*\*\* To perform a hands-on with this and related assets you need to have access to Azure Key Vault Service / APP registrations to Azure \*\**

Table of Contents

[Introduction 1](#_Toc45382295)

[Azure Vault Overview 1](#_Toc45382296)

[Integration Pre-requisites 1](#_Toc45382297)

[Typical NFRs for using Key Vault-Secrets 2](#_Toc45382298)

[Example High level Design with MuleSoft 2](#_Toc45382299)

[Design considerations when using Key Vault to retrieve/use secrets 3](#_Toc45382300)

[Example designs 4](#_Toc45382301)

[**Vault secrets are retrieved on startup of Mule application instance** 4](#_Toc45382302)

[**Vault secrets are retrieved at the transaction time** 8](#_Toc45382303)

[How to create a vault and test Azure Key Vault REST API using Postman 11](#_Toc45382304)

[GIT Example Codes 11](#_Toc45382305)

[References 11](#_Toc45382306)

Introduction

With this article, below will be the key areas we are going to cover:

* Vault Overview and Key things required for Integration
* Designing MuleSoft projects with *secrets* kept in Azure Key Vault
* Including Use Cases / Specific Requirements as per our experience working with Azure Key Vault
* Decision factors/Considerations on using Vault within Integrations
* Laying out pre-requisites to Integrate w.r.t. Azure Key Vault
* Retrieve secrets using MuleSoft (Various Methods and Scenarios)
* How to create a vault and test Azure Key Vault REST API using Postman

Azure Vault Overview

<https://docs.microsoft.com/en-us/azure/key-vault/general/overview>

Key Benefits:

* Storage of Secrets, Keys, Certificates
* Records access of the vault
* No dependency on single person to maintain the credentials
* Store sensitive information
* Control of access/Permissions
* Version Management

Integration Pre-requisites

* Azure Tenant ID: A tenant ID is a unique way to identify an Azure AD instance within an Azure subscription.
* Vault Name: Key Vault name (e.g. folder which consists of all secrets, keys, certificates)
* Client ID: Client ID of the application with which you would authenticate to Azure AD
* Client Secret: Client Secret of the application with which you would authenticate to Azure AD
* OAuth endpoint: This is the endpoint which you will use to authenticate to Azure AD and retrieve OAuth token
* Vault DNS name: This is the complete DNS name of the vault to be used as an end point for specific Key Vault
* Secret Key Name/s

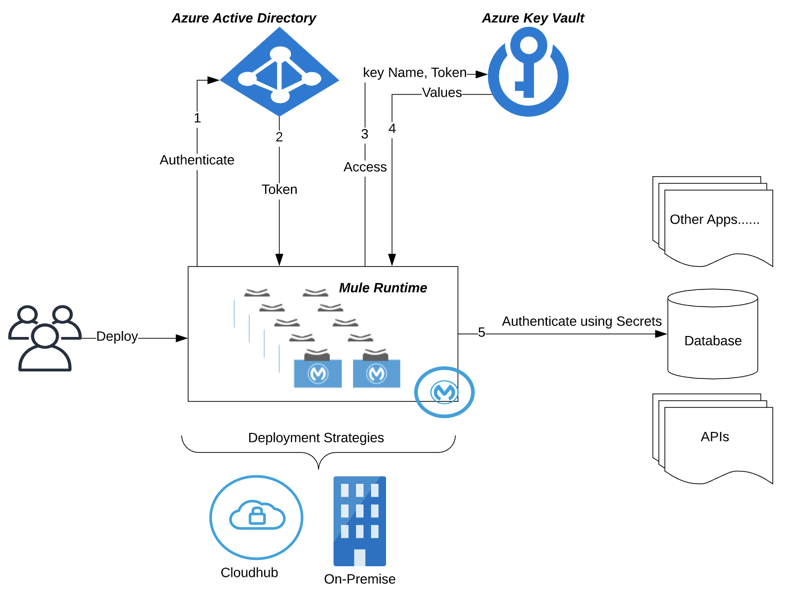
Typical NFRs for using Key Vault-Secrets

These are some of the typical reasons why organizations prefer to use a Vault to keep secrets *(credentials of application systems etc.):*

