TenantID | Y | Azure Data Lake Store Credential – tenant (AAD) ID |
| ApplicationID | Y | Azure Data Lake Store Credential – application ID configured in AAD (web application) |
| SecretKey | Y | Azure Data Lake Store Credential – Key for APP in AAD, configure a key, save and copy. |
| ADLSAccount | Y | Azure Data Lake Store Credential – Azure Data Lake Storage Account (xxxx. azuredatalakestore.net) |
| UseStringDelimiter | N | Use string delimiter for column values in the file |
| StringDelimiterDecimal | Y | Decimal value of string delimiter character used |
| ColumnDelimiterDecimal | N | Decimal value of column delimiter character used |
| CRSubstitutionDecimal | Y | Decimal value of character used in column values that should be substituted with a carriage return. |
| LFSubstitutionDecimal | Y | Decimal value of character used in column values that should be substituted with a line feed. |
| NullColumnValue | Y | The string that is used for null columns, also note that zero length strings are interpreted as null. |
| MaxBadColumnValues | N | Maximum number of error messages per file |
| DiscardRowOnError | N | Try to keep rows that have errors in them – warning; values with errors are replaced with substitutes |
| SilentlyReplaceInvalidDates | N | Other platforms have dates like 0000/00/00, which are out of range for SQL Server – they automatically will be replace with the minimum value for the date data type selected either with a message or not. |
| RowsPerTableWrite | N | Number of rows to accumulate before writing to SQL Server. Note that if there aren’t this number of rows in a file, it writes however many rows it has at the end of every files processing. Note that increasing this will affect memory usage. |
| BufferSize | N | The size of the file read buffer in bytes. Note that the program allocates two of this size buffers for every file being processed, so be aware of potential memory consumption issues. Also note that larger buffers do not necessarily make the load go faster. The reason for two buffers is that the application processes one buffer while asynchronously reading the next buffer. |
| Threads | N | The maximum number of parallel tasks the application will use to load multiple files. Note that each file is processes by a single thread, so this setting has no effect on single file loads. Also note that each file processing activity uses asynchronous activities for reading the next buffer from the file and bulk copying the data to SQL Server, so each task will potentially use 3 threads concurrently. The default threadpool is 32K threads, but unless you have a lot of processors and memory, you will thrash your machine using more than 32 threads. |

**Sample Command Lines**

**DBLoader c:\temp\myfile.txt -Tmytable**

Load myfile.txt located in the local temp directory into mytable on the SQL Server configured in the app.config file.

**DBLoader asb:/<container>/\*.gz -Smyschema -Tmytable**

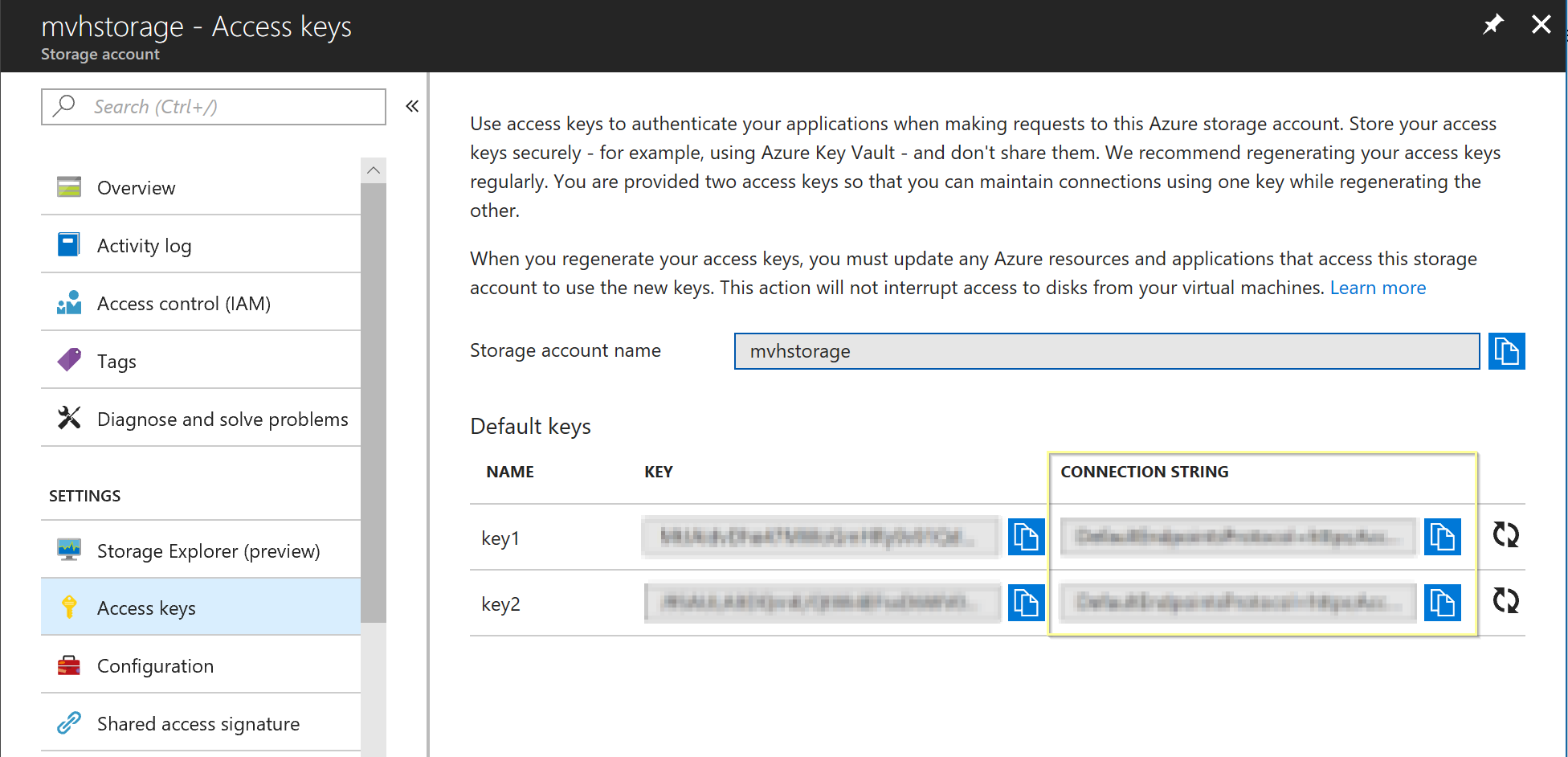
Load all of the gzip files in the blob storage container into mytable on the configured SQL Server.

