**Learning objectives**

In this module, you will learn how to analyze and optimize the Azure Synapse dedicated SQL pools' data storage. You will discover techniques to understand table space usage and column store storage details. Next, you will learn how to compare storage requirements between identical tables that use different data types. Finally, you will observe the impact materialized views have when executed in place of complex queries and learn how to avoid extensive logging by optimizing delete operations.

In this module, you will be able to:

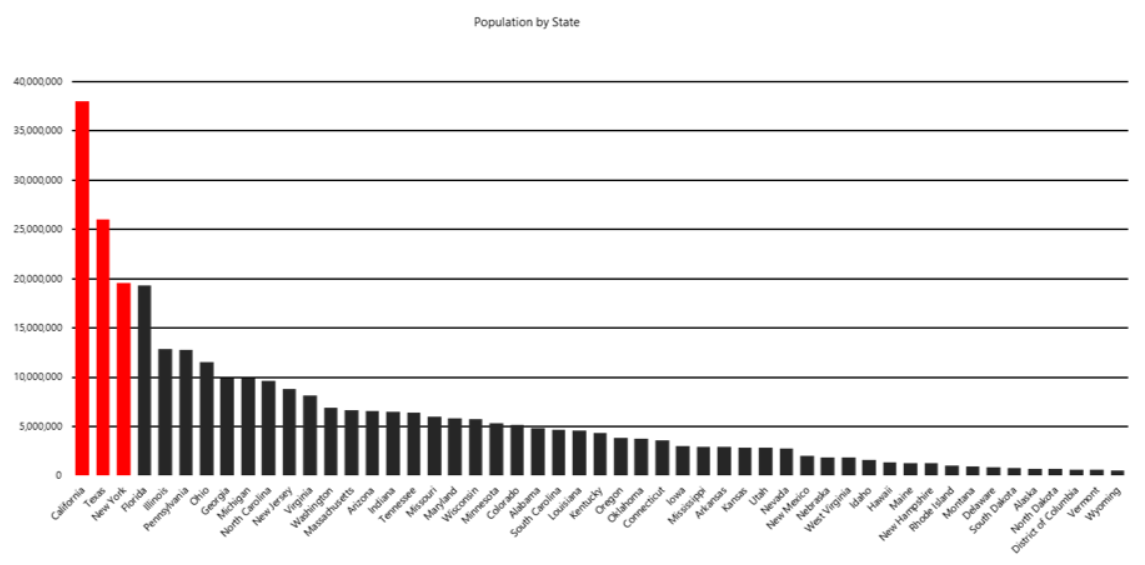
* Check for skewed data and space usage
* Understand column store storage details
* Study the impact of materialized views
* Explore rules for minimally logged operations

Before taking this module, it is recommended that you can:

* Log into the Azure portal
* Create a Synapse Analytics Workspace
* Create to and connect to an Azure Synapse Analytics SQL Pool

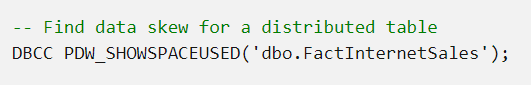
# Understand skewed data and space usage

In simple terms, data skew is an over-represented value. Imagine that you have assigned 50 tax examiners to audit tax returns, one examiner for each US state. The Wyoming examiner, because the population there is small, has little to do. In California, however, the examiner is kept busy because of the state's large population.



In our scenario, the data is unevenly distributed across all tax examiners, which means that some examiners must work more than others. In your job, you frequently experience situations like the tax-examiner example here. In more technical terms, one vertex gets much more data than its peers, a condition that makes the vertex work more than the others and that eventually slows down an entire job. What's worse, the job might fail because vertices might have, for example, a 5-hour runtime limitation and a 6-GB memory limitation.

A quick way to check for data skew is to use [DBCC PDW\_SHOWSPACEUSED](https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-pdw-showspaceused-transact-sql?toc=/azure/synapse-analytics/sql-data-warehouse/toc.json&bc=/azure/synapse-analytics/sql-data-warehouse/breadcrumb/toc.json&view=azure-sqldw-latest&preserve-view=true). The following SQL code returns the number of table rows that are stored in each of the 60 distributions. For balanced performance, the rows in your distributed table should be spread evenly across all the distributions.



Another aspect of data storage in Azure Synapse dedicated SQL pools is to monitor the table data space usage and observe its relationship with different table distribution types. Additionally, it is helpful to know the number of rows and the storage space used for indexing. Below is a list of [System Dynamic Management Views](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/system-dynamic-management-views?view=sql-server-ver15) (DMVs) that you can use to dig for the information. During the next exercise you will create a view using these DMVs to get a better view of the data.

| **Table Name** | **Description** |
| --- | --- |
| sys.schemas | All schemas in the database. |
| sys.tables | All tables in the database. |
| sys.indexes | All indexes in the database. |
| sys.columns | All columns in the database. |
| sys.pdw\_table\_mappings | Maps each table to local tables on physical nodes and distributions. |
| sys.pdw\_nodes\_tables | Contains information on each local table in each distribution. |
| sys.pdw\_table\_distribution\_properties | Holds distribution information for tables (the type of distribution tables have). |
| sys.pdw\_column\_distribution\_properties | Holds distribution information for columns. Filtered to include only columns used to distribute their parent tables (distribution\_ordinal = 1). |
| sys.pdw\_distributions | Holds information about the distributions from the SQL pool. |
| sys.dm\_pdw\_nodes | Holds information about the nodes from the SQL pool. Filtered to include only compute nodes (type = COMPUTE). |
| sys.dm\_pdw\_nodes\_db\_partition\_stats | Returns page and row-count information for every partition in the current database. |

Exercise:

<https://learn.microsoft.com/en-us/training/modules/analyze-optimize-data-warehouse-storage-azure-synapse-analytics/3-exercise-check-for-skewed-data-space-usage>

# Understand column store storage details

Since a columnstore index scans a table by scanning column segments of individual row groups, maximizing the number of rows in each row group enhances query performance. When rowgroups have a high number of rows, data compression improves which means there is less data to read from disk. The number of rows in a rowgroup determines the rowgroup's quality.

For best query performance, the goal is to maximize the number of rows per rowgroup in a columnstore index. A rowgroup can have a maximum of 1,048,576 rows. It's okay to not have the maximum number of rows per rowgroup. Columnstore indexes achieve good performance when rowgroups have at least 100,000 rows.

During a bulk load or columnstore index rebuild, sometimes there isn't enough memory available to compress all the rows designated for each rowgroup. When memory pressure is present, columnstore indexes trim the rowgroup sizes so compression into the columnstore can succeed.

The DMV [sys.dm\_pdw\_nodes\_db\_column\_store\_row\_group\_physical\_stats](https://learn.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-column-store-row-group-physical-stats-transact-sql?toc=/azure/synapse-analytics/sql-data-warehouse/toc.json&bc=/azure/synapse-analytics/sql-data-warehouse/breadcrumb/toc.json&view=azure-sqldw-latest&preserve-view=true) exposes useful information such as number of rows in rowgroups and the reason for trimming, if there was trimming.

There are two columns from dm\_pdw\_nodes\_db\_column\_store\_row\_group\_physical\_stats worth looking into in detail. The state\_desc column provides useful information on the state of a row group:

For additional content on optimizing row group and clustered columnstore indexes review the content [Investigating Row Group Health](https://learn.microsoft.com/en-us/training/modules/optimize-data-warehouse-query-performance-azure-synapse-analytics/5-use-indexes-to-improve)

| **Name** | **Description** |
| --- | --- |
| INVISIBLE | A rowgroup which is being compressed. |
| OPEN | A deltastore rowgroup that is accepting new rows. It is important to remember that an open rowgroup is still in rowstore format and has not been compressed to columnstore format. |
| CLOSED | A deltastore rowgroup that contains the maximum number of rows, and is waiting for the tuple mover process to compress it to the columnstore. |
| COMPRESSED | A row group that is compressed with columnstore compression and stored in the columnstore. |
| TOMBSTONE | A row group that was formerly in the deltastore and is no longer used. |

Additionally, the trim\_reason\_desc column describes the reason that triggered the COMPRESSED rowgroup to have less than the maximum number of rows:

| **Name** | **Description** |
| --- | --- |
| UNKNOWN\_UPGRADED\_FROM\_PREVIOUS\_VERSION | Occurred when upgrading from the previous version of SQL Server. |
| NO\_TRIM | The row group was not trimmed. The row group was compressed with the maximum of 1,048,576 rows. The number of rows could be less if a subset of rows was deleted after delta rowgroup was closed. |
| BULKLOAD | The bulk-load batch size limited the number of rows. This is what you should be looking for when optimizing data loading, as it is an indicator of resource starvation during the loading process. |
| REORG | Forced compression as part of REORG command. |
| DICTIONARY\_SIZE | Dictionary size grew too large to compress all of the rows together. |
| MEMORY\_LIMITATION | Not enough available memory to compress all the rows together. |
| RESIDUAL\_ROW\_GROUP | Closed as part of last row group with rows < 1 million during index build operation. |

# Exercise - View column store storage details

<https://learn.microsoft.com/en-us/training/modules/analyze-optimize-data-warehouse-storage-azure-synapse-analytics/5-exercise-view-column-store-storage-details>

# Understand the impact of wrong choices for column data types

Dedicated SQL pools support the most commonly used data types. With that, it is essential to keep in mind that minimizing the size of data types shortens the row length, which leads to better query performance. Use the smallest data type that works for your data:

* Avoid defining character columns with a large default length. For example, if the longest value is 25 characters, define your column as VARCHAR(25).
* Avoid using NVARCHAR when you only need VARCHAR to support non-unicode data.
* When possible, use NVARCHAR(4000) or VARCHAR(8000) instead of NVARCHAR(MAX) or VARCHAR(MAX).
* Focus on using the minimal data type as possible. Use the minor data type that can reliably contain all possible values you have. For example, use tinyint instead of smallint or int for exact-numbers between 0-255. tinyint would be sufficient for a person's age because no one lives to be more than 255 years old.
* Use DATETIME instead of storing date values in different formats such as string or numeric values. When it comes to querying your data based on date, an additional conversion will put a massive overhead on your queries.
* If you only need to store a date and not the time make sure you use DATE instead of DATETIME. There is a 4 bytes difference between the two.

