🔐 What is a Managed Identity?

Managed Identity provides an automatically managed identity in Azure AD for applications to use when connecting to resources that support Azure AD authentication (e.g., Azure Key Vault, Azure SQL Database, Azure Storage, etc.).

With Managed Identities:

* No credentials are required in the code.
* Identity lifecycle is fully managed by Azure.
* It's more secure and easier to manage compared to using secrets or connection strings.

🧾 Types of Managed Identity

| **Type** | **Description** |
| --- | --- |
| System-assigned | Tied to the lifecycle of the Azure resource (e.g., VM, App Service, Function App). Deleted automatically when the resource is deleted. |
| User-assigned | A standalone Azure resource. Can be shared across multiple resources. Managed independently of any service. |

📘 When to Use Which?

| **Scenario** | **Use** |
| --- | --- |
| A VM, App Service, or Function App needs to access Azure Key Vault | System-assigned |
| Multiple services need to share the same identity | User-assigned |
| You want full control over identity lifecycle | User-assigned |
| Simpler use, tied to resource lifecycle | System-assigned |

🧭 Step-by-Step Guide

🚀 Step 1: Enable Managed Identity

1.1. System-assigned Managed Identity

* Navigate to your Azure resource (e.g., App Service, Function App, VM)
* Go to **Identity > System-assigned**
* Set **Status: On**
* Click **Save**
* Azure AD identity is automatically created for the resource

1.2. User-assigned Managed Identity

* Go to **Azure Portal > Create a Resource > User-assigned managed identity**
* Give it a name, region, and resource group
* Once created, go to your resource (App Service, VM, etc.)
* Go to **Identity > User-assigned**
* Click **Add** and select the identity created earlier

🔐 Step 2: Assign Permissions to the Identity

Use Azure Role-Based Access Control (RBAC) to grant the identity access to the target resource.

Example: Grant Key Vault access

* Go to the **Key Vault**
* Go to **Access control (IAM) > Add role assignment**
* Role: **Key Vault Secrets User** (or Reader, Contributor, etc.)
* Assign to: The identity (either system or user-assigned)

💻 Step 3: Use Managed Identity in Code (Example: Azure Key Vault from Azure Function)

**Prerequisite:**

* Enable managed identity on the Azure Function
* Assign the role in Key Vault

**Sample Code (.NET)**

using Azure.Identity;

using Azure.Security.KeyVault.Secrets;

var credential = new DefaultAzureCredential();

var client = new SecretClient(new Uri("https://<your-keyvault-name>.vault.azure.net/"), credential);

KeyVaultSecret secret = client.GetSecret("<secret-name>");

Console.WriteLine(secret.Value);

DefaultAzureCredential automatically picks the right credentials (including managed identity when running in Azure).

**Python Example**

from azure.identity import DefaultAzureCredential

from azure.keyvault.secrets import SecretClient

credential = DefaultAzureCredential()

client = SecretClient(vault\_url="https://<your-keyvault-name>.vault.azure.net/", credential=credential)

secret = client.get\_secret("<secret-name>")

print(secret.value)

🧪 Step 4: Test the Identity

* Deploy your app to Azure
* Ensure permissions are correct
* Run the app and verify that it can access the resource without any secrets in code

📎 Additional Notes

* You can use az login and az account get-access-token locally to simulate managed identity.
* Managed identities work with Azure CLI, Azure SDKs, and REST APIs.
* Works best when combined with DefaultAzureCredential.

✅ Summary

| **Feature** | **System-assigned** | **User-assigned** |
| --- | --- | --- |
| Tied to resource | Yes | No |
| Can be shared | No | Yes |
| Lifecycle | Auto-managed | Manual |
| Use in multiple resources | ❌ | ✅ |