**DBLoader adl:/<folder path>/\*.gz -Smyschema -Tmytable**

Load all of the gzip files in the Azure Data Lake folder given into mytable on the configured SQL Server.

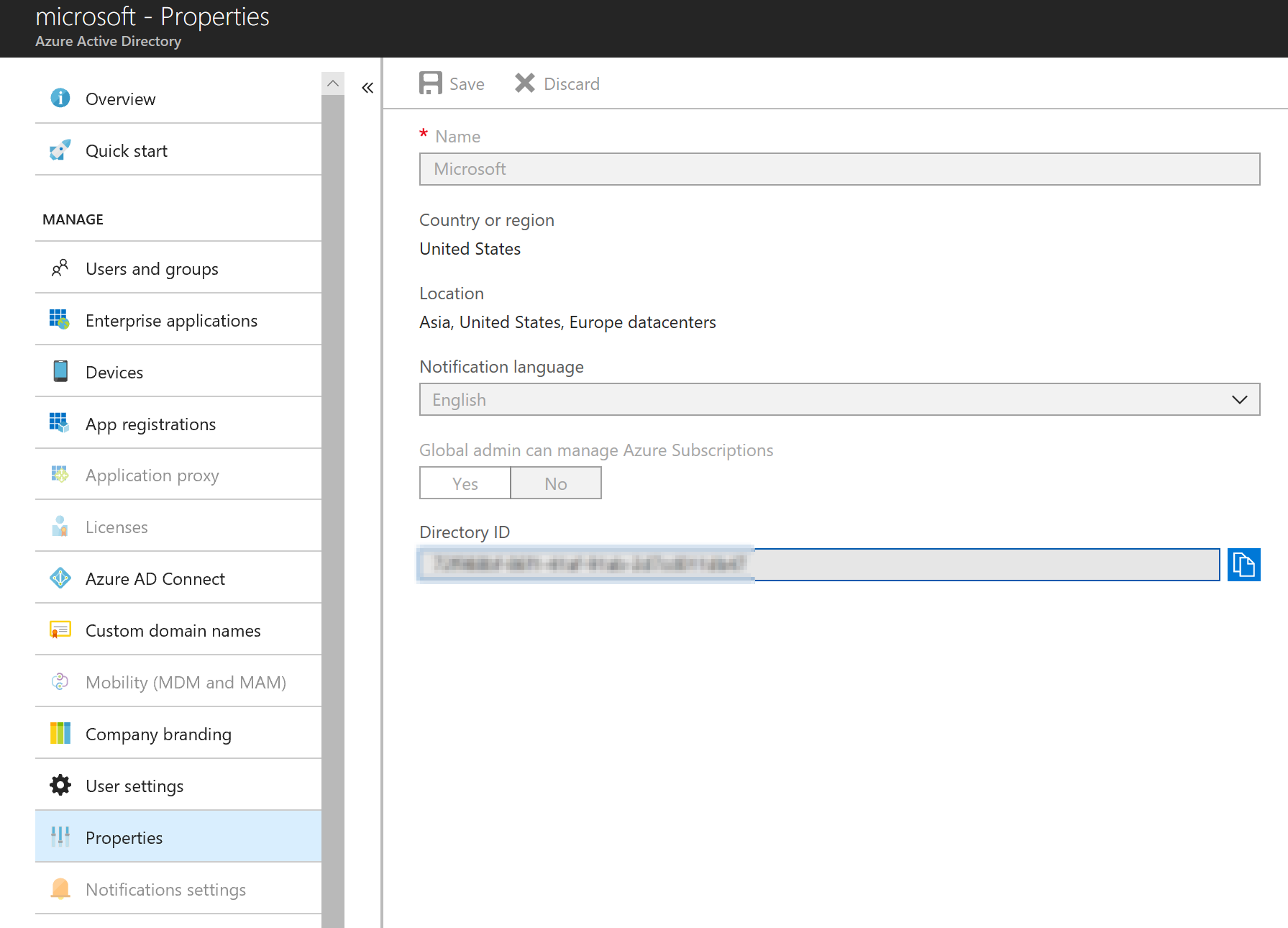
**Configuring Azure Blob Storage File Loads**

