

Active Directory Authentication - Detailed Architecture

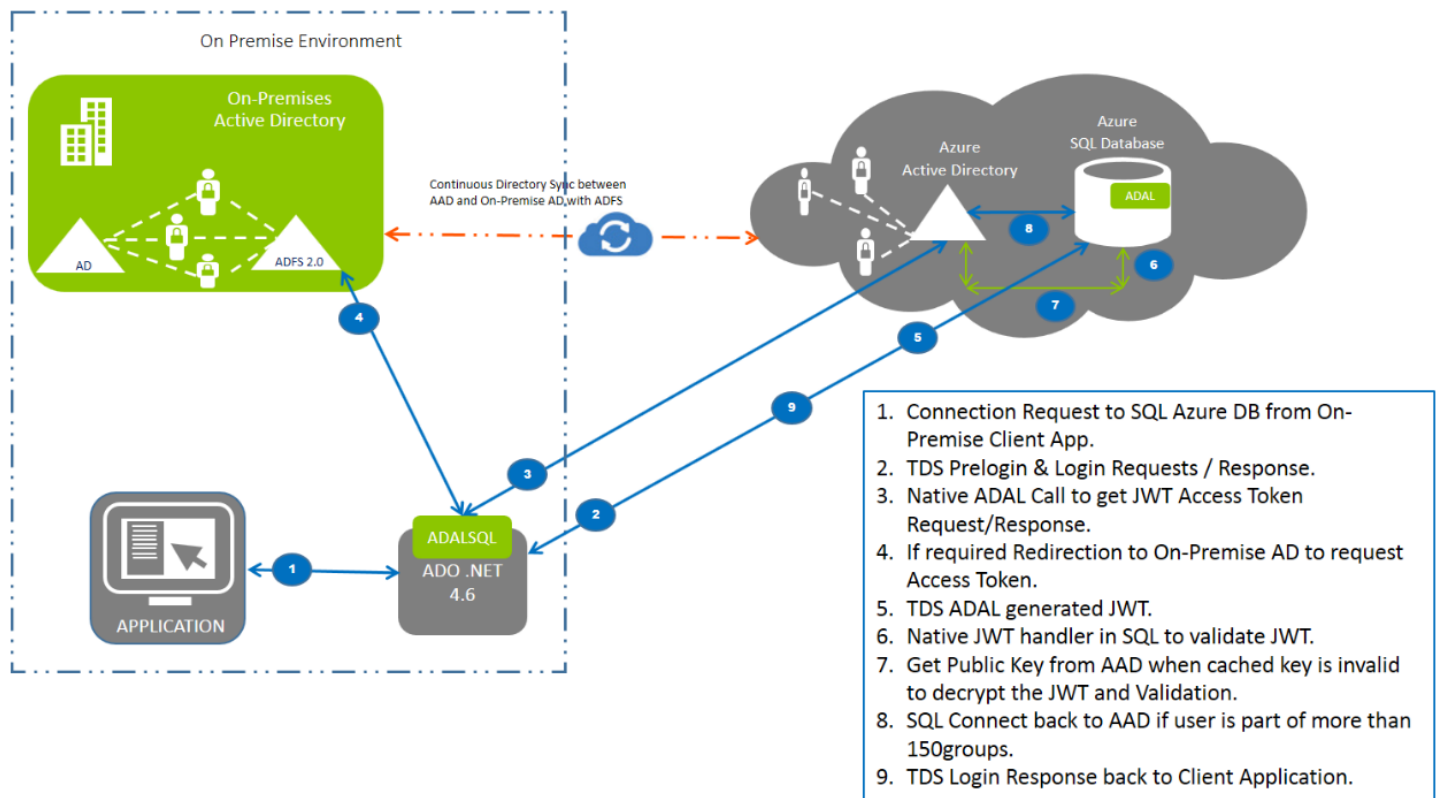
Last updated by | Subbu Kandhaswamy | Jun 28, 2021 at 10:42 AM PDT

Contents

- [AAD Architecture](#)
 - [Components](#)
- [ADAL - Active Directory Authentication Library](#)
 - [Detailed Authentication Scenario](#)
 - [Managed Account Authentication](#)
 - [SSMS Flow](#)
 - [Application Flow](#)
 - [Federated Account Integrated Authentication](#)
 - [SSMS Flow](#)
 - [Application Flow](#)
 - [Token Based ADAL Authentication](#)
 - [Authentication using JSON Web Token for Communication.](#)
 - [Flow](#)