**Note**

If you are using PolyBase external tables to load your SQL pool tables, the table row's defined length cannot exceed 1 MB. When a row with variable-length data exceeds 1 MB, you can load the row with BCP but not with PolyBase.

**Workarounds for unsupported data types**

The following list shows the data types that dedicated SQL pool doesn't support and gives useful alternatives for unsupported data types.

| **Unsupported data type** | **Workaround** |
| --- | --- |
| geometry | varbinary |
| geography | varbinary |
| hierarchyid | nvarchar |
| image | varbinary |
| text | varchar |
| ntext | nvarchar |
| sql\_variant | Split column into several strongly typed columns. |
| table | Convert to temporary tables. |
| timestamp | Rework code to use datetime2 and the CURRENT\_TIMESTAMP function. Only constants are supported as defaults, so current\_timestamp can't be defined as a default constraint. If you need to migrate row version values from a timestamp typed column, use BINARY(8) or VARBINARY(8) for NOT NULL or NULL row version values. |
| xml | varchar |
| user-defined type | Convert back to the native data type when possible. |
| default values | Default values support literals and constants only. |

# Exercise - Compare storage requirements between optimal and sub-optimal column data types

<https://learn.microsoft.com/en-us/training/modules/analyze-optimize-data-warehouse-storage-azure-synapse-analytics/7-exercise-compare-storage-requirements>

# Describe the impact of materialized views

Materialized views in the Azure Synapse SQL pool provide a low maintenance method for complex analytical queries to quickly perform without any query change. SQL pool in Azure Synapse supports standard and materialized views. Both are virtual tables created with SELECT expressions and presented to queries as logical tables. Views encapsulate the complexity of common data computation and add an abstraction layer to computation changes, so there's no need to rewrite queries.

| **Comparison** | **View** | **Materialized view** |
| --- | --- | --- |
| View definition | Stored in SQL pool. | Stored in SQL pool. |
| View content | Generated each time when the view is used. | Pre-processed and stored in SQL pool during view creation. Updated as data is added to the underlying tables. |
| Data refresh | Always updated | Always updated |
| Speed to retrieve view data from complex queries | Slow | Fast |
| Extra storage | No | Yes |
| Syntax | CREATE VIEW | CREATE MATERIALIZED VIEW AS SELECT |

A materialized view pre-computes, stores, and maintains its data in SQL pool just like a table. There's no recomputation needed each time when a materialized view is used. That's why queries that use all or a subset of the data in materialized views can get faster performance. Even better, queries can use a materialized view without directly referencing it, so there's no need to change the application code.

An adequately designed materialized view provides the following benefits:

* Reduce the execution time for complex queries with JOINs and aggregate functions. The more complex the query, the higher the potential for execution-time saving. The most benefit is gained when a query's computation cost is high, and the resulting data set is small.
* The optimizer in the SQL pool can automatically use deployed materialized views to improve query execution plans. This process is transparent to users providing faster query performance, and doesn't require queries to make direct reference to the materialized views.
* Requires low maintenance on the views. All incremental data changes from the base tables are automatically added to the materialized views synchronously. This design allows querying materialized views to return the same data as directly querying the base tables.
* The data in a materialized view can be distributed differently from the base tables.
* Data in materialized views gets the same high availability and resiliency benefits as data in regular tables.

Comparing to other data warehouse providers, the materialized views implemented in Azure Synapse Analytics also provide the following additional benefits:

* Automatic and synchronous data refresh with data changes in base tables. No user action is required.
* Broad aggregate function support. See [CREATE MATERIALIZED VIEW AS SELECT (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/statements/create-materialized-view-as-select-transact-sql?toc=/azure/synapse-analytics/sql-data-warehouse/toc.json&bc=/azure/synapse-analytics/sql-data-warehouse/breadcrumb/toc.json&view=azure-sqldw-latest&preserve-view=true).
* The support for query-specific materialized view recommendation. See [EXPLAIN (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/queries/explain-transact-sql?toc=/azure/synapse-analytics/sql-data-warehouse/toc.json&bc=/azure/synapse-analytics/sql-data-warehouse/breadcrumb/toc.json&view=azure-sqldw-latest&preserve-view=true).

Before you begin to create materialized views, it's essential to have a deep understanding of your workload in terms of query patterns, importance, frequency, and resulting data size. Users can run EXPLAIN WITH\_RECOMMENDATIONS <SQL\_statement> to get the materialized views recommended by the query optimizer. Since these recommendations are query-specific, a materialized view that benefits a single query may not be optimal for other queries in the same workload.

# Exercise - Improve the execution plan of a query with a materialized view

<https://learn.microsoft.com/en-us/training/modules/analyze-optimize-data-warehouse-storage-azure-synapse-analytics/9-exercise-improve-execution-plan-of-query-materialized-view>

# Understand rules for minimally logged operations

**Rules for minimally logged operations**

Unlike fully logged operations, which use the transaction log to keep track of every row change, minimally logged operations keep track of extent allocations and meta-data changes only. Therefore, minimal logging involves logging only the required information to roll back the transaction after a failure or for an explicit request (ROLLBACK TRAN). If the database is damaged or lost when minimal logging is in effect, you cannot recover the database to the point of failure. As much less information is tracked in the transaction log, a minimally logged operation performs better than a similarly sized fully logged operation. Furthermore, because fewer writes go the transaction log, a much smaller amount of log data is generated, and so is more I/O efficient.

Some operations can be minimally logged to reduce their impact on transaction log size. The following operations are capable of being minimally logged:

* CREATE TABLE AS SELECT (CTAS)
* INSERT..SELECT
* CREATE INDEX
* ALTER INDEX REBUILD
* DROP INDEX
* TRUNCATE TABLE
* DROP TABLE
* ALTER TABLE SWITCH PARTITION

**Minimal logging with bulk load**

CTAS and INSERT...SELECT are both bulk load operations. However, both are influenced by the target table definition and depend on the load scenario. The following table explains when bulk operations are fully or minimally logged:

| **Primary Index** | **Load Scenario** | **Logging Mode** |
| --- | --- | --- |
| Heap | Any | **Minimal** |
| Clustered Index | Empty target table | **Minimal** |
| Clustered Index | Loaded rows do not overlap with existing pages in target | **Minimal** |
| Clustered Index | Loaded rows overlap with existing pages in target | Full |
| Clustered Columnstore Index | Batch size >= 102,400 per partition aligned distribution | **Minimal** |
| Clustered Columnstore Index | Batch size < 102,400 per partition aligned distribution | Full |

It is worth noting that any writes to update secondary or non-clustered indexes will always be fully logged operations.

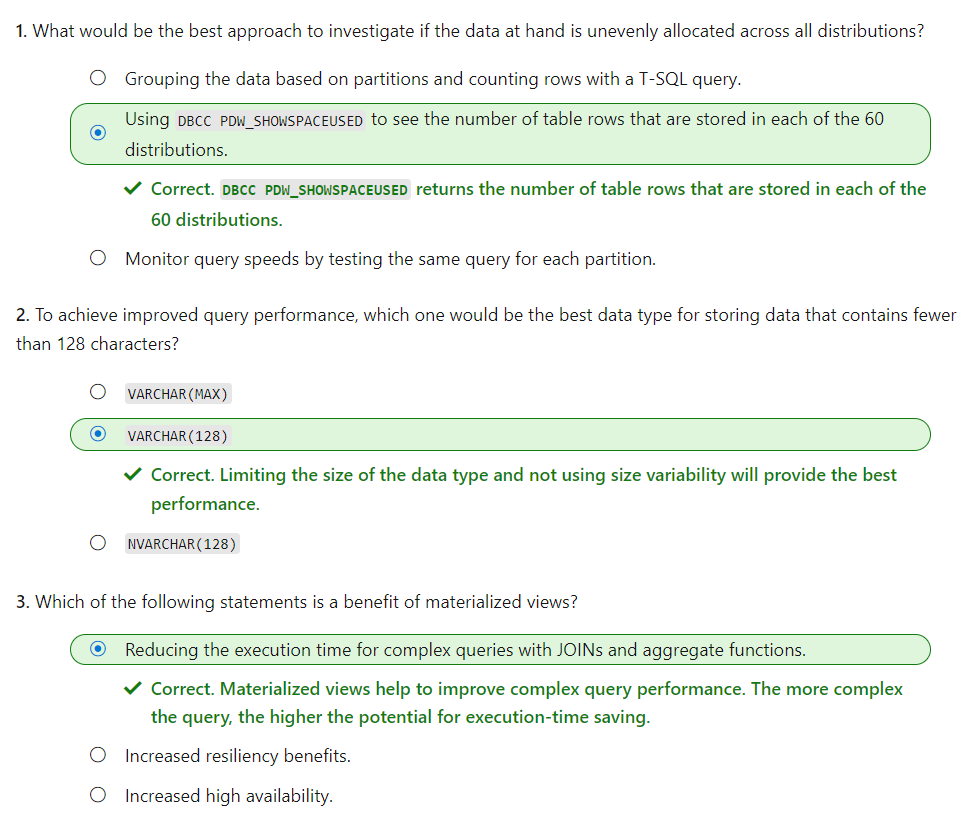
**Important**

A Synapse Analytics SQL pool has 60 distributions. Therefore, assuming all rows are evenly distributed and landing in a single partition, your batch will need to contain 6,144,000 rows or larger to be minimally logged when writing to a Clustered Columnstore Index. If the table is partitioned and the rows being inserted span partition boundaries, then you will need 6,144,000 rows per partition boundary assuming even data distribution. Each partition in each distribution must independently exceed the 102,400 row threshold for the insert to be minimally logged into the distribution.

Loading data into a non-empty table with a clustered index can often contain a mixture of fully logged and minimally logged rows. A clustered index is a balanced tree (b-tree) of pages. If the page being written to already contains rows from another transaction, then these writes will be fully logged. However, if the page is empty then the write to that page will be minimally logged.