Open the Azure Portal (portal.azure.com) and open your Storage Account. Select Access Keys to display the following. Copy one of the connection strings (it does not matter which) and past the whole value into the StorageConnectionString value.

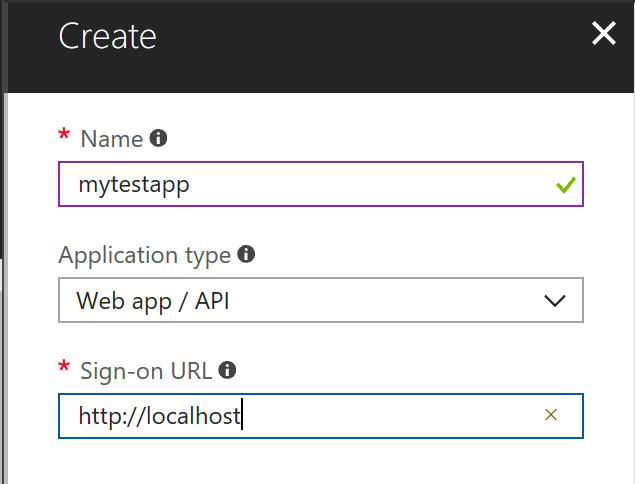


**Configuration of Azure Data Lake Credentials**

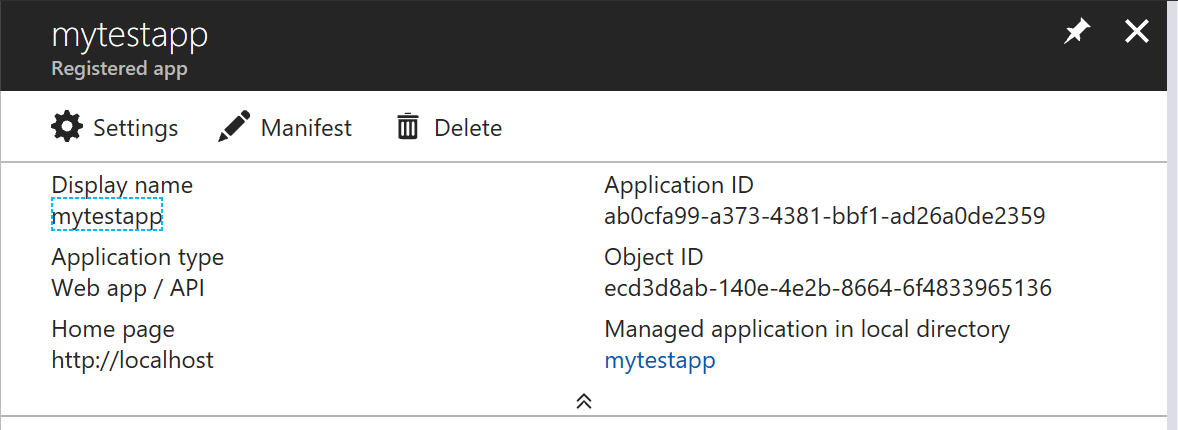
The setup of an AppID for accessing ADLS is a bit involved. You first need to create a Data Lake Store in the Azure Portal. Next go to your Azure Active Directory and select Properties to find your tenant ID (Directory ID) – paste this into the value for TenantID.



Next, go to App Registrations in AAD and create a new Application Registration. Enter the application name (make it memorable, because you need to find it later), select “Web App / API” and use <http://localhost> as the sign-on URL;