AAD Architecture

Azure Active Directory Authentication for SQL Azure DB



Components

Components that got modified to get this feature enabled for SQL Azure DB

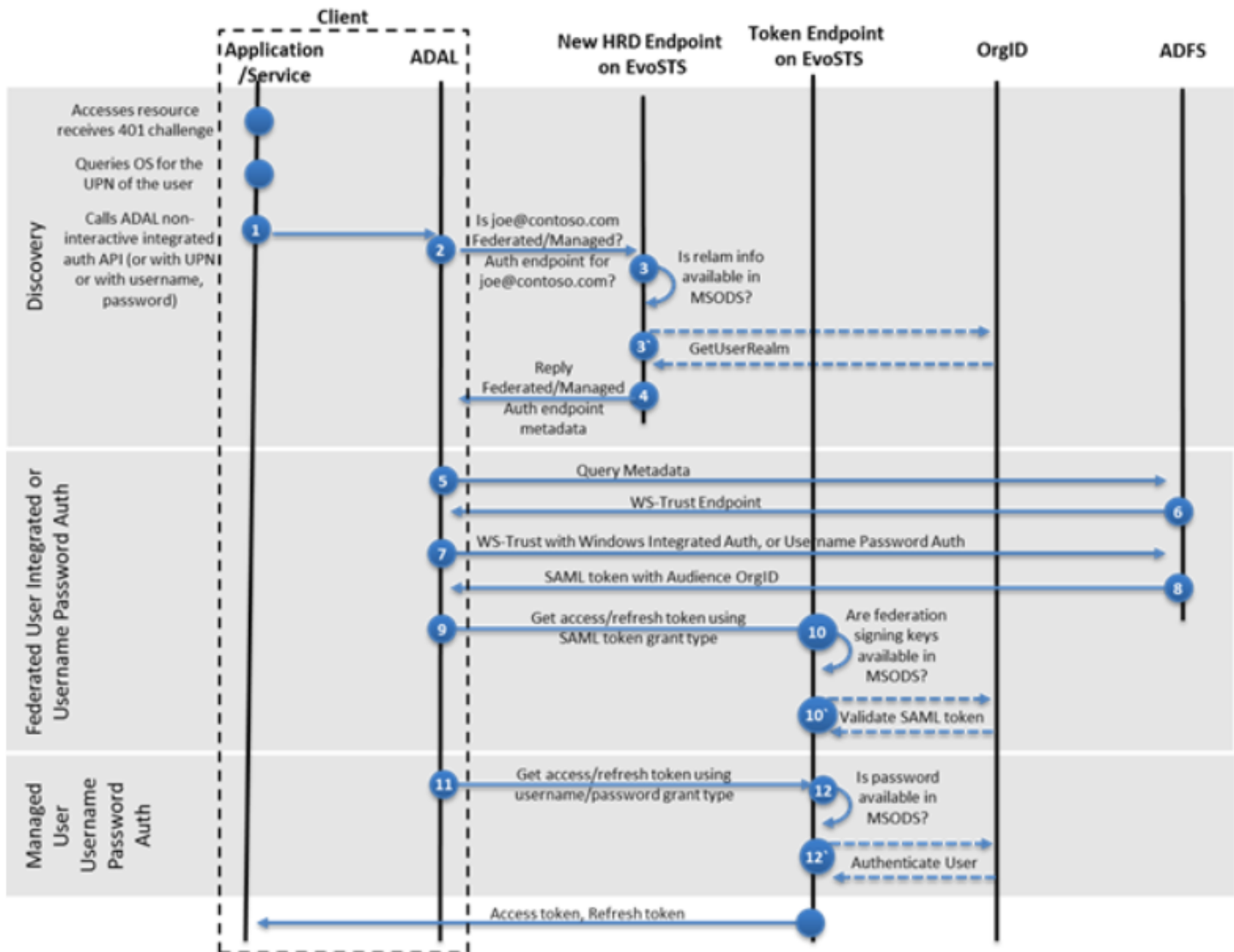
- Client Side : [ADO.NET](#)
- Server Side : Azure SQL DB

The client application has to utilize the new/modified [ADO.NET](#) API's if it needs to use AAD Authentication against SQL Azure DB. With regard to ADAL (Active Directory Authentication Library) AAD team has already released managed ADAL for .NET. However in SQL Azure DB we utilize Native ADAL Api's (C++ version). The Native ADAL implementation is not released for public and will not be provided for public consumption. It is only provided to internal partners, mainly SQL and Outlook/Office

ADAL - Active Directory Authentication Library

The ADAL implementation is embedded into a native C++ dll by name "ADAL.DLL", ADAL.DLL which has the actual native implementation is from Azure AD branch , This DLL is obtained and renamed as ADALSQL.DLL for consumption inside SQL client side product. ADALSQL.DLL gets installed via "ADALSQL.MSI". On the server side the ADAL.DLL is used as it is and gets loaded into SQLServr.exe address space when SQL Server starts.