## Optimize delete operations

<https://learn.microsoft.com/en-us/training/modules/analyze-optimize-data-warehouse-storage-azure-synapse-analytics/11-exercise-optimize-delete-operation>



# **Secure a data warehouse in Azure Synapse Analytics**

In this module, you will learn how to approach and implement security to protect your data with Azure Synapse Analytics.

In this module, you will:

* Understand network security options for Azure Synapse Analytics
* Configure Conditional Access
* Configure Authentication
* Manage authorization through column and row level security
* Manage sensitive data with Dynamic Data masking
* Implement encryption in Azure Synapse Analytics
* Understand advanced data security options for Azure Synapse Analytics

**Prerequisites**

Before taking this module, it is recommended that the student is able to:

* Log into the Azure portal
* Create a Synapse Analytics Workspace
* Create an Azure Synapse Analytics SQL Pool

# Understand network security options for Azure Synapse Analytics

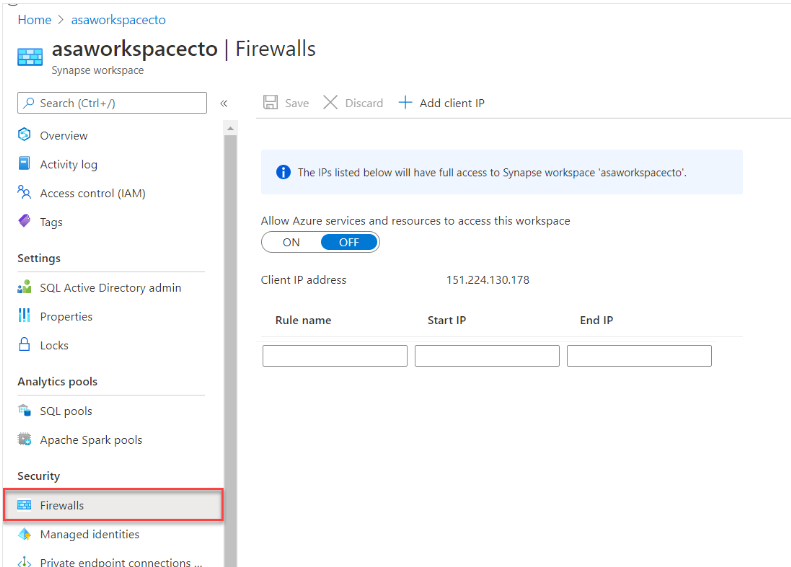
There are a range of network security steps that you should consider to secure Azure Synapse Analytics. One of the first aspects that you will consider is securing access to the service itself. This can be achieved by creating the following network objects including:

* Firewall rules
* Virtual networks
* Private endpoints

## Firewall rules

Firewall rules enable you to define the type of traffic that is allowed or denied access to an Azure Synapse workspace using the originating IP address of the client that is trying to access the Azure Synapse Workspace. IP firewall rules configured at the workspace level apply to all public endpoints of the workspace including dedicated SQL pools, serverless SQL pool, and the development endpoint.

You can choose to allow connections from all IP addresses as you are creating the Azure Synapse Workspaces, although this is not recommended as it does not allow for control access to the workspace. Instead, within the Azure portal, you can configure specific IP address ranges and associate them with a rule name so that you have greater control.



Make sure that the firewall on your network and local computer allows outgoing communication on TCP ports 80, 443 and 1443 for Synapse Studio.

Also, you need to allow outgoing communication on UDP port 53 for Synapse Studio. To connect using tools such as SSMS and Power BI, you must allow outgoing communication on TCP port 1433.

**Virtual networks**

Azure Virtual Network (VNet) enables private networks in Azure. VNet enables many types of Azure resources, such as Azure Synapse Analytics, to securely communicate with other virtual networks, the internet, and on-premises networks. When you create your Azure Synapse workspace, you can choose to associate it to a Microsoft Azure Virtual Network. The Virtual Network associated with your workspace is managed by Azure Synapse. This Virtual Network is called a Managed workspace Virtual Network.

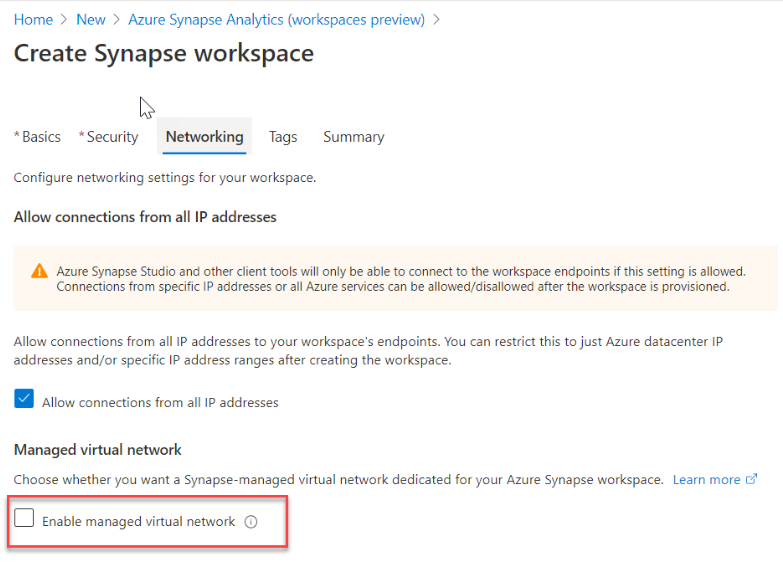
Using a managed workspace virtual network provides the following benefits:

* With a Managed workspace Virtual Network, you can offload the burden of managing the Virtual Network to Azure Synapse.
* You don't have to configure inbound NSG rules on your own Virtual Networks to allow Azure Synapse management traffic to enter your Virtual Network. Misconfiguration of these NSG rules causes service disruption for customers.
* You don't need to create a subnet for your Spark clusters based on peak load.
* Managed workspace Virtual Network along with Managed private endpoints protects against data exfiltration. You can only create Managed private endpoints in a workspace that has a Managed workspace Virtual Network associated with it.
* it ensures that your workspace is network isolated from other workspaces.

If your workspace has a Managed workspace Virtual Network, Data integration and Spark resources are deployed in it. A Managed workspace Virtual Network also provides user-level isolation for Spark activities because each Spark cluster is in its own subnet.

Dedicated SQL pool and serverless SQL pool are multi-tenant capabilities and therefore reside outside of the Managed workspace Virtual Network. Intra-workspace communication to dedicated SQL pool and serverless SQL pool use Azure private links. These private links are automatically created for you when you create a workspace with a Managed workspace Virtual Network associated to it.

You can only choose to enable managed virtual networks as you are creating the Azure Synapse Workspaces.

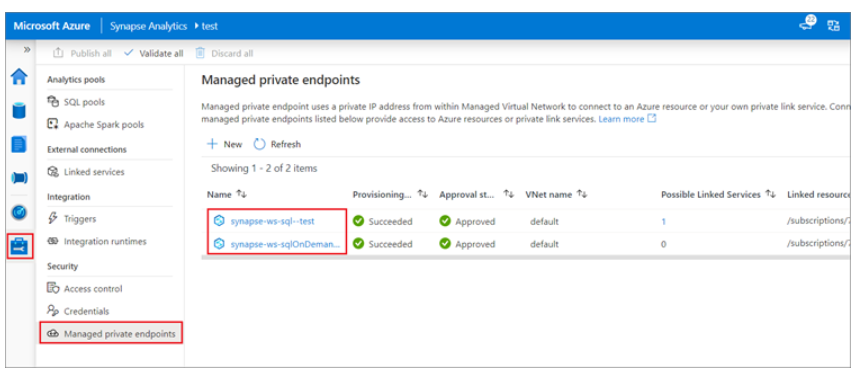


## Private endpoints

Azure Synapse Analytics enables you to connect up its various components through endpoints. You can set up managed private endpoints to access these components in a secure manner known as private links. This can only be achieved in an Azure Synapse workspace with a Managed workspace Virtual Network. Private link enables you to access Azure services (such as Azure Storage and Azure Cosmos DB) and Azure hosted customer/partner services from your Azure Virtual Network securely.

When you use a private link, traffic between your Virtual Network and workspace traverses entirely over the Microsoft backbone network. Private Link protects against data exfiltration risks. You establish a private link to a resource by creating a private endpoint.

Private endpoint uses a private IP address from your Virtual Network to effectively bring the service into your Virtual Network. Private endpoints are mapped to a specific resource in Azure and not the entire service. Customers can limit connectivity to a specific resource approved by their organization. You can manage the private endpoints in the Azure Synapse Studio manage hub.



# Configure Conditional Access

Conditional Access is a feature that enables you to define the conditions under which a user can connect to your Azure subscription and access services. Conditional Access provides an additional layer of security that can be used in combination with authentication to strengthen the security access to your network.