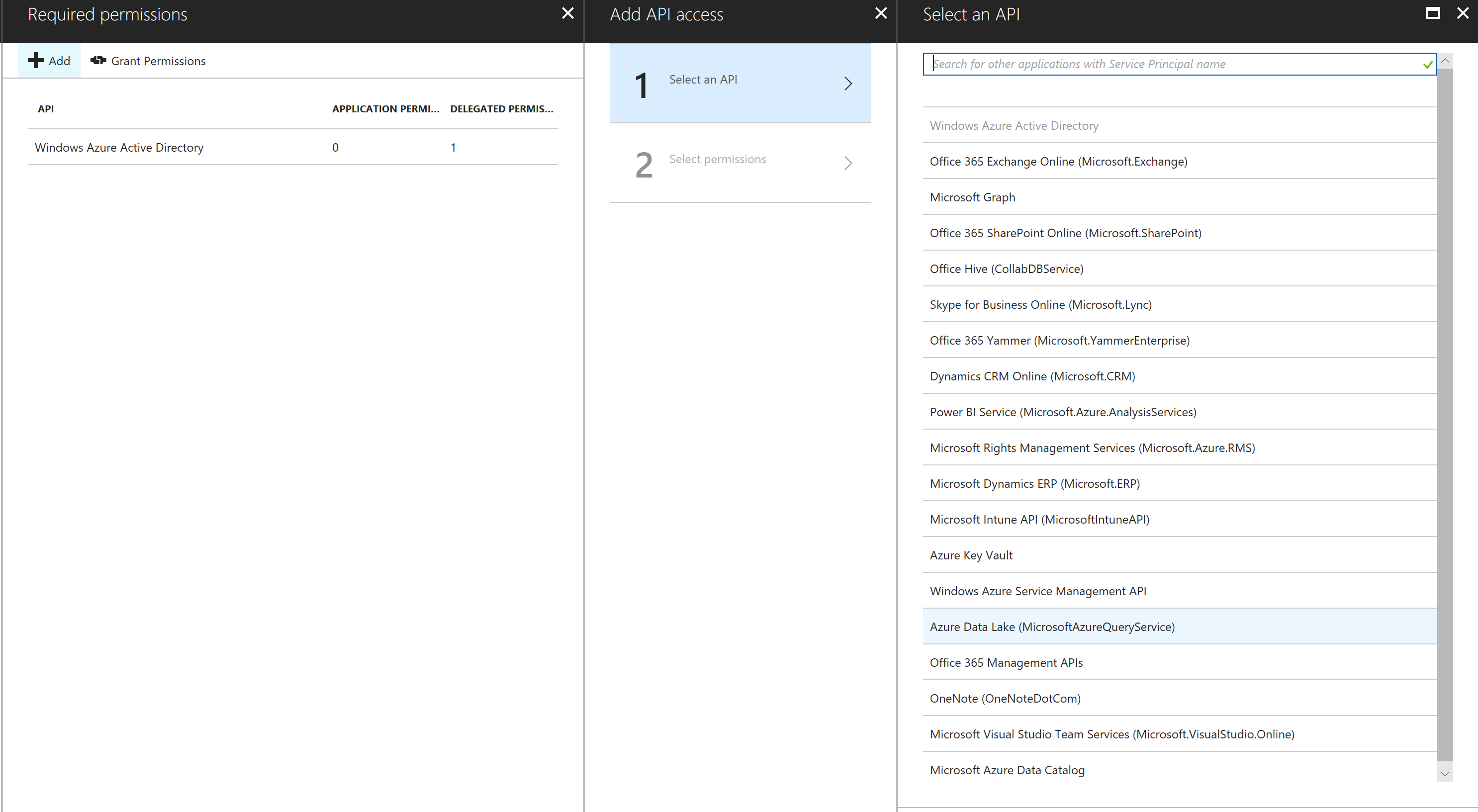


Select the app in the list;

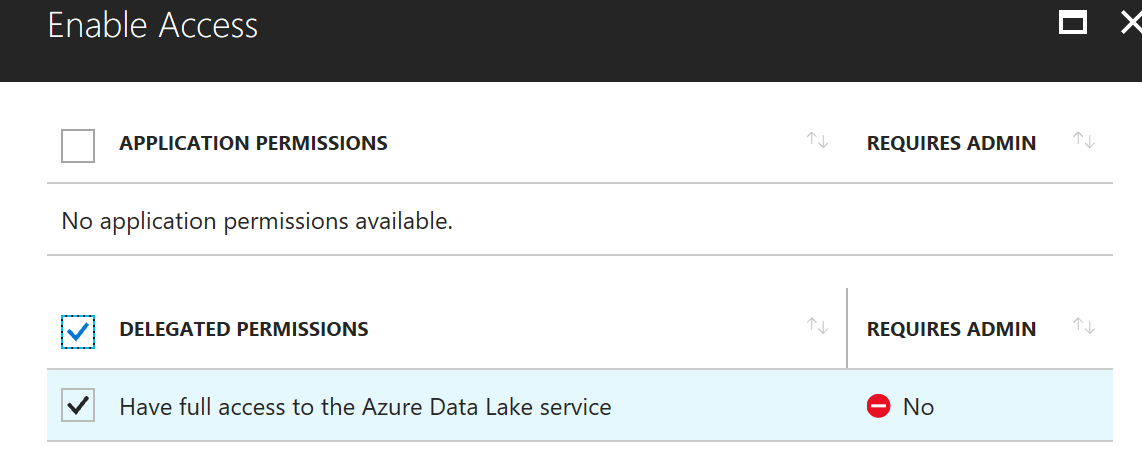


Copy the Application ID and paste it into the value for ApplicationID.