ADAL workflow:



Detailed Authentication Scenario

Once the Azure Active Directory Authentication gets enabled for SQL Azure DB , We will be able to authenticate against SQL Azure DB using 4 options listed below.

- Connecting to Azure SQL DB using the Admin Login created via Portal or other SQL Server Contained Logins
- Connecting to Azure SQL DB using the managed accounts from Azure Active Directory (.OnMicrosoft.com logins)
- Connecting to Azure SQL DB using the Federated accounts from Azure Active Directory using Integrated Authentication (On Premise AD accounts)
- Connecting to Azure SQL DB using the Tokens provided by Applications which use ADAL (Active Directory Authentication Library) to authenticate to cloud or on premise AD

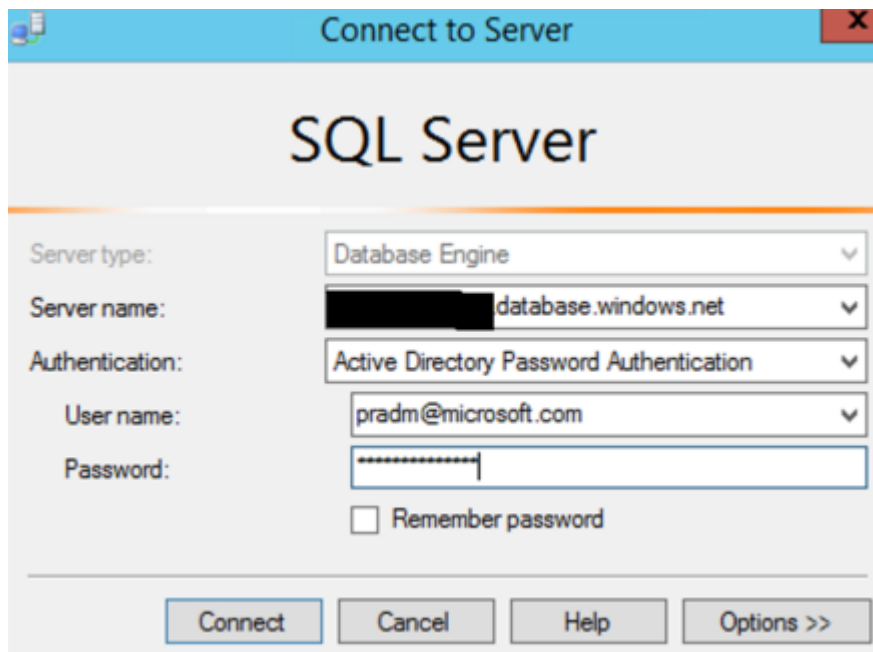
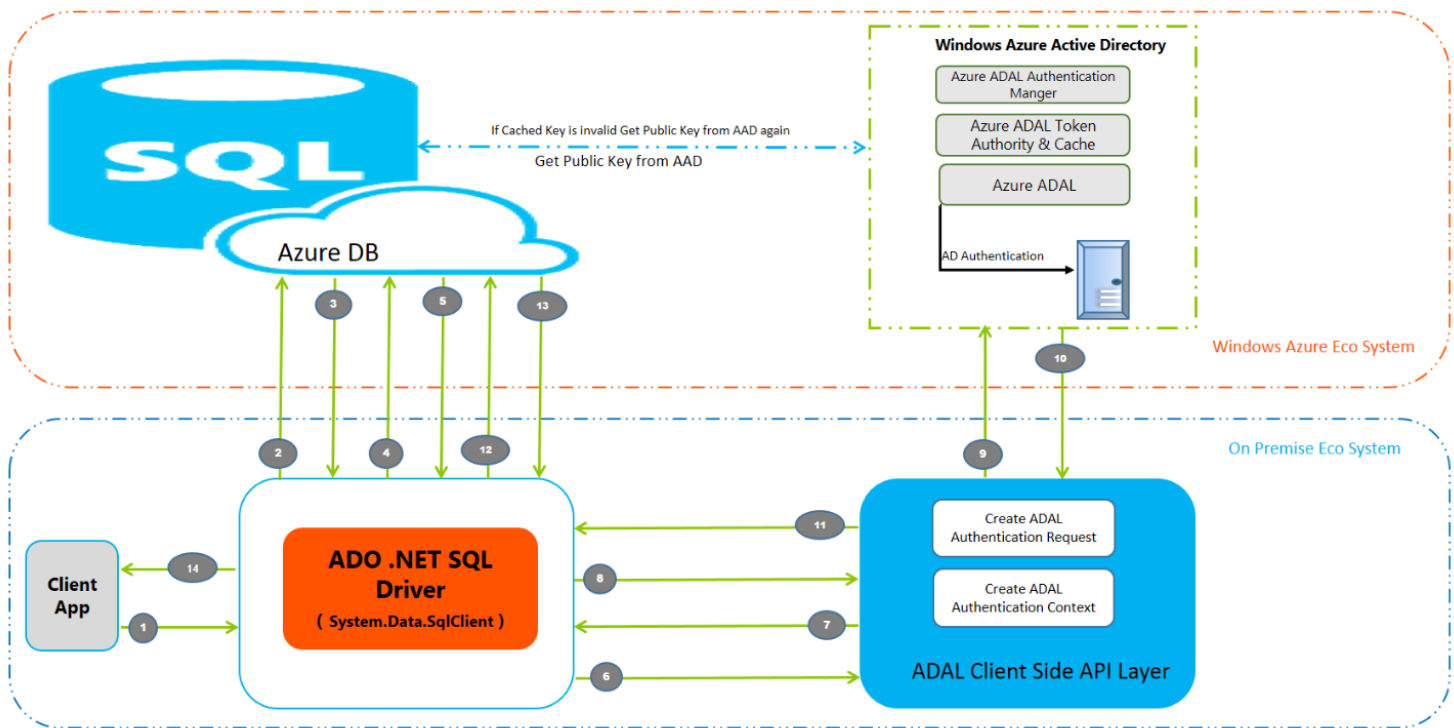
Authentication process flow for following scenarios