Conditional Access policies at their simplest are if-then statements, if a user wants to access a resource, then they must complete an action. As an example, if a Data Engineer wishes to access services in Azure Synapse Analytics, they may be requested by the Conditional Access policy to perform an additional step of multifactor authentication (MFA) to complete the authentication to get onto the service

Conditional Access policies use signals as a basis to determine if Conditional Access should first be applied. Common signals include:

* User or group membership names
* IP address information
* Device platforms or type
* Application access requests
* Real-time and calculated risk detection
* Microsoft Cloud App Security (MCAS)

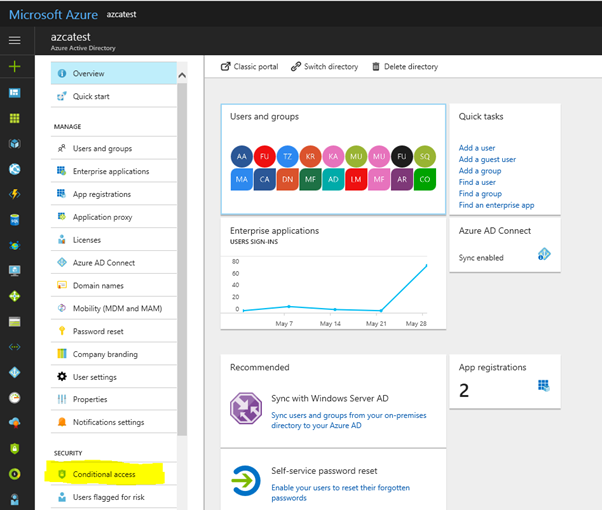
Based on these signals, you can then choose to block access. The alternative is you can grant access, and at the same time request that the user perform an additional action including:

* Perform multifactor authentication
* Use a specific device to connect

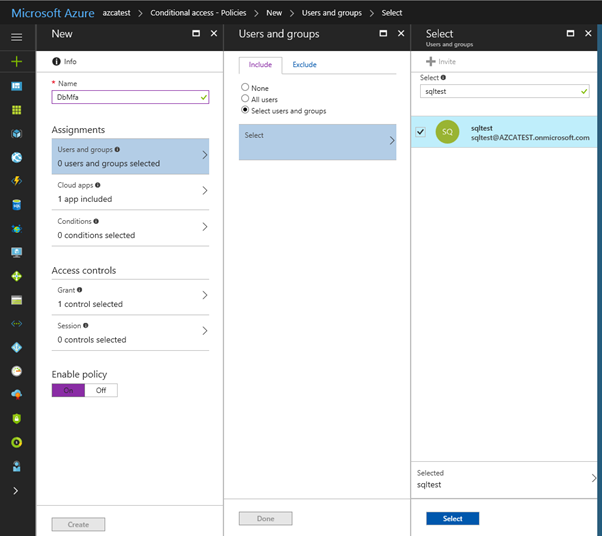
Given the amount of data that could potentially be stored, Azure Synapse Analytics dedicated SQL pools supports Conditional Access to provide protection for your data. It does require that Azure Synapse Analytics is configured to support Azure Active Directory, and that if you chose multifactor authentication, that the tool you are using support it.

To configure Conditional Access, you can perform the following steps:

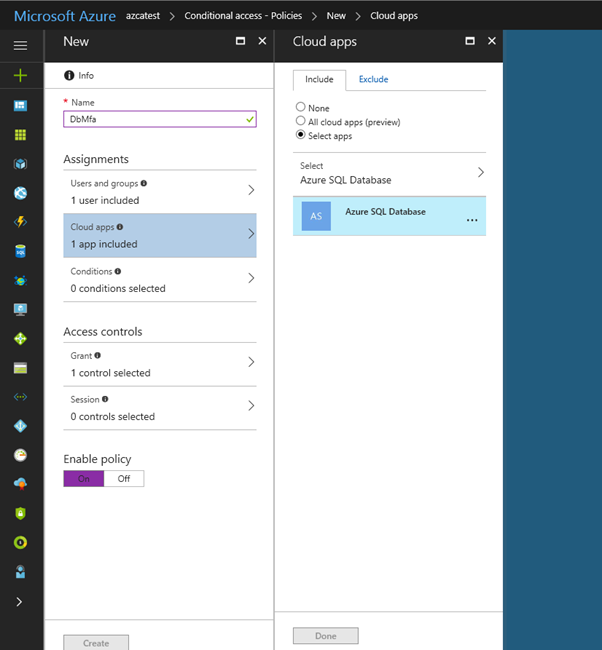
1. Sign in to the Azure portal, select **Azure Active Directory**, and then select **Conditional Access**.



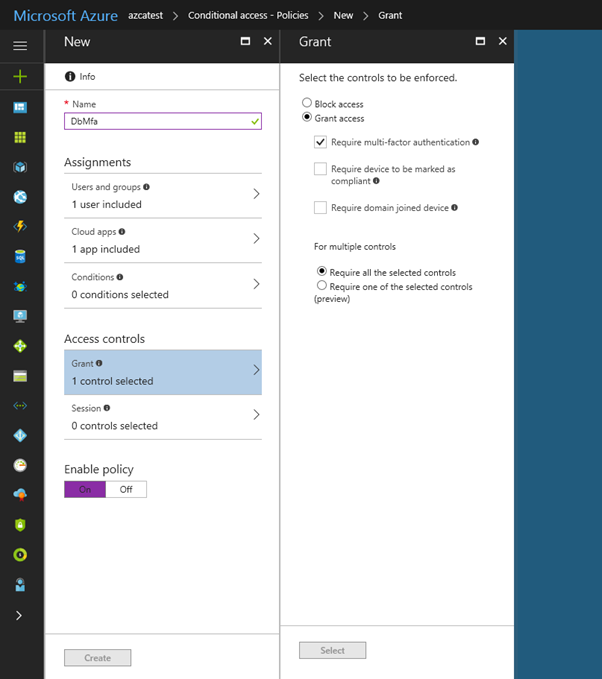
1. In the **Conditional Access-Policies** blade, click **New policy**, provide a name, and then click **Configure rules**.
2. Under **Assignments**, select **Users and groups**, check **Select users and groups**, and then select the user or group for Conditional Access. Click **Select**, and then click **Done** to accept your selection.



1. Select **Cloud apps**, click **Select apps**. You see all apps available for Conditional Access. Select **Azure SQL Database**, at the bottom click **Select**, and then click **Done**.



1. If you can't find **Azure SQL Database** listed in the following third screenshot, complete the following steps:
   * Connect to your database in Azure SQL Database by using SSMS with an Azure AD admin account.
   * Execute CREATE USER [user@yourtenant.com] FROM EXTERNAL PROVIDER.
   * Sign into Azure AD and verify that Azure SQL Database, SQL Managed Instance, or Azure Synapse are listed in the applications in your Azure AD instance.
2. Select **Access controls**, select **Grant**, and then check the policy you want to apply. For this example, we select **Require multifactor authentication**.



# Configure authentication

Authentication is the process of validating credentials as you access resources in a digital infrastructure. This ensures that you can validate that an individual, or a service that wants to access a service in your environment can prove who they are. Azure Synapse Analytics provides several different methods for authentication.

## What needs to be authenticated

There are a variety of scenarios that means that authentication must take place to protect the data that is stored in your Azure Synapse Analytics estate.

The common form of authentication is that of individuals who want to access the data in the service. This is typically seen as an individual providing a username and password to authenticate against a service. However, this is also becoming more sophisticated with authentication requests working in combination with Conditional Access policies to further secure the authentication process with additional security steps.

What is less obvious is the fact that services must authenticate with other services so that they can operate seamlessly. An example of this is using an Azure Synapse Spark or serverless SQL pool to access data in an Azure Data Lake store. An authentication mechanism must take place in the background to ensure that Azure Synapse Analytics can access the data in the data lake in an authenticated manner.

Finally, there are situations where users and services operate together at the same time. Here you have a combination of both user and service authentication taking place under the hood to ensure that the user is getting access to the data seamlessly. An example of this is using Power BI to view reports in a dashboard that is being serviced by a dedicated SQL pool. Here you have multiple levels of authentication taking place that needs to be managed.

## Types of security

The following are the types of authentication that you should be aware of when working with Azure Synapse Analytics.

### Azure Active Directory

Azure Active Directory is a directory service that allows you to centrally maintain objects that can be secured. The objects can include user accounts and computer accounts. An employee of an organization will typically have a user account that represents them in the organizations Azure Active Directory tenant, and they then use the user account with a password to authenticate against other resources that are stored within the directory using a process known as single sign-on.

The power of Azure Active Directory is that they only have to log in once, and Azure Active Directory will manage access to other resources based on the information held within it using pass through authentication. If a user and an instance of Azure Synapse Analytics are part of the same Azure Active Directory, it is possible for the user to access Azure Synapse Analytics without an apparent login. If managed correctly, this process is seamless as the administrator would have given the user authorization to access Azure Synapse Analytics dedicated SQL pool as an example.

In this situation, it is normal for an Azure Administrator to create the user accounts and assign them to the appropriate roles and groups in Azure Active Directory. The Data Engineer will then add the user, or a group to which the user belongs to access a dedicated SQL pool.

### Managed identities

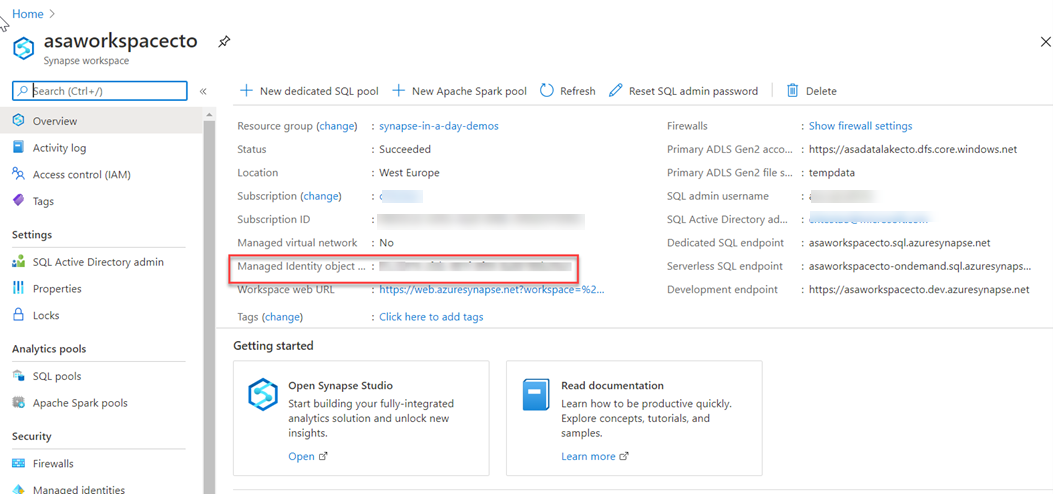
Managed identity for Azure resources is a feature of Azure Active Directory. The feature provides Azure services with an automatically managed identity in Azure AD. You can use the Managed Identity capability to authenticate to any service that supports Azure Active Directory authentication.

Managed identities for Azure resources are the new name for the service formerly known as Managed Service Identity (MSI). A system-assigned managed identity is created for your Azure Synapse workspace when you create the workspace.

