PL-SQL and AZURE KEY VAULT

Table of Contents

1. Introduction

1.1 Overview:

Welcome to the documentation on integrating PL/SQL with Azure Key Vault for secure credential management. This guide is designed to assist developers and database administrators in leveraging Azure Key Vault to store and retrieve sensitive credentials within PL/SQL procedures.

In today's dynamic IT landscape, securing sensitive information is paramount. Azure Key Vault provides a robust solution for managing secrets, keys, and certificates, and integrating it with PL/SQL procedures enhances security measures within Oracle Database environments.

1.2 Audience

This documentation is intended for a diverse audience, encompassing developers, database administrators, and IT professionals involved in the management and security of Oracle Database environments. The following categories of individuals will find this guide beneficial:

i) **Developers**

PL/SQL Developers: If you are involved in writing PL/SQL procedures and wish to enhance the security of credential management, this documentation provides step-by-step instructions on integrating Azure Key Vault.

ii) **Database Administrators (DBAs)**

Oracle Database Administrators: DBAs responsible for securing and managing Oracle Database environments will gain insights into leveraging Azure Key Vault for centralized and secure credential storage.

iii) **IT Professionals**

IT Security Professionals: Individuals focused on implementing robust security practices within Oracle Database systems will discover methods for integrating Azure Key Vault to enhance credential security.

iv) **Technical Decision-Makers**

Architects and Technical Leaders: Those involved in making decisions regarding security architecture and procedures will find valuable information on the benefits and implementation of Azure Key Vault integration.

2. Prerequisite

2.1 Azure Service Principles

Enable Azure service principal authentication to allow Autonomous Database to access Azure services without providing long-term credentials.

To enable Azure service principal authentication on Autonomous Database:

a) Obtain your Microsoft Azure Active Directory tenant ID.
b) Enable Azure service principal with `DBMS_CLOUD_ADMIN.ENABLE_PRINCIPAL_AUTH`. For example:

```
BEGIN
    DBMS_CLOUD_ADMIN.ENABLE_PRINCIPAL_AUTH(
        provider => 'AZURE',
        username => 'adb_user',
        params   => JSON_OBJECT('azure_tenantid' value 'azure_tenantID'));
END;
```

This enables Azure service principal authentication and creates an Azure application on Autonomous Database.

If you want the specified user to have privileges to enable Azure service principal for other users, set the `params` parameter `grant_option` to `TRUE`.

For example:

```
BEGIN
DBMS_CLOUD_ADMIN.ENABLE_PRINCIPAL_AUTH(
provider => 'AZURE',
username => 'adb_user',
params   => JSON_OBJECT('grant_option' value TRUE,
                'azure_tenantid' value 'azure_tenantID'));
END;
```

After you run this command, `adb_user` can enable Azure service principal for another user. For example, if you connect as `adb_user`, you can run the following command:

For Example:

```
BEGIN
DBMS_CLOUD_ADMIN.ENABLE_PRINCIPAL_AUTH(
provider => ‘AZURE’, username => ‘adb_user2’)
END;
```

2.2 Azure Key Vault

From the Azure portal menu, or from the **Home** page, select **Create a resource**.

1. In the Search box, enter **Key Vault**.
2. From the results list, choose **Key Vault**.
3. On the Key Vault section, choose **Create**.
4. On the **Create key vault** section provide the following information:
    - **Name**: A unique name is required. For this quickstart, we use **Contoso-vault2**.
    - **Subscription**: Choose a subscription.
    - Under **Resource Group**, choose **Create new** and enter a resource group name.
    - In the **Location** pull-down menu, choose a location.
    - Leave the other options to their defaults.
5. Select **Create**.

Take note of these two properties:

- **Vault Name**: In the example, this is **Contoso-Vault2**. You'll use this name for other steps.
- **Vault URI**: In the example, the Vault URI is https://contoso-vault2.vault.azure.net/. Applications that use your vault through its REST API must use this URI.

Add a secret to Key Vault

To add a secret to the vault, follow the steps:

1. Navigate to your new key vault in the Azure portal
2. On the Key Vault settings pages, select **Secrets**.
3. Select on **Generate/Import**.
4. On the **Create a secret** screen choose the following values:
    - **Upload options**: Manual.
    - **Name**: Type a name for the secret. The secret name must be unique within a Key Vault. The name must be a 1–127-character string, starting with a letter and containing only 0-9, a-z, A-Z, and -. For more information on naming, see Key Vault objects, identifiers, and versioning
    - **Value**: Type a value for the secret. Key Vault APIs accept and return secret values as strings.
    - Leave the other values to their defaults. Select **Create**.