- Managed Account Authentication

- Federated Account Integrated Authentication
- Token based ADAL Authentication

Managed Account Authentication

SQL Azure DB Managed Account Authentication Work Flow



SSMS Flow

1. Open a version of SSMS that supports Azure AD Authentication and open the connect to Server Dialog -> Enter a Server Name -> Select "Active Directory Password Authentication" in the Authentication Box -> Enter the Azure AD account name and password as below (The account could be a managed account or federated account , To keep it simplified the above workflow is just for managed account. Federated Account process is merged with windows integrated authentication flow).

2. ADO .NET SQL Driver integrated with SSMS posts a TDS Pre-Login connection handshake request to Azure SQL DB (Remember the 7th step in the original SQL Authentication workflow).
3. Azure SQL DB responds back with TDS Pre-Login handshake response.
4. ADO .Net SQL Driver sends a TDS Login request (Indicating the intent to use ADAL against AAD).
5. Azure SQL DB responds & sends back a TDS Login response with an new TDS Token type called as FedAuthInfo Token (SPN Name, Token EndPoint (suffixed with customers TenentID) URL of an STS (AAD) e.g. <https://login.windows.net/> <GUID> from where client should get the token) and an SPN. The TenentID is the GUID here and is an Identifier for customers domain in Azure Active Directory.
6. ADO .Net SQL Driver uses ADAL client side API & prepare an ADAL Authentication context (User Name , password , STS)
7. ADAL client API provides ADAL Authentication Context to the ADO .Net SQL Driver.
8. ADO .Net SQL Driver uses ADAL client side API & prepare and ADAL Authentication request to get an Access Token from Azure Active Directory (AAD).
9. The request is sent to AAD for authentication.
10. AAD authenticates and provides the Access Token (JWT).
11. The Access Token is returned to ADO .Net SQL Driver.
12. ADO .Net SQL Driver sends the TDS Login request back to Azure SQL DB with the Access Token (FedAuth Token).
13. Azure SQL DB validates the Token , The JWT is encrypted hence Azure SQL DB has to decrypt it using a pre-cached Public key already obtained from Azure AD or if the pre-cached key is invalid it requests for a new public key and decrypts the JWT and validates the signature to confirm that the token is actually from Azure AD (Then it checks if there is a contained database users with in the target database which maps to database user's AAD account or one of the groups to which this user account belongs to and if the user has connect permissions on the database).
14. The connection gets established & authentication process is complete (assuming the user has the right permissions) (This is when a security token will be present inside SQL Server for the connected user)

The flow explained above remains same when an application which uses ADAL library to authenticate with password for a Managed / Federated account to Azure AD. The flow of how the request flow out of Azure AD for a federated account authentication is depicted below but the password is passed-on and the same user name and pwd is presented to the AD FS server for authentication instead of a Service Ticket.

Application Flow

1. ADO .Net calls the following LoadADALLibrary()
2. LoadADALLibrary calls the managed wrapper method ADALNativeWrapper.ADALInitialize()
3. ADALNativeWrapper.ADALInitialize calls the native method SNISecADALInitialize()
4. The SNISecADALInitialize native method reads the installation path of ADAL library from the registry , then load it and GetProcAddress for the ADAL APIs we are going to use.

In the connection string you can very well exclude the Authentication keyword , Instead use the SQLCredential property which provides more secure way of maintaining the user name and password information (Note: You will not be able to see the username and password when you take a dump of the process when SQLCredential property is getting used). Development team has confirmed that this is not a supported scenario as how would the SQL Azure DB know if the client intents to use ADAL as the Authentication keyword is removed in this scenario. We dont use such implementation for Azure SQL DB Implementation.