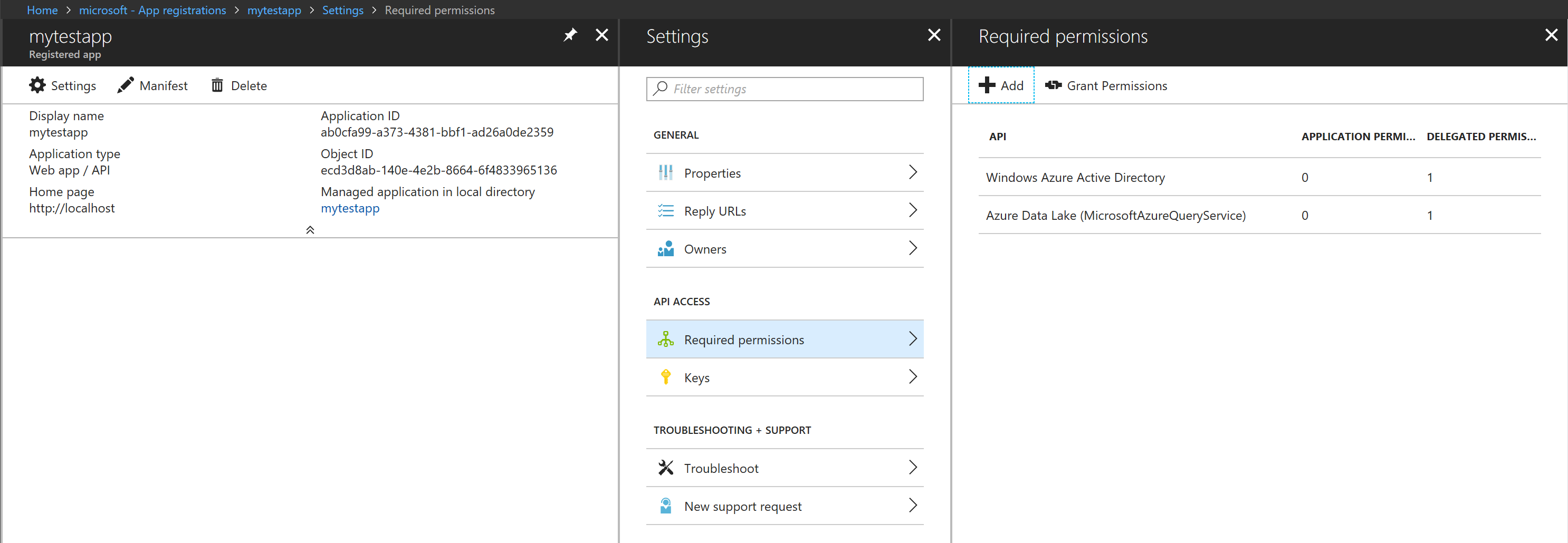
Then select Settings, select Required Permissions, Add, Select an API and Azure Data Lake;



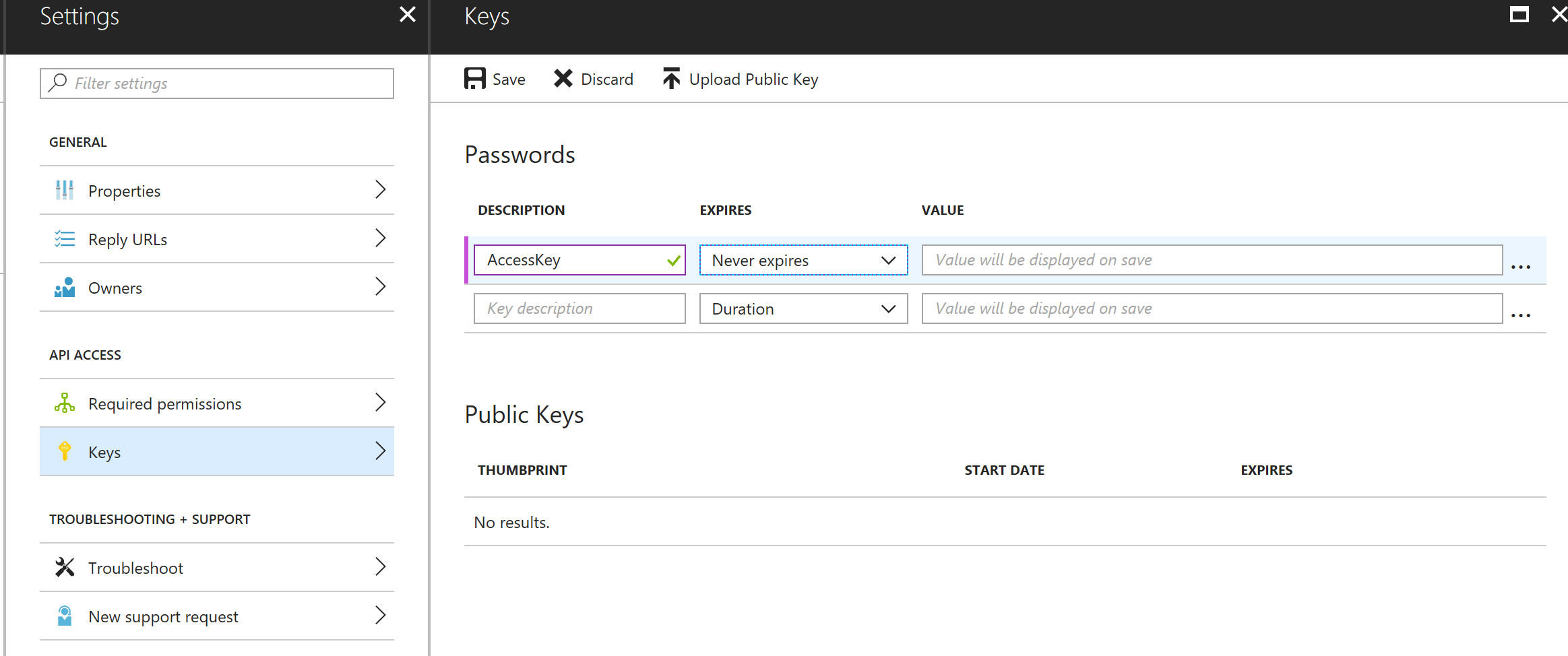
Click on Select and check Delegated Permissions