Azure Synapse also uses the managed identity to integrate pipelines. The managed identity lifecycle is directly tied to the Azure Synapse workspace. If you delete the Azure Synapse workspace, then the managed identity is also cleaned up.

The workspace managed identity needs permissions to perform operations in the pipelines. You can use the object ID or your Azure Synapse workspace name to find the managed identity when granting permissions.

You can retrieve the managed identity in the Azure portal. Open your Azure Synapse workspace in Azure portal and select **Overview** from the left navigation. The managed identity's object ID is displayed to in the main screen.

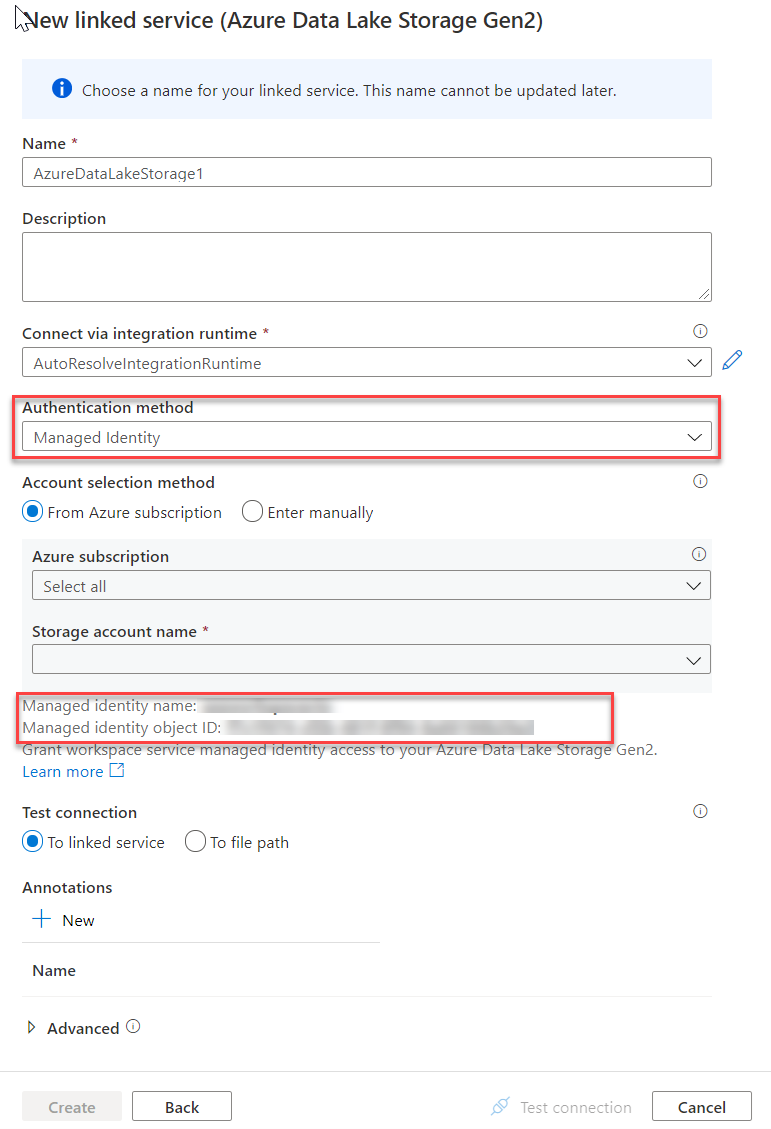


The managed identity information will also show up when you create a linked service that supports managed identity authentication from Azure Synapse Studio.

Launch **Azure Synapse Studio** and select the **Manage** tab from the left navigation. Then select **Linked services** and choose the **+ New** option to create a new linked service.

In the **New linked service** window, type Azure Data Lake Storage Gen2. Select the **Azure Data Lake Storage Gen2** resource type from the list below and choose **Continue**.

In the next window, choose **Managed Identity** for **Authentication method**. You'll see the managed identity's **Name** and **Object ID**.



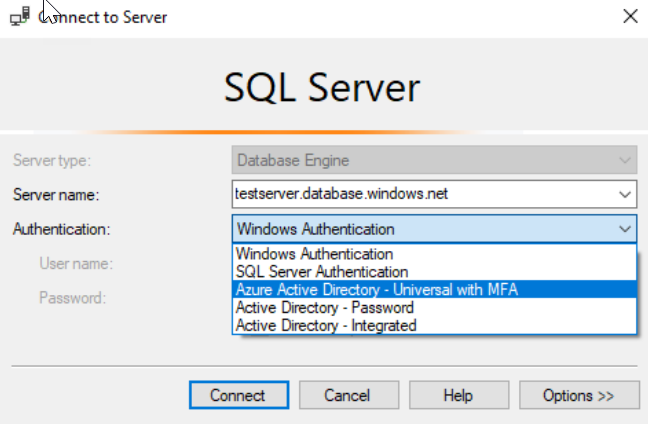
### SQL Authentication

For user accounts that are not part of an Azure Active directory, then using SQL Authentication will be an alternative. In this instance, a user is created in the instance of a dedicated SQL pool. If the user in question requires administrator access, then the details of the user are held in the master database. If administrator access is not required, you can create a user in a specific database. A user then connects directly to the Azure Synapse Analytics dedicated SQL pool where they are prompted to use a username and password to access the service.

This approach is typically useful for external users who need to access the data, or if you are using third party or legacy applications against the Azure Synapse Analytics dedicated SQL pool

### Multifactor authentication

Synapse SQL support connections from SQL Server Management Studio (SSMS) using Active Directory Universal Authentication.



This enables you to operate in environments that use Conditional Access policies that enforce multifactor authentication as part of the policy.

### Keys

If you are unable to use a managed identity to access resources such as Azure Data Lake then you can use storage account keys and shared access signatures.

With a storage account key. Azure creates two of these keys (primary and secondary) for each storage account you create. The keys give access to everything in the account. You'll find the storage account keys in the Azure portal view of the storage account. Just select **Settings**, and then click **Access keys**.

As a best practice, you shouldn't share storage account keys, and you can use Azure Key Vault to manage and secure the keys.

Azure Key Vault is a secret store: a centralized cloud service for storing app secrets - configuration values like passwords and connection strings that must remain secure at all times. Key Vault helps you control your apps' secrets by keeping them in a single central location and providing secure access, permissions control, and access logging.

The main benefits of using Key Vault are:

* Separation of sensitive app information from other configuration and code, reducing risk of accidental leaks
* Restricted secret access with access policies tailored to the apps and individuals that need them
* Centralized secret storage, allowing required changes to happen in only one place
* Access logging and monitoring to help you understand how and when secrets are accessed

Secrets are stored in individual vaults, which are Azure resources used to group secrets together. Secret access and vault management is accomplished via a REST API, which is also supported by all of the Azure management tools as well as client libraries available for many popular languages. Every vault has a unique URL where its API is hosted.

### Shared access signatures

If an external third-party application needs access to your data, you'll need to secure their connections without using storage account keys. For untrusted clients, use a shared access signature (SAS). A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access. You can give a customer a shared access signature token.

### Types of shared access signatures

You can use a service-level shared access signature to allow access to specific resources in a storage account. You'd use this type of shared access signature, for example, to allow an app to retrieve a list of files in a file system or to download a file.

Use an account-level shared access signature to allow access to anything that a service-level shared access signature can allow, plus additional resources and abilities. For example, you can use an account-level shared access signature to allow the ability to create file systems.

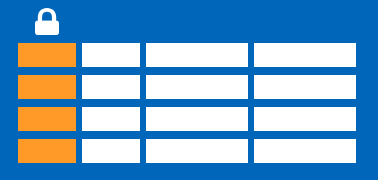
# **Manage authorization through column and row level security**

In this topic, we'll go through how you can manage authorization through column and row level security within Azure Synapse Analytics. We'll start off by talking about column level security in Azure Synapse Analytics, and finish with row level security.

## Column level security in Azure Synapse Analytics

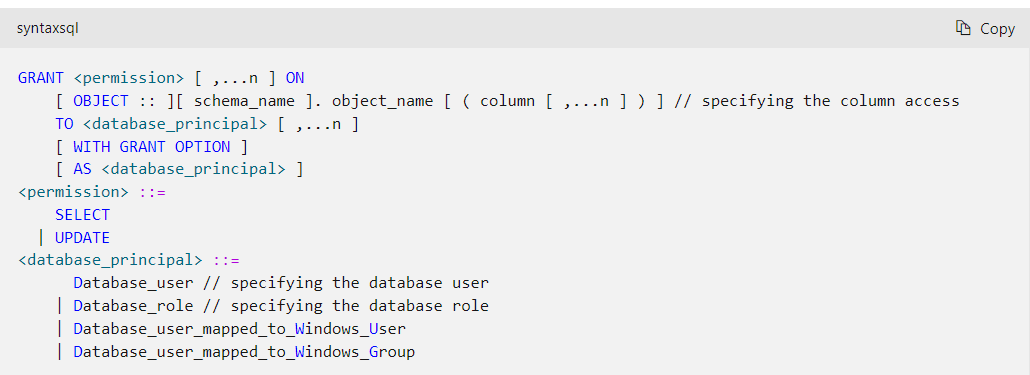
Generally speaking, column level security is simplifying a design and coding for the security in your application. It allows you to restrict column access in order to protect sensitive data. For example, if you want to ensure that a specific user 'Leo' can only access certain columns of a table because he's in a specific department. The logic for 'Leo' only to access the columns specified for the department he works in, is a logic that is located in the database tier, rather than on the application level data tier. If he needs to access data from any tier, the database should apply the access restriction every time he tries to access data from another tier. The reason for doing so is to make sure that your security is reliable and robust since we're reducing the surface area of the overall security system. Column level security will also eliminate the necessity for the introduction of view, where you would filter out columns, to impose access restrictions on 'Leo'

The way to implement column level security is by using the GRANT T-SQL statement. Using this statement, SQL and Azure Active Directory (AAD) support the authentication.



**Syntax**

The syntax to use for implementing column level security looks as follows:



So when would you use column-level security? Let's say that you are a financial services firm, and can only have an account manager allowed to have access to a customer's social security number, phone number, or other personally identifiable information. It is imperative to distinguish the role of an account manager versus the manager of the account managers.