- The on premise domain controller validates the user's credentials
 - The DC generates Kerberos Ticket Granting ticket valid for X hrs, The default is 600min but is configurable by local IT admin policy
2. ADO .NET SQL Driver integrated with SSMS posts a TDS Pre-Login connection handshake request to Azure SQL DB (Remember the 7th step in the original SQL Authentication workflow).
 3. Azure SQL DB responds back with TDS Pre-Login handshake response.
 4. ADO .Net SQL Driver sends a TDS Login request (Indicating the intent to use ADAL against AAD).
 5. Azure SQL DB responds & sends back a TDS Login response with an new TDS Token type called as FedAuthInfo Token (SPN Name, Token EndPoint (suffixed with customers TenentID) URL of an STS (AAD) e.g. <https://login.windows.net/<GUID>> from where client should get the token) and an SPN. The TenentID is the GUID here and is an Identifier for customers domain in Azure Active Directory.
 6. ADO .Net SQL Driver uses ADAL client side API & prepare an ADAL Authentication context (User Name , STS)
 7. ADAL client API provides ADAL Authentication Context to the ADO .Net SQL Driver.
 8. ADO .Net SQL Driver uses ADAL client side API & prepare and ADAL Authentication request to get an Access Token from Azure Active Directory (AAD).
 9. The request is sent to AAD for authentication.
 - Windows Azure AD looks up the DNS name of the on premise AD FS Server that has been previously configured for customer's domain home-realm.
 - It then redirects the client PC to the required AD FS Server DNS name , The client automatically connects back to the ADFS Server.
 - The on premise AD FS server challenges the client PC by requesting authentication.
 - The client PC SSMS passes the previously-generated Kerberos ticket granting ticket to the on premise KDC (ticket granting server) , requesting a service ticket.
 - The KDC issues the client a service ticket.
 - The Client PC presents the service ticket to AD FS. AD FS validates the Kerberos ticket and generates a signed SAML token for windows Azure AD, The ADFS will only send the signed SAML token if the credentials are valid.
 - AD FS performs a redirect with the signed SAML claims as a form POST back to Windows Azure AD , attesting to the valid credentials. (The SAML claims doesn't contain the corporate credentials. The actual user credentials never leave on premise site).
 - The client presents back the SAML token to Windows Azure AD.
 10. AAD transforms the AD FS SAML token to its own signed identity claims (JSON Web Token or JWT) for the ADO .NET Driver request and provides the JWT backAccess Token.
 11. The Access Token is returned to ADO .Net SQL Driver.
 12. ADO .Net SQL Driver sends the TDS Login request back to Azure SQL DB with the Access Token (FedAuth Token) .

13. Azure SQL DB validates the Token , The JWT is encrypted hence Azure SQL DB has to decrypt it using a pre-cached Public key already obtained from Azure AD or if the pre-cached key is invalid it requests for a new public key and decrypts the JWT and validates the signature to confirm that the token is actually from Azure AD (Then it checks if there is a contained database users with in the target database which maps to database user's AAD account or one of the groups to which this user account belongs to and if the user has connect permissions on the database).
14. The connection gets established & authentication process is complete (assuming the user has the right permissions). (This is when a security token will be present inside SQL Server for the connected user).

Application Flow

Managed Account where customer has migrated his AD to Azure (@contoso.onmicrosoft.com) or ADFS (@microsoft.com)

```
string ConnectionString = @"Data Source=pradmeasterling.database.windows.net; Authentication=Active Directory Integrated"; SqlConnection conn = new SqlConnection(ConnectionString); conn.Open();
```