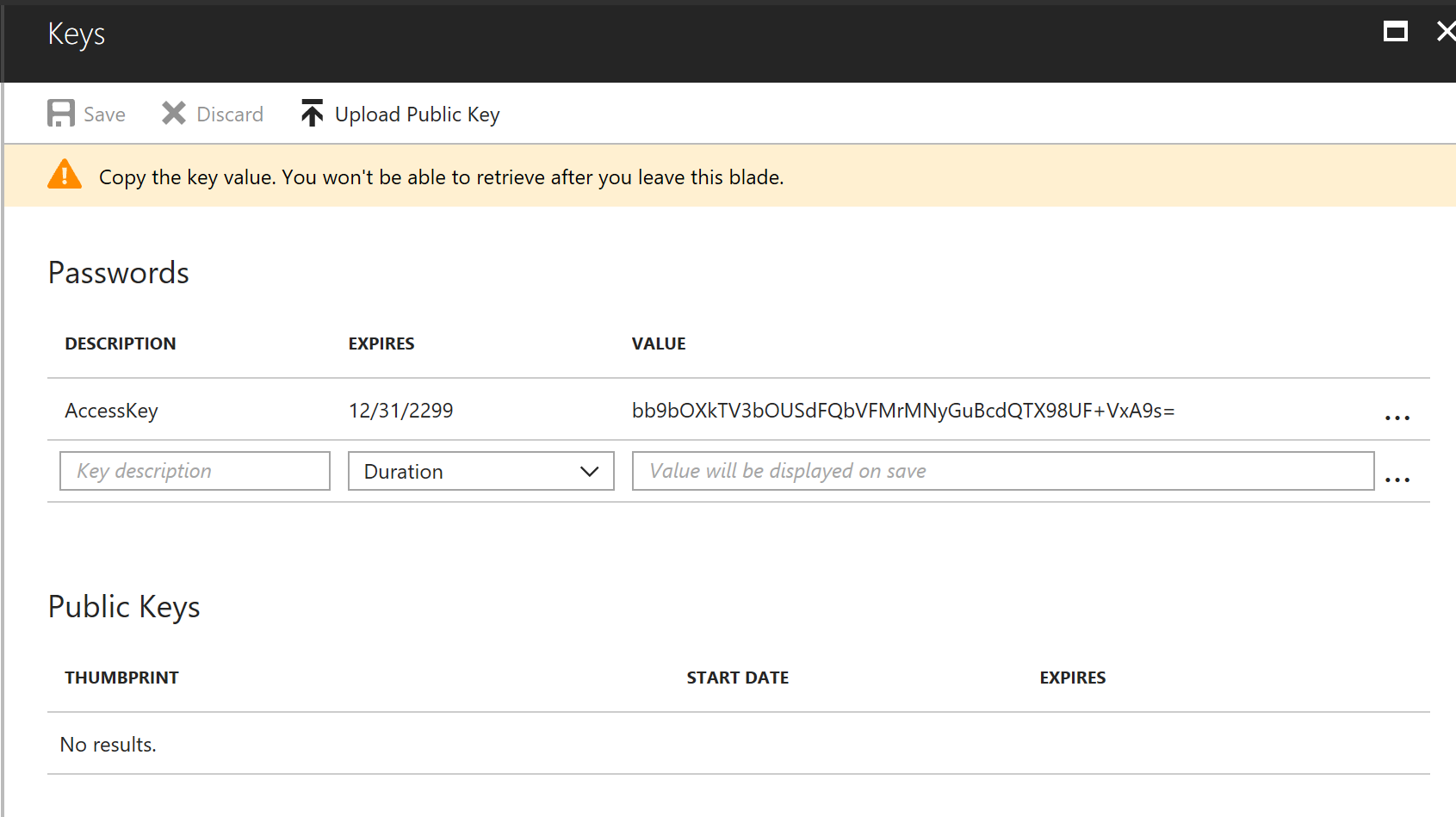
Press Select and Done.



