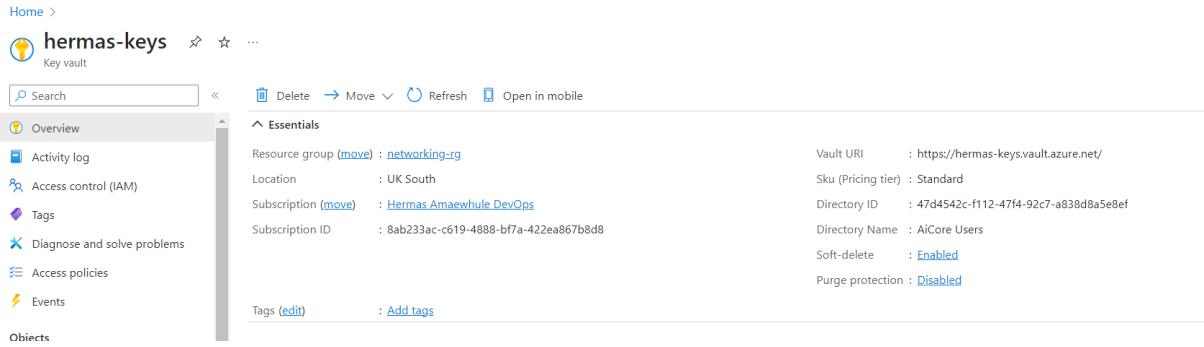
Document: AKS integration with Azure key vault for secrets management

Tasks: To implement a solution to securely store and retrieve the database credentials for the company's application to establish a connection to the Azure SQL database and extract client order information. Currently the credentials for establishing a connection to the database are hardcoded within the application code, posing a significant security risk. This initiative not only fortifies the application against potential security threats, but also adheres t best practices in handling sensitive information.

Steps:

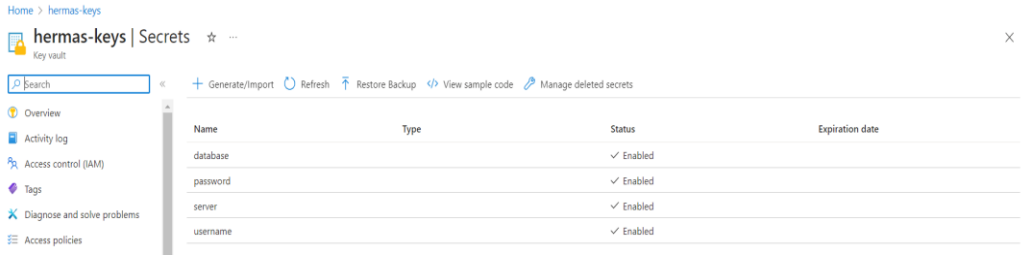
1. Created an Azure Key Vault for securely storing application code and other sensitive information. Followed these steps to do:
    a. While on the Azure Portal used the search feature to locate the Key Vaults service homepage. While on the Key Vaults service home page, clicked on the + Create button to initiate the process of creating the Azure Key Vault. The following basic information were configured:
        i. chose the subscription and selected my resource group.
        ii. Created a unique name for the Key Vault (hermas-keys), which would be used in the Key Vault URI.
        iii. Selected the region for the Key Vault (UK South).
    b. Then clicked the Review + create button to proceed to the next step. After the validation process successfully passes, clicked Create to initiate the deployment of the Azure Key Vault and to provision the resources based on the configurations specified in the step above.



    c. The Key Vault URI is the unique address used to access and interact with the resources stored within the Azure Key Vault. In my case it is https://hermas-keys.vault.azure.net/. This URI is essential for applications or services that need to retrieve or store sensitive information within Azure Key Vault.
2. Assigned the Key Vault Administrator role to my Microsoft Entra ID user to grant to myself the necessary permissions for managing secrets within the Key Vault. This role is one of the primary RBAC roles. It is the highest level of access, that grants full control over the Key Vault. Administrators can manage access policies, configure advanced settings, and perform any operation within the Key Vault. Followed the following steps to assign above role:
    a. While on the Key Vault's service homepage, selected the Access control (IAM) tab from the left-hand side menu to open the dedicated Access Control page.

b. Click on the + Add button, and then choose Add a role assignment. This action redirects to the role assignment configuration page where the Key Vault Administrator role.
c. Within the Members tab, clicked Select members to choose my Microsoft Entra ID user and Selected it to confirm.
d. Clicked on Review + assign to review my configuration before finalizing the role assignment.