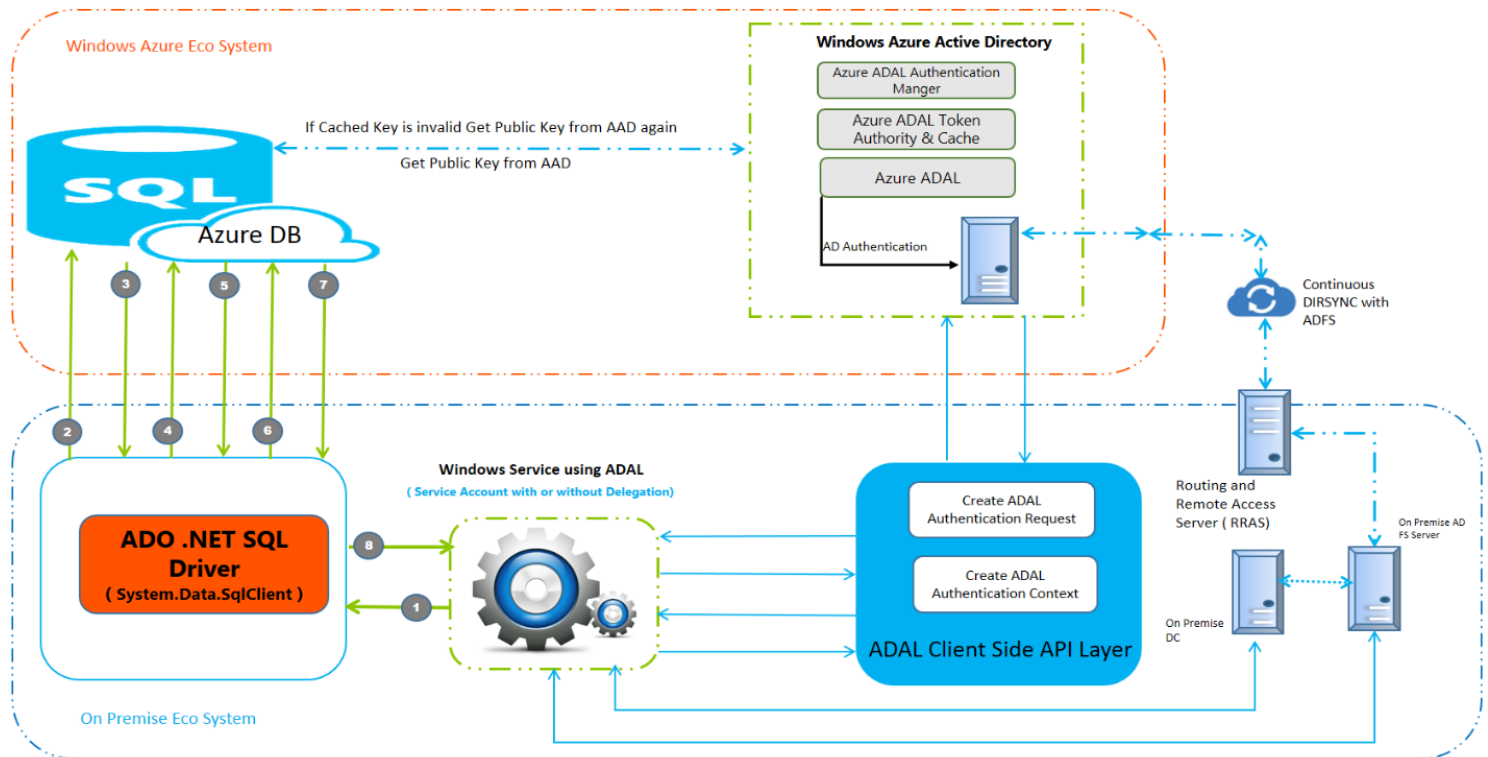
For AAD Integrated Authentication ADALUseWindowsAuthentication API is used, We load ADAL library (ADALSQL.DLL) when Connection.Open() is called and the authentication type is Active Directory Integrated Authentication. The ADAL library is unloaded when the process stops. We will ensure ADAL library is only initialized once and thread safe.

1. ADO .Net calls the following LoadADALLibrary()
2. LoadADALLibrary calls the managed wrapper method ADALNativeWrapper.ADALInitialize()
3. ADALNativeWrapper.ADALInitialize calls the native method SNISecADALInitialize()
4. The SNISecADALInitialize native method reads the installation path of ADAL library from the registry , then load it and GetProcAddress for the ADAL APIs we are going to use.

Token Based ADAL Authentication

Since now we understand how the Managed Account and Federated Account Authentication works in Azure AD , It's time to add some complex scenarios. From the specs this flow is called out as "Middle-Tier Service Authentication with or without Delegation" . In the above diagram more than following the numbering flow it's very important to understand the concept. This scenario cannot be showcased from management studio and is only applicable from the application layer which will be registered as a service.

SQL Azure DB Token based Authentication for Service Account Work Flow



**Concept of Middle-Tier Service Authentication without delegation:-**

1. I have an WEB application which is configured as a Windows Service (This can be a on a premise box or on an IaaS in Azure where we have configured a Azure Web role) . The start-up account of the service is registered in the Azure Active Directory (may be managed or federated).
2. When the service starts it authenticates to AAD with the service account using a certificate , effectively redeeming the certificate for an access token.
3. Now if the service wants to connect to SQL Azure DB it doesn't have to go through all the steps that I have showcased above for authentication since the token is already present for the service account, All it does is it presents the token directly and it will be able to connect to the SQL Azure DB and retrieve the required data based on the permissions assigned.