Select Keys – Enter a key description and duration;



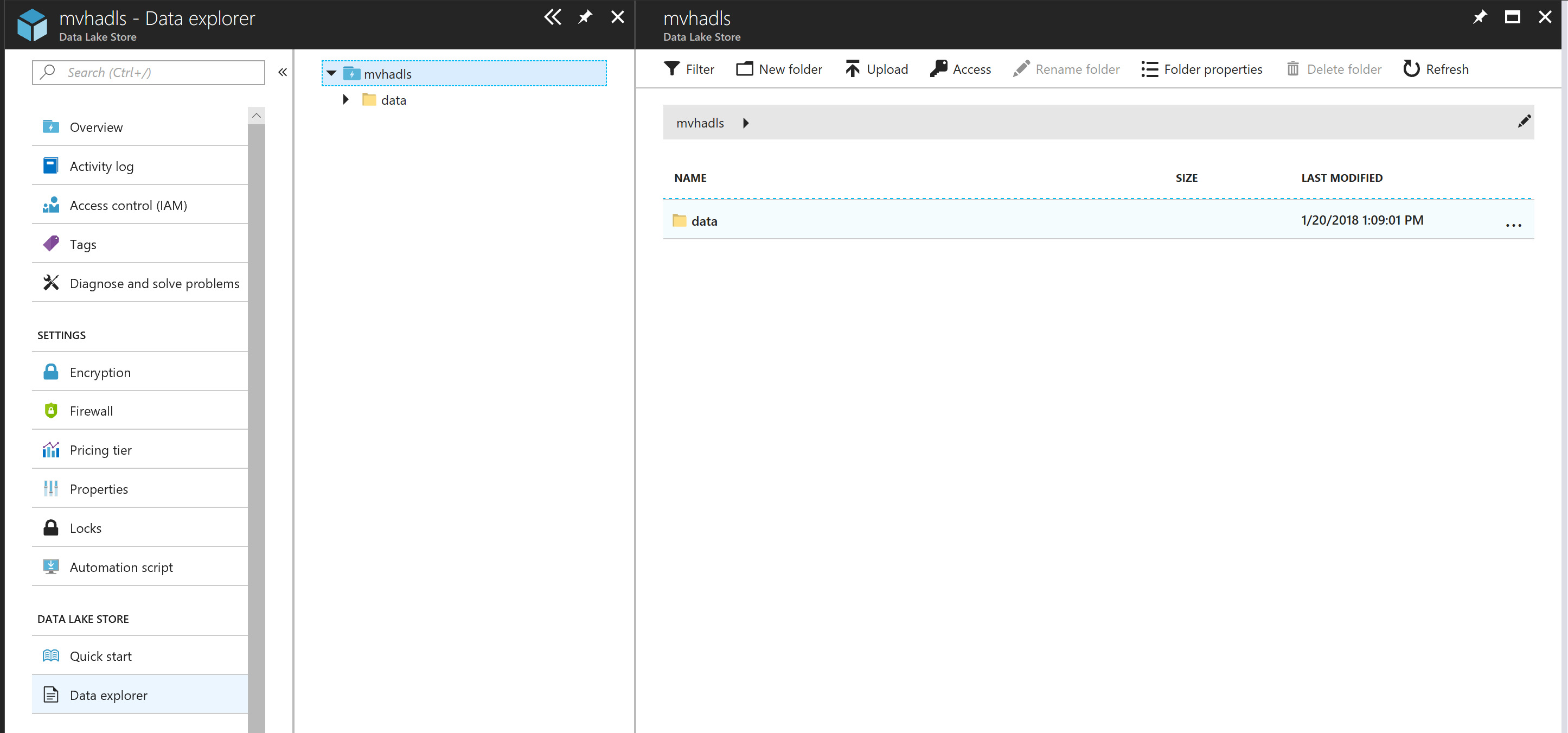
Press Save



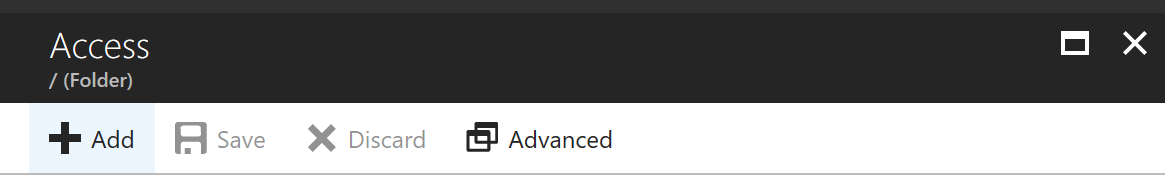
Copy the key value before closing the blade. Paste this into the value for SecretKey.

Go to your Azure Data Lake Storage icon and open it. From the Overview page copy the URL property of the Data Lake and paste it as the value of ADLSAccount – remove the https:// so that only account.azuredatalakestore.net remains in the value.