Once that you receive the message that the secret has been successfully created, you may select on it on the list.

2.3 Access Policies

In the Azure portal you must grant read access for a service principal to access the secret.

a. In the Azure portal, navigate to the "Key Vault" resource that contains the secret you created.
b. Select "Access policies", then select **Create**.
c. Under Permissions, select the **Get** permission in the **Secret permissions** section.
d. Under Principal, enter the name of the service principal in the search field and select the appropriate result.
e. Click **Next**.
f. Under Review + create, review the access policy changes, and click **Create** to save the access policy.
g. Back on the Access policies page, verify that your access policy is listed.

2.4 Azure AD application and necessary permissions

From the Azure portal menu, or from the **Home** page, click on "**Azure Active Directory**."

1. Browse to **Identity** > **Applications** > **App registrations** and select **New registration**.
2. Fill in the required information:
   Name: Give your application a meaningful name.
   Supported account types: Choose an appropriate option.
   Redirect URI: Optional, can be left blank for now. Click "**Register**."
3. Create Client Secret
   - In the Microsoft Entra admin center, in **App registrations**, select your application.
   - Select **Certificates & secrets** > **Client secrets** > **New client secret**.
   - Add a **description** for your client secret.
   - Select an **expiration** for the secret or specify a custom lifetime.
     i. Client secret lifetime is limited to two years (24 months) or less. You can't specify a custom lifetime longer than 24 months.
     ii. Microsoft recommends that you set an expiration value of less than 12 months.
   - Select **Add**.
4. Assign Permissions to Azure Key Vault

3. Creating Vault Secret Credentials and PLSQL Integration

3.1 Procedure Execution and Granting Privileges

This allows you to store a secret in Azure Key Vault and use the secret with the credentials you create to access cloud resources or to access other databases.

To create vault secret credentials where the secret is stored in Azure Key Vault: Create the Azure Key Vault, the secret, and the access policies to allow your Autonomous Database to access secrets in an Azure Key Vault.

1. Use DBMS_CLOUD.CREATE_CREDENTIAL to create a vault secret credential.

   For example:

```
BEGIN DBMS_CLOUD.CREATE_CREDENTIAL(

   credential_name  => 'AZURE_SECRET_CRED',

   params  => JSON_OBJECT(

     'username'         value 'azure_user',

     'secret_id'        value 'sales-secret',

     'azure_vault_name'  value 'azure_keyvault_name' ));

END;
```

   Where:

   - username: is the username of the original credential. It can be the username of any type of username/password credential.
   - secret_id: is the secret name.
   - azure_vault_name: is the name of the vault where the secret is located.

   To create a vault secret credential you must have EXECUTE privilege on the DBMS_CLOUD package.

2. Use the credential to access a cloud resource.

   For example:

```
SELECT count(*) FROM DBMS_CLOUD.LIST_OBJECTS(

    'AZURE_SECRET_CRED',

    'https://adb_user.blob.core.windows.net/adb/' );
```

3.2 Manual Refreshing

Every 12 hours the secret (password) is refreshed from the content in the Azure Key Vault. If you change the secret value in the Azure Key Vault, it can take up to 12 hours for the Autonomous Database instance to pick up the latest secret value.

Run `DBMS_CLOUD.REFRESH_VAULT_CREDENTIAL` to immediately refresh a vault secret credential. This procedure gets the latest version of the vault secret from Azure Key Vault

For Example:

```
BEGIN
DBMS_CLOUD.REFRESH_VAULT_CREDENTIAL(
credential_name => 'AZURE_SECRET_CRED');
END;
```

4. Creating Vault Secret Credentials and PLSQL Integration Using UTL_HTTP

You can access Azure Key Vault with PL/SQL procedures by using the Azure Key Vault REST API. The REST API allows you to perform operations on Azure Key Vault resources such as keys, secrets, and certificates. You can use the REST API to create, read, update, and delete these resources. Before implementing this solution, ensure creating an Azure AD application and assign it the necessary permissions to access the Azure Key Vault.