* Ease in maintaining system Credentials (operations)
* Storing all credentials in one place for various other Internal / External APIs
* User IDs and Passwords for various Application Systems (Legacy, Internal, External, Cloud etc.)
* Easy operational lights on changes to Integrations due to credentials being updated
  + *Note that in Cloudhub, there is zero downtime on such changes, but change to the credentials need to be safely coordinated so that no transactions are impacted*
* Easy migration of Integrations from one environment to other(*e.g. just name of the key vault differs*)
* No need to configure environment level credentials inside Properties/Yaml files.
  + *Secret values may not be shared with developers.*

Example High level Design with MuleSoft

*A typical design of key Vault integration with MuleSoft is illustrated below:*

**

Design considerations when using Key Vault to retrieve/use secrets

* Retrieval of credentials at transaction time OR during startup
  + *depending upon the use case*
* Token management within MuleSoft to interact with Microsoft Azure
  + Time-to-live (TTL) and refresh of token
* SLA’s for the transactions
* Requirement of connectivity validation to application systems during startup of runtime instance.
* Acceptable number of hits to Azure vault

Example designs

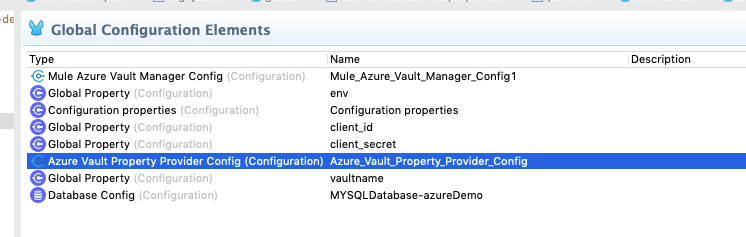
**Vault secrets are retrieved on startup of Mule application instance**

**Why this method?**

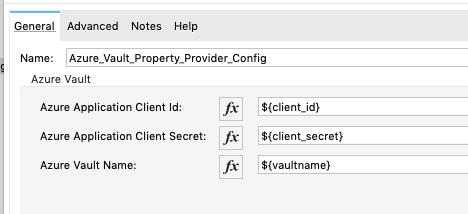
* Secrets retrieved from vault are used to connect to the application databases/external systems on the start of Mule application
* This provides validation of connectivity upfront. i.e. *Before the start of the transactions we need to ensure the connection is working fine as expected*
* This provides validation on accurate secret keys from vault being set up inside the application and also ensuring they are accessible using the provided Azure Client ID and Secret.
* This method prevents hitting Azure key vault multiple times, ideally once per application startup to retrieve secret key values required for the application.
* This method should be selected only when customer agrees that any changes to the vault values, a restart to the application is OKAY.
  + *If you are in cloud Mule environment, then note that there is no downtime for a restart of an application, hence this method could be a go-to solution with Azure Key Vault.*

**How is this typically achieved?**

* Simple solution on this requirement is using custom configuration properties placeholder: <https://docs.mulesoft.com/mule-runtime/4.3/custom-configuration-properties-provider>
  + - GitHub Link: (<https://github.com/mulesoft-catalyst/azurekeyvault-secrets-demo-usages/tree/master/src/test/resources/azure-connector-properties%20-%20no-encryption> )
* Input requirements for this connector:
  + Text Value for Azure Client ID
  + Text Value for Azure Client Secret
  + Text Value for Azure Key Vault Name
* In case you have more than one key vault to be utilized, then you can create more than one Custom property placeholder global configurations within the application as shown below:



* Kindly note that custom property placeholders are loaded in memory before Secure Properties placeholder, hence you may not be able to encrypt the Azure Client ID and Secret by default.
  + If you have encryption requirements, then you would need to customize the Properties placeholder code itself. See the next section for more details
* If there is no encryption requirement on the Azure Client ID/Secret then you may use this connector as is (*you may need to update the Microsoft Azure libraries as needed* with the latest ones) BUT ensure the Azure Client ID and secret along with the Key Vault is passed as JVM parameters (*e.g. mvn clean package …. -Dclient\_id=… -Dclient\_secret=… -Dvaultname=….).* *You cannot have these properties within the local environment level properties files as the local environment properties are loaded in runtime memory after custom properties placeholders.*



**Best ways to encrypt/decrypt Azure Client ID and Secret?**

* The easiest way to is to use Secure Properties tools JAR within the Custom Properties code itself (*via Maven POM dependency by adding the JAR as Custom Jar within the customer's exchange*) and then using ***applyOverString*** method to decrypt  
  *\*\* Only caveat to this would be, the secure properties tools jar will be part of the Azure custom properties code and any changes to the decryption/encryption, the connector code has to be updated \*\**