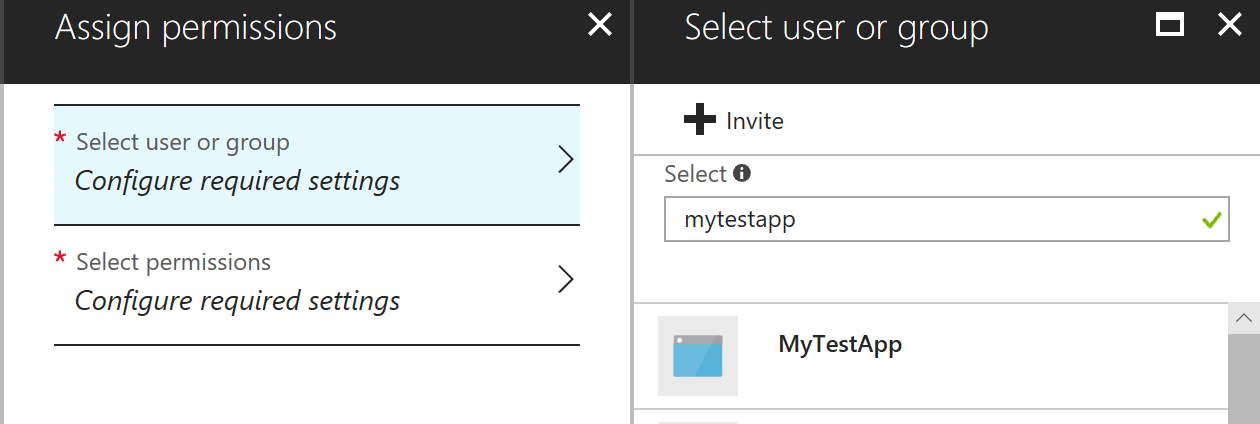
Click on Data Explorer – apply permissions to the top level of the folder structure;



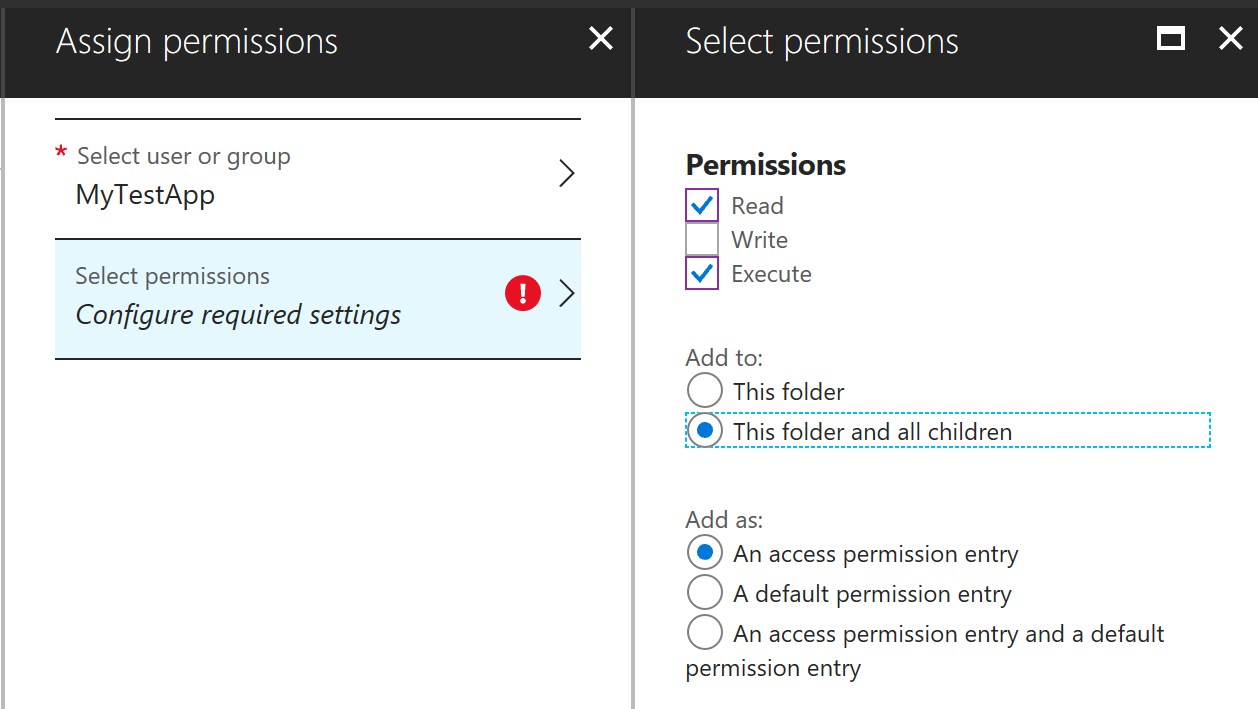
Click on Access;



Click on Add



Find the app you created and give read and execute (it needs read and execute on all parent folders to list files).



Paste the info gathered into the app.config and you can now use the adl:// moniker.