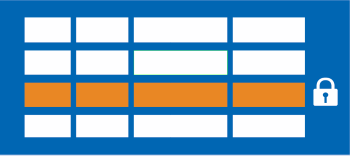
Another use case might be related to the Healthcare Industry. Let's say you have a specific health care provider. This healthcare provider only wants doctors and nurses to be able to access medical records. The billing department should not have access to view this data. Column-level security might be the option to use.

So how does column level security distinguishes from row-level security? Let's look into that.

## Row level security in Azure Synapse Analytics

Row-level security (RLS) can help you to create a group membership or execution context in order to control not just columns in a database table, but actually, the rows. RLS, just like column-level security, can simply help and enable your design and coding of your application security. However, compared to column-level security where it's focused on the columns (parameters), RLS helps you implement restrictions on data row access. Let's say that your employee can only access rows of data that are important to the department, you should implement RLS. If you want to restrict, for example, customer data access that is only relevant to the company, you can implement RLS. The restriction on the access of the rows is a logic that is located in the database tier, rather than on the application level data tier. If 'Leo' needs to access data from any tier, the database should apply the access restriction every time he tries to access data from another tier. The reason for doing so is to make sure that your security is reliable and robust since we're reducing the surface area of the overall security system.

The way to implement RLS is by using the CREATE SECURITY POLICY[!INCLUDEtsql] statement. The predicates are created as inline table-valued functions. It is imperative to understand that within Azure Synapse, only supports filter predicates. If you need to use a block predicate, you won't be able to find support at this moment within Azure synapse.



### Description of row level security in relation to filter predicates

RLS within Azure Synapse supports one type of security predicates, which are Filter predicates, not block predicates.  
What filter predicates do, is silently filtering the rows that are available for reading operations such as SELECT, UPDATE, DELETE.

The access to row-level data in a table is restricted as an inline table-valued function, which is a security predicate. This table-valued function will then be invoked and enforced by the security policy that you need. An application is not aware of rows that are filtered from the result set for filter predicates. So what will happen is that if all rows are filtered, a null set is returned.

When you are using filter predicates, it will be applied when data is read from the base table. The filter predicate affects all get operations such as SELECT, DELETE, UPDATE. You are unable to select or delete rows that have been filtered. It is not possible for you to update a row that has been filtered. What you can do, is update rows in a way that they will be filtered afterward.

### Use cases

We've already mentioned some use cases for RLS. Another use case might where you have created a multi-tenant application where you create a policy where logical separations of a tenant's data rows from another tenant's data rows are enforced. In order to implement this efficiently, it is highly recommended to store data for many tenants in a single table.

When we look at RLS filter predicates, they are functionally equivalent to appending a **WHERE** clause. The predicate can be as sophisticated as business practices dictate, or the clause can be as simple as WHERE TenantId = 42.

When we look at RLS more formally, RLS introduces predicate based access control. The reason why RLS can be used for predicate access control is that it is a flexible, centralized, predicate-based evaluation. The filter predicate can be based on metadata or any other criteria you would determine as appropriate. The predicate is used as a criterion to determine if the user has the appropriate access to the data based on user attributes. Label-based access control can be implemented by using predicate-based access control.

### Permissions

If you want to create, alter or drop the security policies, you would have to use the **ALTER ANY SECURITY POLICY** permission. The reason for that is when you are creating or dropping a security policy it requires **ALTER** permissions on the schema.

In addition to that, there are other permissions required for each predicate that you would add:

* **SELECT** and **REFERENCES** permissions on the inline table-valued function being used as a predicate.
* **REFERENCES** permission on the table that you target to be bound to the policy.
* **REFERENCES** permission on every column from the target table used as arguments.

Once you've set up the security policies, they will apply to all the users (including dbo users in the database) Even though DBO users can alter or drop security policies, their changes to the security policies can be audited. If you have special circumstances where highly privileged users, like a sysadmin or db\_owner, need to see all rows to troubleshoot or validate data, you would still have to write the security policy in order to allow that.

If you have created a security policy where SCHEMABINDING = OFF, in order to query the target table, the user must have the **SELECT** or **EXECUTE** permission on the predicate function. They also need permissions to any additional tables, views, or functions used within the predicate function. If a security policy is created with SCHEMABINDING = ON (the default), then these permission checks are bypassed when users query the target table.

## Best practices

There are some best practices to take in mind when you want to implement RLS. We recommended creating a separate schema for the RLS objects. RLS objects in this context would be the predicate functions, and security policies. Why is that a best practice? It helps to separate the permissions that are required on these special objects from the target tables. In addition to that, separation for different policies and predicate functions may be needed in multi-tenant-databases. However, it is not a standard for every case.

Another best practice to bear in mind is that the **ALTER ANY SECURITY POLICY** permission should only be intended for highly privileged users (such as a security policy manager). The security policy manager should not require **SELECT** permission on the tables they protect.

In order to avoid potential runtime errors, you should take into mind type conversions in predicate functions that you write. Also, you should try to avoid recursion in predicate functions. The reason for this is to avoid performance degradation. Even though the query optimizer will try to detect the direct recursions, there is no guarantee to find the indirect recursions. With indirect recursion, we mean where a second function calls the predicate function.

It would also be recommended to avoid the use of excessive table joins in predicate functions. This would maximize performance.

Generally speaking when it comes to the logic of predicates, you should try to avoid logic that depends on session-specific SET options. Even though this is highly unlikely to be used in practical applications, predicate functions whose logic depends on certain session-specific **SET** options can leak information if users are able to execute arbitrary queries. For example, a predicate function that implicitly converts a string to **datetime** could filter different rows based on the **SET DATEFORMAT** option for the current session.

# Exercise - Manage authorization through column and row level security

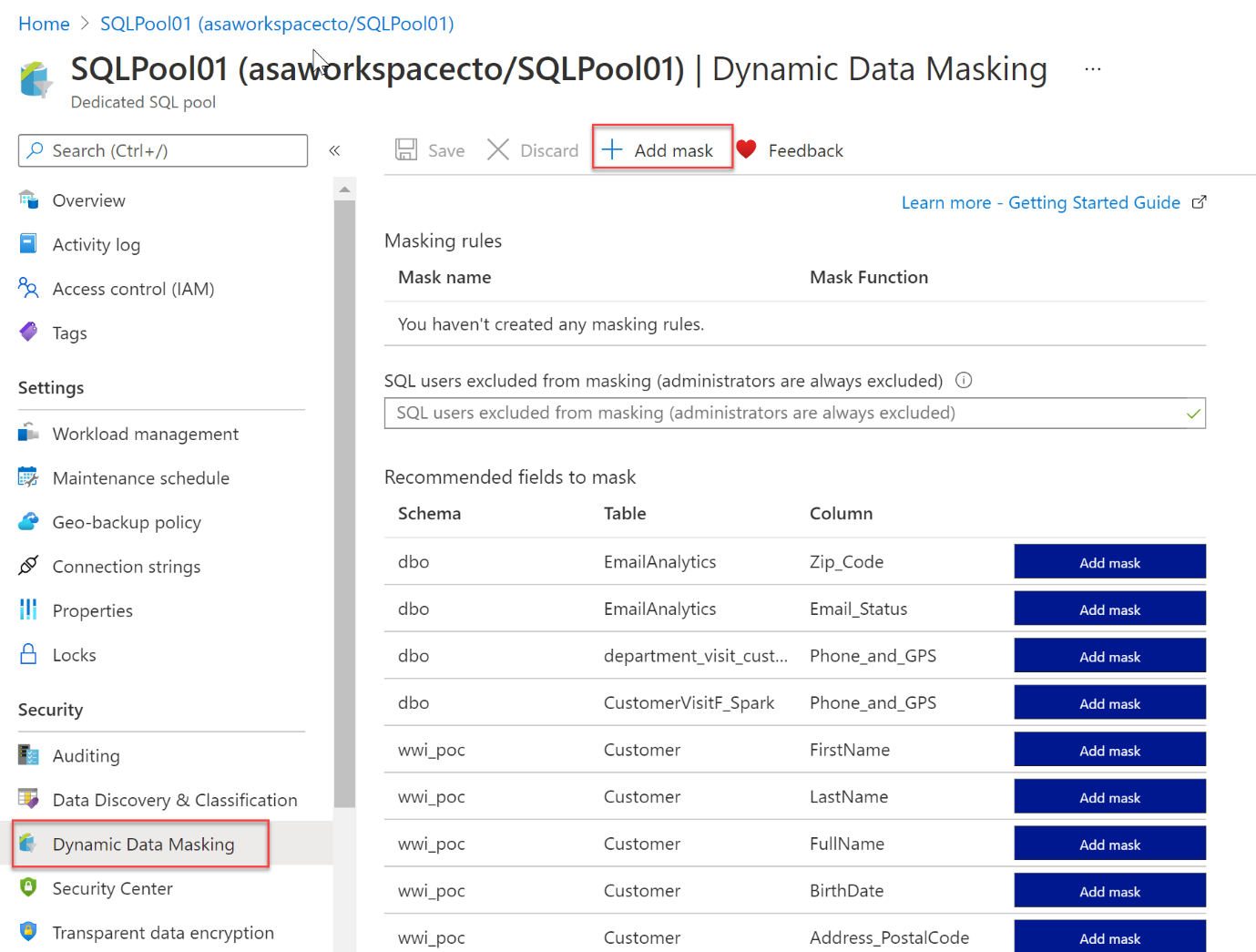
<https://learn.microsoft.com/en-us/training/modules/secure-data-warehouse-azure-synapse-analytics/6-exercise-manage-authorization-through-column-row-level-security>

# **Manage sensitive data with Dynamic Data Masking**

Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics support Dynamic Data Masking. Dynamic Data Masking ensures limited data exposure to non-privileged users, such that they cannot see the data that is being masked. It also helps you in preventing unauthorized access to sensitive information that has minimal impact on the application layer. Dynamic Data Masking is a policy-based security feature. It will hide the sensitive data in a result set of a query that runs over designated database fields.