*Below method shows how to decrypt an encrypted String using Secure tools jar using Java.*

**public** **static** String decrypt(String encryptString, String decodedKey) {

String decrryptedString=SecurePropertiesTool.*applyOverString*("decrypt", "AES", "CBC", decodedKey, **true**, encryptString);

**return** decrryptedString;

}

*Important Note: If you are adding the Secure tools jar within your private exchange, then for the Azure Connector POM you need to ensure the dependency is added like below*

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>secure-properties-tools</artifactId>

<version>1.0.0</version>

<classifier>custom</classifier>

</dependency>

*And note that to retrieve this dependency from Exchange you need the below repository added into the Azure Connector POM too.*

<repository>

<id>anypoint-exchange-v2</id>

<name>Anypoint Exchange</name>

<url>https://maven.anypoint.mulesoft.com/api/v2/maven</url>

<layout>default</layout>

</repository>

*Once you add the above repository, you need to ensure the settings xml for maven is updated with the userid and pwd for accessing anypoint exchange v2 services.*

* If the above method is not suitable for you, then you can write a custom Java method in order to perform the decryption of Azure Client ID and Secret using an agreed method. There are plenty of example Java codes available online. *As per our experience AES/CBC with Random IV with 256 bit key could be one of the better standard methods on encryption which customers prefer.*

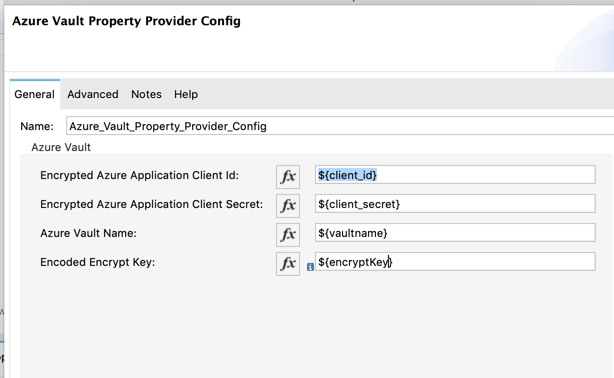
*Important note: If you are using “Random IV” method then you need to ensure the process used to encrypt the values , the same process has to be used for decryption.*

*If you are using a static IV, then the same static IV has to be made available at the decryption process.*

* Best practice would be to make encrypt key as part of the connector itself, so that developers understand the input requirement to work with the Azure connector.

* Please note, you cannot encrypt the encrypt key, but you could either have it Base64 encoded OR clear TEXT (*ensure that you mask it using mule-artifacts.json*). Decoding the encryptKey will have to be done within the Custom properties provider code like shown below:

At the Azure connector configuration within the studio:



Mule XML:

<azure-vault-property-provider:config name=*"Azure\_Vault\_Property\_Provider\_Config"* doc:name=*"Azure Vault Property Provider Config"* doc:id=*"189036b8-565f-4fc7-924c-783179879388"* >

<azure-vault-property-provider:azure-vault **applicationClientId**=*"${client\_id}"* **applicationSecretKey**=*"${client\_secret}"* **azureVaultName**=*"${vaultname}"* **encryptKey**=*"${encryptKey}"* />

</azure-vault-property-provider:config>

Within the Azure connector:

**byte**[] secretkey\_decodedBytes = java.util.Base64.*getDecoder*().decode(parameters1.getStringParameter("encryptKey"));

decodedKey = **new** String(secretkey\_decodedBytes, StandardCharsets.***UTF\_8***);

The asset for Azure Custom Properties provider with Encryption of Client ID and Secret using an Base64 encoded encryptKey is available at :

<https://github.com/mulesoft-catalyst/azure-vault-custom-properties-provider-encrypted>

**Vault secrets are retrieved at the transaction time**

**Why this method?**

* Sometimes secret values do change more frequently, hence this method could be utilized in those cases
* If the connectivity details are kept in Azure vault like userid/pwd for database/application system, then at the time of instance startup you shall not be able to validate the connectivity, unless you write a connectivity test within the code during which the secrets could be loaded in cache OR keep a scheduler to run once per startup *(e.g.*[*https://help.mulesoft.com/s/article/How-to-run-a-flow-once-in-when-an-application-starts-in-Mule4*](https://help.mulesoft.com/s/article/How-to-run-a-flow-once-in-when-an-application-starts-in-Mule4)*)*