3. Created four secrets in the Key Vault to secure the credentials used within the application to connect to the backend database. These secrets include the server's name, server username, server password, and the database name. The values of these secrets were set to the hardcoded values from the application code. The following steps were taken:
    a. While on the Key Vault homepage, clicked on the Secrets tab.
    b. Then clicked on the + Generate/Import button to open the secrets page to create new secrets. The following information to be provided:
        i. Specified names for the new secret as directed in the project.



        ii. Inputed the value of the secret.
        iii. Optionally, configure additional settings like activation and expiration dates. This step allows you to exert control over the secret's lifecycle. But these settings are left unselected, hence the secrets were created without specific activation or expiration dates. This means that the secrets were immediately active upon creation and would persist indefinitely unless manually deleted.
        iv. Clicked Create to finish the process of creating a new secret.

4. Enabled managed identity for the AKS cluster to allow it to authenticate and interact securely with the Key Vault. Followed these steps to do so:
    a. Created a user-managed identity using the following command: az identity create -g myResourceGroup -n myUserAssignedIdentity.
    b. Next, created a new resource with the created user-managed using the following command: az <resource> create --resource-group <myResourceGroup> --name <myResourceName> --assign-identity myUserAssignedIdentity --role contributor --scope <mySubscription>
    c. Assigned Key Vault Permissions. Aassign the 'Key Vault Secrets Officer' role to managed identity. This is an RBAC (Role base access control) role that provides permissions to read, list, set and delete secrets, certificates, and keys within the specified Azure Key Vault. The following command was supposed to be used:
    az role assignment create --role "Key Vault Secrets Officer" \
    --assignee <managed-identity-client-id> \
    --scope /subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.KeyVault/vaults/{key-vault-name}

Note the managed-identity-client-id place holder is the client-id that was noted previously.

d. Integrated Azure Key Vault with AKS following these steps:
    i. Enabled Managed Identity for AKS cluster follow these steps:
        1. Launched a command-line interface on my local machine and signed in to my Azure account using the Azure CLI.
        2. Used the following command to enable managed identity for my AKS cluster: az aks update --resource-group <resource-group> --name <aks-cluster-name> --enable-managed-identity.
        3. Used the following command to get information about the managed identity created for the AKS cluster and took note of the 'client id' under the identity profile for future use: az aks show --resource-group <resource-group> --name <aks-cluster-name> --query identityProfile .

5. Updated the Python application code as follows:
    a. integrated the Azure Identity and Azure Key Vault libraries into the Python application code to facilitate communication with Azure Key Vault.

```
pip install azure-identity
pip install azure-keyvault-secrets
```

    b. Also, modify the code to use managed identity credentials, ensuring secure retrieval of database connection details from the Key Vault.

```python
from azure.identity import ManagedIdentityCredential
from azure.keyvault.secrets import SecretClient

# Replace these values with your Key Vault details
key_vault_url = "https://your-key-vault-name.vault.azure.net/"

# Set up Azure Key Vault client with Managed Identity
credential = ManagedIdentityCredential()
secret_client = SecretClient(vault_url=key_vault_url, credential=credential)

# Access the secret values from Key Vault
secret = secret_client.get_secret("secret-name")

# Access the secret values
secret_value = secret.value
```

    c. Additionally, updated the requirements file for the application's Docker image to include the newly required libraries. This step ensures that the Docker image includes all the necessary dependencies for seamless functioning of the updated code.

6. Performed thorough end-end testing of the modified application as follows:
    a. On local machine, ensured seamless integration with Azure Key Vault. Verified that the application can securely retrieve and utilize the database connections details from Key Vault using managed identity credentials.

b. Deployed the modified application to the AKS cluster using the pre-established Azure DevOps CI/CD pipeline.
c. Conduct end-to-end testing within the AKS environment to validate the functionality of the application, ensuring secure access to Key Vault secrets directly from the Azure DevOps CI/CD pipeline.