Let's give you an example how it works. Let's say you work at a bank as a service representative in a call center. Due to compliance, any caller must identify themselves by providing several digits of their credit card number. In this scenario, the full credit card number should not be fully exposed to the service representative in the call center. You can define a masking rule, that masks all but the last four digits of a credit card number, so that you would get a query that only gives as a result the last four digits of the credit card number. This is just one example that could be equally applied to a variety of personal data such that compliance is not violated. For Azure Synapse Analytics, the way to set up a Dynamic Data Masking policy is using PowerShell or the REST API. The configuration of the Dynamic Data Masking policy can be done by the Azure SQL Database admin, server admin, or SQL Security Manager roles.

In Azure Synapse Analytics, you can find Dynamic Data Masking here;



**Looking into Dynamic Data Masking Policies**:

* **SQL users excluded from Dynamic Data Masking Policies**

The following SQL users or Azure AD identities can get unmasked data in the SQL query results. Users with administrator privileges are always excluded from masking, and will see the original data without any mask.

* **Masking rules** - Masking rules are a set of rules that define the designated fields to be masked including the masking function that is used. The designated fields can be defined using a database schema name, table name, and column name.
* **Masking functions** - Masking functions are a set of methods that control the exposure of data for different scenarios.

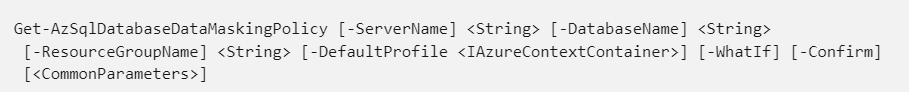
**Set up Dynamic Data Masking for your database in Azure Synapse Analytics using PowerShell cmdlets**

In this part, we are going to look into Dynamic Data Masking for a database in Azure Synapse Analytics using PowerShell cmdlets.

* Data masking policies
* Get-AzSqlDatabaseDataMaskingPolicy

The Get-AzSqlDatabaseDataMaskingPolicy gets the data masking policy for a database.

The syntax for the Get-AzSqlDatabaseDataMaskingPolicy in PowerShell is as follows:



What the **Get-AzSqlDatabaseDataMaskingPolicy** cmdlet does, is getting the data masking policy of an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

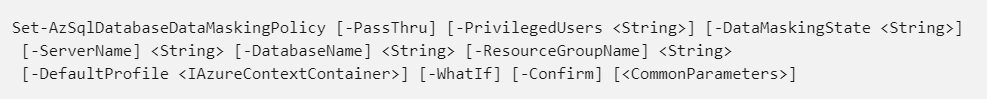
* *ResourceGroupName*: name of the resource group you deployed the database in
* *ServerName*: sql server name
* *DatabaseName* : name of the database

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

* Set-AzSqlDatabaseDataMaskingPolicy

The Set-AzSqlDatabaseDataMaskingPolicy sets data masking for a database.

The syntax for the Set-AzSqlDatabaseDataMaskingPolicy in PowerShell is as follows:



What the **Set-AzSqlDatabaseDataMaskingPolicy** cmdlet does is setting the data masking policy for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database

In addition, you will need to set the *DataMaskingState* parameter to specify whether data masking operations are enabled or disabled.

If the cmdlet succeeds and the *PassThru* parameter is used, it will return an object describing the current data masking policy in addition to the database identifiers.

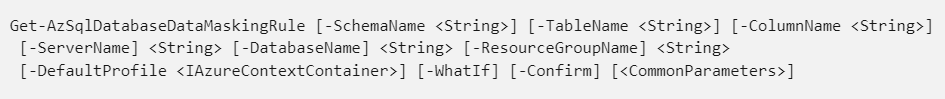
Database identifiers can include, **ResourceGroupName**, **ServerName**, and **DatabaseName**.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

* Data masking rules
* Get-AzSqlDatabaseDataMaskingRule

The Get-AzSqlDatabaseDataMaskingRule Gets the data masking rules from a database.

The syntax for the Get-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:



What the **Get-AzSqlDatabaseDataMaskingRule** cmdlet does it getting either a specific data masking rule or all of the data masking rules for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database

You'd also have to specify the *RuleId* parameter to specify which rule this cmdlet returns.

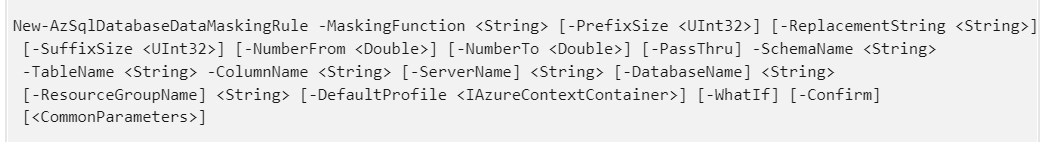
If you do not provide *RuleId*, all the data masking rules for that Azure SQL database are returned.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

* New-AzSqlDatabaseDataMaskingRule

The New-AzSqlDatabaseDataMaskingRule creates a data masking rule for a database.

The syntax for the New-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:



What the **New-AzSqlDatabaseDataMaskingRule** cmdlet does is creating a data masking rule for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule:

* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database

Providing the *TableName* and *ColumnName* is necessary in order to specify the target of the rule.

The *MaskingFunction* parameter is necessary to define how the data is masked.

If *MaskingFunction* has a value of Number or Text, you can specify the *NumberFrom* and *NumberTo* parameters, for number masking, or the *PrefixSize*, *ReplacementString*, and *SuffixSize* for text masking.

If the command succeeds and the *PassThru* parameter is used, the cmdlet returns an object describing the data masking rule properties in addition to the rule identifiers.

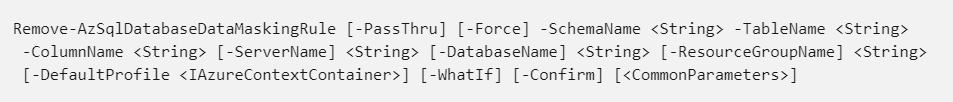
Rule identifiers can be, for example, *ResourceGroupName*, *ServerName*, *DatabaseName*, and *RuleID*.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

* Remove-AzSqlDatabaseDataMaskingRule

The Remove-AzSqlDatabaseDataMaskingRule removes a data masking rule from a database.

The syntax for the Remove-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:



What the **Remove-AzSqlDatabaseDataMaskingRule** cmdlet does, is it removes a specific data masking rule from an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule that needs to be removed:

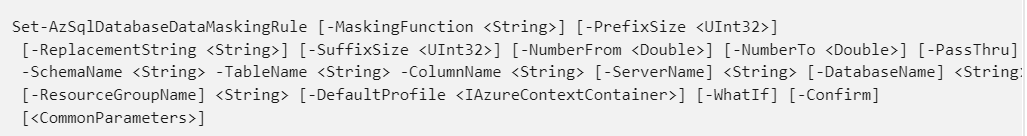
* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *RuleId* : identifier of the rule

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

* Set-AzSqlDatabaseDataMaskingRule

The Set-AzSqlDatabaseDataMaskingRule Sets the properties of a data masking rule for a database.

The syntax for the Set-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:



What the **Set-AzSqlDatabaseDataMaskingRule** cmdlet does is setting a data masking rule for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule:

* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *RuleId* : identifier of the rule

You can provide any of the parameters of *SchemaName*, *TableName*, and *ColumnName* to retarget the rule.

Specify the *MaskingFunction* parameter to modify how the data is masked.

If you specify a value of Number or Text for *MaskingFunction*, you can specify the *NumberFrom* and *NumberTo* parameters for number masking or the *PrefixSize*, *ReplacementString*, and *SuffixSize* parameters for text masking.

If the command succeeds, and if you specify the *PassThru* parameter, the cmdlet returns an object that describes the data masking rule properties and the rule identifiers.

Rule identifiers can be, **ResourceGroupName**, **ServerName**, **DatabaseName**, and **RuleId**.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

**Set up Dynamic Data Masking for your database in Azure Synapse Analytics using the REST API**

For setting up Dynamic Data Masking in Azure Synapse Analytics, you can also make use of the REST API. It will enable you to programmatically manage data masking policy and rules.

The REST API will support the following operations:

* Data masking policies
* Create Or Update

The Create Or Update masking policy using the REST API will create or update a database data masking policy.

In HTTP the following request can be made:

PUT [https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default?api-version=2014-04-01](https://management.azure.com/subscriptions/%7bsubscriptionId%7d/resourceGroups/%7bresourceGroupName%7d/providers/Microsoft.Sql/servers/%7bserverName%7d/databases/%7bdatabaseName%7d/dataMaskingPolicies/Default?api-version=2014-04-01)

The following parameters need to be passed through:

* *SubscriptionID*: the ID of the subscription
* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *dataMaskingPolicyName*: the name of the data masking policy
* *api version*: version of the api that is used.
* Get

The Get policy, gets a database data masking policy.

In HTTP the following request can be made:

GET [https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default?api-version=2014-04-01](https://management.azure.com/subscriptions/%7bsubscriptionId%7d/resourceGroups/%7bresourceGroupName%7d/providers/Microsoft.Sql/servers/%7bserverName%7d/databases/%7bdatabaseName%7d/dataMaskingPolicies/Default?api-version=2014-04-01)

The following parameters need to be passed through:

* *SubscriptionID*: the ID of the subscription
* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *dataMaskingPolicyName*: the name of the data masking policy
* *api version*: version of the api that is used.
* Data masking rules
* Create Or Update

The Create or Update masking rule creates or updates a database data masking rule.

In HTTP the following request can be made:

PUT [https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default/rules/{dataMaskingRuleName}?api-version=2014-04-01](https://management.azure.com/subscriptions/%7bsubscriptionId%7d/resourceGroups/%7bresourceGroupName%7d/providers/Microsoft.Sql/servers/%7bserverName%7d/databases/%7bdatabaseName%7d/dataMaskingPolicies/Default/rules/%7bdataMaskingRuleName%7d?api-version=2014-04-01)

The following parameters need to be passed through:

* *SubscriptionID*: the ID of the subscription
* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *dataMaskingPolicyName*: the name of the data masking policy
* *dataMaskingRuleName*: the name of the rule for data masking
* *api version*: version of the api that is used.
* List By Database

The List By Database request gets a list of database data masking rules.