Concept of Middle-Tier Service Authentication with delegation:-

1. I have an WEB application which is configured as a Windows Service (This can be a on a premise box or on an IaaS in Azure where we have configured a Azure Web role, but the start-up account is a local service account) . The start-up account of the service is not registered in the Azure Active Directory.
2. When the service starts it authenticates to local box SAM.
3. Since this is an Web Application service , A user opens this application for a different box in a browser and performs a log-on to the web application using an AAD username/password.
4. The service authenticates the user with AAD and obtains an Access Token.
5. When the service need to connect to Azure SQL DB as a part of operation the end-user has called , The service posts a login request to the Azure SQL DB using the already issued token of the end user, effectively impersonating the end user and retrieves the required data based on the permission assigned to the end-user's user context in SQL Azure DB

So in both scenarios if you notice the access token is present with the connecting party well before it attempts to connect to SQL Azure DB. From SQL Azure DB perspective we don't really care how did he got the token all we are concerned about is authenticating the token when its presented to SQL Azure DB.

1. The Login request to SQL Azure DB from the service is sent to the [ADO.NET](#)  SQL Driver
2. The [ADO.NET](#)  SQL driver posts a TDS Pre-Login connection handshake request to Azure SQL DB (Remember the 7th step in the original SQL Authentication workflow).
3. Azure SQL DB responds back with TDS Pre-Login handshake response.
4. ADO .Net SQL Driver sends a TDS Login request (Indicating the intent to use ADAL against AAD).
5. Azure SQL DB responds with an acknowledgement.
6. ADO .Net SQL Driver sends the TDS Login request back to Azure SQL DB with the Access Token (This is not FedAuth Token , This token is directly embedded inside LOGIN7 structure. That how SQL understands not to send back anything and to use the token and move ahead with authentication).
7. Azure SQL DB validates the Token , The JWT is encrypted hence Azure SQL DB has to decrypt it using a pre-cached Public key already obtained from Azure AD or if the pre-chached key is invalid it requests for a new public key and decrypts the JWT and validates the signature to confirm that the token is actually from Azure AD (Then it checks if there is a contained database users with in the target database which maps to database user's AAD account or one of the groups to which this user account belongs to and if the user has connect permissions on the database).
8. The connection gets established & authentication process is complete (assuming the user has the right permissions) , (This is when a security token will be present inside SQL Server for the connected user)

Fetching Token - [Sample Code - click to download](#)

In the code look at the Token project in the AzureADAuthenticationExamples , where in we fetch the token from the TokenFactory and directly open the connection. (below is the fetch token part of the code)

```

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();

builder["Data Source"] = "y84lz2iiry.sqltest-eg1.mscds.com"; // replace with your server name

builder["Initial Catalog"] = "AADTest"; // replace with your server name

builder["Connect Timeout"] = 30;
string accessToken = TokenFactor

y.GetAccessToken();

if (accessToken == null)

{

Console.WriteLine("Fail to acuire the token to the database.");

}

using (SqlConnection connection = new SqlConnection(builder.ConnectionString))
{

try
{

connection.AccessToken = accessToken;
connection.Open();

}

}

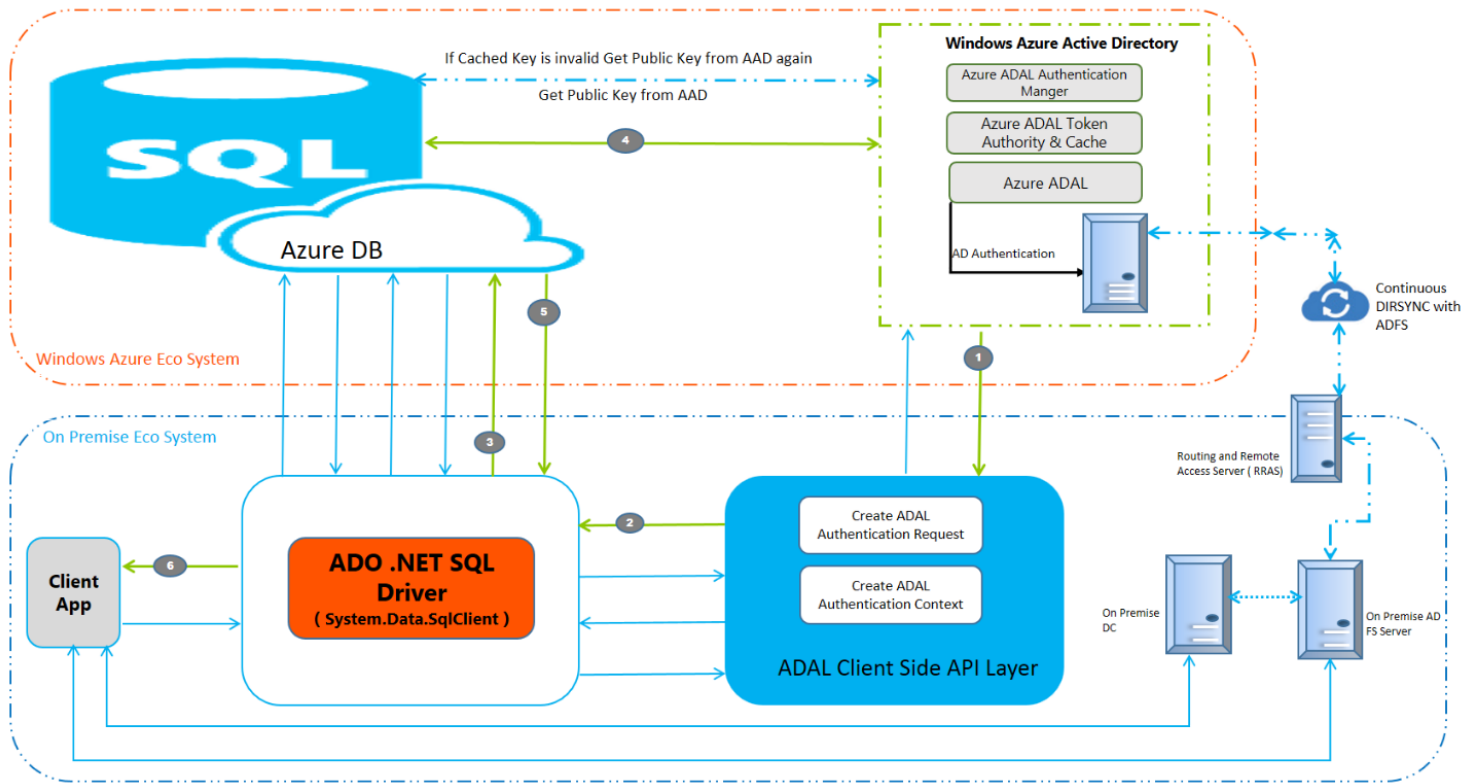
```