4.1 Authenticating with Azure AD

Create an HTTP request to the Azure AD token endpoint to authenticate the Azure AD application. Use the UTL_HTTP package to make HTTP requests in PL/SQL. The code snippet below demonstrates the authentication process:

For Example:

```
DECLARE

l_http_request UTL_HTTP.REQ;
l_http_response UTL_HTTP.RESP;
l_response_text VARCHAR2(32767);

BEGIN
      -- Create an HTTP request to the Azure AD token endpoint
l_http_request := UTL_HTTP.BEGIN_REQUEST('https://login.microsoftonline.com/<tenant-id>/oauth2/token', 'POST', 'HTTP/1.1');

UTL_HTTP.SET_HEADER(l_http_request, 'Content-Type', 'application/x-www-form-urlencoded');

      -- Set the request body to the client ID, client secret, and
      grant type
UTL_HTTP.WRITE_TEXT(l_http_request, 'client_id=<client-id>&client_secret=<client-secret>&grant_type=client_credentials');

      -- Get the HTTP response
l_http_response := UTL_HTTP.GET_RESPONSE(l_http_request);

      -- Read the response text
UTL_HTTP.READ_TEXT(l_http_response, l_response_text);

      -- Close the HTTP response
UTL_HTTP.END_RESPONSE(l_http_response);

END;
```

4.2 Get an Access Token

Parse the access token from the authentication response. The access token is required to authenticate subsequent requests to the Azure Key Vault. The following code snippet illustrates how to extract the access token:

For Example

```
DECLARE
    l_http_request UTL_HTTP.REQ;
    l_http_response UTL_HTTP.RESP;
    l_response_text VARCHAR2(32767);
    l_access_token VARCHAR2(4000); -- Variable to store the access token

BEGIN
    -- Your previous code for obtaining the access token here.

    -- Parse the access token from the response
      l_access_token := SUBSTR(
      l_response_text,
      INSTR(l_response_text, '"access_token"') + 15,
      INSTR(l_response_text, '"token_type"') - INSTR(l_response_text, '"access_token"') - 17 );

    -- Access token is now available for subsequent requests
END;
```

4.3 Access Azure Key Vault

Utilize the access token to make requests to the Azure Key Vault REST API. This involves creating an HTTP request with the access token in the Authorization header. The example below shows how to retrieve a secret from the Azure Key Vault:

For Example :

```
DECLARE
  l_http_request UTL_HTTP.REQ;
  l_http_response UTL_HTTP.RESP;
  l_response_text VARCHAR2(32767);
  l_access_token VARCHAR2(4000); -- Variable to store the access token

BEGIN

    --Your previous code for obtaining the access token here...

    --Create an HTTP request to the Azure Key Vault to retrieve a secret (replace <vault-name>
    and <secret-name> with your values)
  l_http_request := UTL_HTTP.BEGIN_REQUEST('https://<vault-name>.vault.azure.net/secrets/<secret-name>?api-version=7.0', 'GET', 'HTTP/1.1');

  UTL_HTTP.SET_HEADER(l_http_request, 'Authorization', 'Bearer ' || l_access_token);

  -- Get the HTTP response
  l_http_response := UTL_HTTP.GET_RESPONSE(l_http_request);

  --Read the response text
  UTL_HTTP.READ_TEXT(l_http_response, l_response_text);

  -- Process the response as needed...
   IF l_http_response.status_code = UTL_HTTP.HTTP_OK THEN
     l_response_text := REGEXP_SUBSTR(l_response_text, '"value": "([^"]+)"', 1, 1, NULL, 1);
     DBMS_OUTPUT.PUT_LINE('Secret Value: ' || l_response_text);


   -- Close the HTTP response
  UTL_HTTP.END_RESPONSE(l_http_response);

END;
```

5. Conclusion

This documentation has outlined the steps to seamlessly integrate Azure Key Vault with Oracle Autonomous Database through Azure Service Principal authentication. By following these steps, users can securely manage and utilize credentials stored in Azure Key Vault, enabling smooth access to Azure resources from the Oracle Autonomous Database instance.

The process involves enabling Azure Service Principal authentication, creating and configuring vault secret credentials, and utilizing PL/SQL queries to interact with Azure resources. Additionally, the documentation covers the manual refresh procedure to ensure the latest vault secret is always available.

By adopting these practices, organizations can enhance the security, efficiency, and manageability of credential management, promoting a robust and reliable connection between Oracle Autonomous Database and Azure Key Vault.