In HTTP the following request can be made:

GET [https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default/rules?api-version=2014-04-01](https://management.azure.com/subscriptions/%7bsubscriptionId%7d/resourceGroups/%7bresourceGroupName%7d/providers/Microsoft.Sql/servers/%7bserverName%7d/databases/%7bdatabaseName%7d/dataMaskingPolicies/Default/rules?api-version=2014-04-01)

The following parameters need to be passed through:

* *SubscriptionID*: the ID of the subscription
* *ResourceGroupName*: name of the resource group that you deployed the database in
* *ServerName* : sql server name
* *DatabaseName* : name of the database
* *dataMaskingPolicyName*: the name of the data masking policy
* *api version*: version of the api that is used.

# **Implement encryption in Azure Synapse Analytics**

**What is transparent data encryption**

Transparent data encryption (TDE) is an encryption mechanism to help you protect Azure Synapse Analytics. It will protect Azure Synapse Analytics against threats of malicious offline activity. The way TDE will do so is by encrypting data at rest. TDE performs real-time encryption as well as decryption of the database, associated backups, and transaction log files at rest without you having to make changes to the application. In order to use TDE for Azure Synapse Analytics, you will have to manually enable it.

What TDE does is performing I/O encryption and decryption of data at the page level in real time. When a page is read into memory, it is decrypted. It is encrypted before writing it to disk. TDE encrypts the entire database storage using a symmetric key called a Database Encryption Key (DEK). When you start up a database, the encrypted Database Encryption Key is decrypted. The DEK will then be used for decryption and re-encryption of the database files in the SQL Server database engine. The DEK is protected by the Transparent Data Encryption Protector. This protector can be either a service-managed certificated, which is referred to as service-managed transparent data encryption, or an asymmetric key that is stored in Azure Key Vault (customer-managed transparent data encryption).

What is important to understand is that for Azure Synapse Analytics, this TDE protector is set on the server level. There it is inherited by all the databases that are attached or aligned to that server. The term server refers both to server and instance.

**Service-managed transparent data encryption**

As stated above, the DEK that is protected by the Transparent Encryption protector can be service-managed certificates which we call service-managed TDE. When you look in Azure, that default setting means that the DEK is protected by a built-in certificate unique for each server with encryption algorithm AES256. When a database is in a geo-replicated relationship then primary and the geo-secondary database are protected by the primary database's parent server key. If the databases are connected to the same server, they will also have the same built-in AES 256 certificate. As Microsoft we automatically rotate the certificates in compliance with the internal security policy. The root key is protected by a Microsoft internal secret store. Microsoft also seamlessly moves and manages the keys as needed for geo-replication and restores.

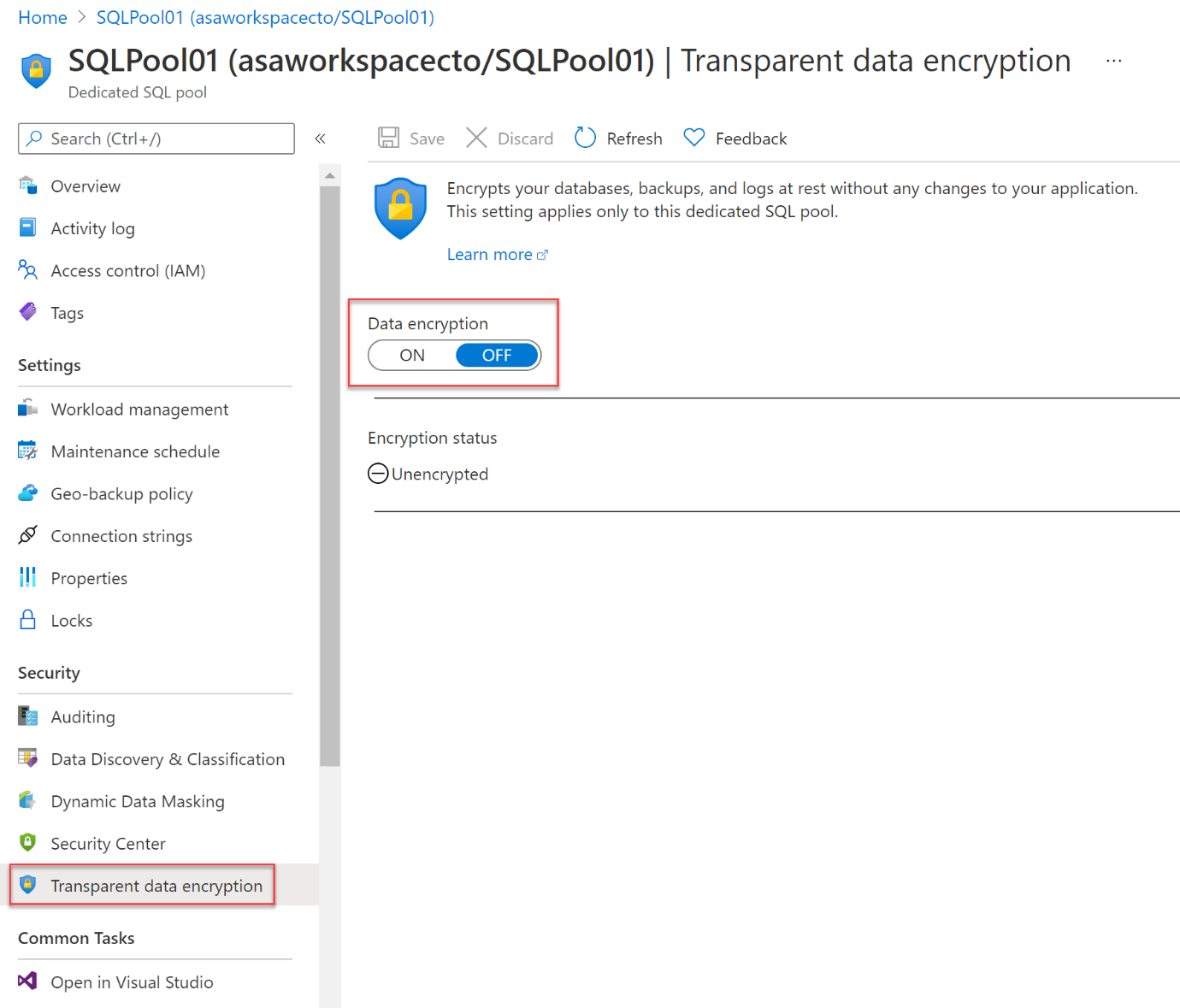
**Transparent data encryption with bring your own key for customer-managed transparent data encryption**

As stated above, the DEK that is protected by the Transparent Data Encryption Protector can also be customer managed by bringing an asymmetric key that is stored in Azure Key Vault (customer-managed transparent data encryption). This is also referred to as Bring Your Own Key (BYOK) support for TDE. When this is the scenario that is applicable to you, the TDE Protector that encrypts the DEK is a customer-managed asymmetric key. It is stored in your own and managed Azure Key Vault. Azure Key Vault is Azure's cloud-based external key management system. This managed key never leaves the key vault. The TDE Protector can be generated by the key vault. Another option is to transfer the TDE Protector to the key vault from, for example, an on-premise hardware security module (HSM) device. Azure Synapse Analytics needs to be granted permissions to the customer-owned key vault in order to decrypt and encrypt the DEK. If permissions of the server to the key vault are revoked, a database will be inaccessible, and all data is encrypted.

By using Azure Key Vault integration for TDE, you have control over the key management tasks such as key rotations, key backups, and key permissions. It also enables you to audit and report on all the TDE protectors when using the Azure Key Vault functionality. The reason for using Key Vault is that it provides you with a central key management system where tightly monitored HSMs are leveraged. It also enables you to separate duties of management of keys and data in order to meet compliance with security policies.

**Manage transparent data encryption in the Azure portal.**

For Azure Synapse Analytics, you can manage TDE for the database in the Azure portal after you've signed in with the Azure Administrator or Contributor account. The TDE settings can be found under your user database.



It is by default that the service-managed TDE is used and therefore a TDE certificate is automatically generated for the server that contains that database.

**Moving a transparent data encryption protected database**

In some use cases you need to move a database that is protected with TDE. Within Azure, there is no need to decrypt the databases. The TDE settings on the source database or primary database, will be inherited on the target. Some of the operations within Azure that inherited the TDE are:

* Geo-restore
* Self-service point-in-time restore
* Restoration of a deleted database
* Active geo-replication
* Creation of a database copy
* Restore of backup file to Azure SQL Managed Instance

If you export a TDE-protected database, the exported content is not encrypted. This will be stored in an unencrypted BACPAC file. You need to make sure that you protect this BACPAC file and enable TDE as soon as the import of the bacpac file in the new database is finished.

**Securing your credentials through linked services with TokenLibrary for Apache Spark**

It is quite a common pattern to access data from external sources. Unless the external data source allows anonymous access, it is highly likely that you need to secure your connection with a credential, secret, or connection string.

Within Azure Synapse Analytics, the integration process is simplified by providing linked services. Doing so, the connection details can be stored in the linked service or an Azure Key Vault. If the Linked Service is created, Apache spark can reference the linked service to apply the connection information in your code. When you want to access files from the Azure Data Lake Storage Gen 2 within your Azure Synapse Analytics Workspace, it uses AAD passthrough for the authentication. Therefore, there is no need to use TokenLibrary. However, to connect to other linked services, you are enabled to make a direct call to the TokenLibrary.

An example can be found below: In order to connect to other linked services, you are enabled to make a direct call to TokenLibrary by retrieving the connection string. In order to retrieve the connection string, use the **getConnectionString** function and pass in the **linked service name**.



If you want to Get the connection string as map and parse specific values from a key in the connection string, you can find an example below:

To parse specific values from a *key=value* pair in the connection string such as

*DefaultEndpointsProtocol=https;AccountName=<AccountName>;AccountKey=<AccountKey>*

use the **getConnectionStringAsMap** function and pass the key to return the value.