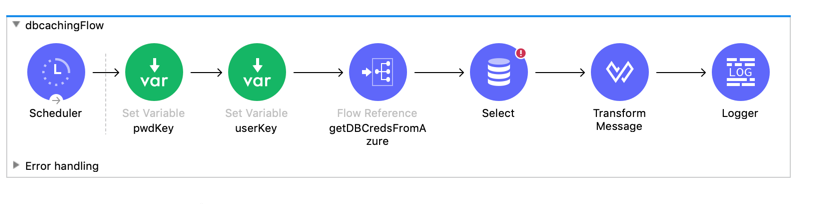
*\*\*\* Please note that there would be more number of hits to Azure Vault and slightly higher response SLA time for the APIs on an average when you chose this method. Additionally, there will be an added dependency on the Azure Vault service being operational during transaction time. \*\*\**

*We have not seen customers choosing this method more often. One of the reason is, if there are any changes to the secrets they cannot be ideally timed at the same time cache is refreshed, hence, there might be still a need to restart the integration.*

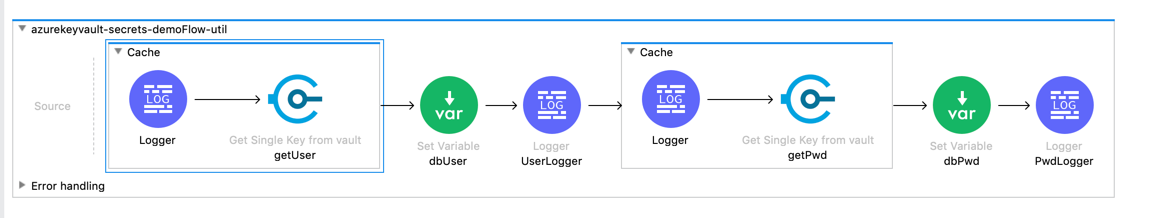
*Hence, instead of building the logic of cache and TTL and maintenance may get complex.*

**How is this typically achieved?**

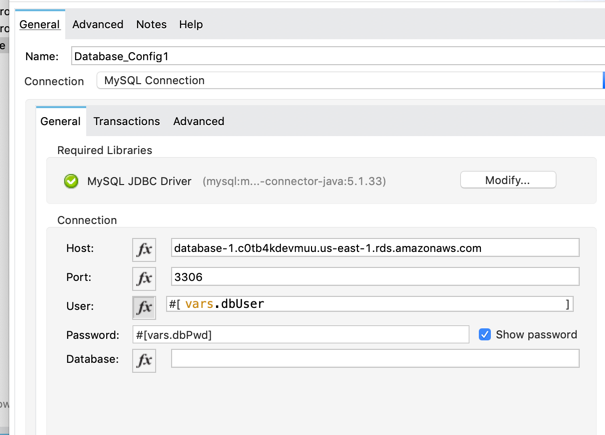
* On a scheduler(*which runs once during the startup*) , setting a caching strategy with the secret keys like user id and pwd key names (*keeping them into variables*). Below are some example snapshots, on how a transactional flow would look like.



Below is the flow, where the values are cached for userid and pwd and they are set into another set of variables(dbUser, dbPwd).

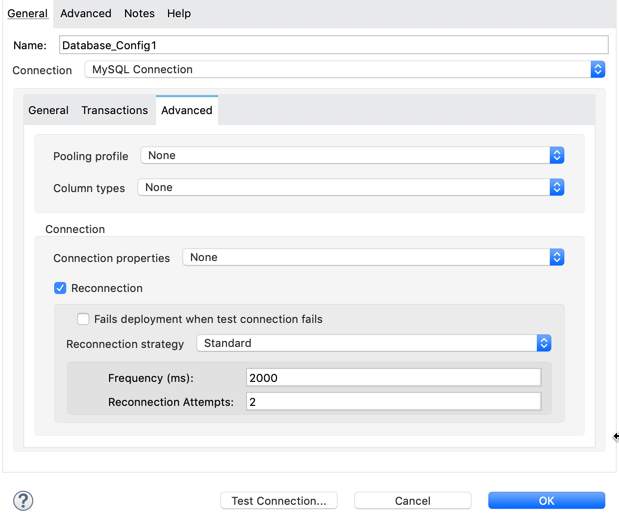


Connections to the systems (like in this case database) will have reference to these variables like shown below:

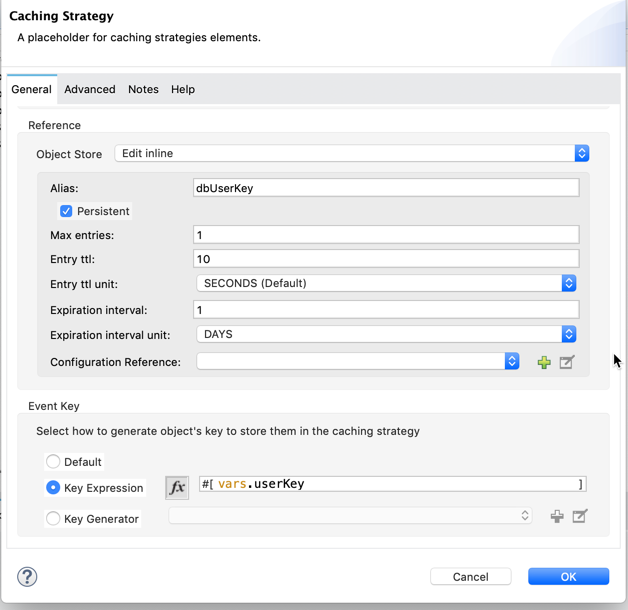


**Important Note:**

* For this method, the connections should have the below flag “unchecked” – *“Fails deployment when test connection fails”*



* Caching strategy in the above example is as below



*This means the userid will be available in the runtime cache for max 1 day after which the first transaction after 24 hrs, will call Azure Key Vault again and retrieve/set the key/values. Please note that we aren’t doing the TTL management on Azure Token or Azure Connection (that’s not the requirement here).*

\*\*\*\* Please note that , to retrieve values from Azure Key vault I am using another custom Asset (<https://github.com/mulesoft-catalyst/azurekeyvault-secrets-demo-usages/tree/master/src/test/resources/azure-connector-onDemand-%20no-encryption> ), ***but you can use Azure REST API as shown in picture below***\*\*\*\*

* Example caching based Asset is available at : <https://github.com/mulesoft-catalyst/azure-secrets-caching-db-connection>

How to create a vault and test Azure Key Vault REST API using Postman

Below are the detailed steps on this   
<https://github.com/mulesoft-catalyst/azurekeyvault-secrets-demo-usages/blob/master/Azure%20Key%20Vault-Creation%20and%20Postman%20REST%20API%20Examples.docx>

GIT Example Codes

Custom property provider for Azure with Encryption / Encoded Encrypt Key: <https://github.com/mulesoft-catalyst/azure-vault-custom-properties-provider-encrypted>

REST API usage of Azure Key Vault in Mule / How to use custom properties for DB connection: <https://github.com/mulesoft-catalyst/azurekeyvault-secrets-demo-usages>

Caching based Asset is available at : <https://github.com/mulesoft-catalyst/azure-secrets-caching-db-connection>

References

* API References;<https://docs.microsoft.com/en-us/rest/api/keyvault/getsecrets/getsecrets>
* Best Practices on Key Vault :<https://docs.microsoft.com/en-us/azure/key-vault/key-vault-best-practices>
* Key vault understanding of Error Codes:<https://docs.microsoft.com/en-us/azure/key-vault/rest-error-codes>
* Java APIs/Samples on Azure Key Vault:<https://docs.microsoft.com/en-us/java/api/overview/azure/keyvault?view=azure-java-stable>
* Complete Developers Guide on Key Vault: <https://docs.microsoft.com/en-us/azure/key-vault/key-vault-developers-guide>
* Key Vault API Reference :<https://docs.microsoft.com/en-us/rest/api/keyvault/>
* Get Secret REST API Reference :<https://docs.microsoft.com/en-us/rest/api/keyvault/getsecret>
* Get Secrets REST API Reference :<https://docs.microsoft.com/en-us/rest/api/keyvault/getsecrets>
* OBD Asset on Azure Key Vault Connector  [Azure Vault Properties Override and Reader for Mule 4](https://wiki.corp.mulesoft.com/display/OBD/Azure+Vault+Properties+Override+and+Reader+for+Mule+4)