Authentication using JSON Web Token for Communication.

The JSON Web token sent by the client to SQL Azure DB might contain AAD Group membership information, Under scenarios where in a AAD Login is trying to connect to SQL is not present in SQL Azure DB rather a AAD Group is added in which the AAD Login is part of. So the JSON WEB Token of the AAD Login will contain AAD Group information which SQL Azure DB will use it for Authentication. In the JSON Web Token we have implemented a limitation of having only 150 groups (This limitation is not to avoid bloated token as TDS doesn't have any limitation of this sort, Rather the HTTP Headers which is used to represent data for all kind of authentications to Azure AD has this limitation and we do use HTTP Headers).

If the user is not part of more than 150 groups then we are good here , we use the ObjectID (ObjectID in Azure AAD World is equal to SID in the On-Prem World) of the AAD Login and the related groups ObjectID and then do the validation and complete the Authentication process. In case if the AAD Login is part of more than 150 groups then in the JSON WEB Token we provide an Indication Claim that there are more than 150 groups that the login belongs to. Under this situation the SQL Azure DB has to manually do the Group Expansion which means that it has to contact AAD again, To do this it internally uses the AAD Login Token itself and then send the request to AAD (SQL Azure DB doesn't use its start-up account rather it uses the AAD Login's token itself which it received from client, This is done to avoid people not to use escalation of privilege to query group information through SQL Azure DB which otherwise they might not have access to). SQL Azure DB uses AAD Graph API's to get the group related information impersonating the actual AAD Login who posted the authentication request as it has his Token now.

SQL Azure DB Impersonation & Communication with AAD (Group Expansion)



Flow

1. The AAD responds back with the JSON WEB Token (JWT) after authentication the AAD Login
2. The ADAL implementation used in the [ADO.NET](#) SQL Driver provides it back to [ADO.NET](#)
3. The [ADO.NET](#) SQL Driver provides the JSON WEB Token to SQL Azure DB for Authorization
4. SQL AzureDB validates the JSON Web Token (Finds that we have a Claim stating the AAD Login has more than 150 Groups)

- SQL AzureDB takes the AAD Login's (who posted the authentication request) JSON Token and presents it to AAD
- AAD provides the group information requested (Between AAD and On-Premise AD there is a continuous Directory)
- SQL Azure DB evaluates the ObjectIDs of all the Groups and verifies if any of the Group is part of SQL Azure

5. Either Login Success or Failure is returned back to [ADO.NET](#) SQL Driver based on the above analysis

6. The Same is returned back to the client and the authentication process is completed.

Azure SQL DB will talk to AAD when it receives a CREATE USER FROM EXTERNAL PROVIDER command , It impersonates the JWT token provided by the client who has already connected to Azure SQL DB and validates if the Login provided in the CREATE statement is valid.

How good have you found this content?

